Sessió 3

Taula de continguts

Aprenentatge Automàtic

> **Nota**
>
> Els documents d'aquest curs contenen adaptacions de diferents materials que s'han anat utilitzant a altres cursos i conté aportacions de varis membres del Departament de Matemàtiques de la UAB.

El paquet `Scikit-Learn`

El paquet `Scikit-Learn` té implementats molts mètodes per crear models d'aprenentatge automàtic. Un dels punts interessants de `Scikit-Learn` és que els models es defineixen sempre amb la mateixa estructura, de forma que podem provar diferents tipus de models sobre un mateix conjunt de dades de forma molt àgil.

Per importar el paque `Scikit-Learn` s'ha de fer amb el nom `sklearn`, tot i que no s'acostuma a importar directament tot el paquet sino que s'importen només els mètodes que es volen fer servir.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.datasets import load_digits

from sklearn.decomposition import PCA
```

Amb el paquet `Scikit-Learn` podem reproduir el càlcul del model de regressió múltiple que vam implementar en la sessió anterior.

```
X_reg = np.array([[1, 1, 1], [4, 5, 6], [9, 8, 7], [0, 3, 3], [2, 1, 1], [6, 7, 8]])
y_reg = np.array([2, 3, 4, 5, 6, 7])
```

```
model = LinearRegression().fit(X_reg, y_reg)
```

En `Python`, si una variable acaba amb un guió baix (_) vol dir que el programador considera que un usuari no hauria d'utilitzar aquesta variable directament. En els models de `Scikit-Learn`, això és així perquè els models incorporen funcions que permeten fer prediccions directament sense consultar els valors dels paràmetres del model. De totes formes, podem accedir per comprovar que obtenim els mateixos valors que vam obtenir quan vam definir el model lineal manualment.

```
model.coef_
```

```
model.intercept_
```

Per fer noves prediccions directament amb `Scikit-Learn` ho podem fer defininint un `array` de `Numpy` amb les dimensions adequades (tantes files com registres tinguem i tantes columnes com variables).

```
model.predict(np.array([1, 2, 3]).reshape(1, -1))
```

```
preds_reg = model.predict(X_reg)
```

El paquet `Scikit-Learn` incorpora també diverses funcions per calcular mètriques amb les que podem avaluar el rendiment dels models.

```
mse = mean_squared_error(y_reg, preds_reg)
mse
```

Els models lineals que hem treballat fins ara són models de regressió, però amb `Scikit-Learn` també podem treballar amb problemes de classificació. Veiem exemples de problemes de classificació amb les dades del conjunt MNIST.

```
data = load_digits(as_frame=True)
```

```
df = data.frame
```

```
df
```

```
df.describe()
```

```
fig, ax = plt.subplots(2, 5, figsize=(24, 9))

for i in range(10):
  ax[i // 5, i % 5].imshow(df.iloc[i, :64].values.reshape(8, 8), cmap="gray")
  ax[i // 5, i % 5].set_title("{}".format(df.iloc[i]["target"]))
  ax[i // 5, i % 5].axis("off")

plt.show()
```

En la majoria de models de machine learning és important normalitzar els valors de les variables. Pel cas del dataset dels dígits, normalitzar les variables entre 0 i 1 és equivalent a dividir el valor de cada píxel per 16.

```
for c in df.columns:
  if c != "target":
    df[c] = df[c] / 16
```

Hem de definir llavors quines són les variables explicatives i quina és la variable que volem predir. En el nostre cas, la variable a predir correspon amb la columna `target` del `DataFrame`, mentre que la resta de columnes, que fan referència als valors dels píxels, són variables explicatives.

```
variables_explicatives = [c for c in df.columns if "target" not in c]
variable_target = "target"
```

Per comoditat, definim ara dues variables, `X` i `y`, on guardarem per separat la informació de les variables explicatives i la variable dependent.

```
X = df[variables_explicatives]
y = df[variable_target]
```

Els mètodes que fa servir `Scikit-Learn` per definir i utilitzar els models segueix una estructura molt marcada. A continuació veiem com podem obtenir les prediccions d'un model de `Random Forest` fent exactament la mateixa crida que amb la regressió lineal, però canviant les dades d'entrada.

Cada tipus de model en el paquet `Scikit-Learn` pot rebre una sèrie de hiperparàmetres diferents. En cada cas, al especificar el nom del model que volem fer servir podem especificar els valors dels hiperparèmtres que considerem adequats (la resta d'hiperparàmetres no especificats tindran un valor per defecte, que es pot consultar a la documentació de `Scikit-Learn`).

```
model = RandomForestClassifier(max_depth=10, n_estimators=10).fit(X, y)
```

```
preds = model.predict(X)
```

En un model de classificació les mètriques que s'acostumen a utilitzar per mesurar el rendiment del model tenen a veure amb l'encert del model.

La mètrica més bàsica que podem utilitzar és l'`accuracy`, que compta el percentatge de casos on el model ha encertat la seva predicció.

```
accuracy_score(y, preds)
```

El propi paquet `Scikit-Learn` conté també funcions per visualitzar diferents mètriques. A continuació veiem que es pot mostrar la matriu de confusió d'un model amb molt poques línies de codi.

```
cm = confusion_matrix(y, preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

Exercici

Quan tenim un conjunt de dades que volem utilitzar per entrenar un model d'aprenentatge automàtic, la validació no la podem fer sobre el mateix conjunt de dades, ja que estaríem sobreestimant el rendiment del model. Per tenir una estimació acurada del rendiment del model podem separar el conjunt de dades en dues parts, el conjunt d'entrenament i el conjunt de test. Llavors, utilitzarem les dades que hi ha en el conjunt d'entrenament per entrenar el model i les dades que hi ha en el conjunt de test per avaluar el rendiment del model.
La següent funció del paquet `Scikit-Learn` ens permet separar el conjunt de dades en els dos conjunts, entrenament i test, de forma fàcil. Utilitzeu la nova partició de les dades per entrenar un model `Random Forest` sobre les dades d'entrenament i avalueu el seu rendiment sobre les dades de test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Aquest problema particular de predir el número present en una imatge sempre es considera un problema de classificació, però podem veure com es resoldria considerant-lo un problema de regressió, per mostrar que l'estructura del codi és molt semblant.

```
model = RandomForestRegressor(max_depth=10, n_estimators=10).fit(X_train, y_train)
```

```
preds_train = model.predict(X_train)
```

```
preds_test = model.predict(X_test)
```

Al tractar-ho com a problema de regressió, les prediccions del model no tenen perquè ser números enters. Això fa que mesurar l'`accuracy` no tingui sentit. Per mesurar el rendiment del model hem d'utilitzar una mètrica de regressió, com el `Mean Squared Error`.

```
mse = mean_squared_error(y_test, preds_test)
mse
```

Exercici

Repetiu el procediment per entrenar un model de classificació per predir els valors presents en cada imatge, però utilitzant un model basat en `Gradient Boosting`.

En l'àmbit de l'aprenentatge automàtic l'objectiu principal acostuma a ser obtenir el millor model possible, en termes de capacitat de predicció i generalització. De totes formes, és molt habitual també necessitar entendre com el model fa les seves prediccions. La majoria de models d'aprenentatge automàtic són massa complexos com per poder analitzar directament com es

fan les prediccions, però es poden aplicar tècniques per intentar entendre en quines variables es fixa més el model.

En el cas del `Random Forest`, es pot obtenir un valor d'importància per a cada variable d'entrada al model. Com que en el nostre cas cada variable representa un píxel de la imatge, podem representar la importància associada a cada píxel també en una imatge, de forma que podem representar visualment en quines zones de la imatge el model posa més atenció a l'hora de fer les prediccions.

```
model = RandomForestClassifier(max_depth=10, n_estimators=10).fit(X_train, y_train)
```

```
plt.imshow(model.feature_importances_.reshape(8, 8), cmap="gray")
plt.show()
```

La següent funció ens permet seleccionar només aquelles imatges que pertanyen a un subconjunt de les categories. Amb aquesta funció podem generar un nou `DataFrame` on les files siguin imatges que contenen només els digits que haguem seleccionat.

```
def filter_dataset(values):
  return df.loc[df["target"].isin(values)]
```

```
df_2 = filter_dataset([0, 8])
```

Exercici

Entreneu un model de `Random Forest` amb 100 estimadors i profunditat màxima 25, utilitzant només un subconjunt de les categories. A continuació, representeu visualment la importància que dóna el model a cadascuna de les variables d'entrada. Creieu que el model està aprenent correctament els patrons que diferencien les imatges entre les categories seleccionades?

El paquet `Scikit-Learn` conté també mètodes d'aprenentatge automàtic no supervisats. És a dir, mètodes on l'objectiu no és fer una predicció, si no obtenir informació sobre la distribució de les nostres dades.

Un dels mètodes més coneguts d'aprenentatge no supervisat és el `PCA`. A continuació veiem com utilitzar `Scikit-Learn` per calcular la projecció de les imatges a dues dimensions utilitzant el mètode `PCA`.

```
X = df[variables_explicatives]
y = df[variable_target]
```

```
pca = PCA(n_components=2).fit(X)
```

```
pca.explained_variance_ratio_
```

```
X_pca = pca.transform(X)
```

```
for n in range(0, 10):
  idx = np.where(y == n)[0]
  plt.scatter(X_pca[idx, 0], X_pca[idx, 1], label="{}".format(n))

plt.legend()
plt.show()
```