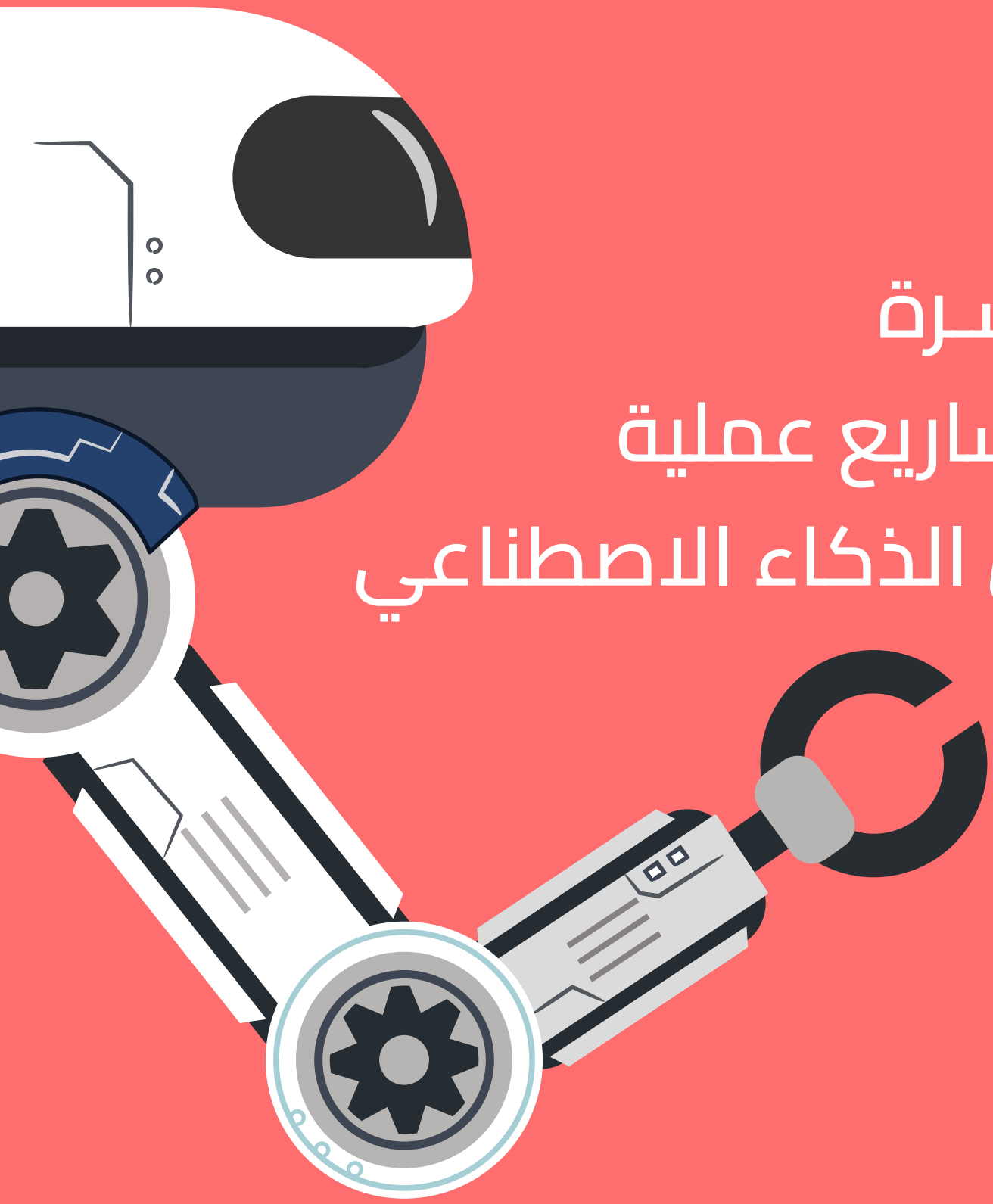


عشـرة مشاريع عملية عن الذكاء الاصطناعي

إعداد

د. باسل الخطيب
خالد الشمعة
محمد لحلح

أكاديمية حسوب

عشرة مشاريع عملية عن الذكاء الاصطناعي

مشاريع عملية تطبيقية عن تعلم الآلة والذكاء الاصطناعي

اسم الكتاب: عشرة مشاريع عملية عن الذكاء الاصطناعي

Book Title: 10 Artificial Intelligence Projects

المؤلف: د. باسل الخطيب، خالد الشمعة، محمد لحلح

Author: dr. Bassel Alkhatib, Khaled Al-Shamaa, Mohamed Lahlah

المحرر: جميل بيلوني

Editor: Jamil Bailony

تصميم الغلاف: سيرين ديرانية

Cover Design: Sirin Diraneyya

سنة النشر: 2022

Publication Year: 2022

رقم الإصدار: 1.0

Edition: 1.0

https://academy.hsoub.com

academy@hsoub.com

عن الناشـر

أُنتج هذا الكتاب برعاية شركة حسوب وأكاديمية حسوب.

تهدف أكاديمية حسوب إلى توفير دروس وكتب عالية الجودة في مختلف المجالات وتقدم دورات شاملة لتعلُّم البرمجة بأحدث تقنياتها معتمدةً على التطبيق العملي الذي يؤهل الطالب لدخول سوق العمل بثقة.

حسوب شركة تقنية في مهمة لتطوير العالم العربي. تبني حسوب منتجات تركِّز على تحسين مستقبل العمل، والتعليم، والتواصل. تدير حسوب أكبر منصتي عمل حر في العالم العربي، مستقل وخمسات ويعمل في فيها فريق شاب وشغوف من مختلف الدول العربية.

المحتويات باختصار

جدول المحتويات

تمهيد

تزداد الأنظمة الحاسوبية المحيطة بنا ذكاءً يومًا بعد يوم -بعد تزويدها بـأدوات تعلم صـنعية- فلم أكن أتخيـل عندما رأيت جهاز الهاتف المحمول أول مرة ذلك الجهاز عـديم الأسـلاك كثـير الأزرار ثقيـل الـوزن أن يتحـول إلى شاشة رقيقة تُحدِّثها وتُحدثك، تنبهك إن تـأخرت عن موعـد نومـك أو أن الطريـق المعتـاد الـذي تسـلكه للعمـل مزدحم لذا تقترح طريقًا بديلًا أو تأجيل الاجتماع الصباحي في العمل.

امتد تطبيق الذكاء الاصطناعي وتعلم الآلة إلى أغلب الأنظمة والتطبيقات المحيطـة بنـا فقلمـا تجـد تطبيقًـا شائعًا لا يتصف بشيء من الذكاء مثل لوحة الكتابة على الجوال التي تقترح عليك كلمات تاليـة عنـد كتابـة كلمـة معينة أو تصحح لك كلمة إن أخطـأت بهـا وأيضًـا مثـل محركـات البحث والمتـاجر الإلكترونيـة الـتي تقـترح على المستخدم اقتراحات توافق ما يطلبه وكأنها تقرأ أفكاره، وهذا بالنسبة للمستخدم النهائي أمـا بالنسـبة لأصـحاب العمل فالتطبيقات كبيرة منها أنظمة مراقبـة المعـاملات المشـبوهة في المصـارف والحـوالات الماليـة وأنظمـة تحليل العملاء وتصنيفهم وأنظمة التنبؤ وغيرها، هذا لم أذكر ما يتعلـق بالروبوتـات والسـيارات وأنظمـة الطـيران والملاحة ذاتية القيادة وغيرها الكثير مما لا يتسع حصره وذكره.

كل ذلـك لم يـأت من فـراغ بـل يقـف خلفـه جيـوش من المهندسـين والخـبراء وبـذلك اعتـني بعلم الـذكاء الاصطناعي وتعلم الآلة أيما اعتناء وألفت فيه مئات الكتب ونشرت آلاف المقالات والأبحاث ومنها هـذا الكتـاب الذي بين يديك!

حول الكتاب

هذا الكتاب هو الجزء الثاني من كتـاب مـدخل إلى الـذكاء الاصـطناعي وتعلم الآلة فبعـد تأسـيس المفـاهيم والمصطلحات التي يقوم عليها مجال الذكاء الاصـطناعي وتعلم الآلـة في الجـزء الأول، ننطلـق في الجـزء الثـاني بتطبيق مشاريع عملية تطبيقية مبنية على بيانات واقعية ويناقش أفكار قابلة للتطبيق في الحياة العملية.

جاءت فكرة الكتاب بجزأيه من كتاب يعرض مشاريع عن الذكاء الاصطناعي -سأذكره لاحقًا- كنا نعمل على ترجمته وكان الكتاب يتطرق إلى مصطلحات ومفاهيم عن الذكاء الاصطناعي وتعلم الآلة دون شرح فتساءلنا كيف للقارئ المبتدئ أن يطبق تلك المشاريع دون فهم تلك المصطلحات؟ وحينئذٍ بدأنا العمل على الجزء الأول.

فصول هذا الكتاب مبنية على عدة مساهمات ساهم بها كل مؤلف:

- الفصول الأربعة الأولى تعتمد على كتاب Python Machine Learning Projects لكاتبه Michelle Morales ونقله للعربية بتصرف محمد لحلح المؤلف للجزء الأول والدكتور باسل الخطيب.
- الفصل الخامس والسادس وهو من كتابة خالد شمعة أثرى بها الكتاب من خبرته الكبيرة في المجال.
- الفصل الحادي عشر مبني على ترجمة سلسلة A Complete Machine Learning Project Walk-Through in Python لكاتبه Will Koehrsen ونقلها للعربية بتصرف الدكتور باسل الخطيب.
- الفصول المتبقية هي من كتابة الدكتور باسل الخطيب المختص في مجال الذكاء الاصطناعي وتعلم الآلة والأستاذ في جامعات عدة مرموقة منها جامعة دمشق.

ما يميز هذا الكتاب قربه من القارئ العربي، إذ لم يقتصر على الترجمة ونقل تطبيقات أجنبية جاهزة أو تعتمد على نصوص أجنبية بل يعرض تطبيقات على نصوص باللغة العربية وهذا ما يفتقر إليه المحتوى العربي في هذا المجال.

قراءة الكتاب وتنفيذ المشاريع

مستوى هذا الكتاب متقدم لذا يجب أن تملك معرفة أساسية بمجال الذكاء الاصطناعي وتعلم الآلة وخبرة بلغة بايثون فالمشاريع كلها مطبقة فيها، فإن كنت مبتدئًا فننصح قبل هذا الكتاب قراءة الكتابيين التاليين:

- البرمجة بلغة بايثون
- مدخل إلى الذكاء الاصطناعي وتعلم الآلة

ترتيب القراءة الأمثل للكتاب يكون من أوله لآخره وفق ما رتبناه لك ولكن يمكنك قراءة الكتاب بأي ترتيب ففصوله لا تعتمد على بعضها باستثناء الفصل الأول الذي يشرح كيفية إعداد بيئة العمل لسائر المشاريع والفصل الأخير الذي يختتم الكتاب بمناقشة مسألة تقييم نماذج تعلم الآلة وحساب مجموعة من مقاييس تقييم الأداء والتي تخبرنا بأداء نموذج التعلم المبني ودرجة تعلمه.

يمكنك تطبيق المشاريع محليًا على حاسوبك مباشرةً أثناء قراءة المشروع وتعلمه وستجد غالبًا في بداية أو نهاية المشروع رابط لتنزيل شيفرته ومقارنتها مع الشيفرة التي كتبتها ولتكون لك مرجعًا، كما يمكنك الاستعانة

بمنصة Google Colab لتنفيذ المشاريع واختصار وقت التنفيذ وإعداد البيئة وقد وفرنا لبعض المشاريع رابطًا على تلك المنصة.

المساهمة

يرجى إرسال بريد إلكتروني إلى academy@hsoub.com إذا كان لديك اقتراح أو تصحيح على النسخة العربية من الكتاب أو أي ملاحظة حول أي مصطلح من المصطلحات المستعملة، ويمكنك تسهيل البحث علينا بتضمين جزء من الجملة التي يظهر الخطأ فيها على الأقل، كما يفضل إضافة أرقام الصفحات والأقسام.

جميل بيلوني

2022-10-21

1. إعداد بيئة العمل للمشاريع مع بايثون

تُناسب بايثون الكثير من التطبيقات البرمجية نظرًا لتمتعها بالعديد من المزايا المهمة كالمرونة العالية وتوفير المكتبات الجاهزة فيها للقيام بعمليات الأتمتة وتحليل البيانات ومعالجة مسائل تعلم الآلة والتطوير الخلفي back-end وغيرها من المهام البرمجية بسرعة وسهولة. ظهرت النسخة الأولى منها في عام 1991 وسُميت تيمنًا بالفرقة الكوميدية البريطانية Monty Python هادفة جعل كتابة الكود فيها أمرًا ممتعًا، ويعد الإصدار الثالث أحدث نسخة وهو المستقبل الواعد لبايثون.

يعرض هذا الفصل خطوات تثبيت بيئة بايثون على الحاسوب المحلي أو الخادم البعيد لتطبيق المشاريع العملية اللاحقة في الكتاب، أما إذا كان بايثون مع أدواته pip و venv جاهزًا على حاسوبك فيُمكنك الانتقال مباشرة إلى الفصل التالي.

1.1 المتطلبات الرئيسية

نعمل في بيئة لينكس Linux أو الشبيهة بيونكس Unix-like ونستخدم ضمن نظام التشغيل ماكنتوش macOS سطر الأوامر command line أو بيئة الطرفية terminal environment، أما في نظام ويندوز، فيمكن استعمال بورشيل PowerShell لتحقيق نفس النتائج عمليًا.

1.2 الخطوة الأولى: تثبيت بايثون

يثبت بايثون على العديد من أنظمة التشغيل بشكل افتراضي. للتأكد من وجود الإصدار 3 من بايثون مثبتًا على جهازك قم بفتح نافذة طرفية واكتب فيها ما يلي:

```
python3 -V
```

سيظهر، في حال كون الإصدار 3 من بايثون مثبتًا، رقم الإصدار والذي يُمكن بالطبع أن يختلف حسب النسخة المُنصبة، وفي جميع الأحوال، يكون الخرج مشابهًا لما يلي:

```
Python 3.7.2
```

في حال لم تحصل على الإظهار السابق، فعليك أولًا تنزيل النسخة من موقع بايثون python.org ومن ثم اتباع خطوات التثبيت المُحدّدة. بعد انتهائك من تثبيت بايثون وتأكدك من ذلك بمعاينة رقم الإصدار باتباع التعليمة السابقة يُمكنك الانتقال للخطوة التالية.

1.3 الخطوة الثانية: تثبيت pip

يجب تثبيت الأداة pip والتي تسمح بتثبيت وإدارة حزم البرمجيات المساندة لبايثون. ستكون هذه الأداة جاهزة فيما لو قمت بتثبيت بايثون من الموقع python.org، أما إذ كنت على خادم أو حاسوب بتوزيعة أبنتو Ubuntu أو ديبيان Debian فيُمكنك تنزيل pip بكتابة ما يلي:

```
sudo apt install -y python3-pip
```

من الآن فصاعدًا، يُمكنك تنزيل أي حزمة برمجيات بكتابة:

```
pip3 install package_name
```

حيث package_name هي اسم أي مكتبة أو حزمة برمجية لبايثون مثل جانغو Django لتطوير مواقع الويب والحزمة NumPy للحسابات العلمية، فإذا كنت مثلًا تريد استخدام المكتبة NumPy فعليك تنزيلها بكتابة الأمر التالي:

```
install numpy
```

يوجد بعض الحزم البرمجية الأساسية الواجب تحميلها للحصول على بيئة عمل مريحة ومرنة:

```
sudo apt install build-essential libssl-dev libffi-dev python3-dev
```

يُمكن الآن بعد الانتهاء من المراحل السابقة إعداد بيئة عمل افتراضية.

1.4 الخطوة الثالثة: إعداد بيئة عمل افتراضية

تسمح بيئة العمل الافتراضية بعزل كل مشروع مع حزمه البرمجية التابعة له عن بقية المشاريع الأخرى وذلك عن طريق تخصيص مساحة خاصة له على الخادم، مما يسمح أيضًا بتنظيم الإصدارات المختلفة للمشروع. وهو أمر ضروري لاسيما عند استخدامنا لحزم برمجية خارجية.

يُمكن إنشاء بيئة عمـل افتراضـية لكـل مشـروع وهي عمليًـا عبـارة عن مجلـد معين على الخـادم مـع بعض الشيفرات البرمجية ضمنه. نستخدم عادةً الأداة venv لإنشـاء بيئـة عمـل افتراضـية والـتي هي جـزء من مكتبـة بايثون وتُثبت تلقائيًا خلال تثبيت بايثون.

إذ كنت على خادم أو حاسوب Ubuntu أو Debian فيُمكنك تحميل venv بكتابة ما يلي:

```
sudo apt install -y python3-venv
```

انتقل لمجلد معين على حاسوبك لوضع بيئة العمل فيه أو قم بإنشاء مجلد جديـد باسـتخدام تعليمـة إنشـاء مجلد mkdir ثم انتقل له باستخدام تعليمة الانتقال لمجلد cd كما يلي:

```
mkdir environments
cd environments
```

يُمكنك الآن إنشاء بيئة العمل ضمن المجلد الذي انتقلت له وذلك بكتابة ما يلي:

```
python3.6 -m venv my_env
```

حيث my_env هو اسم بيئـة العمـل الـذي تريـده، مـع ملاحظـة كتابـة القسـم الأول من رقم الإصـدار الـذي حصلت عليه سـابقًا باسـتخدام python -V (المثـال أعلاه يسـتخدم الإصـدار Python 3.6.3). لـو كُنت مثلًا تستخدم الإصدار Python 3.7.3 فعليك كتابة:

```
python3.7 -m venv my_env
```

يسمح لك ويندوز بتجاهل رقم الإصدار كليةً وكتابة ما يلي:

```
python -m venv my_env
```

بعد تنفيذ الأمر المناسب لإنشاء بيئة العمل، يُمكن التأكد من إتمام العملية بالاستمرار في الخطوات التاليـة والتي من أولها معاينة الملفات التي قام venv بإنشائها في المجلد الموافق والتي يُمكن إظهارها باستخدام أمـر استعراض محتوى مجلد ls:

```
ls my_env
```

ليكون الخرج:

```
bin include lib lib64 pyvenv.cfg share
```

تعمل هذه الملفات على التأكد من العزل الكامل لملفـات مشـروعك عن بـاقي ملفـات حاسـوبك وبـذا فلن تختلط ملفاتك مع ملفات نظام التشغيل أبدًا. تسمح هـذه الممارسـة الجيـدة في التحكم بالإصـدارات المختلفـة للمشروع والتأكد من أن مشروعك يستخدم الحزم البرمجية التي يحتاج إليها.

يُمكن استخدام الحزمة البرمجية الجاهزة Python Wheels التي تسمح بتسريع عملية تطوير البرمجيات وتنفيذ المشروع والموجودة في المجلد share في Ubuntu 18.04، ويجب تفعيل هذه الحزمة قبل بدء استخدامها بكتابة الأمر التالي والذي يقوم باستدعاء السكربت activate:

```
source my_env/bin/activate
```

من الآن فصاعدًا، ستبدأ أوامرك باسم بيئة العمل (المدعوة في أمثلتنا my_env). يُمكن للبادئة أن تظهر أحيانًا بشكل مختلف وذلك حسب الإصدار المستخدم من لينكس ديبيان Debian Linux إلا أنه، وفي جميع الأحوال، يجب أن تبدأ أوامرك باسم بيئة العمل ضمن قوسين:

```
(my_env) sammy@sammy:~/environments$
```

تُحدّد هذه البادئة بأن بيئة العمل my_env هي البيئة النشطة الحالية مما يعني أنه عندما نُنشئ برامج هنا فستستخدم إعدادات ومكتبات هذه البيئة.

لاحظ أنه يُمكنك ضمن بيئة العمل الافتراضية استخدام python عوضًا عن python3 و pip عوضًا عن pip3 إن وجدت ذلك مناسبًا. أما خارج بيئة العمل الافتراضية فلا يُمكنك القيام بذلك بل عليك استخدام python3 و pip3.

1.5 الخطوة الرابعة: إنشاء البرنامج الأول

يُمكنك الآن إنشاء البرنامج التقليدي الترحيبي الأول "Hello, World" مما يسمح لك بالتأكد من جاهزية بيئة العمل.

افتح مثلًا محرر النصوص nano وأنشئ ملفًا جديدًا:

```
(my_env) sammy@sammy:~/environments$ nano hello.py
```

ثم اكتب في نافذة المحرر المفتوحة أول برنامج بسيط في بايثون:

```
print("Hello, World!")
```

أغلق المحرر nano بالضغط على الاختصار Ctrl+X ولا تنسَ حفظ الملف بالإجابة بنعم y عندما تُسأل عن حفظ الملف.

بعد إغلاق nano والعودة لصدفة shell النظام، يُمكنك تنفيذ البرنامج السابق hello.py:

```
(my_env) sammy@sammy:~/environments$ python hello.py
```

والذي سيُظهر على المحطة الطرفية:

```
Hello, World!
```

للخروج من بيئة العمل والعودة للمجلد الأساسي، نفذّ الأمر:

```
deactivate
```

1.6 النتائج

لقد حصلت نتيجة تطبيقك للخطوات السابقة على بيئة بايثون جاهزة لاحتضان المشاريع البرمجية على حاسوبك ويُمكنك الآن الانطلاق في تطبيق المشاريع البرمجية في الفصول التالية!

إن أردت التعرف أكثر على بايثون، فيمكنك الرجوع إلى كتاب البرمجة بلغة بايثون.

2. بناء مصنف يتنبأ بمدى خطورة حالة مرضية

سننفذ في هذا الفصل خوارزميةً بسيطةً لتعلم الآلة بلغة بـايثون Python باسـتخدام مكتبـة Scikit-learn، وهذه المكتبة ما هي إلا أداةٌ لتطبيق تعلّم الآلة بلغة بايثون، كما سنسـتخدم المُصـنّف المعتمـد على قـانون بـايز Naive Bayes -يختصر إلى NB- مع قاعدة بياناتٍ حقيقية لمعلومات ورم سرطان الثدي، والـذي سـيتنبأ إذا مـا كان الورم خبيثًا أم حميدًا. وفي نهاية هذا الفصل ستعرف خطوات وكيفية إنشاء نمـوذج تنبـؤي خـاص بـك لتَعَلّم الآلة بلغة بايثون.

2.1 متطلبات المشروع

قبل البدء بهذا الفصل لا بد من تجهـيز البيئـة المناسـبة، وسنسـتخدم محـرر الشـيفرات البرمجيـة Jupyter Notebooks، وهو مفيد جدًا لتجربة وتشغيل الأمثلة الخاصـة بتَعَلّم الآلـة بطريقـةٍ تفاعليـةٍ، حيث تسـتطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسـرعة، ممـا سيسـهل علينـا اختبـار الشـيفرات البرمجية وتصحيحها.

يُمكنك فتح متصفح الويب والذهاب لموقع المحرر الرسمي jupyter على الوِيب لبدء العمـل بسـرعة، ومن ثمّ انقر فوق "جرّب المحرر التقليدي Try Classic Notebook"، وستنتقل بعـدها لملـفٍ جديـدٍ بـداخل محـرر Jupyter Notebooks التفاعلي، وبذلك تجهّز نفسك لكتابة الشيفرة البرمجية بلغة البايثون.

إذا رغبت بمزيدٍ من المعلومات حول محرر الشيفرات البرمجية Jupyter Notebooks وكيفيّة إعداد بيئتـه الخاصة لكتابة شيفرة بايثون، فيمكنك الاطلاع على فصـل كيفيـة تهيئـة تطـبيق المفكـرة jupyter notebook للعمل مع لغة البرمجة python.

2.2 إعداد بيئة المشروع

ستحتاج أولًا لتثبيت بعض التبعيات، وذلك لإنشاء مساحة عملٍ للاحتفاظ بملفاتنا قبل أن نتمكن من تطوير برنامج التعرف على الصور، وسنستخدم بيئة بايثون 3.8 الافتراضية لإدارة التبعيات الخاصة بمشروعنا.

سَنُنشئ مجلدًا جديدًا خاصًا بمشروعنا وسندخل إليه هكذا:

```
mkdir cancer-demo
cd cancer-demo
```

سننفذّ الأمر التالي لإعداد البيئة الافتراضية:

```
python -m venv cancer-demo
```

سننفذّ الأمر التالي لتشغيل البيئة الافتراضية في لينكس Linux:

```
source cancer-demo/bin/activate
```

أما في ويندوز Windows:

```
"cancer-demo/Scripts/activate.bat"
```

سنستخدم إصداراتٍ محددةٍ من هذه المكتبات، من خلال إنشاء ملف `requirements.txt` في مجلد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

سنفتح الملف `requirements.txt` في محرر النصوص، وسنُضيف الأسطر البرمجية التالية، وذلك لتحديد المكتبات التي نريدها وإصداراتها:

```
jupyter==1.0.0
scikit-learn==1.0
```

سنحفظ التغييرات التي طرأت على الملف وسنخرج من محرر النصوص، ثم سنُثَبت هذه المكتبات بالأمر التالي:

```
(cancer-demo) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، سنُصبح جاهزين لبدء العمل على مشروعنا.

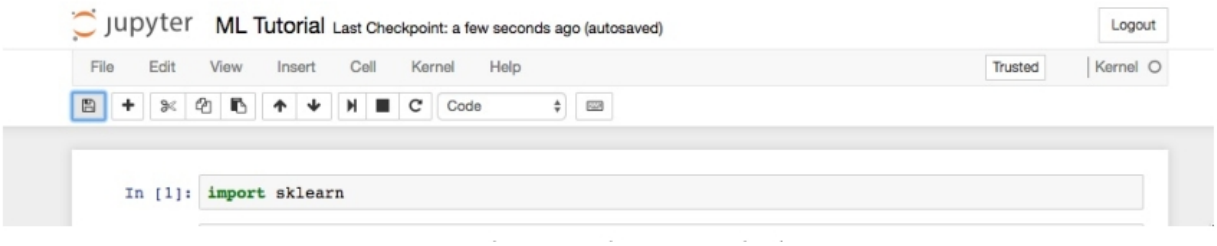
شغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت. هكذا:

```
(cancer-demo) $ jupyter notebook
```

أنشئ ملفًا جديدًا في داخل المحرر بالضغط على الزر new واختيار (ipykernal) python 3 وسمه باسم ML Tutorial مثلًا، حيث ستكون في الخلية الأولى للملف عملية استيراد الوِحدة أو المكتبة البرمجية scikit-learn (لمزيد من المعلومات حول طريقة استيراد وحدة برمجية في لغة بايثون يمكنك الاطلاع على كيفية استيراد الوحدات في بايثون 3 سبق وأن ناقشنا فيه هذه الفكرة بالتفصيل):

```
import sklearn
```

يجب أن يبدو الملف الخاص بك شبيهًا بالملف التالي:



والآن بعد استيرادنا للمكتبة بنجاح، سنبدأ العمل مع مجموعة البيانات لبناء نموذج تعلّم الآلة الخاص بنا.

2.3 استيراد مجموعة بيانات Scikit-Learn's

مجموعة البيانات التي سنتعامل معها في هذا الفصل هي قاعدة بيانات تشخيص مرض سرطان الثدي في ولاية ويسكونسن الأمريكية، وتتضمن هذه المجموعة من البيانات معلوماتٍ مختلفةٍ حول أورام سرطان الثدي، بالإضافة إلى تصنيفات الأورام سواءً كانت خبيثة أم حميدة. كما تحتوي على 569 حالة (أو للدقة بيانات 569 ورمًا)، كما تتضمن معلومات عن 30 ميزة لكلّ ورم، مثل: نصف قطر الورم ونسيجه ونعومته ومساحته.

سنبني نموذجًا لتعلّم الآلة من مجموعة البيانات السابقة باستخدام معلومات الورم فقط للتنبؤ فيما إذا كان الورم خبيثًا أم حميدًا.

يُثَبت مع مكتبة Scikit-learn مجموعات بياناتٍ مختلفةٍ افتراضيًا، ويُمكننا استيرادها لتُصبح متاحةً للاستخدام في بيئتنا مباشرةً، لنفعل ذلك:

```
from sklearn.datasets import load_breast_cancer
# تحميل مجموعة البيانات
data = load_breast_cancer()
```

سيُمثَل المتغير data ككائنٍ في بايثون، والذي سيعمل مثل عمل القاموس الذي هو نوعٌ مُضمَّن في بايثون، بحيث يربط مفاتيح بقيم على هيئة أزواجٍ، وستُؤخذ بالحسبان مفاتيح القاموس، وهي أسماء الحقول المُصنّفة target_names، والقيم الفعلية لها target، وأسماء الميّزات feature_names، والقيم الفعلية لهذه الميزات data.

تُعَد الميّزات جزءًا مهمًا من أي مصنّف، إذ تُمثّل هذه الميزات خصائص مهمةً تصف طبيعة البيانات، كما ستساعدنا في عملية التنبؤ بحالة الورم (ورم الخبيث malignant tumor أو ورم حميد benign tumor)، ومن الميّزات المُفيدة المحتملة في مجموعة بياناتنا هذه، هي حجم الورم ونصف قطره ونسيجه.

أنشئ في الملف نفسه بعد ذلك متغيرات جديدةً لكلّ مجموعة مهمة من هذه المعلومات وأسند لها البيانات:

```
# تنظيم بياناتنا
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

والآن أصبحت لدينا قوائم لكلّ مجموعةٍ من المعلومات، ولفَهم مجموعة البيانات الخاصة بنا فهمًا صحيحًا ودقيقًا، سنُلقي نظرةً عليها من خلال طباعة حقول الصنف مثل طباعة أول عينةٍ من البيانات، وأسماء ميّزاتها، وقيمها هكذا:

```
# الاطلاع على بياناتنا
print(label_names)
print(labels[0])
print(feature_names[0])
print(features[0])
```

إن نفذّت هذه الشيفرة بطريقةٍ صحيحةٍ فسترى النتائج التالية:

```
In [3]: # Look at our data
        print(label_names)
        print(labels[0])
        print(feature_names[0])
        print(features[0])

        ['malignant' 'benign']
        0
        mean radius
        [  1.79900000e+01   1.03800000e+01   1.22800000e+02   1.00100000e+03
           1.18400000e-01   2.77600000e-01   3.00100000e-01   1.47100000e-01
           2.41900000e-01   7.87100000e-02   1.09500000e+00   9.05300000e-01
           8.58900000e+00   1.53400000e+02   6.39900000e-03   4.90400000e-02
           5.37300000e-02   1.58700000e-02   3.00300000e-02   6.19300000e-03
           2.53800000e+01   1.73300000e+01   1.84600000e+02   2.01900000e+03
           1.62200000e-01   6.65600000e-01   7.11900000e-01   2.65400000e-01
           4.60100000e-01   1.18900000e-01]
```

نلاحظ من الصورة أن أسماء الأصناف الخاصة بنا ستكون خبيث malignant وحميد benign (أي أن الورم سيكون إما خبيثًا أو حميدًا)، والمرتبطة بقيم ثنائية وهي إما 0 أو 1، إذ يُمثّل الرقم 0 أورامًا خبيثة ويُمثّل الرقم 1 أورامًا حميدة، لذا فإن أول مثالٍ للبيانات الموجودة لدينا هو ورم خبيثٌ نصف قطره 1.79900000e+01.

والآن بعد تأكدنا من تحميل بياناتنا تحميلًا صحيحًا في بيئة التنفيذ، سنبدأ العمل مع بياناتنا لبناء مصنّف باستخدام طُرق تعلّم الآلة.

2.4 تنظيم البيانات في مجموعات

ينبغي عليك دائمًا اختبار النموذج على البيانات غير المرئية، وذلك لتقييم مدى جودة أداء المُصنّف، لهذا قسّم البيانات الخاصة بك إلى جزأين قبل بناء النموذج، بحيث تكون هناك مجموعةٌ للتدريب ومجموعةٌ للاختبار.

تستطيع استخدام المجموعة المخصصة للتدريب من أجل تدريب وتقييم النموذج أثناء مرحلة التطوير. حيث ستمنحك منهجية تنبؤات هذا النموذج المُدرّب على المجموعة المخصصة للاختبار غير المرئية، فكرةً دقيقةً عن أداء النموذج وقوته.

لحسن الحظ، لدى المكتبة `Scikit-learn` دالة تُدعى `train_test_split()`، والتي ستقسمُ بياناتك لهذه المجموعات. ولكن يجب أن تستورد هذه الدالة أولًا ومن ثَمّ تستخدمها لتقسيم البيانات:

```
from sklearn.model_selection import train_test_split
# تقسيم بياناتنا
train, test, train_labels, test_labels = train_test_split(features,
labels, test_size=0.33, random_state=42)
```

ستُقسّمُ هذه الدّالة البيانات بطريقةٍ عشوائيةٍ باستخدام الوسيط `test_size`. في مثالنا لدينا الآن مجموعةً مخصصةً للاختبار test تُمثّل 33% من مجموعة البيانات الأصلية، وسيُشكّل الجزء المتبقي من البيانات المجموعة المخصصة للتدريب train. كما لدينا حقولٌ مخصصةٌ لكلٍ من المتغيرات، سواء أكانت مُخصّصةً للاختبار أو للتدريب، أي `train_labels` و`test_labels`. لنُدرّب الآن نموذجنا الأول.

2.5 بناء النموذج وتقييمه

هناك العديد من النماذج المُخَصصة لتعلّم الآلة، ولكلّ نموذجٍ منها نقاط قوةٍ وضعفٍ. في هذا الفصل، سنُركّز على خوارزمية بسيطةٍ تؤدي عادةً أداءً جيدًا في مهام التصنيف الثنائية، وهي خوارزمية بايز Naive Bayes.

أولًا، سنستورد الوِحدة البرمجية `GaussianNB` ثم نُهَيّئ النموذج باستخدام الدالة `GaussianNB()`، بعدها سنُدرّب النموذج من خلال مُلاءمته مع البيانات باستخدام الدالة `gnb.fit()`:

```
from sklearn.naive_bayes import GaussianNB
# تهيئة المصنف خاصتنا
gnb = GaussianNB()
# تدريب المصنف
```

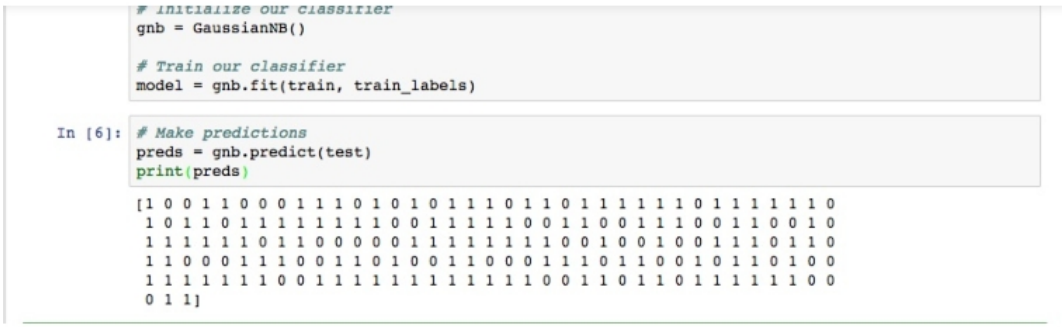
```
model = gnb.fit(train, train_labels)
```

بعد أن نُـدّرب النمـوذج سنسـتخدمه للتنبـؤ على المجموعـة المخصصـة للاختبـار، وسـننفذ ذلـك من خلال الدّالة `predict()`، والتي ستُعيد مجموعةً من التنبؤات لكلّ نسخة بياناتٍ في المجموعة المخصصـة للاختبـار، ثم نطبع تنبؤاتنا لِفَهم ما حدده هذا النموذج.

استخدِم الدالة `predict()` مع مجموعة البيانات المخصصة للاختبار `test` واطبع النتائج:

```
# بناء التوقعات
preds = gnb.predict(test)
print(preds)
```

عند تنفيذك للشيفرة البرمجية تنفيذًا صحيحًا سترى النتائج التالية:



فكما ترى، أعادت الدالة `predict()` مصفوفةً ثُنائية القيم إما 0 أو 1، حيث تُمثل القيم المتوقعة لصـنف الورم (خبيث أم حميد).

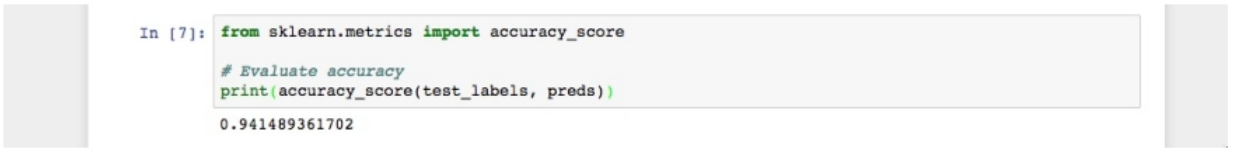
والآن بعد أن حصلنا على توقعاتنا، لِنُقيِّم مدى جودة أداء هذا المُصنّف.

2.6 تقييم دقة النموذج

نُقيّم دقـة القيم المتوقَّعـة لنموذجنـا باسـتخدام مصـفوفة التصـنيفات الناتجـة للأصـناف الحقيقية الـتي لـدينا، وذلـك من خلال موازنـة المصـفوفتين `test_labels` و`preds` باسـتخدام الدالة `accuracy_score()` التابعة للمكتبة `Scikit-learn`، وذلك لتحديد دِقة المُصنّف.

```
from sklearn.metrics import accuracy_score
# تقييم الدقة
print(accuracy_score(test_labels, preds))
```

سترى النتائج التالية:



كما ترى في النتيجة، فإن المُصنّف NB دقيقٌ بنسبة 94.15%. وهـذا يعـني أن المُصـنِّف قـادرٌ على التنبـؤ الصحيح فيما إذا كان الورم خبيثًا أو حميدًا بنسبة 94.15% من الحـالات الكُليّـة. كمـا تُشـير هـذه النتـائج إلى أن مجموعة الميّزات المُكونة من 30 ميزة هي مؤشرات جيدة لصنف الورم.

بهذا تكون قد نجحت في إنشاء مصنِّفك الأول الذي يعتمد في عمله على طرق تعلّم الآلة، والآن لنعد تنظيم الشيفرة البرمجية بوضع جميع عمليات الاستيراد في أعلى الملف، إذ يجب أن تبدو النسخة النهائية من الشيفرة البرمجية خاصتك شبيةً بهذه الشيفرة:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# تحميل البيانات
data = load_breast_cancer()

# تنظيم البيانات
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']

# الاطلاع على البيانات
print(label_names)
print('Class label = ', labels[0])
print(feature_names)
print(features[0])
# تقسيم البيانات
train, test, train_labels, test_labels = train_test_split(features,
labels, test_size=0.33, random_state=42)

# تهيئة المصنّف
gnb = GaussianNB()
```

```
# تدريب المصنف
model = gnb.fit(train, train_labels)

# بناء التوقعات
preds = gnb.predict(test)
print(preds)

# تقييم الدقة
print(accuracy_score(test_labels, preds))
```

والآن بإمكانك إكمال العمل على الشيفرة البرمجية، وتحسين عمل المُصنّف وتوسيعه، وكذا تجربة هذا المصنّف مع مجموعاتٍ فرعيةٍ مختلفةٍ من الميزات، أو حتى تجربة خوارزمياتٍ مختلفةٍ تمامًا.

تستطيع الاطلاع على الموقع الرسمي لمكتبة Scikit-Learn لمزيدٍ من الأفكار حول تطبيق تعلّم الآلة مع البيانات لبناء شيءٍ مفيدٍ.

2.7 الخلاصة

لقد تعلمنا في هذا الفصل كيفية إنشاء مُصنّف بالاعتماد على تعلّم الآلة بلغة بايثون باستخدام المكتبة Scikit-learn، والآن بإمكانك تحميل البيانات في بيئةٍ برمجيةٍ وتنظيمها وتدريبها، وكذا التنبؤ بأشياء بناءً عليها، وتقييم دِقّة المُصنّفات الناتجة.

نتمنى أن تُساعدك هذه الخطوات في تسهيل طريقة العمل مع بياناتك الخاصة بلغة بايثون.

3. بناء شبكة عصبية للتعرف على أرقام مكتوبة بخط اليد

تُستخدم الشبكات العصبية كوسيلةٍ للتعلم العميق، فهي واحدةٌ من المجالات الفرعية العديدة لطُرق تطبيق الذكاء الاصطناعي. وقد اقْتُرحت لأول مرةٍ منذ حوالي 70 عامًا في محاولةٍ لمحاكاة طريقة عمل دماغ الإنسان، إلا أنها أبسط بكثيرٍ من الخلايا العصبية الحقيقة، إذ أن كلّ خليةٍ اصطناعيةٍ مرتبطةٌ بعِدة طبقاتٍ، ولكلّ واحدةٍ منها وزنٌ مُعينٌ يُعبر عن أهمية هذه الطبقة، وذلك لتحديد كيفية استجابة الخلية العصبية عند نشْر البيانات عبرها، وبينما كانت الشبكات العصبية سابقًا محدودةً في عدد الخلايا العصبية التي تستطيع محاكاتها في ذلك الوقت -وهو الأمر الذي انعكس بدوره على تعقيد عملية التعلم التي يمكننا تحقيقها-، إلا أنه في السنوات الأخيرة، ونظرًا للتقدم الكبير في تطوّر الأجهزة، استطعنا أخيرًا بناء شبكاتٍ عصبيةٍ عميقةٍ جدًا، وتدريبها على مجموعات بياناتٍ هائلةٍ وضخمة كذلك، مما أدّى إلى تحقيق قفزاتٍ نوعية في تطور الذكاء الاصطناعي وتحديدًا تعلم الآلة.

سمحت تلك القفزات النوعية للآلات بمُقاربة قدرات البشر، بل وتجاوزتها في أداء بعض المهام المحدودة. ومن بين هذه المهام، قدرتها على التعرف على الكائنات، فعلى الرغم من أن الآلات كانت غير قادرة تاريخيًا على منافسة قوة الرؤية البشرية، إلا أنّ التطورات الحديثة في التعلم العميق جعلت من الممكن بناء شبكاتٍ عصبيةٍ باستطاعتها التعرف على الكائنات والوجوه والنصوص، بل وحتى العواطف!

سنُطبق في هذا الفصل قِسمًا فرعيًا صغيرًا من طُرق التعرف على الكائنات، وتحديدًا التعرف على الأرقام المكتوبة بخط اليد، وذلك باستخدام مكتبة TensorFlow، وهي مكتبة بايثون مفتوحة المصدر التي طُوّرت في مختبرات غوغل Google Brain لأبحاث التعلم العميق، كما أنها من أشهر المكتبات الحالية في التعلم العميق، وسنأخذ صورًا مكتوبٌ عليها الأرقام بخط اليد من الرقم 0 وحتى الرقم 9، وسنبني شبكةً عصبيةً وندربها لكي تتعرف على التصنيف المناسب لكل رقمٍ معروضٍ في الصورة وتتنبأ به، ثم تَنسْبه لصنفٍ من أصناف الأرقام الموجودة.

سنفترض إلمامك بمصطلحات ومفاهيم تعلم الآلة، مثل التدريب والاختبار والميزات والأصناف والتحسين والتقييم. لهذا لن تحتاج لخبرةٍ سابقةٍ في مجال التعلم العميق التطبيقي أو بمكتبة TensorFlow، ولمتابعة وفَهم هذا الفصل جيدًا ننصحك أولًا بالاطلاع على: المفاهيم الأساسية لتعلم الآلة.

3.1 إعداد المشروع

ستحتاج أولًا لتثبيت بعض التبعيات، وذلك لإنشاء مساحة عملٍ للاحتفاظ بملفاتنا قبل أن نتمكن من تطوير برنامج التعرف على الصور، وسنستخدم بيئة بايثون 3.8 الافتراضية لإدارة التبعيات الخاصة بمشروعنا.

سَنُنشئ مجلدًا جديدًا خاصًا بمشروعنا وسندخل إليه هكذا:

```
mkdir tensorflow-demo
cd tensorflow-demo
```

سننفذّ الأوامر التالية لإنشاء البيئة الافتراضية:

```
python -m venv tensorflow-demo
```

ومن ثم الأمر التالي في لينكس Linux لتنشيط البيئة الافتراضية:

```
source tensorflow-demo/bin/activate
```

أما في ويندوز Windows، فيكون أمر التنشيط:

```
"tensorflow-demo/Scripts/activate.bat"
```

بعد ذلك، سنُثبتّ المكتبات التي سنستخدمها.

سنستخدم إصداراتٍ محددةٍ من هذه المكتبات، من خلال إنشاء ملف `requirements.txt` في مجلد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها، لذا نفتح الملف في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبات التي نريدها وإصداراتها:

```
keras==2.6.0
numpy==1.19.5
Pillow==8.4.0
scikit-learn==1.0
scipy==1.7.1
sklearn==0.0
tensorflow==2.6.0
```

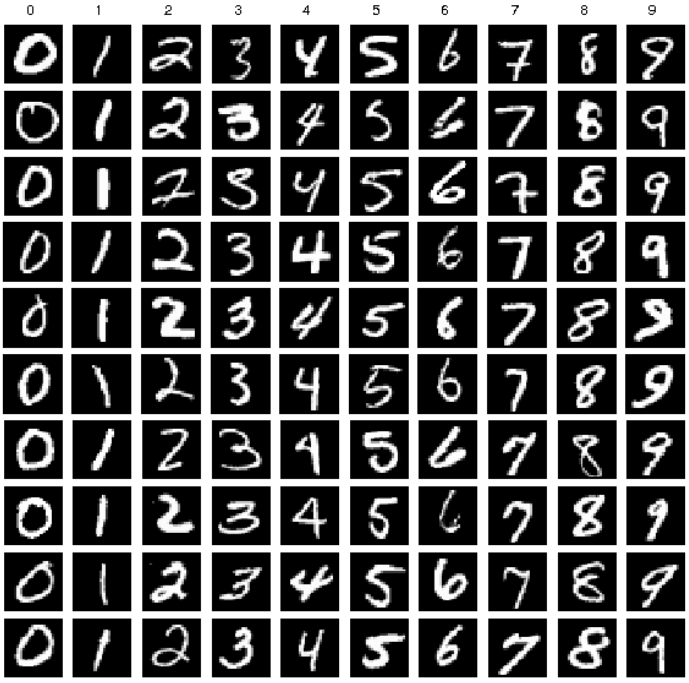
سنحفظ التغيـيرات الـتي طـرأت على الملـف وسـنخرج من محـرر النصـوص، ثم سـنُثَبت هـذه المكتبـات بالأمر التالي:

```
(tensorflow-demo) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، سنُصبح جاهزين لبدء العمل على مشروعنا.

3.2 استيراد مجموعة بيانات MNIST

تُسـمى مجموعـة البيانـات الـتي سنسـتخدمها، بمجموعـة بيانـات MNIST، وهي مجموعـة كلاسـيكية في مجتمع مُطوري تعلم الآلة، وتتكون من صور لأرقام مكتوبة بخـط اليـد، بحجم 28×28 بكسـل. ونسـتعرض فيمـا يلي بعض الأمثلة للأرقام المُتضمنة فيها:



لاحظ أنه ينبغي أن نستخدم ملفًا واحدًا لجميع أعمالنا في هذا الفصل، ولنُنشئ برنـامج بـايثون يتعامـل مـع مجموعة البيانات هذه، فسننشئ ملفًا جديدًا باسم `main.py`، وسنفتح هـذا الملـف بـأي محـرر شـيفرات لـدينا مثل VS code وسنُضيف هذه الأسطر البرمجية لاستيراد المكتبات اللازمة:

```
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from PIL import Image  #مكتبة معالجة الصور
#التوافقية مع إصدار سابق
tf.compat.v1.disable_v2_behavior()
```

وسنضيف أيضًا هذه الأسطر من الشيفرات البرمجية لملفك لاستيراد مجموعة بيانات MNIST وذلك باختيار صور التدريب المتاحة من Tensorflow ومن ثم نُنزلها ونقسمها إلى جزئين: الأول للتدريب والثاني للاختبار:

```
#اختيار بيانات التدريب
mnist = tf.keras.datasets.mnist
#تنزيل بيانات التدريب والاختبار
(x_train, y_train), (x_test, y_test) = mnist.load_data()
#طباعة عدد بيانات التدريب والاختبار
print (len(x_train))
print (len(x_test))
#تحجيم البيانات بين 0 و 1
x_train, x_test = x_train / 255.0, x_test / 255.0
#الترميز الأحادي النشط
y_train = [[i] for i in y_train]
y_test = [[i] for i in y_test]
enc = OneHotEncoder(sparse=True)
enc.fit(y_train)
y_train =  enc.transform(y_train)
y_test = enc.transform(y_test)
```

تقوم الدالة `mnist.load_data` بتنزيل البيانات وتقسيمها إلى مجموعتين واحدة للتدريب (60000 صورة) والمجموعة الثانية للاختبار (10000 صورة).

وعند قراءة البيانات سنستخدم الترميز الأحادي النشط One-Hot Encoding لتمثيل التصنيفات للصور. حيث يَستَخدم الترميز الأحادي النشط One-Hot Encoding متجهًا vector مُكونٌ من قيم ثنائية لتمثيل القيم الرقمية أو الصنفية.

ونظرًا لأن أصنافنا مخصصةٌ لتمثيل الأرقام من 0 إلى 9، فإن المتجه سيحتوي على 10 قيم، واحدةٌ لكلّ رقمٍ ممكنٍ. وتُسنَد إحدى هذه القيم بوضع القيمة 1، وذلك لتمثيل الرقم في هذا المؤشر للمتجه، كما ستُسنَد القيم الباقية بالقيمة 0. فمثلًا، سيُمثلُ الرقم 3 من خلال المتجه هكذا: [0، 0، 0، 0، 0، 0، 1، 0، 0، 0]. وسنلاحظ وجود القيمة 1 في الفهرس 3، لذلك فإن المتجه سيُمثِلُ الرقم 3.

ولتمثيل الصور الفعلية والتي تكون بحجم 28x28 بكسل، يتوجب علينا تسويتها في المتجه 1D بحجم 784 بكسل، وهو ناتج ضرب 28×28. وسنخزن هذه البكسلات والتي ستُشكل الصورة لاحقًا، وذلك في قيم تتراوح بين 0 و255، حيث ستحدّد هذه القيم تدرج اللون الرمادي للبكسل، وستُعرَض صورنا باللونين الأبيض والأسود

فقط. لذلك سيُمثل البكسل الأسود بالقيمة 255، والبكسل الأبيض بالقيمة 0، وذلك مع التدرجات المختلفة للون الرمادي بينهم.

والآن بعد استيرادنا للبيانات، حان الوقت للتفكير في كيفية بناء الشبكة العصبية.

3.3 تحديد بنية الشبكة العصبية

يُشير مصطلح بنية الشبكة العصبية لعناصرٍ متنوعةٍ، مثل عدد الطبقات في الشبكة وعدد الوِحدات في كلّ طبقةٍ، كما يشير إلى كيفية توصيل هذه الوِحدات بين الطبقات المختلفة. ونظرًا لأن الشبكات العصبية مستوحاةٌ من كيفية عمل الدماغ البشري، فسنستخدم مصطلح الوِحدة ليُمثّل ما يُمكن تسميته بيولوجيًا بالخلايا العصبية.

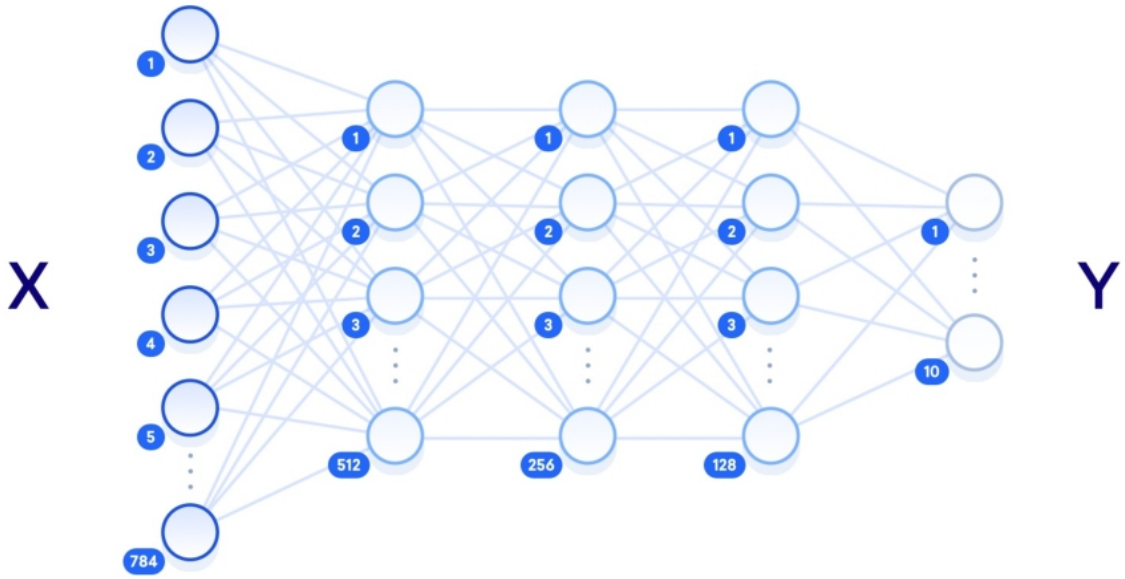
تأخذ الوحدات بعض القيم من الوِحدات السابقة مثل مُدخلاتٍ لها، حيث تتشابه مع الخلايا العصبية التي تُمرر إشاراتها حول الدماغ، ثمّ تُجري عمليةً حسابيةً، وتُمرر القيمة الجديدة مثل مُخرجاتٍ إلى وِحداتٍ أخرى، وهكذا. تُوضع هذه الوِحدات على شكل طبقاتٍ متراكبةٍ فوق بعضها البعض مشكّلةً الشبكة العصبية، بحيث يمكن للشبكة أن تتألف كحدٍ أدنى من طبقتين، طبقةٌ لإدخال القيم، وطبقةٌ أخرى لإخراج القيم. يُستخدم مصطلح الطبقة المخفية لجميع الطبقات الموجودة بين طبقات المُدخلات وطبقات المُخرجات الخارجية، أي أن تلك الطبقات تكون مخفيةً عن العالم الحقيقي.

تحقق البُنى المختلفة للشبكة نتائج مختلفة عن بعضها البعض، ويمكن اتخاذ الأداء مثل معيارٍ للحكم على هذه البُنى المختلفة، كما يمكن اتخاذ عناصر أخرى معيارًا للحكم، مثل الوسطاء والبيانات ومدة التدريب.

سنضيف هذه الأسطر البرمجية التالية لملفك، وذلك لتخزين عدد الوِحدات المُخصصة لكلّ طبقةٍ ووضعها في متغيّراتٍ عامةٍ. وهذه الطريقة ستسمح لنا بتغيير بِنية الشبكة بمكانٍ واحدٍ، وفي نهاية هذا الفصل يمكنك اختبار مدى تأثير الأعداد المختلفة من الطبقات والوِحدات على نتائج نموذجنا:

```
n_input = 784 # input layer (28x28 pixels)
n_hidden1 = 512 # 1st hidden layer
n_hidden2 = 256 # 2nd hidden layer
n_hidden3 = 128 # 3rd hidden layer
n_output = 10 # output layer (0-9 digits)
```

يُوضح الرسم البياني التالي تصورًا للبِنية التي صمّمناها، مع توصيل كلّ طبقةٍ بالطبقات المحيطة بها توصيلًا كاملًا:



ويرتبط مصطلح الشبكة العصبية العميقة Deep Neural Network بعدد الطبقات المخفية، وعادةً ما تُشير كلمة السطحية في مصطلح الشبكة العصبية السطحية إلى وجود طبقةٍ مخفيةٍ واحدةٍ، بينما تُشير كلمة العميقة إلى وجود طبقاتٍ مخفيةٍ متعددةٍ. ونظريًا إذا أُعطِيت الشبكة العصبية السطحية ما يكفي من بياناتٍ للتدريب، فيجب أن تَقدِر على تمثيل أي وظيفة يمكن للشبكة العصبية العميقة أن تؤدّيها. ولكن من ناحية الفعالية الحسابية، فغالبًا ما يكون نتائج استخدام شبكة عصبية عميقة ذات حجمٍ صغيرٍ أفضل من النتائج التي تُعطيها الشبكة العصبية السطحية ذات العدد الكبير من الوِحدات المخفية، وذلك عند تأديتهم لنفس المَهمة.

كما أن الشبكات العصبية السطحية غالبًا ما تواجه مشكلة فَرط التخصيص Overfitting، إذ يكون هدف الشبكة الأساسي هو حفظ بيانات التدريب التي شاهدتها، ولكنها لن تستطيع تعميم المعرفة التي اكتسبتها على البيانات الجديدة، وهذا هو السبب في كون استخدام الشبكات العصبية العميقة أكثر شيوعًا، إذ أنها تسمح للطبقات المتعددة الموجودة بين البيانات المُدخلة الأولية والبيانات المُصنفة الناتجة، بتعلم الميزات على مستوياتٍ متنوعةٍ، مما يُعزز قدرة الشبكة على التعلم وتعميم الفكرة.

ومن العناصر الأخرى للشبكة العصبية التي يجب تعريفها هنا هي الوسطاء الفائقة Hyperparameters، فعلى عكس الوسطاء العادية التي تُحدث قيمها أثناء عملية التدريب، سنُسند قيم الوسطاء الفائقة في البداية وسنثبتها طوال العملية.

أسنِد المتغيّرات بالقيم التالية في ملفك:

```
learning_rate = 1e-4
n_iterations = 1000
batch_size = 128
dropout = 0.5
```

يمثل معدل التعلم Learning Rate مدى تعديل الوسطاء في كلّ خطوةٍ من عملية التعلّم، إذ تُعَد هذه التعديلات مكونًا رئيسيًا للتدريب، فبعد كلّ عملية مرورٍ عبر الشبكة، سنضبط أوزان الطبقات قليلًا لأهمية ذلك في محاولةٍ لتقليل الخسارة، حيث يمكن لمعدل التعلم المرتفع أن يتحقق بسرعة، ولكن يمكن كذلك أن تتجاوز القيم المثلى عند تحديثها في كلّ مرة.

يشير مصطلح عدد التكرارات Number Of Iterations إلى عدد مرات مرورنا على خطوة التدريب، ويشير حجم الدفعة Batch Size لعدد أمثلة التدريب التي نستخدمها في كل خطوة، كما ويمثل المتغير `dropout` الموضع الذي نحذف عنده بعضًا من الوِحدات عشوائيًا. وسنستخدم المتغير `dropout` في الطبقة النهائية المخفية لإعطاء كلّ وِحدة من الوحدات احتمالًا بنسبة 50% للتخلص منها في كلّ خطوة تدريبٍ، وهذا سيساعد على منع ظهور مشكلة فرط التخصيص Overfitting.

حددنا الآن بِنية شبكتنا العصبية والوسطاء الفائقة التي ستُؤثر على عملية التعلّم، والخطوة التالية هي بناء الشبكة مثل مخططٍ بيانيٍ من خلال مكتبة TensorFlow.

3.4 بناء مخطط بياني من خلال مكتبة TensorFlow

لبناء شبكتنا، لابد لنا من إعداد الشبكة مثل مخططٍ بيانيٍ حسابي من خلال مكتبة TensorFlow لتنفيذه. والمفهوم الأساسي لمكتبة TensorFlow هو `tensor`، وهو بنية بياناتٍ مشابهةٍ لبِنية المصفوفة Array، أو القائمة List. وهذا المتغير سيهيَأ ويُعالَج عند مروره عبر المخطط البياني للشبكة عبر عملية التعلّم.

وسنبدأ بتحديد ثلاثة متغيراتٍ `tensors` من نوع `placeholders`، وهو نوع `tensor` تُسندُ قيمته لاحقًا. والآن سنضيف الشيفرة البرمجية التالية إلى الملف الذي نعمل عليه:

```
X = tf.compat.v1.placeholder("float", [None, n_input])
Y = tf.compat.v1.placeholder("float", [None, n_output])
keep_prob = tf.compat.v1.placeholder(tf.float32)
```

إنّ الوسيط الوحيد الذي يتوجب علينا تحديده عند التعريف هو حجم البيانات التي سنُسندها لاحقًا، وبالنسبة للمتغير `X` سنستخدم شكل `[None, 784]`، إذ ستمثل القيمة `None` كميةً غير محددةٍ، وسنُسند كميةً غير محددةٍ من الصور ذات حجم 784 بكسل. بحيث يصبح شكل المتغير `Y` هو `[None, 10]` وستمثل `None` عددًا غير محددٍ من التصنيفات الناتجة، مع وجود 10 أصنافٍ محتملةٍ. وسنستخدم في المتغير `keep_prob tensor` من نوع `placeholders` للتحكم في معدل `dropout`، وسنجعله من نوع `placeholders` وذلك لجعله متغيرًا من نوعٍ قابلٍ للتعديل، بدلًا من كونه متغيرًا من نوعٍ غير قابلٍ للتعديل immutable variable، وذلك لأننا نريد استخدام نفس `tensor` التدريب عند إسناد `dropout` بالقيمة 0.5، ونفس `tensor` الاختبار عند إسناد `dropout` بالقيمة 1.0.

والوسطاء التي ستُحدث قيمها الشبكة العصبية في عملية التدريب هي القيم الخاصة بوزن كلّ طبقة، والتي تُعبر عن الأهمية وقيم التحيز bias values، لذلك سنحتاج لإسنادهم بقيم ابتدائية بدلًا من قيم فارغة. وهذه القيم هي الأساس الذي ستبدأ الشبكة رحلة التعلم انطلاقًا منها، إذ ستُستخدم في تفعيل دوال الشبكة العصبية، والتي تُمثِلُ قوة الاتصالات بين الوِحدات.

ونظرًا لاستمرار تحسين القيم أثناء عملية التدريب، يمكننا ضبطها حاليًا بالقيمة 0. لاحظ أن القيمة الأولية في الواقع لها تأثيرٌ كبيرٌ على الدقة النهائية للنموذج. وسنستخدم التوزيع الاحتمالي الطبيعي المنقطع Truncated normal distribution لتوليد قيم عشوائية لأوزان الطبقات، بحيث يكونون قريبين من الصفر حتى يتمكنوا من التعديل إما باتجاهٍ إيجابيٍ أو سلبيٍ، كما يكونون مختلفين قليلًا، وذلك ليُنتِجوا أخطاءً مختلفةً، وبهذه الطريقة سنضمن بأن يتعلم النموذج شيئًا مفيدًا.

والآن سنضيف هذه الأسطر البرمجية التالية لملفنا الذي نعمل عليه:

```
weights = {
    'w1': tf.Variable(tf.random.truncated_normal([n_input,
    n_hidden1], stddev=0.1)),
    'w2': tf.Variable(tf.random.truncated_normal([n_hidden1,
    n_hidden2], stddev=0.1)),
    'w3': tf.Variable(tf.random.truncated_normal([n_hidden2,
    n_hidden3], stddev=0.1)),
    'out': tf.Variable(tf.random.truncated_normal([n_hidden3,
    n_output], stddev=0.1)),
}
```

بالنسبة للتحيز Bias، سنستخدم قيمةً ثابتةً صغيرةً لضمان تنشيط جميع tensors المراحل الأولية، وبالتالي المساهمة في الانتشار. وستُخزن الأوزان وجميع tensors التحيزات في objects قواميس Dictionary لسهولة الوصول إليها. أضف الشيفرة التالية للملف الذي نعمل عليه وذلك لتعريف التحيز وقِيَمه:

```
biases = {
    'b1': tf.Variable(tf.constant(0.1, shape=[n_hidden1])),
    'b2': tf.Variable(tf.constant(0.1, shape=[n_hidden2])),
    'b3': tf.Variable(tf.constant(0.1, shape=[n_hidden3])),
    'out': tf.Variable(tf.constant(0.1, shape=[n_output]))
}
```

والآن جهّز طبقات الشبكة العصبية من خلال تعريف العمليات التي ستتعامل مع tensors المرحلة الحالية. وأضف هذه الشيفرة البرمجية للملف الذي نعمل عليه:

```
layer_1 = tf.add(tf.matmul(X, weights['w1']), biases['b1'])
layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'])
layer_3 = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'])
layer_drop = tf.nn.dropout(layer_3, keep_prob)
output_layer = tf.matmul(layer_3, weights['out']) + biases['out']
```

ستنفذ كلّ طبقةٍ مخفيةٍ عملية ضربٍ للمصفوفة على نتائج الطبقة التي سبقتها وعلى أوزان الطبقة الحالية، وسيُضاف التحيز لهذه القيم. في الطبقة المخفية الأخيرة، سنُطبِّق عملية التسرب dropout بالقيمة 0.5 للمتغير `Keep_prob` الخاص بنا.

الخطوة الأخيرة في بناء المخطط البياني، هي تحديد دالة الخسارة التي نريد تحسينها. والاختيار الشائع لدالة الخسارة في المكتبة البرمجية TensorFlow هو الأنتروبي المشترك Joint Antropy، والمعروف كذلك باسم فقدان السجل log-loss، وهو الذي يُحدد الفرق بين التوزيعين الاحتماليين لكل من التنبؤات والتصنيف. ويمكن أن تكون قيمة الأنتروبي المشترك 0، وذلك في أفضل الأحوال عند التصنيف المثالي، وذلك مع انعدام الخسارة تمامًا.

سنحتاج كذلك إلى اختيار خوارزمية التحسين المناسبة، والتي سنستخدمها لتقليل الناتج من دالة الخسارة. وتُسمى هذه العملية بعملية تحسين الانحدار التدريجي، وهي طريقةٌ شائعةٌ للعثور على الحد الأدنى للدالة، من خلال اتخاذ خطواتٍ تكراريةٍ على طول التدرج في الاتجاه السلبي التنازلي.

وهناك العديد من الخيارات لخوارزميات تحسين الانحدار التدريجي المُطبقة في المكتبة البرمجية TensorFlow، إلا أننا سنستخدم في هذا الفصل خوارزمية المُحسِّن أدم Adam optimizer، الذي يعتمد على عملية تحسين الانحدار التدريجي باستخدام الزخم أو كمية الحركة Momentum، وذلك بتسريع عملية التنعيم من خلال حساب متوسط مُرجَّحٍ بكثرة للتدرجات، واستخدام ذلك في التعديلات مما يؤدي لتقاربٍ أسرع، وسنضيف هذه الشيفرة للملف الذي نعمل عليه:

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(
        labels=Y, logits=output_layer
    ))
train_step = tf.compat.v1.train.AdamOptimizer(1e-
4).minimize(cross_entropy)
```

عرّفنا حتى الآن الشبكة وبنيناها باستخدام المكتبة البرمجية TensorFlow، والخطوة التالية هي إرسال البيانات عبر المخطط البياني لتدريبها، ومن ثَمّ اختبارها للتحقق فيما إن كانت تعلمت شيئًا بالفعل أم لا.

3.5 التدريب والاختبار

تتضمن عملية التدريب تغذية المخطط البياني للشبكة بمجموعة بيانات التدريب، وتحسين نتيجة دالة الخسارة، إذ أن في كلّ مرةٍ تمر فيها الشبكة عبر مجموعةٍ إضافيةٍ من صور التدريب، فستُحدثُ الوسطاء لتقليل الخسارة، وذلك بهدف تحسين دقة التنبؤ للأرقامٍ؛ أما عملية الاختبار، فتتضمن تشغيل مجموعة بيانات الاختبار الخاصة بنا عبر المخطط البياني المدرّب، كما ستتتبع عدد الصور التي صح التنبؤ بها، حتى نحسب الدقة جيدًا.

قبل البدء في عملية التدريب، سوف نحدد دالة تقييم الدقة لكي نتمكن من طباعتها على مجموعاتٍ صغيرةٍ من البيانات أثناء التدريب. هذه البيانات المطبوعة ستسمح لنا بالتحقق من انخفاض الخسارة وزيادة الدقة، وذلك بدءًا من المرور الأول عبر المخطط البياني، وحتى المرور الأخير؛ كما ستسمح لنا بتتبّع ما إذا نفذنا عمليات مرورٍ كافيةً عبر المخطط البياني للوصول لنتيجةٍ مناسبةٍ ومثاليةٍ أم لا:

```
correct_pred = tf.equal(tf.argmax(output_layer, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

سنستخدم الدالة `arg_max` في المتغيّر `right_pred` للموازنة بين الصور التي صَحَ توقُّعها، وذلك بالنظر لقيمة التنبؤات `output_layer` والتصنيفات Y، وسنستخدم الدالة `equal` لإعادة هذه النتائج مثل قائمة مؤلفة من قيم بولیانية. ويمكننا بعد ذلك تحويل هذه القائمة للنوع `float`، وذلك لحساب المتوسط للحصول على درجة الدقة الإجمالية.

الآن نحن جاهزون لتهيئة الجلسة لتشغيل المخطط البياني، إذ سنرسل للشبكة أمثلة التدريب الخاصة بنا، وبمجرد انتهاء التدريب، سنرسل أمثلة اختبارٍ جديدةٍ عبر المخطط البياني نفسه لتحديد دقة النموذج.

أضف هذه الشيفرة للملف الذي نعمل عليه:

```
init = tf.compat.v1.global_variables_initializer()
sess = tf.compat.v1.Session()
sess.run(init)
```

إن جوهر عملية التدريب في التعلم العميق هو تحسين ناتج دالة الخسارة. ونحن هنا سنهدف إلى تقليل الفرق بين التصنيفات المُتوقعة للصور والتصنيفات الحقيقية لها. وستتضمن هذه العملية أربع خطواتٍ تتكرر لعددٍ محددٍ من مرات المرور عبر المخطط البياني، وهي:

- دفع القيم إلى الأمام عبر الشبكة.
- حساب الخسارة.
- دفع القيم للخلف عبر الشبكة.

- تحديث الوسطاء.

ففي كلّ خطوة تدريب، سنعدِّل الوسطاء قليلًا في محاولةٍ لتقليل نتائج دالة الخسارة. وفي الخطوة التالية مع تَقدُّم عملية التعلّم، يجب أن نشاهد انخفاضًا في الخسارة، حيث سنوقِف التدريب في النهاية، وسنستخدم الشبكة مثل نموذجٍ لاختبار بياناتنا الجديدة.

سنضيف هذه الشيفرة البرمجية للملف الذي نعمل عليه:

```
# التدريب على دفعات صغيرة
for i in range(n_iterations):
    startbatch = (i*batch_size) % len(x_train)
    endbatch = ((i+1)*batch_size) % len(x_train)
    batch_x = np.array(x_train[startbatch:endbatch])
    batch_x = batch_x.reshape(batch_size, -1)
    batch_y = y_train[startbatch:endbatch].toarray()
    if batch_x.shape != (128, 784):
        continue
    sess.run(train_step, feed_dict={
        X: batch_x, Y: (batch_y), keep_prob: dropout
    })
# طباعة الخسارة والدقة لكل دفعة صغيرة
    if i % 100 == 0:
        minibatch_loss, minibatch_accuracy = sess.run(
            [cross_entropy, accuracy],
            feed_dict={X: batch_x, Y: batch_y, keep_prob: 1.0}
        )
        print(
            "Iteration",
            str(i),
            "\t| Loss =",
            str(minibatch_loss),
            "\t| Accuracy =",
            str(minibatch_accuracy)
        )
```

بعد 100 عملية مرورٍ لكلّ خطوة تدريبٍ والتي أرسلنا فيها مجموعةً صغيرةً من الصور عبر الشبكة، سنطبع نتائج دالة الخسارة والدقة لتلك الدفعة. وينبغي ألّا نتوقع هنا انخفاض معدل الخسارة وزيادة الدقة، لأن القيم لكل دفعةٍ صغيرةٍ. إذ أن النتائج ليست للنموذج بأكمله. فنحن نستخدم مجموعاتٍ صغيرةٍ من الصور بدلًا من

إرسال كلّ صورةٍ بمفردها، وذلك لتسريع عملية التدريب والسماح للشبكة برؤية عددٍ من الأمثلة المختلفة قبل تحديث الوسطاء.

وبمجرد اكتمال التدريب، يمكننا تشغيل الجلسة على الصور المخصصة للاختبار. وهذه المرة سنستخدم القيمة 1.0 مثل مُعدلِ تَسرّب dropout للمتغيّر `Keep_prob`، وذلك للتأكد من أن جميع الوِحدات نشطةٌ في عملية الاختبار.

أضف هذه الشيفرة البرمجية للملف الذي نعمل عليه:

```
# إعداد صور الاختبار كمتجهات أحادية طول كل منها 28*28
x_test = x_test.reshape(-1,784)
test_accuracy = sess.run(accuracy, feed_dict={X: x_test, Y:
y_test.toarray(), keep_prob: 1.0})
print("\nAccuracy on test set:", test_accuracy)
```

والآن سنشغّل برنامجنا، لنعرف مدى دقة شبكتنا العصبية في التعرف على الأرقام المكتوبة بخط اليد. وسنحفظ التغييرات في الملف main.py الذي نعمل عليه.

نَفِّذ الأمر التالي في الوِحدة الطرفية لتنفيذ الشيفرة البرمجية:

```
(tensorflow-demo) $ python main.py
```

سترى نتيجةً مشابهةً لما يلي، ويمكن أن تختلف قليلًا نتائج الخسارة والدقة الفردية:

```
Iteration 0 | Loss = 3.67079 | Accuracy = 0.140625
Iteration 100 | Loss = 0.492122 | Accuracy = 0.84375
Iteration 200 | Loss = 0.421595 | Accuracy = 0.882812
Iteration 300 | Loss = 0.307726 | Accuracy = 0.921875
Iteration 400 | Loss = 0.392948 | Accuracy = 0.882812
Iteration 500 | Loss = 0.371461 | Accuracy = 0.90625
Iteration 600 | Loss = 0.378425 | Accuracy = 0.882812
Iteration 700 | Loss = 0.338605 | Accuracy = 0.914062
Iteration 800 | Loss = 0.379697 | Accuracy = 0.875
Iteration 900 | Loss = 0.444303 | Accuracy = 0.90625
Accuracy on test set: 0.9206
```

ولمحاولة تحسين دقة نموذجنا، أو لمعرفة المزيد حول تأثير ضبط الوسطاء الفائقة hyperparameters، يمكننا تغييرها لاختبار تأثيرها المنعكس على معدّل التعلم وعتبة التسرب Dropout Threshold، وكذا حجم الدفعة من الصور في كمية الأمثلة وعدد مرات المرور عبر المخطط، كما يمكننا كذلك تغيير عدد الوِحدات في

طبقاتنا المخفية وتغيير عدد الطبقات المخفية نفسها، وذلك لنرى كيف ستؤثر بنية الشبكة العصبية على النموذج سواءً بزيادة دقته أو بتخفيضها.

وللتأكد من أن الشبكة تتعرف جيدًا على الصور المكتوبة بخط اليد، فسنختبرها على صورةٍ خاصةٍ بنا، فإذا كنت تعمل على جهازك المحلي وترغب في استخدام صورٍ من جهازك، يمكنك استخدام أي محرر رسوماتٍ لإنشاء صورة بأبعاد 28x28 بكسل لأي رقمٍ تريده. مثلًا:

نزل الصورة (اضغط عليها لتنزلها من متصفح الويب أو تجدها ضمن الملفات المرفقة) وانقلها إلى مجلد المشروع (تأكد أنها باسم test_image.png أو غير اسمها في الشيفرة) ثم أضف في نهاية الملف main.py هذا السطر البرمجي التالي لتحميل صورة الاختبار للرقم المكتوب بخط اليد:

```
img = np.array(Image.open("test_image.png").convert('L')).ravel()
```

إن الدالة `open` من مكتبة الصور `Image` تحمّل صورة الاختبار مثل مصفوفةٍ رباعية الأبعاد 4D، حيث تحتوي على قنوات الألوان الثلاث الرئيسية RGB بالإضافة إلى الشفافية، ولكن هذا ليس نفس التمثيل الذي استخدمناه سابقًا عند القراءة من مجموعة البيانات باستخدام المكتبة البرمجية TensorFlow، لذلك سنحتاج للقيام ببعض المهام الإضافية ليتناسب تنسيق هذه الصور مع التنسيق الذي سبق واعتمدناه في الخوارزمية.

سنستخدم الدالة `convert` مع الوسيط `L` لتقليل تمثيل 4D RGBA إلى قناة لونٍ رماديةٍ واحدةٍ، وسنُخزنها على هيئة مصفوفة `numpy`. وسنستدعي `ravel` لتسوية المصفوفة.

الآن بعد أن صححّنا بنية معلومات الصورة، يمكننا تشغيل الجلسة بنفس الطريقة السابقة، ولكن هذه المرة سنُرسل صورةً واحدةً فقط للاختبار، وسنضيف الشيفرة التالية للملف لاختبار الصورة وطباعة التصنيف الناتج:

```
prediction = sess.run(tf.argmax(output_layer, 1), feed_dict={X:
[img]})
print ("Prediction for test image:", np.squeeze(prediction))
```

وتُستدعى الدالة `np.squeeze` على المتغير `prediction` ليُعِيد عددًا صحيحًا وفريدًا إلى المصفوفة. وسيتضح من الناتج أن الشبكة العصبية قد تعرفت على الصورة كرقم 4:

```
Prediction for test image: 4
```

يمكنك الآن تجربة عملية اختبار الشبكة باستخدام صورٍ أكثر تعقيدًا مثل الأرقام المتشابهة مع الأرقام الأخرى، أو أرقامٍ مكتوبةٍ بخطٍ سيئٍ أو حتى خاطئةٍ، وذلك لمعرفة وقياس مدى نجاحها.

3.6 الخلاصة

في هذا الفصل، نجحنا في تدريب شبكةٍ عصبيةٍ لتصنيف مجموعة بياناتٍ MNIST بدقـةٍ تصـل إلى 92%، واختبارها على صورةٍ خاصةٍ بنا، مع العلم بأنه قد تحققت نسبة أعلى في الأبحاث العلمية الحديثة وكـانت حـوالي 99% لنفس الفكرة، وذلك باسـتخدام بُـنىً مختلفـةً لشـبكةٍ عصـبيةٍ ذات تعقيـدٍ أكـبر، بحيث تتضـمن طبقـات تلافيفية. وتستخدم تلك الشبكات بِنيةً ثنائية الأبعاد للصورة لتمثيل المحتويات تمثيلًا أفضل من تمثيل نموذجنا السابق، إذ أن نموذجنا يُسوّي كلّ البكسلات في متجهٍ واحدٍ مكـوّنٍ من 784 وِحـدة. ويمكنـك قـراءة المزيـد على الموقــع الرســمي للمكتبــة TensorFlow، والاطلاع على الأوراق البحثيــة الــتي تُفصِّــل أدق النتــائج على موقع MNIST.

والآن بعد أن تعرفنا على كيفية بناء شبكةٍ عصبيةٍ وتـدريبها، يمكنـك تجربـة هـذا التطـبيق واسـتخدامه على بياناتك الخاصة، أو اختبارها على مجموعات بياناتٍ شائعةٍ مختلفـةٍ عن تلـك الـتي اسـتخدمناها مثـل: مجموعـة البيانات من غوغل أو مجموعة البيانات من CIFAR-10، وذلك للتعرف على صورٍ أكثر عموميةٍ وشموليةٍ.

> يُمكن تنزيل شيفرة المشروع كاملًا مع صورة الاختبار من هذا الرابط أو هذا الرابط.

4. بناء روبوت للعب الألعاب

يُعَدّ التعلم المُعزز مجالًا فرعيًا من فروع نظرية التحكم، والتي تُعنى بالتحكم في الأنظمة التي تتغير بمرور الوقت؛ وتشمل عِدّة تطبيقاتٍ من بينها: السيارات ذاتية القيادة، وبعض أنواع الروبوتات مثل الروبوتات المُخصصة للألعاب. وسنستخدم في هذا الفصل التعلم المعزز لبناء روبوتٍ لألعاب الفيديو المُخصصة لأجهزة آتاري Atari، حيث لن يُمنح هذا الروبوت إمكانية الوصول إلى المعلومات الداخلية للعبة، وإنما سُيمنح إمكانية الوصول لنتائج اللعبة المعروضة على الشاشة، وكذا المكافآت الناتجة عن هذه اللعبة فقط؛ أي أنه سيرى ما يراه أي شخصٍ سيلعبُ هذه اللعبة.

في مجال تعلّم الآلة، يُعرَّف الروبوت Bot بأنه الوكيل Agent، وسيكون الوكيل بمثابة لاعبٍ في النظام، بحيث يعمل وفقًا لدالة اتخاذ القرار، وهدفنا الأساسي هو تطوير روبوتاتٍ ذكيةٍ من خلال تعليمهم وإمدادهم بقدراتٍ قويةٍ تقدر على اتخاذ القرار.

نعرض في هذا الفصل كيفية تدريب روبوت باستخدام التعلم المعزز العميق وفق النموذج الحُر Deep Q-learning للعبة غزاة الفضاء Space Invaders، وهي لعبةٌ مخصصةٌ لجهاز أتاري أركايد Atari Arcade الكلاسيكي.

4.1 فهم التعلم المعزز

يكون هدف اللاعب في أي لعبة هو زيادة درجاته، وسنشير في هذا الفصل لنتيجة اللاعب على أنها مكافأته، ولتعظيم المكافأة يجب على اللاعب أن يكون قادرًا على تحسين قدراته في اتخاذ القرارات، وبصيغةٍ أخرى فلسفية، القرار هو عملية النظر إلى اللعبة، أو مراقبة حالة اللعبة، واختيار فعلٍ معينٍ مناسبٍ لهذه الحالة.

يُمكن بشكل عام نمذجة روبوت لاعب في بيئة لعب بسلسلة من الحالات State(ح S) والأفعال Actions (أ A) والمكافآت Rewards (م R): ح.أ.م.ح.أ.م.ح.أ.م...إلخ. أو بالإنكليزية: S,A,R,S,A,R ...إلخ.

يهدف التعلم إلى اختيار الفعل الأنسب من مجموعة من الأفعال المُمكنة في حالة ما والذي يُمكن أن يـؤدي إلى مكافأة كبيرة في النهاية. ليس من الضروري الحصول على مكافأة جيدة في الحالة التالية، بمعنى أن لا يكـون الهدف محلي (مكافأة جيدة فورًا) بل أن يكون الهدف استراتيجي (ربح اللعبة مع مكافأة كبيرة في نهاية اللعبة.

أي أننا سنضع في حُسباننا ما يلي:

1. عند مشاهدة العديد من الملاحظات حول حالات الروبوت وأفعاله ومكافآته، يمكن للمرء تقدير المكافـأة المناسبة لكلّ حالةٍ وفعل معينٍ وذلك حسب النهاية التي آلت إليها اللعبة (أو مرحلة متقدمة منها).
2. تُعَد لعبة غزاة الفضاء Space Invaders من الألعاب ذات المكافأة المتأخرة؛ أي أن اللاعب يكافَأ عندما يفجّر أحد الأعداء وليس بمجرد إطلاقه للنار فقط، ومع ذلـك فـإن قـرار اللاعب بـإطلاق النـار هـو الـدافع الصحيح للحصول على المكافأة، لذا بطريقةٍ أو بأخرى، يجب أن نتعلم منح الحالة المرتبطة بفعل الإطلاق مكافأةً إيجابيةً.

4.2 متطلبات المشروع

سنحتاج في هذا المشروع إلى المكتبات الأساسية التالية:

- Gym: وهي مكتبة بـايثون تجعـل الألعـاب المختلفـة متاحـةً للأبحـاث، وكـذا جميـع التبعيـات الخاصـة بالألعاب المخصصة لأجهزة أتاري، والتي طورتها شركة OpenAI. حيث تقدم مكتبة Gym معـايير عامـة لكلّ لعبةٍ، وذلك لتمكننا من تقييم أداء الروبوتات والخوارزميات المختلفة بطريقةٍ موحدةٍ.
- Tensorflow: وهي مكتبة التعلم العميق المقدمة من غوغل، تتيح لنا إجراء العمليات الحسابية بطريقةٍ أكثر كفاءة عن ذي قبل، حيث تـؤدي ذلـك من خلال بنـاء دوال رياضـيةً باسـتخدام التجريـدات الخاصـة بمكتبة Tensorflow تحديدًا ، كما تعمل حصريًا على وِحدة المعالجة الرسومية GPU الخاصة بك.
- NumPy: وهي مكتبة الجبر الخطي.

ستحتاج لإكمال هذا الفصل إلى بيئة برمجية للغة بايثون الإصدار 3.8 سـواءً كـان محليًـا أو بعيـدًا. ويجب أن تتضمن هذه البيئة البرمجية مدير الحِزم pip لتثبيت الحِـزم، ومُنشـئ البيئـات الافتراضـية venv لإنشـاء بيئـاتٍ افتراضيةٍ وهذا ما شرحناها في الفصل الأول.

4.3 إعداد بيئة المشروع

قبـل البـدء بهـذا الفصـل لا بـد من تجهـيز البيئـة المناسـبة، وسنسـتخدم محـرر الشـيفرات البرمجية Jupyter Notebooks، وهو مفيـد جـدًا لتجربـة وتشـغيل الأمثلـة الخاصـة بتَعَلّم الآلـة بطريقـةٍ تفاعليـةٍ، حيث تستطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسرعة، مما سيسـهل علينـا اختبـار الشيفرات البرمجية وتصحيحها.

يُمكنك فتح متصفح الويب والذهاب لموقع المحرر الرسمي jupyter على الوِيب لبدء العمـل بسـرعة، ومن ثمّ انقر فوق "جرّب المحرر التقليدي Try Classic Notebook"، وستنتقل بعـدها لملـفٍ جديـدٍ بـداخل محـرر Jupyter Notebooks التفاعلي، وبذلك تجهّز نفسك لكتابة الشيفرة البرمجية بلغة البايثون.

إذا رغبت بمزيدٍ من المعلومات حول محرر الشيفرات البرمجية Jupyter Notebooks وكيفيّة إعداد بيئتـه الخاصة لكتابة شيفرة بايثون، فيمكنك الاطلاع على: كيفية تهيئة تطبيق المفكرة jupyter notebook للعمل مع لغة البرمجة python.

4.4 إعداد المشروع

ستحتاج أولًا لتثبيت بعض التبعيـات، وذلـك لإنشـاء مسـاحة عمـلٍ للاحتفـاظ بملفاتنـا قبـل أن نتمكن من تطوير برنامج التعرف على الصور، وسنستخدم بيئة بايثون 3.8 الافتراضية لإدارة التبعيات الخاصة بمشروعنا.

سَنُنشئ مجلدًا جديدًا خاصًا بمشروعنا وسندخل إليه هكذا:

```
mkdir qlearning-demo
cd qlearning-demo
```

سننفذّ الأمر التالي لإنشاء البيئة الافتراضية:

```
python -m venv qlearning-demo
```

ومن ثم الأمر التالي في لينكس Linux لتنشيط البيئة الافتراضية:

```
source qlearning-demo/bin/activate
```

أما في ويندوز Windows، فيكون أمر التنشيط:

```
"qlearning-demo/Scripts/activate.bat"
```

سنستخدم إصداراتٍ محددةٍ من هذه المكتبـات، من خلال إنشـاء ملـف `requirements.txt` في مجلـد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

نفتح الملف `requirements.txt` في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبـات التي نريدها وإصداراتها:

```
tensorflow==2.6.0
numpy==1.19.5
cmake==3.21.4
tf-slim==1.1.0
```

```
atari-py==0.2.6
gym==0.19.0
matplotlib==3.5.0
PyVirtualDisplay==2.2
jupyter
keras==2.6.*
gym[all]
PyVirtualDisplay
```

سنحفظ التغييرات التي طرأت على الملف وسنخرج من محرر النصوص، ثم سنُثَبت هذه المكتبات بتنفيذ الأمر التالي:

```
(qlearning-demo) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، سنُصبح جاهزين لبدء العمل على مشروعنا.

4.5 كتابة الشيفرة البرمجية

شغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت. هكذا:

```
(qlearning-demo) $ jupyter notebook
```

أنشئ ملفًا جديدًا في داخل المحرر بالضغط على الزر new واختيار (ipykernal) python 3 وسمه باسم main مثلًا، حيث ستكون في الخلية الأولى للملف عملية استيراد الوِحدات (أو المكتبات) البرمجية اللازمة.

> لمزيد من المعلومات حول طريقة استيراد وحدة برمجية في لغة بايثون يمكنك الاطلاع على كيفية استيراد الوحدات في بايثون 3 سبق وأن ناقشنا فيه هذه الفكرة بالتفصيل.

نقوم أولًا بتثبيت مكتبة مساعدة في نمذجة الحركات الفيزيائية:

```
# تثبيت محرك المحاكاة الفيزيائية
!curl https://mujoco.org/download/mujoco210-linux-x86_64.tar.gz --
output mujoco210-linux-x86_64.tar.gz
!tar -xf mujoco210-linux-x86_64.tar.gz
import os
#إضافة متغير يدل على مسار المحرك
os.environ['MUJOCO_PY_MUJOCO_PATH'] = "/content/mujoco210"
```

ثم نقوم باستيراد جميع المكتبات اللازمة:

```
# مكتبة العمليات الرياضية
import numpy as np
# مكتبات الألعاب
import gym
from gym import logger as gymlogger
from gym.wrappers import Monitor
# مكتبات التعلم العميق
import tensorflow as tf
import tf_slim
# مكتبة للرسوميات
import matplotlib.pyplot as plt
%matplotlib inline
# تفعيل توافقية نسخ مكتبة التعلم العميق
tf.compat.v1.disable_eager_execution()
# مكتبة لاستخدام القائمة مفتوحة الطرفين
from collections import deque, Counter
# مكتبة للمسارات
import glob
# مكتبة دخل-خرج
import io
# 64 مكتبة ترميز
import base64
# HTML مكتبة عرض
from IPython.display import HTML
```

نُنشئ بيئة لعبة غزاة الفضاء `SpaceInvaders-v0` باستخدام الدالة `gym.make`. نستخدم الخاصية `action_space.n` لبيئة اللعبة لمعرفة عدد الأفعال الممكنة في اللعبة، ونستخدم الدالة `get_action_meanings` للحصول على أسماء هذه الأفعال الممكنة:

```
# تهيئة بيئة عمل اللعبة
env = gym.make("SpaceInvaders-v0")
# معاينة عدد الأفعال الممكنة
n_outputs = env.action_space.n
print(n_outputs)
# معاينة أسماء الأفعال الممكنة
print(env.get_action_meanings())
```

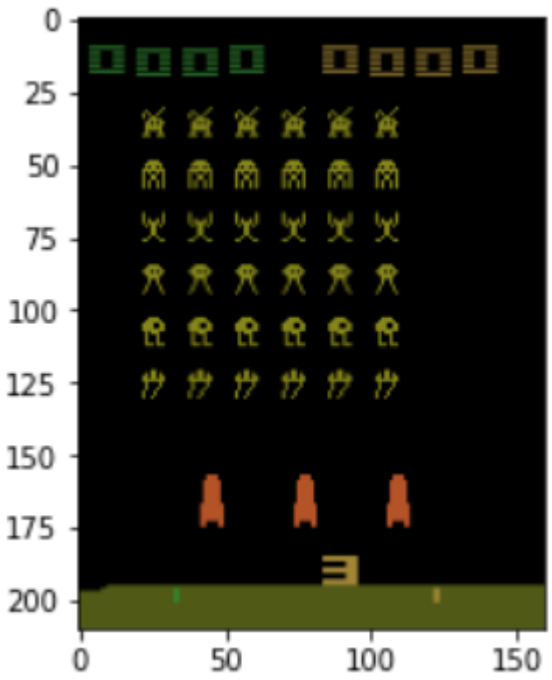
يُبين ناتج التنفيذ عدد الأفعال الممكنة في اللعبة وأسمائها:

```
6
['NOOP', 'FIRE', 'RIGHT', 'LEFT', 'RIGHTFIRE', 'LEFTFIRE']
```

يُمكن طباعة شاشة اللعبة الأولية وذلك بعد استدعاء دالة التهيئة الأولية لبيئة اللعبة `reset`:

```
# التهيئة الأولية
state = env.reset()
# معاينة الوضع الأولي للعبة
plt.imshow(state)
```

مما يُعطي صورة شاشة اللعبة التالية:



نستخدم فيما يلي الدالة الأساسية `env.step`، والتي تُعيد القيم التالية:

- الحالة state: وهي الحالة الجديدة للعبة بعد تطبيق فعلٍ معينٍ.
- المكافأة reward: وهي الزيادة في الدرجة المترتبة عن فعلٍ معينٍ. وفي نمط الألعاب المعتمدة على النقاط، يكون هدف اللاعب هو تعظيم المكافأة الإجمالية، بزيادة مجموع النقاط لأقصى درجةٍ ممكنةٍ.
- حالة انتهاء اللعبة done: تأخذ القيمة `true` عندما يخسر اللاعب ويفقد جميع الفرص المتاحة له.
- المعلومات info: وهي المعلومات الجانبية، وسنتجاوز شرحها حاليًا.

نشرح مبدأ اللعبة عبر الشيفرة التالية والتي تدخل في حلقة تُبقي اللعبة مستمرة إلى أن يخسر اللاعب بموته. تختار الدالة `sample` أحد الأفعال الستة الممكنة بشكل عشوائي. يؤدي تنفيذ الفعل باستخدام الدالة `step` إلى الانتقال إلى حالة جديدة مع مكافأة ممكنة في بعض الحالات. تنتهي اللعبة بموت اللاعب `done=true`.

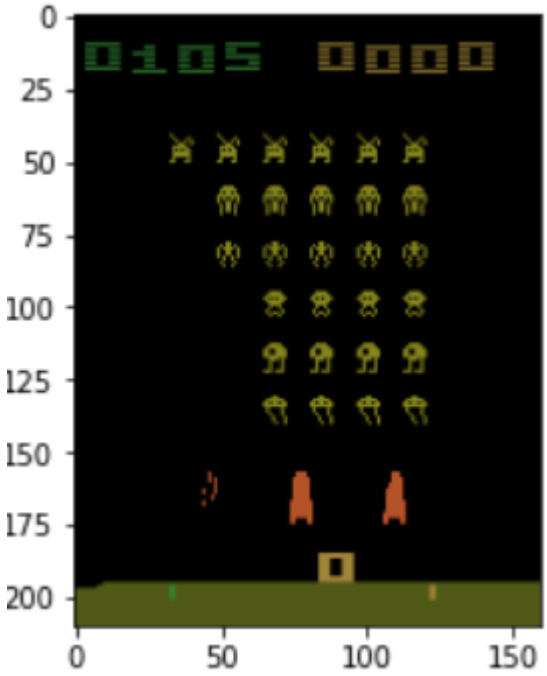
```
env.reset()
# المكافأة الإجمالية
episode_reward = 0
while True:
    # اختيار فعل بشكل عشوائي
    action = env.action_space.sample()
    # تنفيذ الفعل
    # الانتقال لحالة جديدة
    state, reward, done, info = env.step(action)
    if (reward>0):
        # طباعة القيم في حال مكافآة أكبر من الصفر
        print(env.get_action_meanings()[action]," ==> ",  reward, " ,
(",episode_reward,")" )
    # إضافة مكافأة الفعل للمكافأة الإجمالية
    episode_reward += reward
    # اختبار نهاية اللعبة
    if done:
        # طباعة المكافأة النهائية
        print('Reward: %s' % episode_reward)
        break
# معاينة الوضع النهائي للعبة
plt.imshow(state)
plt.show()
```

يكون الخرج مثلًا:

```
FIRE  ==>  5.0  ,( 0.0 )
NOOP  ==>  5.0  ,( 5.0 )
RIGHTFIRE  ==>  10.0  ,( 10.0 )
RIGHT  ==>  15.0  ,( 20.0 )
RIGHT  ==>  20.0  ,( 35.0 )
FIRE  ==>  25.0  ,( 55.0 )
```

```
RIGHT  ==>  10.0  ,( 80.0 )
RIGHTFIRE  ==>  15.0  ,( 90.0 )
Reward: 105.0
```

كما تُبين واجهة اللعبة:



نحتاج إلى إجراء معالجة أولية لصور اللعبة (والتي ستكون لاحقًا دخلًا لشبكات التعلم العصبية)، ولـذا نُعـرّف الدالة التالية والتي تقتطع الجزء الهام من صورة اللعبة وتُنفّذ بعض عمليات المعالجة عليها:

```
def preprocess_state(state):
    # تحجيم الصورة واقتطاع الجزء الهام منها
    img = state[25:201:2, ::2]
    # تحويل الصورة إلى درجات الرمادي
    img = img.mean(axis=2)
    color = np.array([210, 164, 74]).mean()
    # تحسين التباين في الصورة
    img[img==color] = 0
    # تطبيع القيم من -1 إلى +1
    img = (img - 128) / 128 - 1
    return img.reshape(88,80)
```

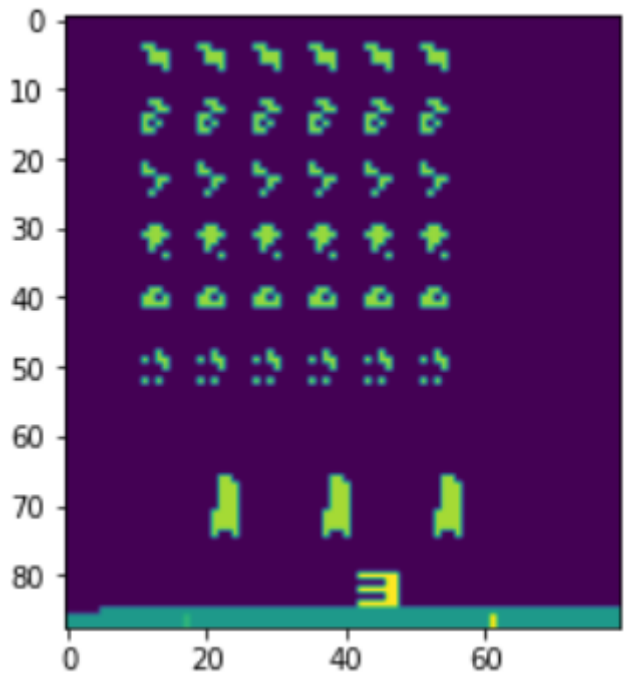
نستدعي الدالة السابقة مثلًا على صورة بداية اللعبة لنعاين ناتج المعالجة:

```
# التهيئة الأولية
state = env.reset()
# معاينة الصورة بعد عمليات التهيئة
state_preprocessed = preprocess_state(state).reshape(88,80)
print(state.shape)
print(state_preprocessed.shape)
plt.imshow(state_preprocessed)
plt.show()
```

يكون الخرج:

```
(210, 160, 3)
(88, 80)
```

والصورة المُعالجة:



نُعرّف فيما يلي الدالة `stack_frames` والتي تُجهز قائمة من الصور عـددها (4 مثلًا)، نخـزن العـدد ضـمن المتغير `stack_size`. تحوي القائمة نفس صورة البداية مكررة العدد نفسه `stack_size` مرة عنـد بدايـة كـل حلقة episode لعب.

لاحظ أننا نستخدم البنية الخاصة في بـايثون المـدعوة deque (قائمـة مفتوحـة الطـرفين) والـتي يُمكن لنـا الإضافة عليها والحذف منها وذلك من طرفيها.

```
# نستخدم 4 صور متعاقبة لملاحقة الحركة
stack_size = 4
# تهيئة قائمة مفتوحة الطرفين
# 0 تحوي أربعة صور ذات قيم
stacked_frames  =  deque([np.zeros((88,80), dtype=np.int) for i in
range(stack_size)], maxlen=stack_size)
def stack_frames(stacked_frames, state, is_new_episode):
    # المعالجة الأولية للصورة
    frame = preprocess_state(state)
    # حلقة جديدة
    if is_new_episode:
        # مسح الصور السابقة المكدّسة
        stacked_frames = deque([np.zeros((88,80), dtype=np.int) for i
in range(stack_size)], maxlen=stack_size)
        # تكرار الصورة المعالجة في كل حلقة جديدة من اللعبة
        for i in range(stack_size):
            stacked_frames.append(frame)
        # تكديس الصور
        stacked_state = np.stack(stacked_frames, axis=2)
    else:
        # بما أن القائمة مفتوحة الطرفين تضيف من أقصى اليمين
        # نُحضر الصورة أقصى اليمين
        frame=stacked_frames[-1]
        # إضافة الصورة
        stacked_frames.append(frame)
        # إنشاء الحالة
        stacked_state = np.stack(stacked_frames, axis=2)
    return stacked_state, stacked_frames
```

نُعرّف شبكة عصبية مؤلفة من ست طبقات: ثلاثة طبقات تلافيفية، تليها طبقة مسـطحة flatten، ومن ثم طبقة موصولة بشكل كامل fully connected، تليها طبقة الخرج.

```
def q_network(X, name_scope):
    # تهيئة الطبقات
    initializer =
tf.compat.v1.keras.initializers.VarianceScaling(scale=2.0)
    with tf.compat.v1.variable_scope(name_scope) as scope:
        # تهيئة الطبقات التلافيفية
```

```
        layer_1 = tf_slim.conv2d(X, num_outputs=32, kernel_size=(8,8),
stride=4, padding='SAME', weights_initializer=initializer)
        tf.compat.v1.summary.histogram('layer_1',layer_1)
        layer_2 = tf_slim.conv2d(layer_1, num_outputs=64,
kernel_size=(4,4), stride=2, padding='SAME',
weights_initializer=initializer)
        tf.compat.v1.summary.histogram('layer_2',layer_2)
        layer_3 = tf_slim.conv2d(layer_2, num_outputs=64,
kernel_size=(3,3), stride=1, padding='SAME',
weights_initializer=initializer)
        tf.compat.v1.summary.histogram('layer_3',layer_3)
        # تسطيح نتيجة الطبقة الثالثة قبل تمريرها إلى الطبقة
        # التالية الموصولة بشكل كامل
        flat = tf_slim.flatten(layer_3)
        # إدراج الطبقة الموصولة بشكل كامل
        fc = tf_slim.fully_connected(flat, num_outputs=128,
weights_initializer=initializer)
        tf.compat.v1.summary.histogram('fc',fc)
        # إضافة طبقة الخرج النهائية
        output = tf_slim.fully_connected(fc, num_outputs=n_outputs,
activation_fn=None, weights_initializer=initializer)
        tf.compat.v1.summary.histogram('output',output)
        # تخزين معاملات الشبكة كأوزان
        vars = {v.name[len(scope.name):]: v for v in
tf.compat.v1.get_collection(key=tf.compat.v1.GraphKeys.TRAINABLE_VARIA
BLES, scope=scope.name)}
        # إرجاع كل من المتغيرات والخرج
        #Return both variables and outputs together
        return vars, output
```

نُعرّف الدالة epsilon_greedy لتنفيذ السياسة المدعوة بسياسة إبسلون الشرهة epsilon greedy policy والتي تختار أفضل فعل باحتمال "1 - إبسلون" أو أي فعل آخر عشوائي باحتمال إبسلون.

لاحظ أننا نُخفّض قيمة إبسلون مع مرور الوقت (زيادة عدد الخطوات) كي تستخدم هذه السياسة الأفعال الجيدة فقط.

```
epsilon = 0.5
eps_min = 0.05
eps_max = 1.0
eps_decay_steps = 500000
```

```
def epsilon_greedy(action, step):
    p = np.random.random(1).squeeze() # متجهة أحادية
    epsilon = max(eps_min, eps_max - (eps_max-eps_min) *
step/eps_decay_steps)
    if p < epsilon:
        return np.random.randint(n_outputs)
    else:
        return action
```

نهيئ فيما يلي ذاكرة تخزين مؤقت من النمط قائمة مفتوحة الطرفين deque لنضع فيها معلومـات اللعب من الشكل SARSA.

نُخزّن كل خبرة الروبوت أي (الحالة، الفعل، المكافأة) state, action, rewards في الـذاكرة المؤقتـة وهي الخبرة المكتسبة المفيدة عند إعـادة اللعب experience replay buffer ونقـوم لاحقًـا بأخـذ عينـات منهـا y-values لتوليد القيم التي سنُدرب الشبكة عليها:

```
buffer_len = 20000
# نستخدم لذاكرة التخزين المؤقت قائمة مفتوحة الطرفين
exp_buffer = deque(maxlen=buffer_len)
```

نُعرّف الدالة `sample_memories` التي تنتقي مجموعة عشوائية من تجارب التدريب:

```
# اختيار مجموعة عشوائية من تجارب التدريب
# بطول حجم دفعة التدريب
def sample_memories(batch_size):
    perm_batch = np.random.permutation(len(exp_buffer))[:batch_size]
    mem = np.array(exp_buffer)[perm_batch]
    return mem[:,0], mem[:,1], mem[:,2], mem[:,3], mem[:,4]
```

نُعرّف فيما يلي جميع المعاملات المُترفعة hyperparameters للشبكة مع قيمهـا. اخترنـا هـذه القيم بعـد بناء على الخبرة في الشبكات العصبية وبعد العديد من التجارب.

```
num_episodes = 2000
batch_size = 48
input_shape = (None, 88, 80, 1)
learning_rate = 0.001
# التعديل لملائمة الصور المكدّسة
X_shape = (None, 88, 80, 4)
```

```
discount_factor = 0.97
global_step = 0
copy_steps = 100
steps_train = 4
start_steps = 2000
```

نُعرّف حاوية الدخل X:

```
tf.compat.v1.reset_default_graph()
# تعريف حاوية دخل الشبكة العصبية أي حالة اللعبة
X = tf.compat.v1.placeholder(tf.float32, shape=X_shape)
# تعريف متغير بولياني لقلب حالة التدريب
in_training_mode = tf.compat.v1.placeholder(tf.bool)
```

نبني فيما يلي شبكتين: الشبكة الأولية والشبكة الهدف مما يسمح بتوليد البيانات والتدريب بشكل متزامن:

```
# بناء الشبكة التي تولد جميع قيم التعلم لجميع الأفعال في الحالة
mainQ, mainQ_outputs = q_network(X, 'mainQ')

# بشكل مشابهة نبني الشبكة الهدف لقيم التعلم
targetQ, targetQ_outputs = q_network(X, 'targetQ')

# تعريف حاوية قيم الأفعال
X_action = tf.compat.v1.placeholder(tf.int32, shape=(None,))
Q_action = tf.reduce_sum(input_tensor=targetQ_outputs *
tf.one_hot(X_action, n_outputs), axis=-1, keepdims=True)

# نسخ قيم معاملات الشبكة إلى الشبكة الهدف
copy_op = [tf.compat.v1.assign(main_name, targetQ[var_name]) for
var_name, main_name in mainQ.items()]
copy_target_to_main = tf.group(*copy_op)
```

نُعرّف فيما يلي الخرج ونقوم بحساب الخسارة:

```
# تعريف حاوبة الخرج
y = tf.compat.v1.placeholder(tf.float32, shape=(None,1))
# حساب الخسارة والتي هي الفرق بين القيمة الحقيقية والقيمة المتوقعة
loss = tf.reduce_mean(input_tensor=tf.square(y - Q_action))
```

```
# تحسين الخسارة
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(loss)
```

نبدأ الآن جلسة Tensorflow ونُشغّل النموذج model:

1. نُجري أولًا المعالجة الأولية لصورة شاشة اللعب ومن ثم نُمرر الناتج (الحالة s) إلى الشبكة العصبية DQN والتي تُعيد جميع قيم التعلم Q-values.
2. نختار أحد الأفعال a باستخدام سياسة إبسلون الجشعة.
3. نُنفذ الفعل a على الحالة s وننتقل إلى حالة جديدة snew مع حصولنا على مكافأة (تكون الحالة الجديدة هي الصورة المعالجة لشاشة اللعب التالية)
4. نُخزّن هذا الانتقال في الذاكرة المؤقتة على الشكل <s,a,r,snew>.
5. ننتقي بعض الانتقالات من الذاكرة العشوائية ونحسب الخسارة.
6. نُعدّل معاملات الشبكة لتخفيض الخسارة.
7. ننسخ أوزان شبكة التدريب إلى الشبكة الفعلية.
8. نكرر الخطوات السابقة عددًا من المرات (num_episodes).

```
with tf.compat.v1.Session() as sess:
    init = tf.compat.v1.global_variables_initializer()
    init.run()

    # من أجل كل دورة
    history = []
    for i in range(num_episodes):
        done = False
        obs = env.reset()
        epoch = 0
        episodic_reward = 0
        actions_counter = Counter()
        episodic_loss = []

        # تكديس الصور في الخطوة الأولى
        obs,stacked_frames= stack_frames(stacked_frames,obs,True)
```

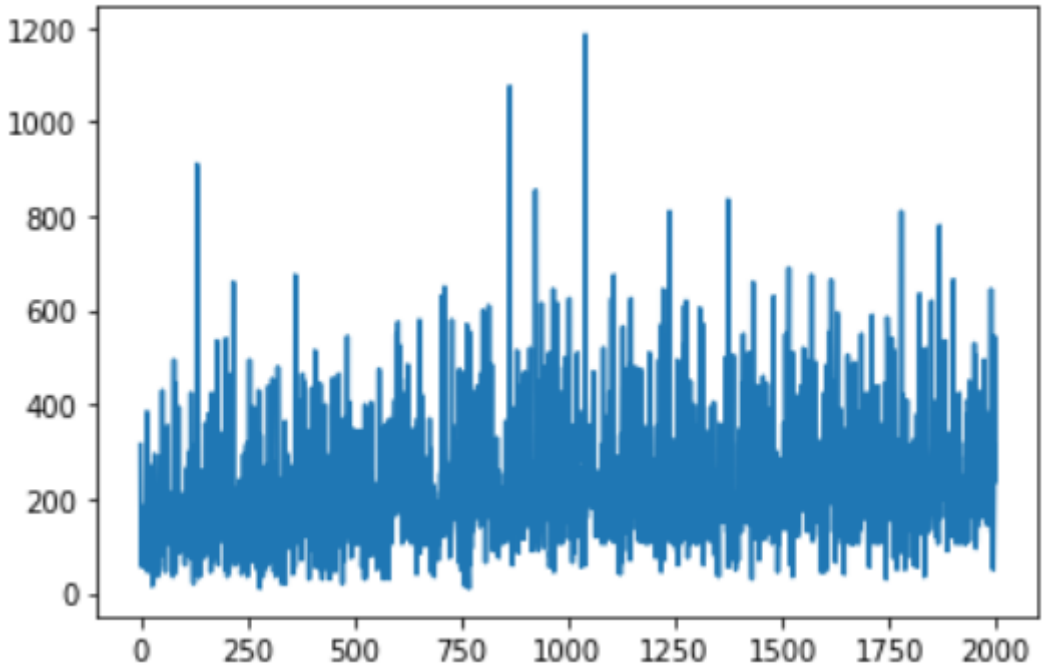
```
        # الدوران طالما لم نصل للحالة النهائية
        while not done:
            # توليد البيانات باستخدام الشبكة غير المدربة
            # إدخال صورة اللعبة والحصول على قيم التعلم
            # من أجل كل فعل
            actions = mainQ_outputs.eval(feed_dict={X:[obs],
in_training_mode:False})

            # اختيار  الفعل
            action = np.argmax(actions, axis=-1)
            actions_counter[str(action)] += 1

            # استخدام سياسة ابسلون الشرهة لاختيار الفعل
            action = epsilon_greedy(action, global_step)

            # تنفيذ الفعل
            # والانتقال للحالة التالية وحساب المكافآة
            next_obs, reward, done, _ = env.step(action)
            # تكديس مابين الدورات
            next_obs, stacked_frames = stack_frames(stacked_frames,
next_obs, False)

            # تخزين الانتقال كتجربة في
            # الذاكرة المؤقتة لإعادة اللعب
            exp_buffer.append([obs, action, next_obs, reward, done])

            # تدريب الشبكة من الذاكرة المؤقتة لإعادة اللعب بعد عدة خطوات معينة
            if global_step % steps_train == 0 and global_step >
start_steps:
                # تحوي الذاكرة المؤقتة لإعادة اللعب
                # كل ما تمت معالجته وتكديسه
                # mem[:,0], mem[:,1], mem[:,2], mem[:,3], mem[:,4]
                o_obs, o_act, o_next_obs, o_rew, o_done =
sample_memories(batch_size)

                # الحالات
                o_obs = [x for x in o_obs]
```

```
                # الحالات التالية
                o_next_obs = [x for x in o_next_obs]


                # الأفعال التالية
                next_act = mainQ_outputs.eval(feed_dict={X:o_next_obs,
in_training_mode:False})


                # المكافآت
                y_batch = o_rew + discount_factor * np.max(next_act,
axis=-1) * (1-o_done)


                train_loss, _ = sess.run([loss, training_op],
feed_dict={X:o_obs, y:np.expand_dims(y_batch, axis=-1),
X_action:o_act, in_training_mode:True})
                episodic_loss.append(train_loss)


            # نسخ أوزان الشبكة الرئيسية إلى الشبكة الهدف
            if (global_step+1) % copy_steps == 0 and global_step >
start_steps:
                copy_target_to_main.run()


            obs = next_obs
            epoch += 1
            global_step += 1
            episodic_reward += reward
        next_obs=np.zeros(obs.shape)
        exp_buffer.append([obs, action, next_obs, reward, done])
        obs= env.reset()
        obs,stacked_frames= stack_frames(stacked_frames,obs,True)
        history.append(episodic_reward)
        print('Epochs per episode:', epoch, 'Episode Reward:',
episodic_reward,"Episode number:", len(history))
```

نرسم المكافآت وفق الحلقات:

```
plt.plot(history)
plt.show()
```

مما يُعطي مثلًا:

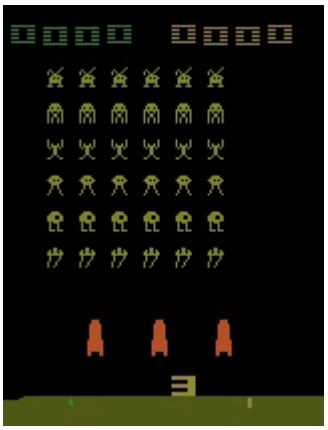


يُمكن في النهاية معاينة أداء الروبوت وتسجيل فيديو للعبة:

```
# تسجيل فيديو بهدف التقييم
gymlogger.set_level(40)
def show_video():
  # مسار ملف الفيديو
  mp4list = glob.glob('video/*.mp4')
  # التأكد من وجود الفيديو
  if len(mp4list) > 0:
    mp4 = mp4list[0]
    # فتح ملف الفيديو
    video = io.open(mp4, 'r+b').read()
    # التحويل لكود 64
    encoded = base64.b64encode(video)
    # عرض الفيديو في حالة حاسوب محلي
    # بتوليد HTML
    ipythondisplay.display(HTML(data='''<video alt="test" autoplay
                loop controls style="height: 400px;">
                <source src="data:video/mp4;base64,{0}"
type="video/mp4" />
             </video>'''.format(encoded.decode('ascii'))))
  else:
```

```
    print("Could not find video")
def wrap_env(env):
  env = Monitor(env, './video', force=True)
  return env

# تقييم النموذج
environment = wrap_env(gym.make('SpaceInvaders-v0'))
done = False
observation = environment.reset()
new_observation = observation
prev_input = None
with tf.compat.v1.Session() as sess:
    init.run()
    observation, stacked_frames = stack_frames(stacked_frames,
observation, True)
    while True:
        # الحصول على قيم التعلم
        actions = mainQ_outputs.eval(feed_dict={X:[observation],
in_training_mode:False})

        # الحصول على الفعل
        action = np.argmax(actions, axis=-1)
        actions_counter[str(action)] += 1

        # اختيار الفعل باستخدام سياسة إبسلون الشرهة
        action = epsilon_greedy(action, global_step)
        # يجب إضافة التعليمة التالية على حاسوب محلي
        # environment.render()
        new_observation, stacked_frames = stack_frames(stacked_frames,
new_observation, False)
        observation = new_observation
        # تنفيذ الفعل و الانتقال للخطوة التالية
        new_observation, reward, done, _ = environment.step(action)
        if done:
          break
    environment.close()
```

ستحصل على ناتج فيديو لديك يشبه الصورة المتحركة التالية (اضغط عليها لتراها في متصفح الويب):



4.6 الخلاصة

بنينا في هذا الفصل روبوتًا يلعب لعبة غزو الفضاء وذلك باستخدام التعلم المعزز العميق وفق النموذج الحُر Deep Q-learning.

السؤال الطبيعي الذي سيخطر ببالنا هو، هل يمكننا بناء روبوتاتٍ لألعابٍ أكثر تعقيدًا مثل لعبة StarCraft 2؟ فكما اتضح لنا، إن هذا سؤالٌ بحثيٌ معلقٌ ومدعومٌ بأدوات مفتوحة المصدر من شركاتٍ كبيرةٍ مثل غوغل Google وديب مايند DeepMind وشركة بليزارد Blizzard. فإذا كانت هذه المشاكل تهمك فعلًا، فيمكنك الاطلاع على المشاكل الحالية التي يواجهونها.

> يُمكن تجربة المثال كاملًا من موقع Google Colab من هذا الرابط.

4.7 المصادر

- Optimized Space Invaders using Deep Q-learning: An Implementation in Tensorflow 2.0
- Getting Started with Gym
- SpaceInvaders-v0
- Python Machine Learning Projects

5. تصنيف الصور والتعرف على الوجه

تتفق جميع التقنيات والخوارزميات الـتي تنـدرج تحت مسـمى الـذكاء الصـنعي Artificial Intelligence وتعلم الآلـة Machine Learning بـاختلاف أطيافهـا على البـدء بتحليـل البيانـات الخـام والبحث فيهـا (بطـرق خوارزمية ممنهجة) لاستكشاف ما تحتويه من أنمـاط وعلاقـات في داخلهـا وبين كياناتهـا. إن معرفـة مثـل تلـك الأنماط والعلاقات ستساعدنا على بناء نماذج قادرة على استقراء الجوهر العام المشترك الذي يجمع ويصنف تلك الكتل من البيانات على الرغم من الاختلافات في التفاصيل الهامشية والتي قد تكون كبـيرة ظاهريـا ومربكـة في عين من لا يتمتع بالخبرة في مجال الدراسة.

لقد أثبتت مثل هذه النظم فعاليتها في حوسبة العديد من المهام الـتي كـان يعتقـد حـتى وقت قـريب أنهـا عصية على عالم البرمجيات، ساعد على ذلك الطفرة التي شهدتها عتاديات الحواسيب في العقد الأخـير والتقـدم الهائل في إمكانيات الحوسبة المتوازية لوحدات المعالجة الرسومية GPU، وكذلك توافر حلول الحوسبة السحابية لشـريحة أكـبر من فـرق التطـوير البرمجيـة الـتي كـان يصـعب عليهـا في الماضـي الحصـول على حواسـيب فائقة بسهولة.

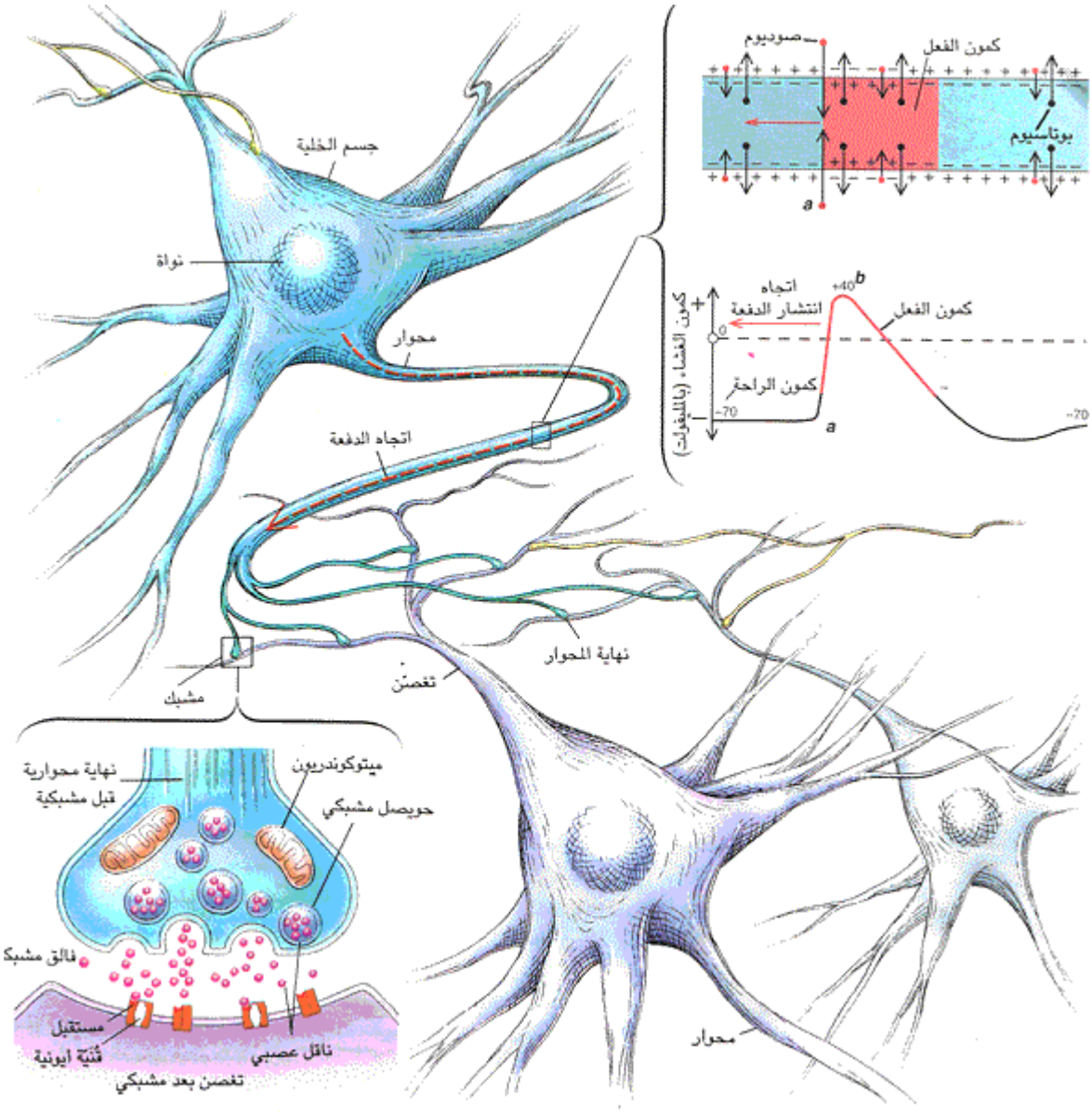
إننا نشهد في الفترة الحالية انتقالا وتحولا جذريا في مفاهيم علم البرمجة مقارنة بمـا سـبق وبمـا تعلمنـاه أو استخدمناه خلال القرن الماضي (كالبرمجة الوظيفية ومخططات سير العمل، أو حـتى البرمجـة الغرضـية التوجـه بكائناتها وطرائقها)، فالمبرمج هو من كان يضـع منطـق العمـل وخوارزميتـه ليحـدد أسـلوب معالجـة المـدخلات وتوليد المخرجات، وقد امتازت فترة ثمانينيات وتسعينيات القرن الماضي بوفرة القدرة الحاسوبية مقارنة بكميـة البيانات المتوافرة للمعالجة أصلا، لكن ما نشهده الآن في القرن الحادي والعشرين قد قلب هـذه المعادلـة رأسـا على عقب، فقد أصبح لدينا فائض كبير من البيانات التي تجمع من كل حدب وصوب على مدار الساعة بـدءا من جوالاتنا (التي ما عادت وسيلة للاتصال فقط) وصولا إلى الأقمار الاصطناعية في مداراتها حول الأرض.

إن هذا المستوى العالي من الأتمتة في حياتنا ومجتمعاتنا نتج عنه كم هائل من البيانات التي تجاوزت بمراحل القدرة على تحليلها واستخلاص المعلومات منها بالطرق التقليدية، وهو ما أعاد فتح الأبواب مجددا لتقنيات الذكاء الصنعي التي كان لها مجد سابق منتصف القرن الماضي لكنه زال بعد أن عجزت القدرات العتادية في ذلك الحين عن مواكبة شطحات المتنورين من رواد هذا الحقل من العلوم، لكننا اليوم وبتضافر جهود الباحثين في مجالات شتى مثل علم البيانات والإحصاء والحواسيب بلا شك، أصبحنا قادرين على تنفيذ ما كانوا يتحدثون عنه ضمن طيف واسع من التطبيقات ربما بات يتجاوز أقصى طموحاتهم.

5.1 الشبكات العصبية Neural Networks

في جزء من لوحة رافائييل الجصيّة "مدرسة أثينا" تم رسم أفلاطون وأرسطو بشكل يعبّر عن نظريّة كل منهما في المعرفة، فأرسطو يومئ نحو الأرض فيما يشير أفلاطون بإصبعه نحو السماء، حيث كان ينظر أرسطو إلى الطبيعة بحثا عن إجابات في حين أن أفلاطون يبحث عن المثالي. وها نحن ننتقل في عالم البرمجة من فلسفة أفلاطون التي كنا نتّبعها في أدوات بناء تطبيقاتنا من لغات برمجة وما تحويه من متغيرات وتوابع وعبارات شرطية وحلقات وسواها، إلى فلسفة أرسطو حيث نبحث في كيفية عمل أدمغتنا ونحاول محاكاتها في بنيتها وطريقة عملها، وهكذا تماما ولد مجال الشبكات العصبية ضمن علوم الذكاء الصنعي.

حيث ابتدأت المحاكاة على مستوى الخلية العصبية (العصبون)، فالعصبونات هي خلايا تتألف من جسم مركزي يتضمن نواة الخلية ويمتد منه استطالة وحيدة تكوّن ليف عصبي طويل يقوم مقام السلك الناقل للإشارة الخارجة من الخلية عند نهايته تفرعات كثيرة تنتهي بعقد مشبكية صغيرة، أما في الطرف الآخر من جسم الخلية العصبية فتبرز تفرّعات تخرج في كافة الاتجاهات والتي عن طريقها ترد إلى جسم الخلية البيانات الداخلة، ولابد للإشارات لكي تمر من عصبون إلى آخر أن تجتاز الفجوة الضيقة ما بين عقدة المشبك والجسم أو التفرّع للعصبون التالي.



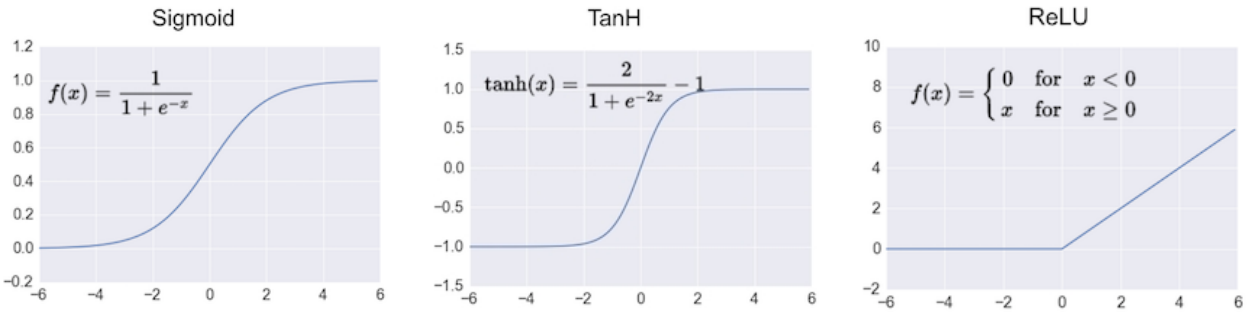
تتلقى الخلية العصبية عدة تنبيهات من الخلايا المجاورة تؤدي إلى شحنها، فإذا وصلت تلك الشحنة إلى عتبة معينة ينبثق كمون كهربائي عند قاعدة المحور وينتشر دفعة واحدة على طوله. لا تستجيب العصبونات لمختلف التنبيهات بشكل متشابه، فلكل منبه درجة أهمية تزيد أو تنقص، كما أن خرج العصبون لا يمتاز بالتدرج، فإما أن تكون هناك نبضة عصبية تنتشر عبر المحور إلى الخلايا المجاورة أو لن يكون هنالك شيء على الإطلاق. طبعا هذا الوصف لبنية العصبون وطريقة عمله فيه تبسيط شديد لحقيقة الأمر، لكن هذا المستوى من الشرح يفي بالغرض.

على الرغم من أننا عندما نحاكي عمل العصبون (بفرض أننا نعلم طريقة عمله تماما) لن نحصل على عصبون حقيقي، لكننا سنحصل على معالجة حقيقية للمعلومات كما لو كان العصبون هو من قام بها، وهذا هو بيت القصيد. إن التمثيل الرياضي للعصبون يفترض أن لدينا n إشارة دخل سنرمز إليها بالمقادير X1, X2, ..., Xn يرتبط كل منها بوزن أو تثقيل للتعبير عن دور المشابك العصبية كون العصبونات لا تستجيب بشكل متشابه

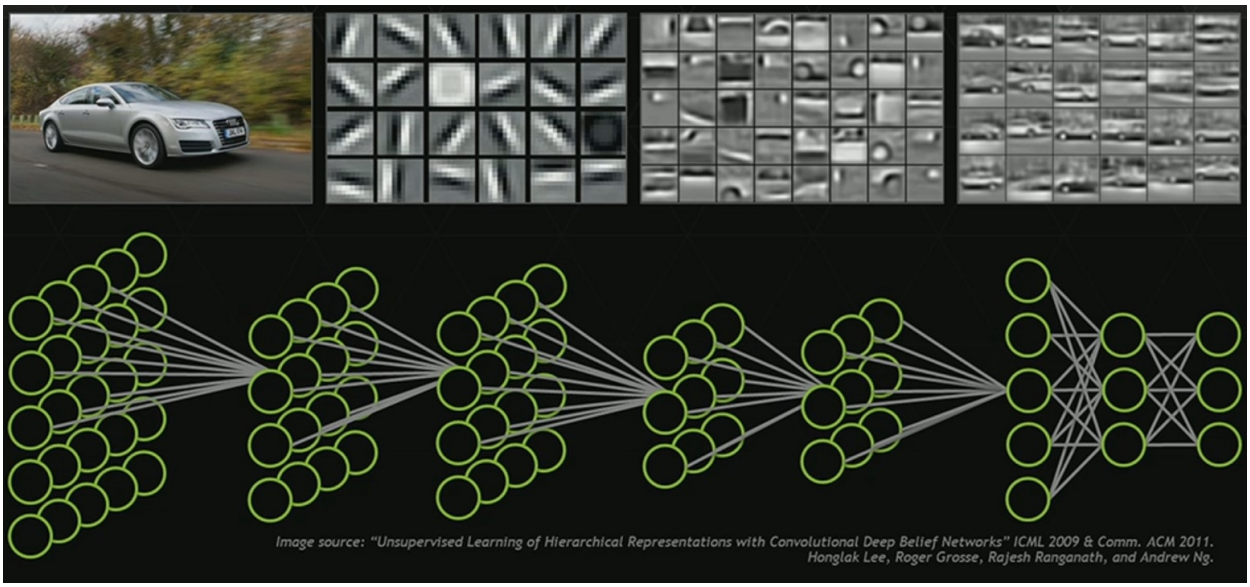
لمختلف التنبيهات كما اتفقنا سابقا. سنرمز إلى تلك الأوزان بالمقادير W1, W2, ..., Wn والتي قد تكون قيما موجبة أو حتى سالبة تكبر أو تصغر بحسب طبيعة ودور إشارة الدخل المرتبطة بها سواء كانت محفّزة أم مثبّطة وإلى أي قدر هي كذلك. وهكذا يمكننا التعبير عن مجمل الدخل الآتي إلى العصبون المفرد بالشكل الرياضي التالي:

$$X = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n = \sum_{i=1}^{n} w_i * x_i$$

من جهة أخرى، لتمثيل إشارة خرج العصبون نحتاج إلى دالة رياضية تستطيع توصيف عمل قانون الكل أو لا شيء تبعا لعتبة معيّنة، وهناك عدّة خيارات رياضية شائعة قد يتم تفضيل إحداها على الأخرى بحسب طبيعة البيانات التي نتعامل معها نذكر منها على سبيل المثال لا الحصر: Sigmoid (للقيم الثنائية 0/1 أو نعم/لا أو حتى ذكر/أنثى)، TanH (للفئات أو التصنيفات المتقطعة مثل الأعراق: عربي، أوروبي، أفريقي، آسيوي، هندي، الخ.)، وكذلك ReLU (للقيم المتصلة كالعمر مثلا). يوضح الشكل التالي الصيغة الرياضية والتمثيل البياني الذي يظهر العلاقة فيما بين الدخل والخرج لكل منها:



بعد أن قمنا بتوصيف العصبون بشكل رياضي مبسّط، علينا الانتقال للحديث عن بنية الشبكات التي تنتظم بها تلك العصبونات وطرق ارتباطها بعضها ببعض، سنتناول بالشرح شكلا واحدا من أشكال بناء الشبكات العصبية، لكن ذلك لا يعني عدم وجود معماريات أخرى لبناء تلك الشبكات. سنفترض أن العصبونات تنتظم في طبقات ولكل طبقة عدد من العصبونات بحيث أن خرج أي عصبون من هذه الطبقة يتم إيصاله إلى كل عصبونات الطبقة التالية.



يمكن تمييز الطبقة الأولى على أنها طبقة الإدخال والتي تتلقى بياناتها من الوسط الخارجي للشبكة العصبية، كذلك نعرّف طبقة الإخراج على أنها الطبقة الأخيرة والتي ترسل ناتج معالجة البيانات ثانية إلى الوسط الخارجي، في حين يسمّى كل ما عدى ذلك من طبقات بين هاتين الطبقتين بالطبقات الخفية. كما نلاحظ فإن عدد العصبونات في كل من طبقتي الإدخال والإخراج محدد بحسب طبيعة الوظيفة المناطة بالشبكة العصبية (مثلا عدد العنصورات Pixels الإجمالي في الصورة لكل قناة لونية RGB في الدخل، وعدد الفئات المراد تصنيفها في الخرج)، أمّا عدد الطبقات الخفية وعدد العصبونات في كل منها فلا توجد قواعد ضابطة وواضحة لذلك.

5.2 مكتبة TensorFlow من Google

مكتبة TensorFlow هي منصة متكاملة لتدريب وبناء تطبيقات الذكاء الصنعي وتعلم الآلة بالاعتماد على تقنية الشبكات العصبية طورتها شركة Google باستخدام لغة Python ونشرتها تحت ترخيص البرمجيات الحرة والمفتوحة المصدر، وهي تعد في الوقت الراهن واحدة من أكثر المكتبات شهرة واستخداما في هذا المجال (رغم أنها ليست الوحيدة قطعا)، وذلك نظرا لغزارة وتنوع المصادر والأدوات المتوافرة لها والتي تتيح للباحثين القدرة على بناء واستخدام تطبيقات الذكاء الصنعي في أعمالهم. للمزيد حول TensorFlow يمكنك الإطلاع على الموقع الرسمي لها على الرابط التالي: https://www.tensorflow.org.

5.3 مكنز TensorFlow Hub للنماذج

إحدى أهم التحديات في عالم الذكاء الصنعي بشكل خاص، وعلوم البيانات بشكل عام، هي القدرة على إعادة استخدام ما سبق وما توصل إليه فريق تطوير آخر سابقًا، لذا قدمت Google هذا المكنز لوضع طريقة معيارية في مشاركة نماذج الشبكات العصبية بحيث تتضمن كافة المعلومات المطلوبة لإعادة استخدامها سواء كانت بنية الشبكة العصبية ذاتها (من حيث عدد الطبقات، ونوعها، وعدد العصبونات في كل منها، ونوع الروابط فيما

بينها، ...إلخ.)، إضافة إلى قيم الوسطاء والأوزان المختلفة في تلك الشبكة العصبية بعد إتمام عملية تدريبها. يمكن أخذ هذه النماذج وإعادة استخدامها في مهام مختلفة أو حتى إعادة تدريبها بسهولة مستخدمين تقنية تدعى نقل التعلم والتي سنتحدث عنها لاحقا في هذا الفصل. لمزيد من المعلومات حول هذا المكنز يمكنكم زيارة الموقع الرسمي له على الرابط التالي: https://www.tensorflow.org/hub.

5.4 نموذج MobileNet V2 للرؤية الحاسوبية

صُمِّمت هذه العائلة من نماذج الشبكات العصبية المخصَّصة لمهام الرؤية الحاسوبية عامة الأغراض مثل التصنيف وتحديد الأجزاء والعناصر في الصورة وسواها من الوظائف مع مراعاة المصادر المحدودة التي قد تكون متوافرة على الأجهزة المحمولة. إن القدرة على تشغيل تطبيقات التعلم العميق على أجهزة الجوال الشخصية ستحسن من تجربة المستخدم نظرا لأنها ستكون متاحة في أي زمان ومكان بغض النظر عن الحاجة إلى الاتصال بمصادر خارجية على الإنترنت، وهو ما سيترافق مع فوائد إضافية لجهة الأمان والخصوصية والاقتصاد في استهلاك الطاقة. لمزيد من المعلومات يمكنك الإطلاع على الرابط المدرج ضمن قسم المراجع رقم 1.

5.5 تقنية نقل التعلم Transfer Learning

إن النماذج الحديثة للتعرف على الصور تحتوي على الملايين من الوسطاء (من أوزان للروابط ما بين الآلاف من العصبونات المرصوفة في العشرات من الطبقات الخفية) والتي يتطلب تدريبها من الصفر كما كبيرا من بيانات التدريب من جهة، والكثير من الطاقة الحاسوبية من جهة أخرى (تقدّر بالمئات من ساعات الحساب على وحدات المعالجة الرسومية GPU أو حتى أكثر بكثير). إن تقنية نقل التعلم تعتمد على حيلة ذكية لاختصار كم المصادر الكبير الذي نحتاج إليه لتطوير نموذج رؤية حاسوبية جديد تخصص مهاراته في التعرف على نمط مختلف من الصور (مثلا صور الأشعة السينية لتشخيص وجود أورام سرطانية محتملة عوض التعرف على الأنواع المختلفة من الأزهار البرية).

تقوم الفكرة على استبدال الطبقة الأخيرة فقط من شبكة عصبية سبق وأن تم تدريبها بشكل جيد على تصنيف الصور ولو لغايات مختلفة، مستفيدين بذلك من كم المهارات التي اكتسبتها بنية الطبقات الخفية في ذلك النموذج، ابتداء من التعرف على أنماط النسج والأشكال وترابط الأجزاء وعلاقات الألوان وغيرها مما هو مشترك بالعموم بين كافة نظم الرؤية الحاسوبية، والتي يمكن إعادة استخدامها ومشاركتها بين النماذج المختلفة. وحدها الطبقة الأخيرة فقط الخاصة بالأصناف تتغير بتغير الغاية والهدف من النموذج الذي يجري تطويره وتخصيصه، لذا هي وحدها التي سيتم تدريبها فعليا عند استخدام تقنية نقل التعلم هذه. لمزيد من المعلومات حول هذه التقنية في التعليم يمكنكم الإطلاع على رابط ورقة البحث العلمي المدرجة ضمن قسم المراجع أدناه رقم 2.

5.6 تجهيز بيئة العمل

سنستخدم منصة دوكر Docker المخصصة لتطوير ونشر وإدارة التطبيقات باستخدام فكرة الحاويات وذلك لتسهيل بناء بيئة العمل لدينا من أجل تنفيذ التطبيق العملي في هذه الجلسة، حيث أن الحاويات هي عبارة عن حزم تنفيذية خفيفة وقائمة بذاتها لتطبيق ما، تحوي كل المكتبات وملفات الإمداد والاعتماديات والأجزاء الأخرى الضرورية ليعمل التطبيق ضمن بيئة معزولة، هذا عدى عن أنها خفيفة لأنها لا تتطلب حملا إضافيا كالأجهزة الافتراضية كونها تعمل ضمن نواة النظام المضيف مباشرة دون الحاجة إلى نظام ضيف، وبذلك نزيل عن كاهلنا في هذه المرحلة أي تعقيدات تختص بالتنصيب والربط والإعداد لمختلف مكونات بيئة التطوير الخاصة بمكتبة TensorFlow وهي مهمة ليست باليسيرة على المبتدئ، لذا عليك القيام بتثبيت دوكر Docker على حاسوبك الشخصي من الموقع الرسمي https://www.docker.com قبل الانتقال إلى الخطوة التالية.

> على الرغم من كون الحاوية المخصصة لمكتبة TensorFlow والتي سوف نستخدمها في مثالنا التالي تعمل بنظام تشغيل لينكس، إلا أنها تعمل دون مشاكل تذكر على أجهزة مضيفة تعمل بنظام تشغيل ويندوز.

5.7 تطبيق عملي يقوم بتحديد جنس الشخص من صورة وجهه

بداية نقوم بتثبيت TensorFlow على دوكر ضمن حاوية تحت تسمية hsoub-ft، قد تتطلب هذه الخطوة بعض الوقت نظرا لكون حجم صورة الحاوية التي سيتم سحبها وتنزيلها عبر الإنترنت يتجاوز 1GB:

```
docker pull tensorflow/tensorflow
docker run --name hsoub-tf -it -d tensorflow/tensorflow:latest
```

بعد ذلك نقوم بالدخول إلى سطر الأوامر ضمن الحاوية ونقوم بتثبيت الإصدار 2.0 على الأقل من مكتبة TensorFlow وكذلك الإصدار 0.6 على الأقل من نموذج تصنيف الصور3 الذي سنستخدمه والمستضاف في مكنز TensorFlow Hub، و بعد إتمام هذه الخطوات نخرج باستخدام الأمر exit في سطر الأوامر للعودة إلى الجهاز المضيف:

```
docker exec -it hsoub-tf bash
pip install "tensorflow~=2.0"
pip install "tensorflow-hub[make_image_classifier]~=0.6"
exit
```

الخطوة التالية هي تحضير بيانات التدريب وذلك من خلال الحصول على الصور التي سيتم تدريب الشبكة عليها، في مثالنا هذا استخدمنا مجموعة جزئية تتكون من حوالي 2000 صورة مقسمة إلى فئتين ذكور وإناث وهي مقتطعة من مجموعة بيانات أكبر تدعى UTKFace4 المتاحة للاستخدامات غير التجارية والتي تتضمن بالأساس ما يزيد عن 20 ألف صورة وجه. تم اختيار الصور التي سوف نستخدمها في عملية التدريب بحيث تكون متوازنة من حيث الجنس (أي أن عدد الصور الخاصة بالذكور يساوي تقريبا عددها للإناث) والعرقيات (أي

تقارب عدد الأشخاص من ذوي الأصول الأوروبية والأفريقية والهندية ...الخ.) وذلك لضمان عدم التحيز، إضافة إلى أن الصور تخص أشخاصا تتراوح أعمارهم ما بين 20 إلى 40 سنة.

للمتابعة عليك تحميل الملف المضغوط المرفق مع هذا المحتوى، ثم قم بفك ضغطه على حاسوبك وستحصل على مجلد باسم training داخله ثلاث مجلدات هي images و output وكذلك test. ستلاحظ ضمن مجلد images أن هنالك مجلد فرعي لك تصنيف تريد من شبكتك العصبية أن تتعرف عليه (في حالتنا هذه هناك مجلدان فقط بتسمية Male و Female) داخل كل منهما مجموعة الصور التي تنتمي إلى ذلك التصنيف. من جهة أخرى فإن مجلد output هو فارغ حاليا لكنه المكان الذي سيتم فيه حفظ الشبكة العصبية بعد إتمام عملية تدريبها (أي حيث سنخزن النموذج الناتج)، أخيرا ستجد في المجلد الثالث test شيفرة برمجية بسيطة لاختبار النموذج الناتج وبعض الصور التي لم يسبق له أن رآها من قبل (أي أنها لم تكن موجودة أصلا ضمن صور وبيانات التدريب).

نحن بحاجة إلى نقل كل هذه المجلدات وما فيها من ملفات إلى داخل الحاوية hsoub-tf وذلك باستخدام الأمر التالي على الجهاز المضيف:

```
docker cp ./training hsoub-tf:/training
```

الآن نستطيع الانتقال مجددا إلى سطر الأوامر ضمن الحاوية باستخدام الأمر التالي:

```
docker exec -it hsoub-tf bash
```

وبعد ذلك يمكننا بدء عملية التدريب باستخدام الأمر التالي:

```
make_image_classifier \
 --image_dir training/images dir \
 --tfhub_module
https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4 \
 --image_size 224 \
 --saved_model_dir training/output/model \
 --labels_output_file training/output/class_labels.txt \
 --tflite_output_file training/output/mobile_model.tflite
```

إن وسطاء الأمر التنفيذي السابق تحدد المسار الذي يحتوي على مجلدات الصور التي سيتم استخدامها في عملية التدريب (أي مسار المجلد ضمن الحاوية وليس على الجهاز المضيف)، وكذلك مصدر النموذج الأصلي الذي ستطبق عليه تقنية نقل التعلم باستبدال طبقته الأخيرة، بعد ذلك نحدد الأبعاد المطلوبة للصور ليتم تحجيمها تلقائيا لتلائم هذه الأبعاد (فهذا محدد ومرتبط بطبقة الإدخال في نموذج الشبكة العصبية المستخدمة)، في حالتنا نحن نستخدم نموذج MobileNet V2 والذي يفترض أبعاد الصورة هي 224 عنصورة/بيكسل للطول والعرض، بعد ذلك نحدد المكان الذي سيتم فيه حفظ النموذج المدرّب الناتج، وكذلك أين سيتم حفظ

التسميات المرافقة لعصبونات الخرج من شبكتنا العصبية بالترتيب، وأخيرا أين سيتم تصدير هذا النموذج بصيغة tflite المحزومة والموضبة لتطبيقات الجوال.

في نهاية عملية التدريب (والتي قد تطول تبعا لعدد الصور من جهة، ولسرعة الحاسوب الذي تتم عليه عملية التدريب من جهة أخرى)، سنجد أننا حصلنا على نموذج شبكة عصبية يستطيع تمييز جنس الشخص من صورته بدقة تتعدى 80% وذلك بمجهود بسيط جدا دون الحاجة إلى عمليات ضبط ومعايرة فائقة التخصص لوسطاء بناء وتدريب الشبكة العصبية، ودون الحاجة كذلك إلى ساعات وساعات من التدريب ولا إلى آلاف وآلاف من الصورة للتدرب عليها! إنها بحق نتيجة مرضية لأوائل مغامراتك مع الشبكات العصبية، لكنها مجرد البداية.

ها قد حان الوقت لتجربة النموذج الذي قمنا بتدريبه على صور لم يسبق له أن رآها من قبل، للقيام بذلك سوف نستخدم برنامج معد مسبقا لهذه الغاية5 وهو مكتوب بلغة بايثون Python وموجود داخل المجلد training/test صحبة بضعة صور كأمثلة للتجريب (ويمكن أن تستخدم صورك الخاصة في هذه المرحلة)، حيث ستتم عملية الاختبار لكل صورة من خلال الأمر التالي (وفيه نشير إلى مسار الشبكة العصبية التي سيتم تحميلها واستخدامها، والملف النصي الذي يتضمن اسم التصنيف الخاص بكل عصبون خرج ضمن هذه الشبكة بالترتيب، وأخيرا مسار الصورة المراد تصنيفها):

```
python training/test/label_image.py \
 --input_mean 0 --input_std 255 \
 --model_file training/output/mobile_model.tflite \
 --label_file training/output/class_labels.txt \
 --image training/test/Ahmed_Zewail.jpg
```

بعد إتمام تدريبنا واختبارنا للشبكة العصبية حان الوقت للخروج من بيئة التدريب هذه باستخدام الأمر `exit` للعودة إلى سطر الأوامر على الجهاز المضيف، ومن ثم سنرغب طبعا باستخراج النموذج الذي قمنا بتعليمه خلال هذه الجلسة من داخل حاوية دوكر وهو ما نستطيع القيام به باستخدام الأمر التالي:

```
docker cp hsoub-tf:/training/output ./training
```

5.8 نقاط تستحق التأمل

قد يرى البعض في تقنيات الذكاء الصنعي حلا لجميع المشكلات، لكنها في حقيقة الأمر لا تأتي خالية من المخاطر التي قد ترقى إلى حد اعتبارها نقاط ضعف أو عيوب يجب أن تؤخذ بعين الاعتبار، نذكر منها:

- باعتبار أن النماذج في الشبكات العصبية تبنى انطلاقا من بيانات التدريب، لذا فإنها غير قادرة على حل أي مشاكل كانت موجودة أصلا في تلك البيانات كالأخطاء أو الانحياز أو عدم التكامل والشمول لمختلف الحالات المراد التعامل معها.

- يصعب في الشبكات العصبية تفسير المنطق الذي استخدمه النموذج لإعطاء إجابته، فالتعامل معه أقرب ما يكون للصندوق الأسود الذي لا نعلم كمستخدمين آلية عمله الدقيقة في الداخل، بل نؤمن بأن ما نحصل عليه من إجابات هو أفضل المتاح، ولهذه الفكرة تداعياتها الفلسفية والاجتماعية التي تصعّب من استخدام هكذا أدوات في الحالات التي تتطلب تقديم تبرير (مثلا عند تطبيق عقوبة أو حرمان من مكافأة).
- تطوير نماذج الشبكات العصبية يتطلب قدرا كبيرا من بيانات التدريب المشروحة والمجهزة بشكل جيد وملائم، وهو ما قد يعني الكثير من الجهد لتوفير مثل هكذا مصادر للتدريب.
- عملية تدريب الشبكات العصبية هي عملية متطلّبة من ناحية قدرات المعالجة الحسابية المتوفّرة للعتاد المستخدم، وذلك على عكس عملية استخدامها في التطبيقات عقب إتمام التدريب، لذا قد يجد المطور في الحوسبة السحابية فرصا متاحة بميزانيات في المتناول.

إن بناء النماذج باستخدام خوارزميات تعلم الآلة يتطلب مطورين أذكياء وموهوبين، فعلى الرغم من أن تطبيق المثال الذي قمنا بعرضه يبدو سهلا ويسيرا، إلا أن نظرة أكثر تعمقا ستكشف لك الكثير من التفاصيل التي بحاجة إلى معايرة وضبط، منها على سبيل المثال لا الحصر: معمارية الشبكة، وطريقة توصيف تابع تفعيل العصبونات فيها، وعدد طبقاتها الخفية، وعدد العصبونات في كل منها، وطريقة ربطها بعضها ببعض، وكيفية حساب الخطأ ما بين النتيجة المحسوبة والمطلوبة، ومعدل سرعة التعلم، وعدد مرات تكرار عرض الأمثلة على الشبكة العصبية أثناء التدريب، وغيرها الكثير. إن انتقاء التوليفة الأكثر ملائمة لجملة معايير الضبط هذه يؤثر بشكل حاسم على جودة ودقة النموذج الناتج.

5.9 مراجع للاستزادة

1. MobileNetV2: The Next Generation of On-Device Computer Vision Networks
2. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition
3. Making your own TensorFlow model for image classification
4. UTKFaceLarge Scale Face Dataset
5. TensorFlow Lite Python image classification demo

6. تدريب شبكة عصبية صنعية للتعرف على الوجوه

نتطرق في هذا الفصل إلى الشبكات العصبية الصنعية ونغوص في بنيتها واستخدامها ثم سنبني واحدة وندربها لنستخدمها في التعرف على الوجوه.

6.1 متطلبات المشروع

سنبني هذا المشروع فقط باستخدام لغة جافاسكربت JavaScript مع نموذج صفحة HTML بسيط، لذا يفضل أن تكون لك خبرة أساسية بهذه اللغة لفهم الشيفرة، بالإضافة إلى إعطاء صلاحيات الوصول إلى الكاميرا سواءً لحاسوبك أو جوالك لالتقاط الصور ومعالجتها للتعرف عليها أي تطبيق النموذج المبني عليها.

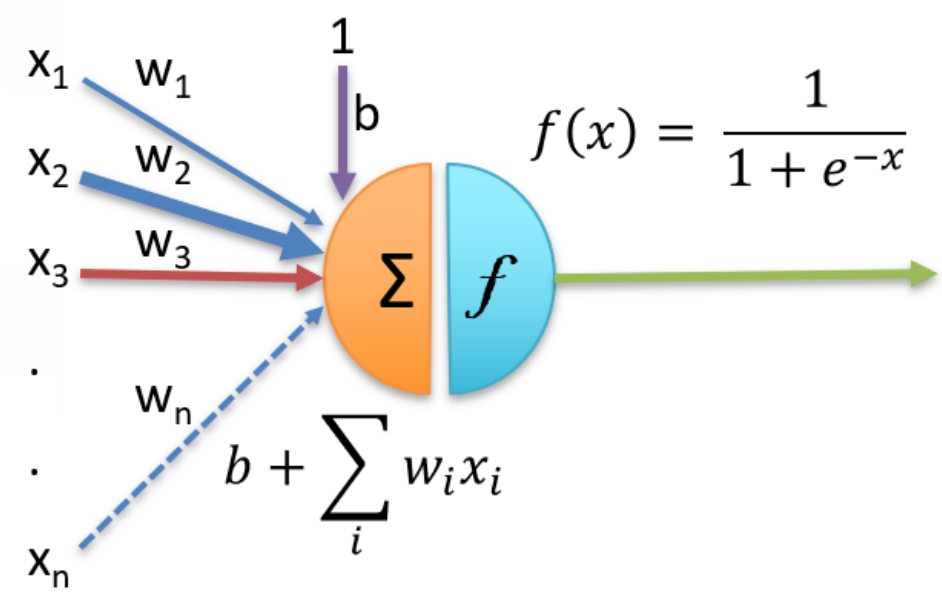
6.2 بنية الشبكات العصبية الصنعية

تتكون الشبكات العصبية الصنعية من عناصر أولية بسيطة من حيث طريقتها في معالجة البيانات تدعى العصبونات، ويمكن تمثيل أي من تلك العصبونات رياضيًا -وفق مفاهيم البرمجة كائنية التوجه- على شكل كائن برمجي Object يتضمن كل نسخة Instance منه على مصفوفة من أوزان الدخل يساوي حجمها عدد إشارات الدخل الواردة لهذا العصبون والتي قد تتراوح ما بين العشرات والآلاف تبعا لبنية الشبكة العصبية وهيكليتها.

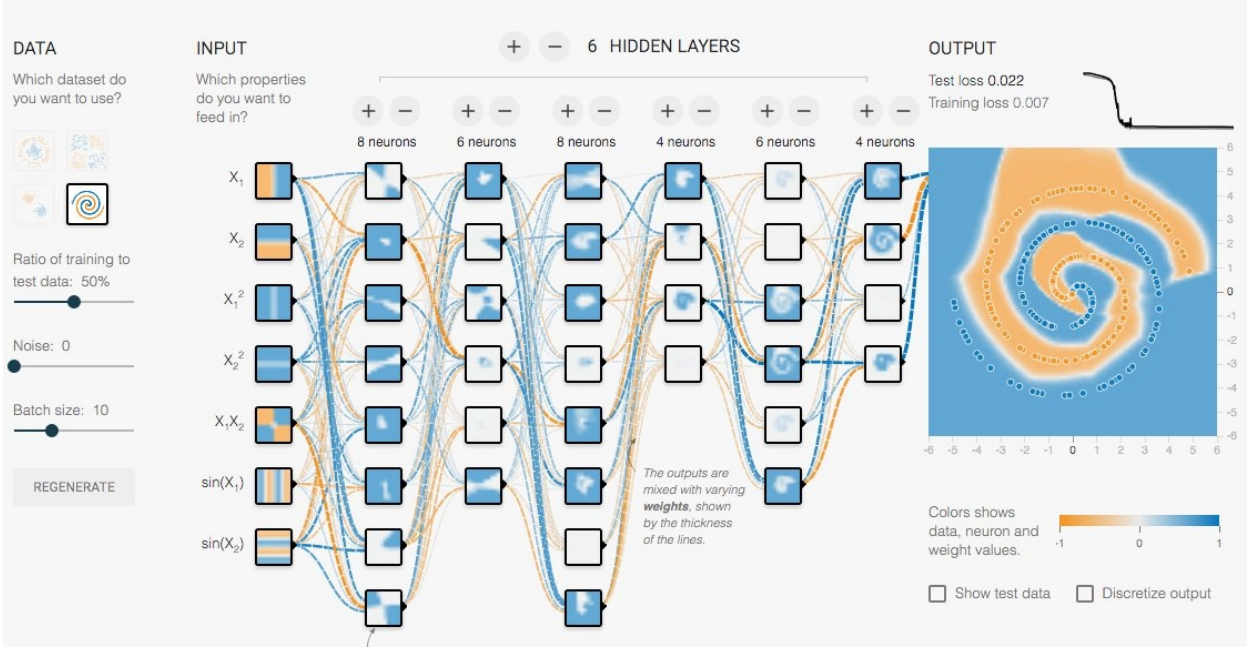
إن معالجة العصبون الواحد للإشارات الواردة بسيطة من الناحية الرياضية، حيث يتم ضرب كل إشارة دخل بقيمة الوزن المقابل لها ومن ثم تجمع كافة النواتج لنحصل على مقدار كلي يمثل درجة استثارة العصبون أو شحنته الإجمالية، ومن ثم يرسل هذا الرقم إلى تابع التفعيل، وهو ببساطة علاقة رياضية تحاكي إلى حد ما سلوك العصبونات الحقيقية فيما يخص قانون الكل أو لاشيء مع بعض التصرف لاعتبارات تتعلق بخوارزميات التعليم المستخدمة (إمكانية حساب المشتق الرياضي كما في حالة تابع Sigmoid) أو لاعتبارات براغماتية عملية

(سهولة البرمجة وسرعة التنفيذ كما في حالة التابع ReLU) أو حتى لاعتبارات تتعلق بطبيعة البيانات المعالجة (كشيوع استخدام التابع TanH عند التعامل مع بيانات الفئات والأصناف).

في نهاية المطاف تستخدم قيمة الاستثارة الإجمالية للعصبون كدخل لتابع التفعيل x هذا، وبتطبيق العلاقة الرياضية الخاصة بتابع التفعيل المحدد للعصبون نحسب قيمة y المقابلة والتي تمثل الآن خرج هذا العصبون.



تتألف الشبكات العصبية من بضع عشرات أو مئات أو حتى الآلاف من تلك الوحدات البنائية الأساسية والتي تنتظم في معمارية على شكل طبقات، حيث يحدد لكل طبقة عدد العصبونات التي تتضمنها، ونوع تابع التفعيل المستخدم لعصبونات تلك الطبقة (أي أن تابع التفعيل قد يختلف من طبقة إلى أخرى لكنه عادة ما يكون ذاته لجميع عصبونات الطبقة الواحدة)، وطريقة ارتباطها بالطبقة التي تسبقها والتي تليها (هل هو ربط كامل كأن تصل إشارة دخل واردة من كل عصبون موجود في الطبقة السابقة، أم هو جزئي يأخذ دخله من عصبونات الطبقة السابقة المجاورة لموضعه الحيزي كما هو حال الشبكات العصبية الصنعية التي تحاكي الباحات البصرية في الدماغ بغية التعرف على الأنماط في الصور).



6.3 الفرق ما بين تدريب واستخدام الشبكات العصبية الصنعية

إن تدريب الشبكات العصبية الصنعية يقصد به إيجاد القيم الأنسب لكافة أوزان الدخل لعصبونات الشبكة بحيث تكون قادرة على إعطاء الإجابات الصحيحة بأقل هامش خطأ في مخرجاتها وذلك للمدخلات المعروضة عليها، في حين يقع على عاتق المطور فن تحديد المعمارية الملائمة من حيث عدد الطبقات ونوع تابع تفعيل كل منها وعدد عصبوناته وطريقة ربطها مع بعضها البعض.

قد تكون مهمة تدريب الشبكات العصبية الصنعية تحديا حسابيا هائلا تتطلب كما كبيرا من المعالجات الرياضية والتي قد تستغرق زمنا طويلا حتى لو أجريت على حواسيب فائقة، لكن بمجرد الوصول إلى الحل وإيجاد قيم الأوزان الملائمة التي تعطي نتائج مرضية عند معالجتها للبيانات، تصبح مسألة الاستخدام والتطبيق في منتهى البساطة، فكل ما عليك القيام به هو تمرير وحيد عبر طبقات الشبكة يتم خلاله إجراء بضع عمليات ضرب وجمع ومن ثم تطبيق تابع رياضي ما (تابع التفعيل)، وهكذا تمرر القيم من طبقة إلى أخرى وصولا إلى طبقة الخرج النهائية لتظهر من خلالها الإجابة.

6.4 كيفية حفظ وتصدير الشبكات العصبية الصنعية

بعد إتمام عملية تدريب الشبكة العصبية الصنعية والرضى عن نتائجها، كل ما نحتاج إليه هو حفظ بنيتها من حيث عدد الطبقات، وعدد العصبونات في كل طبقة، وطريقة ارتباط عصبونات الطبقات المختلفة بعضها ببعض، وكذلك تابع التفعيل المستخدم في كل طبقة، إضافة إلى قيم الأوزان لكل عصبون على حدة. هذا كل مافي الأمر! فإن حصلت على هذه الأرقام مجددا فأنت قادر على إعادة بناء واستخدام تلك الشبكة العصبية الصنعية بشكلها النهائي بعد أن اكتملت عملية تدريبها.

عادة ما تقوم مكتبة TensorFlow بحفظ نموذج الشبكة العصبية الصنعية بكافة وسطائها وقيمها ضمن مجلد خاص بذلك خلال عملية التدريب والذي يمكن قراءته واستعادة بنية شبكته العصبية باستخدام لغة بايثون Python (ستجد ضمنه ملف بامتداد pb.* إضافة إلى مجلدين فرعيين هما variables وassets)، لكن هذه الصيغة من الحفظ ملائمة فقط لبيئة التطوير ولا تصلح للتطبيق العملي. عوضا عن ذلك أتاحت Google مجموعة من الصيغ المعيارية المناسبة للتطبيقات المختلفة من أهمها صيغة tfjs.* الموجهة للمبرمجين بلغة JavaScript ذائعة الصيت وواسعة الانتشار في عالم الويب سواء باستخدامها على طرف المستعرض أو حتى على طرف المخدم من خلال تقنية Node.js، كذلك لدينا صيغة tflite.* والمصممة أصلا لتعمل بموارد محدودة من سرعة معالجة ومساحة ذاكرة وذلك بهدف استخدامها على أجهزة الجوال أو إنترنت الأشياء.

لا يقتصر الأمر على ذلك فحسب، لكن حزم الشبكات العصبية الصنعية وإعدادها للاستخدام العملي قد يتضمن بعضا من المقايضة بهدف تصغير حجمها وزيادة سرعة عملها ولو كان ذلك على حساب انخفاض طفيف ومقبول في دقة نتائجها، إحدى تلك الخوارزميات على سبيل المثال تقوم بإزالة الروابط التي تقترب قيم أوزانها من الصفر باعتبارها لا تساهم بشكل كبير في مقدار الاستثارة الكلية للعصبون المرتبطة به، في المقابل نكون قد خفضنا عدد الأوزان التي نحن بحاجة إلى حفظها وكذلك قللنا الزمن الكلي اللازم للحساب كون عدد المدخلات أصبح أقل.

6.5 مكتبة Face-API

كما هو واضح من المقدمة السابقة، فنحن لن نقوم هنا ببناء وتدريب شبكة عصبية صنعية من الصفر، لكننا في المقابل سوف نستعرض كيفية استخدام شبكة عصبية صنعية سبق وأن تم تدريبها وذلك ضمن تطبيقنا الخاص. لهذه الغاية أود تعريفكم بمكتبة face-api.js والتي تم بناؤها لأغراض التعرف على الوجوه وتمييزها ضمن بيئة المتصفح انطلاقًا من مكتبة TensorFlow.js، وهي بذلك تقدم مجموعة من نماذج الشبكات العصبية الصنعية المدربة والجاهزة لطيف متنوع من التطبيقات يمكنك الإطلاع عليها من خلال النقر على الرابط التالي لصفحة التوثيق الخاص بهذه المكتبة:

https://justadudewhohacks.github.io/face-api.js/docs

المثال الذي سوف نبنيه فيما يلي سيستخدم كاميرا حاسوبك أو جوالك من خلال صفحة ويب ستبرمجها بنفسك لكي يأخذ صورة المستخدم ويقدّر منها العمر والجنس، فتخيل لو أننا نتحدث عن صفحة تسجيل مستخدم جديد في تطبيقك المستقبلي، سيكون من اللطيف والذكي في آن معا لو استطاع التطبيق تحديد القيم الافتراضية للعمر والجنس بناء على صورة المستخدم بطريقة تحسن من تجربة المستخدم.

بني نموذج التعرف على العمر والجنس باستخدام شبكة عصبية صنعية متعددة المهام توظف فيها طبقة استخلاص سمات الوجه لتقدير العمر وكذلك لتصنيف الجنس، يبلغ حجم حزمة النموذج بالإجمال قرابة 420kb، وقد تم تدريبها واختبارها باستخدام قواعد بيانات متعددة لصور الوجوه الموسومة بالعمر والجنس الحقيقيين

(منها على سبيل المثال موسوعة Wikipedia وقاعدة IMDB للأفلام والممثلين)، وتبلغ دقتها 95% في تحديد الجنس، وهامش خطئها في تحديد العمر يقارب 4 سنوات ونصف.

6.6 استخدام الشبكات العصبية في نافذة متصفحك

لنقم بداية بإنشاء مجلد جديد وليكن اسمه face-api للمشروع الذي سنعمل عليه اليوم ضمن مخدم الويب المحلي الذي لديك لتجربة تطبيقات الويب التي تعمل عليها (أي ضمن المجلد C:\xampp\htdocs على سبيل المثال في حال كنت تستخدم مخدم XAMPP)، سنضيف داخل ذلك المجلد الرئيسي مجلدين فرعيين باسم js وكذلك models ومن ثم نقوم بنسخ الملفات التي نحتاج إليها من مكتبة Face-API الأصلية على الشكل التالي:

أولًا، نُنزِّل الصيغة المصغرة/المضغوطة من المكتبة ذاتها (أي الملف "face-api.min.js") ونسخها إلى داخل المجلد الفرعي js في مشروعنا، يمكن الحصول على ذلك الملف من العنوان التالي:

https://github.com/justadudewhohacks/face-api.js/tree/master/dist

ثانيًا، نُنَزِّل نماذج الشبكات العصبية الصنعية التي سنستخدمها، حيث يوجد ملفين لكل نموذج شبكة عصبية أحدهما ينتهي اسمه بكلمة "shard1" تحفظ فيه قيم أوزان الشبكة، أما الآخر فهو ملف بصيغة json يصف بنية الشبكة حتى يتم قراءة وتحميل قيم الأوزان السابقة بشكل سليم. في مثالنا الحالي سنحتاج إلى استخدام شبكتين عصبيتين هما "age_gender_model" و "tiny_face_detector_model"، أي أننا بحاجة إلى تنزيل أربع ملفات ومن ثم نسخها إلى داخل مجلد models في مشروعنا. يمكنك الحصول على تلك الملفات من العنوان التالي:

https://github.com/justadudewhohacks/face-api.js/tree/master/weights

بعد ذلك سنُنشِئ ملفين جديدين أحدهما سندعوه index.html سنضعه في المجلد الرئيسي للمشروع، أما الآخر فسندعوه main.js ونضعه ضمن المجلد الفرعي js المخصص لشيفرات لغة جافاسكربت JavaScript البرمجية. سنقوم بداية بتنقيح محتوى صفحة HTML والتي سنجعلها بأبسط شكل ممكن بغرض التركيز على العناصر الأساسية التي نهتم بها:

```
<html>
  <head>
    <title>Face-API Example</title>
  </head>
  <body>

  </body>
</html>
```

ضمن قسم body سنقوم بإضافة عنصر div فيه عنصر video لعرض الفيديو الذي تلتقطه كاميرا حاسوبك ندعوه webcam، ويتم ضبطها لتعمل تلقائيا ضمن النمط الصامت بمجرد تحميل الصفحة:

```
<div class="container">
    <video id="webcam" height="360" width="360" autoplay
muted></video>
</div>
```

ثم نضيف حقل إدخال input مخصص للعمر ندعوه age:

```
Age (years): <input id="age" size="3"><br />
```

بعد ذلك نضيف مجموعة أزرار اختيار من نوع radio ندعوها gender فيها اختيارين اثنين هما ذكر male وأنثى female:

```
<input id="male" type="radio" name="gender" value="male">Male<br />
<input id="female" type="radio" name="gender" value="female">Female<br
/>
```

أخيرا وبعد اكتمال تحميل وعرض كافة عناصر الصفحة، نقوم بتضمين ملفات جافاسكريبت في نهاية محتويات الصفحة حتى نكون متأكدين من وصولها برمجيا لكافة العناصر ضمن تلك الصفحة:

```
<script src="js/face-api.min.js"></script>
<script src="js/main.js"></script>
```

> بسبب إعدادات الأمان فإن معظم متصفحات الويب تعرض رسالة للمستخدم تبيّن رغبة الصفحة في الوصول إلى الكاميرا وتطلب الإذن للسماح لها بذلك.

هذا كل ما هو مطلوب فعليا ضمن صفحة HTML الخاصة بهذا المثال، أما الآن فعلينا الانتقال إلى لكتابة شيفرات لغة جافاسكريبت التي ستقوم بتحميل نموذج الشبكة العصبية الصنعية، وتمرير صور الكاميرا إليه، ومن ثم قراءة الخرج الناتج عن الشبكة العصبية الصنعية وعرضه ضمن عناصر الإدخال المقابلة في صفحة HTML السابقة.

لذلك سنبدأ بكتابة محتوى الملف main.js من الشيفرات البرمجية مع شرح مختصر لكل قسم منها، فالسطر الأول يقوم بتعيين اسم الثابت video للإشارة إلى العنصر المدعوة webcam ضمن صفحة HTML السابقة:

```
const video = document.getElementById("webcam");
```

بعد ذلك نقوم بتحميل نموذجي الشبكتين العصبيتين الخاصتين بالتعرف على الوجوه ضمن الصورة وتحديد الجنس وتقدير العمر من خلال سمات تلك الوجوه وذلك باستخدام آلية الوعود Promise.all، ومن بعد إتمام ذلك يتم استدعاء التابع `startVideo` الذي سنقوم بتعريفه لاحقا:

```
Promise.all([
 faceapi.nets.tinyFaceDetector.loadFromUri("models"),
 faceapi.nets.ageGenderNet.loadFromUri("models")
]).then(startVideo);
```

يقوم هذا التابع بقراءة دفق إشارة الفيديو من الكاميرا وعرضها ضمن الكائن `video`، فإن حدث خطأ يتم كتابة رسالة الخطأ ضمن `console.error` في المستعرض:

```
function startVideo() {
 navigator.getUserMedia(
 { video: {} },
 stream => (video.srcObject = stream),
 err => console.error(err)
 );
}
```

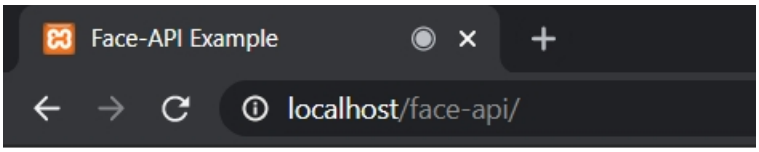
ومن ثم نضيف حدث يقوم بالإنصات لما يتم عرضه ضمن عنصر فيديو `video` بحيث يستدعي بشكل غير متزامن `async` (أي دون الحاجة لانتظار الإجابة حتى يتابع عرض الفيديو) وذلك كل ثانية (حيث تم ضبط الفترة الزمنية `setInterval` لتكون قيمتها 1000 ميلي ثانية). يقوم هذا الاستدعاء الدوري المتكرر بتعيين قيمة الكائن detection بواسطة مكتبة faceapi حيث تحدد جميع الوجوه الموجودة ضمن كائن video بالاستعانة بنموذج TinyFaceDetector مستخدمين الإعدادات الافتراضية، ومن ثم يمرر الناتج (أي الوجوه التي تم التعرف عليها في الصورة)، إلى الشبكة العصبية الأخرى التي تتعرف على العمر والجنس:

```
video.addEventListener("playing", () => {
 setInterval(async () => {
 const detections = await faceapi
 .detectAllFaces(video, new faceapi.TinyFaceDetectorOptions())
 .withAgeAndGender();
```

إن العبارة الشرطية التالية تتأكد إن كان الكائن `detections` موجودا أصلا (تذكر أن تعيين قيمة هذا الكائن تتم من خلال استدعاء غير متزامن وبالتالي يمكن أن تكون القيمة غير موجودة في البداية ريثما تصل إجابة ذلك الاستدعاء). ليس هذا فحسب، بل يجب أن يتضمن على الأقل وجه واحد على الأقل. فإن تحقق هذين الشرطين يتم قراءة الجنس المتوقع للوجه الأول `detections[0].gender` وتعيينه لعنصر الإدخال المقابل له، وكذلك قراءة العمر المتوقع لذات الوجه الأول `detections[0].age` وتقريبه إلى أقرب عدد صحيح باستخدام التابع `Math.round` ومن ثم إسناده كقيمة لعنصر الإدخال المدعو age ضمن صفحة HTML السابقة:

```
  if (detections && Object.keys(detections).length > 0) {
    document.getElementById(detections[0].gender).checked = true;
    document.getElementById("age").value =
Math.round(detections[0].age);
  }
  }, 1000);
  });
```

هذا كل شيء! تستطيع الآن تجربة هذا التطبيق البسيط من خلال عـرض الصـفحة على نافـذة المسـتعرض مستخدمين عنوانها على مخدم الويب المحلي وليكن على سبيل المثال: http://localhost/face-api:



Age (years): 44
Male
Female

6.7 نقاط تستحق التأمل

من منا لا تغريه كم الإسقاطات التي تطرحها تقنية الشبكات العصبية الصنعية على مفهوم البرمجة ككل؟ إنها بذور جيل جديد مختلف جذريا عن كل ما سبق وأن اعتدنا عليه، فرغم أنها تعتمد على وحدات بسيطة في صميمها هي العصبونات، والتي قمنا بوصف آلية عملها بطريقة خوارزمية محددة تماما وواضحة، إلا أننا مع نمو شبكاتنا في الحجم والتعقيد استبدلنا الثقة بالأمل، والتحليل بالتجريب.

لنأخذ على سبيل المثال نموذج اللغة العربية، فحتى نستطيع تشكيل أي جملة نستخدم علم النحو والإعراب الذي تعلمناه في مدارسنا، ومن خلال قواعده نحدد بناء الجملة وعناصرها كالفعل والفاعل أو المبتدأ والخبر وسواها من القواعد التي تساعدنا على ضبط أواخر الكلمات. هذه هي تماما الطريقة التي نبرمج بها حواسيبنا حاليا (منذ أيام تشارلز بابيج وحاسوبه الميكانيكي قبل مئتي عام)، فنحن نضع القواعد الصارمة لتدفق البيانات ونترجمها إلى عبارات شرطية وحلقات وسواها من بنى التحكم في عدد منتهي من الخطوات ندعوها بالخوارزمية.

لكن لحظة، هل هذه هي الطريقة الصحيحة فعلا التي تجري بها الأمور؟ هل هذا ما نفعله حقيقة عندما يلتبس علينا تشكيل كلمة ما في نص نقرأه أو جملة نتحدث بها؟ أغلب الظن ستكون إجابتك هي النفي! فمعظمنا لا يعرب الجملة في رأسه ليضبط أواخر الكلمات، بل نجرب الاحتمالات المختلفة ونرى أيها يملك الوقع الأكثر تجانسا وانسجاما في نفسنا (على سبيل المثال: هل أباك/أبوك/أبيك موجود؟). إنها ببساطة ما ندعوه بالسليقة اللغوية، وهي التي تفسر قدرة الطفل الصغير على التكلم بلغة صحيحة حتى قبل معرفته بوجود علم النحو والإعراب أصلا، فهو يتعلم مما يسمع وينسج على منواله، فكلما كانت الأمثلة كثيرة وصحيحة ازداد إتقانه للغة (ولهذا السبب كان العرب يرسلون أبنائهم إلى البادية لسلامة اللغة من اللحن الأعجمي هناك).

إن التعلم من خلال الأمثلة هو ما تقوم به تماما الشبكات العصبية الصنعية التي نقوم بتطويرها، إنها تقوم ببناء خبرتها وسليقتها وفراستها الخاصة، في نهاية المطاف يمتلك كل من المبتدئ والخبير القواعد ذاتها، لكن عين الخبير رأت الكثير من الأمثلة التي صقلت بها خبرتها، حتى وصلت من التعقيد إلى درجة قد يصعب معها التفسير رغم قوة الإحساس ووضوحه في ذهنه. وهكذا ينتقل دور المبرمج رويدا رويدا من وضع القواعد وبنى التحكم لبرمجياته إلى ما بات يدعى اليوم بعلم البيانات وهندسة عملية استخلاص المزايا والخصائص المؤثرة في عملية التعلم من تلك البيانات، حيث أن دقة وجودة النموذج الناتج تعتمد بشكل حاسم على تكامل أمثلة التعليم وشمولها وصحتها، وأصبح امتلاك مثل هكذا مكانز رقمية مُوصَّفة بشكل دقيق ومبنية بطريقة هيكلية سهلة الولوج والاستعلام يعادل قطع ما يتجاوز نصف الطريق إلى الوصول لعالم الذكاء الصنعي التطبيقي وقطف ثماره.

6.8 مراجع للاستزادة

- TensorFlow.js library
- Age and Gender Recognition Model
- TensorFlow Model Optimization

7. تحليل المشاعر في النصوص العربية

تُعدّ مسألة تحليل تعليقات وتغريدات الأشخاص على مواقع التواصل الاجتماعي من المسائل المهمة والتي لها الكثير من التطبيقات العملية. مثلًا: تهتم المتاجر الإلكترونية كثيرًا بتحليل تعليقات الزبائن على منتجاتهم لاستكشاف توجهات الزبائن ومواطن الضعف والقوة في المتجر.

نعرض في هذا الفصل استخدام تقنيات التعلّم العميق في تحليل المشاعر لنصوص مكتوبة باللغة العربية وباللهجة السعودية بمعنى أن اللغة المستخدمة ليست بالضرورة اللغة العربية الفصحى بل يُمكن أن تدخل فيها ألفاظ عامية يستخدمها المغردون عادةً.

7.1 بيانات التدريب

تحوي مجموعة البيانات المتوفرة (نزلها من هذا الرابط) حوالي 23500 تغريدات لتعليقات الأشخاص وملاحظاتهم حول مجموعة من الأماكن العامة في المملكة العربية السعودية. جُمّعت هذه التغريدات عن طريق مجموعة من الطلاب الجامعيين وذلك من مجموعة متنوعة من مواقع التواصل الاجتماعي.

نُعطي فيما يلي أمثلة عن هذه التغريدات (تعليقات حول حديقة حيوانات مثلًا):

"أنصحكم والله بزيارته مكان جميل جدا مرتب ونظيف"

"جميلة وكبيرة وتحتاج واحد عنده لياقه يمشي فيها"

"حيوانات قليلة جدا ولا يوجد اهتمام مكثف"

"كان يعيبها وقت الافتتاح والاغلاق وعدم وجود خريطه"

"أول مرة أزور حديقة حيوانات"

"أيام العوائل الخميس والجمعة والسبت"

يُمكننا، كبشر، تصنيف التغريدات السابقة وبشكل سريع إلى ثلاث فئات:

- التغريدات الموجبة (الأولى والثانية) أي التغريدات التي تحمل معاني إيجابية تُعّبر عن الرضا والارتياح.
- التغريدات السالبة (الثالثة والرابعة) أي التغريدات التي تحمل معاني سلبية تُعبّر عن الاستياء.
- التغريدات المحايدة (الخامسة والسادسة) أي التغريدات التي يُمكن أن تُعطي معلومات ولا تحمل أية مشاعر فيها سواء موجبة أم سالبة.

نتعلم في هذا المشروع بناء مُصنّف حاسوبي آلي يُصنّف أي جملة عربية إلى موجبة أو سالبة أو محايدة.

7.2 تصنيف بيانات التدريب

يتطلب استخدام خوارزميات تعلّم الآلة (خوارزميات تصنيف النصوص في حالتنا) توفر بيانات للتدريب أي مجموعة من النصوص مُصنفّة مُسبقًا إلى: موجبة، سالبة، محايدة.

يُمكن، في بعض الأحيان، اللجوء إلى الطرق اليدوية: أي الطلب من مجموعة من الأشخاص قراءة النصوص وتصنيفها. وهو حل يصلح في حال كان عدد النصوص صغيرًا نسبيًا. يتميز هذا الحل بالدقة العالية لأن الأشخاص تُدرك، بشكل عام، معاني النصوص من خلال خبرتها اللغوية المُكتسبة وتُصنّف النصوص بشكل صحيح غالبًا.

نستخدم، في حالتنا، حلًا إحصائيًا بسيطًا لتصنيف نصوص التدريب إلى موجبة، سالبة، محايدة وذلك باستخدام قاموس للكلمات الموجبة وقاموس آخر للكلمات السالبة.

يحوي قاموس الكلمات الموجبة على مجموعة من الكلمات الموجبة الشائعة مع نقاط لكل كلمة (1 موجبة، 2 موجبة جدًا، 3 موجبة كثيرًا). مثلًا:

- روعة، 3
- جيد، 2
- معقول، 1

يحوي قاموس الكلمات السالبة على مجموعة من الكلمات السالبة الشائعة مع نقاط لكل كلمة (1- سالبة، 2- سالبة جدًا، 3- سالبة كثيرًا). مثلًا:

- مقرف، 3-

- سيء، 2-
- زحمه، 1-

اختيرت كلمات القواميس الموجبة والسالبة من قبل مجموعة من الطلاب بعد أن طلبنا منهم استعراض التغريدات المُتاحة وانتقاء الكلمات التي تُعطي التغريدة معنى موجب أو معنى سالب، وإعطاء كل كلمة موجبة نقاط تدل على شدة الإيجابية لها (1،2،3) وكل كلمة سالبة نقاط تدل على شدة السلبية (3-،2-،1-).

نعدّ، فيما يلي، نصًا ما أنه موجبًا إذا كان مجموع نقاط الكلمات الموجبة الواردة ضمن النص أكبر من مجموع نقاط الكلمات السالبة الواردة ضمنه. وبالمقابل، نعدّ نصًا ما أنه سالبًا إذا كان مجموع نقاط الكلمات السالبة الواردة ضمن النص أكبر من مجموع نقاط الكلمات الموجبة الواردة ضمنه. يكون نصًا ما محايدًا إذا تساوى مجموع نقاط الكلمات الموجبة فيه مع مجموع نقاط الكلمات السالبة.

بالطبع، لا تُعدّ هذه الطريقة صحيحة دومًا إذ يُمكن أن تُخطئ في بعض الحالات إلا أنها على وجه العموم تُستخدم عوضًا عن الطريقة اليدوية.

7.3 المعالجة الأولية للنصوص

تهدف المعالجة الأولية إلى الحصول على الكلمات المهمة فقط من النصوص وذلك عن طريق تنفيذ بعض العمليات اللغوية عليها. تُنفذّ هذه العمليات على كل من التغريدات وكلمات القواميس.

لتكن لدينا مثلًا الجملة التالية:

"أنا أحب الذهاب إلى الحديقة، كل يوم 9 صباحاً، مع رفاقي هؤلاء!"

سنقوم بتنفيذ العمليات التالية:

أولًا، حذف إشارات الترقيم المختلفة كالفواصل وإشارات الاستفهام وغيرها، ويكون ناتج الجملة السابقة:

"أنا أحب الذهاب إلى الحديقة كل يوم 9 صباحاً مع رفاقي هؤلاء".

ثانيًا، حذف الأرقام الواردة في النص، فيكون ناتج الجملة السابقة:

"أنا أحب الذهاب إلى الحديقة كل يوم صباحاً مع رفاقي هؤلاء"

ثالثًا، حذف كلمات التوقف stop words وهي الكلمات التي تتكرر كثيرًا في النصوص ولا تؤثر في معانيها كأحرف الجر (من، إلى، ...) والضمائر (أنا، هو، ...) وغيرها، فيكون ناتج الجملة السابقة:

"أحب الذهاب الحديقة يوم صباحاً رفاقي"

رابعًا، تجذيع الكلمات stemming أي إرجاع الكلمات المتشابهة إلى كلمة واحدة (جذع أو جذر) مما يُساهم في إنقاص عدد الكلمات الكلية المختلفة في النصوص، ومطابقة الكلمات المتشابهة مع بعضها البعض. مثلًا: يكون للكلمات الأربع: (رائع، رايع، رائعون، رائعين) نفس الجذع المشترك: (رايع). فيكون ناتج الجملة السابقة:

"احب ذهاب حديق يوم صباح رفاق "

ننتبه إلى أن الجذع يختلف عن الجذر اللغوي إذ الجذر هو عملية لغوية لرد الكلمة إلى أصلها وجذرها الأساسي لأغراض مختلفة أشهرها البحث في القاموس أما الجذع فهو كلمة مشتركة بين مجموعة من الكلمات لا توجد بالضرورة في قاموس اللغة العربية وإنما إيجاد شكل موحد للكلمات.

7.4 إعداد المشروع

يُمكن تنزيل بيانات التدريب والقواميس والشيفرة البرمجية من الملف المرفق في نهاية الفصل.

يحتاج تنفذ شيفرات هذا الفصل بيئةً برمجيةً للغة بايثون الإصدار 3.8. ويجب أن تتضمن هذه البيئة البرمجية مدير الحِزم pip لتثبيت الحِزم، ومُنشئ البيئات الافتراضية venv لإنشاء بيئاتٍ افتراضيةٍ.

نستخدم محرر الشيفرات البرمجية Jupyter Notebooks، وهو مفيد جدًا لتجربة وتشغيل الأمثلة الخاصة بتَعَلّم الآلة بطريقةٍ تفاعليةٍ، حيث نستطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسرعة، مما يُسهّل علينا اختبار الشيفرات البرمجية وتصحيحها.

نحتاج أولًا لتثبيت بعض التبعيات، وذلك لإنشاء مساحة عملٍ للاحتفاظ بملفاتنا قبل أن نتمكن من تطوير برنامجنا.

نُنشئ مجلدًا جديدًا خاصًا بمشروعنا وندخل إليه هكذا:

```
mkdir sa
cd sa
```

نُنفذّ الأمر التالي لإنشاء البيئة الافتراضية:

```
python -m venv sa
```

ومن ثم الأمر التالي في لينكس Linux لتنشيط البيئة الافتراضية:

```
source sa/bin/activate
```

أما في ويندوز Windows، فيكون أمر التنشيط:

```
"sa/Scripts/activate.bat"
```

نستخدم إصداراتٍ محددةٍ من المكتبات اللازمة، من خلال إنشـاء ملـف `requirements.txt` في مجلـد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

نفتح الملف `requirements.txt` في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبـات التي نريدها وإصداراتها:

```
jupyter==1.0.0
keras==2.6.0
Keras-Preprocessing==1.1.2
matplotlib==3.5.1
nltk==3.6.5
numpy==1.19.5
pandas==1.3.5
scikit-learn==1.0.1
seaborn==0.11.2
sklearn==0.0
snowballstemmer==2.2.0
tensorflow==2.6.0
wordcloud==1.8.1
python-bidi==0.4.2
arabic-reshaper==2.1.3
```

نحفظ التغييرات التي طرأت على الملف ونخرج من محرر النصوص، ثم نُثَبت هذه المكتبات بالأمر التالي:

```
(sa) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، نُصبح جاهزين لبدء العمل على مشروعنا.

7.5 كتابة شيفرة برنامج تحليل المشاعر في النصوص العربي

نُشغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت. هكذا:

```
(sa) $ jupyter notebook
```

ثم نُنشئ ملفًا جديدًا في داخل المحرر ونُسمّه باسم asa مثلًا.

يجب أولًا وضع كل من الملفات التالية في مجلد المشروع:

- ملف التغريدات: tweets.csv
- القواميس: lexicon_positive.csv و lexicon_negative.csv

- ملف الخط العربي: DroidSansMono.ttf

7.5.1 تحميل البيانات

نبدأ أولًا بتحميل التغريدات من الملف `tweets.csv` ضمن إطار من البيانات `DataFrame` من مكتبة `Pandas` ومن ثم عرض بعضها:

```
import pandas as pd
# قراءة التغريدات وتحميلها ضمن إطار من البيانات
tweets_data = pd.read_csv('tweets.csv',encoding = "utf-8")
tweets = tweets_data[['tweet']]
# إظهار الجزء الأعلى من إطار البيانات
tweets.head()
```

يظهر لنا أوائل التغريدات:

	tweet
0	...احد خيارات التنزهة بمدينة الرياض حديقة الحيوان
1	...حديقة جميلة للاطفال وسعر دخول ممتاز واكشاك بس
2	...ممتاز جدا عندي بس بعض الملاحظات والاقتراحات\n\n
3	...الخدم\nحديقة منظمة وبها عديد كبير من الحيوانات
4	... فيه تجديدات حلوة ومكان جميل وسعر مناسب، نتمنى

نُحمّل قاموس الكلمات الموجبة positive.csv وقاموس الكلمات السالبة negative.csv:

```
# قراءة قاموس الكلمات الموجبة
positive_data = pd.read_csv('positive.csv' ,encoding = "utf-8")
positive = positive_data[['word', 'polarity']]
# قراءة قاموس الكلمات السالبة
negative_data = pd.read_csv('negative.csv' ,encoding = "utf-8")
negative = negative_data[['word', 'polarity']]
positive.head()
```

نُظهر مثلًا أوائل الكلمات الموجبة ونقاطها:

	word	polarity
0	ممتاز	3
1	رائع	3
2	مبهر	3
3	جميل	3
4	ساحر	3

7.5.2 المعالجة الأولية للنصوص

نستخدم فيما يلي بعض الخدمات التي توفرها المكتبة `nltk` لمعالجة اللغات الطبيعية كتوفير قائمة كلمات التوقف باللغة العربية (حوالي 700 كلمة) واستخراج الوحدات tokens من النصوص. كما نستخدم مُجذِّع الكلمات العربية من مكتبة `snowballstemmer`.

```
# مكتبة السلاسل النصية
import string
# مكتبة التعابير النظامية
import re
# مكتبة معالجة اللغات الطبيعية
import nltk
nltk.download('punkt')
nltk.download('stopwords')
# مكتبة كلمات التوقف
from nltk.corpus import stopwords
# مكتبة استخراج الوحدات
from nltk.tokenize import word_tokenize
# مكتبة المجذع العربي
from snowballstemmer import stemmer
ar_stemmer = stemmer("arabic")

# دالة حذف المحارف غير اللازمة
def remove_chars(text, del_chars):
    translator = str.maketrans('', '', del_chars)
    return text.translate(translator)
```

```
# دالة حذف المحارف المكررة
def remove_repeating_char(text):
    return re.sub(r'(.)\1{2,}', r'\1', text)

# دالة تنظيف النصوص
def cleaningText(text):
    # حذف الأرقام
    text = re.sub(r'[0-9]+', '', text)
    # حذف المحارف غير اللازمة
    # علامات الترقيم العربية
    arabic_punctuations = '''`÷×؛–“…”!|+¦~{}',.؟":/؛][%^&*()_<>؛'''
    # علامات الترقيم الانكليزية
    english_punctuations = string.punctuation
    # دمج علامات الترقيم العربية والانكليزية
    punctuations_list = arabic_punctuations + english_punctuations
    text = remove_chars(text, punctuations_list)
    # حذف المحارف المكررة
    text = remove_repeating_char(text)
    # استبدال الأسطر الجديدة بفراغات
    text = text.replace('\n', ' ')
    # حذف الفراغات الزائدة من اليمين واليسار
    text = text.strip(' ')
    return text

# دالة تقسيم النص إلى مجموعة من الوحدات
def tokenizingText(text):
    tokens_list = word_tokenize(text)
    return tokens_list

# دالة حذف كلمات التوقف
def filteringText(tokens_list):
    # قائمة كلمات التوقف العربية
    listStopwords = set(stopwords.words('arabic'))
    filtered = []
    for txt in tokens_list:
        if txt not in listStopwords:
```

```
            filtered.append(txt)
    tokens_list = filtered
    return tokens_list

# دالة التجذيع
def stemmingText(tokens_list):
    tokens_list = [ar_stemmer.stemWord(word) for word in tokens_list]
    return tokens_list

# دالة دمج قائمة من الكلمات في جملة
def toSentence(words_list):
    sentence = ' '.join(word for word in words_list)
    return sentence
```

شرح الدوال التي كتبناها في الشيفرة:

- `cleaningText`: تحذف الأرقام وعلامات الترقيم العربية والإنكليزية من النص.
- `remove_repeating_char`: تحذف المحارف المكررة والتي قد يستخدمها كاتب التغريدة.
- `tokenizingText`: تعمل على تجزئة النص إلى قائمة من الوحدات tokens.
- `filteringText`: تحذف كلمات التوقف من قائمة الوحدات.
- `stemmingText`: تعمل على تجذيع كلمات قائمة الوحدات المتبقية.

يُبين المثال التالي تجذيع بعض الكلمات المتشابهة:

```
# مثال
stem = ar_stemmer.stemWord(u"رابع")
print (stem)
stem = ar_stemmer.stemWord(u"رائع")
print (stem)
stem = ar_stemmer.stemWord(u"رائعون")
print (stem)
stem = ar_stemmer.stemWord(u"رائعين")
print (stem)
```

يكون ناتج التنفيذ:

```
رابع
رابع
رابع
رابع
```

يُبين المثال التالي نتيجة استدعاء كل دالة من الدوال السابقة:

```
# مثال
text= " أنا أحب الذهاب إلى الحديقة، كل يووووم 9 صباحاً، مع رفاقي هؤلاء!"
print(text)
text=cleaningText(text)
print(text)
tokens_list=tokenizingText(text)
print(tokens_list)
tokens_list=filteringText(tokens_list)
print(tokens_list)
tokens_list=stemmingText(tokens_list)
print(tokens_list)
```

يكون ناتج التنفيذ:

```
أنا أحب الذهاب إلى الحديقة، كل يووووم 9 صباحاً، مع رفاقي هؤلاء!

أنا أحب الذهاب إلى الحديقة كل يوم  صباحاً مع رفاقي هؤلاء

['أنا' ,'أحب' ,'الذهاب' ,'إلى' ,'الحديقة' ,'كل' ,'يوم' ,'صباحاً' ,'مع' ,'رفاقي' ,'
هؤلاء']

['أحب' ,'الذهاب' ,'الحديقة' ,'يوم' ,'صباحاً' ,'رفاقي']

['احب' ,'ذهاب' ,'حديق' ,'يوم' ,'صباح' ,'رفاق']
```

تعرض الشيفرة التالية تنفيذ جميع دوال المعالجة الأولية على نصوص التغريدات ومن ثم حفـظ النتـائج في ملف جديد `tweet_clean.csv`. وبنفس الطريقة، نُنفذ دوال المعالجة الأولية على قاموس الكلمـات الموجبـة وقاموس الكلمات السـالبة ونحفـظ النتـائج في ملفـات جديـدة لاسـتخدامها لاحقًـا: `positive_clean.csv` و `negative_clean.csv`.

```
# المعالجة الأولية للتغريدات
tweets['tweet_clean'] = tweets['tweet'].apply(cleaningText)
tweets['tweet_preprocessed'] =
tweets['tweet_clean'].apply(tokenizingText)
tweets['tweet_preprocessed'] =
tweets['tweet_preprocessed'].apply(filteringText)
tweets['tweet_preprocessed'] =
tweets['tweet_preprocessed'].apply(stemmingText)

# حذف التغريدات المكررة
tweets.drop_duplicates(subset = 'tweet_clean', inplace = True)
# التصدير إلى ملف
tweets.to_csv(r'tweet_clean.csv',encoding="utf-8", index = False,
header = True,index_label=None)

# معالجة القاموس الموجب
positive['word_clean'] = positive['word'].apply(cleaningText)
positive.drop(['word'], axis = 1, inplace = True)
positive['word_preprocessed'] =
positive['word_clean'].apply(tokenizingText)
positive['word_preprocessed'] =
positive['word_preprocessed'].apply(filteringText)
positive['word_preprocessed'] =
positive['word_preprocessed'].apply(stemmingText)

# حذف التكرار والخطأ
positive.drop_duplicates(subset = 'word_clean', inplace = True)
nan_value = float("NaN")
positive.replace("", nan_value, inplace=True)
positive.dropna(subset= ['word_clean'], inplace=True)

# التصدير إلى ملف
positive.to_csv(r'positive_clean.csv',encoding="utf-8", index = False,
header = True,index_label=None)

# معالجة القاموس السالب
negative['word_clean'] = negative['word'].apply(cleaningText)
negative.drop(['word'], axis = 1, inplace = True)
```

```
negative['word_preprocessed'] =
negative['word_clean'].apply(tokenizingText)
negative['word_preprocessed'] =
negative['word_preprocessed'].apply(filteringText)
negative['word_preprocessed'] =
negative['word_preprocessed'].apply(stemmingText)


# حذف التكرار والخطأ
negative.drop_duplicates(subset = 'word_clean', inplace = True)
negative.replace("", nan_value, inplace=True)
negative.dropna(subset= ['word_clean'], inplace=True)


# التصدير إلى ملف
negative.to_csv(r'negative_clean.csv', encoding="utf-8", index =
False, header = True,index_label=None)
```

تعرض الشيفرة التالية بناء قاموسين `dict` الأول للكلمات الموجبة والثاني للكلمات السالبة وبحيث يكون المفتاح key هو الكلمة والقيمة value هي نقاط الكلمة. يُعدّ استخدام بنية القاموس `dict` في بايثون مفيدًا جدًا للوصول المباشر إلى نقاط أي كلمة دون القيام بأي عملية بحث.

لاحظ أننا نقرأ الكلمات الموجبة والسالبة من الملفات الجديدة ناتج المعالجة الأولية لملفات الكلمات الأصلية.

```
# التصريح عن قاموس للكلمات الموجبة
dict_positive = dict()


# بناء قاموس الكلمات الموجبة
myfile = 'positive_clean.csv'
positive_data = pd.read_csv(myfile, encoding='utf-8')
positive = positive_data[['word_clean', 'polarity']]
for i in range(len(positive)):
    dict_positive[positive_data['word_clean'][i].strip()] =
int(positive_data['polarity'][i])


# التصريح عن قاموس للكلمات السالبة
dict_negative = dict()


# بناء قاموس الكلمات السالبة
```

```
myfile = 'negative_clean.csv'
negative_data = pd.read_csv(myfile, encoding='utf-8')
negative = negative_data[['word_clean', 'polarity']]
for i in range(len(negative)):
    dict_negative[negative_data['word_clean'][i].strip()] =
int(negative_data['polarity'][i])
```

تقوم الدالة التالية `sentiment_analysis_dict_arabic` بحساب مجموع نقاط `score` قائمة من الكلمات وذلك بجمع نقاط الكلمات الواردة في قاموسي الكلمات الموجبة والسالبة. وفي النهاية تُعدّ قطبية polarity قائمة الكلمات موجبة `positive` إذا كان مجموع نقاطها أكبر من الصفر، وتُعدّ سالبة `negative` إذا كان مجموع نقاطها أصغر من الصفر، وإلا فإنها تكون محايدة `neutral`.

```
# دالة حساب قطبية قائمة من الكلمات
def sentiment_analysis_dict_arabic(words_list):
    score = 0
    for word in words_list:
        if (word in dict_positive):
            score = score + dict_positive[word]
    for word in words_list:
        if (word in dict_negative):
            score = score + dict_negative[word]
    polarity=''
    if (score > 0):
        polarity = 'positive'
    elif (score < 0):
        polarity = 'negative'
    else:
        polarity = 'neutral'
    return score, polarity
```

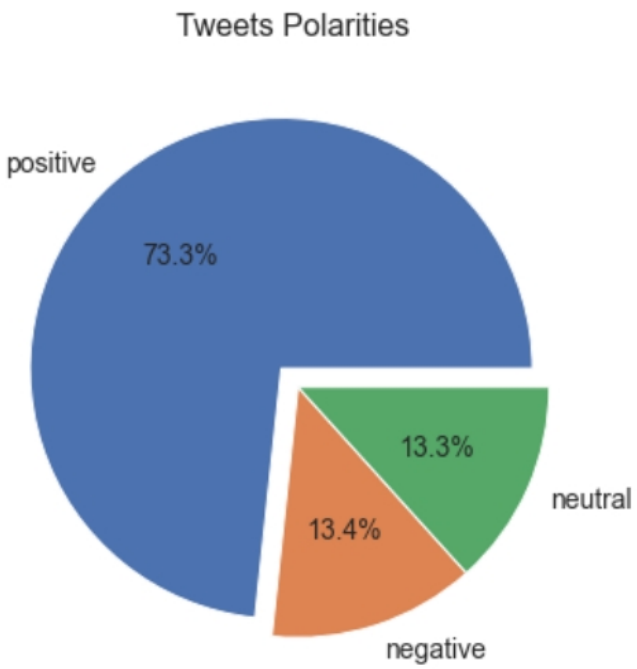
نستخدم الدالة السابقة في حساب قطبية كل تغريدة وذلك بتنفيذ الدالة على قائمة الكلمات التي حصلنا عليها بعد المعالجة الأولية لنص التغريدة `tweet_preprocessed`. أي أنه سيكون لكل تغريدة في نهاية المطاف قطبية `polarity` موجبة أو سالبة أو محايدة وفق مجموع النقاط الحاصلة عليها `polarity_score`. نحفظ نتائج الحساب في ملف جديد `tweets_clean_polarity.csv`.

```
# حساب قطبية التغريدات
results =
tweets['tweet_preprocessed'].apply(sentiment_analysis_dict_arabic)
results = list(zip(*results))
tweets['polarity_score'] = results[0]
tweets['polarity'] = results[1]

# كتابة النتائج في ملف
tweets.to_csv(r'tweets_clean_polarity.csv', encoding='utf-8', index =
False, header = True,index_label=None)
```

تعرض الشيفرة التالية حساب عدد التغريدات من كل قطبيـة (موجبـة، سـالبة، محايـدة) ومن ثم اسـتخدام المكتبة `matplotlib` لرسم مخطط بياني من النوع `pie` يعرض نسب قطبية التغريدات:

```
# رسم نسب قطبية التغريدات
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize = (6, 6))

# حساب عدد التغريدات من كل قطبية
x = [count for count in tweets['polarity'].value_counts()]

# تسميات الرسم
labels = list(tweets['polarity'].value_counts().index)
explode = (0.1, 0, 0)

# تنفيذ الرسم
ax.pie(x = x, labels = labels, autopct = '%1.1f%%', explode = explode,
textprops={'fontsize': 14})

# عنوان الرسم
ax.set_title('Tweets Polarities ', fontsize = 16, pad = 20)

# الإظهار
plt.show()
```

يكون الإظهار:



يُمكن الآن إظهار التغريدات الأكثر إيجابيةً باستخدام الشيفرة التالية:

```
# طباعة أكثر التغريدات إيجابية
pd.set_option('display.max_colwidth', 3000)
positive_tweets = tweets[tweets['polarity'] == 'positive']
positive_tweets = positive_tweets[['tweet_clean', 'polarity_score',
'polarity']].sort_values(by = 'polarity_score',
ascending=False).reset_index(drop = True)
positive_tweets.index += 1
positive_tweets[0:10]
```

يكون الإظهار:

متحف جميل وراقي التنظيم ممتاز الاستعلام والاستقبال جيد طريقة العرض ممتازة والتصنيفات غنية ومتنوعة وجيدة جدا ديكورات عصرية مناسب جدا للزيارات العائلية اسعار التذاكر مقبولة يوجد فيه بازارات للهدايا والتحف يشتمل على سرد كامل وجميل لتارخ الجزيرة العربية والمملكة العربية السعودية في العصور القديمة والحديثة انصح بزيارتة	27	positive
تجربة مثيرة خارجة عن المألوف وممتعة جدا لمختلف الأعمار مع اهتمام كبير من قبل العاملين على مساعدة الزبائن للحصول على أكبر قدر من المتعة و بالرغم من الصعوبة النسبية لبعض الألغاز و التي تتطلب الكثير من البحث و التجربة و الخطأ إلا أن التلميحات التي يقوم بها القيم ماستر تساعد على تخطي هذه الصعوبات أشكر القيم ماستر محمد الناس على جعل تجربتنا تجربة مميزة جدا حيث استمتع الأولاد بتجربة فريدة ونتمنى زيارة المكان مرة أخرى لتجربة غرفة أخرى و شكرا جزيلا مرة أخرى على حسن تعاملك يا استاذ محمد و نتطلع إلى زيارة المكان في القريب العاجل	26	positive
مكان رائع وجميله راقي تاريخي له قيمة كبيرة قطع أثرية قيمه متنوع الاقسام مكيف وتنوع العرض ومجهز بكثير من الإمكانات ويقام فيه كثير من المعارض والفعاليات السياحية والثقافية	25	positive
مكان رائع وجميل وخدمه ممتازه من العماله والمشرفين والاكل ممتاز وسرعه في تقديم الطلب والجلسات في الدور الثاني مره جميله مع الإضاءة والهدوء جميل المنظر في الأعلى والاسعار مناسبه جدا ومكان جميل للعوائل	25	positive

كما يُمكن إظهار التغريدات الأكثر سلبيةً:

```
# طباعة أكثر التغريدات سلبية
pd.set_option('display.max_colwidth', 3000)
negative_tweets = tweets[tweets['polarity'] == 'negative']
negative_tweets = negative_tweets[['tweet_clean', 'polarity_score',
'polarity']].sort_values(by = 'polarity_score', ascending=True)
[0:10].reset_index(drop = True)
negative_tweets.index += 1
negative_tweets[0:10]
```

يكون الإظهار:

1	دورات مياة النساء قذرة جداً جداً كلمة قذرة قليلة بحقها بعض الأبواب ماتتسكر غير أن بعض النساء ترمي زبالتها وزبالة اطفالها بالأرض والزبالة مليانة والطشاش بالأرض بين الوسخ واحياناً مافي مويه الحمامات متهالكه تحتاج صيانة وتجديد يعني مادري كيف اوصف القذارة إلي شفتها اتمنى مايتكرر المشهد بقية المرافق جميلة جداً حقيقة	negative	-14
2	والله المكان رائع زي ماتركته من ٧ سنين لكن الاوضاع مازالت زي ماهي اماكن الحيوانات مش نضيفه وحتى المي حقتهم وسخة جدا في اماكن جديدة معتنى فيها احسن من الباقي الفيله تزعل تحسها كئيبة وبشرتها جافة ولونهم أصفر وفوق كده دخلنا متأخر جدا وكان انتظار طويل ياريت تنتبهو	negative	-14
3	للاسف البيك في المنطقه الشرقيه ادارتهم فاشله خمسين موظف في الفرع رئيسهم فاشل عشان تشتري راح تحتاج ساعه او اكثر اذا قدامك ٣٠ شخص بس طابور نص ساعه او اكثر والكاشير بطيئ ١٠ دقائق او اكثر اذا قدامك ٤ او ٥ اشخاص واللي يجهز الطلب ياخذ نص	negative	-13
4	مشوار طويل لمهرجان عادي جدا فقد هو مجمع للبيع بأغلى سعر مع شدة الرطوبة كانت تجربة سيئة جدا والحشمة والحجاب في المكان تبدو مستغربة جدا	negative	-13
5	كان المكان كالمعتاد مزدحم جدا جدا ووقت الانتظار يحتاج الى توسعة وموقع مناسب يوفر مقاعد اكثر ومواقف لسيارات الاسعار مرتفعة قليلا	negative	-12
6	خايس جدا مافي شى يستاهل تدفع فلوس تلاقي ناس زباله اهمال في كل شى غيرك العينات اللي تجيب المرض أعوذ بالله	negative	-12

يُمكن استخدام مكتبة سحابة الكلمات `WordCloud` لرسم مجموعة من الكلمات بشكل فني كما تُبين الشيفرة التالية:

```
# سحابة الكلمات
from wordcloud import WordCloud

# مكتبة للغة العربية
import arabic_reshaper
from bidi.algorithm import get_display

# انتقاء بعض الكلمات المعالجة
list_words=''
i=0
for tweet in tweets['tweet_preprocessed']:
    for word in tweet:
```

```
        i=i+1
        if i>100:
            break
        list_words += ' '+(word)

# ضبط اللغة العربية
reshaped_text = arabic_reshaper.reshape(list_words)
artext = get_display(reshaped_text)

# إعدادات سحابة الكلمات
wordcloud = WordCloud(font_path='DroidSansMono.ttf', width = 600,
height = 400, background_color = 'black', min_font_size =
10).generate(artext)
fig, ax = plt.subplots(figsize = (8, 6))

# عنوان السحابة
ax.set_title('Word Cloud of Tweets', fontsize = 18)
ax.grid(False)
ax.imshow((wordcloud))
fig.tight_layout(pad=0)
ax.axis('off')
plt.show()
```

يكون الإظهار:

تُجمّع الدالة التالية `words_with_sentiment` الكلمات الموجبة والكلمات السالبة المستخرجة من قائمة الكلمات المُمررة للدالة `list_words` في قائمتين منفصلتين الأولى للكلمات الموجبة والثانية للكلمات السالبة. تستخدم الدالة قاموسي الكلمات الموجبة والسالبة السابقين.

```
# تجميع الكلمات الموجبة والكلمات السالبة
def words_with_sentiment(list_words):
    positive_words=[]
    negative_words=[]
    for word in list_words:
        score_pos = 0
        score_neg = 0
        if (word in dict_positive):
            score_pos = dict_positive[word]
        if (word in dict_negative):
            score_neg = dict_negative[word]

        if (score_pos + score_neg > 0):
            positive_words.append(word)
        elif (score_pos + score_neg < 0):
            negative_words.append(word)

    return positive_words, negative_words
```

نستخدم الدالة السابقة في الشيفرة التالية لاستخراج قائمة الكلمات الموجبة وقائمة الكلمات السالبة من التغريدات، ومن ثم إنشاء سحابتي كلمات لكل منهما لعرض الكلمات الموجبة والكلمات السالبة بشكل فني:

```
# سحابة الكلمات الموجبة والسالبة
# فرز الكلمات الموجبة والسالبة
sentiment_words =
tweets['tweet_preprocessed'].apply(words_with_sentiment)
sentiment_words = list(zip(*sentiment_words))

# قائمة الكلمات الموجبة
positive_words = sentiment_words[0]

# قائمة الكلمات السالبة
negative_words = sentiment_words[1]
```

```
# سحابة الكلمات الموجبة
fig, ax = plt.subplots(1, 2,figsize = (12, 10))
list_words_postive=''
for row_word in positive_words:
    for word in row_word:
        list_words_postive += ' '+(word)
reshaped_text = arabic_reshaper.reshape(list_words_postive)
artext = get_display(reshaped_text)

wordcloud_positive = WordCloud(font_path='DroidSansMono.ttf',width =
800, height = 600, background_color = 'black', colormap = 'Greens'
                               , min_font_size = 10).generate(artext)
ax[0].set_title(' Positive Words', fontsize = 14)
ax[0].grid(False)
ax[0].imshow((wordcloud_positive))
fig.tight_layout(pad=0)
ax[0].axis('off')

# سحابة الكلمات السالبة
list_words_negative=''
for row_word in negative_words:
    for word in row_word:
        list_words_negative += ' '+(word)
reshaped_text = arabic_reshaper.reshape(list_words_negative)
artext = get_display(reshaped_text)

wordcloud_negative = WordCloud(font_path='DroidSansMono.ttf',width =
800, height = 600, background_color = 'black', colormap = 'Reds'
                               , min_font_size = 10).generate(artext)
ax[1].set_title('Negative Words', fontsize = 14)
ax[1].grid(False)
ax[1].imshow((wordcloud_negative))
fig.tight_layout(pad=0)
ax[1].axis('off')

plt.show()
```

يكون الإظهار:

Positive Words

Negative Words

7.6 تحويل النصوص إلى أشعة رقمية

لا تقبل بنى تعلم الآلة النصوص كمدخلات لها، بل تحتاج إلى أشعة رقمية كمدخلات، لذا نستخدم الشيفرة التالية لتحويل الشعاع النصي لكل تغريده `tweet_preprocessed` إلى شعاع رقمي:

```
# تحويل التغريدات إلى أشعة رقمية
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# تركيب جمل التغريدات من المفردات المعالجة
sentences = tweets['tweet_preprocessed'].apply(toSentence)
print(sentences.values[25])
max_words = 5000
max_len = 50

#    التصريح عن المجزئ
#    مع تحديد عدد الكلمات التي ستبقى
#  بالاعتماد على تواترها
tokenizer = Tokenizer(num_words=max_words )

# ملائمة المجزئ لنصوص التغريدات
tokenizer.fit_on_texts(sentences.values)

# تحويل النص إلى قائمة من الأرقام
S = tokenizer.texts_to_sequences(sentences.values)
print(S[0])
```

```
# توحيد أطوال الأشعة
X = pad_sequences(S, maxlen=max_len)
print(X[0])
X.shape
```

نقاط في الشيفرة السابقة لشرحها:

- يُحدّد المتغير `max_words` عدد الكلمات الأعظمي التي سيتم الاحتفاظ بها حيث يُحسب تواتر كل كلمة في كل النصوص ومن ثم تُرتب حسب تواترها (المرتبة الأولى للكلمة ذات التواتر الأكبر). ستُهمل الكلمات ذات المرتبة أكبر من `max_words`.
- يُحدّد المتغير `max_len` طول الشعاع الرقمي النهائي. إذا كان طول الشعاع الرقمي الموافق لنص أقل من `max_len` تُضاف أصفار للشعاع حتى يُصبح طوله مساويًا إلى `max_len`. أما إذا كان طوله أكبر يُقتطع جزءًا منه ليُصبح طوله مساويًا إلى `max_len`.
- تقوم الدالة `fit_on_texts(sentences.values)` بملائمة المُجزِّء tokenizer لنصوص جمل التغريدات أي حساب تواتر الكلمات والاحتفاظ بالكلمات ذات التواتر أكبر أو يساوي `max_words`.

نطبع في الشيفرة السابقة، بهدف التوضيح، ناتج كل مرحلة. اخترنا مثلًا شعاع التغريدة 25 بعد المعالجة:

```
[مكان جميل انصح زيار رسوم دخول]
```

تكون نتيجة تحويل الشعاع السابق النصي إلى شعاع من الأرقام:

```
[246, 1401, 467, 19, 87, 17, 74, 515, 2602, 330, 218, 579, 507, 465,
270, 45, 54, 343, 587, 7, 33, 58, 434, 30, 74, 144, 233, 451, 468]
```

وبعد عملية توحيد الطول يكون الشعاع الرقمي النهائي الناتج:

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0  246 1401  467   19   87   17   74
515 2602  330  218  579  507  465  270   45   54  343  587    7   33
58  434   30   74  144  233  451  468]
```

7.7 تجهيز دخل وخرج الشبكة العصبية

تعرض الشيفرة التالية حساب شعاع الخرج أولًا حيث نقوم بترميز القطبيات الثلاث إلى 0 للسالبة و1 للمحايدة و2 للموجبة.

نستخدم الدالة `train_test_split` لتقسيم البيانات المتاحة إلى 80% منها لعملية التدريب و20% لعملية الاختبار وحساب مقاييس الأداء:

```
# ترميز الخرج
polarity_encode = {'negative' : 0, 'neutral' : 1, 'positive' : 2}

# توليد شعاع الخرج
y = tweets['polarity'].map(polarity_encode).values

# مكتبة تقسيم البيانات إلى تدريب واختبار
from sklearn.model_selection import train_test_split

# تقسيم البيانات إلى تدريب واختبار
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

نطبع في الشيفرة السابقة حجوم أشعة الدخل والخرج للتدريب وللاختبار:

```
(16428, 50) (16428,)

(4107, 50) (4107,)
```

7.8 نموذج الشبكة العصبية المتعلم

تُعدّ المكتبة Keras من أهم مكتبات بايثون التي توفر بناء شبكات عصبية لمسائل التعلم الآلي.

تعرض الشيفرة التالية التصريح عن دالة بناء نموذج التعلّم create_model مـع إعطـاء جميـع المعـاملات المترفعة قيمًا ابتدائية:

```
# تضمين النموذج التسلسلي
from keras.models import Sequential

# تضمين الطبقات اللازمة
from keras.layers import Embedding, Dense, LSTM

# دوال التحسين
from tensorflow.keras.optimizers import Adam, RMSprop

# التصريح عن دالة إنشاء نموذج التعلم
```

```
# مع إعطاء قيم أولية للمعاملات المترفعة
def create_model(embed_dim = 32, hidden_unit = 16, dropout_rate = 0.2,
optimizers = RMSprop, learning_rate = 0.001):
    # التصريح عن نموذج تسلسلي
    model = Sequential()
    # طبقة التضمين
    model.add(Embedding(input_dim = max_words, output_dim = embed_dim,
input_length = max_len))
    # LSTM
    model.add(LSTM(units = hidden_unit ,dropout=dropout_rate))
    # الطبقة الأخيرة
    model.add(Dense(units = 3, activation = 'softmax'))
    # بناء النموذج
    model.compile(loss = 'sparse_categorical_crossentropy', optimizer
= optimizers(learning_rate = learning_rate), metrics = ['accuracy'])
    # طباعة ملخص النموذج
    print(model.summary())


    return model
```

نستخدم من أجل مسألتنا نموذج شبكة عصبية تسلسلي يتألف من ثلاث طبقات:

7.8.1 الطبقة الأولى: طبقة التضمين Embedding

نستخدم هذه الطبقة لتوليد ترميز مكثف للكلمات dense word encoding مما يُساهم في تحسين عملية التعلم. نطلب تحويل الشعاع الذي طوله `input_length` (في حالتنا 50) والذي يحوي قيم ضمن المجال `input_dim` (من 1 إلى 5000 في مثالنا) إلى شعاع من القيم ضمن المجال `output_dim` مثلًا 32 قيمة.

7.8.2 الطبقة الثانية LSTM

يُحدّد المعامل المترفع `units` عدد الوحدات المخفية لهذه الطبقة. يُساهم المعامل `dropout` في معايرة الشبكة خلال التدريب حيث يقوم بإيقاف تشغيل الوحدات المخفية بشكل عشوائي أثناء التدريب، وبهذه الطريقة لا تعتمد الشبكة بنسبة 100% على جميع الخلايا العصبية الخاصة بها، وبدلاً من ذلك، تُجبر نفسها على العثور على أنماط أكثر أهمية في البيانات من أجل زيادة المقياس الذي تحاول تحسينه (الدقة مثلًا).

7.8.3 الطبقة الثالثة Dense

يُحدّد المعامل `units` حجم الخرج لهذه الطبقة (3 في حالتنا: 0 سالبة، 1 محايدة، 2 موجبة) ويُبين الشكل التالي ملخص النموذج:

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 50, 32)            160000
_________________________________________________________________
lstm_12 (LSTM)               (None, 16)                3136
_________________________________________________________________
dense_12 (Dense)             (None, 3)                 51
=================================================================
Total params: 163,187
Trainable params: 163,187
Non-trainable params: 0
```

7.9 معايرة المعاملات الفائقة وصولًا لنموذج أمثلي

يُمكن الوصول لنموذج تعلم أمثلي بمعايرة معاملاته الفائقة وفق معطيات المشروع. لنُبين أولًا الفرق بين المعاملات الفائقة لنموذج والمعاملات الأخرى له:

- المعاملات الفائقة hyperparameters: هي إعدادات خوارزمية التعلّم قبل التدريب (والتي وضعها مصممو الخوارزمية) .
- المعاملات parameters: هي المعاملات التي يتعلّمها النموذج أثناء التدريب مثل أوزان الشبكة العصبية.

تؤثر عملية معايرة المعاملات الفائقة على أداء النموذج لاسيما لجهة التوزان المطلوب بين مشكلة قلة التخصيص underfitting ومشكلة فرط التخصيص overfitting واللتان تؤديان إلى نموذج غير قادر على تعميم أمثلة التدريب وبالتالي لن يتمكن من التصنيف مع معطيات جديدة (يُمكن العودة للرابط من أكاديمية حسوب للمزيد من التفصيل حول هاتين المشكلتين).

تظهر مشكلة قلة التخصيص عندما لا يكون للنموذج درجات حرية كافية ليتعلّم الربط بين الميزات والهدف، وبالتالي يكون له انحياز كبير نحو قيم معينة للهدف. يُمكن تصحيح قلة التخصيص بجعل النموذج أكثر تعقيدًا. أما مشكلة فرط التخصيص فتظهر عندما يقوم النموذج بتخزين بيانات التدريب فيكون له بالتالي تباين كبير والذي يُمكن تصحيحه بالحد من تعقيد النموذج باستخدام التسوية regularization.

تكمن المشكلة في معايرة المعاملات الفائقة بأن قيمها المثلى تختلف من مسألة لأخرى! وبالتالي، فإن الطريقة الوحيدة للوصول لهذه القيم المثلى هي تجريب قيم مختلفة مع كل مجموعة بيانات تدريب جديدة.

يوفر Scikit-Learn العديد من الطرق لتقويم المعاملات الفائقة وبالتالي سنعتمد في مشروعنا عليها دون أن نُعقّد الأمور أكثر.

7.9.1 البحث الشبكي مع التقييم المتقاطع

تُدعى الطريقة التي سنستخدمها في إيجاد القيم المثلى بالبحث الشبكي مع التقويم المتقاطع grid search with cross validation:

- البحث الشبكي: نُعرّف شبكة grid من بعض القيم المُمكنة ومن ثم نولد كل التركيبات المُمكنة بينها.
- التقييم المتقاطع: وهو الطريقة المستخدمة لتقييم مجموعة قيم مُحدّدة للمعاملات الفائقة. عوضًا عن تقسيم البيانات إلى بيانات للتدريب وبيانات للتقييم مما يُخفّض من البيانات التي يُمكن لنا استخدامها للتدريب، نستخدم التقييم المتقاطع مع عدد محدّد من الحاويات K-Fold. تُقسم بيانات التدريب إلى عدد K من الحاويات ومن ثم نقوم بتكرار ما يلي K مرة: في كل مرة نقوم بتدريب النموذج مع بيانات K-1 حاوية ومن ثم تقويمه مع بيانات الحاوية K. في النهاية، يكون مقياس الأداء النهائي هو متوسط الخطأ لكل التكرارات.

يُمكن تلخيص خطوات البحث الشبكي مع التقييم المتقاطع كما يلي:

1. إعداد شبكة من المعاملات الفائقة
2. توليد كل تركيبات قيم المعاملات الفائقة
3. إنشاء نموذج لكل تركيب من القيم
4. تقييم النموذج باستخدام التقويم المتقاطع
5. اختيار تركيب قيم المعاملات ذو الأداء الأفضل

بالطبع، لن نقوم ببرمجة هذه الخطوات لأن الكائن `GridSearchCV` في Scikit-Learn يقوم بكل ذلك:

> يجب ملاحظة أن تنفيذ الشيفرة قد يستغرق بعض الوقت، حوالي الساعة على حاسوب ذو مواصفات عالية.

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# حساب القيم الأمثلية للمعاملات المترفعة
```

```
model = KerasClassifier(build_fn = create_model, epochs = 25,
batch_size=128)


# بعض القيم الممكنة للمعاملات المترفعة
embed_dim = [32, 64]
hidden_unit = [16, 32, 64]
dropout_rate = [0.2]
optimizers = [Adam, RMSprop]
learning_rate = [0.01, 0.001, 0.0001]
epochs = [10, 15,  25 ]
batch_size = [128, 256]
param_grid = dict(embed_dim = embed_dim, hidden_unit = hidden_unit,
dropout_rate = dropout_rate,
                  learning_rate = learning_rate, optimizers =
optimizers, epochs = epochs, batch_size = batch_size)


# تقويم النموذج لاختيار أفضل القيم
grid = GridSearchCV(estimator = model, param_grid = param_grid, cv =
3)
grid_result = grid.fit(X_train, y_train)


results = pd.DataFrame()
results['means'] = grid_result.cv_results_['mean_test_score']
results['stds'] = grid_result.cv_results_['std_test_score']
results['params'] = grid_result.cv_results_['params']
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))


# حفظ النتائج
results.to_csv(r'gridsearchcv_results.csv', index = False, header =
True)
results.sort_values(by='means', ascending =
False).reset_index(drop=True)
```

يُبين خرج الشيفرة السابقة أفضل القيم للمعاملات المترفعة:

```
Best: 0.898588 using {'batch_size': 256, 'dropout_rate': 0.2,
'embed_dim': 32, 'epochs': 10, 'hidden_unit': 64, 'learning_rate':
0.001, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}
```

نحفظ نتائج حساب المعاملات الأمثلية في الملف gridsearchcv_results.csv.

يُمكن معاينة هذه القيم:

```
# قراءة نتائج معايرة المعاملات المترفعة
results = pd.read_csv('gridsearchcv_results.csv')
results.sort_values(by='means', ascending =
False).reset_index(drop=True)
print (results)
```

يكون الخرج:

	means	stds	params
0	0.898588	0.004178	{'batch_size': 256, 'dropout_rate': 0.2, 'embed_dim': 32, 'epochs': 10, 'hidden_unit': 64, 'learning_rate': 0.001, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}
1	0.898162	0.003017	{'batch_size': 128, 'dropout_rate': 0.2, 'embed_dim': 32, 'epochs': 10, 'hidden_unit': 64, 'learning_rate': 0.001, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}
2	0.897553	0.000789	{'batch_size': 256, 'dropout_rate': 0.2, 'embed_dim': 64, 'epochs': 10, 'hidden_unit': 64, 'learning_rate': 0.001, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}
3	0.897370	0.001613	{'batch_size': 128, 'dropout_rate': 0.2, 'embed_dim': 32, 'epochs': 10, 'hidden_unit': 32, 'learning_rate': 0.001, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}
4	0.896883	0.002410	{'batch_size': 128, 'dropout_rate': 0.2, 'embed_dim': 64, 'epochs': 10, 'hidden_unit': 32, 'learning_rate': 0.01, 'optimizers': <class 'keras.optimizer_v2.adam.Adam'>}

7.9.2 حساب أوزان الصفوف

يُمكن أن نلاحظ أن عدد التغريدات ذات القطبية الموجبة (73% من التغريدات) تطغى على عدد التغريـدات السلبية (13%) والتغريدات المحايدة (13%) مما قد يؤدي إلى انحراف نتائج التعلم نحو القطبية الموجبة. يُمكن تلافي ذلك عن طريق الموازنة بين هذه الصفوف الثلاثة.

نحسب في الشيفرة التالية عدد التغريدات الموجبة والسالبة والمحايدة ونسبها:

```
# حساب أوزان القطبيات
posCount=0
negCount=0
neuCount=0

# حساب عدد التغريدات الموجبة والسالبة والمحايدة
for index, row in tweets.iterrows():
    if row['polarity']=='negative':
        negCount=negCount+1
    elif row['polarity']=='positive':
        posCount=posCount+1
```

```
    else:
        neuCount=neuCount+1
print(negCount,  neuCount, posCount)
total=posCount+ negCount+  neuCount

# حساب النسب
weight_for_0 = (1 / negCount) * (total / 3.0)
weight_for_1 = (1 / neuCount) * (total / 3.0)
weight_for_2 = (1 / posCount) * (total / 3.0)
print(weight_for_0, weight_for_1, weight_for_2)

class_weight = {0: weight_for_0, 1: weight_for_1, 2:weight_for_2}
```

يكون ناتج طباعة هذه الأوزان ما يلي (لا حظ الوزن الأصغر للتغريدات الموجبة):

```
2.4954429456799123 2.504573728503476 0.45454545454545453
```

7.9.3 بناء نموذج التعلم النهائي

نستخدم الدالة `KerasClassifier` من `scikit` لبناء المُصنف مع الدالة السابقة `create_model` :

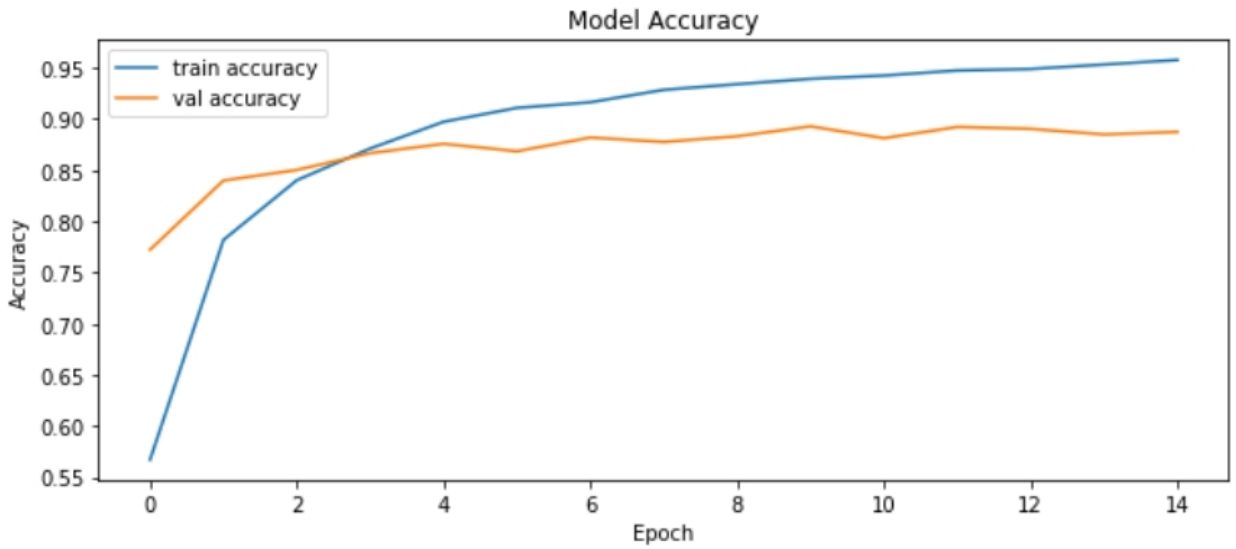
```
# مكتبة التصنيف
from keras.wrappers.scikit_learn import KerasClassifier

# إنشاء النموذج مع قيم المعاملات المترفعة الأمثلية

model = KerasClassifier(build_fn = create_model,
                        # معاملات النموذج
                        dropout_rate = 0.2,
                        embed_dim = 32,
                        hidden_unit = 64,
                        optimizers = Adam,
                        learning_rate = 0.001,
                        # معاملات التدريب
                        epochs=10,
                        batch_size=256,
                        # نسبة بيانات التقييم
                        validation_split = 0.1)
```

```
# ملائمة النموذج مع بيانات التدريب
# مع موازنة الصفوف الثلاثة
model_prediction = model.fit(X_train, y_train,
class_weight=class_weight)
```

يُمكن الآن رسم منحني الدقة accuracy لكل من بيانات التدريب والتقييم (لاحظ أننا في الشيفرة السـابقة احتفاظنا بنسبة 10% من بيانات التدريب للتقييم):

```
# معاينة دقة النموذج
# التدريب والتقييم
fig, ax = plt.subplots(figsize = (10, 4))
ax.plot(model_prediction.history['accuracy'], label = 'train
accuracy')
ax.plot(model_prediction.history['val_accuracy'], label = 'val
accuracy')
ax.set_title('Model Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylabel('Accuracy')
ax.legend(loc = 'upper left')
plt.show()
```

يكون للمنحني الشكل التالي:



7.9.4 حساب مقاييس الأداء

يُمكن الآن حساب مقاييس الأداء المعروفة في مسائل التصـنيف (الصـحة Accuracy، الدقـة Precision، الاستذكار Recall، المقياس F1) للنموذج المتعلم باستخدام الشيفرة التالية:

```
# مقاييس الأداء
# مقياس الصحة
from sklearn.metrics import accuracy_score

# مقياس الدقة
from sklearn.metrics import precision_score

# مقياس الاستذكار
from sklearn.metrics import recall_score

# f1
from sklearn.metrics import f1_score

# مصفوفة الارتباك
from sklearn.metrics import confusion_matrix

# تصنيف بيانات الاختبار
y_pred = model.predict(X_test)

# حساب مقاييس الأداء
accuracy = accuracy_score(y_test, y_pred)
precision=precision_score(y_test, y_pred , average='weighted')
recall= recall_score(y_test, y_pred, zero_division=1,
average='weighted')
f1= f1_score(y_test, y_pred, zero_division=1,  average='weighted')

print('Model Accuracy on Test Data:', accuracy*100)
print('Model Precision on Test Data:', precision*100)
print('Model Recall on Test Data:', recall*100)
print('Model F1 on Test Data:', f1*100)

confusion_matrix(y_test, y_pred)
```

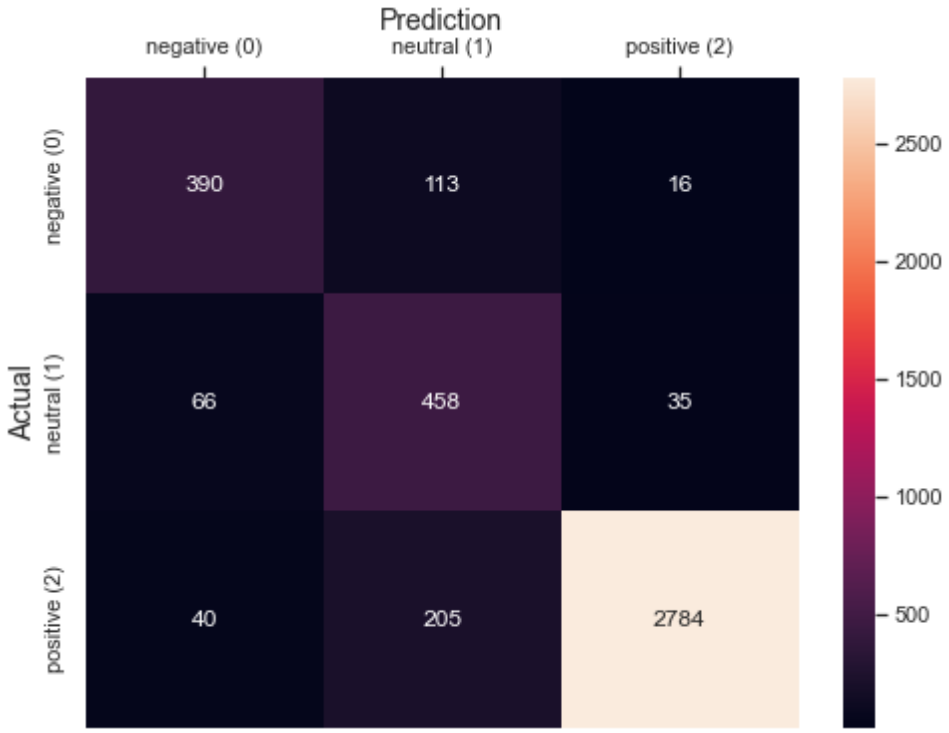
تكون النتائج (لاحظ ارتفاع جميع المقاييس مما يعني جودة المُصنف):

```
Model Accuracy on Test Data: 90.1144387630874

Model Precision on Test Data: 90.90281584915091

Model Recall on Test Data: 90.1144387630874

Model F1 on Test Data: 90.32645671662543

array([[ 366,  129,   24],
       [  53,  444,   62],
       [  20,  118, 2891]], dtype=int64)
```

يُمكن رسم مصفوفة الارتباك confusion matrix بشكل أوضح باستخدام المكتبة seaborn:

```
# رسم مصفوفة الارتباك
import seaborn as sns
sns.set(style = 'whitegrid')

fig, ax = plt.subplots(figsize = (8,6))
sns.heatmap(confusion_matrix(y_true = y_test, y_pred = y_pred), fmt =
'g', annot = True)
ax.xaxis.set_label_position('top')
ax.xaxis.set_ticks_position('top')
ax.set_xlabel('Prediction', fontsize = 14)
ax.set_xticklabels(['negative (0)', 'neutral (1)', 'positive (2)'])
ax.set_ylabel('Actual', fontsize = 14)
ax.set_yticklabels(['negative (0)', 'neutral (1)', 'positive (2)'])
plt.show()
```

مما يُظهر:



يُمكن حسـاب بعض مقـاييس الأداء الأخـرى المُسـتخدمة في حالـة وجـود أكـثر من صـف في المسـألة (Micro, Macro, Weighted):

```
# مقاييس الأداء في حالة أكثر من صفين
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))


print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,
average='micro')))


print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='macro')))
print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,
average='macro')))


print('Weighted Precision: {:.2f}'.format(precision_score(y_test,
y_pred, average='weighted')))
```

```
print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred,
average='weighted')))


# تقرير التصنيف
from sklearn.metrics import classification_report
print('\nClassification Report\n')
print(classification_report(y_test, y_pred, target_names=['Class 1',
'Class 2', 'Class 3']))
```

مما يُعطي (لاحظ ارتفاع جميع المقاييس مما يعني جودة المُصنف):

```
Accuracy: 0.88

Micro Precision: 0.88
Micro Recall: 0.88
Micro F1-score: 0.88

Macro Precision: 0.79
Macro Recall: 0.83
Macro F1-score: 0.80

Weighted Precision: 0.90
Weighted Recall: 0.88
Weighted F1-score: 0.89

Classification Report

              precision    recall  f1-score   support

     Class 1       0.79      0.75      0.77       519
     Class 2       0.59      0.82      0.69       559
     Class 3       0.98      0.92      0.95      3029

    accuracy                           0.88      4107
   macro avg       0.79      0.83      0.80      4107
weighted avg       0.90      0.88      0.89      4107
```

يُمكن أيضًا اختيار مجموعة تغريدات عشوائية جديدة وتصنيفها وحفظ النتائج في ملف results.csv:

```
# تصنيف مجموعة اختبار
text_clean = tweets['tweet_clean']
text_train, text_test = train_test_split(text_clean, test_size = 0.2,
random_state = 0)
result_test = pd.DataFrame(data = zip(text_test, y_pred), columns =
['text', 'polarity'])
polarity_decode = {0 : 'Negative', 1 : 'Neutral', 2 : 'Positive'}
result_test['polarity'] = result_test['polarity'].map(polarity_decode)
pd.set_option('max_colwidth', 300)

# حفظ النتائج
result_test.to_csv("results.csv")
result_test
```

تكون النتائج مثلًا:

Neutral	ليت بس التذاكر يقل سعره تكون ريال كافي
Positive	لمحبين التصوير من الاماكن الجملية ولمعرفة اوقات الافتتاح التواصل مع هيئة السياحة
Positive	متخف جميل يحكي تاريخ المدينة وسيرة الرسول صلى الله عليه وسلم يوجد مرشد معك اثناء الزيارة يقدم لك لمحة تاريخية عن المعرض رسوم الدخول ريال للشخص الواحد
Positive	يوجد به قطع أثرية من عصر صدر الإسلام والعصر العباسي والعصر الجاهلي بصراحه رائع
Neutral	تستحق الزيارة مكان ممتع وجميل لكن للأسف اسعار المطاعم والكافيات والتسوق مرتفعه جدا

7.10 الخلاصة

عرضنا في هذا الفصل خطوات بناء نموذج تعلّم لتصنيف النصوص العربية إلى موجبة وسالبة ومحايدة.

> يُمكن تجربة المثال كاملًا من موقع Google Colab من هذا الرابط ولا تنسَ الاطلاع على مجموعة البيانات المتوفرة الخاصة بهذا الفصل.

8. تجزئة عملاء متجر إلكتروني باستخدام خوارزميات العنقدة

تُعدّ مسألة عنقدة عملاء أي تجزئتهم إلى مجموعات متشابهة في سلوكها (عمليات الشراء والتسوق) من المسائل الهامة في عالم التسويق وذلك بهدف إعداد حملات تسويقية مختلفة مناسبة لكل مجموعة من المجموعات المُستهدفة.

نعرض في هذا الفصل استخدام خوارزميات العنقدة Clustering لاستكشاف مجموعات العملاء أو الزبائن المختلفة.

يُمكن تنزيل بيانات التدريب والشيفرة البرمجية من الملف المضغوط المرفق بهذا الرابط.

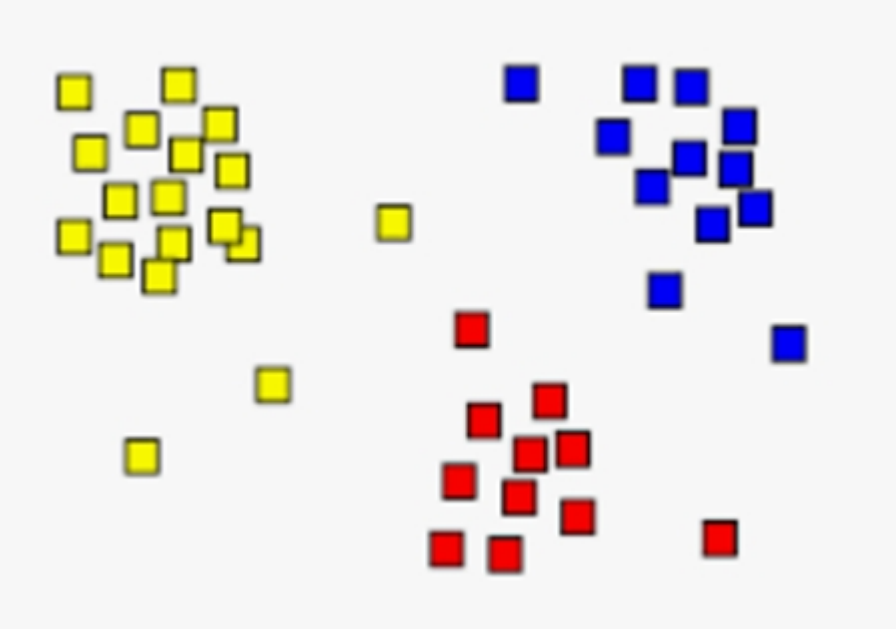
نستخدم في هذا الفصل التعليمية مجموعة بيانات حول 200 زبون تتألف من عمودين:

- الدخل السنوي للزبون بآلاف الدولارات Annual Income.
- تقييم الزبون Spending Score وهو رقم يتراوح بين 1 و 100 ويعكس مدى ولاء الزبون للمتجر والمبالغ التي أنفقها في الشراء منه (100 يعني الزبون الأكثر إنفاقًا).

يُمكن تنزيل هذه البيانات المتاحة على موقع Kaggle من هذا الرابط أو يمكنك تنزيلها هذا الرابط من أكاديمية حسوب.

8.1 ما هي العنقدة Clustering

العنقدة هي عملية تجزئة (تقسيم) مجموعة من البيانات إلى عناقيد clusters (مجموعات) وبحيث تكون العناصر الموجودة في عنقود واحد متشابهة فيما بينها وأقل تشابهًا مع عناصر العناقيد الأخرى.



تنـــدرج خوارزميـــات العنقـــدة ضـــمن خوارزميـــات التعلم الآلي المـــدعوة "ليســـت تحت الإشـــراف" أو unsupervised بمعنى أن البيانات المُقدّمة في البدايـة لا تحـوي أي تسـميات labels أو تصـنيفات بشـكل مُسبق. سنرى مثًلا أن مسألة تحديد عدد العناقيد هو مسألة مهمة يجب الاعتناء بهـا إذ لا نعـرف مُسـبقًا مـا هـو عدد العناقيد الأمثلي.

نعرض في هذا الفصل نوعين أساسيين من خوارزميات العنقدة:

- خوارزميات التجزئة Partitioning: تبدأ هذه الخوارزميات بتجزئة عناصر البيانات إلى k عنقـود (مختـارة بشكل عشوائي) ومن ثم تُكرر تعديل هذه المجموعات وصولًا إلى حل جيد. من أشهر هـذه الخوارزميـات الخوارزمية المدعوة k-means (أي k وسطي).
- الخوارزميات التكتلية Agglomerative: تبني هذه الخوارزمية هرمية من العناقيد حيث تبدأ بوضع كـل عنصر من عناصر البيانات في عنقود مستقل ثم تكرر دمج العناقيد المتشابهة وصـولًا إلى عنقـود واحـد في نهاية المطاف.

8.2 الخوارزمية K-Means

تُعدّ الخوارزمية K-Means من أشهر وأبسط خوارزميات العنقدة بالتجزئة:

دخل الخوارزمية:

- عدد العناقيد المطلوب k
- بيانات التدريب وهي عبارة عن مجموعة من العناصر (أشعة رقمية): {x1, x2, . . . , xN} حيث xi هو شعاع من الأرقام.

خرج الخوارزمية:

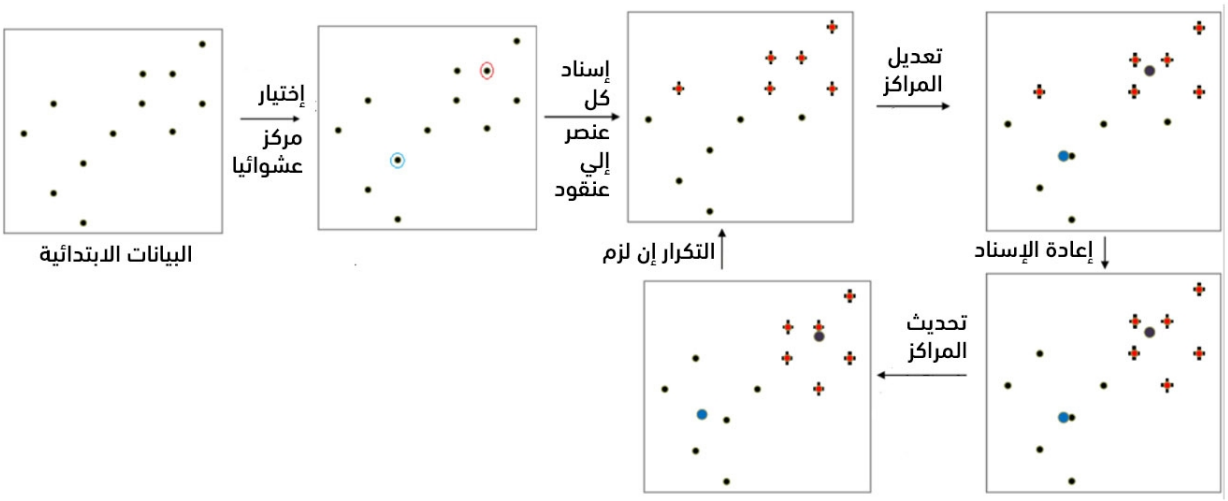
- تُسندّ الخوارزمية كل عنصر من عناصر الدخل إلى أحد العناقيد (التي عددها k).

المعالجة:

- تختار الخوارزمية أولًا k شعاع من أشعة الدخل بشكل عشوائي كمراكز العناقيد.
- تُكرر الخوارزمية ما يلي وصولًا إلى التقارب:
  - إسناد كل شعاع من أشعة الدخل إلى المركز الأقرب
  - حساب المركز الجديد لكل عنقود

يعني التقارب عدم انتقال أي شعاع من عنقوده الحالي الموضوع فيه إلى عنقود آخر.

يُبين الشكل التالي المراحل الأساسية للخوارزمية:



لفهم مراحل الخوارزمية نعرض أولًا خطوات تنفيذها على المثال التعليمي التالي:

ليكن لدينا مجموعة العناصر العشرة التالية:

```
D = { (5,3), (10,15), (15,12), (24,10), (30,45), (85,70), (71,80),
(60,78), (55,52), (80,91) }
```

وبفرض أننا نريد تقسيمها إلى عنقودين (k=2).

لنختار مثلًا العنصر الأول كمركز للعنقود الأول والعنصر الثاني مركزًا للعنقود الثاني أي:

```
c1(x) = 5, c1(y)=3


c2(x) = 10 , c2(y)=15
```

نحسب الآن بُعد (المسافة الإقليدية) كل عنصر من عناصر البيانات عن كل من المركزين ونُسند العنصر إلى المركز الأقرب. يكون لدينا:

رقم	العنصر	المسافة الإقليدية عن المركز الأول	المسافة الإقليدية عن المركز الثاني	العنقود المُسند إليه
1	(5,3)	0	13	C1
2	(10,15)	13	0	C2
3	(15,12)	13.45	5.83	C2
4	(24,10)	20.24	14.86	C2
5	(30,45)	48.87	36	C2
6	(85,70)	104.35	93	C2
7	(71,80)	101.41	89	C2
8	(60,78)	93	80	C2
9	(55,52)	70	58	C2
10	(80,91)	115.52	103.32	C2

نحسب الآن مركزي العنقودين فنحصل على:

```
c1(x) = 5, c1(y)=3

c2(x) = (10 + 15 + 24 + 30 + 85 + 71 + 60 + 55 + 80) / 9 = 47.77

c2(y) = (15 + 12 + 10 + 45 + 70 + 80 + 78 + 52 + 91) / 9 = 50.33
```

نُعيد العملية لنحصل على:

رقم	العنصر	المسافة الإقليدية عن المركز الأول	المسافة الإقليدية عن المركز الثاني	العنقود المُسند إليه
1	(5,3)	0	63.79	C1
2	(10,15)	13	51.71	C1
3	(15,12)	13.45	50.42	C1
4	(24,10)	20.24	46.81	C1
5	(30,45)	48.87	18.55	C2
6	(85,70)	104.35	42.1	C2
7	(71,80)	101.41	37.68	C2

رقم	العنصر	المسافة الإقليدية عن المركز الأول	المسافة الإقليدية عن المركز الثاني	العنقود المُسند إليه
8	(60,78)	93	30.25	C2
9	(55,52)	70	7.42	C2
10	(80,91)	115.52	51.89	C2

نحسب الآن مركزي العنقودين الجديدين فنحصل على:

```
c1(x) = (5, 10, 15, 24) / 4 = 13.5

c1(y) = (3, 15, 12, 10) / 4 = 10.0

c2(x) = (30 + 85 + 71 + 60 + 55 + 80) / 6 = 63.5

c2(y) = (45 + 70 + 80 + 78 + 52 +91) / 6 = 69.33
```

نُعيد العملية مرة أخرى لنحصل على:

رقم	العنصر	المسافة الإقليدية عن المركز الأول	المسافة الإقليدية عن المركز الثاني	العنقود المُسند إليه
1	(5,3)	11.01	88.44	C1
2	(10,15)	6.1	76.24	C1
3	(15,12)	2.5	75.09	C1
4	(24,10)	10.5	71.27	C1
5	(30,45)	38.69	41.4	C1
6	(85,70)	93.33	21.51	C2
7	(71,80)	90.58	13.04	C2
8	(60,78)	82.37	9.34	C2
9	(55,52)	59.04	19.3	C2
10	(80,91)	104.8	27.23	C2

نحسب الآن مركزي العنقودين الجديدين فنحصل على:

```
c1(x) = (5, 10, 15, 24, 30) / 5 = 16.8

c1(y) = (3, 15, 12, 10, 45) / 5 = 17.0
```

```
c2(x) = (85 + 71 + 60 + 55 + 80) / 5 = 70.2

c2(y) = (70 + 80 + 78 + 52 + 91) / 5 = 74.2
```

نُعيد العملية فلا يتغير إسناد أي عنصر مما يعني التقارب وثبات المراكز والعناقيد.

8.3 إعداد المشروع

يحتاج تنفذ شيفرات هـذا الفصـل بيئـةً برمجيـةً للغـة بـايثون الإصـدار 3.8. ويجب أن تتضـمن هـذه البيئـة البرمجية مدير الحِزم pip لتثبيت الحِزم، ومُنشئ البيئات الافتراضية venv لإنشاء بيئاتٍ افتراضيةٍ.

نستخدم محرر الشيفرات البرمجية Jupyter Notebooks، وهو مفيد جدًا لتجربة وتشغيل الأمثلة البرمجية بطريقةٍ تفاعليةٍ، حيث نستطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسـرعة، ممـا يُسهّل علينا اختبار الشيفرات البرمجية وتصحيحها.

نحتـاج أولًا لتثـبيت بعض التبعيـات، وذلـك لإنشـاء مسـاحة عمـلٍ للاحتفـاظ بملفاتنـا قبـل أن نتمكن من تطوير برنامجنا.

نُنشئ مجلدًا جديدًا خاصًا بمشروعنا وندخل إليه هكذا:

```
mkdir clustering
cd clustering
```

نُنفذّ الأمر التالي لإنشاء البيئة الافتراضية:

```
python -m venv clustering
```

ومن ثم الأمر التالي في Linux لتنشيط البيئة الافتراضية:

```
source clustering/bin/activate
```

أما في Windows، فيكون أمر التنشيط:

```
"clustering/Scripts/activate.bat"
```

نستخدم إصداراتٍ محددةٍ من المكتبات اللازمة، من خلال إنشـاء ملـف `requirements.txt` في مجلـد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

نفتح الملف `requirements.txt` في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبـات التي نريدها وإصداراتها:

```
jupyter==1.0.0
numpy==1.21.5
scikit-learn==1.0.1
scipy==1.7.3
pandas==1.3.5
matplotlib==3.5.1
```

نحفظ التغييرات التي طرأت على الملف ونخرج من محرر النصوص، ثم نُثَبت هذه المكتبات بالأمر التالي:

```
(clustering) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، نُصبح جاهزين لبدء العمل على مشروعنا.

8.4 كتابة شيفرة تطبيق العنقدة للعملاء

نُشغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت. هكذا:

```
(clustering) $ jupyter notebook
```

ثم نُنشئ ملفًا جديدًا في داخل المحرر ونُسمّه باسم clust مثلًا.

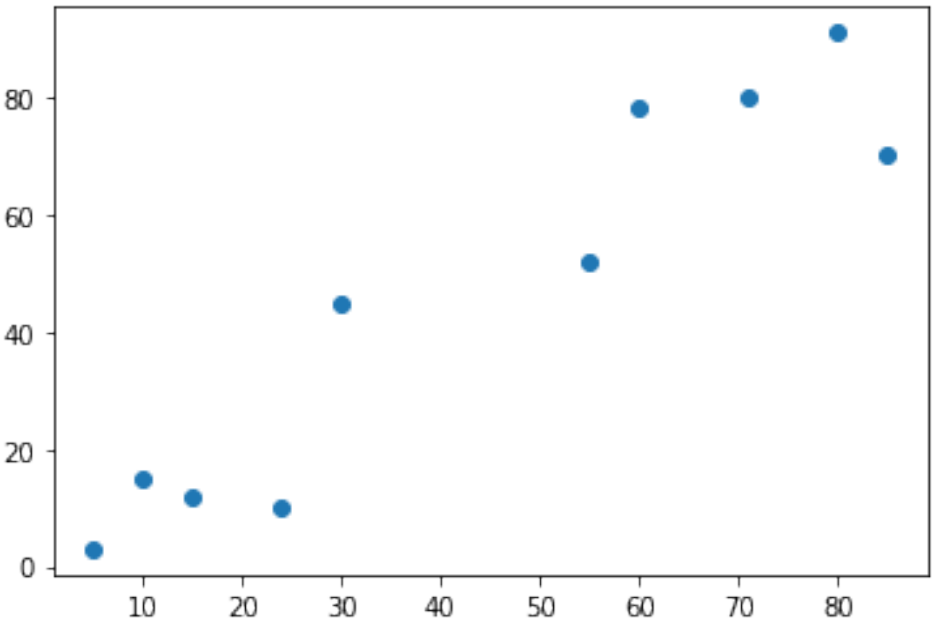
8.4.1 تنفيذ المثال التعليمي

نبدأ أولًا بإنشاء بيانات المثال التعليمي السابق ورسمها باستخدام المكتبة `matplotlib`:

```
# المكتبة الرقمية
import numpy as np
# مكتبة الرسم
import matplotlib.pyplot as plt
# إنشاء البيانات
X = np.array([[5,3],
     [10,15],
     [15,12],
     [24,10],
     [30,45],
     [85,70],
     [71,80],
     [60,78],
     [55,52],
```

```
    [80,91],])
# الرسم
plt.scatter(X[:,0],X[:,1])
plt.show()
```

مما يُظهر:



تُبين الشيفرة التالية استخدام الصف `KMeans` من مكتبة العنقدة `sklearn.cluster`. نُنشئ أولًا غـرض (نموذج model) من هذا الصف مع تحديد عدد العناقيد المطلوبة. تُنفذّ الطريقـة `fit` خوارزميـة KMeans على بيانات التدريب لإيجاد العناقيد المطلوبة. تُرقّم الخوارزميـة العناقيـد بـدءًا من الصـفر (تسـميات labels). يُمكن طبعًا طباعة مراكز العناقيد.

```
# KMeans
from sklearn.cluster import KMeans
# إنشاء غرض من الصف
# تحديد عدد العناقيد المطلوب
kmeans = KMeans(n_clusters=2)
# الملائمة مع البيانات
kmeans.fit(X)
# طباعة المراكز
print(kmeans.cluster_centers_)
# طباعة تسميات العناقيد
print(kmeans.labels_)
```

يكون الخرج:

```
[[70.2 74.2] [16.8 17. ]]
[1 1 1 1 1 0 0 0 0 0]
```

يُمكن الآن استخدام النموذج المُتعلم السابق للتنبؤ بعنقود أي مثال جديد وذلك باستخدام الطريقة `predict` كما تُبين الشيفرة التالية:

```
# التنبؤ
print(kmeans.predict([[10,10]]))
```

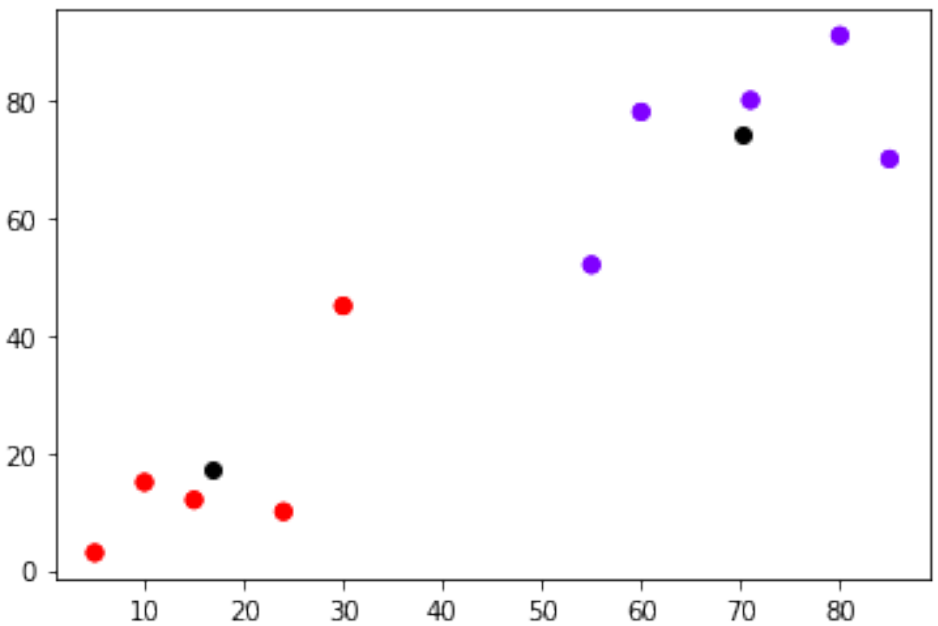
حيث يكون الخرج مثلًا:

```
[1]
```

يُمكن أيضًا رسم نقاط التدريب مع مراكزها بلون مميز كما تُبين الشيفرة التالية:

```
# الرسم مع المراكز
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1]
, color='black')
```

يكون الخرج:



ننتقل الآن لمعالجة مسألتنا الأساسية وهي عنقدة الزبائن. تعرض الشيفرة التالية تحميل البيانات من الملف `shopping-data.csv` ووضعها في إطار بيانات من المكتبة `pandas`.

```
# مكتبة إطار البيانات
import pandas as pd

# تحميل البيانات
customer_data = pd.read_csv('shopping-data.csv')

# إظهار ترويسة البيانات
customer_data.head()
```

تظهر أوائل البيانات:

	CustomerID	Annual Income (k$)	Spending Score (1-100)
0	1	15	39
1	2	15	81
2	3	16	6
3	4	16	77
4	5	17	40

نحذف في الشيفرة التالية رقم الزبون من إطار البيانات كي لا يدخل في عملية حساب العناقيد. ثم نُنشئ نموذجًا متعلمًا من الصف KMeans مع عدد العناقيد مساويًا لخمسة (نعرض لاحقًا كيفية تحديدنا لهذا الرقم):

```
# حذف رقم الزبون
data = customer_data.iloc[:, 1:3].values
# إنشاء النموذج المتعلم
kmeans = KMeans(n_clusters=5)
# الملائمة مع البيانات
kmeans.fit(data)
# طباعة المراكز
print(kmeans.cluster_centers_)
# طباعة التسميات
print(kmeans.labels_)
```

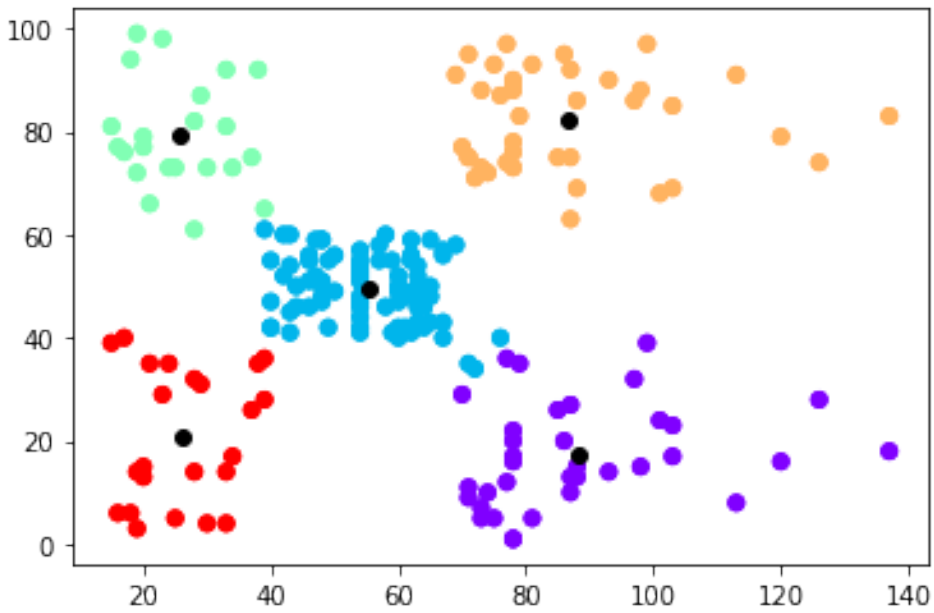
يكون الناتج:

```
[[88.2        17.11428571] [55.2962963  49.51851852] [25.72727273
79.36363636] [86.53846154 82.12820513] [26.30434783 20.91304348]]

[4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4
2 4 2 4 2 4 2 4 1 4 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 0 3 1 3 0 3 0 3 1 3 0 3 0 3 0 3
0 3 1 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0
3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3 0 3]
```

يُمكن الآن أن نرسم النقاط مع المراكز باستخدام الشيفرة التالية:

```
# الرسم
plt.scatter(data[:,0],data[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(data[:,0], data[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1]
, color='black')
plt.show()
```

مما يُعطي:



8.5 تفسير نتائج العنقدة تقييمها

يُمكن لنا بمعاينة الشكل السابق استنتاج ما يلي (وهو الهدف الأساسي من عملية العنقدة):

- الزبائن في أعلى اليمين (نقاط البيانات الصفراء) هم الزبائن ذوو الرواتب المرتفعة والإنفاق المرتفع. هؤلاء هم الزبائن الواجب استهدافهم دومًا والمحافظة عليهم.

- الزبائن في أعلى اليسار (نقاط البيانات الخضراء) هم الزبائن ذوو الرواتب المنخفضة والإنفاق المرتفع. هؤلاء هم الزبائن الواجب العناية بهم واستهدافهم بحملات التخفيضات مثلًا.
- الزبائن في أسفل اليمين (نقاط البيانات البنفسجية) هم الزبائن ذوو الرواتب المرتفعة والإنفاق المنخفض. هؤلاء هم الزبائن الواجب جذبهم بالطرق التسويقية ما أمكن ذلك.
- الزبائن في أسفل اليسار (نقاط البيانات الحمراء) هم الزبائن ذوو الرواتب المنخفضة والإنفاق المنخفض. لا داع لإضاعة الكثير من الوقت والجهد معهم.
- الزبائن في الوسط (نقاط البيانات الزرقاء) هم الزبائن أصحاب الدخل المتوسط والإنفاق المتوسط. يُمكن استهدافهم أيضًا لاسيما أن عددهم كبيرًا كما يُبين الشكل السابق.

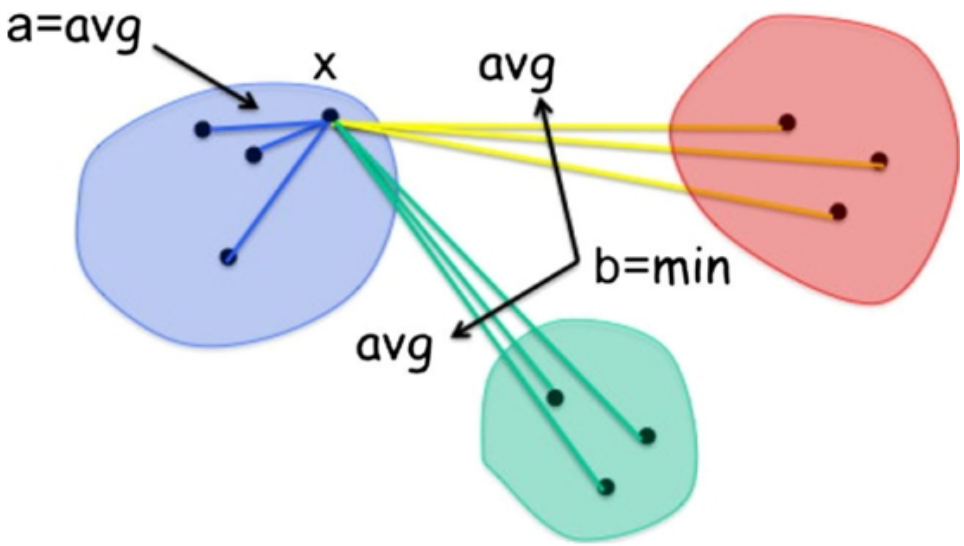
يُمكن تقييم نتائج العنقدة باستخدام المعامل المدعو Silhouette Coefficient، ويُحسب هذا المعامل من أجل عنصر بيانات x كما يلي:

$$SC_x = \frac{(b-a)}{max(a,b)}$$

حيث:

- a هي وسطي المسافات بين x وبين العناصر الموجودة في نفس العنقود الموجود فيه العنصر x.
- b هي وسطي المسافات بين x وبين العناصر الموجودة في العناقيد القريبة من عنقود x.

كما يُبين الشكل التالي:



يُمكن ملاحظة ما يلي حول هذا المعامل:

- أفضل قيمة له هي 1 وأسوأ قيمة له هي 1- .
- تُشير القيمة 0 إلى وجد تراكب overlapping في العناقيد.
- تدل القيمة السالبة على وجود العنصر x في عنقود خاطئ إذ يوجد عنقود آخر أقرب إلى x.

يكون المعامل `silhouette_score` لمجموعة من البيانات وسطي قيمة المعامل لكل منها أي:

$$silhouette\_score = Average(SC_{x\ in\ X})$$

يُمكن تضمين حساب هذا المعامل من `sklearn.metrics` كما تُبين الشيفرة التالية:

```
from sklearn.metrics import silhouette_score
cluster_labels = kmeans.fit_predict(data)

# حساب المعامل
silhouette_avg = silhouette_score(data, cluster_labels)
print(silhouette_avg)
```

يكـون لطريقـة حسـاب المعامـل `silhouette_score` معـاملي دخـل الأول هـو البيانـات والثـاني هـو تسميات العناقيد الناتجة عن استدعاء الطريقة `fit_predict` أولًا.

يكون الناتج في مثالنا:

```
0.553931997444648
```

8.6 إيجاد عدد العناقيد الأمثلي

يُمكن استخدام المعامل Silhouette Coefficient لإيجاد عدد العناقيد الأمثلي المناسب لبيانات التدريب وذلك بحساب المعامل مع قيم مختلفة لعدد العناقيد ومن ثم اختيار العدد الذي يُعطي قيمـة أعظميـة للمعامـل كما تُبين الشيفرة التالية:

```
# مجموعة القيم الممكنة
range_n_clusters = [2, 3, 4, 5, 6, 7]
for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters)
    cluster_labels = clusterer.fit_predict(data)
    # حساب المعامل
    silhouette_avg = silhouette_score(data, cluster_labels)
```

```
    print(
        "For n =",
        n_clusters,
        "silhouette_score is :",
        silhouette_avg,
    )
```

مما يُظهر المعامل من أجل كل قيمة محتملة لعدد العناقيد:

```
For n = 2 silhouette_score is : 0.2968969162503008
For n = 3 silhouette_score is : 0.46761358158775435
For n = 4 silhouette_score is : 0.4931963109249047
For n = 5 silhouette_score is : 0.553931997444648
For n = 6 silhouette_score is : 0.5376203956398481
For n = 7 silhouette_score is : 0.5270287298101395
```

وبالتالي نلاحظ أن القيمة الأمثلية هي خمسة عناقيد.

8.7 الخوارزميات التكتلية

تمتاز الخوارزميات التكتلية بأنها توفر طريقة معاينة بسيطة لإيجاد عدد العناقيد الأمثلي عن طريق رسم شجرة دمج العناقيد مع بعضها البعض dendrogram:

دخل الخوارزمية:

- بيانات التدريب وهي عبارة عن مجموعة من الأشعة الرقمية: {x1, x2, . . . , xN} حيث xi هو شعاع من الأرقام.

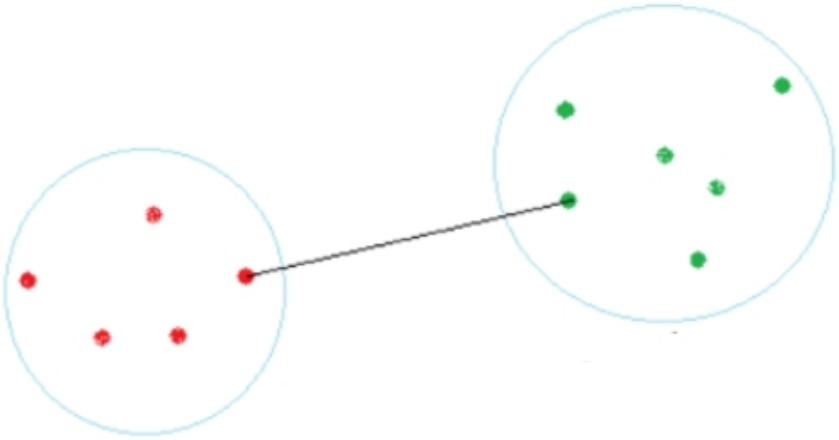
خرج الخوارزمية:

- شجرة دمج العناقيد مع بعضها البعض dendrogram.

المعالجة:

- تبدأ الخوارزمية بوضع كل عنصر من عناصر البيانات في عنقود.
- تُكرر الخوارزمية ما يلي وصولًا إلى عنقود وحيد:
  - دمج أقرب عنقودين مع بعضهما البعض

يوجد عدة طرق لإيجاد أقرب عنقودين، من أبسطها الطريقة المسماة بالربط البسيط Single Linkage والتي تحسب المسافة بين عنقودين كما يلي: المسافة بين عنقودين هي أصغر مسافة بين عنصرين من هذين العنقودين.

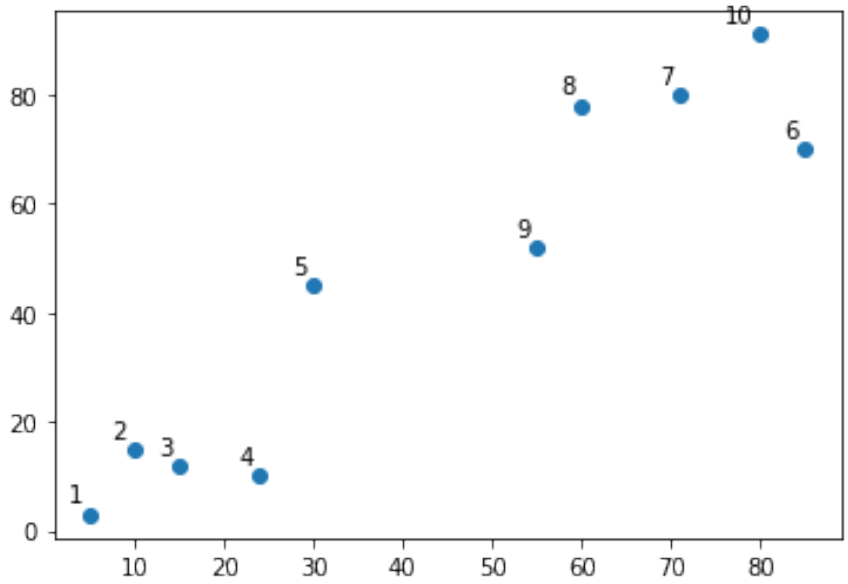


نستخدم فيما يلي الخوارزمية التكتلية مع البيانات التعليمية ومن ثم مع بيانات الزبائن.

تعرض الشيفرة التالية طريقة بسيطة لتسمية نقاط البيانات التعليمية (العشرة) لتسهيل عملية المشاهدة والمتابعة:

```
# 10 ,... ,2,1 : التسميات
labels = range(1, 11)
plt.subplots_adjust(bottom=0.1)
plt.scatter(X[:,0],X[:,1])

# تسمية النقاط
for label, x, y in zip(labels, X[:, 0], X[:, 1]):
    plt.annotate(
        label,
        xy=(x, y), xytext=(-3, 3),
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```

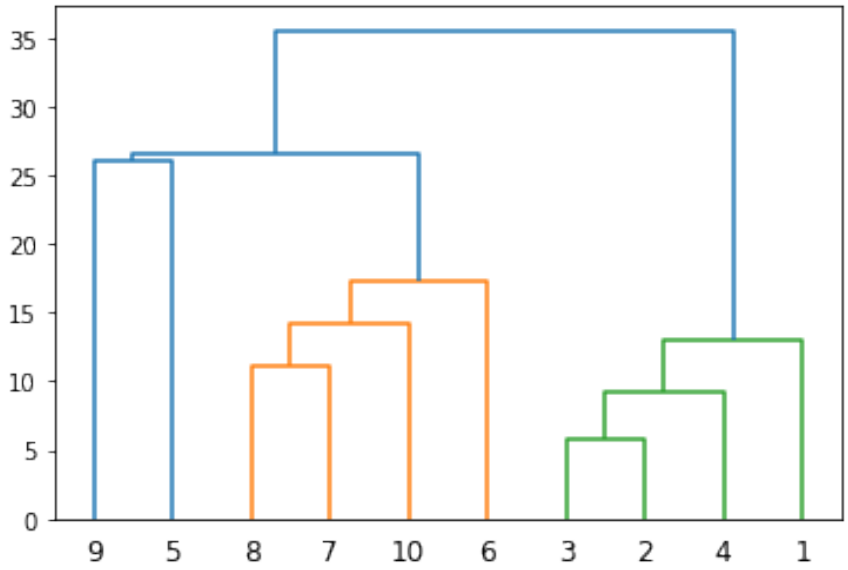
يكون الخرج:



نبدأ أولًا بالشيفرة اللازمة لرسم شجرة الدمج dendrogram من أجل إيجاد عدد العناقيد الأمثل:

```
# المكتبات اللازمة
from scipy.cluster.hierarchy import dendrogram, linkage

# اختيار الربط البسيط
linked = linkage(X, 'single')

# التسميات: 1,2, .. ,10
labelList = range(1, 11)

# dendrogram
dendrogram(linked,
            orientation='top',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()
```

نُعاين شجرة الدمج الناتجة والتي تُبين أن عدد العناقيد المُمكن هو عنقودين:



نستدعي الآن الدالة `AgglomerativeClustering` مع طلب عدد العناقيد مساويًا إلى 2:

```
from sklearn.cluster import AgglomerativeClustering

# العنقدة التكتلية
cluster = AgglomerativeClustering(n_clusters=2)

# الملائمة مع البيانات
cluster.fit_predict(X)
```

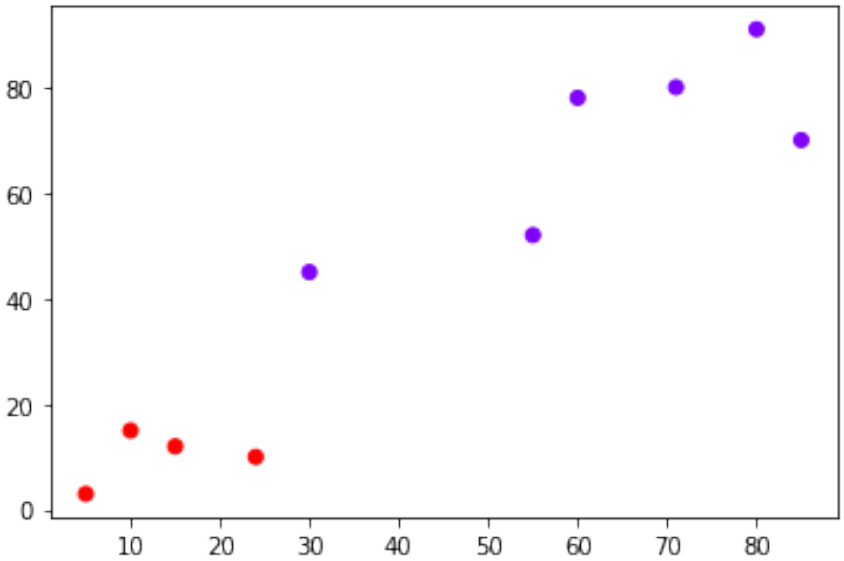
والتي تُظهر تسميات عنقود كل عنصر:

```
array([1, 1, 1, 1, 0, 0, 0, 0, 0, 0], dtype=int64)
```

يُمكن أيضًا رسم العناقيد:

```
# الرسم
plt.scatter(X[:,0],X[:,1], c=cluster.labels_, cmap='rainbow')
plt.show()
```

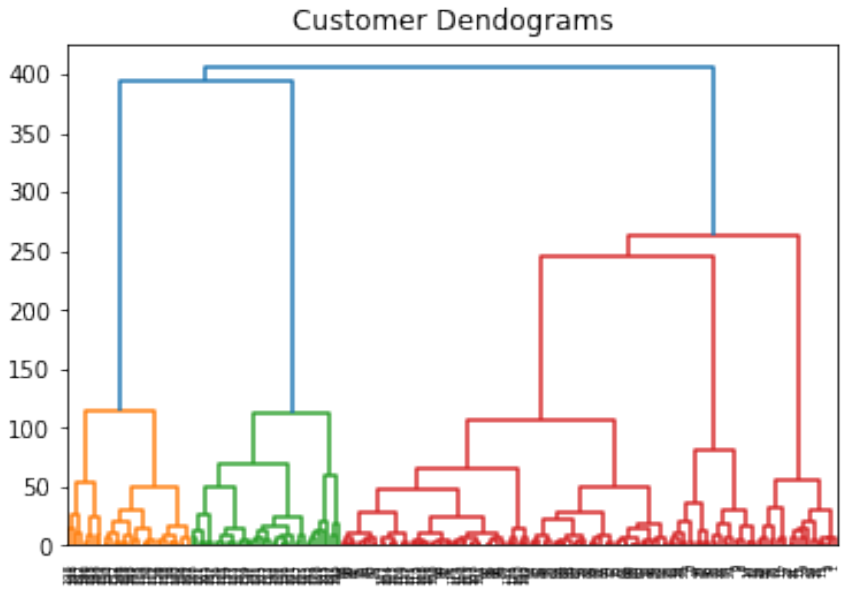
يكون الإظهار:



ننتقل الآن إلى مسألة الزبائن باستخدام شيفرة مماثلة:

```
# نوع الربط
linked = linkage(data, 'ward')

# dendrogram
dendrogram(linked,
            orientation='top',
            distance_sort='descending',
            show_leaf_counts=True)

# الرسم
plt.title("Customer Dendograms")
plt.show()
```

يُمكن معاينة شجرة الدمج الناتجة:

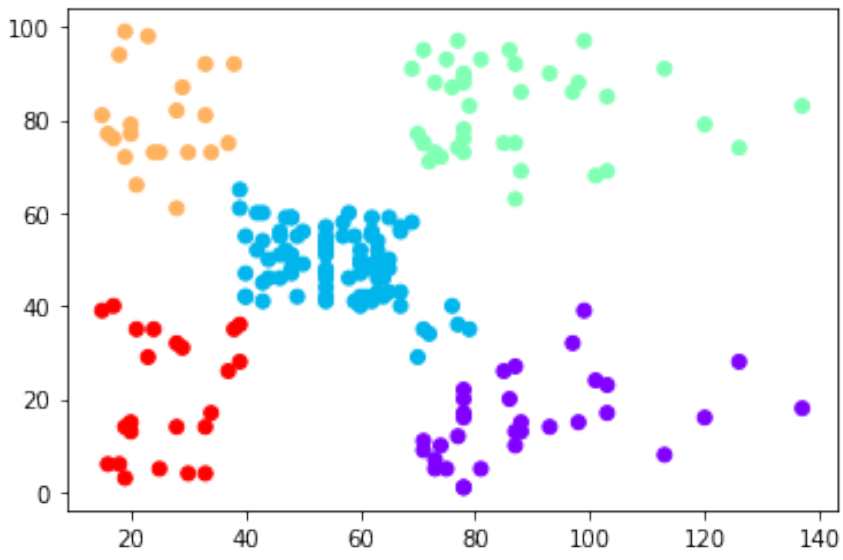


والتي تُبين أن عدد العناقيد المناسب هو 5.

نستدعي الدالة `AgglomerativeClustering` في الشيفرة التالية:

```
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')
cluster.fit_predict(data)
# الرسم
plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
plt.show()
```

ويكون الخرج:



وهو، بالطبع، يُشابه ما حصلنا عليه سابقًا مع الخوارزمية KMeans.

8.8 الخلاصة

عرضنا في هذا الفصل خطوات بناء نموذج تعلّم لعنقدة مجموعة من البيانات مع آليات اختيار عدد العناقيد المناسب للمسألة.

> يُمكن تجربة المثال كاملًا من موقع Google Colab من هذا الرابط ولا تنس الاطلاع على ملف بيانات التدريب والشيفرة البرمجية.

9. تصنيف الشخصيات بالاعتماد على تغريداتهم العربية

تُعدّ مسألة تصنيف الشخصيات بالاعتماد على تغريداتهم على وسائل التواصل الاجتماعي من المسائل المهمة مثلًا يُمكن أن يكون من المفيد تحديد الأشخاص الذين يكتبون تحليلات مهمة في مجالات الأعمال أو الاقتصاد أو الرياضة وغيرها مما يتيح للآخرين معرفة تقييمات أو آراء الأشخاص المغردين واتخاذ الإجراء المناسب.

نعرض في هذا الفصل استخدام تقنيات التعلم العميق في تصنيف الشخصيات المغردة بنصوص مكتوبة باللغة العربية وباللهجة السعودية بمعنى أن اللغة المستخدمة ليست بالضرورة اللغة العربية الفصحى بل يُمكن أن تدخل فيها ألفاظ عامية يستخدمها المغردون عادةً.

لتصنيف شخص ما، سنلجأ أولًا إلى تصنيف تغريداته (أي إسناد كل تغريده إلى صف معين)، ومن ثم تصنيفه بناء على نتائج تصنيف تغريداته (الصفوف ذات التواتر الكبر).

9.1 بيانات التدريب

تحوي مجموعة البيانات المتوفرة حوالي 26500 تغريدة لمجموعة متنوعة من المغردين المشهورين في المملكة العربية السعودية (يمكنك تنزيلها من هذا الرابط). جُمّعت هذه التغريدات عن طريق مجموعة من الطلاب الجامعيين وذلك من موقع تويتر Twitter. بالطبع، لا يوجد تصنيف متوفر لهذه التغريدات على تويتر. تجدها موجودة هي وبقية ملفات هذا الفصل في الملف المضغوط المرفق بالرابط.

9.2 تصنيف بيانات التدريب

يتطلب استخدام خوارزميات تعلم الآلة (خوارزميات تصنيف التغريدات في حالتنا) توفر بيانات للتدريب أي مجموعة من التغريدات مُصنفّة مُسبقًا إلى مجموعة من الصفوف، ويُمكن، كما في حالتنا، اللجوء إلى الطرق

اليدوية أي الطلب من مجموعة من الأشخاص قراءة النصوص وتصنيفها وهو حل يصلح في حال كان عدد النصوص صغيرًا نسبيًا، ويتميز هذا الحل بالدقة العالية لأن الأشخاص تُدرك، بشكل عام، معاني النصوص من خلال خبرتها اللغوية المُكتسبة وتُصنّف النصوص بشكل صحيح غالبًا.

طلبنا من الطلاب المشاركين في مشروعنا تصنيف كل تغريدة إلى واحدة من المواضيع (الصفوف) التالية:

الصف بالعربي	English
الرياضة	Sports
الأعمال	Business
التكنولوجيا	Technology
المجتمع	Social
السياسة	Politics
الأخبار	News
الاقتصاد	Economy

نُعطي فيما يلي أمثلة عن هذه التغريدات (بعد تصنيفها):

"الحمد لله مباراة الدوري وهدف غالي يشابه هدف القناص مباراة الفتح الرائد الهلال" (الرياضة).

"قريبا جدا... تتابع السوق السعودي وشركاته والأمريكي وشركاته والعملات والكربتو النفط والذهب بمنصة واحدة مباشرة وتقدر ت..." (الأعمال).

"فايروس خطير جدا يستهدف مستخدمي نظام الاندرويد قادر يسرق بيانات حساسه الصور وتسجيل صوت المستخدم والوصول لمعلوماتهم وسرقة..." (التكنولوجيا)

"أالسلام عليكم ورحمة الله وبركاتة اللهم فرج يفرج همه ويبارك ماله ويفتح أبواب الخيروال..." (المجتمع)

"مبادرون لأجل فلسطين.. تكريم كويتي لرافضي التطبيع في الفعاليات الرياضية والأدبية" (السياسة)

"الخط العربي ومخاوف التقنية في الزمن الرقمي.. تجربة عالمية أم خروج عن تقاليد الفن العريق؟" (الأخبار)

"معلومة فنية (٢)للشراء المضاربي ١- اقفال السوق ٥ دقايق اذا كانت شمعة ايجابية للحصول ربح مضاربي يوم ٢-عند..." (الاقتصاد).

نعرض في هذا الفصل كيفية بناء مُصنّف حاسوبي آلي يُصنّف أي تغريدة عربية إلى أحد الصفوف السبعة السابقة ومن ثم تصنيف الشخص المغرد لواحد أو أكثر من هذه الصفوف (الأكثر تواترًا).

9.3 المعالجة الأولية للنصوص

تتميز التغريدات، بشكل عام، باحتوائها على مجموعة من الأمور غير المهمة في مسألة التصنيف مثل الروابط Links والوسوم Hashtags والرموز التعبيرية Emojis وغيرها.

يُبين الشكل التالي مثالًا عن التغريدات المُجمّعة:

	A	
1	**tweet**	**topic**
2	@sssdddnnn333 @MohammedAlDeaye حارس حتى عنده ضعف نظر مدري كيف يحرص. عموما حاله حال كل لاعبي هلال العب حتى تتعب. حال... https://t.co/v9AhSaCk8e	Sports
3	RT @Alhilal_FC: ⏰ بداية الشوط الثاني #١ الهلال x ٠ الاتحاد #الهلال_الاتحاد ⚽ @salem_d29 https://t.co/wpwiLcdVjo	Sports
4	RT @fahadaljehani: الكثير من المتعة كانت حاضره بالملعب صنعت الفرق في المبادرة والتسديد وال@salem_d القيمة الفنية العاليه ل #سالم_الدوسري...29	Sports
5	RT @aboaljorya: @AhmedAllshehri عذيب 👀 السعر الحالي ٥١,٩٠ 😳	Business
6	RT @salem_d29: الحمدلله ، شعوري لا يوصف ، ولن انسى فضل جمهور الهلال علي شخصيا وعلى اخواني اللاعبين ووقفتهم ودعمهم لنا في الحلوه والمره ، وا...	Sports
7	أبل أبدعت بهذي الميزة الأمر مستحيل ومجرد خيال سنة ايباد ماك ايماك	Technology

تهدف المعالجة الأولية إلى الحصول على الكلمات المهمة فقط من النصوص وذلك عن طريق تنفيذ بعض العمليات اللغوية عليها.

لتكن لدينا مثلًا الجملة التالية:

"أنا أحب الذهاب إلى الحديقة ، كل يوم 9 صباحاً، مع رفاقي هؤلاء! @toto "

سنقوم بتنفيذ العمليات التالية:

1. حذف الروابط والوسوم والرموز التعبيرية: يكون ناتج الجملة السابقة:

"أنا أحب الذهاب إلى الحديقة، كل يوم 9 صباحاً، مع رفاقي هؤلاء!"

2. حذف إشارات الترقيم المختلفة كالفواصل وإشارات الاستفهام وغيرها: يكون ناتج الجملة السابقة:

"أنا أحب الذهاب إلى الحديقة كل يوم 9 صباحاً مع رفاقي هؤلاء"

3. حذف الأرقام الواردة في النص: يكون ناتج الجملة السابقة:

"أنا أحب الذهاب إلى الحديقة كل يوم صباحاً مع رفاقي هؤلاء"

4. حذف كلمات التوقف stop words وهي الكلمات التي تتكرر كثيرًا في النصوص ولا تؤثر في معانيها كأحرف الجر (من، إلى، ...) والضمائر (أنا، هو، ...) وغيرها فيكون ناتج الجملة السابقة:

" أحب الذهاب الحديقة يوم صباحاً رفاقي "

5. تجذيع الكلمات stemming أي إرجاع الكلمات المتشابهة إلى كلمة واحدة (جذع) مما يُساهم في إنقاص عدد الكلمات الكلية المختلفة في النصوص، ومطابقة الكلمات المتشابهة مع بعضها البعض. مثلًا: يكون للكلمات الأربع: (رائع، رايع، رائعون، رائعين) نفس الجذع المشترك: (رايع) فيكون ناتج الجملة السابقة:

" احب ذهاب حديق يوم صباح رفاق "

9.4 إعداد المشروع

يحتاج تنفذ شيفرات هذا الفصل بيئةً برمجيةً للغة بايثون الإصدار 3.8. ويجب أن تتضمن هذه البيئة البرمجية مدير الحِزم pip لتثبيت الحِزم، ومُنشئ البيئات الافتراضية venv لإنشاء بيئاتٍ افتراضيةٍ.

نستخدم محرر الشيفرات البرمجية Jupyter Notebooks، وهو مفيد جدًا لتجربة وتشغيل الأمثلة الخاصة بتَعَلّم الآلة بطريقةٍ تفاعليةٍ، حيث نستطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسرعة، مما يُسهّل علينا اختبار الشيفرات البرمجية وتصحيحها.

نحتاج أولًا لتثبيت بعض التبعيات، وذلك لإنشاء مساحة عمل للاحتفاظ بملفاتنا قبل أن نتمكن من تطوير برنامجنا.

نُنشئ مجلدًا جديدًا خاصًا بمشروعنا وندخل إليه هكذا:

```
mkdir arc
cd arc
```

نُنفذّ الأمر التالي لإنشاء البيئة الافتراضية:

```
python -m venv arc
```

ومن ثم الأمر التالي في لينكس Linux لتنشيط البيئة الافتراضية:

```
source arc/bin/activate
```

أما في ويندوز Windows، فيكون أمر التنشيط:

```
"arc/Scripts/activate.bat"
```

نستخدم إصداراتٍ محددةٍ من المكتبات اللازمة، من خلال إنشـاء ملـف `requirements.txt` في مجلـد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

نفتح الملف `requirements.txt` في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبـات التي نريدها وإصداراتها:

```
absl-py==1.0.0
asttokens==2.0.5
astunparse==1.6.3
backcall==0.2.0
cachetools==5.1.0
certifi==2021.10.8
charset-normalizer==2.0.12
click==8.1.3
colorama==0.4.4
cycler==0.11.0
debugpy==1.6.0
decorator==5.1.1
entrypoints==0.4
executing==0.8.3
flatbuffers==1.12
fonttools==4.33.3
gast==0.4.0
google-auth==2.6.6
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.46.1
h5py==3.6.0
idna==3.3
imbalanced-learn==0.9.1
imblearn==0.0
importlib-metadata==4.11.3
ipykernel==6.13.0
ipython==8.3.0
jedi==0.18.1
joblib==1.1.0
jupyter-client==7.3.1
```

```
jupyter-core==4.10.0
keras==2.6.0
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
kiwisolver==1.4.2
libclang==14.0.1
Markdown==3.3.7
matplotlib==3.5.2
matplotlib-inline==0.1.3
nest-asyncio==1.5.5
nltk==3.7
numpy==1.22.3
oauthlib==3.2.0
opt-einsum==3.3.0
packaging==21.3
pandas==1.4.2
parso==0.8.3
pickleshare==0.7.5
Pillow==9.1.1
prompt-toolkit==3.0.29
protobuf==3.20.1
psutil==5.9.0
pure-eval==0.2.2
PyArabic==0.6.14
pyasn1==0.4.8
pyasn1-modules==0.2.8
Pygments==2.12.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
#pywin32==304
pyzmq==22.3.0
regex==2022.4.24
requests==2.27.1
requests-oauthlib==1.3.1
rsa==4.8
```

```
scikit-learn==1.1.0
scipy==1.8.0
seaborn==0.11.2
six==1.16.0
sklearn==0.0
snowballstemmer==2.2.0
stack-data==0.2.0
tensorboard==2.9.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.9.0
tensorflow-estimator==2.9.0
tensorflow-io-gcs-filesystem==0.26.0
termcolor==1.1.0
threadpoolctl==3.1.0
tornado==6.1
tqdm==4.64.0
traitlets==5.2.1.post0
typing-extensions==4.2.0
urllib3==1.26.9
wcwidth==0.2.5
Werkzeug==2.1.2
wrapt==1.14.1
zipp==3.8.0
```

نحفظ التغييرات التي طرأت على الملف ونخرج من محرر النصوص، ثم نُثَبت هذه المكتبات بالأمر التالي:

```
(arc) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، نُصبح جاهزين لبدء العمل على مشروعنا.

9.5 كتابة الشيفرة البرمجية

نُشغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت هكذا:

```
(arc) $ jupyter notebook
```

ثم نُنشئ ملفًا جديدًا في داخل المحرر ونُسمّه باسم ac مثلًا.

9.5.1 تحميل البيانات

نبـدأ أولًا بتحميـل التغريـدات من الملـف tweets.csv ضـمن إطـار من البيانـات DataFrame من مكتبـة Pandas ومن ثم عرض بعضها:

```
import pandas as pd
# قراءة التغريدات وتحميلها ضمن إطار من البيانات
tweets = pd.read_csv('tweets.csv',encoding = "utf-8")
# إظهار الجزء الأعلى من إطار البيانات
tweets.head()
```

يظهر لنا أوائل التغريدات:

	tweet	topic
0	@sssdddnnn333 @MohammedAlDeaye ض...حارس حتى عنده	Sports
1	RT @Alhilal_FC: ⏰ بداية الشوط الثاني\n#الهلال ...	Sports
2	RT @fahadaljehani: الكثير من المتعة كانت حاضره...	Sports
3	RT @aboaljorya: @AhmedAllshehri عذيب 👀\nالسعر ...	Business
4	RT @salem_d29: ولن ، شعوري لا يوصف ، الحمدلله ...	Sports

يُمكن استخدام خاصية الشكل shape لمعرفة عدد الصفوف والأعمدة للبيانات المُحمّلة:

```
print('Data size:', tweets.shape)
```

يكون الناتج:

```
Data size: (25334, 4)
```

9.5.2 المعالجة الأولية للنصوص

نستخدم فيما يلي بعض الخـدمات الـتي توفرهـا المكتبـة nltk لمعالجـة اللغـات الطبيعيـة كتوفـير قائمـة كلمات التوقف باللغة العربيـة (حـوالي 700 كلمـة) واسـتخراج الوحـدات tokens من النصـوص. كمـا نسـتخدم مجذع الكلمات العربية من مكتبة snowballstemmer.

كمـا نسـتخدم التعـابير النمطيـة regular expressions للتعـرف على الرمـوز التعبيريـة والوسـوم والروابط وحذفها.

```
# مكتبة السلاسل النصية
import string
# مكتبة التعابير النظامية
import re
# مكتبة معالجة اللغات الطبيعية
import nltk
#nltk.download('punkt')
#nltk.download('stopwords')
# مكتبة كلمات التوقف
from nltk.corpus import stopwords
# مكتبة استخراج الوحدات
from nltk.tokenize import word_tokenize
# مكتبة المجذع العربي
from snowballstemmer import stemmer
ar_stemmer = stemmer("arabic")

# دالة حذف المحارف غير اللازمة
def remove_chars(text, del_chars):
    translator = str.maketrans('', '', del_chars)
    return text.translate(translator)

# دالة حذف المحارف المكررة
def remove_repeating_char(text):
    return re.sub(r'(.)\1{2,}', r'\1', text)

# دالة تنظيف التغريدات
def clean_tweet(tweet):
    stop_words = stopwords.words('arabic')
    # محارف الرموز التعبيرية
    emoj = re.compile("["
            u"\U0001F600-\U0001F64F"
            u"\U0001F300-\U0001F5FF"
            u"\U0001F680-\U0001F6FF"
            u"\U0001F1E0-\U0001F1FF"
            u"\U00002500-\U00002BEF"
            u"\U00002702-\U000027B0"
```

```
            u"\U00002702-\U000027B0"
            u"\U000024C2-\U0001F251"
            u"\U0001f926-\U0001f937"
            u"\U00010000-\U0010ffff"
            u"\u2640-\u2642"
            u"\u2600-\u2B55"
            u"\u200d"
            u"\u23cf"
            u"\u23e9"
            u"\u231a"
            u"\ufe0f"
            u"\u3030"
                        "]+", re.UNICODE)
    tweet = str(tweet)
    # حذف @ وما يتبعها
    tweet = re.sub("@[^\s]+","",tweet)
    tweet = re.sub("RT","",tweet)
    # حذف الروابط
    tweet = re.sub(r"(?:\@|http?\://|https?\://|www)\S+", "", tweet)
    # حذف الرموز التعبيرية
    tweet =  re.sub(emoj, '', tweet)
    # حذف كلمات التوقف
    tweet = ' '.join(word for word in tweet.split() if word not in
stop_words)
    # حذف الإشارات #
    tweet = tweet.replace("#", "").replace("_", " ")
    # حذف الأرقام
    tweet = re.sub(r'[0-9]+', '', tweet)
    # حذف المحارف غير اللازمة
    # علامات الترقيم العربية
    arabic_punctuations = '''`÷×؛–“…”!|+¦~{}',.؟":/؛][%^&*()_<>؛'''
    # علامات الترقيم الإنجليزية
    english_punctuations = string.punctuation
    # دمج علامات الترقيم العربية والإنكليزية
    punctuations_list = arabic_punctuations + english_punctuations
    tweet = remove_chars(tweet, punctuations_list)
    # حذف المحارف المكررة
```

```
    tweet = remove_repeating_char(tweet)
    # استبدال الأسطر الجديدة بفراغات
    tweet = tweet.replace('\n', ' ')
    # حذف الفراغات الزائدة من اليمين واليسار
    tweet = tweet.strip(' ')
    return tweet

# دالة تقسيم النص إلى مجموعة من الوحدات
def tokenizingText(text):
    tokens_list = word_tokenize(text)
    return tokens_list

# دالة حذف كلمات التوقف
def filteringText(tokens_list):
    # قائمة كلمات التوقف العربية
    listStopwords = set(stopwords.words('arabic'))
    filtered = []
    for txt in tokens_list:
        if txt not in listStopwords:
            filtered.append(txt)
    tokens_list = filtered
    return tokens_list

# دالة التجذيع
def stemmingText(tokens_list):
    tokens_list = [ar_stemmer.stemWord(word) for word in tokens_list]
    return tokens_list

# دالة دمج قائمة من الكلمات في جملة
def toSentence(words_list):
    sentence = ' '.join(word for word in words_list)
    return sentence
```

شرح الدوال السابقة:

- الدالة `clean_tweet` تحذف الرموز التعبيرية والروابط والوسوم والأرقام وعلامات الترقيم العربية والإنكليزية من النص وذلك باستخدام التعابير النظامية الموافقة.

- الدالة `remove_repeating_char` تحذف المحارف المكررة والتي قد يستخدمها كاتب التغريدة.
- الدالة `tokenizingText` تعمل على تجزئة النص إلى قائمة من الوحدات tokens.
- الدالة `filteringText` تحذف كلمات التوقف من قائمة الوحدات.
- الدالة `stemmingText` تعمل على تجذيع كلمات قائمة الوحدات المتبقية.

يُبين المثال التالي نتيجة استدعاء كل دالة من الدوال السابقة:

```
# مثال
text= "أنا أحب الذهاب إلى الحديقة ، كل يوم 9 صباحاً، مع رفاقي هؤلاء! @toto  "
print(text)
text=clean_tweet(text)
print(text)
tokens_list=tokenizingText(text)
print(tokens_list)
tokens_list=filteringText(tokens_list)
print(tokens_list)
tokens_list=stemmingText(tokens_list)
print(tokens_list)
```

يكون ناتج التنفيذ:

```
أنا أحب الذهاب إلى الحديقة ، كل يوم 9 صباحاً، مع رفاقي هؤلاء! @toto
أحب الذهاب الحديقة يوم صباحاً رفاقي هؤلاء
['أحب', 'الذهاب', 'الحديقة', 'يوم', 'صباحاً', 'رفاقي', 'هؤلاء']
['أحب', 'الذهاب', 'الحديقة', 'يوم', 'صباحاً', 'رفاقي']
['احب', 'ذهاب', 'حديق', 'يوم', 'صباح', 'رفاق']
```

تعرض الشيفرة التالية التصريح عن الدالة `process_tweet` والتي تقوم أولًا بتنظيف التغريدات وذلك باستدعاء الدالة `clean_tweet`، ومن ثم تحويل التغريدات إلى وحدات وذلك باستدعاء الدالة `tokenizingText`، وأخيرًا تجذيع الوحدات عن طريق الدالة `stemmingText`.

ومن ثم تنفيذ دالة المعالجة الأولية على جميع التغريدات:

```
# دالة معالجة التغريدات
def process_tweet(tweet):
    # تنظيف التغريدة
    tweet=clean_tweet(tweet)
```

```
    # التحويل إلى وحدات
    tweet=tokenizingText(tweet)
    # التجذيع
    tweet=stemmingText(tweet)
    return tweet

# المعالجة الأولية للتغريدات
tweets['tweet'] = tweets['tweet'].apply(process_tweet)
```

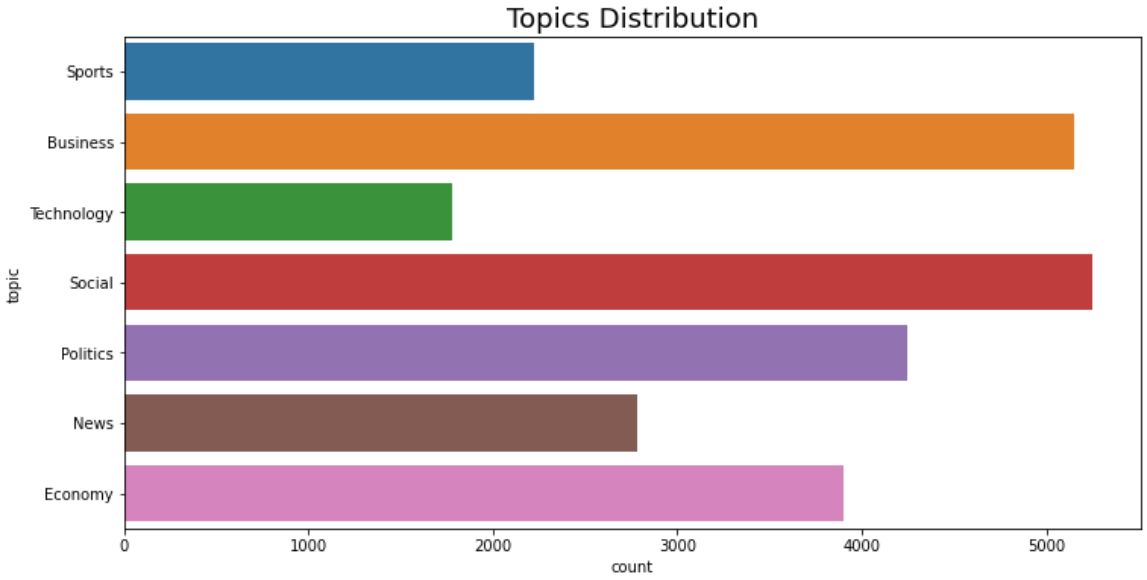
9.6 موازنة الصفوف

تتطلب معظم خوارزميات التصنيف توفر بيانات للتدريب وبصفوف متوازنة، بمعنى أن يكون عدد أسطر البيانات من كل صف متساوية تقريبًا تجنبًا لانحياز المُصنف إلى الصف ذي العدد الأكبر من الأسطر في بيانات التدريب، ومن المفيد إذًا أولًا معاينة توزع أعداد الصفوف.

نستخدم في الشيفرة البرمجية التالية مكتبات الرسم اللازمة لرسم أشرطة تُمثّل أعداد الصفوف:

```
# مكتبات الرسم
import matplotlib.pyplot as plt
import seaborn as sns
# حجم الرسم
plt.figure(figsize=(12, 6))
# رسم عدد كل صف
sns.countplot(data=tweets, y='topic');
plt.title('Topics Distribution', fontsize=18)
plt.show()
```

يُبين الشكل الناتج عدم توازن الصفوف على الإطلاق:



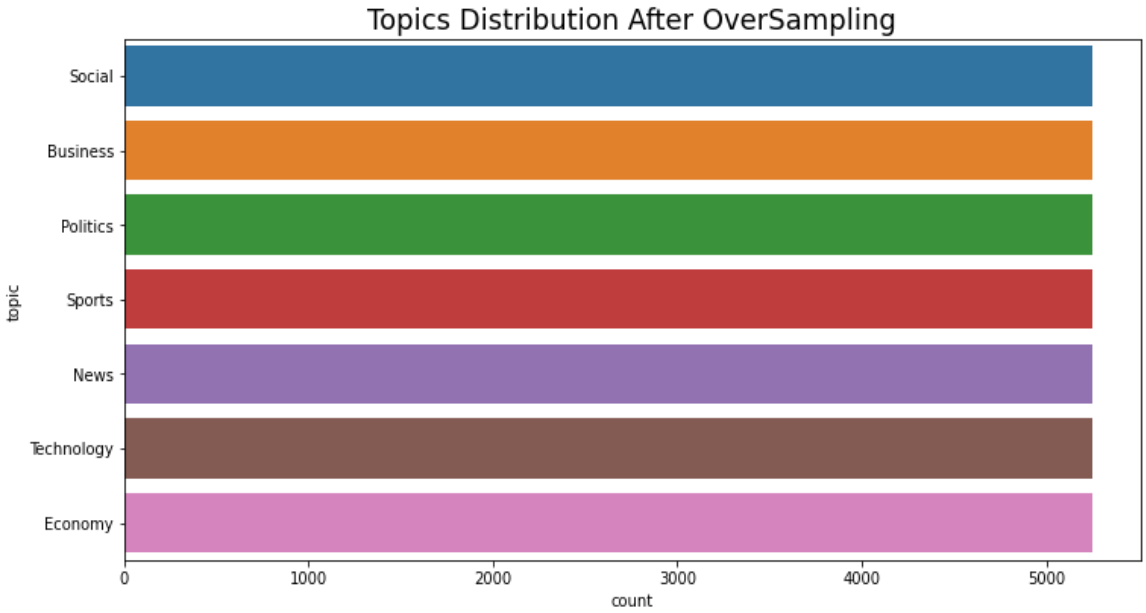
نستخدم في الشيفرة التالية الصنف RandomOverSampler من المكتبة imblearn.over_sampling والذي يقوم باختيار أسطر عشوائية من الصفوف القليلة العدد ويُكررها في مجموعة البيانات.

```
# استيراد مكتبة الموازنة
from imblearn.over_sampling import RandomOverSampler
# إنشاء كائن من الصنف
oversample = RandomOverSampler()
# توليد سطر عشوائي
tweets = tweets.sample(frac=1)
# توليد الأسطر الجديدة
tweets, Y = oversample.fit_resample(tweets, tweets.topic)
```

نعيد رسم أعداد الصفوف بعد الموازنة:

```
# إعادة رسم أعداد الصفوف
# بعد الموازنة
plt.figure(figsize=(12, 6))
sns.countplot(data=tweets, y='topic');
plt.title('Topics Distribution After OverSampling', fontsize=18)
plt.show()
```

يُبين الشكل الناتج توازن الصفوف بشكل كامل:



9.7 ترميز الصفوف

لا تقبل بنى تعلم الآلة النصوص كمدخلات لها، تقوم الشيفرة التاليـة باسـتخدام الصـنف `LabelEncoder` من المكتبة `sklearn.preprocessing` لترميز الصفوف باستخدام الأرقام.

نطبع في النهاية أسماء الصفوف والترميز المقابل لها:

```
from sklearn.preprocessing import LabelEncoder
# ترميز الصفوف
le_topics = LabelEncoder()
tweets['topic'] = tweets[['topic']].apply(le_topics.fit_transform)
classes = le_topics.classes_ # الصفوف
n_classes = len(classes) # عدد الصفوف
print("No. of classes:", n_classes)
print("Classes:", classes)
print("Coding: ", le_topics.transform(classes))
```

يكون الناتج:

```
No. of classes: 7

Classes: ['Business' 'Economy' 'News' 'Politics' 'Social' 'Sports'
'Technology']

Coding:  [0 1 2 3 4 5 6]
```

9.8 تحويل النصوص إلى أشعة رقمية

لا تقبل بنى تعلم الآلة النصوص كمدخلات لها، بل تحتاج إلى أشعة رقمية كمدخلات.

نستخدم الشيفرة التالية لتحويل الشعاع النصي لكل تغريدة `tweet_preprocessed` إلى شعاع رقمي:

```
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
# نركيب جمل التغريدات من المفردات المعالجة
sentences = tweets['tweet'].apply(toSentence)
print(sentences[6])
# عدد الكلمات الأعظمي ذات التواتر الأكبر
# التي ستُستخدم
max_words = 5000
# الطول الأعظمي لشعاع الأرقام
max_len = 50
#    التصريح عن المجزئ
#    مع تحديد عدد الكلمات التي ستبقى
#  بالاعتماد على تواترها
tokenizer = Tokenizer(num_words=max_words )
# ملائمة المجزئ لنصوص التغريدات
tokenizer.fit_on_texts(sentences)
# تحويل النص إلى قائمة من الأرقام
S = tokenizer.texts_to_sequences(sentences)
print(S[0])
# توحيد أطوال الأشعة
X = pad_sequences(S, maxlen=max_len)
print(X[0])
X.shape
```

نجد من الشيفرة أن المتغير `max_words` يُحدّد عدد الكلمات الأعظمي التي سيتم الاحتفاظ بها حيث يُحسب تواتر كل كلمة في كل النصوص ومن ثم تُرتب حسب تواترها (المرتبة الأولى للكلمة ذات التواتر الأكبر). ستُهمل الكلمات ذات المرتبة أكبر من `max_words`، و يُحدّد المتغير `max_len` طول الشعاع الرقمي النهائي فإذا كان موافقًا لنص أقل من `max_len` تُضاف أصفار للشعاع حتى يُصبح طوله مساويًا إلى `max_len`. أما إذا كان طوله أكبر يُقتطع جزءًا منه ليُصبح طوله مساويًا إلى `max_len`.

بينما تقوم الدالة `fit_on_texts` بملائمة المُجزء tokenizer لنصوص جمل التغريدات `sentences.values` أي حساب تواتر الكلمات والاحتفاظ بالكلمات ذات التواتر أكبر أو يساوي `max_words`.

نطبع في الشيفرة السابقة، بهدف التوضيح، ناتج كل مرحلة.

اخترنا مثلًا شعاع التغريدة 6 بعد المعالجة:

دخل يوم يوافق يوم نوم عالم وفي يتم تذكير اهم نوم فوايد عظيم لجسم انس تبخل نفس

تكون نتيجة تحويل الشعاع السابق النصي إلى شعاع من الأرقام:

```
[1839, 1375, 2751, 315, 975, 1420, 1839, 3945, 436, 794, 919, 445,
1290, 312, 258]
```

وبعد عملية توحيد الطول يكون الشعاع الرقمي النهائي الناتج:

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0 1839 1375 2751  315  975 1420 1839
436  794  919  445 1290  312  258]
```

9.9 تجهيز دخل وخرج الشبكة العصبية

تعرض الشيفرة التالية حساب شعاع الخرج أولًا وذلك بإسناد العمود `topic` من إطار البيانات، والذي يحوي الترميز الرقمي للصفوف إلى المتغير `y`.

نستخدم الدالة `train_test_split` لتقسيم البيانات المتاحة إلى 80% منها لعملية التدريب و20% لعملية الاختبار وحساب مقاييس الأداء:

```
# توليد شعاع الخرج
y = tweets['topic']
# مكتبة تقسيم البيانات إلى تدريب واختبار
from sklearn.model_selection import train_test_split
# تقسيم البيانات إلى تدريب واختبار
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.2, random_state = 0)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

نطبع في الشيفرة السابقة حجوم أشعة الدخل والخرج للتدريب وللاختبار:

```
(29388, 50) (29388,)
(7348, 50) (7348,)
```

9.10 نموذج الشبكة العصبية المتعلم

تُعدّ المكتبة Keras من أهم مكتبات بايثون التي توفر بناء شبكات عصبية لمسائل التعلم الآلي.

تعرض الشيفرة التالية التصريح عن دالة بناء نموذج التعلّم `create_model` مـع إعطـاء جميـع المعـاملات المترفعة قيمًا ابتدائية:

```
# تضمين النموذج التسلسلي
from keras.models import Sequential
# تضمين  الطبقات اللازمة
from keras.layers import Embedding, Dense, LSTM
# دوال التحسين
from tensorflow.keras.optimizers import Adam, RMSprop


# التصريح عن دالة إنشاء نموذج التعلم
# مع إعطاء قيم أولية للمعاملات المترفعة
def create_model(embed_dim = 32, hidden_unit = 16, dropout_rate = 0.2,
optimizers = RMSprop, learning_rate = 0.001):
    # التصريح عن نموذج تسلسلي
    model = Sequential()
    # طبقة التضمين
    model.add(Embedding(input_dim = max_words, output_dim = embed_dim,
input_length = MAX_LENGTH))
    # LSTM
    model.add(LSTM(units = hidden_unit ,dropout=dropout_rate))
    # الطبقة الأخيرة
    model.add(Dense(units = len(classes), activation = 'softmax'))
    # بناء النموذج
    model.compile(loss = 'sparse_categorical_crossentropy', optimizer
= optimizers(learning_rate = learning_rate), metrics = ['accuracy'])
    # طباعة ملخص النموذج
    print(model.summary())


    return model
```

نستخدم من أجل مسألتنا نموذج شبكة عصـبية تسلسـلي يتـألف من ثلاث طبقـات هي: طبقـة التضـمين Embedding وطبقة LSTM (شبكة ذات ذاكرة طويلة المدى) والطبقة الكثيفة Dense.

9.10.1 الطبقة الأولى: طبقة التضمين Embedding

نستخدم هذه الطبقة لتوليد ترميز مكثف للكلمات word encoding dense مما يُساهم في تحسين عملية التعلم. نطلب تحويل الشعاع الذي طوله `input_length` (في حالتنا 50) والذي يحوي قيم ضمن المجال `input_dim` (من 1 إلى 5000 في مثالنا) إلى شعاع من القيم ضمن المجال `output_dim` مثلًا 32 قيمة.

9.10.2 الطبقة الثانية: شبكة ذات ذاكرة طويلة المدى Long Short-Term Memory

يُحدّد المعامل المترفع `units` عدد الوحدات المخفية لهذه الطبقة. يُساهم المعامل `dropout` في معايرة الشبكة خلال التدريب حيث يقوم بإيقاف تشغيل الوحدات المخفية بشكل عشوائي أثناء التدريب، وبهذه الطريقة لا تعتمد الشبكة بنسبة 100% على جميع الخلايا العصبية الخاصة بها. وبدلاً من ذلك، تُجبر نفسها على العثور على أنماط أكثر أهمية في البيانات من أجل زيادة المقياس الذي تحاول تحسينه (الدقة مثلًا).

9.10.3 الطبقة الثالثة: الكثيفة Dense

يُحدّد المعامل `units` حجم الخرج لهذه الطبقة (7 في حالتنا: عدد الصفوف) ويُبين الشكل التالي ملخص النموذج:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 50, 32)            160000
_________________________________________________________________
lstm_5 (LSTM)                (None, 64)                24832
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 455
=================================================================
Total params: 185,287
Trainable params: 185,287
Non-trainable params: 0
_________________________________________________________________
```

9.11 معايرة المعاملات الفائقة للوصول لنموذج أمثلي

يمكن الوصول لنموذج تعلم أمثلي بمعايرة معاملاته الفائقة وفق معطيات المشروع. لنُبين أولًا الفرق بين المعاملات الفائقة لنموذج والمعاملات الأخرى له:

- المعاملات الفائقة hyperparameters: هي إعدادات خوارزمية التعلّم قبل التدريب (والتي وضعها مصممو الخوارزمية).
- المعاملات parameters: هي المعاملات التي يتعلّمها النموذج أثناء التدريب مثل أوزان الشبكة العصبية.

تؤثر عملية معايرة المعاملات الفائقة على أداء النموذج لاسيما لجهة التوزان المطلوب بين مشكلة قلة التخصيص underfitting ومشكلة فرط التخصيص overfitting واللتان تؤديان إلى نموذج غير قادر على تعميم أمثلة التدريب وبالتالي لن يتمكن من التصنيف مع معطيات جديدة (يُمكن العودة لمقال تعلم الآلة: التحديات الرئيسية وكيفية التوسع في المجال من أكاديمية حسوب للمزيد من التفصيل حول هاتين المشكلتين).

تظهر مشكلة قلة التخصيص عندما لا يكون للنموذج درجات حرية كافية ليتعلّم الربط بين الميزات والهدف، وبالتالي يكون له انحياز كبير Bias-variance tradeoff نحو قيم معينة للهدف. يُمكن تصحيح قلة التخصيص بجعل النموذج أكثر تعقيدًا. أما مشكلة فرط التخصيص فتظهر عندما يقوم النموذج بتخزين بيانات التدريب فيكون له بالتالي تباين كبير والذي يُمكن تصحيحه بالحد من تعقيد النموذج باستخدام التسوية regularization.

تكمن المشكلة في معايرة المعاملات الفائقة بأن قيمها المثلى تختلف من مسألة لأخرى! وبالتالي، فإن الطريقة الوحيدة للوصول لهذه القيم المثلى هي تجريب قيم مختلفة مع كل مجموعة بيانات تدريب جديدة.

يوفر Scikit-Learn العديد من الطرق لتقويم المعاملات الفائقة وبالتالي سنعتمد في مشروعنا عليها دون أن نُعقّد الأمور أكثر.

يُمكن العودة إلى فصل تحليل المشاعر في اللغة العربية فقرة "البحث الشبكي مع التقييم المتقاطع" لشرح طريقة حسابنا لأفضل القيم للمعاملات المترفعة.

9.11.1 بناء نموذج التعلم النهائي

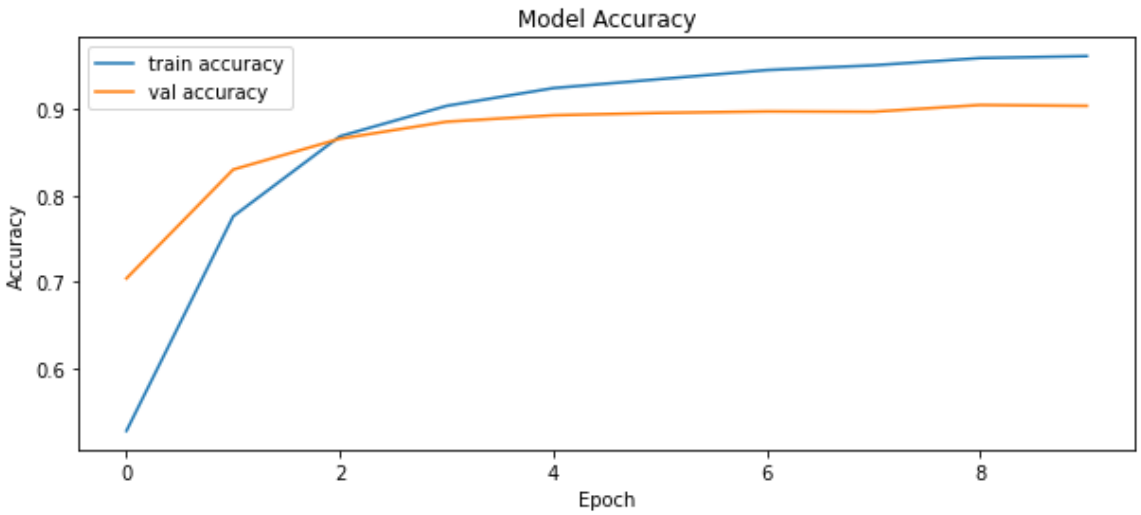
نستخدم الدالة `KerasClassifier` من `scikit` لبناء المُصنف مع الدالة السابقة `create_model`:

```
# مكتبة التصنيف
from keras.wrappers.scikit_learn import KerasClassifier

# إنشاء النموذج مع قيم المعاملات المترفعة الأمثلية
model = KerasClassifier(build_fn = create_model,
                        # معاملات النموذج
                        dropout_rate = 0.2,
                        embed_dim = 32,
                        hidden_unit = 64,
                        optimizers = Adam,
                        learning_rate = 0.001,
                        # معاملات التدريب
                        epochs=10,
                        batch_size=256,
                        # نسبة بيانات التقييم
                        validation_split = 0.1)
# ملائمة النموذج مع بيانات التدريب
model_prediction = model.fit(X_train, y_train)
```

يُمكن الآن رسم منحني الدقة accuracy لكل من بيانات التدريب والتقييم (لاحظ أننا في الشيفرة السـابقة احتفاظنا بـ 10% من بيانات التدريب للتقييم):

```
# معاينة دقة النموذج
# التدريب والتقييم
fig, ax = plt.subplots(figsize = (10, 4))
ax.plot(model_prediction.history['accuracy'], label = 'train
accuracy')
ax.plot(model_prediction.history['val_accuracy'], label = 'val
accuracy')
ax.set_title('Model Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylabel('Accuracy')
ax.legend(loc = 'upper left')
plt.show()
```

يكون للمنحي الشكل التالي:



9.11.2 حساب مقاييس الأداء

يُمكن الآن حساب مقاييس الأداء المعروفة في مسائل التصـنيف (الصـحة Accuracy، الدقـة Precision، الاستذكار Recall، المقياس F1) للنموذج المتعلم باستخدام الشيفرة التالية:

```
# مقاييس الأداء

# مقياس الصحة
from sklearn.metrics import accuracy_score
# مقياس الدقة
from sklearn.metrics import precision_score
# مقياس الاستذكار
from sklearn.metrics import recall_score
# f1
from sklearn.metrics import f1_score
# مصفوفة الارتباك
from sklearn.metrics import confusion_matrix
# تصنيف بيانات الاختبار
y_pred = model.predict(X_test)
# حساب مقاييس الأداء
accuracy = accuracy_score(y_test, y_pred)
precision=precision_score(y_test, y_pred , average='weighted')
recall= recall_score(y_test, y_pred, zero_division=1,
average='weighted')
f1= f1_score(y_test, y_pred, zero_division=1,  average='weighted')
```

```
print('Model Accuracy on Test Data:', accuracy*100)
print('Model Precision on Test Data:', precision*100)
print('Model Recall on Test Data:', recall*100)
print('Model F1 on Test Data:', f1*100)

confusion_matrix(y_test, y_pred)
```

تكون النتائج:

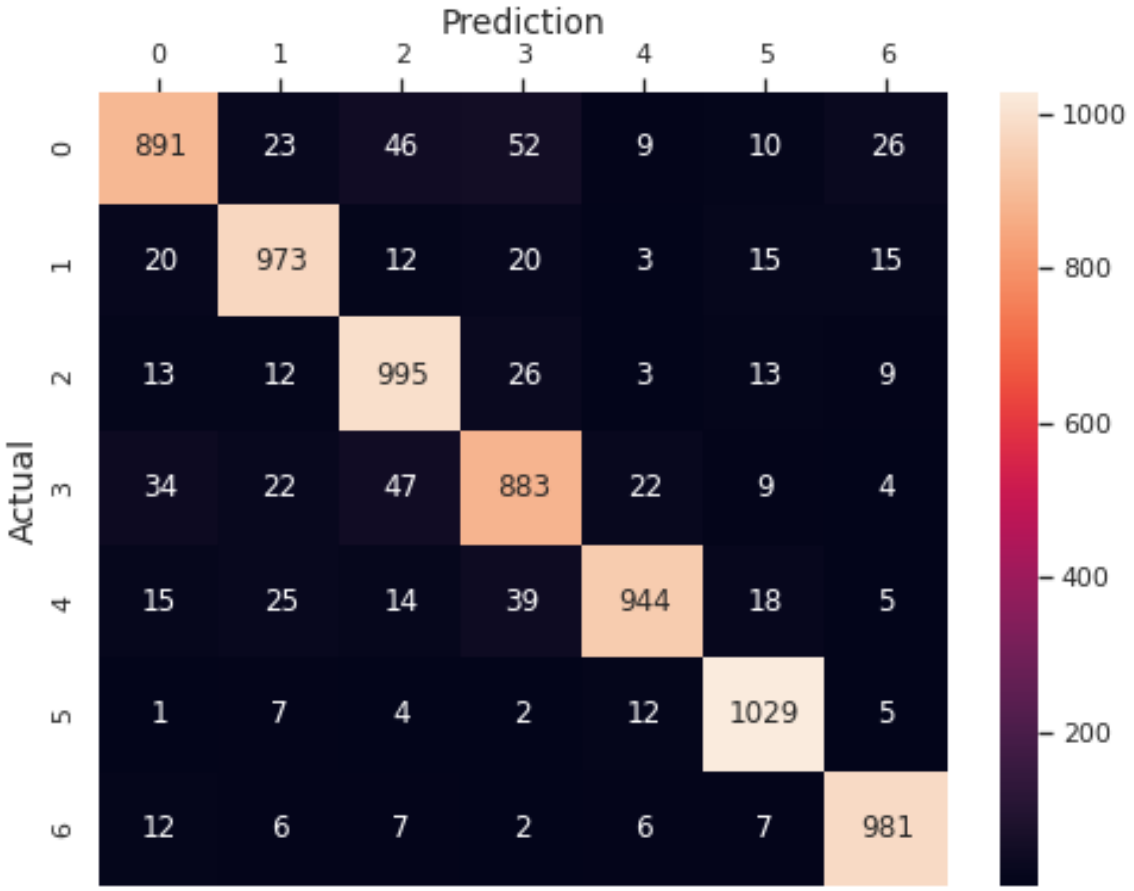
```
Model Accuracy on Test Data: 90.90
Model Precision on Test Data: 90.88
Model Recall on Test Data: 90.90
Model F1 on Test Data: 90.86
array([[ 964,   22,   24,   57,   16,   12,   19],
       [  18, 1029,   26,   20,   15,    9,    8],
       [  13,   18, 1033,   19,    5,    8,    3],
       [  45,   19,   52,  871,   59,   12,   10],
       [  11,   25,   18,   38,  990,   11,    3],
       [   3,   10,   13,    3,   15,  986,    3],
       [   4,   10,    8,    2,    0,    8, 1063]], dtype=int64)
```

لاحظ ارتفاع قيم جميع المقاييس مما يعني جودة المُصنف.

يُمكن رسم مصفوفة الارتباك confusion matrix بشكل أوضح باستخدام المكتبة seaborn:

```
# رسم مصفوفة الارتباك
import seaborn as sns
sns.set(style = 'whitegrid')

fig, ax = plt.subplots(figsize = (8,6))
sns.heatmap(confusion_matrix(y_true = y_test, y_pred = y_pred), fmt =
'g', annot = True)
ax.xaxis.set_label_position('top')
ax.xaxis.set_ticks_position('top')
ax.set_xlabel('Prediction', fontsize = 14)
ax.set_ylabel('Actual', fontsize = 14)
plt.show()
```

مما يُظهر المخطط التالي:



يُمكن حساب بعض مقاييس الأداء الأخرى المُستخدمة في حالة وجود أكثر من صف في المسألة (Micro, Macro, Weighted):

```
# مقاييس الأداء في حالة أكثر من صفين
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))


print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
average='micro')))
print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='micro')))
print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,
average='micro')))


print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred,
average='macro')))
print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='macro')))
```

```
print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred,
average='macro')))


print('Weighted Precision: {:.2f}'.format(precision_score(y_test,
y_pred, average='weighted')))
print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred,
average='weighted')))
print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred,
average='weighted')))


# تقرير التصنيف
from sklearn.metrics import classification_report
print('\nClassification Report\n')
print(classification_report(y_test, y_pred, target_names=classes))
```

مما يُعطي الناتج التالي:

```
Accuracy: 0.91


Micro Precision: 0.91
Micro Recall: 0.91
Micro F1-score: 0.91

Macro Precision: 0.91
Macro Recall: 0.91
Macro F1-score: 0.91

Weighted Precision: 0.91
Weighted Recall: 0.91
Weighted F1-score: 0.91

Classification Report

              precision    recall  f1-score   support

    Business       0.90      0.84      0.87      1057
     Economy       0.91      0.92      0.92      1058
        News       0.88      0.93      0.91      1071
```

```
      Politics       0.86      0.86      0.86      1021
        Social       0.94      0.89      0.92      1060
        Sports       0.93      0.97      0.95      1060
    Technology       0.94      0.96      0.95      1021

      accuracy                           0.91      7348
     macro avg       0.91      0.91      0.91      7348
  weighted avg       0.91      0.91      0.91      7348
```

لاحظ ارتفاع جميع المقاييس مما يعني جودة المُصنف.

9.12 تصنيف الأشخاص

يُمكن الآن تصنيف الأشخاص وذلك وفق تصنيفات تغريداتهم.

تقوم الشيفرة التالية بتصنيف تغريدة واحدة، فيكون دخل الدالة `classify_tweet` سلسلة نصية نضعها أولًا في متجهة تحوي النص. نستدعي دالة تحويل النص إلى أرقام `tokenizer.texts_to_sequences` ومن ثم توحيد طول المتجهة الرقمية الناتجة وذلك باستدعاء الدالة `pad_sequences`.

```
# دالة تصنيف تغريدة
def classify_tweet(tweet):
    # تحويل شعاع الكلمات إلى جملة
    tweet = toSentence(tweet)
    # وضع الجملة في شعاع
    ar=[]
    ar.append(tweet)
    # تحويل النص إلى قائمة من الأرقام
    seq = tokenizer.texts_to_sequences(ar)
    # توحيد طول المتجهة الرقمية
    pseq = pad_sequences(seq, maxlen=max_len)
    # استدعاء دالة التنبؤ للنموذج
    pred = model.predict(pseq)
    return pred
```

تقوم الدالة `classify_person` بتصنيف شخص، ويكون معامل الدالة اسم الشخص (بفرض أن ملف تغريداته يحمل نفس الاسم مع اللاحقة csv)، إذ تعمل الدالة أولًا على تحميل تغريدات الشخص في إطار بيانات `df`.

وأخيرًا تكون دالة تصنيف الشخص `classify_person` هي:

```
# دالة تصنيف الشخص
def classify_person(person_name):
    # تحميل تغريدات الشخص
    # في إطار بيانات
    path = person_name + '.csv'
    df = pd.read_csv(path)
    # إنشاء قاموس لعد
    # التغريدات من كل صف
    classes_count=dict()
    # 0 إعطاء قيم ابتدائيه
    for i in range(len(classes)):
        key=classes[i]
        classes_count[key]=0
    # الحد الأدنى لطول التغريدة
    min_tweet_len=5

    total=0
    for _, row in df.iterrows():
        tweet=row['tweet']
        # تنظيف التغريدة
        processed_tweet=process_tweet(tweet)
        if len(processed_tweet)>min_tweet_len:
          # تصنيف التغريدة
          c= classify_tweet(processed_tweet)
          # إيجاد اسم الصف من رمزه
          topic=le_topics.inverse_transform(c)[0]
          # إضافة 1 للصف الموافق
          classes_count[topic]=classes_count[topic]+1
          total=total+1

    # ترتيب الصفوف وفق العدد
    # تنازلياً

    sorted_classes = sorted(classes_count,
key=classes_count.get,reverse=True)
```

```
    # القاموس النهائي
    sorted_classes_cleaned = {}
    min_display=total/25
    # إهمال الصفوف ذات العدد الصغير
    for w in sorted_classes:
      if classes_count[w]>min_display:
        sorted_classes_cleaned[w] = classes_count[w]

    # طباعة النتائج
    print(sorted_classes_cleaned)
    n=0
    for key, value in sorted_classes_cleaned.items():
      n=n+value

    print(person_name, "is classified as :")
    for key, value in sorted_classes_cleaned.items():
      print(key, "(", "{:.2f}".format((value/n)*100) , "%)")

    # رسم فطيرة أعداد الصفوف
    x = sorted_classes_cleaned.keys()
    y = sorted_classes_cleaned.values()

    import matplotlib.pyplot as plt
    # pie
    plt.figure(figsize=(9,9));
    plt.title(person_name, fontdict = {'fontsize':20})
    plt.pie(y, labels = x,autopct='%1.1f%%')
    plt.show()
# مثال
classify_person("salem")
```

نجد من الشيفرة السابقة، تُنشئ الدالة القاموس `classes_count` والذي يكون مفتاحه اسم الصف وقيمته ستكون عدد التغريدات من هذا الصف (تُعطي أولًا قيمة ابتدائية 0 لكل قيم القاموس).

يُحدّد المتغير `min_tweet_len` الطول الأدنى للتغريدة لتؤخذ بعين الاعتبار (لا تحمل التغريدات الصغيرة جدًا معاني بل تكون على الأغلب عبارات مجاملة وترحيب).

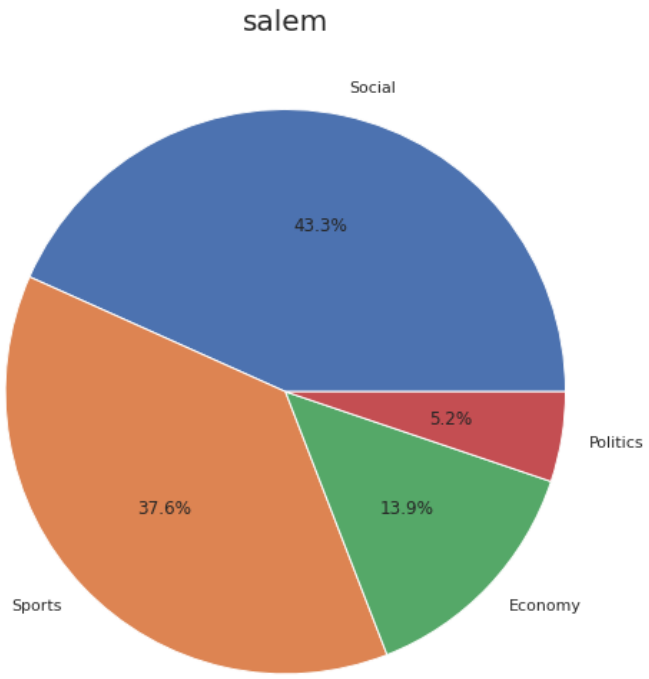
تـدور الدالـة على تغريـدات الشـخص وتسـتدعي من أجـل كـل تغريـدة دالـة المعالجـة الأوليـة للتغريـدة `process_tweet` ثم دالة التصنيف السابقة `classify_tweet`.

يكون ناتج التصنيف رقم (ترميز الصف) ولذا تستدعي الدالة `inverse_transform` للحصـول على اسـم الصف ثم نزيد عدد الصف الموافق في القاموس بـ 1.

يحسب المتغير `total` العـدد الكلي للتغريـدات الـتي تم أخـذها بعين الاعتبـار، وتُسـتخدم الدالـة المتغـير `sorted` لترتيب القاموس تنازليا (المعامل `reverse=True`) وفق العدد.

نضع في المتغير `min_display` مثلًا القيمة `total/25` وذلك بهدف عدم إظهار صفوف الشخص قليلـة التواتر في تغريداته، ونضع أخيرًا في القـاموس `sorted_classes_cleaned` الصـفوف الأكـثر تـواترًا ونطبـع النتائج ونرسم فطيرة تُمثل النسب المئوية لتغريدات الشخص.

تكون النتائج مثلًا:



9.13 الخلاصة

عرضنا في هذا الفصل خطوات بناء نموذج تعلّم لتصنيف الأشـخاص وفـق تغريـداتهم باللغـة العربيـة وهـو مثال قلما تجده في المحتوى العربية ويساعدك على التعامل مع النصوص العربية في الحالات التي تريدها.

> يُمكن تجربة المثال كاملًا من موقع Google Colab من هذا الرابط ولا تنس تنزيل كامل ملفات المشروع من هذا الرابط من أكاديمية حسوب.

10. تحليل مبيعات متجر واستكشاف ترابط منتجاته

تُعدّ مسألة استكشاف قواعد الترابط في مبيعات المتاجر من المسائل المهمة جدًا لأصحاب المتاجر الإلكترونية، إذ يسمح إيجاد هذه القواعد بإظهار توصيات recommendations الشراء المناسبة للزبائن مما يساهم في زيادة مبيعات المتجر مثلًا لو عرفنا أن معظم الزبائن تشتري السلعة B مع السلعة A دومًا، فسيكون من المناسب إظهار توصية بشراء السلعة B لكل زبون يطلب شراء السلعة A مما يُحقق، في نهاية المطاف، رضى الزبون وزيادة أرباح المتجر.

نعرض في هذا الفصل استخدام تقنيات تعلم الآلة لإيجاد قواعد الترابط انطلاقًا من حركات الشراء السابقة للمتجر.

10.1 بيانات التدريب

نستخدم بيانات عمليات الشراء لأحد متاجر مبيعات التجزئة الأجنبية الغربية والمتاحة على الرابط أو يمكنك تنزيلها من هذا الرابط من أكاديمية حسوب.

تأتي هذه البيانات في الملف Groceries.csv المرفق والذي يحوي حوالي 10000 صف. يحوي كل صف بيانات عربة تسوق واحدة أي مجموعة من السلع التي اشتراها أحد الزبائن معًا كما يُبين الشكل التالي:

	A	B	C	D
1	citrus fruit	semi-finished bread	margarine	ready soups
2	tropical fruit	yogurt	coffee	
3	whole milk			
4	pip fruit	yogurt	cream cheese	meat spreads
5	other vegetables	whole milk	condensed milk	long life bakery product
6	whole milk	butter	yogurt	rice
7	rolls/buns			

10.2 أساسيات في التنقيب عن قواعد الترابط

نعرض فيما يلي بعض التعاريف الأساسية في مسألة التنقيب عن قواعد الترابط.

10.2.1 التنقيب عن قواعد الترابط

تُعرّف مسألة التنقيب عن قواعد الترابط كما يلي:

بفرض أن لدينا مجموعة من الإجراءات transactions، يتألف كل إجراء من مجموعة من العناصر items.

يكون المطلوب إيجاد جميع الترابطات correlations بين ظهور مجموعة جزئية من العناصر مع مجموعة جزئية أخرى. يُبين الشكل التالي مثالًا توضيحيًا:

نستخدم فيما يلي المثال التالي التعليمي والذي يحوي 5 إجراءات:

الإجراء	Transaction
خبز، حليب	Bread,Milk
خبز، فوط، عصير، بيض	Bread, Diaper, Juice, Eggs
حليب، فوط، عصير، كولا	Milk, Diaper, Juice, Coke
خبز، حليب، فوط، عصير	Bread, Milk, Diaper, Juice
خبز، حليب، فوط، كولا	Bread, Milk, Diaper, Coke

نُعطي أولًا أهم التعاريف الأساسية:

I. مجموعة عناصر itemset

وهي مجموعة من العناصر مثلًا: { Diaper , Bread , Milk} (فوط أطفال، خبز، حليب) كما ندون k-itemset للدلالة على مجموعة عناصر تحوي k عنصر.

ب. عدد الدعم support count لمجموعة من العناصر

وهو عدد مرات تواتر (ظهور) مجموعة من العناصر في الإجراءات مثلًا يكون عدد الـدعم في مثالنـا التعليمي لمجموعة العناصر السابقة:

```
sc({Milk, Bread, Diaper}) = 2
```

ج. الدعم support لمجموعة من العناصر

وهو النسبة المئوية لظهور مجموعة من العناصر في الإجراءات، مثلًا يكـون عـدد الـدعم في مثالنـا التعليمي لمجموعة العناصر سابقة الذكر:

```
s({Milk, Bread, Diaper}) = (2/5)*100 = 40%
```

د. الحد الأدنى للدعم minimum support

وهو حد أدنى تجريبي للدعم (يُمكن أن يكون رقم أو نسبة مئوية) نُحدّده لخوارزميات التنقيب عن القواعد.

> يُمثّل الدعم عمليًا أهمية العناصر، بمعنى أنه كلما كان الدعم مرتفعًا فهذا يحصر اهتمامنا في السلع التي تُباع مرارًا في المتجر.

ه. مجموعة عناصر متواترة frequent itemset

نقول عن مجموعة عناصر أنها متواترة إذا كان دعم المجموعة أكبر أو يساوي الحد الأدنى للدعم.

و. قاعدة ترابط association rule

وهي اقتضاء من الشكل:

```
{X} => {Y}
```

وحيث X و Y هي مجموعات من العناصر (لا تحوي، بالطبع، عناصر مشتركة) مثلاً:

```
{Milk, Diaper} => {Bread}
```

ز. الدعم لقاعدة ترابط association rule support

يكون الدعم لقاعدة ترابط X=>Y هو الدعم لاجتماع العناصر X و Y معًا. أي نسبة الإجراءات التي تحوي X و Y معًا من عدد الإجراءات الكلي:

$$Support(\{X\}\rightarrow\{Y\})=\frac{Transactions\ \ containing\ \ both\ \ X\ \ \text{and}\ \ Y}{Total\ \ number\ \ of\ \ transactions}$$

ح. الثقة في قاعدة ترابط association rule confidence

وهو الاحتمال الشرطي لظهور مجموعة العناصر Y في إجراء يحوي X، أي عمليًا احتمال (الثقة) أن تظهر مجموعة العناصر Y في عربة تسوق تحوي العناصر X.

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ \ containing\ \ both\ \ X\ \ \text{and}\ \ Y}{Transactions\ \ containing\ \ X}$$

مثلًا يكون الدعم لمجموعة العناصر {Milk, Diaper, Bread} في المثال السابق مساويًا 40% لأن هذه العناصر ظهرت مع بعضها البعض مرتين في الإجراءات الخمسة الكلية.

يُمكن من هذه العناصر الثلاثة {Milk, Diaper, Bread} توليد مجموعة من القواعد المختلفة (يكفي أن نوزع هذه العناصر على الطرف اليساري واليمني بكل الطرق الممكنة لنحصل على جميع القواعد). يُمكن أن يكون لكل قاعدة معامل ثقة مختلف كما تُبين الأمثلة التالية (المحسوبة من مثالنا التعليمي):

```
{Milk, Diaper} => {Bread} (s=0.4, c=0.67)
{Milk, Bread} => {Diaper} (s=0.4, c= 0.67)
{Diaper, Bread} => {Milk} (s=0.4, c=0.67)
{Bread} => {Milk, Diaper} (s=0.4, c=0.50)
{Diaper} => {Milk, Bread} (s=0.4, c=0.5)
{Milk} => {Diaper, Bread} (s=0.4, c=0.5)
```

نعرض في هذا الفصل كيفية استخراج قواعد الترابط التي تحقق دعم وثقة معينين.

10.3 إعداد المشروع

يحتاج تنفذ شيفرات هذا الفصل بيئةً برمجيةً للغة بايثون الإصدار 3.8. ويجب أن تتضمن هذه البيئة البرمجية مدير الحِزم pip لتثبيت الحِزم، ومُنشئ البيئات الافتراضية venv لإنشاء بيئاتٍ افتراضيةٍ.

نستخدم محرر الشيفرات البرمجية Jupyter Notebooks، وهو مفيد جدًا لتجربة وتشغيل الأمثلة الخاصة بتَعَلّم الآلة بطريقةٍ تفاعليةٍ، حيث نستطيع من خلاله تشغيل كتلًا صغيرةً من الشيفرات البرمجية ورؤية النتائج بسرعة، مما يُسهّل علينا اختبار الشيفرات البرمجية وتصحيحها.

نحتاج أولًا لتثبيت بعض التبعيات، وذلك لإنشاء مساحة عملٍ للاحتفاظ بملفاتنا قبل أن نتمكن من تطوير برنامجنا، نُنشئ مجلدًا جديدًا خاصًا بمشروعنا وندخل إليه هكذا:

```
mkdir asoc
cd asoc
```

نُنفذّ الأمر التالي لإنشاء البيئة الافتراضية:

```
python -m venv asoc
```

ومن ثم الأمر التالي في Linux لتنشيط البيئة الافتراضية:

```
source asoc/bin/activate
```

أما في ويندوز Windows، فيكون أمر التنشيط:

```
"asoc/Scripts/activate.bat"
```

نستخدم إصداراتٍ مُحددةٍ من المكتبات اللازمة، من خلال إنشـاء ملـف `requirements.txt` في مجلـد المشروع، وسيُحدِّد هذا الملف المتطلبات والإصدارات التي سنحتاج إليها.

نفتح الملف `requirements.txt` في محرر النصوص، ونُضيف الأسطر التالية، وذلك لتحديد المكتبـات التي نريدها وإصداراتها:

```
asttokens==2.0.5
backcall==0.2.0
colorama==0.4.4
cycler==0.11.0
debugpy==1.6.0
decorator==5.1.1
entrypoints==0.4
executing==0.8.3
fonttools==4.33.3
ipykernel==6.13.0
ipython==8.4.0
jedi==0.18.1
joblib==1.1.0
jupyter-client==7.3.1
jupyter-core==4.10.0
kiwisolver==1.4.2
matplotlib==3.5.2
matplotlib-inline==0.1.3
mlxtend==0.20.0
nest-asyncio==1.5.5
numpy==1.22.4
```

```
packaging==21.3
pandas==1.4.2
parso==0.8.3
pickleshare==0.7.5
Pillow==9.1.1
prompt-toolkit==3.0.29
psutil==5.9.1
pure-eval==0.2.2
Pygments==2.12.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
pywin32==304
pyzmq==23.0.0
scikit-learn==1.1.1
scipy==1.8.1
seaborn==0.11.2
six==1.16.0
stack-data==0.2.0
threadpoolctl==3.1.0
tornado==6.1
traitlets==5.2.2.post1
wcwidth==0.2.5
```

نحفظ التغييرات التي طرأت على الملف ونخرج من محرر النصوص، ثم نُثَبت هذه المكتبات بالأمر التالي:

```
(asoc) $ pip install -r requirements.txt
```

بعد تثبيتنا لهذه التبعيات، نُصبح جاهزين لبدء العمل على مشروعنا.

10.4 كتابة الشيفرة البرمجية

نُشغّل محرر الشيفرات البرمجية Jupyter Notebook بمجرد اكتمال عملية التثبيت هكذا:

```
(asoc) $ jupyter notebook
```

ثم نُنشئ ملفًا جديدًا في داخل المحرر ونُسمّه باسم asc مثلًا.

10.4.1 توليد قواعد الترابط

نعرض في الشيفرة التالية توليد قواعد الترابط للمثال السابق بهدف التعرف على المكتبات اللازمة وآلية استخدامها.

نضع إجراءات المثال السابق أولًا في مصفوفة ثنائية وبحيث يكون كل عنصر منها هو مصفوفة من عناصر إجراء واحد.

نحتاج أولًا إلى توليد إطار بيانات dataframe وبحيث تكون رؤوس الأعمدة هي العناصر وقيم الخلايا هي إما True في حال وجود العنصر في صف row الإجراء الموافق أو False في حال عدم وجوده.

نستخدم الصنف `TransactionEncoder` من المكتبة mlxtend.preprocessing للوصول إلى ذلك.

```
# الإجراءات
transactions = [['Bread', 'Milk'],
['Bread', 'Diaper', 'Juice', 'Eggs'],
['Milk', 'Diaper', 'Juice', 'Coke' ],
['Bread', 'Milk', 'Diaper', 'Juice'],
['Bread', 'Milk', 'Diaper', 'Coke']]

# مكتبة ترميز الإجراءات
from mlxtend.preprocessing import TransactionEncoder

# إنشاء غرض من الصف
te = TransactionEncoder()
# ملائمة المرمز مع البيانات
te_model = te.fit(transactions)
# تحويل الإجراءات
rows=te_model.transform(transactions)
# استيراد مكتبة إطار البيانات
import pandas as pd
# بناء إطار بيانات الإجراءات
df = pd.DataFrame(rows, columns=te_model.columns_)
print(df)
```

وبالنتيجة يكون لدينا إطار بيانات الإجراءات التالي:

	Bread	Coke	Diaper	Eggs	Juice	Milk
0	True	False	False	False	False	True
1	True	False	True	True	True	False
2	False	True	True	False	True	True
3	True	False	True	False	True	True
4	True	True	True	False	False	True

لاحظ مثلًا أن الصف الأول من إطار البيانات يوافق الإجراء الأول: {Bread, Milk}.

نستدعي في الشيفرة التالية الدالة `apriori` من المكتبة `mlxtend.frequent_patterns` والتي تحسب العناصر المتواترة في إطار البيانات السابق `df` وفق حد أدنى معين للدعم `min_support` يساوي 40% في مثالنا.

يكون ناتج تطبيق هذه الدالة إطار بيانات `frequent_itemsets` ذي عمودين: مجموعة العناصر المتواترة `itemsets` والدعم support. نُضيف عمود محسوب جديد لإطار البيانات الناتج يحسب طول كل مجموعة عناصر `length`.

```
# مكتبة خوارزمية إيجاد العناصر المتواترة
from mlxtend.frequent_patterns import apriori
# توليد المجموعات المتواترة مع تحديد الحد الأدنى للدعم
frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
# حساب أطوال مجموعات العناصر
frequent_itemsets['length'] =
frequent_itemsets['itemsets'].apply(lambda x: len(x))
print(frequent_itemsets)
```

تُظهر الطباعة دعم وطول كل مجموعة من مجموعات العناصر المتواترة:

	support	itemsets	length
0	0.8	(Bread)	1
1	0.4	(Coke)	1
2	0.8	(Diaper)	1
3	0.6	(Juice)	1
4	0.8	(Milk)	1
5	0.6	(Bread, Diaper)	2
6	0.4	(Bread, Juice)	2
7	0.6	(Milk, Bread)	2
8	0.4	(Coke, Diaper)	2
9	0.4	(Milk, Coke)	2
10	0.6	(Juice, Diaper)	2
11	0.6	(Milk, Diaper)	2
12	0.4	(Milk, Juice)	2
13	0.4	(Bread, Diaper, Juice)	3
14	0.4	(Milk, Bread, Diaper)	3
15	0.4	(Milk, Coke, Diaper)	3
16	0.4	(Milk, Juice, Diaper)	3

يُمكن الآن توليد قواعد الترابط باستخدام إطار بيانات العناصر المتـواترة `frequent_itemsets` السـابق، كما تُبين الشيفرة التالية.

نستخدم الدالة `association_rules` من المكتبة `mlxtend.frequent_patterns` مع تحديد الحد الأدنى للثقة `min_threshold` (لنأخذ القيمة 60% في مثالنا).

```
# مكتبة خوارزمية إيجاد قواعد الترابط
from mlxtend.frequent_patterns import association_rules
# توليد القواعد مع تحديد الحد الأدنى للثقة
rules =
association_rules(frequent_itemsets,metric="confidence",min_threshold=
0.6)
# الترتيب التنازلي وفق معامل الثقة
rules = rules.sort_values(['confidence'], ascending =[False])
print(rules)
```

يُبين الشكل التالي قواعد الترابط الناتجة:

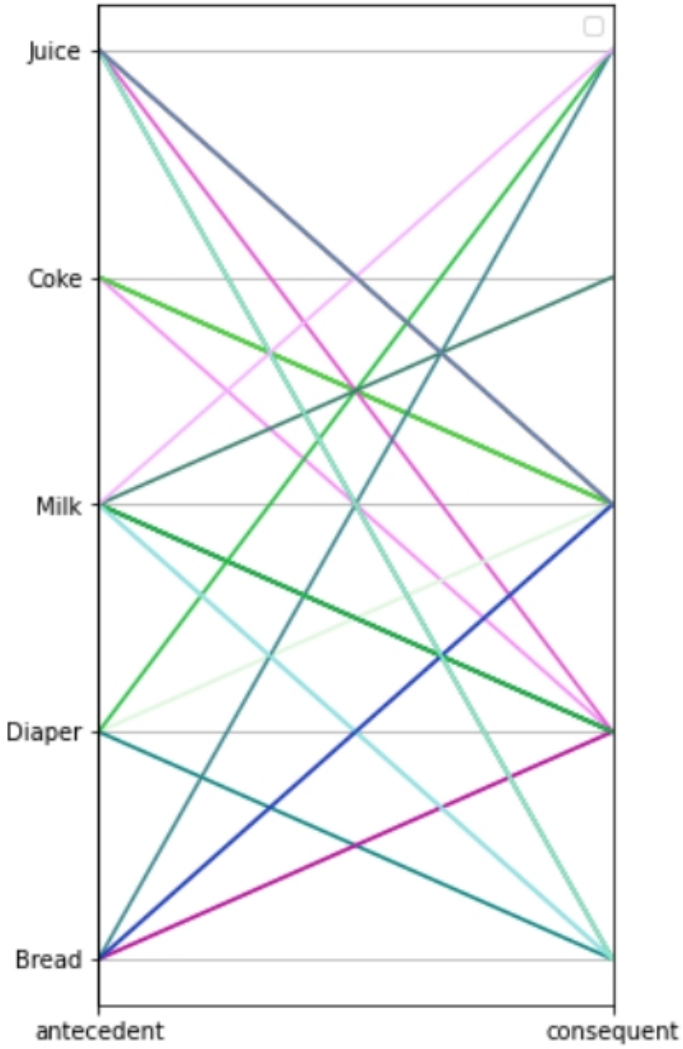
	antecedents	consequents	antecedent support	consequent support	support	confidence
13	(Bread, Juice)	(Diaper)	0.4	0.8	0.4	1.000000
23	(Milk, Juice)	(Diaper)	0.4	0.8	0.4	1.000000
22	(Coke)	(Milk, Diaper)	0.4	0.6	0.4	1.000000
21	(Coke, Diaper)	(Milk)	0.4	0.8	0.4	1.000000
5	(Coke)	(Diaper)	0.4	0.8	0.4	1.000000
6	(Coke)	(Milk)	0.4	0.8	0.4	1.000000
7	(Juice)	(Diaper)	0.6	0.8	0.6	1.000000
19	(Milk, Coke)	(Diaper)	0.4	0.8	0.4	1.000000
10	(Diaper)	(Milk)	0.8	0.8	0.6	0.750000
1	(Diaper)	(Bread)	0.8	0.8	0.6	0.750000
0	(Bread)	(Diaper)	0.8	0.8	0.6	0.750000
9	(Milk)	(Diaper)	0.8	0.8	0.6	0.750000
8	(Diaper)	(Juice)	0.8	0.6	0.6	0.750000
4	(Bread)	(Milk)	0.8	0.8	0.6	0.750000
3	(Milk)	(Bread)	0.8	0.8	0.6	0.750000
11	(Juice)	(Milk)	0.6	0.8	0.4	0.666667
12	(Bread, Diaper)	(Juice)	0.6	0.6	0.4	0.666667
14	(Juice, Diaper)	(Bread)	0.6	0.8	0.4	0.666667
15	(Juice)	(Bread, Diaper)	0.6	0.6	0.4	0.666667
16	(Milk, Bread)	(Diaper)	0.6	0.8	0.4	0.666667
17	(Milk, Diaper)	(Bread)	0.6	0.8	0.4	0.666667
18	(Bread, Diaper)	(Milk)	0.6	0.8	0.4	0.666667
20	(Milk, Diaper)	(Coke)	0.6	0.4	0.4	0.666667
2	(Juice)	(Bread)	0.6	0.8	0.4	0.666667
24	(Milk, Diaper)	(Juice)	0.6	0.6	0.4	0.666667
25	(Juice, Diaper)	(Milk)	0.6	0.8	0.4	0.666667
26	(Juice)	(Milk, Diaper)	0.6	0.6	0.4	0.666667

يُمكن استخدام رسم الإحداثيات المتوازية `parallel_coordinates` من المكتبة `pandas.plotting` لرسم مشاهدة توضيحية للقواعد السابقة.

```
from matplotlib import pyplot as plt
from pandas.plotting import parallel_coordinates
# دالة تحويل القواعد إلى إحداثيات
def rules_to_coordinates(rules):
    rules['antecedent'] = rules['antecedents'].apply(lambda
antecedent: list(antecedent)[0])
    rules['consequent'] = rules['consequents'].apply(lambda
consequent: list(consequent)[0])
    rules['rule'] = rules.index
    return rules[['antecedent','consequent','rule']]
# توليد الإحداثيات المتوازية
coords = rules_to_coordinates(rules)
# توليد رسم الإحداثيات المتوازية
plt.figure(figsize=(4,8))
parallel_coordinates(coords, 'rule')
plt.grid(True)
```

```
plt.show()
```

يكون الرسم البياني الناتج:



يُبين الرسم ارتباطات العناصر، وذلك برسم خط بين العنصر من الجهة اليسرى وبين نهاية الخط الأفقي للعنصر الآخر المرتبط معه من الجهة اليمنى، مثلًا يرتبط الحليب Milk مع كل من الخبز Bread والفوط Diaper.

10.4.2 تحميل بيانات المتجر

نبدأ أولًا بتحميل بيانات المتجر من الملف Groceries.csv ضمن إطار من البيانات DataFrame من مكتبة Pandas ومن ثم عرض بعضها:

```
# تحميل بيانات المتجر
df = pd.read_csv('Groceries.csv',header=None)
df.head()
```

يظهر لنا أوائل صفوف الملف:

	0	1	2	3	4	5	6	7	8	9
0	citrus fruit	semi-finished bread	margarine	ready soups	NaN	NaN	NaN	NaN	NaN	NaN
1	tropical fruit	yogurt	coffee	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	whole milk	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	pip fruit	yogurt	cream cheese	meat spreads	NaN	NaN	NaN	NaN	NaN	NaN
4	other vegetables	whole milk	condensed milk	long life bakery product	NaN	NaN	NaN	NaN	NaN	NaN

نلاحظ وجود قيم كثيرة فارغة NaN وذلك لأن عدد العناصر في كل صف غير متساوي.

نحذف في الشفرة التالية القيم الفارغة، ثم نُنشئ مصفوفة الإجراءات والتي هي مصفوفة يكـون كـل عنصـر منها مصفوفة من عناصر صف واحد من إطار البيانات:

```
# حذف القيم الفارغة
# وإنشاء مصفوفة
transactions =  df.T.apply(lambda x: x.dropna().tolist()).tolist()
print(transactions[1:10]))
```

يُبين الشكل التالي مثلًا العناصر العشرة الأولى من مصفوفة الإجراءات الناتجة:

```
[['tropical fruit', 'yogurt', 'coffee'], ['whole milk'], ['pip fruit', 'yogurt', 'cream cheese ', 'meat spreads'], ['other vegetables',
'whole milk', 'condensed milk', 'long life bakery product'], ['whole milk', 'butter', 'yogurt', 'rice', 'abrasive cleaner'], ['rolls/buns'],
['other vegetables', 'UHT-milk', 'rolls/buns', 'bottled beer', 'liquor (appetizer)'], ['pot plants'], ['whole milk', 'cereals']]
```

نستخدم في الشيفرة التالية مُرمز الإجراءات للحصول على إطار بيانات الإجراءات:

```
# إنشاء غرض من الصف
te = TransactionEncoder()
# ملائمة المرمز مع البيانات
te_model = te.fit(transactions)
# تحويل الإجراءات
```

```
rows=te_model.transform(transactions)
# بناء إطار بيانات الإجراءات
df = pd.DataFrame(rows, columns=te_model.columns_)
print(df.shape)
```

مما يُعطي:

```
(9835, 169)
```

لاحظ أن عدد أعمدة إطار البيانات الناتج هـو 169 عمـودًا ممـا يعـني وجـود 169 عنصـرًا مختلفًـا فقـط في الإجراءات البالغ عددها 9835 إجراء.

نستدعي في الشيفرة التالية الدالة `apriori` والتي تحسب العناصر المتـواترة في إطـار البيانـات `df` وفـق حد أدنى معين للدعم `min_support` يساوي 0.5% (هو رقم تجريبي حصلنا عليه بتكرار توليد العناصر المتواترة وقواعد الترابط حتى الوصول لقواعد ترابط عددها محدود نسبيًا).

يكون ناتج تطبيق هذه الدالة إطار بيانات frequent_itemsets ذي عمـودين: مجموعـة العناصـر المتـواترة `itemsets` والدعم `support`. نُضيف عمود محسوب جديد لإطار البيانات الناتج يحسـب طـول `length` كـل مجموعة عناصر.

```
# توليد المجموعات المتواترة مع تحديد الحد الأدنى للدعم
frequent_itemsets = apriori(df, min_support=0.005, use_colnames=True)
# حساب أطوال مجموعات العناصر
frequent_itemsets['length'] =
frequent_itemsets['itemsets'].apply(lambda x: len(x))
print(frequent_itemsets)
```

تُبين النتائج أن لدينا 1001 مجموعة من العناصر المتواترة المُحققة للحد الأدنى للـدعم ويـتراوح طولهـا بين عدد 1 و 4، انظر الصورة التالية:

```
      support                                           itemsets  length
0    0.008033                       (Instant food products)       1
1    0.033452                                      (UHT-milk)       1
2    0.017692                                  (baking powder)       1
3    0.052466                                        (beef)       1
4    0.033249                                     (berries)       1
...       ...                                             ...     ...
996  0.005186  (other vegetables, root vegetables, whole milk...       4
997  0.007829  (other vegetables, root vegetables, yogurt, wh...       4
998  0.007626  (other vegetables, tropical fruit, yogurt, who...       4
999  0.005592  (other vegetables, yogurt, whole milk, whipped...       4
1000 0.005694  (tropical fruit, root vegetables, whole milk, ...       4

[1001 rows x 3 columns]
```

يُمكن الآن توليد قواعد الترابط باستخدام إطار بيانات العناصر المتواترة السابق كما تُبين الشيفرة التالية. نستخدم الدالة association_rules مع تحديد الحد الأدنى للثقة 55% (رقم تجريبي حصلنا عليه بعد عدة محاولات لتوليد قواعد الترابط حتى وصلنا لمجموعة معقولة من القواعد).

```
# توليد القواعد مع تحديد الحد الأدنى للثقة
rules =
association_rules(frequent_itemsets,metric="confidence",min_threshold=
0.55)
# الترتيب التنازلي وفق معامل الثقة
rules = rules.sort_values(['confidence'], ascending =[False])
print(rules)
```

يُبين الشكل التالي قواعد الترابط الناتجة (حوالي 50 قاعدة):

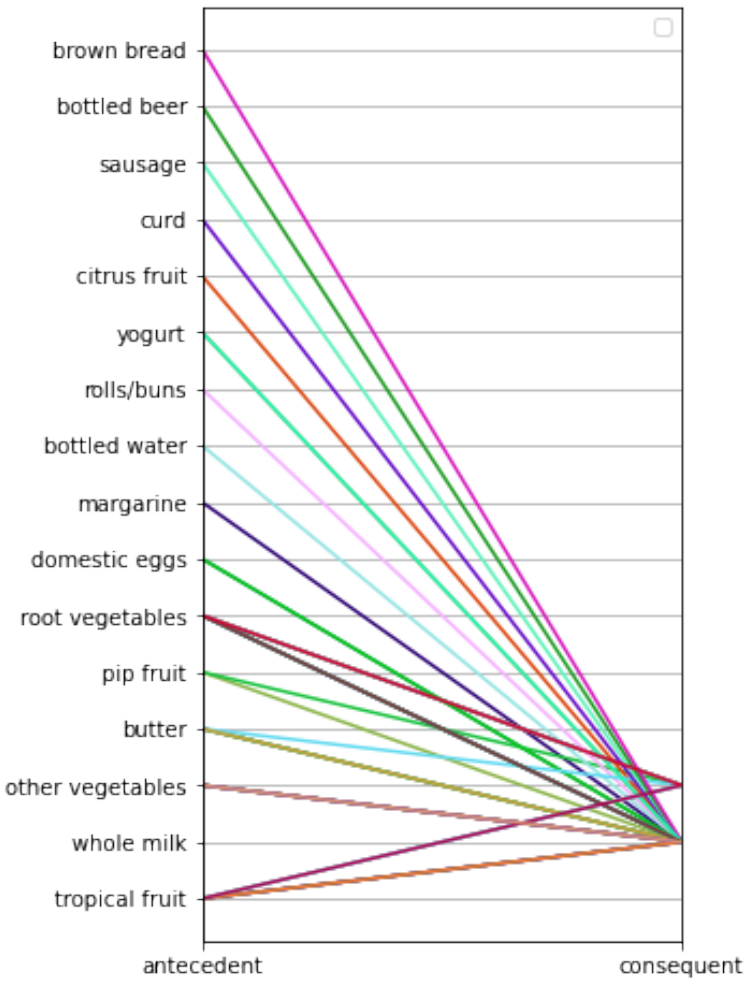
antecedents	consequents	antecedent support	consequent support	support	confidence
(tropical fruit, root vegetables, yogurt)	(whole milk)	0.008134	0.255516	0.005694	0.700000
(other vegetables, root vegetables, pip fruit)	(whole milk)	0.008134	0.255516	0.005491	0.675000
(butter, whipped/sour cream)	(whole milk)	0.010168	0.255516	0.006711	0.660000
(pip fruit, whipped/sour cream)	(whole milk)	0.009253	0.255516	0.005999	0.648352
(butter, yogurt)	(whole milk)	0.014642	0.255516	0.009354	0.638889
(butter, root vegetables)	(whole milk)	0.012913	0.255516	0.008236	0.637795
(tropical fruit, curd)	(whole milk)	0.010269	0.255516	0.006507	0.633663
(root vegetables, citrus fruit, whole milk)	(other vegetables)	0.009151	0.193493	0.005796	0.633333
(other vegetables, yogurt, pip fruit)	(whole milk)	0.008134	0.255516	0.005084	0.625000
(domestic eggs, pip fruit)	(whole milk)	0.008643	0.255516	0.005389	0.623529
(tropical fruit, butter)	(whole milk)	0.009964	0.255516	0.006202	0.622449
(margarine, domestic eggs)	(whole milk)	0.008338	0.255516	0.005186	0.621951

يُمكن استخدام رسم الإحداثيات المتوازية لرسم مشاهدة توضيحية للقواعد السابقة:

```
from pandas.plotting import parallel_coordinates
# دالة تحويل القواعد إلى إحداثيات
def rules_to_coordinates(rules):
```

```
    rules['antecedent'] = rules['antecedents'].apply(lambda
antecedent: list(antecedent)[0])
    rules['consequent'] = rules['consequents'].apply(lambda
consequent: list(consequent)[0])
    rules['rule'] = rules.index
    return rules[['antecedent','consequent','rule']]
# توليد الإحداثيات المتوازية
coords = rules_to_coordinates(rules)
# توليد رسم الإحداثيات المتوازية
plt.figure(figsize=(4,8))
parallel_coordinates(coords, 'rule')
plt.legend([])
plt.grid(True)
plt.show()
```

يكون الرسم البياني الناتج:



10.5 الخلاصة

عرضنا في هذا الفصل خطوات بناء إيجاد قواعد الترابط بين مبيعات العناصر في المتاجر.

> يُمكن تجربة المثال كاملًا من موقع Google Colab من هذا الرابط ولا تنسَ تنزيل ملفات المشروع كاملة من هذا الرابط من أكاديمية حسوب.

11. تحليل بيانات الطاقة لمدينة نيويورك

قد تظن عندما تتابع دورةً تدريبيةً أو تقرأ كتابًا في علم البيانات، أنك تملكت من المطلوب لتنفيذ مشروعًا كاملًا في تعلّم الآلة، لكن لن تكون لديك الخبرة الكافية بالضرورة لتكامل جميع أجزاء ومراحل تطوير نظام تعلّم آلة لحل مسألة حقيقية على أرض الواقع. ربما يُشكّل تنفيذك المشروع الأول تحديًا كبيرًا لكن إتمامك له بنجاح سيمنحك الثقة والخبرة اللازمتين لتنفيذ مشاريع لاحقة بعدها.

يعرض هذا الفصل آلية تسلسل خطوات معالجة مسائل تعلّم الآلة عبر القيام بتنفيذ مشروع متكامل مع بيانات حقيقية.

نعمل وفق المنهجية العامة المتبعة عادةً في مسائل تعلّم الآلة خطوةً بخطوة وفق ما يلي:

1. تنظيف البيانات وتنسيقها.
2. استكشاف وتحليل البيانات.
3. هندسة الميزات واختيار المناسب منها.
4. اعتماد مقاييس الأداء وموازنة نماذج التعلّم وفقها.
5. ضبط قيم المعاملات الفائقة لنموذج التعلّم الأفضل.
6. تقويم النموذج الأفضل باستخدام مجموعة بيانات الاختبار.
7. تفسير وشرح نتائج النموذج.
8. عرض الاستنتاجات وتوثيق العمل.

نُنفذ فيما يلي كل خطوة من الخطوات السابقة مع بيان تكامل كل خطوة مع الخطوة التالية لها، ويُمكن الحصول على الشيفرة الكاملة للمشروع من هذا المستودع.

> نفذّ كاتب المقال هذا المشروع بصفة اختبار قبول لوظيفة في شركة ناشئة، وأُسندت الوظيفة الشاغرة له بالنتيجة، إلا أن المدير التنفيذي للشركة استقال ولم تتابع الشركة الناشئة العمل. ويبدو أن الأمور تجري هكذا في الشركات الناشئة.

11.1 تعريف المسألة

قبل البدء بالعمل البرمجي لابدّ من فهم المسألة ومعاينة البيانات المتاحة. نحلل في هذا المشروع ومعالجة بيانات الطاقة لمدينة نيويورك والمنشورة للعموم.

يهدف المشروع إلى تحليل بيانات الطاقة للوصول إلى نموذج متعلّم يُمكنه التنبؤ بمعامل نجمة الطاقة Energy Star Score الذي يعتمده برنامج نجمة الطاقة وهو برنامج تديره وكالة حماية البيئة الأمريكية ووزارة الطاقة الأمريكية لتعزيز كفاءة الطاقة. كما يجب في النهاية تفسير النتائج لمعرفة العوامل التي يُمكنها التأثير في هذا المعامل.

تحوي البيانات المتاحة قيم معامل نجمة الطاقة، مما يعني أن المسألة المطروحة هي مسألة تعلّم آلة من نمط موجه عبر الانحدار supervised regression machine learning:

- تعليم موجه Supervised: تحوي البيانات المتاحة الميزات والنتيجة معًا، لذلك تدريب نموذج يتعلّم الربط بين هذه الميزات والنتيجة هو المطلوب في هذا النمط.
- عبر الانحدار Regression: لأن المعامل المطلوب هو قيمة رقمية حقيقية.

نهدف إلى بناء نموذج تعلّم دقيق يُمكنه التنبؤ بقيمة قريبة من القيمة الحقيقية لمعامل نجمة الطاقة، ويجب أن تكون نتائج النموذج المبني قابلةً للتفسير والفهم أيضًا. توجّه هذه الأهداف قراراتنا المتخَذة لاحقًا أثناء تحليلنا البيانات وبناء النموذج.

11.2 تنظيف البيانات

تُعَد عمليات تنظيف البيانات وتنسيقها في البداية من العمليات الأساسية، إذ أن بيانات العالم الحقيقة غالبًا ما تكون ناقصةً وتحوي العديد من القيم المتطرفة (بخلاف مجموعات بيانات التدريب المُعدّة لأهداف تعليمية مثل mtcars وiris). حيث تكون بيانات العالم الحقيقية في فوضى كبيرة، مما يستلزم تنظيفها وإعادة تشكيلها قبل البدء بتحليلها. قد تبدو مهمة تنظيف البيانات غير ممتعة إلا أنها ضرورية في جميع مسائل العالم الحقيقي.

نبدأ أولًا بتحميل البيانات ضمن إطار من البيانات DataFrame من مكتبة Pandas ومن ثم عرضها:

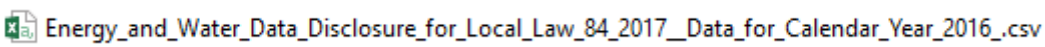
```
import pandas as pd
import numpy as np

# قراءة البيانات وتحميلها ضمن إطار من البيانات
data =
pd.read_csv('data/Energy_and_Water_Data_Disclosure_for_Local_Law_84_20
17__Data_for_Calendar_Year_2016_.csv')

# إظهار الجزء الأعلى من إطار البيانات
data.head()
```

يظهر لنا أولًا:

3rd Largest Property Use Type - Gross Floor Area (ft²)	Year Built	Number of Buildings - Self-reported	Occupancy	Metered Areas (Energy)	Metered Areas (Water)	ENERGY STAR Score	Site EUI (kBtu/ft²)	Weather Normalized Site EUI (kBtu/ft²)	Weather Normalized Site Electricity Intensity (kWh/ft²)	Weather Normalized Site Natural Gas Intensity (therms/ft²)
Not Available	1963	2	100	Whole Building	Not Available	Not Available	305.6	303.1	37.8	Not Available
Not Available	1969	12	100	Whole Building	Whole Building	55	229.8	228.8	24.8	2.4
Not Available	1924	1	100	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available	Not Available

يُظهِر الشكل مجموعةً جزئيةً من البيانات والتي تحوي 60 عمـودًا، ممـا يطـرح المسـألة الأولى التاليـة: نريـد التنبؤ بمعامل نجمة الطاقة Energy Star Score، إلا أننا لا نعرف أي عمود من الأعمدة يوافق هذا المعامل. قد تكون عدم معرفة معاني الأعمدة في بعض المسائل غير مهمة ويُمكن بناء نماذج دقيقة إلا أنـه في حالتنـا، وبمـا أن المطلوب تفسير النتائج فيجب فهم معاني بعض الأعمدة على الأقل.

لم يُرد كاتب المقال[1] عندما كُلّف بالعمل على هذا المشروع، سؤال أحـد عن معـاني الأعمـدة، إلا أنـه عنـدما تمعن في اسم الملف:



1 هذا الفصل مترجم عن سلسلة مقالات A Complete Machine Learning Project Walk-Through in Python لكاتبها Will Koehrsen كما ذكرنا في فصل التمهيد.

قرر البحث عن القانون المدعو بـ Local Law 84، مما قاده إلى هذه الصفحة التي تفرض على جميع مباني مدينة نيويورك اعتبارًا من حجم معين، تقديم تقرير عن استهلاك الطاقة فيها. وبمزيد من البحث توصل إلى تعريفات جميع الأعمدة.

يجب بدء العمل بتأن وروية دائمًا وعدم نسيان أي أمر قد يكون هامًا مثل معاينة اسم الملف، إذ لا نحتاج عمليًا لدراسة جميع الأعمدة، إلا أنه يجب أن نفهم معامل نجمة الطاقة والمُوصِّف على أساس نسبة مئوية (رقم صحيح بين 1 و100) لتقييم استهلاك الطاقة في سنة معينة، والذي يُقدّر لكل بناء في مدينة نيويورك، وهو مقياس نسبي يُستخدم لموازنة كفاءة الطاقة في المباني.

بعد حل المسألة الأولى نلتفت إلى المسألة الثانية وهي مشكلة القيم الناقصة، إذ تحوي البيانات المتاحة عبارة "غير متوفر Not Available" في الكثير من الخلايا التي لا تُعرَف قيمتها، ستُجبِر هذه القيمة النصية بايثون على تخزين العمود (ولو كان عمودًا رقميًا) كائن مثلًا object وذلك لأن مكتبة Pandas تعد جميع قيم العمود نصية بمجرد وجود قيمة نصية واحدة ضمن هذا العمود، ويُمكن معاينة نوع بيانات الأعمدة باستخدام طريقة إطار البيانات `dataframe.info`:

```
# معاينة بيانات الأعمدة والقيم غير الناقصة
data.info()
```

ويكون الناتج:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11746 entries, 0 to 11745
Data columns (total 60 columns):
Order                                                11746 non-null int64
Property Id                                          11746 non-null int64
Property Name                                        11746 non-null object
Parent Property Id                                   11746 non-null object
Parent Property Name                                 11746 non-null object
BBL - 10 digits                                      11735 non-null object
NYC Borough, Block and Lot (BBL) self-reported       11746 non-null object
NYC Building Identification Number (BIN)             11746 non-null object
Address 1 (self-reported)                            11746 non-null object
Address 2                                            11746 non-null object
Postal Code                                          11746 non-null object
Street Number                                        11622 non-null object
Street Name                                          11624 non-null object
Borough                                              11628 non-null object
DOF Gross Floor Area                                 11628 non-null float64
Primary Property Type - Self Selected                11746 non-null object
List of All Property Use Types at Property           11746 non-null object
Largest Property Use Type                            11746 non-null object
Largest Property Use Type - Gross Floor Area (ft²)   11746 non-null object
2nd Largest Property Use Type                        11746 non-null object
2nd Largest Property Use - Gross Floor Area (ft²)    11746 non-null object
3rd Largest Property Use Type                        11746 non-null object
3rd Largest Property Use Type - Gross Floor Area (ft²)  11746 non-null object
```

تُخزّن الكثير من الأعمدة الرقمية مثل كائن object مشابه للمساحة بالأقدام المربعة ft²، مما يستلزم تحويلها إلى رقم حقيقي float كي نستطيع إنجاز عمليات التحليل المطلوبة والتي لا يُمكن تطبيقها على السلاسل النصية.

سنعرض فيما يلي شيفرة بايثون التي تحول السلسلة النصية Not Available إلى القيمة p.nan أي "ليس رقمًا" not a number والتي يُمكن معاملتها مثل الأرقام ومن ثم تحويل العمود الموافق إلى نمط البيانات عدد حقيقي float:

```
# استبدال القيم الناقصة
data = data.replace({'Not Available': np.nan})

# المرور على الأعمدة عمودًا عمودًا
for col in list(data.columns):
    # اختيار الأعمدة التي يجب أن تكون رقميةً
    if ('ft²' in col or 'kBtu' in col or 'Metric Tons CO2e' in col or
'kWh' in
        col or 'therms' in col or 'gal' in col or 'Score' in col):
        # تحويل نمط البيانات إلى عدد
        data[col] = data[col].astype(float)
```

بعد الانتهاء من تعديل الأعمدة المناسبة إلى أرقام، ننتقل لمعاينة البيانات.

11.3 البيانات الناقصة والمتطرفة

من المشاكل الشائعة أيضًا وجود العديد من القيم الناقصة في البيانات لأسباب عديدة. يجب حذف هذه القيم أو إيجاد طريقة لملئها بقيم متوقعة قبل بناء نموذج التعلّم، لنبدأ أولًا بتحديد حجم هذه المشكلة لكل عمود (يُمكن العودة لرابط المشروع من أجل الحصول على الشيفرة).

نُظهر الجدول التالي والذي يحسب نسبة القيم الناقصة لكل عمود باستخدام الشيفرة الموجودة في هذا السؤال من موقع Stack Overflow.

```
Your selected dataframe has 60 columns.
There are 46 columns that have missing values.
```

	Missing Values	% of Total Values
Fuel Oil #1 Use (kBtu)	11737	99.9
Diesel #2 Use (kBtu)	11730	99.9
Address 2	11539	98.2
Fuel Oil #5 & 6 Use (kBtu)	11152	94.9
District Steam Use (kBtu)	10810	92.0
Fuel Oil #4 Use (kBtu)	10425	88.8
3rd Largest Property Use Type - Gross Floor Area (ft²)	10262	87.4
3rd Largest Property Use Type	10262	87.4
Fuel Oil #2 Use (kBtu)	9165	78.0
2nd Largest Property Use Type	8005	68.2
2nd Largest Property Use - Gross Floor Area (ft²)	8005	68.2
Metered Areas (Water)	4609	39.2

يجب توخي الحذر عند حذف عمود يحوي نسبةً كبيرةً من القيم الناقصة إذ قد يكون مفيدًا لنموذج التعلّم المطلوب. تعتمد العتبة والتي من فوقها نحذف العمود على المسألة المطروحة (تجد في هذا الرابط مناقشة لمسألة العتبة) وفي مشروعنا سنحذف أي عمود يحوي أكثر من 50% قيمًا ناقصةً.

نحذف القيم المتطرفة أيضًا في هذه المرحلة والتي يُمكن أن تظهر في البيانات نتيجة أخطاء طباعية أو أخطاء في الوحدات units المستخدمة، لكن يجب الملاحظة أنها في بعض الأحيان قد تكون صحيحةً إلا أنها بعيدةً كثيرًا عن باقي القيم. سنحذف القيم بالاعتماد على تعريف القيم المتطرفة البعيدة outliers التالي:

- أقل من الربع الأول – 3 × الانحراف الربيعي
- فوق الربع الثالث + 3 × الانحراف الربيعي

يُمكن العودة للمشروع للاطلاع على شيفرة حذف الأعمدة والقيم المتطرفة.

في نهاية هذه المرحلة، بقي لدينا أكثر من 11000 مبنى مع 49 ميزة.

11.4 تحليل البيانات الاستكشافي

بعد الانتهاء من المرحلة السابقة الضرورية جدًا رغم صعوبتها بعض الشيء، يُمكن لنا الانتقال إلى مرحلة تحليل البيانات الاستكشافي، والذي نعني به تطبيق بعض الحسابات الإحصائية ورسم بعض المخططات بهدف إيجاد الاتجاهات العامة trends والمتطرفات anomalies والأنماط patterns والعلاقات relationships الموجودة ضمن البيانات. وباختصار يُعدّ الهدف من هذه المرحلة هو استكشاف المعلومات المُضمّنة في

البيانات والتي يُمكن لها لعب دورًا مهمًا في توجيه خياراتنا عند بناء نماذج التعلّم، مثلًا: من هي المـيزات الأكـثر ارتباطًا مع الهدف؟

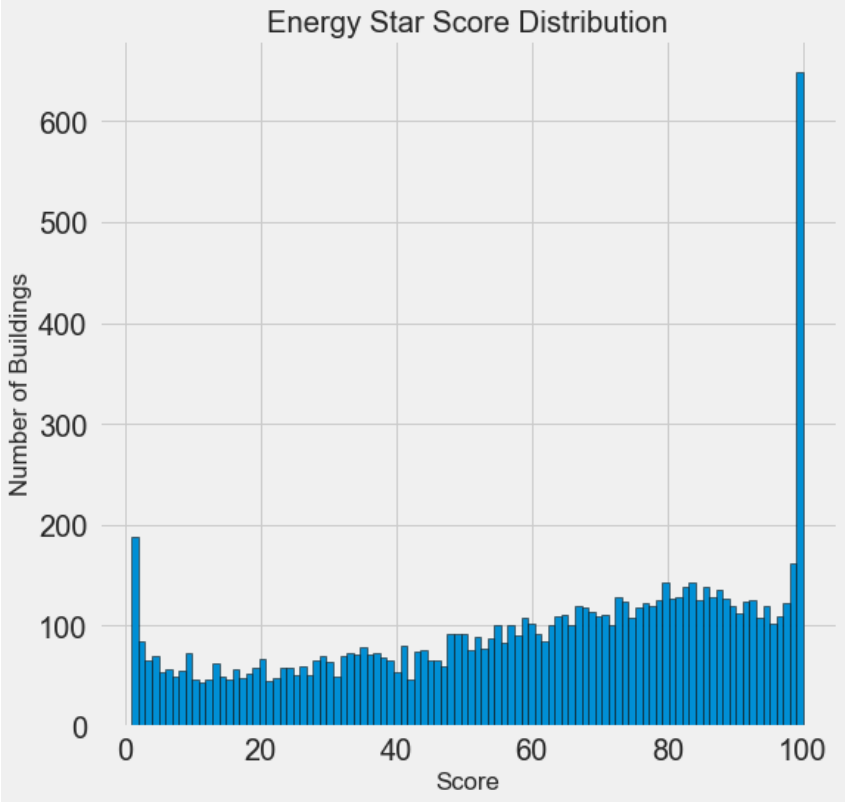
نعمل عادةً هذه الخطوة بشكل تدريجي، أي نبدأ من نظرة عامة كلية قد تقودنا في بعض الأحيان إلى التركـيز على بيانات محدّدة معينة.

11.4.1 رسم متغير وحيد

هدفنا التنبؤ بمعامل نجمة الطاقة (ندعوه 'score' في بياناتنا)، لذلك من الطبيعي البدء باستكشـاف تـوزع هذا المعامل، وبالطبع فأسهل شيء يُمكن اللجـوء لـه هـو رسـم المـدرج التكـراري Histogram، والـذي يسـمح بمعاينة توزع متغير لاسيما أنه يُمكن إنجاز ذلك بسهولة باستخدام المكتبة `matplotlib` كما يلي:

```
import matplotlib.pyplot as plt

# المدرج التكراري لمعامل نجمة الطاقة
plt.style.use('fivethirtyeight')
plt.hist(data['score'].dropna(), bins = 100, edgecolor = 'k')
plt.xlabel('Score'); plt.ylabel('Number of Buildings')
plt.title('Energy Star Score Distribution')
```

ويكون الناتج ما يلي:



يُظهِر هذا المدرج التكراري الملاحظة التالية المثيرة للانتباه: بالرغم من أن مُعامل نجمة الطاقة هو نسبة مئوية، أي يُفترض تشكيل توزيعًا منتظمًا تقريبًا، إذ يوجد اختلاف كبير بين عدد الأبنية ذات معامل يساوي 100 وعدد الأبنية ذات معامل يساوي 1، أي أنه كلما كان معامل نجمة الطاقة كبيرًا كانت كفاءة الطاقة أفضل.

إذا عُدنا إلى تعريف معامل نجمة الطاقة فسنجد أنه تقدير نسبي يُعطيه الأشخاص مما يُفسّر الظاهرة السابقة، لربما كان الطلب من سكان الأبنية تقدير استهلاكهم من الطاقة يشابه الطلب من طلاب صف تقويم أنفسهم في امتحان ما، وبالنتيجة فإننا نعتقد أن هذا المعيار غير موضوعي لقياس كفاءة الطاقة في الأبنية.

يُمكن التحري عن السبب في اختلاف قيم المعيار بين الأبنية عن طريق إيجاد الميزات المشتركة بين الأبنية التي لها تقريبًا نفس قيم المعيار؛ إلا أننا لن نفعل ذلك لأن مهمتنا هي التنبؤ بقيم هذا المعيار وليس إيجاد طرق أفضل لتقويم طاقة الأبنية. سنضع في تقريرنا النهائي ملاحظاتنا عن التوزيع غير المنتظم إلا أننا سنركز على مسألة التنبؤ.

11.4.2 إيجاد العلاقات

تهدف مرحلة تحليل البيانات الاستكشافية أساسًا إلى إيجاد العلاقات الكامنة بين الميزات وبين المعامل الهدف، حيث تكون الميزات المترابطة مع الهدف مفيدةً جدًا لنموذج التعلّم، لأنه يُمكن استخدامها في عملية التنبؤ بالهدف.

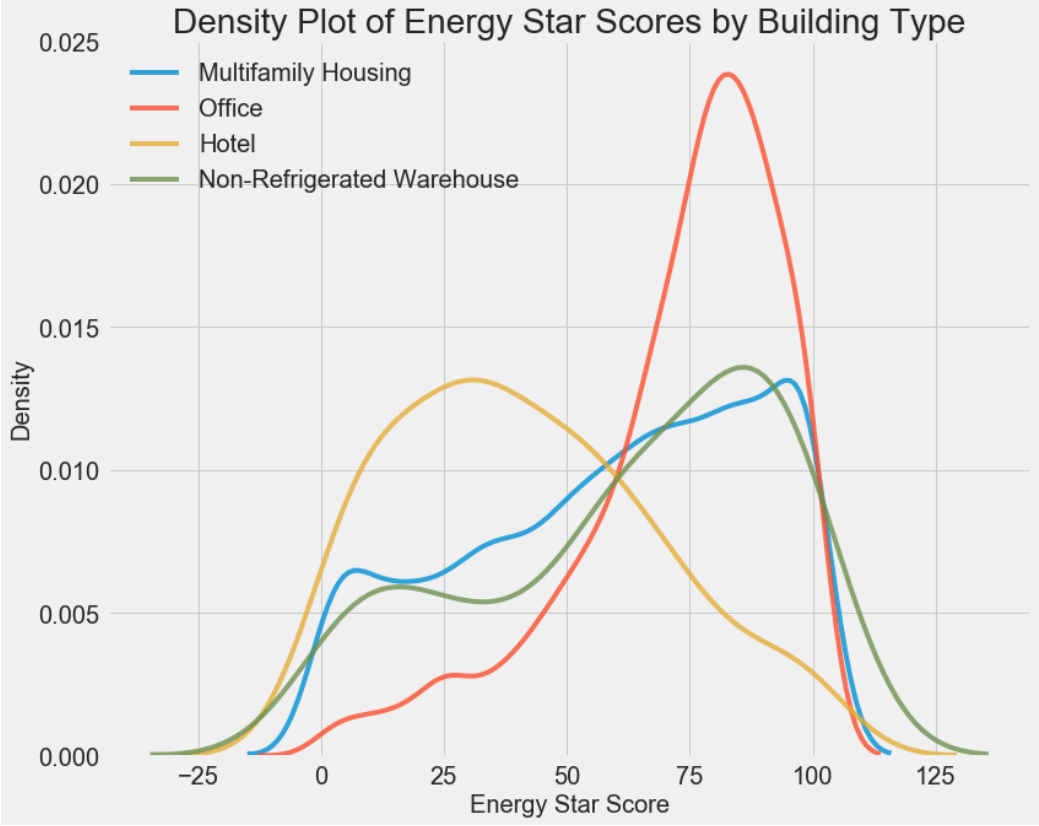
يُمكن رسم مخطط الكثافة لاستكشاف تأثير متغير فئوي (يأخذ مجموعةً مُحدّدةً من القيم) على الهدف وذلك باستخدام المكتبة seaborn، ويُمكن النظر لمخطط الكثافة مثل تنعيم للمدرج التكراري لأنه يُظهر توزع متغير وحيد أيضًا.

يُمكن تلوين مخطط الكثافة حسب الفئة لنرى كيف يؤثر المتغير الفئوي على التوزيع، حيث ترسم الشيفرة التالية مخططات الكثافة لمعامل الطاقة ملون حسب نوع المبنى (يقتصر على أنواع المباني التي تحتوي على أكثر من 100 سطر بيانات):

```
# إنشاء قائمة من الأبنية التي لها أكثر من 10 قياس
types = data.dropna(subset=['score'])
types = types['Largest Property Use Type'].value_counts()
types = list(types[types.values > 100].index)

# رسم توزيع المعامل وفق فئة البناء
figsize(12, 10)

# رسم كل بناء
for b_type in types:
```

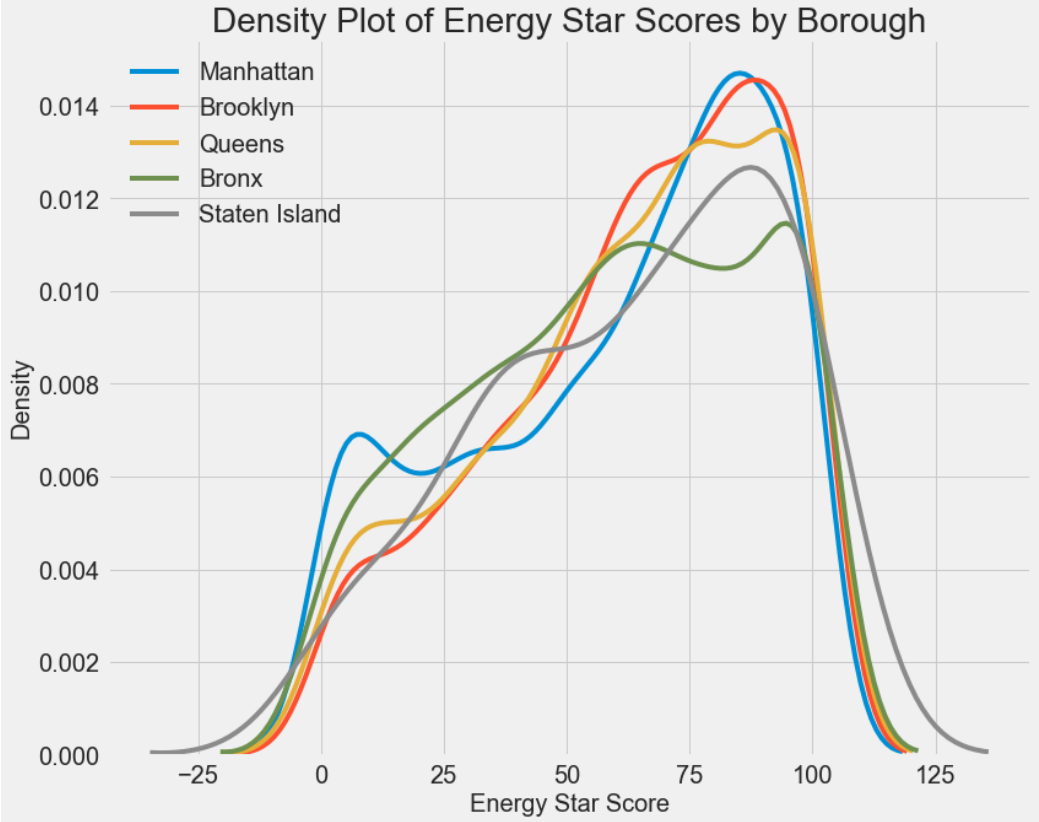
```
    # اختيار فئة البناء
    subset = data[data['Largest Property Use Type'] == b_type]

    # رسم مخطط الكثافة لمعامل الطاقة
    sns.kdeplot(subset['score'].dropna(),
                label = b_type, shade = False, alpha = 0.8);

# عنونة المخطط
plt.xlabel('Energy Star Score', size = 20); plt.ylabel('Density', size
= 20);
plt.title('Density Plot of Energy Star Scores by Building Type', size
= 28);
```

ويكون الناتج مخططات الكثافة حسب نوع البناء (الأزرق للمنازل السكنية، والأحمر للمكاتب، أما الأصفر فللفنادق، بينما الأخضر فالمستودعات غير المبردّة):



تُبرهن المخططات أعلاه على أن نوع المبنى يؤثر كثيرًا في معامل نجمة الطاقة، إذ يكون لمباني المكاتب معاملات عالية بينما تأخذ الفنادق القيم الأدنى، مما يستلزم تضمين نوع المبنى في نموذج التعلّم لأنه يؤثر بوضوح على الهدف بصفة متغير فئوي، وسيتوجب علينا استخدام ترميز ساخن واحد one-hot.

يُمكن تكرار العمل لدراسة العلاقة بين حي السكن ومعامل الطاقة حيث ينتج لدينا:

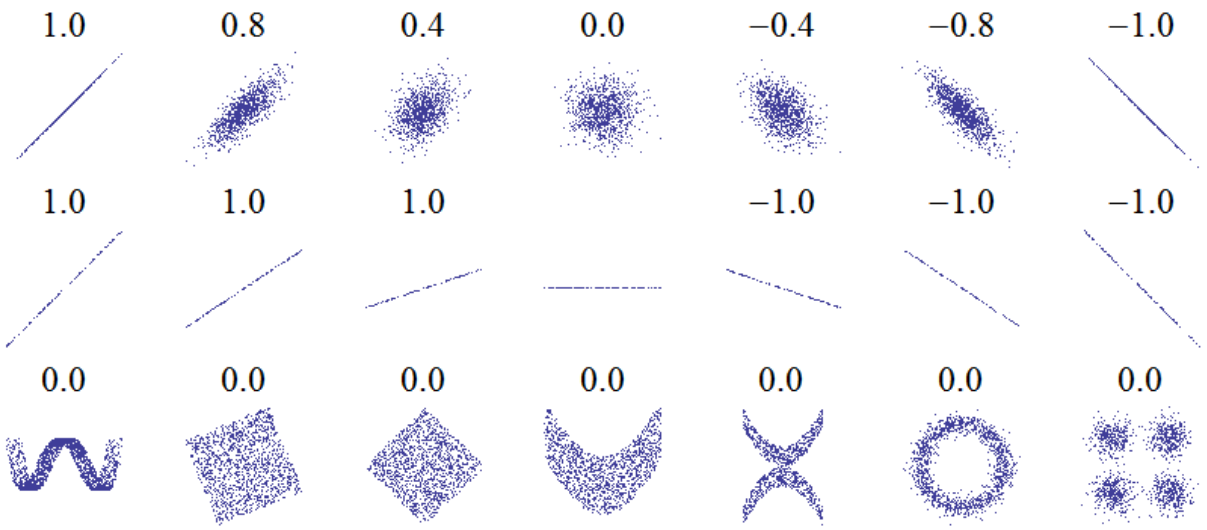


تُبرهن المخططات أعلاه عدم وجود تأثير كبير للحي على معامل نجمة الطاقة، ومع ذلك سنضمن الحي في نموذجنا نظرًا لوجود بعض الاختلافات الطفيفة بين الأحياء.

يُمكن استخدام معامل الارتباط لبيرسون Pearson correlation coefficient لتحديد نوع العلاقات بين المتغيرات:

- القيمة 1 تعني علاقة تناسب طردي.
- القيمة -1 تعني تناسب عكسي.
- أما القيمة 0 تعني عدم وجود أي علاقة.

يُبين الشكل التالي بعض قيم معامل الارتباط لبيرسون:



لا يكشف معامل الارتباط العلاقات غير الخطية وعلى الرغم من ذلك، فهو يبقى في البداية طريقةً جيدةً لاستكشاف ترابط المتغيرات مع بعضها البعض، يُمكن بسهولة إيجاد معاملات الارتباط باستخدام المكتبة Pandas:

```
# إيجاد قيم جميع الارتباطات وترتيبها تصاعديًا
correlations_data = data.corr()['score'].sort_values()
```

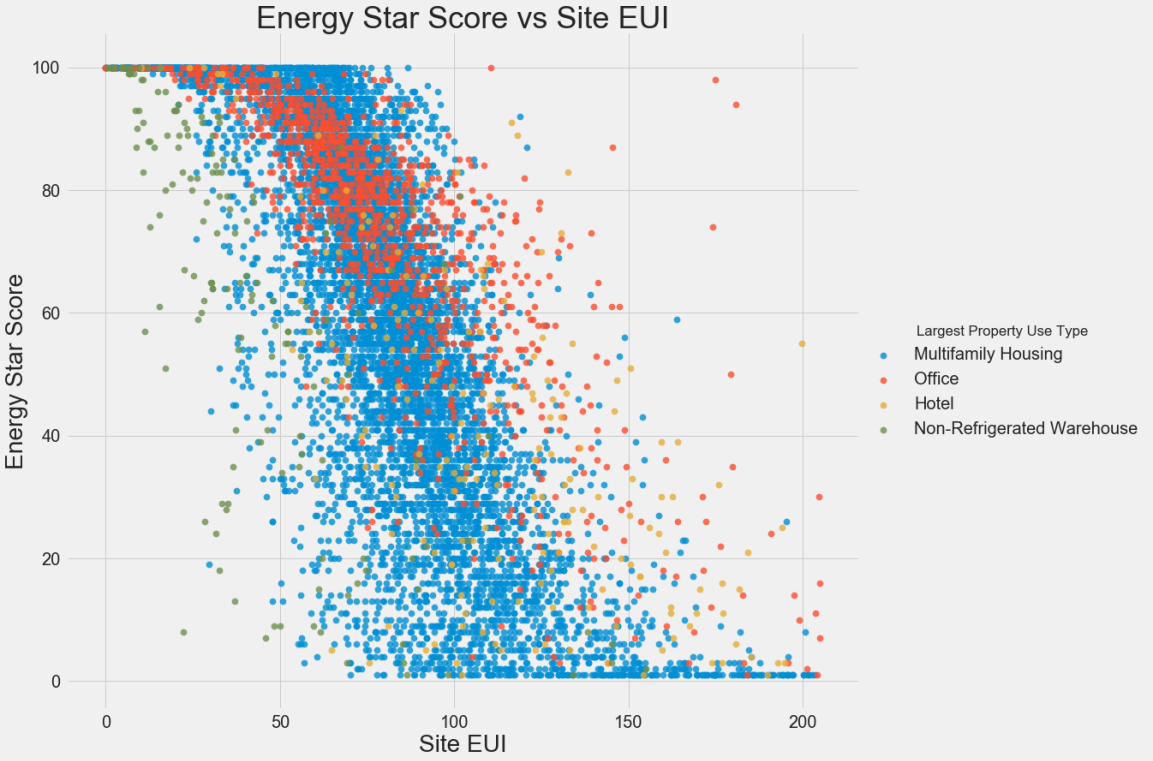
يُبين الشكل التالي علاقات الارتباط الناتجة السلبية (على اليسار، تناسب عكسي) والإيجابية (على اليمين، تناسب طردي):

```
Site EUI (kBtu/ft²)                                  -0.723864
Weather Normalized Site EUI (kBtu/ft²)               -0.713993
Weather Normalized Source EUI (kBtu/ft²)             -0.645542
Source EUI (kBtu/ft²)                                -0.641037
Weather Normalized Site Electricity Intensity (kWh/ft²) -0.358394
DOF Gross Floor Area                                  0.013001
Property GFA - Self-Reported (ft²)                    0.017360
Largest Property Use Type - Gross Floor Area (ft²)    0.018330
Order                                                 0.036827
Community Board                                       0.056612
Council District                                      0.061639
```

نلاحظ وجود العديد من الارتباطات السلبية (العكسية) بين الميزات ومعامل نجمة الطاقة الهدف. لنأخذ مثلًا الميزة EUI والتي هي كثافة استخدام الطاقة في المبنى أي الطاقة المستهلكة للمبنى مقسومًا على مساحة المبنى، بالطبع كلما كان هذا المقدار أصغر كانت الكفاءة أكبر (والتي يُعبّر عنها مقياس نجمة الطاقة).

11.4.3 رسم مخطط متغيرين

لمعاينة العلاقة بين متغيرين مستمرين (قيم رقمية حقيقية) يُمكن اللجوء إلى المخططات من النمط المبعثر scatterplots، كما يُمكن إضافة معلومات أخرى مثل المتغيرات الفئوية لتلوين نقاط المخطط، حيث يُبين المخطط التالي العلاقة بين كثافة استخدام الطاقة EUI ومعامل نجمة الطاقة ملونة حسب نوع البناء:



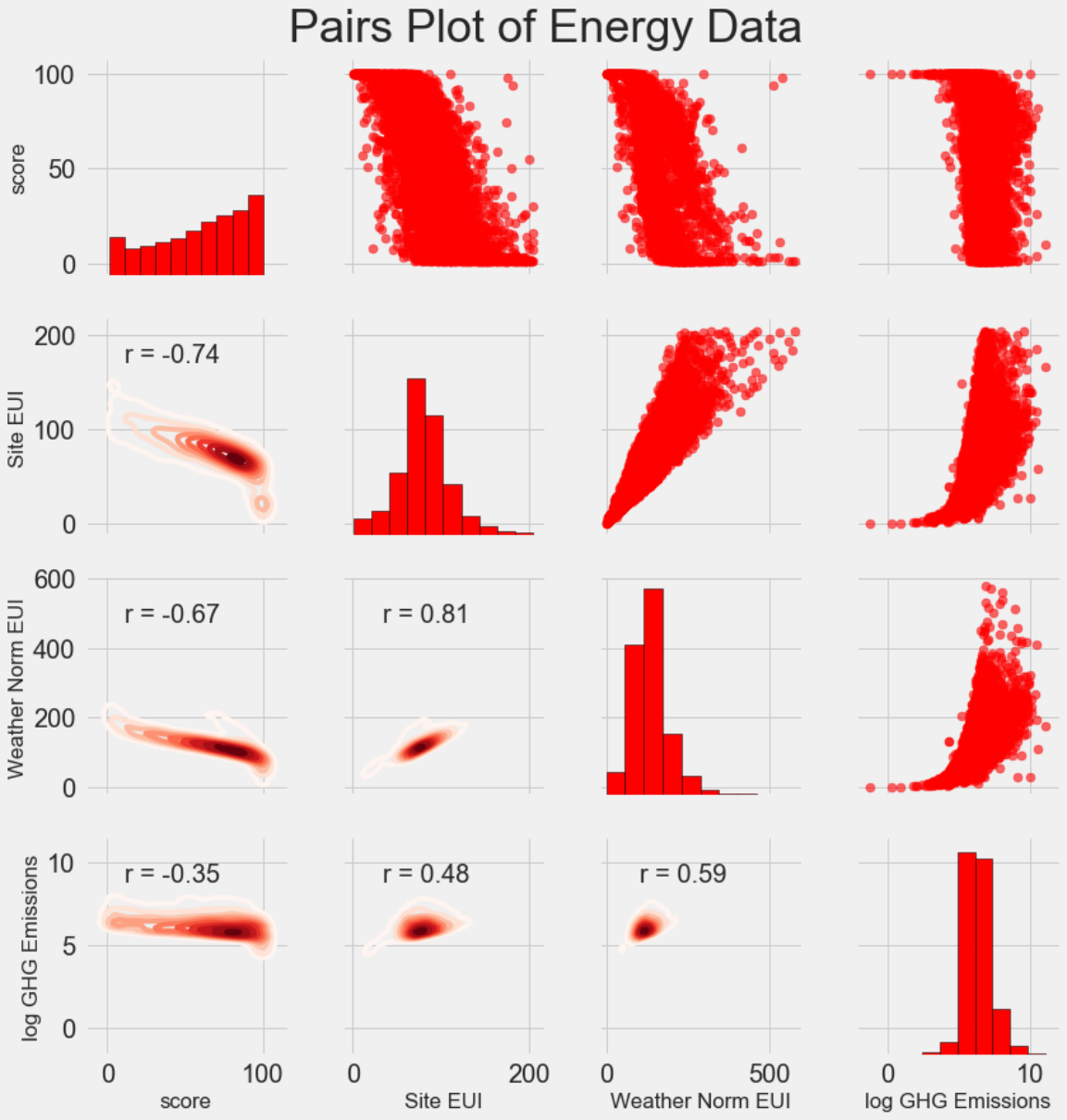
يُظهر هذا المخطط علاقةً عكسيةً بين كثافة استخدام الطاقة ومعامل نجمة الطاقة، فكلما انخفضت الكثافة ارتفع معامل نجمة الطاقة (يكون معامل الترابط حوالي -0.7).

أخيرًا، لنرسم مخطط باريس Paris Plot، والذي يوفر أداةً ممتازةً لاستكشاف البيانات، إذ يُمكّننا من معاينة العلاقات بين عدة أزواج من المتغيرات إضافة إلى توزيعات متغيرات وحيدة. سنستخدم مكتبة المعاينة seaborn والتابع `PairGrid` لرسم مخطط باريس وحيث نضع في المثلث العلوي المخططات المتناثرة وعلى القطر المدرجات التكرارية، بينما نضع مخططات كثافة النواة ثنائية الأبعاد 2D kernel density مع معاملات الترابط في المثلث السفلي.

```
# تحديد أعمدة المخطط
plot_data = features[['score', 'Site EUI (kBtu/ft²)',
                      'Weather Normalized Source EUI (kBtu/ft²)',
                      'log_Total GHG Emissions (Metric Tons CO2e)']]


# استبدال اللانهاية بليس رقمًا
plot_data = plot_data.replace({np.inf: np.nan, -np.inf: np.nan})
# إعادة تسمية الأعمدة
plot_data = plot_data.rename(columns = {'Site EUI (kBtu/ft²)': 'Site
EUI',
                                        'Weather Normalized Source
EUI (kBtu/ft²)': 'Weather Norm EUI',
```

```
                                        'log_Total GHG Emissions (Metric
Tons CO2e)': 'log GHG Emissions'})

# حذف القيم غير الرقمية
plot_data = plot_data.dropna()

# تابع حساب معامل الارتباط بين عمودين
def corr_func(x, y, **kwargs):
   r = np.corrcoef(x, y)[0][1]
   ax = plt.gca()
   ax.annotate("r = {:.2f}".format(r),
               xy=(.2, .8), xycoords=ax.transAxes,
               size = 20)

# إنشاء كائن الشبكة
grid = sns.PairGrid(data = plot_data, size = 3)

# الأعلى هو المخطط المبعثر
grid.map_upper(plt.scatter, color = 'red', alpha = 0.6)

# القطر هو المدرج التكراري
grid.map_diag(plt.hist, color = 'red', edgecolor = 'black')

# الأسفل هو مخطط الارتباط والكثافة
grid.map_lower(corr_func)
grid.map_lower(sns.kdeplot, cmap = plt.cm.Reds)

# العنوان الكلي للمخطط
plt.suptitle('Pairs Plot of Energy Data', size = 36, y = 1.02)
```

فنحصل على المخططات التالية:



ننظر لتقاطع سطر المتغير الأول مع عمود المتغير الثاني لمعاينة التفاعل بين متغيرين، مثلًا: لمعاينة الارتباط بين كثافة الكهرباء وفق الطقس `Weather Norm EUI` والمعامل الهدف `score`، ننظر إلى تقاطع سطر `Weather Norm EUI` مع عمود `score` لنجد معامل ارتباط قيمته -0.67 أي أن العلاقة بينهما عكسيةً.

يُمكن أن يُساعد هذا المخطط إضافة لمظهر المخطط السابق الرائع، في تحديد الميزات المفيدة في عملية النمذجة.

11.5 هندسة الميزات والاختيار

تختصر عملية هندسة الميزات وحسن اختيار المناسب منها الوقت اللازم لمعالجة مسائل تعلّم الآلـة. لنبـدأ أولًا بتعريف هاتين المهمتين:

- **هندسة الميزات Feature engineering**: وهي عملية استخراج أو إنشاء ميزات جديدة من البيانات الأساسية المتاحة، قد يستلزم ذلك إجراء بعض التحويلات على المتغيرات طبعًا، مثل إيجاد مربع القيمة أو اللوغاريتم الطبيعي لها، أو ترميز المتغيرات الفئوية لتُصبح قابلةً للاستخدام في النموذج. يُمكن النظـر لهندسة الميزات عمومًا على أنها عملية إنشاء ميزات جديدة.
- **اختيار الميزات Feature selection**: وهو عملية اختيـار المـيزات الأنسـب للنمـوذج، حيث نحـذف عادةً الميزات غـير المرتبطـة مـع الهـدف كي نُسـاعد النمـوذج على التعميم الأفضـل للبيانـات الجديـدة وللحصول على نموذج قابل للشرح والفهم. يُمكن النظـر لاختيـار المـيزات بشـكل عـام على أنهـا عمليـة حذف الميزات غير المرتبطة مع الهدف والإبقاء على الميزات المهمة فقط.

يتعلّم نموذج تعلّم الآلة فقط من البيانات التي نُقدّمها لـه، لـذا يُعـدّ التأكـد من أن البيانـات تتضـمن جميـع الميزات الهامة أمرًا أساسيًا، وإذا لم نغذي النموذج بالبيانات الصـحيحة، فإننـا نعـدّه ليفشـل ولا يجب أن نتوقـع منه أن يتعلّم عمليًا.

ننجز بمشروعنا في مرحلة هندسة الميزات بما يلي:

- ترميز المتغيرات الفئوية باستخدام ترميز ساخن واحد (ميزة الحي السكني وميزة نوع استخدام الملكية).
- إضافة اللوغاريتم الطبيعي للمتغيرات الرقمية.

يلزم استخدام هذا الترميز one-hot لتضمين متغير فئوي categorical variables في النموذج، إذ لا تفهم خوارزمية تعلّم الآلة بأن نوع البناء مكتبًا مثلًا، إذ يجب استخدام 1 إذا كان نوع البناء مكتبًا وإلا 0.

يُمكن أن تساهم إضافة ميزات جديدة ناتجة عن تطـبيق بعض التوابـع الرياضـية على المـيزات الرقميـة في مساعدة النموذج على تعلّم العلاقات غير الخطية الموجودة ضـمن البيانـات، ومن الممارسـات الشـائعة حسـاب الجذر التربيعي أو القوة من مرتبة معينة أو اللوغاريتم الطبيعي، والتي تعتمد عادةً على التجربة أو المعرفة بمجـال المسألة، ونضيف في مشروعنا اللوغاريتم الطبيعي لجميع الميزات الرقمية.

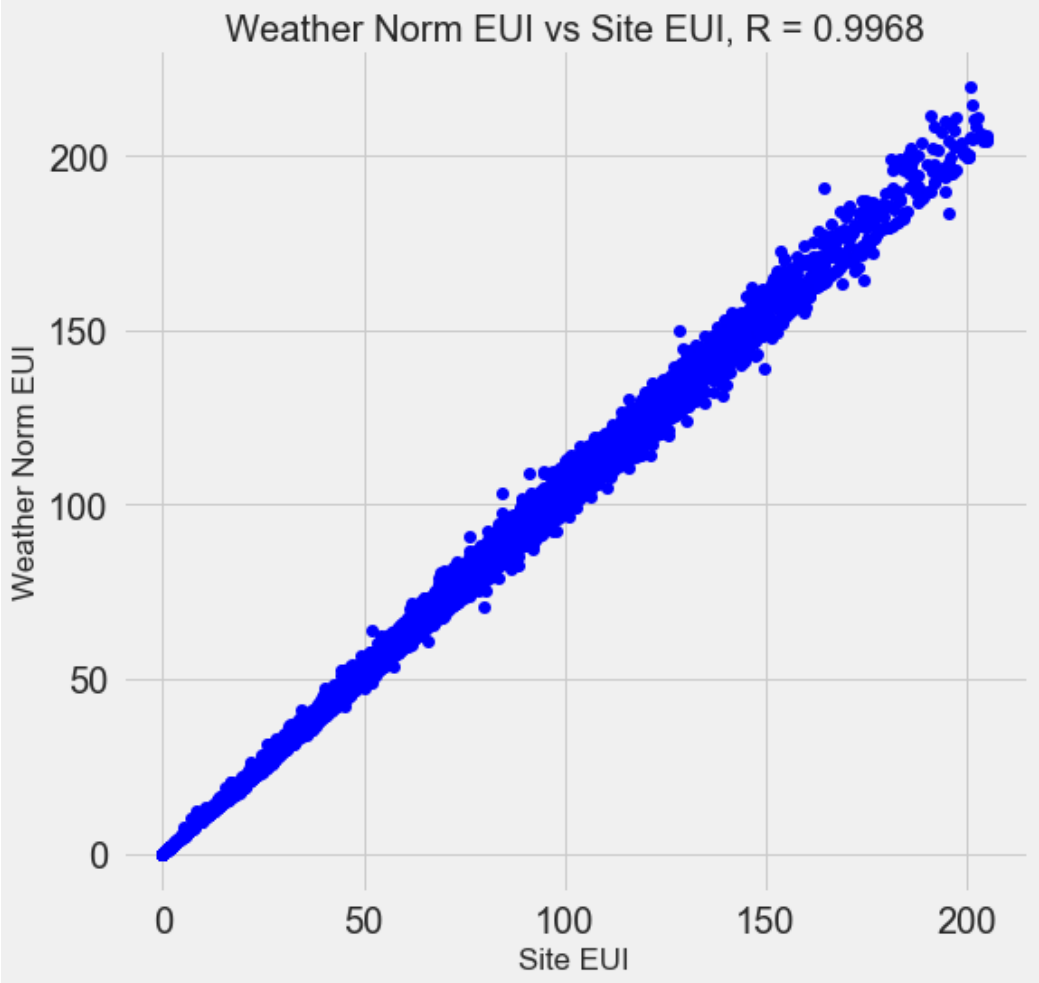
تختار الشيفرة التالية الميزات الرقمية ومن ثم يوجد اللوغـاريتم الطـبيعي لهـا ثم يختـار المـيزتين الفئويـتين ويرمزهما ثم يدمج الكل، قد يبدو أن هذا يتطلب الكثير من الجهد لكن استخدام المكتبة Pandas يُسهّل العمل:

```
# نسخ البيانات الأولية
features = data.copy()

# اختيار الأعمدة الرقمية
numeric_subset = data.select_dtypes('number')

# إنشاء أعمدة جديدة للوغاريتم الطبيعي للأعمدة الرقمية
for col in numeric_subset.columns:
  # تجاوز عمود معامل الطاقة
  if col == 'score':
        next
  else:
        numeric_subset['log_' + col] = np.log(numeric_subset[col])

# اختيار الأعمدة الفئوية
categorical_subset = data[['Borough', 'Largest Property Use Type']]

# ترميز واحد ساخن
categorical_subset = pd.get_dummies(categorical_subset)

# وصل إطاري العمل الناتجين
# اضبط قيمة العمود على 1 لإنشاء عمود ربط
features = pd.concat([numeric_subset, categorical_subset], axis = 1)
```

بعد هذه المعالجة سيكون لـدينا أكـثر من 11000 مبـنى مـع 110 عمـود (مـيزة)، حيث لن تكـون كـل هـذه الميزات مفيدةً في عملية التنبؤ بمعامل الطاقة لذا نحذف بعضها.

11.5.1 اختيار الميزات

ترتبط العديد من الميزات مع بعضها البعض بصورة وثيقة مما يعني حصول تكرار عمليًا، إذ يُـبين المخطـط التالي مثلًا ترابط ميزة كثافة الطاقة Site EUI مـع مـيزة كثافـة الكهربـاء Site EUI Weather Normalized، وبقيمة ترابط عالية جدًا 0.997.



نحذف عادةً الميزات المترابطة مع بعضها كثيرًا (تدعى علاقة خطية متداخلة collinear، مما يُساعد على الحصول على نموذج تعلّم عام قابل للتفسير. حيث نشير هنا إلى الارتباط بين الميزات مع بعضها البعض وليس إلى ارتباط الميزات مع الهدف.

توجد العديد من الطرق لحساب علاقات التداخل الخطية بين الميزات، حيث يُمكن استخدام عامل تضخم التباين variance inflation factor. سنستخدم في هذا المشروع معامل الارتباط لإيجاد الميزات المترابطة ونحذف ميزةً من أجل كل ميزتين إذا كان معدل الارتباط بينهما أكبر من 0.6، حيث يمكنك الحصول على الشيفرة أيضًا.

لاحظ أن القيمة المعتمدة 0.6 هي قيمة تجريبية اختيرت بعد العديد من التجارب، لنتذكر أن تعلّم الآلة هو حقل تجريبي وغالبًا ما نجرب عدة تجارب ومحاولات لاختيار الأفضل.

أخيرًا وبعد مرحلة اختيار الميزات تبقى لدينا 64 عمودًا وهدفًا وحيدًا.

```
# حذف الأعمدة ذات القيم غير الرقمية
features  = features.dropna(axis=1, how = 'all')
print(features.shape)
```

ويكون الناتج:

```
(11319, 65)
```

11.6 إنشاء خط الأساس

بعد أن أنهينا مرحلة تنظيف البيانات وتحليل البيانات الاستكشافي وهندسة الميزات، الآن يجب تحديد خط الأساس المبدئي Baseline قبل البدء بعملية بناء النموذج، وهو عبارة عن تخمين يُستخدم للحكم فيما إذا كانت نتائج نموذج التعلّم مقبولةً أم لا.

يُمكن في مسائل الانحدار اعتماد القيمة الأوسط median للهدف والمحسوبة لبيانات اختبار الخط الذي يجب لنموذج التعلّم تجاوزه وهو عمليًا هدف سهل الوصول.

نستخدم متوسط الخطأ المطلق MAE أي mean absolute error، والذي يقيس متوسط الفروقات بالقيمة المطلقة بين تنبؤ النموذج والقيم الحقيقية. توجد العديد من المقاييس في الواقع إلا أنه يُمكن اتباع نصيحة اعتماد مقياس واحد دومًا إضافةً إلى أن حساب متوسط الخطأ المطلق سهل وقابل للشرح والتفسير.

قبل حساب خط الأساس يجب تقسيم البيانات إلى مجموعتين هما مجموعة التدريب ومجموعة الاختبار:

- **مجموعة التدريب**: وهي مجموعة البيانات التي نُقدّمها للنموذج ليتعلّم منها، حيث تتألف من مجموعة الميزات إضافةً إلى القيمة الهدف (معامل الطاقة في حالتنا).
- **مجموعة الاختبار**: تُستخدم هذه المجموعة لتقويم كفاءة نموذج التعلّم حيث نختبر النموذج المدرب مع هذه البيانات، ومن ثم نوازن جواب النموذج مع الجواب المعروف لدينا في هذه البيانات، مما يسمح لنا بحساب قيمة متوسط الخطأ المطلق.

نقسم البيانات إلى 70% للتدريب و30% للاختبار:

```
# تقسيم البيانات إلى 70% للتدريب و30% للاختبار
X, X_test, y, y_test = train_test_split(features, targets,
                                        test_size = 0.3,
                                        random_state = 42)
```

يُمكن لنا الآن حساب خط الأساس:

```
# تابع حساب متوسط الخطأ المطلق
def mae(y_true, y_pred):
    return np.mean(abs(y_true - y_pred))
baseline_guess = np.median(y)
```

```
print('The baseline guess is a score of %0.2f' % baseline_guess)
print("Baseline Performance on the test set: MAE = %0.4f" %
mae(y_test, baseline_guess))
```

والذي يُعطي:

```
The baseline guess is a score of 66.00
Baseline Performance on the test set: MAE = 24.5164
```

مما يعني أن الخطأ من مرتبة 25% وهي عتبة سهلة التجاوز.

11.7 النتائج

استعرضنا إلى الآن الخطوات الثلاث الأولى لمعالجة مسألة تعلّم الآلة، بعد تحديد المشكلة نفعل ما يلي:

1. تنظيف وتنسيق البيانات الخام المتاحة.
2. تحليل استكشافي للبيانات بهدف التعرّف على البيانات.
3. تطوير مجموعةً من الميزات للاستخدام في نموذج التعلّم.

وفي النهاية، أجرينا عملية تحديد خط الأساس الذي سيسمح لنا بقبول نموذج التعلّم أو رفضه.

نعرض في القسم الثاني اختيار النموذج وتقييمه كيفية استخدام Scikit-Learn لتقييم نماذج التعلّم واختيار الأفضل منها، إضافةً إلى آليات معايرة المعاملات الفائقة للنموذج للوصول إلى نموذج أمثلي، أما في القسم الثالث تفسير وفهم النموذج فيعرض كيفية تفسير النموذج وعرض النتائج.

11.8 اختيار النموذج وتقييمه

نعرض في هذا القسم من الفصل تسلسل خطوات تنفيذ مسألة حقيقية في تعلّم الآلة بهدف تسليط الضوء على أهم الصعوبات التي يُمكنها مواجهة المبرمج وكيفية التغلب عليها.

شرحنا فيما سبق من الفصل الخطوات الثلاثة الأولى في مسألة تعلّم الآلة، وهي تنظيف البيانات أولًا، ومن ثم إجراء التحليل الاستكشافي لها بهدف اختيار الميزات المناسبة للمسألة، وأخيرًا تحديد خط الأساس الذي سنقيّم أداء نموذج التعلّم وفقه.

نتابع في هذا القسم عرض إنشاء عدة نماذج تعلّم مختلفة باستخدام بايثون، وذلك بهدف موازنة هذه النماذج لاختيار الأفضل منها، ومن ثم آلية ضبط ومعايرة المعاملات الفائقة للنموذج المختار بشكل أمثلي. وأخيرًا كيفية تقويم النموذج النهائي باستخدام بيانات الاختبار المتوفرة.

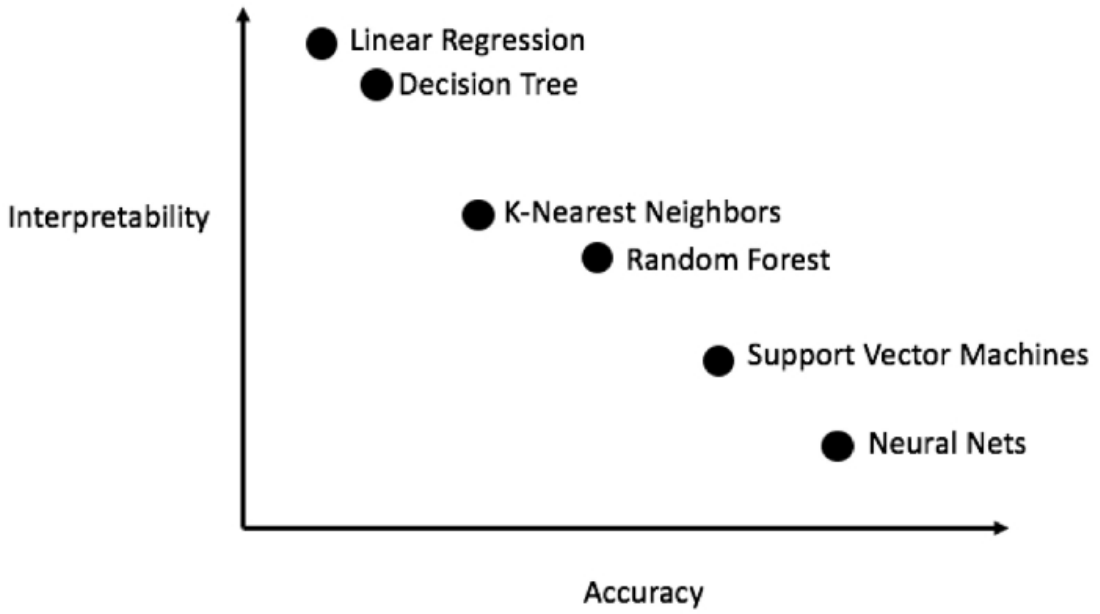
يُمكن تنزيل واستخدام ومشاركة الشيفرة المتاحة للعموم من مستودع GitHub.

نُذكّر بأننا نريد بناء نموذج يتنبأ بمعامل نجمة الطاقة Energy Star Score لمدينة نيويورك، مع التركيز على دقة النتائج وقابلية تفسيرها وذلك باستخدام بيانات الطاقة المتاحة للعموم. وبهذا تكون المسألة المطروحة مسألة تعلّم آلة من نمط موجه عبر الانحدار كما عرضنا سابقًا.

كي لا نغوص في الموازنات الكثيرة بين نماذج تعلّم الآلة المعروفة ونتمعن في مخططات موازنة النماذج سنجرب العديد منها عمليًا لاختيار الأمثل.

> نتذكر أن تعلّم الآلة مازال حقلًا تجريبيًا، وقد يكون من المستحيل معرفة النموذج الأمثل مسبقًا.

نبدأ عادةً بتجربة نموذج بسيط قابل للتفسير مثل نموذج الانحدار الخطي linear regression، وفي حال كانت دقته غير مقبولة، فسننتقل لنماذج أكثر تعقيدًا. يُبين الشكل التالي (المرسوم بشكل تجريبي) العلاقة بين الدقة accuracy وقابلية التفسير interpretability.



سنجرب النماذج الخمسة التالية:

- الانحدار الخطي Linear Regression.
- انحدار أقرب الجيران K-Nearest Neighbors Regression.
- انحدار الغابة العشوائية Random Forest Regression.
- الانحدار المعزز بالتدرج Gradient Boosted Regression.
- انحدار آلة متجهة الدعم Support Vector Machine Regression.

لن نغوص في الخلفية النظرية لهذه النماذج بل سـنُركز على برمجتهـا وتقويمهـا، ويُمكن العـودة للمزيـد من التفاصيل للمقال العربي المفاهيم الأساسية لتعلم الآلة، وللمرجع Introduction to Statistical Learning، وللكتاب Hands-On Machine Learning with Scikit-Learn and TensorFlow لأخذ معرفة أكثر.

11.8.1 احتساب القيم الناقصة

حذفنا في مرحلة تنظيف البيانـات الأعمـدة الـتي يتجـاوز فيهـا عـدد القيم الناقصـة 50% من قيم العمـود. وسيتعين علينا الآن ملء القيم الناقصة المتبقية لأن جميع خوارزميات التعلّم لا تعمل بوجود قيم ناقصة.

لنبدأ أولًا بعرض كل البيانات باستخدام الشيفرة التالية:

```
import pandas as pd
import numpy as np

# قراءة البيانات ووضعها في إطار بيانات
train_features = pd.read_csv('data/training_features.csv')
test_features = pd.read_csv('data/testing_features.csv')
train_labels = pd.read_csv('data/training_labels.csv')
test_labels = pd.read_csv('data/testing_labels.csv')

Training Feature Size:  (6622, 64)
Testing Feature Size:   (2839, 64)
Training Labels Size:   (6622, 1)
Testing Labels Size:    (2839, 1)
```

والذي يُظهر:

Year Built	Number of Buildings - Self-reported	Occupancy	Site EUI (kBtu/ft²)	Weather Normalized Site Electricity Intensity (kWh/ft²)	Weather Normalized Site Natural Gas Intensity (therms/ft²)	Water Intensity (All Water Sources) (gal/ft²)	Latitude	Longitude	Community Board	Census Tract	log_Direct GHG Emissions (Metric Tons CO2e)	log_Water Intensity (All Water Sources) (gal/ft²)	Borough_Staten Island
1950	1	100	126.0	5.2	1.2	99.41	NaN	NaN	NaN	NaN	6.088818	4.599253	0
1926	1	100	95.4	4.7	0.9	NaN	40.835496	-73.887745	3.0	161.0	5.384036	NaN	0
1954	1	100	40.4	3.8	0.3	NaN	40.663206	-73.949469	9.0	329.0	5.017280	NaN	0
1992	1	100	157.1	16.9	1.1	NaN	40.622968	-74.078742	1.0	27.0	6.510853	NaN	1
1927	1	100	62.3	3.5	0.0	28.65	40.782421	-73.972622	7.0	165.0	6.123589	3.355153	0
1929	1	90	52.9	9.7	0.2	4.80	40.725136	-74.004438	2.0	37.0	5.516649	1.568616	0
1942	1	100	66.8	3.0	0.6	67.14	40.637833	-73.973045	12.0	490.0	5.426271	4.206780	0
1938	1	100	78.4	5.7	NaN	30.73	40.776035	-73.964418	8.0	142.0	6.067036	3.425239	0
1959	1	100	63.0	3.4	0.5	41.96	NaN	NaN	NaN	NaN	6.170447	3.736717	0
1941	1	100	97.8	4.3	0.8	86.88	NaN	NaN	NaN	NaN	5.680855	4.464528	0
1922	1	100	55.4	4.5	0.0	NaN	40.762510	-73.970085	5.0	11203.0	1.335001	NaN	0

لاحظ أن كل قيمة "ليست رقمًا" NaN تُمثّل قيمةً ناقصة، وهنا سنستبدل ببساطة كل قيمة ناقصة بالقيمة الأوسط median لعمودها[2].

نُنشئ في الشيفرة التالية كائنًا Scikit-Learn من النمط `Imputer` مع إسناد استراتيجيته إلى القيمة الأوسط، ونُدرّب بعدها هذا الكائن باستخدام بيانات التدريب عن طريق تابع الملاءمة `imputer.fit`، ومن ثم نستخدمه لملء القيم الناقصة في كل من بيانات التدريب والاختبار باستخدام تابع التحويل `imputer.transform`. لاحظ أن القيم الناقصة في بيانات الاختبار تُستبدل بالقيمة الأوسط لبيانات التدريب.

نستخدم هذه الطريقة في الحساب لتجنب الوقوع في فخ تسرب بيانات الاختبار test data leakage وبهذا لا تختلط بيانات الاختبار مع بيانات التدريب أبدًا.

```
# إنشاء كائن الحساب مع استراتيجية القيمة الأوسط
imputer = Imputer(strategy='median')

# الملاءمة مع ميزات التدريب
imputer.fit(train_features)

# حساب كل من بيانات التدريب والاختبار
X = imputer.transform(train_features)
X_test = imputer.transform(test_features)

Missing values in training features:  0
Missing values in testing features:   0
```

يؤدي تنفيذ الشيفرة السابقة إلى الحصول على ميزات لها قيم حقيقية منتهية وبدون أي نقص فيها.

11.8.2 تحجيم الميزات

نعني بتحجيم الميزات Data processing الإجرائية العامة التي تُغير مجال قيم الميزة، وهي عملية ضرورية لأن الميزات تأتي عادةً بوحدات مختلفة، وبالتالي ستتوزع قيمها على مجالات مختلفة، تتأثر بعض النماذج كثيرًا ولاسيما تلك التي تحسب المسافة بين الأمثلة بمجالات هذه القيم، مثل نموذج آلة متجهة الدعم support vector machine ونموذج أقرب الجيران، بينما لا تتطلب بعض النماذج بالضرورة تحجيم الميزات، مثل نموذج الانحدار الخطي linear regression ونموذج الغابة العشوائية random forest. وتكون هذه الخطوة ضروريةً في جميع الأحوال عند الحاجة لموازنة نماذج التعلّم المختلفة.

2 يمكنك العودة للورقة البحثية A Comparison of Six Methods for Missing Data Imputation لمعاينة طرق أخرى لاستبدال القيم الناقصة.

نفعّل من أجل إرجاع قيم أي ميزة إلى المجال [0-1] من أجل كل قيمة في ميزة بطرح أصغر قيمة للميزة من هذه القيمة، ومن ثم القسمة على ناتج طرح أعلى قيمة للميزة من أصغر قيمة للميزة (مجال الميزة). تُدعى هذه العملية أحيانًا بالتسوية normalization أو التوحيد standardization.

يُمكن لنا برمجة العملية الحسابية السابقة طبعًا بسهولة، إلا أننا نستخدم الكائن `MinMaxScaler` في Scikit-Learn والذي ينجزها، وتُطابق شيفرة التحجيم شيفرة الحساب السابق مع استبدال scaler بـ imputer. سنؤكد على أننا نُدرّب باستخدام بيانات التدريب فقط ونحسب لكل البيانات بما فيها بيانات الاختبار كما نوّه سابقًا.

```
# إنشاء كائن التحجيم مع المجال 0 إلى 1
scaler = MinMaxScaler(feature_range=(0, 1))

# الملاءمة مع بيانات التدريب
scaler.fit(X)

# تحويل كل من بيانات التدريب والاختبار
X = scaler.transform(X)
X_test = scaler.transform(X_test)
```

تكون لكل ميزة بعد تنفيذ هذه الشيفرة قيمة صغرى هي 0 وقيمة عظمى هي 1.

يُستحسن فهم كل من عملية احتساب القيم الناقصة وعملية تحجيم الميزات جيدًا لأنهما موجودتان عمليًا في أي مشروع تعلّم آلة.

11.8.3 تنفيذ نماذج تعلم الآلة باستخدام Scikit-Learn

يُعدّ إنشاء النماذج وتدريبها ومن ثم استخدامها للتنبؤ أمرًا يسيرًا بعد أن أنهينا مراحل تنظيف البيانات وإعدادها. نستخدم المكتبة Scikit-Learn في بايثون والتي يتوفر لها توثيق ممتاز وطريقة بناء موحدة لكل نماذج التعلّم، مما يسمح بتنفيذ العديد من النماذج بسرعة.

تُبين الشيفرة التالية آلية إنشاء نموذج الانحدار المعزز بالتدرج Gradient Boosted Regression وتدريبه باستخدام تابع الملاءمة `fit`، ومن ثم اختباره باستخدام تابع التنبؤ `predict`.

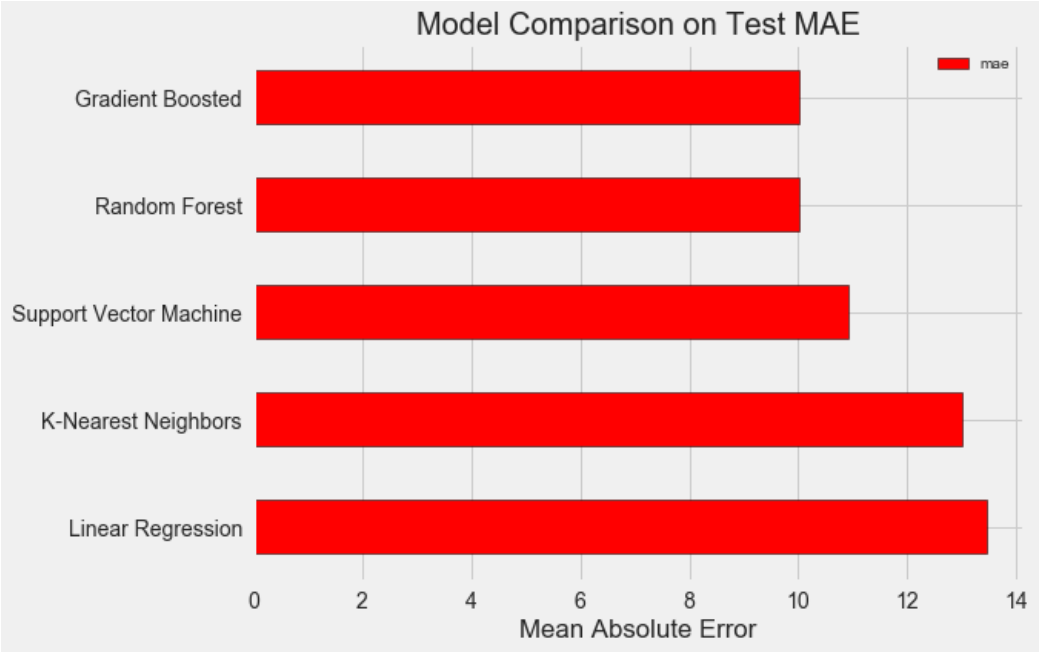
```
from sklearn.ensemble import GradientBoostingRegressor

# إنشاء النموذج
gradient_boosted = GradientBoostingRegressor()
```

```
# ملاءمة النموذج مع بيانات التدريب
gradient_boosted.fit(X, y)

# التنبؤ باستخدام بيانات الاختبار
predictions = gradient_boosted.predict(X_test)

# تقويم النموذج
mae = np.mean(abs(predictions - y_test))

print('Gradient Boosted Performance on the test set: MAE = %0.4f' %
mae)
```

حيث يكون الناتج:

```
Gradient Boosted Performance on the test set: MAE = 10.0132
```

لاحظ أن كل عملية من العمليات الأساسية (إنشاء وتدريب واختبار) تأخذ سطرًا واحدًا فقط، وبالتالي لبناء بقية النماذج يكفي تغيير اسم النموذج المطلوب في الشيفرة. يُبين الشكل التالي نتائج النماذج المختبرة:



تُثبت الأرقام في الشكل أن جميع نماذج تعلّم الآلة قابلة للتطبيق في مسألتنا لأنها جميعها تتميز بمتوسط قيم خطأ مطلق أصغر من خط الأساس المُحدّد لمسألتنا (وهو 24.5) والذي حُسب باستخدام القيمة الأوسط للهدف (معامل نجمة الطاقة).

لاحظ تقارب متوسط الخطأ المطلق لكل من نموذج الانحدار المعزز MAE=10.013 ونموذج الغابة العشوائية MAE=10.014. قد لا تكون الموازنة في هذه المرحلة عادلةً بين النماذج لاسيما بالنسبة لنموذج آلة

متجهة الدعم support vector machine لأننا تركنا القيم الافتراضية لمعاملات النماذج دون أي معايرة أو ضبط.

11.9 معايرة المعاملات الفائقة وصولا لنموذج أمثلي

يُمكن الوصول لنموذج أمثلي بمعايرة معاملاته الفائقة وفق معطيات المشروع. لنُبين أولًا الفرق بين المعاملات الفائقة لنموذج والمعاملات الأخرى له:

- **المعاملات الفائقة hyperparameters**: هي إعدادات خوارزمية التعلّم قبل التدريب (والتي وضعها مصممو الخوارزمية) مثل عدد الجيران في نموذج أقرب الجيران أو عدد الأشجار في نموذج الغابة العشوائية.
- **المعاملات parameters**: هي المعاملات التي يتعلّمها النموذج أثناء التدريب مثل أوزان نموذج الانحدار الخطي.

تؤثر عملية معايرة المعاملات الفائقة على أداء النموذج لاسيما لجهة التوزان المطلوب بين مشكلة قلة التخصيص underfitting ومشكلة فرط التخصيص overfitting، واللتان تؤديان إلى نموذج غير قادر على تعميم أمثلة التدريب، وبالتالي لن يتمكن من التنبؤ مع معطيات جديدة. ويُمكن العودة إلى فصل التحديات الرئيسية في تعلم الآلة من أكاديمية حسوب للمزيد من التفصيل حول هاتين المشكلتين.

تظهر مشكلة قلة التخصيص عندما لا يكون للنموذج درجات حرية كافية ليتعلّم الربط بين الميزات والهدف، وبالتالي يكون له انحياز كبير نحو قيم معينة للهدف. يُمكن تصحيح قلة التخصيص بجعل النموذج أكثر تعقيدًا، بينما تظهر مشكلة فرط التخصيص عندما يخزن النموذج بيانات التدريب، فيكون له بالتالي تباين كبير، والذي يُمكن تصحيحه بالحد من تعقيد النموذج باستخدام التسوية regularization.

تكمن المشكلة في معايرة المعاملات الفائقة بأن قيمها المثلى تختلف من مسألة لأخرى، وبالتالي تُعَد الطريقة الوحيدة للوصول لهذه القيم المثلى هي تجريب قيم مختلفة مع كل مجموعة بيانات تدريب جديدة.

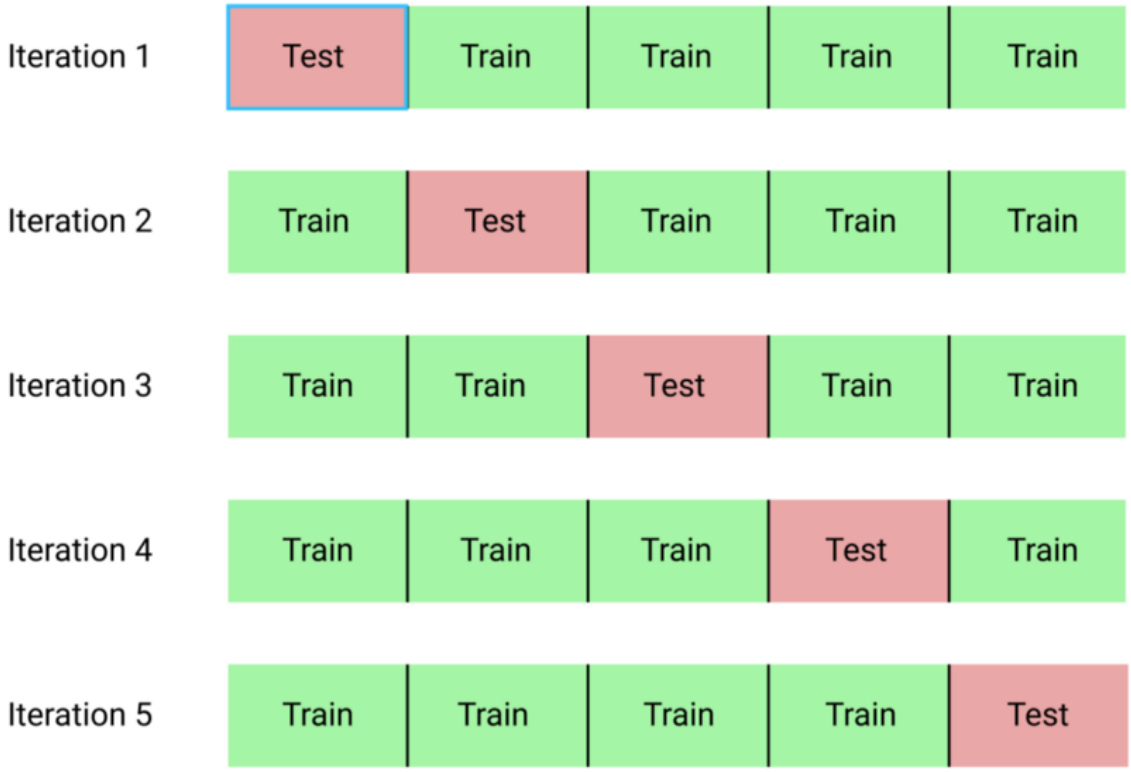
يحاول الكثيرون مثل مخبر Epistasis الوصول للقيم المثلى باستخدام خوارزميات مناسبة مثل الخوارزميات الجينية، إذ يوفر Scikit-Learn لحسن الحظ العديد من الطرق لتقويم المعاملات الفائقة وبالتالي سنعتمد في مشروعنا عليها دون تعقيد الأمور أكثر.

11.9.1 البحث العشوائي مع التقييم المتقاطع

تُدعى الطريقة التي سنستخدمها في إيجاد القيم المثلى بالبحث العشوائي مع التقويم المتقاطع أيrandom search with cross validation:

- **البحث العشوائي**: نُعرّف شبكة grid من قيم المعاملات الفائقة، ومن ثم نختار تركيبات مختلفة منها عشوائيًا، أي أننا لا نختار كل التركيبات الممكنة مثل سلوك البحث الشبكي (يكون أداء البحث العشوائي لحسن الحظ قريب من البحث الشبكي مع تخفيض كبير في الزمن اللازم).
- **التقييم المتقاطع**: وهو الطريقة المستخدمة لتقويم مجموعة قيم محدّدة للمعاملات الفائقة، عوضًا عن تقسيم البيانات إلى بيانات للتدريب وبيانات للتقويم، مما يُخفّض من البيانات التي يُمكن لنا استخدامها للتدريب، كما نستخدم التقويم المتقاطع مع عدد محدّد من الحاويات K-Fold. إذ تُقسم بيانات التدريب إلى عدد K من الحاويات ومن ثم ننكرر ما يلي K مرة: في كل مرة ندرب النموذج مع بيانات K-1 حاوية، ومن ثم تقويمه مع بيانات الحاوية K. حيث يكون مقياس الأداء النهائي هو متوسط الخطأ لكل التكرارات.

يوضح الشكل التالي فكرة التقويم المتقاطع من أجل K=5:



يُمكن تلخيص خطوات البحث العشوائي مع التقويم المتقاطع كما يلي:

1. إعداد شبكة من المعاملات الفائقة.
2. اختيار مجموعة عشوائية من تركيبات قيم المعاملات الفائقة.
3. إنشاء نموذج مع قيم المعاملات المختارة.

4. تقويم النموذج باستخدام التقويم المتقاطع.
5. اختيار تركيب قيم المعاملات ذو الأداء الأفضل.

لن نبرمج هذه الخطوات طبعًا لأن الكائن `RandomizedSearchCV` في Scikit-Learn ينجز كل ذلك.

11.9.2 طرق التعزيز المتدرج مرة أخرى

يُعدّ نموذج الانحدار المعزز بالتدرج Gradient Boosted Regression (المستخدم في حالتنا) من طرق المجموعات، بمعنى أنه يدرب مجموعةً من المتدربين الضعفاء (أشجار قرار) تسلسليًا، وبحيث أن كل متدرب يستفيد من أخطاء المتدرب السابق، بخلاف نموذج الغابة العشوائية والذي يدرب مجموعةً من المتدربين الضعفاء على التوازي ومن ثم يتنبأ عن طريق الانتخاب بينهم.

تصدرت طرقُ التعزيز طرقَ تعلّم الآلة في السنوات الأخيرة لاسيما بعد فوزها في العديد من مسابقات التعلّم، إذ تستخدم طريقة التعزيز المتدرجِ النزولَ المتدرجَ Gradient Descent لتخفيض تابع الكلفة عن طريق تدريب المتدربين واحدًا تلو الآخر، بحيث يستفيد كل متدرب من أخطاء المتدرب السابق.

تسبق بعض المكتبات أداء المكتبة Scikit-Learn المستخدمة في مشروعنا مثل XGBoost، إلا أننا سنحافظ على استخدامها مع بياناتنا الصغيرة نسبيًا نظرًا لدقتها الواضحة.

11.9.3 معايرة المعاملات الفائقة في نموذج الانحدار المعزز بالتدريج

يُمكن العودة لتوثيق Scikit-Learn لتفاصيل المعاملات الفائقة، والتي سنجد القيم الأمثل لها:

- `loss` تابع الخسارة والذي يجب تخفيضه.
- `n_estimators` عدد المتدربين الضعفاء (أشجار القرار).
- `max_depth` العمق الأعظم لشجرة القرار.
- `min_samples_leaf` العدد الأصغر للأمثلة في ورقة شجرة قرار.
- `min_samples_split` العدد الأصغر المطلوب لتقسيم عقدة في شجرة قرار.
- `max_features` العدد الأعظم للميزات المطلوب لتقسيم عقدة في شجرة قرار.

قد لا نجد أحدًا يفهم كيفية تأثير هذه المعاملات على بعضها البعض، ولابدّ من التجريب للوصول إلى القيم المثلى لها.

نبني في الشيفرة التالية شبكةً من قيم المعاملات الفائقة، حيث نُنشئ كائن `RandomizedSearchCV` ونبحث باستخدام 4 حاويات للتقويم المتقاطع مع 25 تركيبةً مختلفةً لقيم المعاملات الفائقة:

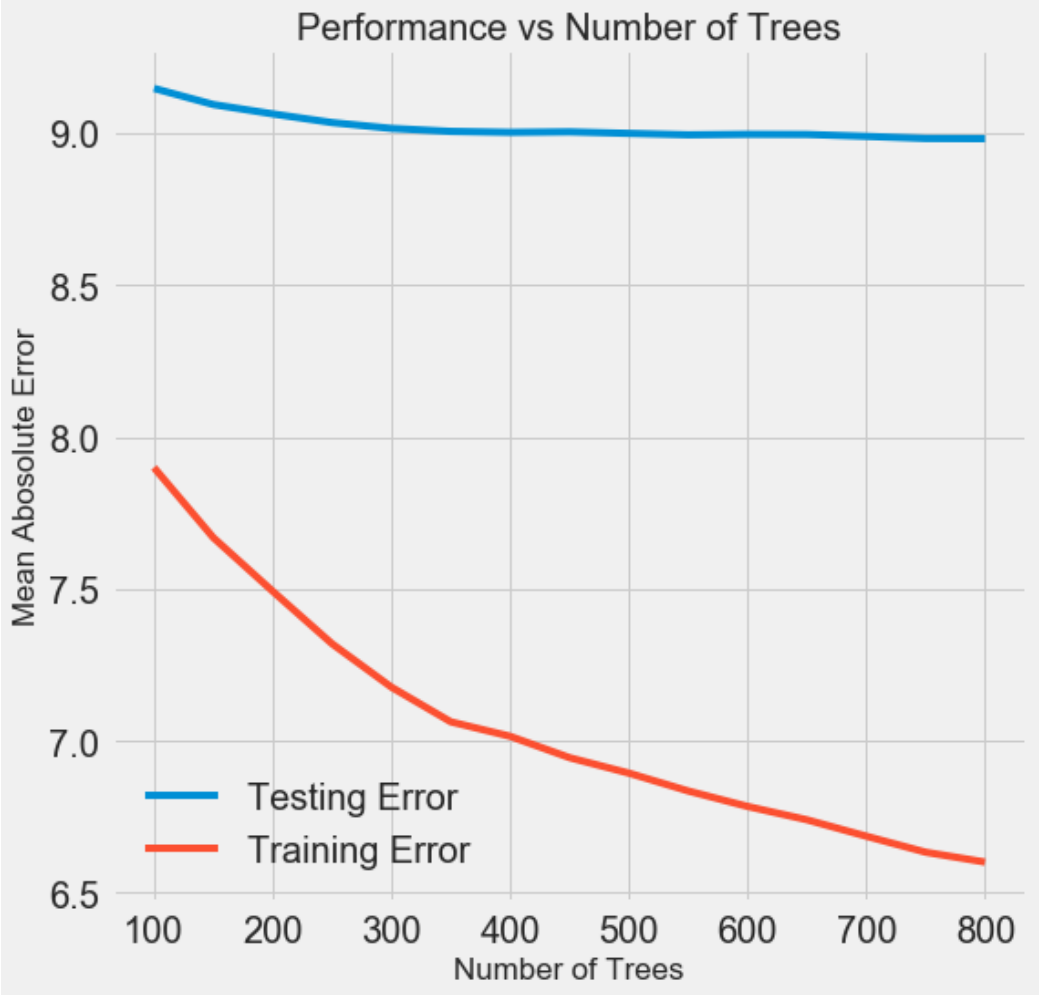
```
# تابع الخسارة المطلوب تخفيضه
loss = ['ls', 'lad', 'huber']
# عدد أشجار القرار المستخدمة
n_estimators = [100, 500, 900, 1100, 1500]
# العمق الأعظم لكل شجرة
max_depth = [2, 3, 5, 10, 15]
# عدد الأمثلة الأصغر لكل ورقة
min_samples_leaf = [1, 2, 4, 6, 8]
# عدد الأمثلة الأصغر لتقسيم عقدة
min_samples_split = [2, 4, 6, 10]
# العدد الأعظم للميزات المستخدمة لتقسيم عقدة
max_features = ['auto', 'sqrt', 'log2', None]
# تعريف شبكة المعاملات الفائقة للبحث فيها
hyperparameter_grid = {'loss': loss,
                       'n_estimators': n_estimators,
                       'max_depth': max_depth,
                       'min_samples_leaf': min_samples_leaf,
                       'min_samples_split': min_samples_split,
                       'max_features': max_features}
# إنشاء نموذج معايرة المعاملات الفائقة
model = GradientBoostingRegressor(random_state = 42)

# إعداد البحث العشوائي مع 4 حاويات للتقويم المتقاطع
random_cv = RandomizedSearchCV(estimator=model,

param_distributions=hyperparameter_grid,
                               cv=4, n_iter=25,
                               scoring = 'neg_mean_absolute_error',
                               n_jobs = -1, verbose = 1,
                               return_train_score = True,
                               random_state=42)
# الملاءمة مع بيانات التدريب
random_cv.fit(X, y)
```

نعاين الكائن `RandomizedSearchCV` بعد الانتهاء من البحث لنعرف النموذج الأمثل:

```
# إيجاد أفضل تركيبة ممكنة من القيم
random_cv.best_estimator_
GradientBoostingRegressor(loss='lad', max_depth=5,
                          max_features=None,
                          min_samples_leaf=6,
                          min_samples_split=6,
                          n_estimators=500)
```

يُمكن لنا استخدام هذه النتائج لإنجاز البحث الشبكي باستخدام معاملات للشبكة قريبة من هذه المعـاملات المثلى. لن يؤدي المزيد من البحث والمعايرة إلى تحسين ملحوظ على الأرجح، وذلك لأن عملية هندسة المـيزات التي عملنا عليها في البداية كان لها التأثير الأكبر في تحسين نتائج النموذج، إذ يطبـق في الحقيقـة أيضًـا قـانون تناقص العوائد law of diminishing في تعلّم الآلة والذي يقول: لقد أعطت هندسة المـيزات الأثـر الأكـبر في تحسين النموذج ولن تؤدي معايرة المعاملات الفائقة إلا للقليل من التحسن.

يُمكن تجربة تغيير عدد المتدربين مثلًا (أشجار القرار)، مع الحفـاظ على بقيـة قيم المعـاملات الفائقـة ثابتـةً ممـا يسمح لنا بمعاينة تأثير هذا المعامل. يُمكن الاطلاع على الشيفرة الموافقة والتي تعطي النتائج التالية:



لاحظ أنه كلما ازداد عدد الأشجار يقل الخطأ سواءً في بيانات التدريب أو في بيانات الاختبار، كما يتناقص الخطأ في التدريب بسرعة أكبر من الخطأ في الاختبار. لكن يبدو أن نموذجنا ذو فرط تخصيص overfitting، حيث أن له أداءً ممتازًا مع بيانات التدريب لا يصل له مع بيانات الاختبار.

نتوقع دائمًا بعض الانخفاض في الأداء مع بيانات الاختبار عن بيانات التدريب (لا تنسى أن النموذج يرى كل بيانات التدريب)، لكن وجود فارق كبير في الأداء بين التدريب والاختبار يعني مشكلة فرط التخصيص، والتي يُمكن حلها بزيادة بيانات التدريب أو تخفيض تعقيد النموذج عبر المعاملات الفائقة. نحافظ على قيم المعاملات الفائقة كما هي ونترك للقارئ محاولة البحث عن حل لمشكلة فرط التخصيص.

نعتمد من أجل النموذج النهائي القيمة 800 لمعامل عدد الأشجار لأنها تُعطي أقل خطأ نتيجة التقويم المتقاطع.

11.10 التقويم باستخدام بيانات الاختبار

سنستخدم بيانات الاختبار والتي أخفيناها تمامًا عن نموذج التعلّم أثناء مرحلة التدريب، إذ سيُعطي اختبار النموذج طبعًا مع هذه البيانات مؤشرًا لأداء النموذج مع البيانات الحقيقية لاحقًا.

تحسب الشيفرة التالية متوسط الخطأ المطلق (معيار الأداء) لكل من النموذج الأولي والنموذج النهائي الذي حصلنا عليه بعد معايرة المعاملات الفائقة:

```
# التنبؤ باستخدام النموذج الأولي والنهائي
default_pred = default_model.predict(X_test)
final_pred = final_model.predict(X_test)

Default model performance on the test set: MAE = 10.0118.
Final model performance on the test set:   MAE = 9.0446.
```

أدت معايرة المعاملات الفائقة إلى تحسين الأداء بنسبة 10% تقريبًا وهي نسبة قد تكون مهمةً في التطبيقات الحقيقية رغم الوقت الذي خصصناه لتنفيذها.

يُمكن معايرة الوقت الذي يستغرقه تدريب النموذج باستخدام التعليمة السحرية `%timeit` في محيط التطوير المستخدم Jupyter Notebooks:

```
%%timeit -n 1 -r 5
default_model.fit(X, y)
```

ويكون ناتج الشيفرة مثلًا:

```
1.09 s ± 153 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

تُظهر النتيجة أن تدريب النموذج الأولي يحتاج لحوالي ثانية واحدة فقط وهو زمن معقول جدًا؛ أما النموذج النهائي فهو ليس بهذه السرعة إذ يستغرق حوالي 12 ثانيةً:

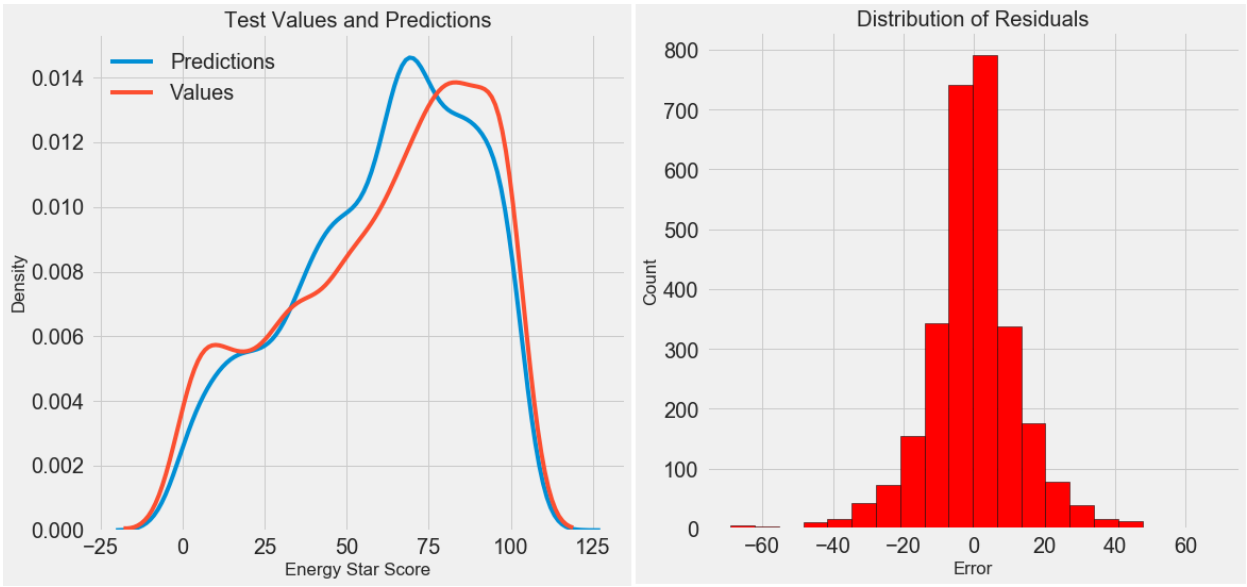
```
%%timeit -n 1 -r 5
final_model.fit(X, y)
```

حيث يكون ناتج الشيفرة مثلًا:

```
12.1 s ± 1.33 s per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

يُعزز ذلك مبدأ المقايضة trade-offs العام في تعلّم الآلة، إذ نسعى دومًا لمقايضة الدقة مع قابلية التفسير وانحراف النتائج مع تباينها، والدقة مع زمن التنفيذ وغيرها. ربما يبدو 12 ضعفًا رقمًا كبيرًا نسبيًا إلا أنه ليس بهذا السوء مطلقًا.

يُمكننا الآن وبعد حصولنا على التنبؤات النهائية فحص فيما إذا يوجد أي انحراف أو مشكلة ما فيها. يُبين الشكل التالي (على اليسار) مخطط الكثافة لتوقعات النموذج (الأزرق) مع القيم الحقيقية (الأحمر)؛ أما على اليمين فيَظهر المدرج التكراري لقيمة الخطأ المرتكب.



يُبين مخطط الكثافة أن توقعات النموذج تتبع تقريبًا توزع القيم الحقيقية على الرغم من أن قمة كثافة التوقعات قريبة من القيمة الأوسط لقيم التدريب (66)، بينما تكون قمة كثافة القيم الحقيقية أقرب للقيمة العظمى (100)؛ أما المدرج التكراري فيُظهر توزعًا طبيعيًا تقريبًا للخطأ المرتكب مع ملاحظة وجود قيم سالبة للخطأ مما يعني توقعات أقل من الحقيقة. كما سنسلط الضوء لاحقًا على النتائج.

11.11 النتائج

تابعنا في هذا الفصل استعراض العديد من الخطوات في معالجة مسألة تعلّم الآلة:

- احتساب القيم الناقصة وتحجيم الميزات.
- تقويم وموازنة عدة نماذج تعلّم.
- معايرة المعاملات الفائقة باستخدام البحث العشوائي الشبكي والتقويم المتقاطع.
- تقويم النموذج الأفضل مع بيانات الاختبار.

تُبرهن النتائج على إمكانية استخدام البيانات المتوفرة لبناء نموذج تعلّم الآلة بهدف توقع معامل نجمة الطاقة للمباني، إذ يتنبأ نموذج التعلّم من نمط الانحدار المعزز بالتدرج بمعامل نجمة الطاقة مع خطأ من مرتبة 9.1 نقطةً بالنسبة للقيمة الحقيقية في بيانات الاختبار. وقد ساعدت عملية معايرة المعاملات الفائقة رغم الوقت المستغرق لإنجازنا لها لتحسين أداء النموذج، وهي واحدة من المقايضات التي ننجزها في مسائل تعلم الآلة.

سنحاول في القسم التالي فهم النموذج المتعلّم الدقيق الذي توصلنا له أكثر لاسيما آلية قيامه بالتنبؤات، كما سنحاول تحديد العوامل المؤثرة في معامل نجمة الطاقة. فمع العلم بأن النموذج المتعلّم الذي توصلنا إليه يبدو دقيقًا، إلا أننا نريد أن نفهم آلية وصوله لنتائجه.

11.12 تفسير وفهم النموذج

يوجه النقد لنماذج تعلّم الآلة غالبًا بأنها صناديق سوداء نُدخَل فيها البيانات من جهة للحصول على أجوبة دقيقة في أغلب الأحيان، ومن جهة أخرى دون أي تفسير واضح لكيفية الحصول على الجواب.

نعاين في هذا الجزء الثالث من بناء نموذج تعلّم آلة في بايثون نموذج التعلّم المطوّر لمحاولة فهم كيفية وصوله للتنبؤ الدقيق، وما يُمكن تعلمه حول مسألتنا المطروحة (التنبؤ بمعامل نجمة الطاقة) من هذا النموذج. وسنختم بمناقشة جزءٍ مهمٍ في مشاريع تعلّم الآلة وهو توثيق العمل وعرض النتائج بوضوح.

عرضنا في الجزء الأول مسألة تنظيف البيانات وتحليلها الاستكشافي، ومن ثم آليات هندسة الميزات لاختيار المناسب منها؛ أما في الجزء الثاني فشرحنا آلية احتساب القيم الناقصة وعرضنا الشيفرة اللازمة لإنشاء نماذج تعلّم، كما وازنا بين عدة نماذج تعلّم ممكنة وصولًا إلى اختيار النموذج الأمثل لمسألتنا. وبهدف تحسين أداء النموذج المختار عالجنا مسألة معايرة المعاملات الفائقة للنموذج باستخدام البحث العشوائي مع التقويم المتقاطع. وفي النهاية حسبنا أداء النموذج مع بيانات الاختبار. ننصح الجميع بتنزيل الشيفرة ومعاينتها ومشاركتها، فهي متاحة للعموم.

نُذكّر بأننا نعمل على تطوير نموذج تعلّم آلة من نمط موجه عبر الانحدار، وذلك باستخدام بيانات الطاقة لمباني نيويورك المتاحة للعموم بهدف التنبؤ بمعامل نجمة الطاقة للمبنى، ووصلنا في النهاية إلى بناء نموذج تعلّم من نمط الانحدار المعزز بالتدرج مع متوسط خطأ مطلق يساوي إلى 9.1 نقطة، بحيث يتراوح معامل نجمة الطاقة بين 1 و100 نقطة.

يُعدّ الانحدار المعزز بالتدرج من النماذج المعقدة، إلا أنه يتموضع في منتصف مقياس إمكانية تفسير النموذج نظرًا لاحتوائه على أشجار القرار Decision tree القابلة للشرح والتفسير.

سننظر في الطرق الثلاثة التالية لمحاولة فهم آلية التنبؤ:

1. أهمية الميزات
2. معاينة شجرة قرار واحدة
3. التفسيرات المحلية المحايدة للنموذج

تختص الطريقة الأولى والثانية بأشجار القرار؛ أما الطريقة الثالثة فهي طريقة عامة يُمكن استخدامها مع أي نموذج، وهي حزمة برمجية جديدة وخطوة مهمة نحو فهم آلية التنبؤ في مسائل تعلّم الآلة.

11.12.1 أهمية الميزات

تُحسب أهمية ميزة ما بمدى تأثيرها في الوصول إلى القيمة الهدف. وهنا لن نخوض في حسابات الأهمية المعقدة مثل حساب متوسط الانخفاض في عدم النقاء أو حساب انخفاض الخطأ عند تضمين الميزة، إذ توفر مكتبات Scikit-Learn إمكانية حساب أهمية كل ميزة لأي نموذج متعلّم يعتمد على الأشجار.

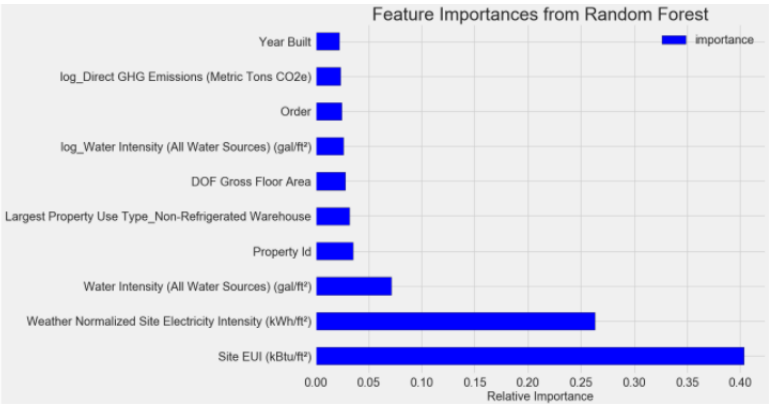
تحسب الشيفرة التالية أهمية الميزات باستخدام `model.feature_importances`، حيث `model` هو النموذج المطور، ثم نضع الأهميات المحسوبة في إطار بيانات بهدف رسم العشر الأوائل الأكثر أهمية:

```
import pandas as pd

# model النموذج المطور
importances = model.feature_importances_

# train_features إطار بيانات المزايا
feature_list = list(train_features.columns)

# وضع أهميات المزايا في إطار بيانات
feature_results = pd.DataFrame({'feature': feature_list,
                                'importance': importances})
```

```
# إظهار الميزات العشر الأكثر أهمية
feature_results = feature_results.sort_values('importance',
                  ascending = False).reset_index(drop=True)
feature_results.head(10)
```

وتظهر لنا الميزات العشر الأكثر أهمية:

	feature	importance
0	Site EUI (kBtu/ft²)	0.403532
1	Weather Normalized Site Electricity Intensity ...	0.263059
2	Water Intensity (All Water Sources) (gal/ft²)	0.071286
3	Property Id	0.035165
4	Largest Property Use Type_Non-Refrigerated War...	0.031924
5	DOF Gross Floor Area	0.027900
6	log_Water Intensity (All Water Sources) (gal/ft²)	0.026058
7	Order	0.024592
8	log_Direct GHG Emissions (Metric Tons CO2e)	0.023655
9	Year Built	0.022100



يُظهِر الشكل بوضوح أن أهم ميزتين (تُشكّلان لوحدهما حوالي 66% من الأهمية الكلية) هما كثافة استخدام الطاقة Site EUI، وكثافة الكهرباء وفق الطقس Weather Normalized Site Electricity Intensity.

تنخفض أهمية الميزات الأخرى بعد هاتين الميزتين مما يعني أننا لسنا مضطرين لاستخدام كل الميزات في النموذج (64 ميزة) للوصول إلى دقة كبيرة، وقد اخترنا عمليًا على سبيل التجريب عشر ميزات فقط وقمنا بإعادة بناء النموذج إلا أن دقته لم تكن جيدةً. وبهذا تعرّفنا السريع عن أهم الميزات المؤثرة في معامل نجمة الطاقة دون الاضطرار للغوص كثيرًا في الخلفية النظرية لهذا الموضوع.

11.12.2 معاينة شجرة قرار وحيدة

تُعَد أشجار القرار من أبسط الأشياء القابلة للفهم بخلاف الانحدار المعزز بالتدرج، حيث نستخدم في الشيفرة التالية التابع `export_graphviz` من Scikit-Learn لمعاينة أي شجرة من غابة الأشجار، إذ نستخرج أولًا شجرةً من غابة الأشجار ثم نحفظها في ملف نقطي dot.

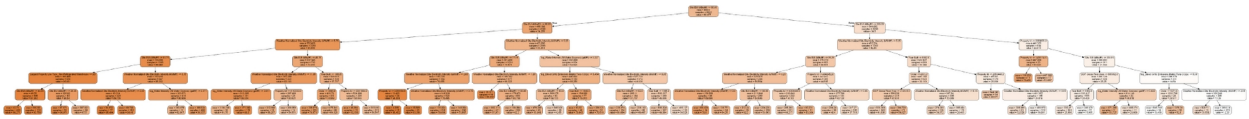
```
from sklearn import tree

# استخراج الشجرة (105)
single_tree = model.estimators_[105][0]

# حفظ الشجرة في ملف نقطي
tree.export_graphviz(single_tree, out_file = 'images/tree.dot',
```

```
                feature_names = feature_list)
```

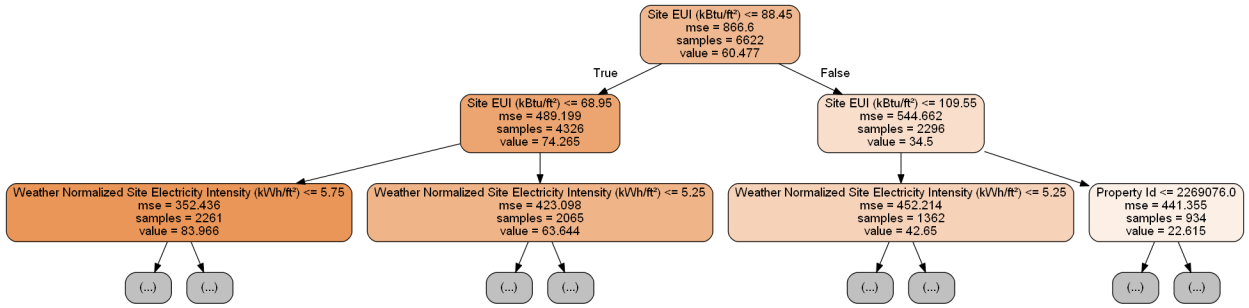
نستخدم برمجية المعاينة من Graphviz لحفظ الصورة بصيغة png وذلك باستخدام التعليمة cmd التالية:

```
dot -Tpng images/tree.dot -o images/tree.png
```

ويكون الناتج عبارة عن شجرة كاملة:

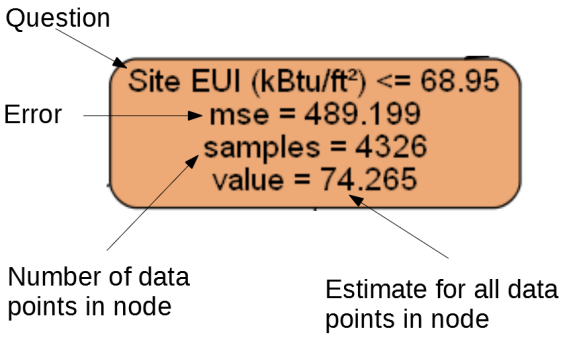


لا يُمكن تفحص الشجرة السابقة جيدًا على الرغم من أن عمقها 6 فقط لذا سنعيد توليد الشجرة بعمق 2 عن طريق تعديل معامل العمق للتابع `export_graphviz`:



تحوي كل عقدة من الشجرة المعلومات التالية:

1. اختبار منطقي لقيمة ميزة ما، حيث تُحدّد نتيجة هذا الاختبار المنطقي الاتجاه التـالي في الشـجرة نـزولًا يمينًا أو يسارًا.
2. `mse` قياس الخطأ في العقدة.
3. `samples` عدد الأمثلة في العقدة.
4. `value` تقدير القيمة الهدف عند هذه العقدة.



> لن يكون في أوراق الشجرة إلا قيم الخطأ وتقدير القيمة الهدف.

تتنبأ شجرة القرار بالنتيجة كما يلي: تبدأ من العقدة الأولى (الجذر) وتتحرك نحو الأسفل حتى تصل لورقة من أوراق الشجرة، ويُحدّد الاختبار المنطقي نعم/لا في كل عقدة اتجاه الحركة يمينًا أو يسارًا. فمثلًا، تسأل عقدة الشكل السابق عما إذا كانت قيمة ميزة كثافة الطاقة EUI أقل من 68.95 أم لا؟ فإذا كان الاختبار صحيحًا ينتقل للعقدة الأدنى اليسرى، وإلا للعقدة الأدنى اليمنى، وهكذا حتى الوصول لورقة فتكون القيمة فيها هي قيمة التنبؤ النهائي. وإذا حوت الورقة على عدد من الأمثلة فسيكون لهم جميعًا نفس قيمة التنبؤ.

وبالطبع فكلما نزلنا في الشجرة ينقص الخطأ المرتكب في التنبؤ لأننا ننعم عمليًا الأمثلة أكثر فأكثر، إلا أن زيادة العمق قد توقعنا في مشكلة فرط التخصيص ولن تتمكن الشجرة عندها من تعميم الأمثلة.

عايرنا المعاملات الفائقة للنموذج في الخطوات السابقة والتي تتحكم في شجرة القرار الناتجة، مثل العمق الأعظم للشجرة وعدد الأمثلة الأصغر المطلوب في ورقة، وهما معاملان يلعبان دورًا كبيرًا في المقايضة الملائمة/عدم الملائمة الفائضة.

تسمح معاينة شجرة القرار لنا بفهم تأثير هذه المعاملات الفائقة عمليًا، وعلى الرغم من أن معاينة جميع الأشجار أمرًا صعبًا، إلا أنه يكفي لنا معاينة شجرة واحدة فقط لنتمكن من فهم آلية التنبؤ التي ينجزها النموذج. وتبدو هذه الطريقة المعتمدة على مخطط انسيابي Flowchart مشابهةً لما يفعله الإنسان عند اتخاذ قراراته بالإجابة على سؤال حول قيمة معينة في كل مرة.

تراكب أشجار القرار المعتمدة على مجموعات يبيّن تنبؤات العديد من أشجار القرار الفردية من أجل إنشاء نموذج أكثر دقة مع تباين أقل، وتميل مجموعات الأشجار عامةً إلى أن تكون دقيقةً للغاية، كما أنها سهلة الشرح.

11.12.3 التفسيرات المحلية المحايدة للنموذج LIME

تهدف هذه الأداة الجديدة إلى شرح تنبؤ وحيد لأي نموذج، وذلك بإنشاء تقريب محلي للنموذج يكون قريبًا من المثال المدروس باستخدام نموذج بسيط مثل الانحدار الخطي[3].

سنستخدم LIME لتفحص تنبؤ خاطئ ينجزه نموذجنا المطور، وللحصول على تنبؤ خاطئ نستخرج المثال الذي يكون معامل الخطأ المطلق له أكبر ما يُمكن:

```
from sklearn.ensemble import GradientBoostingRegressor


# إنشاء النموذج مع المعاملات الفائقة الأمثل
model = GradientBoostingRegressor(loss='lad', max_depth=5,
max_features=None,
```

3 للحصول على تفاصيل أكثر يُمكن مراجعة الورقة البحثية Explaining the Predictions of Any Classifier

```
                                    min_samples_leaf=6,
min_samples_split=6,
                                    n_estimators=800, random_state=42)
# ملائمة واختبار النموذج
model.fit(X, y)
model_pred = model.predict(X_test)

# إيجاد الأخطاء
residuals = abs(model_pred - y_test)

# استخراج أكثر تنبؤ خاطئ
wrong = X_test[np.argmax(residuals), :]

print('Prediction: %0.4f' % np.argmax(residuals))
print('Actual Value: %0.4f' % y_test[np.argmax(residuals)])
```

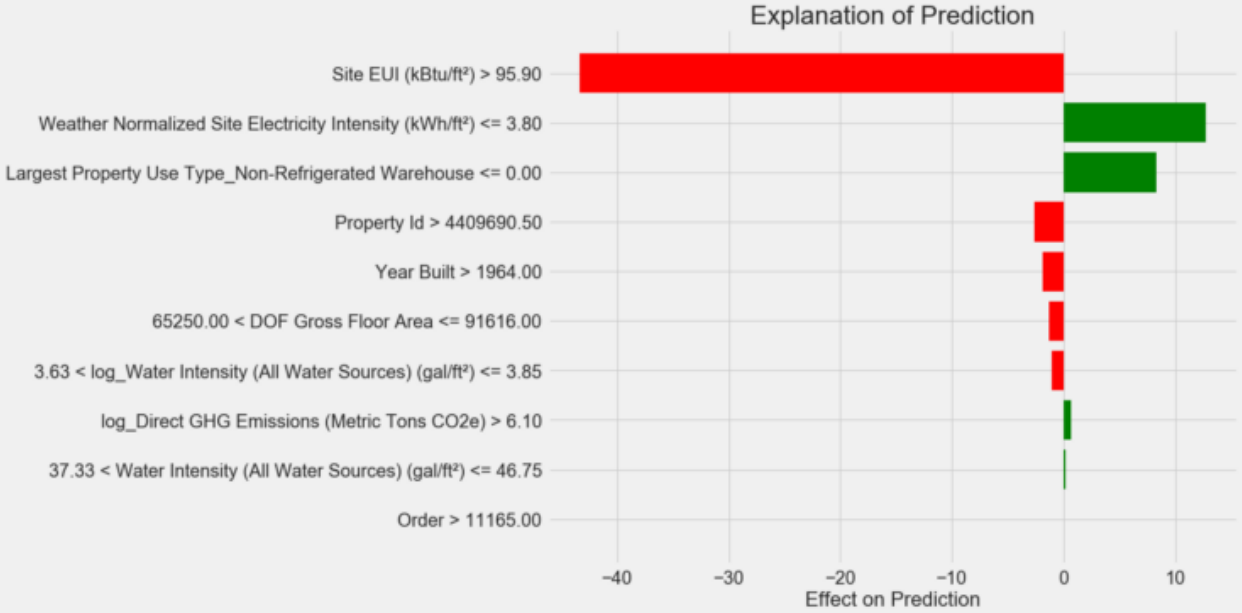
يكون الناتج:

```
Prediction: 12.8615
Actual Value: 100.0000
```

تُنشئ الشيفرة التالية كائن شرح مع تمرير المعاملات التالية له: بيانـات التـدريب والنمـط وعنـاوين بيانـات التدريب وأسماء الميزات.

```
import lime

# إنشاء كائن شرح
explainer = lime.lime_tabular.LimeTabularExplainer(training_data = X,
                                                   mode = 'regression',
                                                   training_labels = y,
                                             feature_names = feature_list)

# شرح أسوء تنبؤ
exp = explainer.explain_instance(data_row = wrong,
                                 predict_fn = model.predict)

# رسم شرح التنبؤ
exp.as_pyplot_figure();
```

ويكون الناتج ما يلي:

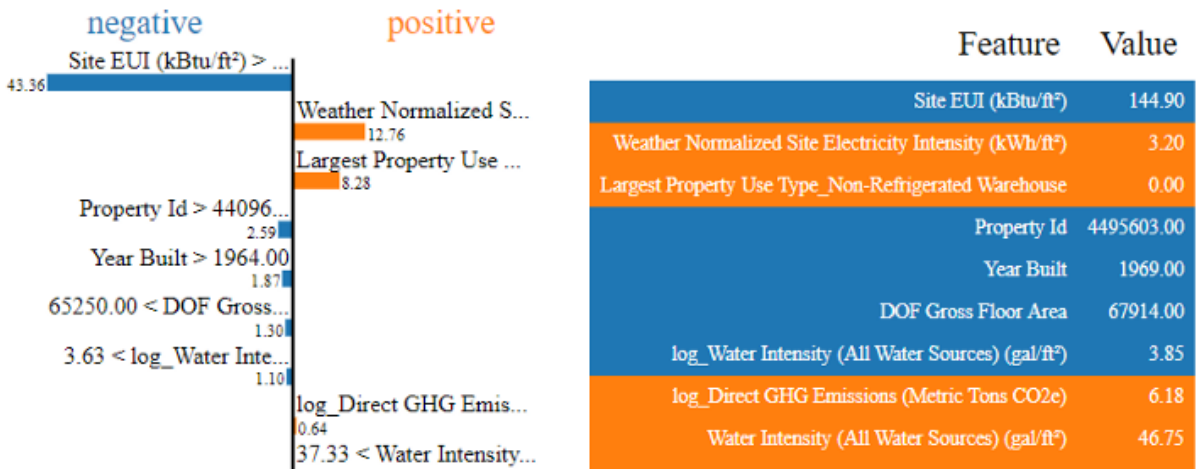


يُمكن تفسير الشكل كما يلي: يُظهِر المحور y الميزات التي أوصلت للنتيجة مع شريط أخضر في حال كانت قيمة الميزة تؤثر إيجابًا على القيمة الهدف؛ أما إذا كان تأثير قيمة الميزة سلبًيا فيكـون لـون الشـريط أحمـر. مثلًا: تؤدي قيمة الميزة الأولى والتي هي أكبر من 95.90 إلى طرح حـوالي 40 من القيمـة الهـدف؛ أمـا المـيزة الثانيـة والتي قيمتها أقل من 3.80 فتجمع 10 للتنبـؤ، حيث تكـون قيمـة التنبـؤ النهائيـة حاصـل جمـع كـل هـذه القيم الموجبة والسالبة.

يُمكن الحصول على نظرة أخرى لنفس المعلومات باستخدام الطريقة `show_in_notebook`:

```
# إظهار الشروحات
exp.show_in_notebook()
```

الناتج:



يُبين الشكل آلية المحاكمة التي ينجزها النموذج وصولًا للهدف من خلال عرض تأثير كل ميزة على التنبؤ (يسار الشكل)، وعرض القيم الفعلية (يمين الشكل).

للأسف، يُظهر المثال تنبؤ النموذج للقيمة الهدف بحوالي 12 بينما القيمة الفعلية هي 100؛ إلا أنه بتفحص قيم الميزات يتبين أن قيمة معامل كثافة الطاقة مرتفعة نسبيًا، مما يعني أن معامل نجمة الطاقة يجب أن يكون منخفضًا لأن الارتباط بين كثافة الطاقة ومعامل نجمة الطاقة هو ارتباط سلبي كما عرضنا سابقًا. في مثل هذه الحالة يجب البحث عن سبب إعطاء المبنى معاملًا مرتفعًا على الرغم من أن كثافة الطاقة مرتفعة. وهنا قد تبدو أدوات الشرح المتوفرة غير كاملة، إلا أنها وفي جميع الأحوال ساعدتنا على فهم النموذج المتعلّم، مما يسمح لنا باتخاذ قرارات أفضل.

11.13 توثيق العمل وإعداد تقارير النتائج

يغفل الكثيرون في المشاريع التقنية عن ضرورة توثيق العمل وعرض النتائج بوضوح، حيث يُمكن أن يُهمَل عملنا ولو كان رائعًا في حال لم نعرض النتائج بطريقة مناسبة.

يهدف التوثيق إلى وضع جميع الشروحات والتعليقات التي تُمكّن الآخرين من قراءة الشيفرة المكتوبة وفهمها وإعادة بنائها بسرعة عند الحاجة، إذ يُعدّ استخدام أدوات مثل Jupyter Notebooks والتي تسمح بوضع الشروحات المطلوبة أمرًا أساسيًا كي نتمكن نحن أولًا، ومن ثم الآخرين بمراجعة الشيفرة وتعديلها إن لزم ولو بعد أشهر من كتابتها. تسمح بعض الإضافات للأداة السابقة بإخفاء الشيفرة في التقرير النهائي، إذ قد لا يرغب البعض بمشاهدة قطع بايثون في كل فقرة.

ذكر كاتب المقال أنه يجد صعوبةً في تلخيص العمل المنجز وإخفاء التفاصيل، إلا أنه يوجز العمل كله في النقطتين التاليتين:

1. يُمكن بناء نموذج تعلّم للتنبؤ بمعامل نجمة الطاقة بالاعتماد على بيانات الطاقة لمدينة نيويورك مع خطأ ممكن بحوالي 9.1 نقطة.
2. تلعب الميزتان التاليتان الدور الأكبر في تحديد معامل نجمة الطاقة:
    - كثافة استخدام الطاقة Site EUI' Energy Use Intensity
    - وكثافة الكهرباء وفق الطقس Weather Normalized Site Electricity Intensity.

أُنجز هذا المشروع بصفة اختبار لكاتب المقال لقبوله في وظيفة في شركة ناشئة، كما طُلب منه عرضُ عمله واستنتاجاته في النهاية، ولذا فقد طور ملفًا من النمط Jupyter Notebook لتسليمه للشركة، وعوضًا عن تحويل الملف إلى pdf فورًا استخدم Latex لتحويله إلى ملف من النمط tex، ومن ثم تحريره باستخدام texStudio قبل توليد الملف النهائي بصيغة pdf. علمًا أنه يُمكن توليد الصيغة pdf من Jupyter مباشرةً وتحريرها لإجراء بعض التحسينات عليها.

وفي نهاية المطاف تلعب مهارة عرض النتائج دورًا أساسـيًا في إبـراز أهميـة العمـل ونتائجـه الـتي يُمكن أن تُبنى القرارات وفقها.

11.14 النتائج

استعرضنا في هذا الفصل خطوات بناء مشروع تعلّم آلة كامل من البداية إلى النهايـة، حيث بـدأنا بتنظيـف البيانات ثم انتقلنا إلى بناء النموذج ومن ثم لمحاولة فهم نموذج التعلّم.

نُعيد التذكير أخيرًا بخطوات بناء نموذج تعلّم الآلة:

1. تنظيف البيانات وتنسيقها.
2. تحليل البيانات الاستكشافي.
3. هندسة الميزات والاختيار منها.
4. موازنة نماذج تعلّم الآلة باستخدام مقياس للأداء.
5. معايرة المعاملات الفائقة لنموذج التعلّم.
6. تقويم النموذج الأمثل مع بيانات الاختبار.
7. تفسير نتائج النموذج إلى أقصى حد ممكن.
8. استخلاص النتائج وتوثيق العمل.

قد تختلف هذه الخطوات وفق المسألة المطروحة طبعًا، وغالبًا ما يكون العمل في مسائل تعلّم الآلة تكراريًـا وليس تسلسليًا، أي يُمكن أن نعود لخطوات سابقة دومًا.

نأمل أن يوفر لك هذا الفصل دليلًا جيدًا يُساعدك أثناء معالجتك لمشاريع تعلّم الآلة المستقبلية، كما نتمـنى أن تكون قد أصبحت قادرًا على تنفيذ مشاريع تعلّم الآلة الخاصة بك. وتذكر بأن أحدًا لا يعمل بمفـرده وأنـه يوجـد الكثير من المجموعات الداعمة والتي يُمكن طلب المساعدة منها.

12. تقييم واختيار نماذج تعلم الآلة

تُعدّ مسألة تقييم نماذج تعلم الآلة، أي حساب مجموعة من مقاييس تقييم الأداء والتي تُـبرهن على وصـول النموذج المُتعلم إلى درجة جيدة من التعلم وبحيث يُمكن الاعتماد عليه واستخدامه في مسألة مـا، أمـرًا أساسـيًا في مسائل تعلم الآلة.

تسمح هذه المقاييس، في نهاية المطاف، بالمقارنة بين مختلف نمـاذج التعلم المُمكن اسـتخدامها (والـتي يولد كل منها باستخدام خوارزمية ما معينة)، مما يسمح باختيار الأنسب منها.

نعرض في هذا الفصل مجموعة المقاييس الأكثر استخدامًا لتقـييم نمـاذج تعلم التصـنيف classification وتقـييم نمـاذج الانحـدار regression ومن ثم نعـرض كيفيـة المقارنـة بين عـدة نمـاذج تعلم ممكنـة واختيـار الأفضل منها.

12.1 مقاييس تقييم نماذج تعلم التصنيف

تهدف نماذج تعلم التصنيف إلى بنـاء مُصـنف يُمكن اسـتخدامه لتصـنيف غـرض مـا إلى صـف مُعين مثلًا ليكن لدينا مسألة تعلم تصنيف صورة إلى كلب أو قط، وبفرض أن لـدينا عشـرة صـور لهـا قيم الصـفوف التاليـة (كلب أم قط):

```
Actual values = ['cat', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog',
'cat', 'dog', 'cat']
```

وبفرض أن نموذج تصنيف أعطى التصنيفات التالية لنفس الصور (ندعوها عادةً تنبؤ نموذج التعلم):

```
Predicted values = ['cat', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog',
'dog', 'dog', 'dog']
```

من الواضح أن نموذج التصنيف أصاب في بعض الحالات وأخطأ في البعض الآخر. والسؤال المطروح هنا بشكل أساسي: ما كفاءة (تقييم) هذا النموذج؟

12.1.1 مصفوفة الارتباك confusion matrix

وهي عبارة عن جدول يُستخدم لبيان كفاءة نموذج تعلم التصنيف، إذ يعرض عدد حالات الصواب والخطأ المُمكنة المختلفة.

يُبين الجدول التالي مثلًا الحالات المختلفة في المثال السابق (ندعو، للتبسيط، صف الكلب بالموجب وصف القط بالسالب):

تنبؤ نموذج التصنيف	القيمة الحقيقية: كلب (موجب)	القيمة الحقيقية: قط (سالب)
كلب (موجب)	3	1
قط (سالب)	2	4

يُبين الجدول السابق أن نموذج التعلم أصاب في 3 حالات (الحقيقة: كلب / التنبؤ: كلب) + 4 حالات (الحقيقة: قط / التنبؤ: قط) = 7 حالات وأخطأ في حالتين (الحقيقة: كلب / التنبؤ: قط) + حالة واحدة (الحقيقة: قط / التنبؤ: كلب) = 3 حالات.

ندعو، بشكل عام، مصفوفة الارتباك الجدول التالي:

تنبؤ نموذج التصنيف	القيمة الحقيقية: (موجب)	القيمة الحقيقية: (سالب)
موجب	عدد الحالات الصحيحة الموجبة	عدد الحالات الخاطئة الموجبة
سالب	عدد الحالات الخاطئة السالبة	عدد الحالات الصحيحة السالبة

نشرح الحالات وفق التالي:

- **الحالات الصحيحة الموجبة**: الحالات الصحيحة الموجبة True Positive وتختصر إلى TP وهي الحالات التي يكون من أجلها تنبؤ النموذج موجب والقيمة الحقيقة موجب (تنبؤ صحيح).
- **الحالات الصحيحة السالبة**: الحالات الصحيحة السالبة True Negative وتختصر إلى TN وهي الحالات التي يكون من أجلها تنبؤ النموذج سالب والقيمة الحقيقة سالب (تنبؤ صحيح).
- **الحالات الخاطئة الموجبة**: الحالات الخاطئة الموجبة False Positive وتختصر إلى FP وهي الحالات التي يكون من أجلها تنبؤ النموذج موجب والقيمة الحقيقة سالب (تنبؤ خاطئ).
- **الحالات الخاطئة السالبة**: الحالات الخاطئة السالبة وتختصر إلى FN وهي الحالات التي يكون من أجلها تنبؤ النموذج سالب والقيمة الحقيقة موجب (تنبؤ خاطئ).

12.1.2 مقياس الصحة Accuracy

مقياس الصحة Accuracy وهو نسبة التصنيفات الصحيحة من العدد الكلي للأمثلة:

```
Accuracy = (TP + TN) / (TP + TN + FP + FN) * 100
```

تكون الصحة في مثالنا السابق:

```
Accuracy = (3 + 4) / 10 * 100 = 70%
```

12.1.3 مقياس الدقة Precision

مقياس الدقة Precision هو نسبة التصنيفات الصحيحة للأمثلة الموجبة على العدد الكلي للأمثلة التي صُنّفت موجبة:

```
Precision = TP/(TP + FP) * 100
```

تكون الدقة في مثالنا السابق:

```
Precision = 3/(3+1) * 100 = 75%
```

12.1.4 مقياس الاستذكار Recall

مقياس الاستذكار Recall هو نسبة التصنيفات الصحيحة للأمثلة الموجبة على العدد الكلي للأمثلة الموجبة (يدعى هذا المقياس أيضًا بالحساسية Sensitivity أو نسبة الموجب الصحيح True Positive Rate):

```
Recall = TP / (TP + FN) * 100
```

يكون الاستذكار في مثالنا السابق:

```
Recall = 3/(3+2) * 100 = 60%
```

12.1.5 المقياس F1

المقياس F1 وهو مقياس يوازن بين الدقة والاستذكار في قيمة واحدة:

```
F1-Score = (2 * Precision * Recall) / (Precision + Recall)
```

يكون F1 في مثالنا السابق:

```
F1-Score = (2*75*60)/(75+60) = 66.66%
```

12.1.6 الخصوصية Specificity

الخصوصية Specificity هو نسبة التصنيفات الخاطئة للأمثلة الموجبة على العدد الكلي للأمثلة السالبة (يدعى هذا المقياس أيضًا بنسبة الموجب الخاطئ False Positive Rate):

```
FPR = FP / (FP + TN) * 100
```

تكون الخصوصية في مثالنا السابق:

```
FPR = 1/(1+4) * 100 = 20%
```

12.1.7 حساب مقاييس الأداء في بايثون

توفر المكتبة sklearn.metrics في بايثون إمكانية حساب كل المقاييس السابقة، كما تُبين الشيفرة البرمجية التالية:

```
# تقييم نماذج التصنيف
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
# الصفوف
actual = ['cat', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog', 'cat',
'dog', 'cat']
# التنبؤ
predicted = ['cat', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog',
'dog', 'dog']
# مصفوفة الارتباك
cf_matrix = confusion_matrix(actual , predicted, labels=["dog", "cat"]
)
print ('Confusion Matrix :')
print (cf_matrix)
# مقاييس الأداء
print ('Accuracy Score :{:.2f}'.format(accuracy_score(actual,
predicted)*100))
print ('Precision Score :{:.2f}'.format(precision_score(actual,
predicted, pos_label='dog')*100))
print ('Recall Score :{:.2f}'.format(recall_score(actual, predicted,
pos_label='dog')*100))
```

```
print ('F1 Score :{:.2f}'.format(f1_score(actual, predicted,
pos_label='dog')*100))
print ('Specificity :
{:.2f}'.format(cf_matrix[0,1]/(cf_matrix[0,1]+cf_matrix[1,1])*100))
```

نستخدم في الشيفرة السابقة كل من الدوال التالية والـتي يكـون لهـا معـاملين همـا قائمـة القيم الحقيقيـة وقائمة قيم التنبؤ:

الوصف	الدالة
حساب مصفوفة الارتباك	confusion_matrix
حساب الصحة	accuracy_score
حساب الدقة	precision_score
حساب الاستذكار	recall_score
حساب F1	f1_score

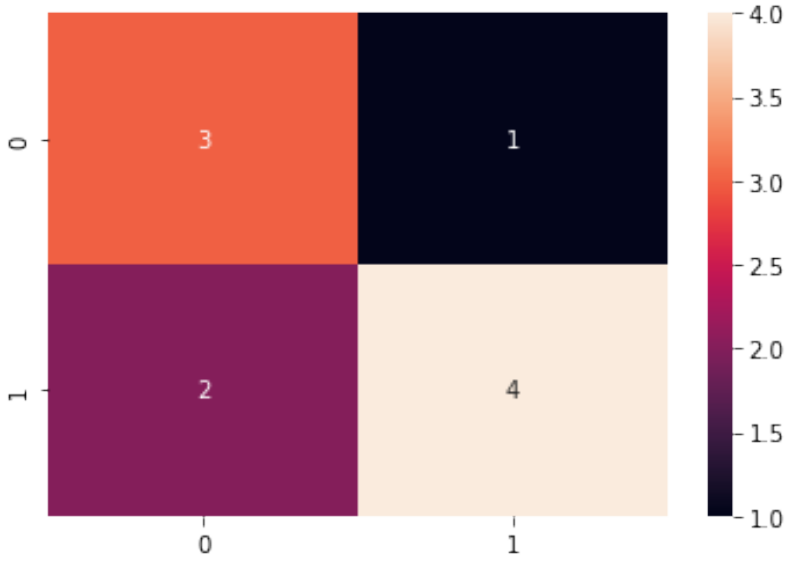
تكون نتائج مثالنا:

```
Confusion Matrix :
[[3 1]
 [2 4]]
Accuracy Score :70.00
Precision Score :60.00
Recall Score :75.00
F1 Score :66.67
```

يُمكن أيضًا رسم مصفوفة الارتباك باستخدام المكتبة seaborn كما يلي:

```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True)
```

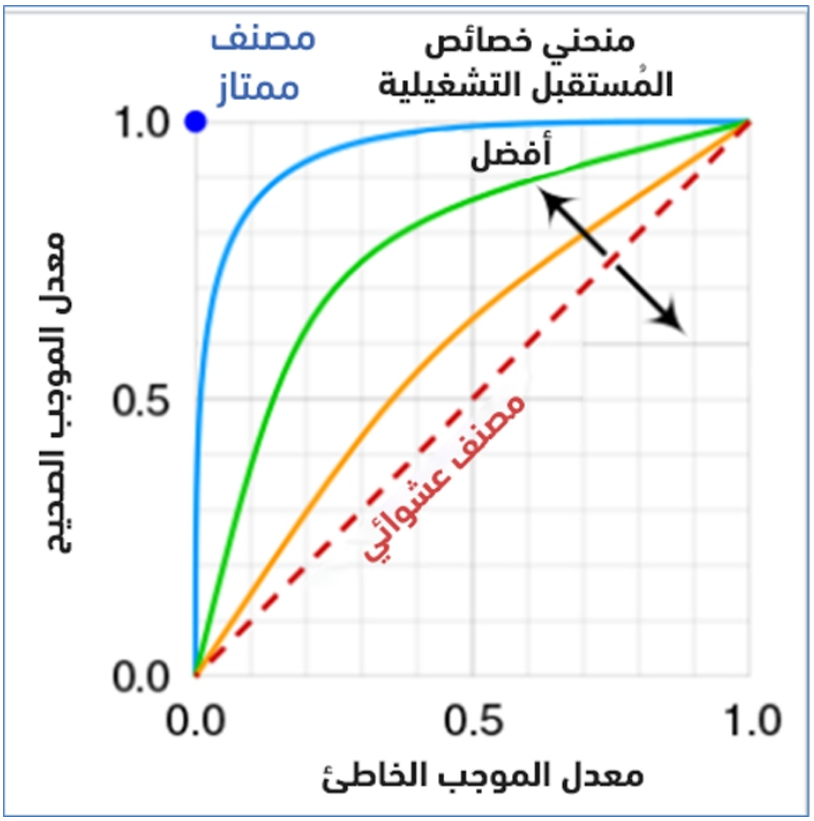
مما يُظهر الشكل التالي:



12.1.8 خصائص المُستقبل التشغيلية ROC

خصائص المُستقبل التشغيلية ROC -اختصار Receiver Operating Characteristic- هو منحني يُبين كفاءة نموذج التصنيف في قدرته على الفصل بين الصفوف الموجبة والسالبة، وكي يكون نموذج تصنيف ممتازًا يجب أن يصل إلى الزاوية العليا اليسارية أي أن تكون نسبة الصفوف الموجبة الصحيحة (الاستذكار) TPR أقــرب للواحد ونسبة الصفوف الموجبة الخاطئة (الخصوصية) FPR أقرب للصفر.

يُبين الشكل التالي منحنيات ROC مختلفة ممكنة. كلما اقتربنا من أعلى اليسار كان المُصنّف أفضل:



12.1.9 المساحة تحت المنحني AUC

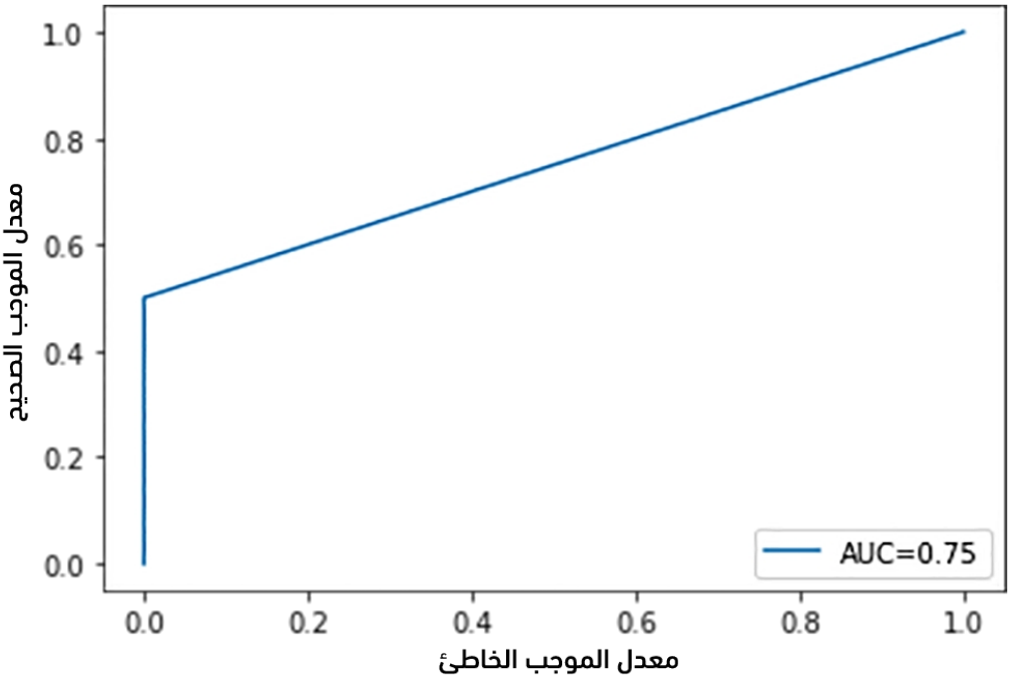
كلما كانت المساحة AUC -اختصار Area Under the Curve- تحت منحني ROC أكبر (أقرب من الواحد)، كان المُصنّف أفضل (لأن ذلك يعني أن المنحني أقرب للأعلى يسارًا).

يُمكن رسم منحني ROC في بايثون وحساب المساحة تحت المنحني AUC كما تُبين الشيفرة البرمجية التالية لمثالنا السابق:

```
# مكتبة الترميز
from sklearn.preprocessing import LabelEncoder
# الصفوف
actual = ['cat', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog', 'cat',
'dog', 'cat']
# التنبؤ
predicted = ['cat', 'cat', 'cat', 'dog', 'cat', 'cat', 'dog', 'dog',
'dog', 'dog']
# ترميز الصفوف
le = LabelEncoder()
# ترميز الصفوف كأرقام
actual=le.fit_transform(actual)
# ترميز التنبؤ كأرقام
predicted = le.fit_transform(predicted)
# مكتبة المقاييس اللازمة
from sklearn import metrics
# مكتبة الرسم
import matplotlib.pyplot as plt
# حساب المقاييس
fpr, tpr, _ = metrics.roc_curve(actual, predicted)
auc = metrics.roc_auc_score(actual, predicted)
# رسم المنحني
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

لاحظ استخدام الصف LabelEncoder من المكتبة sklearn.preprocessing لترميز الصفوف كأرقام.

يكون الإظهار:



لاحظ أن المساحة تحت المنحني تُساوي إلى 0.75 مما يعني أن المُصنف جيد نسبيًا.

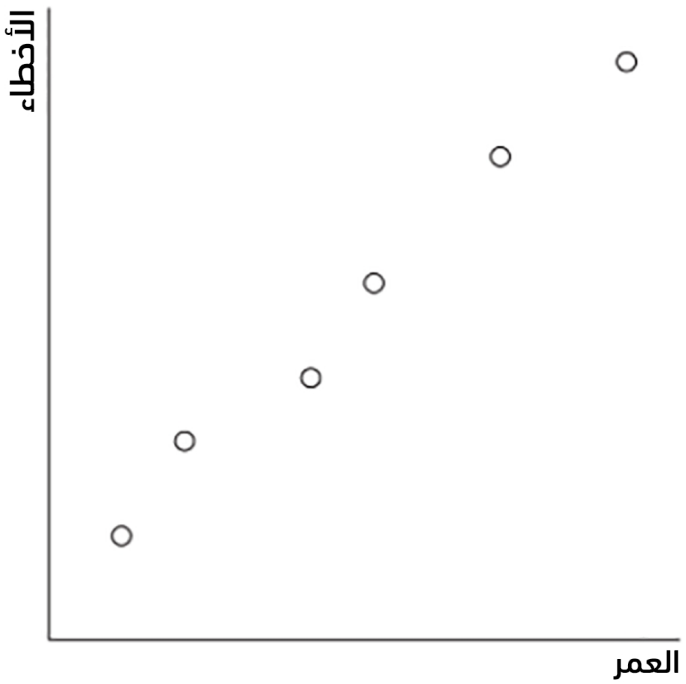
12.2 تقييم نماذج الانحدار

نُذكّر أولًا بأن الهدف من نماذج الانحدار هو التنبؤ بقيمة رقمية y انطلاقًا من قيمة (أو مجموعة من القيم) x.

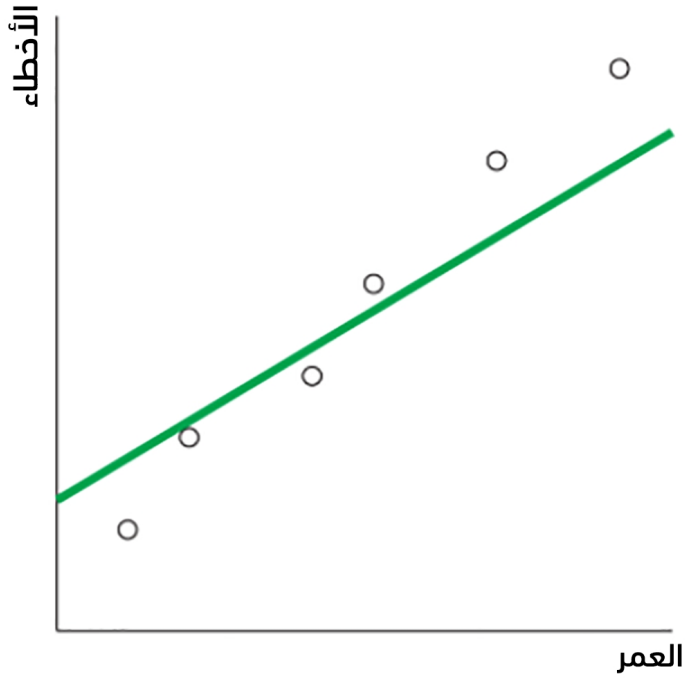
ليكن لدينا مثلًا جدول بيانات التدريب التالي والذي يُعطي عدد الأعطال لآلة وفق عمر الآلة بالسنوات:

Age	Failures
10	15
20	30
40	40
50	55
70	75
90	90

يُبين الشكل التالي هذه النقاط على محور العمر Age والأعطال Failures:



يُعدّ الانحدار الخطي linear regression من أكثر أنواع الانحدار استخدامًا نظرًا لبساطته في كل من التعلم والتنبؤ، فيُمكن مثلًا أن يتلاءم fit الانحدار الخطي مع البيانات السابقة ليولد المستقيم التالي:



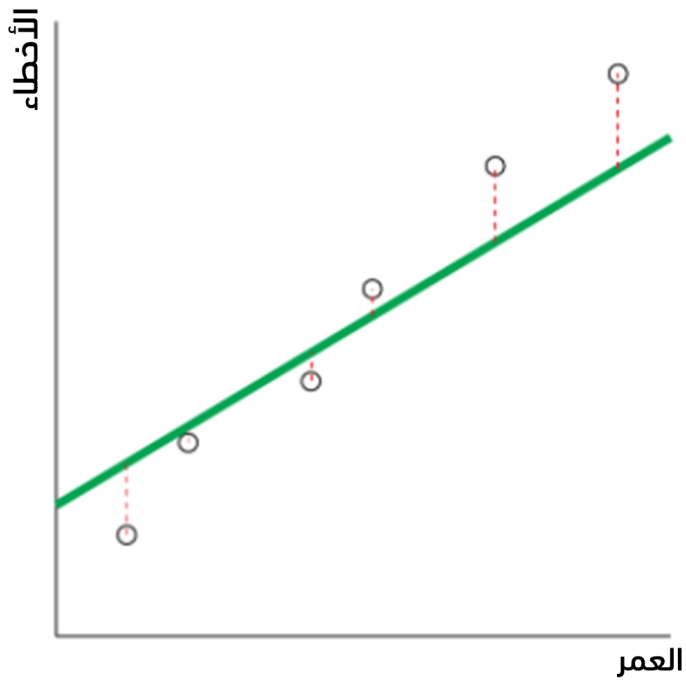
يُمكن حساب التنبؤ Predictions لكل من قيم الجدول السابق باستخدام معادلة المستقيم الناتجة عن نموذج الانحدار، مما يُعطي:

Predictions	Age	Failures
26	10	15
32	20	30
44	40	40
50	50	55
62	70	75
74	90	90

يُمكن الآن قياس مدى ملائمة المستقيم للبيانات بحساب المسافة بين النقاط الأساسية والمستقيم، وذلك بحساب الأخطاء Errors الحاصلة أي الفروقات بين القيم الأساسية و قيم التنبؤ، كما يُبين الجدول التالي:

Age	Failures	Predictions	Errors
10	15	26	11
20	30	32	2
40	40	44	4
50	55	50	5-
70	75	62	13-
90	90	74	16-

يُبين الشكل التالي هذه الفروقات أيضًا:



تُستخدم المقاييس التالية لتقييم أداء الانحدار:

- الجذر التربيعي لمتوسطات مربعات الأخطاء Root Mean Square Error (ويختصر إلى RMSE)
- متوسط الأخطاء بالقيمة المطلقة Mean Absolute Error (ويختصر إلى MAE)

12.2.1 الجذر التربيعي لمتوسطات مربعات الأخطاء RMSE

تُستخدم عملية التربيع أولًا للتخلص من الإشارة السالبة للأخطاء:

Age	Failutres	Predictions	Errors	Errors2
10	15	26	11	121
20	30	32	2	4
40	40	44	4	16
50	55	50	5-	25
70	75	62	13-	169
90	90	74	16-	256

ومن ثم نحسب متوسط مربعات الأخطاء، مما يُعطي 89.5. وأخيًرا نجذر المتوسط السابق فيكون الناتج 9.9 قيمة المقياس RMSE أي الجذر التربيعي لمتوسطات مربعات الأخطاء Root Mean Square Error.

بالطبع، كلما كان هذا العدد صغيرًا فهذا يعني أن الملائمة أفضل (الحالة المثالية هي الصفر).

12.2.2 متوسط الأخطاء بالقيمة المطلقة MAE

يُمكن أيضًا أن نأخذ القيمة المطلقة للأخطاء لنتخلص من الإشارة السالبة:

Age	Failutres	Predictions	Errors	|Errors|
10	15	26	11	11
20	30	32	2	2
40	40	44	4	4
50	55	50	5-	5
70	75	62	13-	13
90	90	74	16-	16

ومن ثم نحسب المتوسط الحسابي لها، مما يُعطي: 8.5 في مثالنا وهو مقياس MAE أي متوسط الأخطاء بالقيمة المطلقة Mean Absolute Error.

يُمكن حساب هذه المقاييس في بايثون باستخدام الدوال الموافقة من المكتبة sklearn.metrics، كما تُبين الشيفرة البرمجية التالية والتي نستخدم فيها بيانات مثالنا السابق:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from math import sqrt
y_true = [15, 30, 40, 55, 75, 90]
y_pred = [26, 32, 44, 50, 62, 74]
print ('MAE :{:.2f}'.format(mean_absolute_error(y_true, y_pred)))
print ('RMSE :{:.2f}'.format(sqrt(mean_squared_error (y_true,
y_pred))))
```

يكون الناتج:

```
RMSE :9.92
MAE :8.50
```

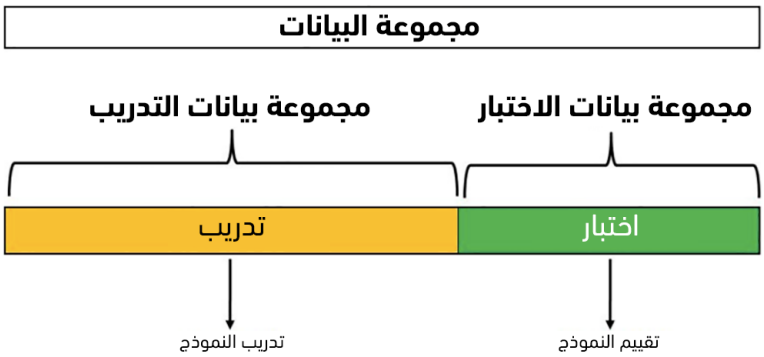
12.3 تقييم نماذج التعلم عبر التقسيم العشوائي Hold-out method

لبناء نموذج مُتعلم، يجب توفير مجموعة من البيانات dataset لتنفيذ خوارزمية التعلم عليها أولًا، ومن ثم تقييم نموذج التعلم الناتج.

يجب لتقييم نموذج التعلم، بشكل حيادي، أن نمرر له مجموعة من البيانات التي لم يرها خلال تدريبه وذلك تفاديًا لحصولنا على مقاييس تقييم ممتازة مزيفة نتيجة وقوع نموذج التعلم في مطب مشكلة فرط التخصيص overfitting، أي أن النموذج تعلم بشكل جيد على الأمثلة المُمره له فقط، وهو غير قادر على تعميم التعلم ليتعامل مع أمثلة جديدة بشكل جيد.

تُعدّ طريقة التقسيم العشوائي من أبسط الطرق المستخدمة لتقييم نماذج التعلم والتي تُقسّم البيانات المتاحة إلى قسمين بشكل عشوائي: ندعو القسم الأول ببيانات التدريب training data (عادة حوالي 80% من البيانات)، والقسم الآخر المتبقي (حوالي 20%) ببينات الاختبار testing data.

نعمد إلى تدريب وبناء نموذج التعلم باستخدام بيانات التدريب فقط، ومن ثم تقييم النموذج الناتج باستخدام بيانات الاختبار.



توفر المكتبة sklearn في بايثون كل ما يلزم للقيام بذلك كما تبين الشيفرة البرمجية التالية والتي نبني فيها نموذج تصنيف باستخدام مُصنّف بايز Bayes وذلك على مجموعة بيانات تصنيف أزهـار السوسـن iris المتاحـة من المكتبة نفسها.

```
# مجموعة بيانات
# أزهار السوسن
from sklearn.datasets import load_iris
# مكتبة تقسيم البيانات إلى تدريب واختبار
from sklearn.model_selection import train_test_split
# مكتبة مُصنف بايز
from sklearn.naive_bayes import GaussianNB
# مكتبات مقاييس التقييم
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
# تحميل البيانات
X, y = load_iris(return_X_y=True)
# تقسيم البيانات إلى تدريب و اختبار
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
# بناء مصنف بايز
gnb = GaussianNB()
# الملائمة على بيانات التدريب
# والتنبؤ على بيانات الاختبار
y_pred = gnb.fit(X_train, y_train).predict(X_test)
# حساب مقاييس التقييم
# بمقارنة بيانات الاختبار
# مع تنبؤ المصنف
```

```
print ('Accuracy Score :{:.2f}'.format(accuracy_score(y_test,
y_pred)*100))
print ('Precision Score :{:.2f}'.format(precision_score(y_test,
y_pred, average='macro')*100))
print ('Recall Score ::{:.2f}'.format(recall_score(y_test, y_pred,
average='macro')*100))
print ('F1 Score ::{:.2f}'.format(f1_score(y_test, y_pred,
average='macro')*100))
```

لاحظ أن الدالة train_test_split والتي نُمرر لها كل من:

- X: مصفوفة ثنائية يكون كل عنصر فيها مصفوفة من 4 عناصر (طول وعرض كـل من السـبال والبتلات لزهرة السوسن).
- y: مصفوفة أحادية (تصنيف الزهرة الموافق)
- test_size: النسبة المئوية لبيانات الاختبار من البيانات الكلية (20% بشكل افتراضي)

تُعيد ما يلي:

- X_train: بيانات التدريب المختارة عشوائيًا من X.
- X_test: بيانات الاختبار المختارة عشوائيًا من X.
- y_train: صفوف بيانات التدريب.
- y_test: صفوف بيانات الاختبار.

يُمكن تنفيذ الشيفرة لمعاينة قيم مقاييس الأداء:

```
Accuracy Score :93.33
Precision Score :92.59
Recall Score :93.94
F1 Score :92.50
```

يجب الانتباه إلى أن هذه القيم هي قيم استرشادية، بمعنى أنها ستختلف عند كل تنفيـذ للشـيفرة البرمجيـة إذ أن حسابها يعتمد في كل مرة على مجموعة مختلفة من بيانات الاختبـار (إذ أن هـذه البيانـات تُختـار كـل مـرة بشكل عشوائي) مثلًا يُعطي تنفيذ ثاني لنفس الشيفرة السابقة النتائج التالية:

```
Accuracy Score :90.00
Precision Score :92.31
Recall Score :92.86
```

```
F1 Score :91.65
```

12.4 تقييم نماذج التعلم عبر التقييم المتقاطع Cross Validation method

تهدف هذه الطريقة بشكل أساسي إلى استخدام كل البيانات للتدريب (بخلاف الطريقة السابقة والتي تستخدم قسمًا منها فقط)، كما أن قيم مقاييس التقييم ستكون نفسها من أجل كل تنفيذ للشيفرة (بخلاف الطريقة السابقة كما عرضنا أعلاه).

عوضًا عن تقسيم البيانات إلى بيانات للتدريب وبيانات للاختبار (التقييم) مما يُخفّض من البيانات التي يُمكن لنا استخدامها للتدريب، نستخدم التقويم المتقاطع مع عدد محدّد من الحاويات K-Fold.

تُقسم بيانات التدريب إلى عدد K من الحاويات ومن ثم نقوم بتكرار ما يلي K مرة: في كل مرة i نقوم بتدريب النموذج مع بيانات K-1 حاوية (كل الحاويات ما عدا الحاوية i) ومن ثم تقييمه مع بيانات الحاوية i. في النهاية، يكون مقياس الأداء النهائي هو متوسط مقياس التقييم لكل التكرارات (i:1..K).

توفر المكتبة sklearn في بايثون الدالة cross_val_score لتنفيذ التقييم المتقاطع مع تحديد عدد الحاويات المطلوب، كما تُبين الشيفرة البرمجية التالية والتي نحسب فيها مقياس الدقة مثلًا:

```
# مجموعة بيانات
# أزهار السوسن
from sklearn.datasets import load_iris
# مكتبة تقسيم البيانات إلى تدريب واختبار
from sklearn.model_selection import train_test_split
# مكتبة مُصنف بايز
from sklearn.naive_bayes import GaussianNB
# مكتبة التقييم المتقاطع
from sklearn.model_selection import cross_val_score
# مكتبات مقاييس التقييم
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

```
from sklearn.metrics import f1_score
# تحميل البيانات
X, y = load_iris(return_X_y=True)
# تقسيم البيانات إلى تدريب و اختبار
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
# بناء مصنف بايز
gnb = GaussianNB()
# حساب مقياس الصحة
scores = cross_val_score(gnb, X, y, cv=5, scoring="accuracy")
print(scores)
# حساب المتوسط
meanScore = scores.mean()
print(meanScore * 100)
```

يكون ناتج التنفيذ (في كل مرة):

```
[0.93333333 0.96666667 0.93333333 0.93333333 1.        ]
95.33333333333334
```

12.5 اختيار نموذج التعلم الأفضل

يُمكن استخدام طريقة التقويم المتقاطع السابقة لتقييم مجموعة من نماذج التعلم. نُعرّف في الشيفرة البرمجية التالية الدالة cv_comparison_classification والتي نُمرر لها مجموعة من نماذج التعلم models والبيانات كلها (X,y) وعدد الحاويات cv، وتُعيد هذه الدالة إطار بيانات يُظهر مقياس الدقة لكل نموذج.

تدور حلقة for في الدالة على النماذج المُمرَّرة، وتستدعي من أجل كل نموذج دالة التقويم المتقاطع من المكتبة sklearn.model والتي تُعيد قائمة بقيم مقياس الصحة (كل عنصر في القائمة هو قيمة مقياس الصحة من أجل حاوية اختبار ما).

نحسب متوسط mean القائمة السابقة، ونضيف عمودًا جديدًا إلى إطار البيانات يكون اسمه اسم النموذج والقيمة في الخلية الموافقة للسطر الوحيد في إطار البيانات (والمفهرس بـ Accuracy) قيمة مقياس الصحة.

```
# مكتبة أطر البيانات
import pandas as pd
# مكتبة التقييم المتقاطع
from sklearn.model_selection import cross_val_score
# دالة لمقارنة مجموعة من النماذج
```

```
def cv_comparison_classification(models, X, y, cv):
    # تهيئة إطار بيانات لمقاييس التقييم
    cv_df = pd.DataFrame()
    # الدوران على النماذج
    # تطبيق التقييم المتقاطع
    for model in models:
        #  حساب مقياس الصحة لكل حاوية
        acc = cross_val_score(model, X, y,scoring='accuracy',  cv=cv)
        # حساب متوسط الصحة للنموذج
        acc_avg = round(acc.mean(), 4)
        # كتابة النتيجة في إطار البيانات
        cv_df[str(model)] = [ acc_avg]
        cv_df.index = ['Accuracy']
    return cv_df
```

نستدعي في الشيفرة البرمجية التالية الدالة السابقة للمقارنة بين ثلاثة نماذج للتصنيف:

- مصنف بايز GaussianNB
- مصنف أشجار القرار DecisionTreeClassifier
- مصنف أقرب الجيران KNeighborsClassifier

نستخدم بيانات تصنيف أزهار السوسن المتاحة من sklearn.datasets لتنفيذ نماذج التعلم عليها.

```
# مجموعة بيانات
# أزهار السوسن
from sklearn.datasets import load_iris
# مصنف بايز
from sklearn.naive_bayes import GaussianNB
# مصنف شجرة القرار
from sklearn.tree import DecisionTreeClassifier
# مصنف أقرب الجيران
from sklearn.neighbors import KNeighborsClassifier
# تحميل البيانات
X, y = load_iris (return_X_y=True)
# إنشاء متغيرات النماذج
mlr_g = GaussianNB()
mlr_d = DecisionTreeClassifier()
```

```
mlr_k = KNeighborsClassifier()
# وضع النماذج في قائمة
models = [mlr_g, mlr_d, mlr_k]
# استدعاء دالة المقارنة
comp_df = cv_comparison_classification(models, X, y, 4)
# إظهار إطار البيانات للمقارنة
print(comp_df)
```

يُظهر تنفيذ الشيفرة السابقة إطار البيانات التالي والذي يسمح لنا بمعاينة مقياس الصحة لكل نموذج مما يسمح لنا باختيار الأنسب منها:

	GaussianNB()	DecisionTreeClassifier()	KNeighborsClassifier()
Accuracy	0.9534	0.96	0.9667

12.6 الخلاصة

عرضنا في هذا الفصل مقاييس تقييم نماذج التعلم المختلفة وكيفية اختيار النموذج الأفضل منها لمسألة معينة والذي به ستصبح قادرًا على تقييم نماذج تعلم الآلة التي تطبقها وبذلك نختتم هذا الكتاب.

> يُمكن تجربة جميع أمثلة الفصل من موقع Google Colab من هذا الرابط أو من هذا الرابط من أكاديمية حسوب.

أحدث إصدارات أكاديمية حسوب