

Pràctica 2 - L'enfonsament del Titànic i les possibilitats de sobreviure

Inteligència Artificial - Machine Learning

Grau en Enginyeria Informàtica - Universitat de les Illes Balears

Autors: Lluís Barca Pons i Victor Canelo Galera

Des de l'assignatura d'Intel·ligència Artificial se'ns ha demanat elaborar un informe sobre el famós enfonsament del vaixell Titànic, l'any 1912. Durant aquest naufragi, van morir 1502 de 2224 persones registrades, entre passatgers i tripulants.

El més interessant és que sembla que existien diferents probabilitats de sobreviure entre alguns grups de persones. Per tant, l'objectiu d'aquesta pràctica és realitzar un model predictiu, utilitzant les dades reals recollides, que ens mostri quins tipus de persones tenien més probabilitats de viure.

Llavors, la pràctica es dividirà en dues parts:

1. Entrenar una sèrie de predictors i comparar el seu rendiment.
2. Obtenir i analitzar la importància de cada característica per cada un dels models generats.

1. Primera part

Per a la configuració del nostre entorn, necessitarem els corresponents imports de les diferents llibreries de Python que utilitzarem durant tota la pràctica.

In []:

```
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, accuracy_score
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

1.1. Neteja del dataset

```
In [ ]: df = pd.read_csv("dades.csv")
df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500
...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

891 rows × 12 columns



En primer lloc, comprovarem si tenim alguna columna del nostre dataset amb algun valor *NaN* (no numèric):

```
In [ ]: df.isnull().any()
```

```
Out[ ]: PassengerId    False
         Survived      False
         Pclass        False
         Name         False
         Sex           True
         Age           True
         SibSp        False
         Parch        False
         Ticket       False
         Fare          False
         Cabin         True
         Embarked     True
dtype: bool
```

Ens adonem que tres de les dotze variables tenen valors no numèrics. Però, ens interessa saber quin tant per cent de les files no ens donen cap informació, ja que podria ser una informació d'utilitat per descartar directament qualche variable.

```
In [ ]: na_ratio = ((df.isnull().sum() / len(df))*100).sort_values(ascending = False)
print(na_ratio)
```

```
Cabin      77.104377
Age        19.865320
Embarked   0.224467
PassengerId 0.000000
Survived   0.000000
Pclass      0.000000
Name        0.000000
Sex         0.000000
SibSp      0.000000
Parch      0.000000
Ticket     0.000000
Fare       0.000000
dtype: float64
```

Analitzem els valors no numèrics de les tres primeres variables:

- **Cabin:** Els valors no numèrics ens indiquen realment que aquell passatger no tenia cabina. Llavors, realment un 77% dels passatgers anaven sense cabina. Això ens indica que no és necessari eliminar aquesta variable (de primeres), ja que ens podria estar donant una informació útil; que valorarem en fer la correlació. El que farem serà modificar la taula i els passatgers amb valor no numèric passaran a tenir un 0 i els que tinguin qualche cabina 1. D'aquesta manera sabrem quin passatger té o no cabina.
- **Age:** L'edat sol ser una informació de gran valor, ja que ens podria donar molta informació de quin grup de persones tenen més probabilitats de sobreviure. Per tant, haurem de substituir els valors no numèrics per la mitjana d'edat de la resta de persones al vaixell; amb l'objectiu de minimitzar l'impacte que podrien tenir en els nostres resultats aquesta falta d'informació d'alguns tripulants.
- **Embarked:** Aquesta variable ens aporta poca informació útil, com que no hi ha una relació directa entre les persones que embarquen en x port, amb la probabilitat de sobreviure. El vaixell es va enfonsar en mig de l'Atlàctic, i a causa d'un iceberg. Però, no la descartarem fins a realitzar la respectiva correlació, i com el nombre de valors no numèrics és insignificant, tampoc la netejarem.

A continuació, el procés de neteja del dataset:

```
In [ ]: model = df

# Eliminam les variables PassengerId, Name i Ticket
del model["PassengerId"]
del model["Name"]
del model["Ticket"]

# Calculam la mitjana d'edat de la gent del vaixell
edat = model[["Age"]]
edat = edat.dropna()
edat = int(edat.mean())
edat = float(edat)

# Modificam els valors no numèrics per la mitjana
model["Age"] = model["Age"].replace(np.nan, edat)

# Modificam el valor Cabin
model["Cabin"] = model["Cabin"].notnull().astype("int")

model
```

```
Out[ ]:   Survived  Pclass    Sex   Age  SibSp  Parch     Fare Cabin Embarked
0         0       3  male  22.0      1      0    7.2500     0        S
1         1       1 female  38.0      1      0   71.2833     1        C
2         1       3 female  26.0      0      0    7.9250     0        S
3         1       1 female  35.0      1      0   53.1000     1        S
4         0       3  male  35.0      0      0    8.0500     0        S
...
886        0       2  male  27.0      0      0   13.0000     0        S
887        1       1 female  19.0      0      0   30.0000     1        S
888        0       3 female  29.0      1      2   23.4500     0        S
889        1       1  male  26.0      0      0   30.0000     1        C
890        0       3  male  32.0      0      0    7.7500     0        Q
891 rows × 9 columns
```

1.2. Caracterització de les dades

Realitzarem la caracterització de les variables categòriques:

```
In [ ]: model["Fare"] = pd.cut(model["Fare"], bins=[0, 7.91, 14.45, 31, 120], labels=["L", "M", "C", "A", "U"])

model["Age"] = pd.cut(model["Age"], bins=[0, 12, 18, 40, 100], labels=["Kid", "Teen", "Adult", "Old"])

model = pd.get_dummies(model, columns=["Pclass", "Sex", "Age", "Fare", "Embarked"])

model
```

Out[]:

	Survived	SibSp	Parch	Cabin	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Age
0	0	1	0	0	0	0	1	0	1	
1	1	1	0	1	1	0	0	1	0	
2	1	0	0	0	0	0	1	1	0	
3	1	1	0	1	1	0	0	1	0	
4	0	0	0	0	0	0	1	0	1	
...
886	0	0	0	0	0	1	0	0	1	
887	1	0	0	1	1	0	0	1	0	
888	0	1	2	0	0	0	1	1	0	
889	1	0	0	1	1	0	0	0	1	
890	0	0	0	0	0	0	1	0	1	

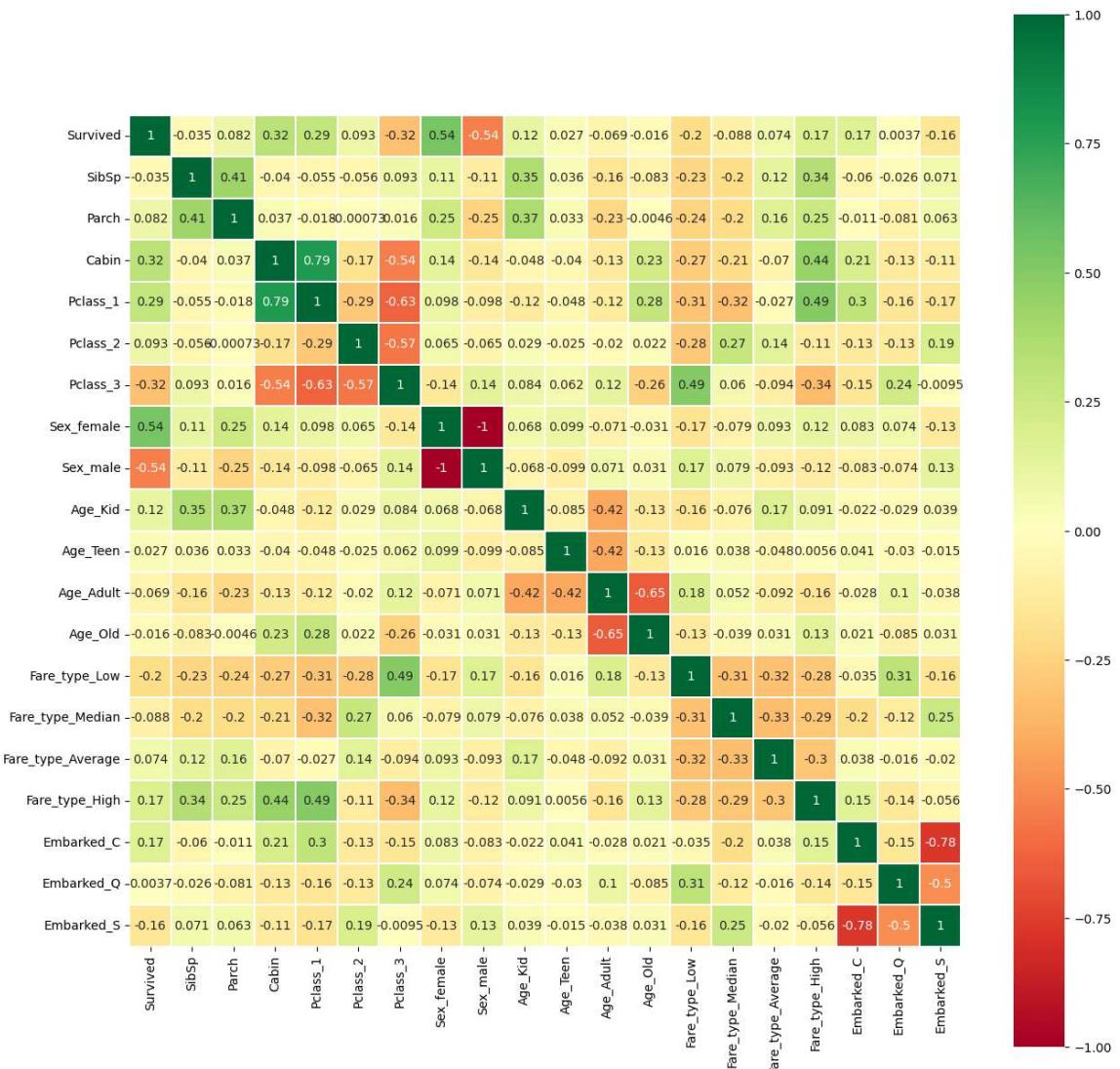
891 rows × 20 columns

1.3. Anàlisis del dataset

En primer lloc, realitzarem una correlació de totes les variables per analitzar quines ens poden interessar i quines no:

In []: corr_fig = plt.figure(figsize=(15,15))
sns.heatmap(model.corr(), annot=True, square=True, cmap="RdYlGn", linewidths=0.2)

Out[]: <AxesSubplot: >



A continuació realitzarem una segona correlació amb una variable nova. Aquesta serà la variable Family, que es crearà a partir de la suma de la variable Sibsp i Parch. L'objectiu d'aquesta prova és comprovar si el fet de tenir o no familiars a bord, va ser un motiu de pes a l'hora de sobreviure. Intuïtivament, podríem pensar que sí, però hem de comprovar-ho.

```
In [ ]: sibsp = model["SibSp"]
          parch = model["Parch"]
          family = []

          # Cream la nova columna family
          for i in range(len(sibsp)):
              family.append(sibsp[i] + parch[i])

          # Eliminam les variables sibsp i parch del dataset
          del model["SibSp"]
          del model["Parch"]

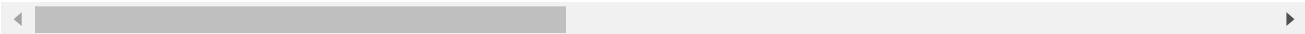
          # Afegim la columna family
          model.insert(5, "Family", family, True)

          model
```

Out[]:

	Survived	Cabin	Pclass_1	Pclass_2	Pclass_3	Family	Sex_female	Sex_male	Age_Kid	A
0	0	0	0	0	1	1	0	1	0	
1	1	1	1	0	0	1	1	0	0	
2	1	0	0	0	1	0	1	0	0	
3	1	1	1	0	0	1	1	0	0	
4	0	0	0	0	1	0	0	1	0	
...
886	0	0	0	1	0	0	0	1	0	
887	1	1	1	0	0	0	1	0	0	
888	0	0	0	0	1	3	1	0	0	
889	1	1	1	0	0	0	0	1	0	
890	0	0	0	0	1	0	0	1	0	

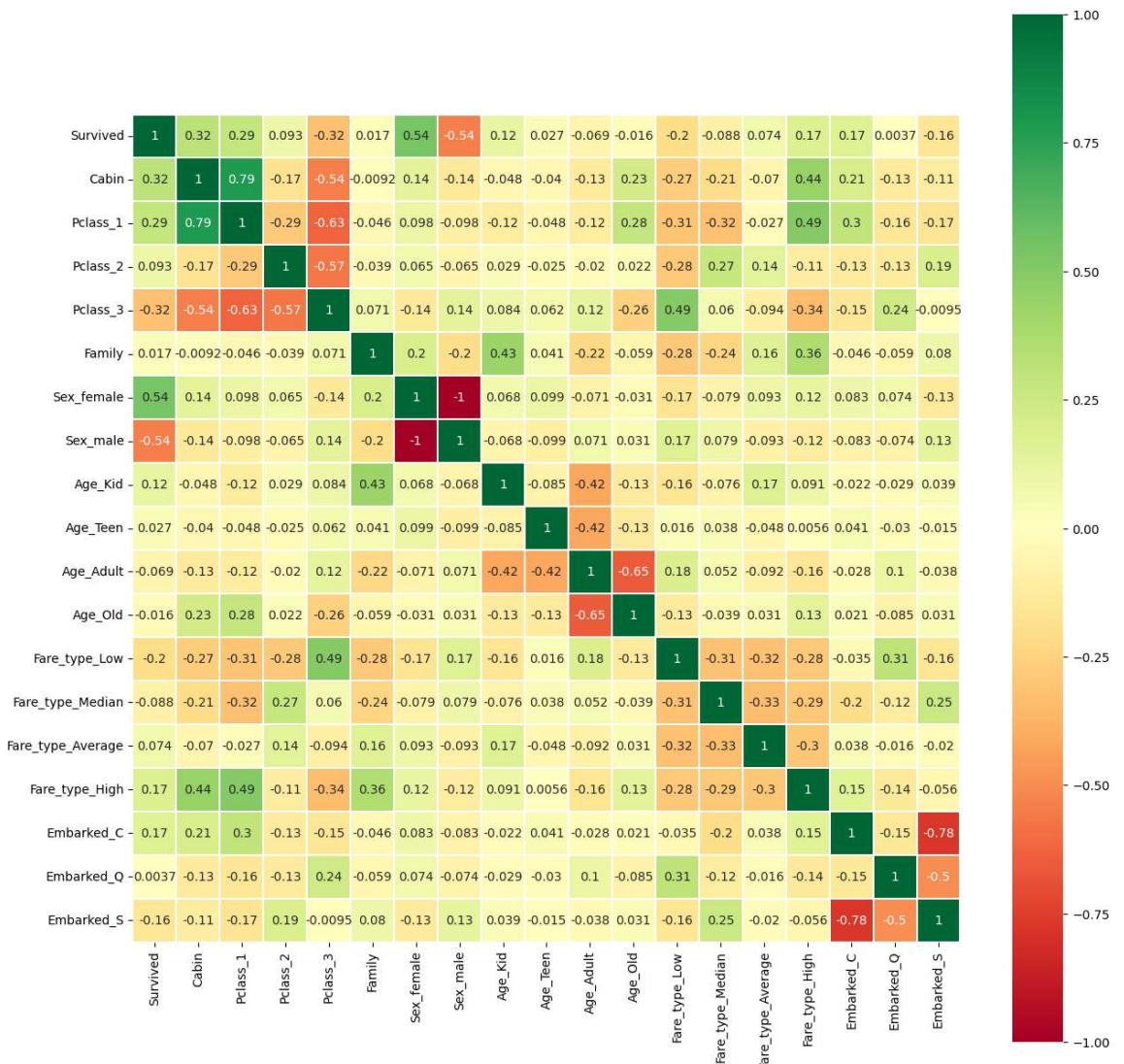
891 rows × 19 columns



Tornem a fer la correlació amb la nova variable:

In []: corr_fig = plt.figure(figsize=(15,15))
sns.heatmap(model.corr(), annot=True, square=True, cmap="RdYlGn", linewidths=0.2)

Out[]: <AxesSubplot: >



Llavors ens adonem que aquesta variable no afecta pràcticament i, per tant, podríem descartar-la. Llavors, les variables que més ens interessen són les que comparades amb "Survived", prenen valors o molt a prop d'1 o de -1; és a dir, que tindran molta correlació.

```
In [ ]: model
```

Out[]:

	Survived	Cabin	Pclass_1	Pclass_2	Pclass_3	Family	Sex_female	Sex_male	Age_Kid	A
0	0	0	0	0	1	1	0	1	0	
1	1	1	1	0	0	1	1	0	0	
2	1	0	0	0	1	0	1	0	0	
3	1	1	1	0	0	1	1	0	0	
4	0	0	0	0	1	0	0	1	0	
...
886	0	0	0	1	0	0	0	1	0	
887	1	1	1	0	0	0	1	0	0	
888	0	0	0	0	1	3	1	0	0	
889	1	1	1	0	0	0	0	1	0	
890	0	0	0	0	1	0	0	1	0	

891 rows × 19 columns

Per tant, podem descartar també les següents variables:

```
In [ ]: model_fit = model.copy()

del model_fit["Pclass_2"]
del model_fit["Family"]
del model_fit["Age_Kid"]
del model_fit["Age_Teen"]
del model_fit["Age_Adult"]
del model_fit["Age_Old"]
del model_fit["Fare_type_Median"]
del model_fit["Fare_type_Average"]
del model_fit["Embarked_Q"]

model_fit
```

Out[]:

	Survived	Cabin	Pclass_1	Pclass_3	Sex_female	Sex_male	Fare_type_Low	Fare_type_Hig
0	0	0	0	1	0	1	1	1
1	1	1	1	0	1	0	0	0
2	1	0	0	1	1	0	0	0
3	1	1	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
...
886	0	0	0	0	0	1	0	0
887	1	1	1	0	1	0	0	0
888	0	0	0	1	1	0	0	0
889	1	1	1	0	0	1	0	0
890	0	0	0	1	0	1	1	1

891 rows × 10 columns

Com ens queda un dataset bastant petit, hem decidit realitzar la part d'entrenament tant amb el dataset complet, com amb el que hem filtrat prèviament. Després de fer els respectius entrenaments amb els diferents models, escollirem el model que més precisió ens doni.

2. Entrenament

2.1. Model complet

En aquest apartat es duran a terme una sèrie d'entrenaments segons diferents predictors i finalment es compararan els seus rendiments. Abans, haurem de separar el nostre conjunt de dades per a entrenament, validació i testeig. També normalitzarem les dades perquè les puguem comparar entre elles.

```
In [ ]: scaler = preprocessing.MinMaxScaler()

y = model["Survived"]
x = model.drop("Survived", axis=1)

# Escalam les dades en funció de x
x = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

Out[]: ((623, 18), (268, 18), (623,), (268,))

2.1.1. Regressió Logística

A la biblioteca scikit-learn, la classe LogisticRegression s'utilitza per implementar la regressió logística. Aquesta classe proporciona diversos hiperparàmetres que ens permeten personalitzar el comportament del model.

Aquests són alguns dels hiperparàmetres principals de la classe LogisticRegression:

- **penalty**: Aquest hiperparàmetre especifica el tipus de regularització a utilitzar. Els valors possibles són "*l1*" per a la regularització L1 i "*l2*" per a la regularització L2. La regularització L1 afegeix una penalització basada en el valor absolut dels coeficients, mentre que la regularització L2 afegeix una penalització basada en el quadrat dels coeficients.
- **C**: Aquest hiperparàmetre controla la força de la regularització. Un valor més petit de C correspon a una regularització més forta, que pot ajudar a evitar el sobreajustament.
- **solver**: Aquest hiperparàmetre especifica l'algorisme a utilitzar per a l'optimització. Els valors possibles són "*newton-cg*", "*lbfgs*", "*liblinear*" i "*sag*".
- **max_iter**: Aquest hiperparàmetre especifica el nombre màxim d'iteracions a utilitzar quan s'ajusta el model.
- **tol**: Aquest hiperparàmetre especifica la tolerància per a l'optimització. Si la diferència entre la funció de cost en dues iteracions consecutives és menor que la tolerància, l'optimització s'aturarà.
- **multi_class**: Aquest hiperparàmetre especifica l'estrategia a utilitzar quan hi ha diverses classes. Els valors possibles són '*ovr*' per a un contra repòs i '*multinomial*' per a la pèrdua multinomial.
- **fit_intercept**: Aquest hiperparàmetre especifica si s'ha d'ajustar un terme d'intercepció al model.
- **class_weight**: Aquest hiperparàmetre us permet especificar pesos de classe per equilibrar la distribució de classes a les dades d'entrada.

És important ajustar aquests hiperparàmetres amb cura, ja que poden afectar significativament el rendiment del model

```
In [ ]: warnings.filterwarnings("ignore")

params = {
    "penalty" : ["l1", "l2", "elasticnet", None],
    "tol" : [0.000001, 0.00001, 0.0001, 0.001],
    "C" : [0.1, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25],
    "fit_intercept" : [True, False],
    "class_weight" : ["balanced", None],
    "solver" : ["lbfgs", "liblinear", "newton-cg", "newton-cholesky", "sag", "s
    "max_iter" : [20, 25, 50, 100],
    "multi_class" : ["auto", "ovr", "multinomial"],
}
```

```

grid = GridSearchCV(estimator=LogisticRegression(), param_grid=params, n_jobs=4)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'C': 0.23, 'class_weight': None, 'fit_intercept': True, 'max_iter': 20, 'multi_class': 'multinomial', 'penalty': None, 'solver': 'sag', 'tol': 0.001}

Out[ ]: 0.8233935483870967

```

Apliquem els paràmetres òptims aconseguits:

```

In [ ]: log = LogisticRegression(
    C= 0.23,
    class_weight=None,
    fit_intercept=True,
    max_iter= 20,
    multi_class="multinomial",
    penalty=None,
    solver="sag",
    tol=0.001,
    random_state=42).fit(x_train, y_train)
y_hat = log.predict(x_test)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendibilitat
logistic_predicts = []
for i in range(10):
    logistic_predicts.append(round(accuracy_score(log.predict(x_test),y_test)*100))

log_pred = np.array(logistic_predicts).mean()

print(f"La precisió és del {log_pred}%")

```

La precisió és del 80.6%

2.1.2 Perceptró

A la biblioteca scikit-learn, la classe *Perceptron* s'utilitza per implementar un model de perceptron d'una sola capa. Aquesta classe proporciona diversos hiperparàmetres que ens permeten personalitzar el comportament del model.

Aquests són alguns dels principals hiperparàmetres de la classe *Perceptron*:

- **penalty**: Aquest hiperparàmetre especifica el tipus de regularització a utilitzar. Els valors possibles són "*l1*" per a la regularització L1 i "*l2*" per a la regularització L2. La regularització L1 afegeix una penalització basada en el valor absolut dels coeficients, mentre que la regularització L2 afegeix una penalització basada en el quadrat dels coeficients.
- **alpha**: Aquest hiperparàmetre controla la força de la regularització. Un valor més petit d'alfa correspon a una regularització més forta, que pot ajudar a evitar el sobreajust.

- **fit_intercept**: Aquest hiperparàmetre especifica si s'ha d'ajustar un terme d'intercepció al model.
- **max_iter**: Aquest hiperparàmetre especifica el nombre màxim d'iteracions a utilitzar quan s'ajusta el model.
- **tol**: Aquest hiperparàmetre especifica la tolerància per a l'optimització. Si la diferència entre la funció de cost en dues iteracions consecutives és menor que la tolerància, l'optimització s'aturarà.
- **shuffle**: Aquest hiperparàmetre especifica si s'han de barrejar les dades d'entrenament abans d'ajustar el model.
- **early_stopping**: Aquest hiperparàmetre especifica si s'ha d'utilitzar l'aturada anticipada per finalitzar el procés d'ajust quan la puntuació de validació no millora.
- **validation_fraction**: Aquest hiperparàmetre especifica la fracció de les dades d'entrenament que cal utilitzar per a la validació quan l'aturada anticipada està habilitada.

És important ajustar aquests hiperparàmetres amb cura, ja que poden afectar significativament el rendiment del model.

```
In [ ]: warnings.filterwarnings("ignore")

params = {
    "penalty" : ["l2", "l1", "elasticnet", None],
    "alpha" : [0.00001, 0.0001, 0.001, 0.01, 0.1],
    "fit_intercept" : [True, False],
    "max_iter" : [50, 100, 150, 200],
    "tol" : [0.0001, 0.001, 0.01, 0.1],
    "shuffle" : [True, False],
    "early_stopping" : [True, False],
    "validation_fraction" : [0.00001, 0.0001, 0.001, 0.1],
}

grid = GridSearchCV(estimator=Perceptron(), param_grid=params, n_jobs=4)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'alpha': 1e-05, 'early_stopping': False, 'fit_intercept': True, 'max_iter': 50, 'penalty': 'l2', 'shuffle': True, 'tol': 0.0001, 'validation_fraction': 1e-05}

Out[ ]: 0.8041548387096775
```

Apliquem els paràmetres òptims aconseguits:

```
In [ ]: perc = Perceptron(
    alpha=0.00001,
    early_stopping=False,
    fit_intercept=True,
    max_iter=50,
```

```

penalty="l2",
shuffle=True,
tol=0.0001,
validation_fraction=0.00001,
random_state=42).fit(x_train, y_train)
y_hat = perc.predict(x_test)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de renda
perceptron_predicts = []
for i in range(10):
    perceptron_predicts.append(round(accuracy_score(perc.predict(x_test), y_test)))

percep_pred = np.array(perceptron_predicts).mean()

print(f"La precisió és del {percep_pred}%")

```

La precisió és del 78.73%

2.1.3. Arbres de decisió

A la biblioteca scikit-learn, la classe RandomForestClassifier s'utilitza per implementar un classificador de bosc aleatori. Aquesta classe proporciona diversos hiperparàmetres que ens permeten personalitzar el comportament del model.

Aquests són alguns dels hiperparàmetres principals de la classe RandomForestClassifier:

- **n_estimators**: Aquest hiperparàmetre especifica el nombre d'arbres del bosc. Un nombre més gran d'arbres normalment donarà lloc a un millor model, però també trigarà més temps a entrenar.
- **criterion**: Aquest hiperparàmetre especifica la funció per mesurar la qualitat d'una divisió. Els valors possibles són 'gini' per a la impuresa de Gini i 'entropia' per a guany d'informació.
- **max_depth**: Aquest hiperparàmetre especifica la profunditat màxima de cada arbre del bosc. Un valor més gran normalment donarà lloc a un model més complex, que pot sobreajustar les dades d'entrenament.
- **min_samples_split**: Aquest hiperparàmetre especifica el nombre mínim de mostres necessàries per dividir un node intern.
- **min_samples_leaf**: Aquest hiperparàmetre especifica el nombre mínim de mostres necessaris per estar en un node fulla.
- **max_features**: Aquest hiperparàmetre especifica el nombre de funcions que cal tenir en compte a l'hora de buscar la millor divisió.
- **bootstrap**: Aquest hiperparàmetre especifica si s'han d'utilitzar mostres d'arrencada quan es construeix cada arbre.
- **class_weight**: Aquest hiperparàmetre us permet especificar pesos de classe per equilibrar la distribució de classes a les dades d'entrada.

És important ajustar aquests hiperparàmetres amb cura, ja que poden afectar significativament el rendiment del model.

```
In [ ]: warnings.filterwarnings("ignore")

params = {
    "n_estimators" : [10, 20, 30, 40],
    "criterion" : ["gini", "entropy", "log_loss"],
    "max_depth" : [None, 3, 4, 5],
    "min_samples_split": [2, 3, 4],
    "min_samples_leaf": [1, 2],
    "max_features" : ["sqrt", "log2"],
    "bootstrap" : [True, False],
    "class_weight" : [None, "balanced", "balanced_subsample"],
}

grid = GridSearchCV(estimator=RandomForestClassifier(), param_grid=params, n_jobs=-1)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'bootstrap': True, 'class_weight': 'balanced', 'criterion': 'log_loss', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 4, 'n_estimators': 20}

Out[ ]: 0.8474580645161292
```

Apliquem els paràmetres òptims aconseguits:

```
In [ ]: forest_class = RandomForestClassifier(
    bootstrap=True,
    class_weight="balanced",
    criterion="log_loss",
    max_depth=None,
    max_features="sqrt",
    min_samples_split=4,
    min_samples_leaf=2,
    n_estimators=20,
    random_state=42).fit(x_train, y_train)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendibilitat
forest_predicts = []
for i in range(10):
    forest_predicts.append(round(accuracy_score(forest_class.predict(x_test), y_test)))

for_pred = np.array(forest_predicts).mean()

print(f"La precisió és del {for_pred}%")
```

La precisió és del 80.6%

La biblioteca scikit-learn proporciona una implementació del *Decision Tree Classifier* que ens permet entrenar i avaluar un model d'arbre de decisió sobre les nostres dades.

Aquests són alguns dels hiperparàmetres principals de la classe *Decision Tree Classifier*.

- **criterion:** Aquesta és la funció que s'utilitza per mesurar la qualitat d'un split. El valor predeterminat és "gini", que mesura la impuresa de la divisió en funció de l'índex d'impureses de Gini. També es pot utilitzar "entropia" per mesurar la impuresa de la divisió en funció del guany d'informació.
- **splitter:** Determina l'estrategia utilitzada per triar la divisió a cada node. El valor per defecte és "best", el que significa que l'algorisme intentarà trobar la millor divisió segons el criteri. També es pot utilitzar "random" per seleccionar la divisió a l'atzar.
- **max_depth:** Aquesta és la profunditat màxima de l'arbre. El valor per defecte és *None*, el que significa que l'arbre pot créixer indefinidament fins que totes les fulles siguin pures. Establir una profunditat màxima pot ajudar a evitar un sobreajustament.
- **min_samples_split:** Aquest és el nombre mínim de mostres necessàries per dividir un node intern. El valor predeterminat és 2. Augmentar aquest valor pot ajudar a evitar un sobreajustament.
- **min_samples_leaf:** Aquest és el nombre mínim de mostres necessàries per estar en un node fulla. El valor predeterminat és 1. Augmentar aquest valor pot ajudar a evitar un sobreajustament.
- **max_features:** Aquest és el nombre màxim de funcions considerades a l'hora de buscar la millor divisió. El valor predeterminat és *None*, el que significa que es tenen en compte totes les característiques. Establir un nombre màxim de funcions pot ajudar a evitar el sobreajustament.
- **random_state:** Aquesta és la llavor utilitzada pel generador de números aleatoris. Establir aquest valor us permet reproduir els mateixos resultats si entrenem el model diverses vegades.

És important ajustar aquests hiperparàmetres amb cura, ja que poden afectar significativament el rendiment del model.

```
In [ ]: warnings.filterwarnings("ignore")

params = {
    "criterion" : ["gini", "entropy", "log_loss"],
    "splitter" : ["best", "random"],
    "max_depth": [None, 2, 3, 4],
    "min_samples_split": [1, 2, 3],
    "min_samples_leaf" : [1, 2, 3, 4, 5],
    "max_features": ["auto", "sqrt", "log2"],
}

grid = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=params, n_jobs=-1)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_
```

```
{'criterion': 'entropy', 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 2, 'min_samples_split': 1, 'splitter': 'random'}  
Out[ ]: 0.8346064516129033
```

Aplicuem els paràmetres òptims aconseguits:

```
In [ ]: decission_tree = DecisionTreeClassifier(  
        criterion="entropy",  
        max_depth=None,  
        max_features="log2",  
        min_samples_split=1,  
        min_samples_leaf=2,  
        random_state=42,  
        splitter="random").fit(x_train, y_train)  
  
# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de renda  
tree_predicts = []  
for i in range(10):  
    tree_predicts.append(round(accuracy_score(decission_tree.predict(x_test),y_t  
  
tree_pred = np.array(tree_predicts).mean()  
  
print(f"La precisió és del {tree_pred}%")
```

La precisió és del 77.24%

2.1.4. Comparació dels rendiments

```
In [ ]: print(f'Regressió Logistica: {log_pred}, Perceptró: {percep_pred}, Boscos aleatoris: {tree_pred}')
```

Regressió Logistica: 80.6, Perceptró: 78.73, Boscos aleatoris: 80.6, Arbres de decisió: 77.24

Per tant, procedim a comparar aquests rendiments per veure quant millor és un que l'altre. El que farem serà comparar quant de millor és el model de boscos aleatoris, ja que és el que millor resultat ens ha donat.

```
In [ ]: print(f"El model de Boscos aleatoris és un {round(((for_pred/log_pred) - 1)) * 100} % millor que la Regressió Logistica")  
print(f"El model de Boscos aleatoris és un {round(((for_pred/percep_pred) - 1)) * 100} % millor que el Perceptró")  
print(f"El model de Boscos aleatoris és un {round(((for_pred/tree_pred) - 1)) * 100} % millor que el d'Arbres de decisió")
```

El model de Boscos aleatoris és un 0.0% millor que la Regressió Logistica
El model de Boscos aleatoris és un 2.38% millor que el Perceptró
El model de Boscos aleatoris és un 4.35% millor que el d'Arbres de decisió

2.2. Model reduït

En aquesta secció seguirem la mateixa metodologia que amb el model complet. Per tant, no comentarem explícitament totes les passes, ja que són les mateixes.

```
In [ ]: scaler = preprocessing.MinMaxScaler()  
  
y = model_fit["Survived"]  
x = model_fit.drop("Survived", axis=1)  
  
# Escalam les dades en funció de x
```

```

x = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=x_train.shape, x_test.shape, y_train.shape, y_test.shape)

```

Out[]: ((623, 9), (268, 9), (623,), (268,))

2.2.1. Regressió Logistica

```

In [ ]: warnings.filterwarnings("ignore")

params = {
    "penalty": ["l1", "l2", "elasticnet", None],
    "tol": [0.000001, 0.00001, 0.0001, 0.001],
    "C": [0.1, 0.2, 0.21, 0.22, 0.23, 0.24, 0.25],
    "fit_intercept": [True, False],
    "class_weight": ["balanced", None],
    "solver": ["lbfgs", "liblinear", "newton-cg", "newton-cholesky", "sag", "saga"],
    "max_iter": [20, 25, 50, 100],
    "multi_class": ["auto", "ovr", "multinomial"],
}
grid = GridSearchCV(estimator=LogisticRegression(), param_grid=params, n_jobs=4)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'C': 0.21, 'class_weight': None, 'fit_intercept': True, 'max_iter': 25, 'multi_class': 'multinomial', 'penalty': None, 'solver': 'sag', 'tol': 1e-05}

Out[ ]: 0.8121548387096773

```

Apliquem els paràmetres òptims aconseguits:

```

In [ ]: log = LogisticRegression(
    C= 0.21,
    class_weight=None,
    fit_intercept=True,
    max_iter= 25,
    multi_class="multinomial",
    penalty=None,
    solver="sag",
    tol=0.0001,
    random_state=42).fit(x_train, y_train)
y_hat = log.predict(x_test)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendibilitat
logistic_predicts = []
for i in range(10):
    logistic_predicts.append(round(accuracy_score(log.predict(x_test), y_test)*100))

log_pred = np.array(logistic_predicts).mean()

print('The accuracy is', log_pred)

```

The accuracy is 73.51

2.2.2. Perceptró

```
In [ ]: warnings.filterwarnings("ignore")

params = {
    "penalty" : ["l2", "l1", "elasticnet", None],
    "alpha" : [0.00001, 0.0001, 0.001, 0.01, 0.1],
    "fit_intercept" : [True, False],
    "max_iter" : [50, 100, 150, 200],
    "tol" : [0.0001, 0.001, 0.01, 0.1],
    "shuffle" : [True, False],
    "early_stopping" : [True, False],
    "validation_fraction" : [0.0001, 0.001, 0.01, 0.1]
}

grid = GridSearchCV(estimator=Perceptron(), param_grid=params, n_jobs=4)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'alpha': 0.0001, 'early_stopping': False, 'fit_intercept': False, 'max_iter': 50, 'penalty': 'l1', 'shuffle': True, 'tol': 0.0001, 'validation_fraction': 1e-05}

Out[ ]: 0.7896387096774193
```

Apliquem els paràmetres òptims aconseguits:

```
In [ ]: perc = Perceptron(
    alpha=0.0001,
    early_stopping=False,
    fit_intercept=False,
    max_iter=50,
    penalty="l1",
    shuffle=True,
    tol=0.0001,
    validation_fraction=0.0001,
    random_state=42).fit(x_train, y_train)
y_hat = perc.predict(x_test)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendibilitat
perceptron_predicts = []
for i in range(10):
    perceptron_predicts.append(round(accuracy_score(perc.predict(x_test), y_test)))

percep_pred = np.array(perceptron_predicts).mean()

print('The accuracy is', percep_pred)

The accuracy is 77.24
```

2.2.3. Arbres de decisió

A continuació utilitzarem el *Random Forest Classifier*:

```
In [ ]: warnings.filterwarnings("ignore")
```

```

params = {
    "n_estimators" : [10, 20, 30, 40],
    "criterion" : ["gini", "entropy", "log_loss"],
    "max_depth" : [None, 3, 4, 5],
    "min_samples_split": [2, 3, 4],
    "min_samples_leaf": [1, 2, 3],
    "max_features" : ["sqrt", "log2"],
    "bootstrap" : [True, False],
    "class_weight" : ["balanced", "balanced_subsample"],
}
grid = GridSearchCV(estimator=RandomForestClassifier(), param_grid=params, n_jobs=-1)
grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_
{'bootstrap': True, 'class_weight': 'balanced_subsample', 'criterion': 'gini',
 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 10}

Out[ ]: 0.8089548387096773

```

Apliquem els paràmetres òptims aconseguits:

```

In [ ]: forest = RandomForestClassifier(
    bootstrap=True,
    class_weight="balanced_subsample",
    criterion="gini",
    max_depth=None,
    max_features="sqrt",
    min_samples_leaf=1,
    min_samples_split=4,
    n_estimators=10,
    random_state=42).fit(x_train, y_train)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendibilitat
forest_predicts = []
for i in range(10):
    forest_predicts.append(round(accuracy_score(forest.predict(x_test),y_test)*100))

for_pred = np.array(forest_predicts).mean()

print('The accuracy is', for_pred)

```

The accuracy is 77.99

A continuació utilitzarem el *Decision Tree Classifier*:

```

In [ ]: warnings.filterwarnings("ignore")

params = {
    "criterion" : ["gini", "entropy", "log_loss"],
    "splitter" : ["best", "random"],
    "max_depth": [None, 2, 3, 4, 5, 6, 7],
    "min_samples_split": [1, 2, 3],
    "min_samples_leaf" : [1, 2, 3],
    "max_features": ["auto", "sqrt", "log2"],
}

```

```

grid = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=params, n_jobs=-1)

grid.fit(x_train, y_train)

print(grid.best_params_)
grid.best_score_

{'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2', 'min_samples_leaf': 3, 'min_samples_split': 3, 'splitter': 'random'}

Out[ ]: 0.8137419354838709

```

Apliquem els paràmetres òptims aconseguits:

```

In [ ]: tree = DecisionTreeClassifier(
            criterion="entropy",
            max_depth=4,
            max_features="log2",
            min_samples_leaf=3,
            min_samples_split=3,
            splitter="random",
            random_state=42).fit(x_train, y_train)

# Array de 10 valors per fer-ne la mitjana per a la posterior comparació de rendiment
tree_predicts = []
for i in range(10):
    tree_predicts.append(round(accuracy_score(tree.predict(x_test), y_test)*100,2))

tree_pred = np.array(tree_predicts).mean()

print('The accuracy is', tree_pred)

```

The accuracy is 77.99

2.2.4. Comparació de rendiments

```

In [ ]: print(f'Regressió Logistica: {log_pred}, Percptró: {percep_pred}, Boscos aleatoris: {tree_pred}, Decisió: {dec_pred}')

```

Regressió Logistica: 73.51, Percptró: 77.24, Boscos aleatori: 77.99, Decisió: 81.72

Per tant, procedim a comparar aquests rendiments per veure quant millor és un que l'altre. El que farem serà comparar quant de millor és el model de boscos aleatori, ja que és el que millor resultat ens ha donat.

```

In [ ]: print(f"El model de Boscos aleatori és un {round(((for_pred/log_pred) - 1)) * 100} % millor que la Regressió Logistica")
print(f"El model de Boscos aleatori és un {round(((for_pred/percep_pred) - 1)) * 100} % millor que el Perceptró")
print(f"El model de Boscos aleatori és un {round(((for_pred/tree_pred) - 1)) * 100} % millor que el d'Arbres de decisió")

```

El model de Boscos aleatori és un 6.09% millor que la Regressió Logistica
 El model de Boscos aleatori és un 0.97% millor que el Perceptró
 El model de Boscos aleatori és un -4.56% millor que el d'Arbres de decisió

Llavors, podem concloure, que el model amb totes les variables ens dona resultats més dispers entre les diferents prediccions; però ens aporta una major precisió. Concretament en el model de *Random Forest Regression* ens dona una precisió superior al 80%. És per això, que haurem d'agafar el model amb el màxim de variables per a poder tenir una

major perspectiva de com influeixen aquestes en la predicció. Això es comentarà en el següent apartat.

3. Anàlisis de les característiques

La *feature importance* és una mesura de quant contribueix una característica a la predicció d'una variable objectiu en un model d'aprenentatge automàtic. S'utilitza habitualment en algorismes d'aprenentatge supervisat per identificar les característiques més importants per fer prediccions precises.

Hi ha diverses maneres de calcular la *feature importance* en l'aprenentatge automàtic, incloses les següents:

- **Disminució de la mitjana d'impuresa:** Aquest mètode calcula la disminució de la impuresa causada per cada característica quan s'utilitza per dividir les dades. La impuresa es mesura mitjançant un criteri com l'índex d'impureses de Gini o el guany d'informació.
- **Precisió de la disminució de la mitjana:** Aquest mètode calcula la disminució de la precisió del model causada per permutar (remenar aleatoriament) els valors de cada característica. La idea és que si una característica és important, remenar els seus valors hauria de disminuir significativament la precisió del model.
- **Valors dels coeficients:** En models lineals, com ara la regressió lineal i la regressió logística, la magnitud dels coeficients associats a cada característica es pot utilitzar per determinar la seva importància. Els coeficients més grans indiquen una relació més forta amb la variable objectiu.

En aquest cas, ens centrarem en l'última.

```
In [ ]: scaler = preprocessing.MinMaxScaler()

y = model["Survived"]
x = model.drop("Survived", axis=1)

# Escalam les dades en funció de x
x = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify=x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
Out[ ]: ((623, 18), (268, 18), (623,), (268,))
```

3.1. Regressió Logistica

En la regressió logística, els coeficients representen el canvi en les probabilitats logarítmiques de la variable dependent per a un canvi d'una unitat en la variable predictora (en el nostre cas, la variable *Survived*), mantenint constants totes les altres variables predictores.

Un coeficient positiu indica que a mesura que augmenta el valor de la variable predictora, també augmenta la probabilitat logarítmica de la variable dependent. Això significa que la probabilitat que la variable dependent sigui "verdadera" augmenta a mesura que incrementa el valor de la variable predictora.

D'altra banda, un coeficient negatiu indica que a mesura que augmenta el valor de la variable predictora, la probabilitat logarítmica de la variable dependent disminueix. Això significa que la probabilitat que la variable dependent sigui "verdadera" disminueix a mesura que augmenta el valor de la variable predictora.

És important tenir en compte que la magnitud del coeficient també pot proporcionar informació sobre la força de la relació entre el predictor i les variables dependents. Un coeficient de magnitud més gran indica una relació més forta.

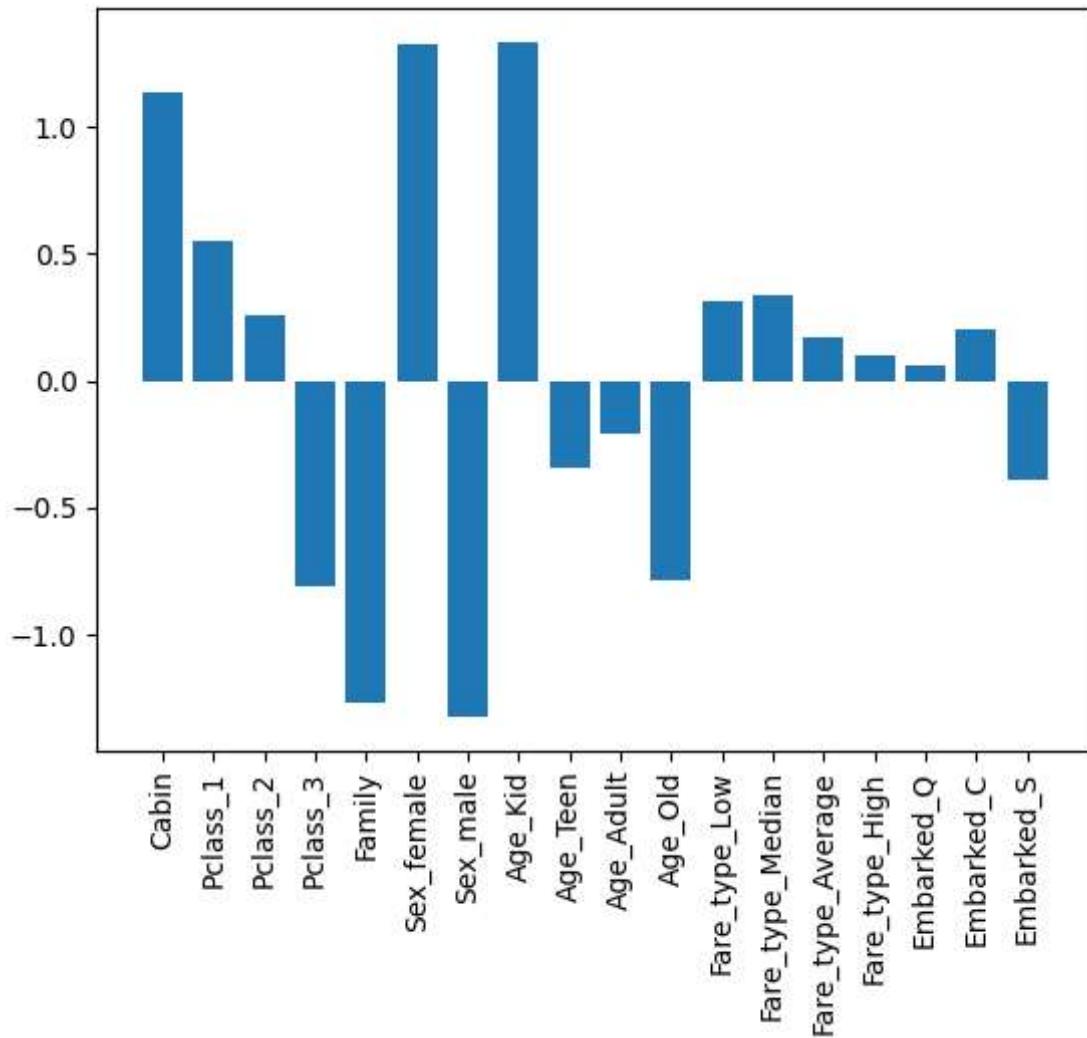
```
In [ ]: log = LogisticRegression()

log.fit(x_train, y_train)

importance = log.coef_[0]

data = {
    "Cabin" : importance[0],
    "Pclass_1" : importance[1],
    "Pclass_2" : importance[2],
    "Pclass_3" : importance[3],
    "Family" : importance[4],
    "Sex_female" : importance[5],
    "Sex_male" : importance[6],
    "Age_Kid" : importance[7],
    "Age_Teen" : importance[8],
    "Age_Adult" : importance[9],
    "Age_Old" : importance[10],
    "Fare_type_Low" : importance[11],
    "Fare_type_Median" : importance[12],
    "Fare_type_Average" : importance[13],
    "Fare_type_High" : importance[14],
    "Embarked_Q" : importance[15],
    "Embarked_C" : importance[16],
    "Embarked_S" : importance[17]
}

plt.bar(data.keys(), data.values())
plt.xticks(rotation="vertical")
plt.show()
```



En aquest cas, les variables que influirien més positivament serien *Cabin*, *Sex_female* i *Age_Kid*. Això ens indica que les persones amb cabina, dones i fillets petits, tenien més possibilitats de sobreviure al naufragi del Titànic.

D'altra banda, les variables que influirien més negativament a l'hora de sobreviure serien *Pclass_3*, *Family*, *Sex_male* i *Age_Old*. Això ens indica que les persones de tercera classe (les més pobres), amb família a bord, homes i ancians, tenien menys possibilitats de sobreviure al naufragi.

3.2. Perceptró

En un perceptró, els coeficients representen els pesos assignats a cada característica d'entrada del model. El perceptró fa prediccions calculant una suma ponderada de les característiques d'entrada i aplicant una funció de pas al resultat.

Un coeficient positiu indica que la funció d'entrada corresponent té una influència positiva en la predicción feta pel perceptró. Això vol dir que un augment del valor de la característica d'entrada s'associa amb un augment de la predicción feta pel perceptró.

D'altra banda, un coeficient negatiu indica que la funció d'entrada corresponent té una influència negativa en la predicción feta pel perceptró. Això significa que un augment del

valor de la característica d'entrada s'associa amb una disminució de la predicción feta pel perceptró.

És important tenir en compte que la magnitud del coeficient també pot proporcionar informació sobre la força de la influència de la característica d'entrada en la predicción. Un coeficient de magnitud més gran indica una influència més forta.

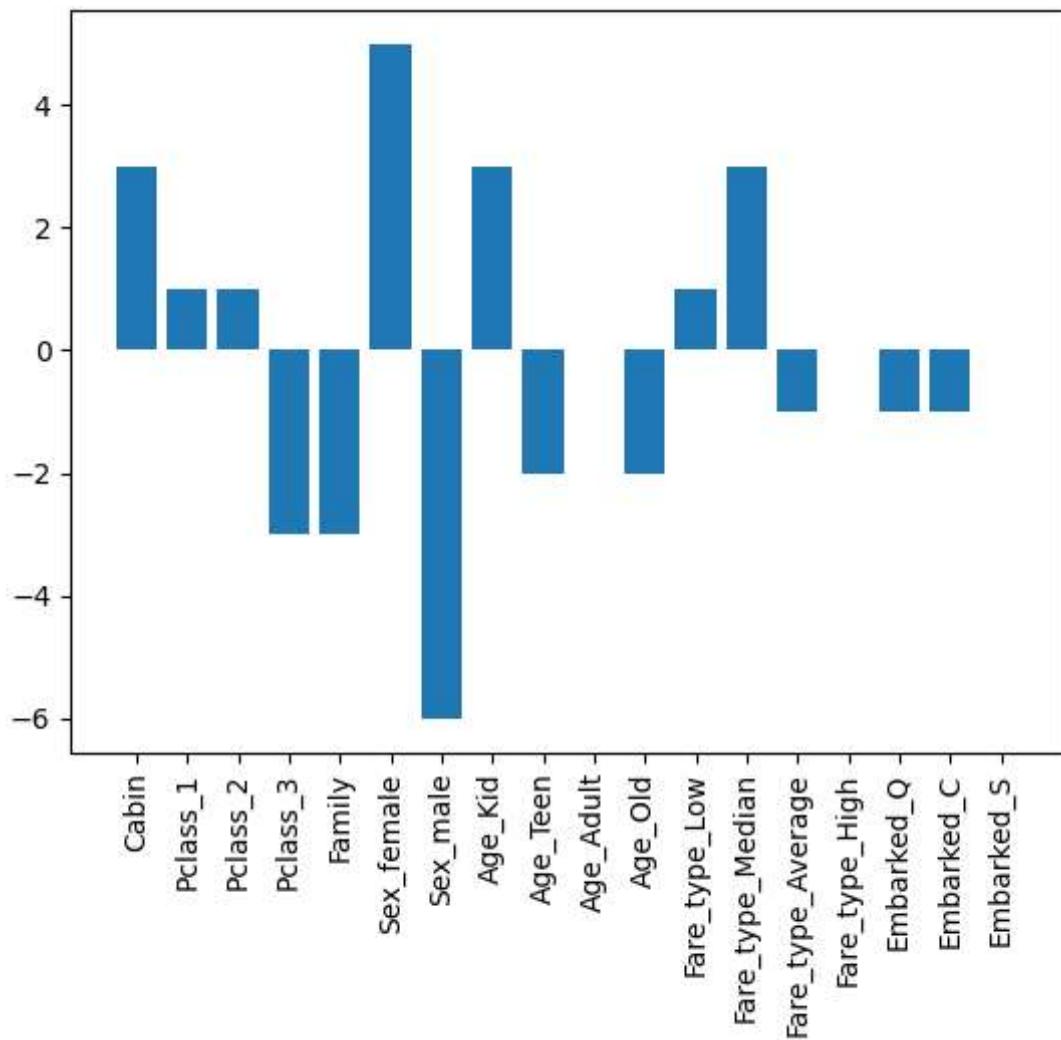
```
In [ ]: perc = Perceptron()

perc.fit(x_train, y_train)

importance = perc.coef_[0]

data = {
    "Cabin" : importance[0],
    "Pclass_1" : importance[1],
    "Pclass_2" : importance[2],
    "Pclass_3" : importance[3],
    "Family" : importance[4],
    "Sex_female" : importance[5],
    "Sex_male" : importance[6],
    "Age_Kid" : importance[7],
    "Age_Teen" : importance[8],
    "Age_Adult" : importance[9],
    "Age_Old" : importance[10],
    "Fare_type_Low" : importance[11],
    "Fare_type_Median" : importance[12],
    "Fare_type_Average" : importance[13],
    "Fare_type_High" : importance[14],
    "Embarked_Q" : importance[15],
    "Embarked_C" : importance[16],
    "Embarked_S" : importance[17]
}

plt.bar(data.keys(), data.values())
plt.xticks(rotation="vertical")
plt.show()
```



En aquest cas, les variables que influirien més positivament serien *Cabin*, *Sex_female*, *Age_Kid* i *Fare_type_Average*. Això ens indica que les persones amb cabina, dones, fills petits i poder adquisitiu mitjà, tenen més possibilitats de sobreviure al naufragi del Titànic.

D'altra banda, les variables que influirien més negativament a l'hora de sobreviure serien *Pclass_3*, *Family* i *Sex_male*. Això ens indica que les persones de tercera classe (les més pobres), amb família a bord i homes, tenen menys possibilitats de sobreviure al naufragi.

3.3. Boscos aleatoris

En un classificador de boscos aleatoris, els valors d'importància de les característiques representen quant contribueix cada característica a la reducció global del criteri utilitzat per dividir els nodes en els arbres de decisió. Els valors d'importància de les característiques es calculen a partir de la reducció mitjana del criteri en tots els arbres del bosc.

Un valor alt d'importància de la característica indica que la característica corresponent és important per fer prediccions precises. És probable que una característica amb un valor d'importància elevada sigui un fort predictor per a la variable objectiu.

D'altra banda, un valor d'importància de la característica baixa indica que la característica corresponent no és tan important per fer prediccions precises. Una característica amb un valor d'importància baixa pot no tenir una relació forta amb la variable objectiu o pot estar altament correlacionada amb altres característiques, en aquest cas el classificador de bosc aleatori pot optar per confiar en les altres característiques.

És important tenir en compte que els valors d'importància de les característiques són relatius entre si, el que significa que els valors s'han d'interpretar en relació amb els altres valors d'importància de les característiques del mateix model. Per exemple, si una característica té un valor d'importància de 0,5 i una altra característica té un valor d'importància de 0,1, vol dir que la primera característica és més important que la segona, però no vol dir necessàriament que la primera característica sigui més important en termes absoluts.

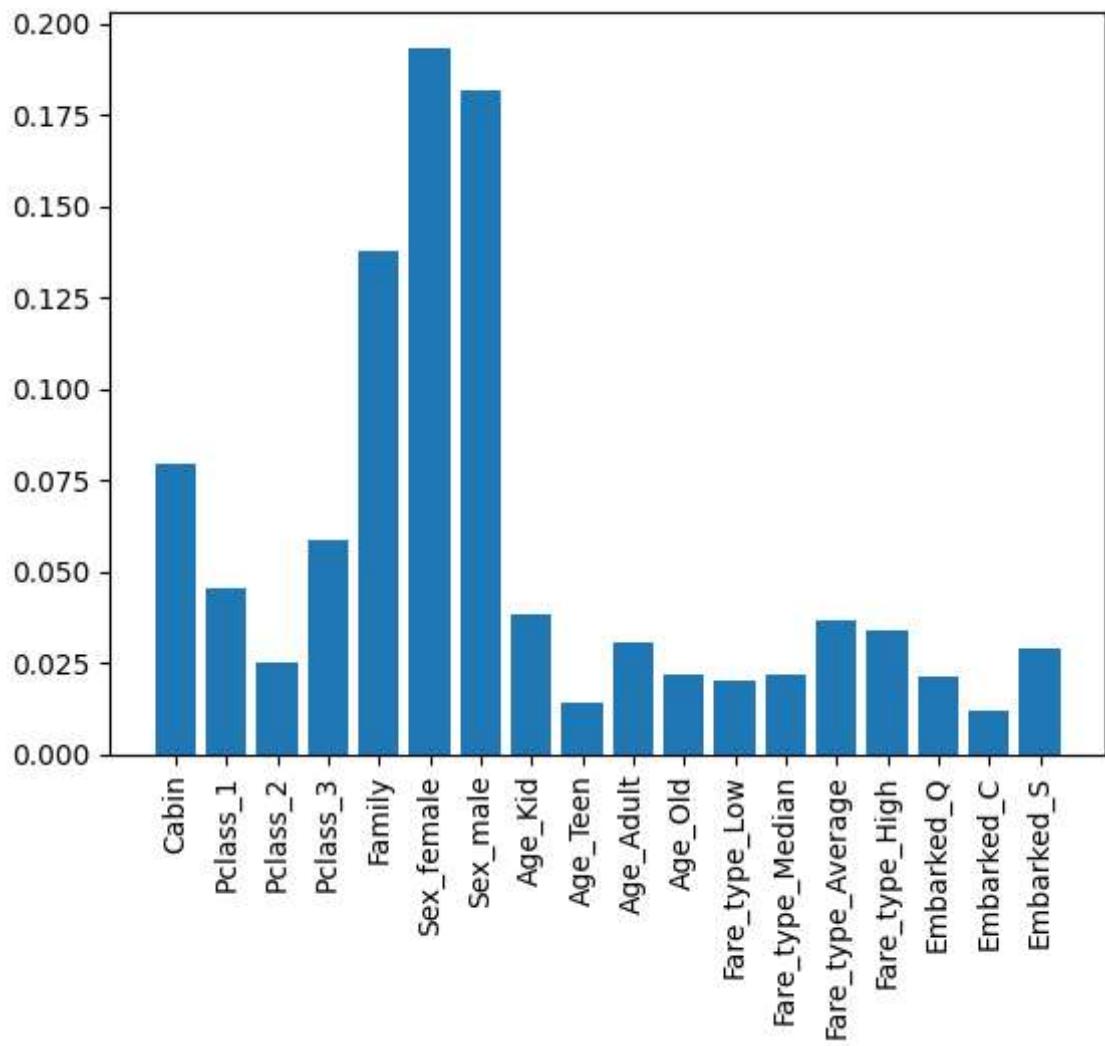
```
In [ ]: forest = RandomForestClassifier()

forest.fit(x_train, y_train)

importance = forest.feature_importances_

data = {
    "Cabin" : importance[0],
    "Pclass_1" : importance[1],
    "Pclass_2" : importance[2],
    "Pclass_3" : importance[3],
    "Family" : importance[4],
    "Sex_female" : importance[5],
    "Sex_male" : importance[6],
    "Age_Kid" : importance[7],
    "Age_Teen" : importance[8],
    "Age_Adult" : importance[9],
    "Age_Old" : importance[10],
    "Fare_type_Low" : importance[11],
    "Fare_type_Median" : importance[12],
    "Fare_type_Average" : importance[13],
    "Fare_type_High" : importance[14],
    "Embarked_Q" : importance[15],
    "Embarked_C" : importance[16],
    "Embarked_S" : importance[17]
}

plt.bar(data.keys(), data.values())
plt.xticks(rotation="vertical")
plt.show()
```



En aquest cas, les variables que influirien més serien *Cabin*, *Family*, *Sex_female* i *Sex_male*. Això ens indica que si una persona tenia o no cabina, familiars a bord, era una dona o un home, influiria molt a l'hora de sobreviure al naufragi del Titànic.

Segurament, seguiria un patró similar al dels coeficients dels predictors anteriors i el fet de tenir cabina o ser dona influiria positivament a la probabilitat de sobreviure i el fet de tenir familiars a bord o ser home, negativament.

3.4. Arbres de decisió

En un classificador d'arbres de decisió, els valors d'importància de les característiques representen quant contribueix cada característica a la reducció global del criteri utilitzat per dividir els nodes de l'arbre (igual que en el de boscos aleatoris). Els valors d'importància de la característica es calculen en funció de la reducció mitjana del criteri en tots els nodes que utilitzen la característica per dividir-la.

Un valor alt d'importància de la característica indica que la característica corresponent és important per fer prediccions precises. És probable que una característica amb un valor d'importància elevada sigui un fort predictor per a la variable objectiu.

D'altra banda, un valor d'importància de la característica baixa indica que la característica corresponent no és tan important per fer prediccions precises. Una característica amb un

valor d'importància baixa pot no tenir una relació forta amb la variable objectiu o pot estar altament correlacionada amb altres característiques, en aquest cas el classificador de l'arbre de decisions pot optar per confiar en les altres característiques.

És important tenir en compte que els valors d'importància de les característiques són relatius entre si, el que significa que els valors s'han d'interpretar en relació amb els altres valors d'importància de les característiques del mateix model.

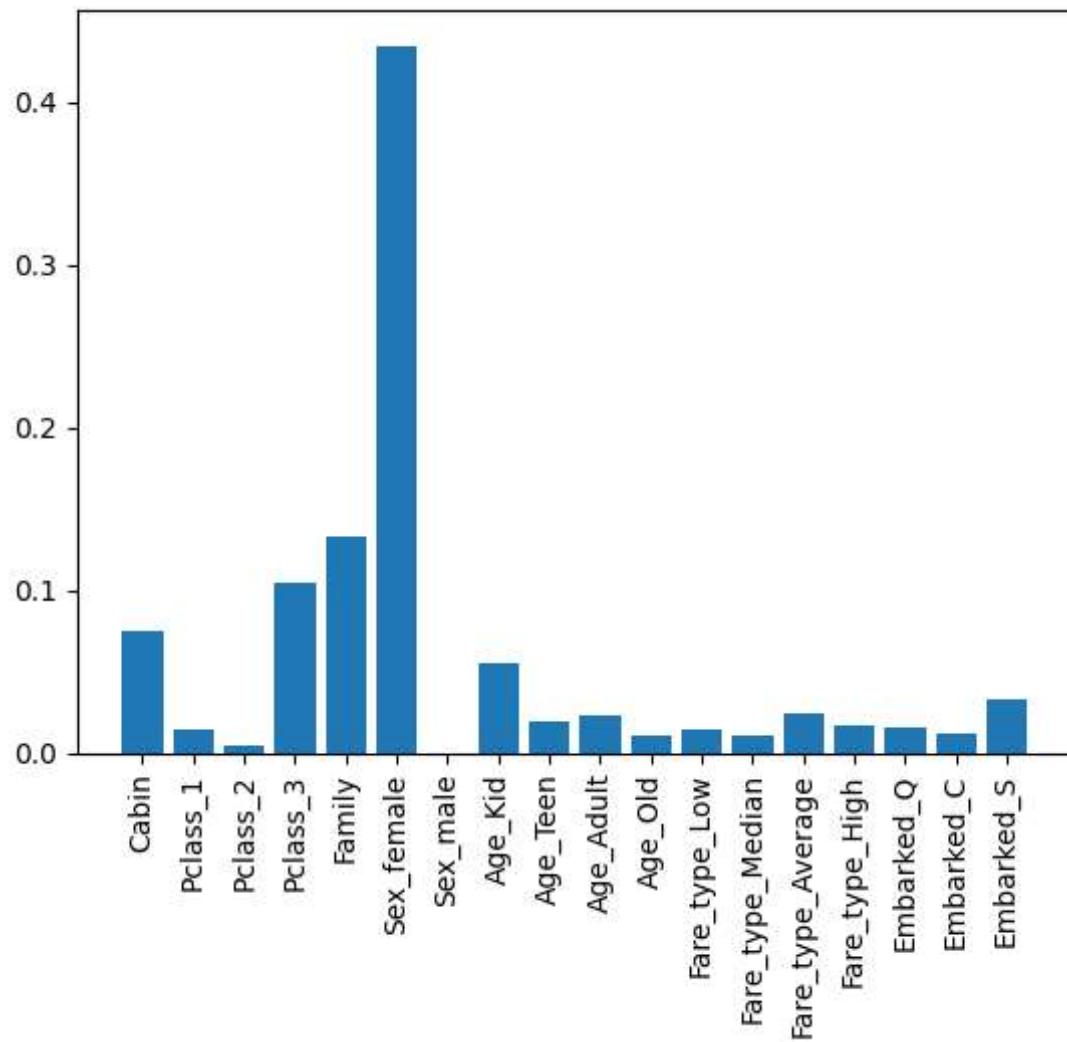
```
In [ ]: tree = DecisionTreeClassifier()

tree.fit(x_train, y_train)

importance = tree.feature_importances_

data = {
    "Cabin" : importance[0],
    "Pclass_1" : importance[1],
    "Pclass_2" : importance[2],
    "Pclass_3" : importance[3],
    "Family" : importance[4],
    "Sex_female" : importance[5],
    "Sex_male" : importance[6],
    "Age_Kid" : importance[7],
    "Age_Teen" : importance[8],
    "Age_Adult" : importance[9],
    "Age_Old" : importance[10],
    "Fare_type_Low" : importance[11],
    "Fare_type_Median" : importance[12],
    "Fare_type_Average" : importance[13],
    "Fare_type_High" : importance[14],
    "Embarked_Q" : importance[15],
    "Embarked_C" : importance[16],
    "Embarked_S" : importance[17]
}

plt.bar(data.keys(), data.values())
plt.xticks(rotation="vertical")
plt.show()
```



En aquest cas, la variable que destaca més és *Sex_female*. Això ens indica que si una persona era dona, segurament tenia moltes possibilitats de sobreviure.