Práctica 2 (35 % nota final)

El enlace a github de esta práctica es: https://github.com/jdemas001/PRA2_JdM

Componentes del grupo

Esta PRAC1 ha sido realizada de forma individual por Jordi de Mas, debido a la dificultad para poder encontrar franjas horarias compatibles con otros compañeros de asignatura a causa de mis obligaciones laborales. El hecho de que nuestra profesora enviara una comunicación indicando que se podía hacer individualmente, ha facilitado mucho encontrar una solución al problema.

Descripción de la Práctica a realizar

El objetivo de esta actividad será el tratamiento del dataset "Titanic: Machine Learning from Disaster" (https://www.kaggle.com/c/titanic).

A continuación pasaremos a la realización de las diferentes etapas del proyecto analítico.

1.- Descripción del dataset

El fichero "Titanic: Machine Learning from Disaster" recoge los datos de los pasajeros que viajaban en este barco durante su viaje inaugural el 15 de Abril de 1912. Este buque era considerado imposible de hundir, pero al colisionar con un iceberg, naufragó. Desgraciadamente, no había suficientes botes salvavidas para todos los pasajeros, lo que ocasionó la muerte de 1502 de los 2224 pasajeros y tripulación que viajaban en él en ese momento.

Posiblemente hubo algún factor suerte relacionado con la supervivencia, pero parece ser que algunos grupos de personas fueron más afortunados que otros respecto a la supervivencia.

De lo que se trata, es de construir un modelo predictivo que responda a la pregunta: ¿Qué grupos de personas tenían más probabilidades de sobrevivir?

Los ficheros con los que trabajaremos son dos:

- "Train.csv": contiene un subconjunto de datos de 891 de los pasajeros a bordo, juntamente con la indicación de si sobrevivieron o no.

- "Test.csv": contiene otro subconjunto de datos de 418 de los pasajeros, pero sin la información correspondiente a si sobrevivieron o no. Eso es lo que debemos predecir.

Es importante que conozcamos la información contenida en los ficheros, puesto que es la base sobre la que vamos a trabajar y con la que podremos realizar el análisis y la predicción.

La información contenida en los ficheros referente a los pasajeros es la siguiente:

Atributo	**Descripción**
PassengerId	Identificador del pasajero
Survived	Indica si sobrevivió (Dato sólo presente en fichero Train): 0: No 1: Si
Pclass	Clase del pasajero (status social / económico): 1st – Alto 2nd – Medio 3rd - Bajo
Name	Nombre del pasajero
Sex	Género del pasajero
Age	Edad (fraccional si menor que 1). En años.
SibSp	Relación familiar: Sibling: hermano/a, hermanastro/a Spouse: marido / esposa (amantes y novias ignoradas)
Parch	Relación familiar:

Atributo	Descripción
	Parent: padre, madre Child: hijo/a, hijastro/a Algunos niños viajaban sólo con una nanny (Parch = 0)
Ticket	Número del billete
Fare	Tarifa del billete
Cabin	Camarote
Embarked	Puerto de embarque: C: Cherbourg Q: Queenstown S: Southampton

Veamos a continuación algunos registros contenidos en el fichero “train”:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Seguidamente comprobamos los tipos de datos y la existencia de “missing values”.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Podemos ver que tres columnas contienen “missing values”: Age, Cabin y Embarked. Esto es importante porque posteriormente, en la etapa de limpieza de datos, se deberán tratar.

2.- Integración y selección de los datos de interés a analizar

La mayoría de los atributos contenidos en el conjunto de datos se corresponden con características que se pueden tener en consideración para realizar los análisis. De todos modos, hay algunos campos de los que, a priori, podríamos prescindir, como: Name, Ticket o Fare. Dichos campos (Nombre del pasajero, número de billete y tarifa del billete) no parecen tener excesiva relación con la posibilidad de supervivencia en el naufragio del buque.

De la misma manera, podemos prescindir del campo Cabin, puesto que el número de cabina no parece tener, en principio, mucha significación respecto a si se sobrevive o no. Aunque podríamos pensar que el número de camarote, por su posición en el buque (cubierta, etc.) podría tener su sentido en cuanto a determinar la supervivencia, creo que eso ya está cubierto con la clase (Pclass) del pasaje. Además, el campo Cabin tiene una gran cantidad de missing values (77,10%).

El campo PassengerId también se podría eliminar, pero por el momento, lo dejamos.

3.- Limpieza de los datos

La limpieza de datos la efectuaremos sobre los conjuntos de train y test.

En primer lugar, excluiremos del conjunto de datos las columnas que hemos indicado en el punto anterior como prescindibles y haremos el análisis con las restantes.

Una vez eliminadas las columnas, el conjunto de datos queda de la siguiente manera:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	1	0	3	male	22.0	1	0	S
1	2	1	1	female	38.0	1	0	C
2	3	1	3	female	26.0	0	0	S
3	4	1	1	female	35.0	1	0	S
4	5	0	3	male	35.0	0	0	S

3.1.- Missing values y conversión variables categóricas a números

A continuación procederemos a tratar los missing values para los campos Embarked y Age.

Para el campo Embarked hay dos missing values. Al ser sólo dos. Creo que lo mejor es rellenarlos con el puerto de embarque mayoritario en ese fichero.

Para ello, hacemos un describe de la columna Embarked. El resultado es el siguiente:

```
count     889
unique      3
top         S
freq      644
Name: Embarked, dtype: object
```

Vemos que el puerto de Southampton es dónde embarcó la mayoría del pasaje, por lo que ese será el valor con el que rellenaremos los dos registros con missing values en ese campo.

Los missing values del campo Age los rellenaremos con la mediana de las edades de los pasajeros contenidos en el fichero.

Una vez realizados ambos cambios comprobamos la información de los datos del fichero. El resultado es el siguiente:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Sex            891 non-null object
Age            891 non-null object
SibSp          891 non-null int64
Parch          891 non-null int64
Embarked       891 non-null object
dtypes: int64(5), object(3)
memory usage: 55.8+ KB
```

Podemos apreciar que el conjunto de datos ya no contiene missing values.

A continuación procedemos a cambiar variables categóricas por valores numéricos.

En primer lugar modificamos los valores de la variable Sex con los siguientes valores:

- Male: 0
- Female: 1

Después pasamos los valores de Age, de tipo float a tipo int.

Y finalmente convertimos los datos correspondientes a los puertos de embarque a números:

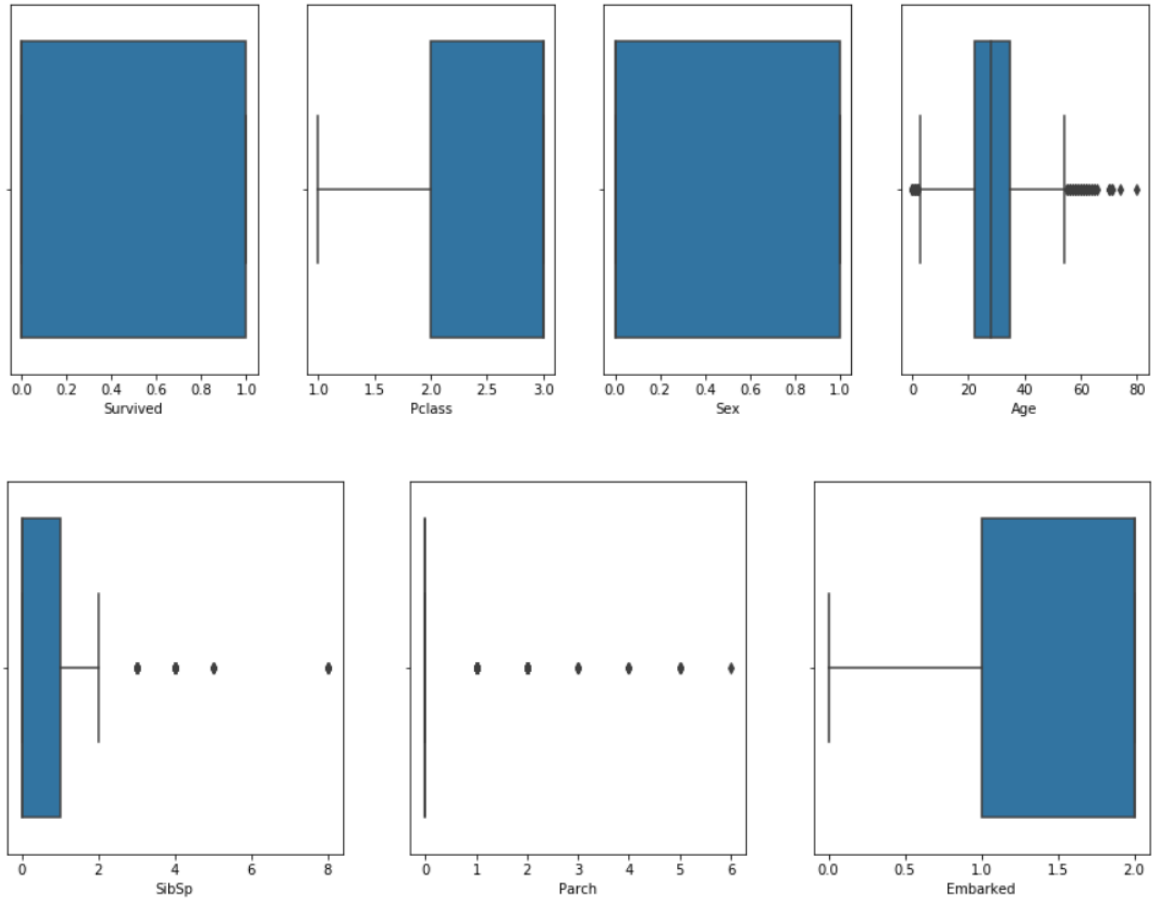
- C pasa a ser 0
- Q pasa a ser 1
- S pasa a ser 2

Vemos como quedan los primeros registros del fichero tras realizar los cambios:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	1	0	3	0	22	1	0	2
1	2	1	1	1	38	1	0	0
2	3	1	3	1	26	0	0	2
3	4	1	1	1	35	1	0	2
4	5	0	3	0	35	0	0	2

3.2.- Identificación y tratamiento de valores extremos

Hacemos una identificación de outliers en cada una de las variables del conjunto de datos. El resultado es el siguiente:



Como podemos apreciar, hay tres variables con valores extremos: Age, SibSp y Parch.

Podemos pensar en crear intervalos de edades para evitar este problema. Pero finalmente decidimos dejarlo como está ya que los valores de las edades son perfectamente correctos y se pueden generar valores NaN en caso de hacerlo.

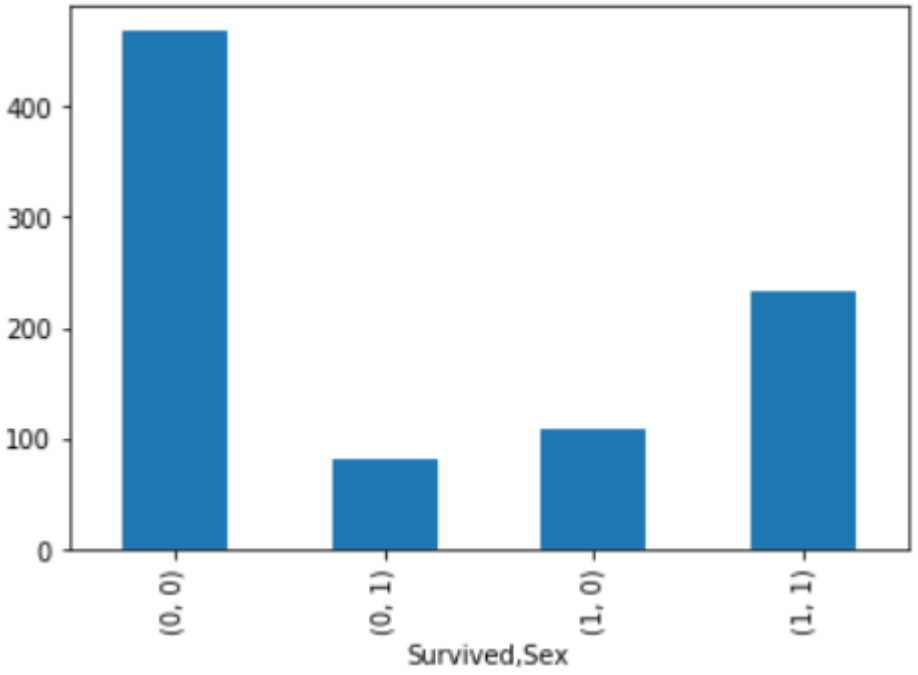
Las otras dos variables las dejamos igual por el momento. Los valores extremos que se dan en las variables SibSp y Parch pueden darse, por lo que en estos dos casos, el tratamiento de los valores extremos consiste en dejarlos tal como están.

4.- Análisis de los datos

4.1.- Selección de los grupos de datos a analizar

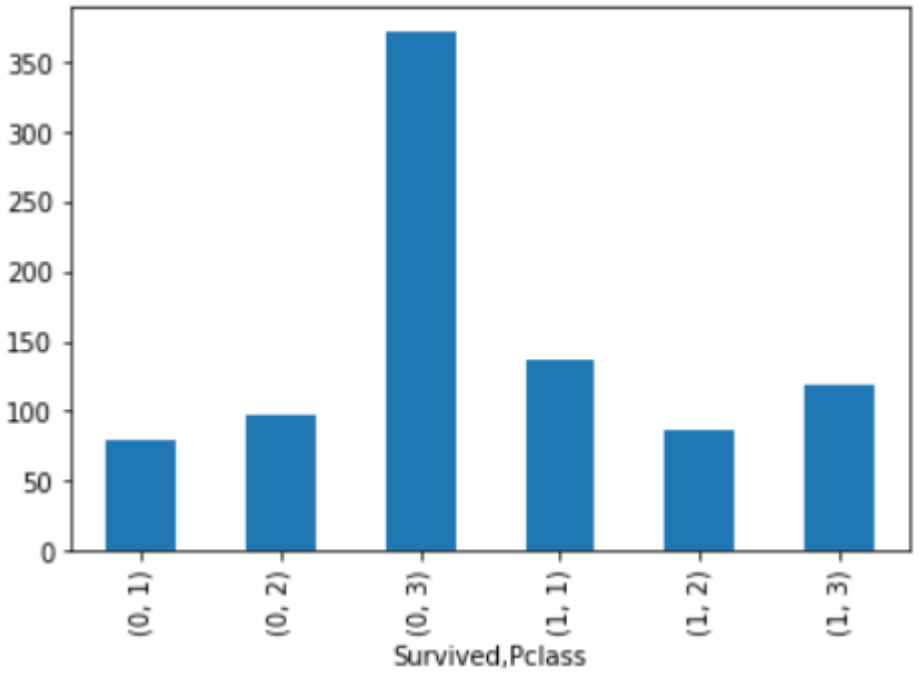
Seleccionamos los grupos de datos que pueden ser interesantes para analizar, aunque no es seguro que todos se utilicen.

La variable Sex es una de las más importantes para averiguar quien tiene más probabilidad de salvarse.



Observando el gráfico, podemos comprobar que se salvaron más mujeres que hombres. En el caso de los que no sobrevivieron al naufragio, se puede apreciar una diferencia muy grande entre hombres (0) y mujeres (1).

Veamos a continuación, el análisis por clase (Pclass).



Se puede apreciar que la gran mayoría de los pasajeros de la clase más baja (3) murieron en el naufragio. Y los que más consiguieron salvarse fueron los de clase alta (1).

4.2.- Comprobación de la normalidad y homogeneidad de la varianza

Usaremos el test de normalidad de Anderson-Darling para comprobar que los valores de las variables cuantitativas provienen de una población distribuida normalmente. De esta manera comprobamos que para cada prueba se obtiene un p-valor superior al nivel de significación prefijado α = 0,5. Si esto se cumple, se considera que la variable sigue una distribución normal.

Aplicando el test a nuestras variables cuantitativas obtenemos el siguiente resultado:

```
Test Anderson-Darling - Pclass:
15.000: 0.573, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.087, data does not look normal (reject H0)

Test Anderson-Darling - Sex:
15.000: 0.573, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.087, data does not look normal (reject H0)

Test Anderson-Darling - SibSp:
15.000: 0.573, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.087, data does not look normal (reject H0)

Test Anderson-Darling - Parch:
15.000: 0.573, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.087, data does not look normal (reject H0)

Test Anderson-Darling - Embarked:
15.000: 0.573, data does not look normal (reject H0)
10.000: 0.653, data does not look normal (reject H0)
5.000: 0.784, data does not look normal (reject H0)
2.500: 0.914, data does not look normal (reject H0)
1.000: 1.087, data does not look normal (reject H0)
```

Observamos que nuestras variables no siguen una distribución normal.

Como parece que las poblaciones de origen no son normales, es mejor utilizar el test no paramétrico de Fligner-Killeen basado en la mediana. Lo usamos para comprobar la homegeneidad de varianzas para los tres valores presentes en Pclass.

4.3.- Aplicación de modelos de Machine Learning

Entrenaremos varios modelos de Machine Learning y compararemos sus resultados. Para realizar el entrenamiento utilizaremos sólo el fichero de train, ya que es el que contiene el campo con la indicación de si sobrevivieron o no.

En primer lugar separamos del fichero de entrenamiento la columna con la información de los supervivientes.

Seguidamente hemos creado varios modelos de Machine Learning. Veamos que puntuación han obtenido cada uno de ellos:

- Logistic Regression 80,25 %
- K-Nearest Neighbors 76,99 %
- Linear Support Vector Machine 72,05 %
- Decision Tree 100,00 %
- Random Forest 100,00 %

Podemos ver que Decision Tree y Random Forest tienen un porcentaje de acierto del 100% lo que parece un poco exagerado.

Para ver si conseguimos una cifra más realista, vamos a utilizar K-Fold Cross Validation y así evaluaremos tantas veces como particiones hagamos del fichero de entrenamiento.

Decidimos partirlo en diez trozos y procedemos a hacer una nueva evaluación. El resultado de Random Forest ahora arroja una precisión media de 79,25 % con una desviación estándar del 3%.

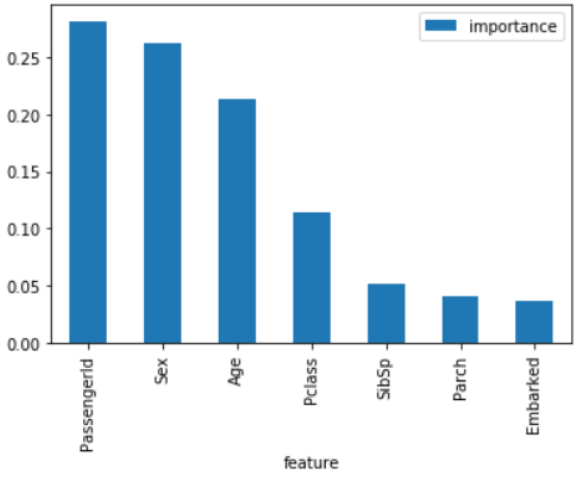
Realizamos la misma operación para Decision Tree usando K-Fold Cross Validation. La precisión media es ahora de 74,76 % con una desviación estándar del 5%.

Ahora los resultados de ambos métodos son mucho más realistas que antes. De ellos, escogemos Random Forest para intentar afinar más el resultado.

Vamos a medir la importancia relativa de cada variable viendo en cuantos nodos se usa dicha variable. Una vez obtenidos los resultados, podremos eliminar aquellas que no tienen mucha importancia para el resultado y después podremos volver a medir la precisión del método.

La importancia relativa de las variables es:

feature	importance
PassengerId	0.282
Sex	0.262
Age	0.214
Pclass	0.114
SibSp	0.052
Parch	0.040
Embarked	0.037



Podríamos eliminar las dos últimas variables o incluso las tres últimas, pero considero que entonces el método de ML se ejecutaría sobre un número demasiado limitado de factores.

5.- Representación de los resultados

Vamos a realizar una representación de los resultados obtenidos a través de diferentes herramientas y usando tablas y/o gráficas.

En primer lugar calculamos una matriz de confusión con el siguiente resultado:

```
array([[473,  76],
       [101, 241]], dtype=int64)
```

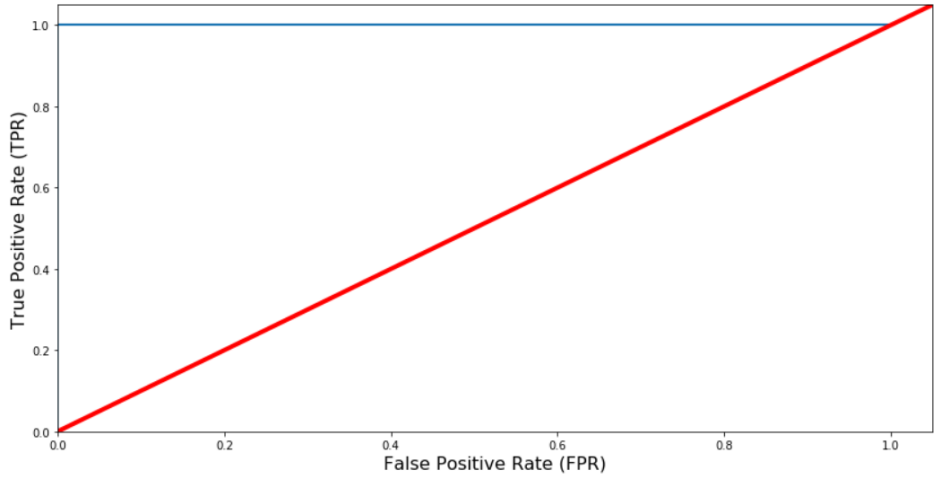
La primera fila de la matriz hace referencia a las predicciones de los no supervivientes: 473 pasajeros han sido correctamente clasificados como no supervivientes (true negatives) y 76 han sido clasificados erróneamente como no supervivientes (false positives).

La segunda fila de la matriz se refiere a las predicciones de los supervivientes: 101 pasajeros han sido clasificados erróneamente como supervivientes (false negatives) y 241 pasajeros se han clasificado correctamente como supervivientes (true positives).

```
Precision: 0.7507987220447284
Recall: 0.6871345029239766
```

La precisión del modelo es del 75,08%. Y el campo Recall nos indica que el modelo ha predecido la supervivencia del 68,71% de las personas que sobrevivieron.

A continuación podremos ver en forma gráfica, la curva (ROC AUC curve) que dibuja la tasa de positivos verdaderos (true positive) que también se conoce por recall contra el ratio de falsos positivos.



La línea roja es sólo un clasificador aleatorio, por lo que nuestro clasificador debe estar lo más lejos posible de esa línea. En nuestro caso, el modelo Random Forest trabaja bastante bien.

Si calculamos el ROC AUC Score da 1, lo que significa que es totalmente correcto. Si hubiera sido 0.5 significaría un clasificador totalmente aleatorio.

Tenemos este resultado porque es el que nos ha dado el modelo Random Forest cuando lo hemos ejecutado. Probablemente hubiera variado si hubiéramos decidido eliminar las dos o tres variables menos relevantes que hemos hallado antes, pero finalmente hemos decidido no hacerlo. Por esa razón el resultado sigue siendo 100%, el gráfico de la curva ROC AUC es como hemos visto (recta en el 1) y el ROC AUC Score da 1.

6.- Resolución del problema y conclusiones

Hemos realizado una descripción del conjunto de datos, hemos seleccionado los datos de interés para el análisis que se debía realizar. Hemos procedido a efectuar una limpieza de datos, eliminando aquellas columnas que no se consideraban determinantes para analizar y predecir los valores de supervivencia de los pasajeros del fichero de test.

Hemos transformado los valores de aquellas variables que se han considerado necesarias para garantizar un mejor proceso de análisis y predicción. Hemos tratado los valores nulos o perdidos y hemos tratado también los valores extremos, aunque al final se ha optado por mantener los outliers en los análisis, ya que éstos no son atípicos, sino perfectamente plausibles, para el conjunto de valores que representan los pasajeros del buque.

Con todo ello, hemos procedido a efectuar el análisis de los datos y hemos hecho las pruebas correspondientes con cinco modelos diferentes de Machine Learning. Finalmente, tras comprobar la precisión de los modelos, nos hemos decidido por usar el método Random Forest y trabajar con él de forma más detallada.

Random Forest nos ha dado una precisión buena pero a mi entender, ofrece posibilidad de mejora para poder conseguir una mayor precisión en el resultado. No hemos de entender el resultado del 100% como definitivo porque hubiéramos podido variarlo

eliminando las 2 o 3 variables menos relevantes (SibSp, Parch y Embarked) que hemos decidido no eliminar.

En el futuro, creo que sería bueno probar una combinación de clasificadores para realizar el mismo ejercicio y ver si la precisión se puede mejorar.

7.- Código

A continuación adjuntamos el código Python.

```
# -*- coding: utf-8 -*-
"""
Editor de Spyder

Este es el archivo PRA2_JdM_code

Created on Fri Jan  3 18:50:58 2020

@author: Jordi de Mas
"""

# Importamos las librerias Python que usaremos
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats
from matplotlib import pyplot as plt
from matplotlib import style
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# Cargamos los datos que hemos obtenido de la web de Kaggle
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

# Veamos algunos registros del fichero train
df_train.head()

# Comprobamos los tipos de datos y la existencia de "missing values"
df_train.info()

# Eliminamos las columnas que, a priori, carecen de interés para el análisis.
df_cleaned = df_train.loc[:, ['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp',
'Parch', 'Embarked']]

# Hacemos lo mismo para el fichero de test.
df_test = df_test.loc[:, ['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked']]

# Vemos como queda el conjunto de datos después de eliminar las columnas
indicadas.
df_cleaned.head()

# Sustituimos los missing values del campo Embarked (sólo 2) por el mayoritariamente
presente en el fichero.
df_cleaned['Embarked'].describe()

# Sustituimos los missing values de Embarked con el valor "S" que es el mayoritario.
df_cleaned['Embarked'] = df_cleaned['Embarked'].fillna('S')
# Seguimos el mismo criterio en el fichero de test.
df_test['Embarked'] = df_test['Embarked'].fillna('S')
```

```
# Sustituimos los missing values de Age por la mediana de las edades.
age_median = df_cleaned['Age'].median()
df_cleaned['Age'] = df_cleaned['Age'].fillna(age_median)
# Hacemos lo mismo para el fichero de test
age_median_t = df_test['Age'].median()
df_test['Age'] = df_test['Age'].fillna(age_median_t)

# Cambiamos la variable Sex por valores numéricos
df_cleaned['Sex'].replace(['male', 'female'], [0, 1], inplace = True)
# Lo cambiamos también en el fichero de test
df_test['Sex'].replace(['male', 'female'], [0, 1], inplace = True)

# Cambiamos la variable Age por valores numéricos (int64)
age2 = df_cleaned['Age'].copy()
df_cleaned['Age'] = age2.astype(int)
# También lo cambiamos en el fichero de test
age3 = df_test['Age'].copy()
df_test['Age'] = age3.astype(int)

# Cambiamos también los datos del puerto de embarque por números
df_cleaned['Embarked'].replace(['C', 'Q', 'S'], [0, 1, 2], inplace = True)
# Hacemos lo mismo en el fichero de test
df_test['Embarked'].replace(['C', 'Q', 'S'], [0, 1, 2], inplace = True)

# Vemos como queda el fichero después de los cambios
df_cleaned.info()
df_cleaned.head()

# Identificación y tratamiento de outliers
f = plt.figure(figsize = (15, 5));
plt.subplot(1,4,1)
sns.boxplot(x = df_cleaned['Survived'])
plt.subplot(1,4,2)
sns.boxplot(x = df_cleaned['Pclass'])
plt.subplot(1,4,3)
sns.boxplot(x = df_cleaned['Sex'])
plt.subplot(1,4,4)
```

```
sns.boxplot(x = df_cleaned['Age'])

f = plt.figure(figsize = (15, 5));
plt.subplot(1,3,1)
sns.boxplot(x = df_cleaned['SibSp'])
plt.subplot(1,3,2)
sns.boxplot(x = df_cleaned['Parch'])
plt.subplot(1,3,3)
sns.boxplot(x = df_cleaned['Embarked'])

# Creamos intervalos de edades.
#bins = [0, 10, 18, 25, 40, 60, 100]
#names = ['1', '2', '3', '4', '5', '6']
#df_cleaned['Age'] = pd.cut(df_cleaned['Age'], bins, labels = names)
#df_test['Age'] = pd.cut(df_test['Age'], bins, labels = names)

# Comprobamos la nueva distribución de edades
#df_cleaned['Age'].value_counts()

# Analizamos la distribución de salvados por sexo.
df_cleaned.groupby(['Survived','Sex']).count().PassengerId
df_cleaned.groupby(['Survived','Sex']).count().PassengerId.plot(kind='bar')

# Distribución de salvados por clase.
df_cleaned.groupby(['Survived','Pclass']).count().PassengerId
df_cleaned.groupby(['Survived','Pclass']).count().PassengerId.plot(kind='bar')

# Test de normalidad de Anderson-Darling
print('Test Anderson-Darling - Pclass: ')
result_anderson = scipy.stats.anderson(df_cleaned['Pclass'], dist = 'norm')
for i in range(len(result_anderson.critical_values)):
    sl, cv = result_anderson.significance_level[i], result_anderson.critical_values[i]
    if result_anderson.statistic < result_anderson.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))
```

```
print('')
print('Test Anderson-Darling - Sex: ')
result_anderson = scipy.stats.anderson(df_cleaned['Sex'], dist = 'norm')
for i in range(len(result_anderson.critical_values)):
    sl, cv = result_anderson.significance_level[i], result_anderson.critical_values[i]
    if result_anderson.statistic < result_anderson.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))

print('')
print('Test Anderson-Darling - SibSp: ')
result_anderson = scipy.stats.anderson(df_cleaned['SibSp'], dist = 'norm')
for i in range(len(result_anderson.critical_values)):
    sl, cv = result_anderson.significance_level[i], result_anderson.critical_values[i]
    if result_anderson.statistic < result_anderson.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))

print('')
print('Test Anderson-Darling - Parch: ')
result_anderson = scipy.stats.anderson(df_cleaned['Parch'], dist = 'norm')
for i in range(len(result_anderson.critical_values)):
    sl, cv = result_anderson.significance_level[i], result_anderson.critical_values[i]
    if result_anderson.statistic < result_anderson.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))

print('')
print('Test Anderson-Darling - Embarked: ')
result_anderson = scipy.stats.anderson(df_cleaned['Embarked'], dist = 'norm')
for i in range(len(result_anderson.critical_values)):
    sl, cv = result_anderson.significance_level[i], result_anderson.critical_values[i]
    if result_anderson.statistic < result_anderson.critical_values[i]:
        print('%.3f: %.3f, data looks normal (fail to reject H0)' % (sl, cv))
```

```
    else:
        print('%.3f: %.3f, data does not look normal (reject H0)' % (sl, cv))

# Homogeneidad de varianzas mediante test de Fligner-Killeen
class1 = df_cleaned[df_cleaned['Pclass'] == '1']
class2 = df_cleaned[df_cleaned['Pclass'] == '2']
class3 = df_cleaned[df_cleaned['Pclass'] == '3']
stat1, pval1 = scipy.stats.fligner(class1, class2, class3, center = 'median')
stat2, pval2 = scipy.stats.fligner(class1, class2, class3, center = 'trimmed')
print(stat1, pval1)
print(stat2, pval2)

# Empezamos a construir modelos de Machine Learning
# Separamos la columna de Survived del fichero de entrenamiento.
X_train = df_cleaned.drop('Survived', axis = 1)
Y_train = df_cleaned['Survived']
X_test = df_test

# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_log

# KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)

Y_pred = knn.predict(X_test)

acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn

# Linear Support Vector Machine
```

```
linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)

Y_pred = linear_svc.predict(X_test)

acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
acc_linear_svc

# Decision Tree
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

Y_pred = decision_tree.predict(X_test)

acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
acc_decision_tree

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
acc_random_forest

# Análisis de Random Forest usando K-Fold Cross Validation
rf = RandomForestClassifier(n_estimators=100)
scores = cross_val_score(rf, X_train, Y_train, cv=10, scoring = "accuracy")
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())

# Análisis de Decision Tree usando K-Fold Cross Validation
dt = DecisionTreeClassifier()
scores = cross_val_score(dt, X_train, Y_train, cv=10, scoring = "accuracy")
```

```
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())

# Importancia de las variables en Random Forest.
importances =
pd.DataFrame({'feature':X_train.columns,'importance':np.round(random_forest.feature_
importances_,3)})
importances =
importances.sort_values('importance',ascending=False).set_index('feature')
importances.head(15)

# Graficamos los resultados
importances.plot.bar()

# Matriz de confusión
predictions = cross_val_predict(random_forest, X_train, Y_train, cv=10)
confusion_matrix(Y_train, predictions)

# Precision and Recall
print("Precision:", precision_score(Y_train, predictions))
print("Recall:",recall_score(Y_train, predictions))

# Calculamos ROC AUC curve (true positive rate and false positive rate). Se dibujan
uno contra otro.
# Calculamos las probabilidades de la predicción
y_scores = random_forest.predict_proba(X_train)
y_scores = y_scores[:,1]

false_positive_rate, true_positive_rate, thresholds = roc_curve(Y_train, y_scores)

def plot_roc_curve(false_positive_rate, true_positive_rate, label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2, label=label)
    plt.plot([0, 1.05], [0, 1.05], 'r', linewidth=4)
    plt.axis([0, 1.05, 0, 1.05])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)
```

```
plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()

# Calculamos el ROC AUC Score
r_a_score = roc_auc_score(Y_train, y_scores)
print("ROC-AUC-Score:", r_a_score)

# Convertir dataframe de entrenamiento tras los cambios a fichero csv.
df_cleaned.to_csv("train_final.csv", index = False)
print('Fichero guardado: train_final.csv')

# Convertir dataframe de test tras los cambios a fichero csv.
df_test.to_csv("test_final.csv", index = False)
print('Fichero guardado: test_final.csv')

# Convertir predicciones a fichero csv para envío a Kaggle.
def envio_kaggle_titanic(filename, predictions):
    envio_k = pd.DataFrame({'PassengerId':df_test['PassengerId'],
'Survived':predictions})
    envio_k.to_csv(filename, index = False)

# Llamo a la función con las predicciones realizadas cuando he evaluado el modelo
Random Forest.
envio_kaggle_titanic("titanic_predictions.csv", Y_prediction)
print('Fichero guardado: titanic_predictions.csv')
```

En el GitHub además del fichero .py con el código utilizado, también incluyo un notebook de Python (.ipynb) porque creo que resultará más claro de seguir.

Tabla de contribuciones

Al ser un trabajo individual todas las contribuciones tienen un único participante.

Contribuciones	Firma
Descripción del dataset	JdM
Integración y selección de los datos de interés a analizar	JdM
Limpieza de los datos	JdM
Análisis de los datos	JdM
Representación de los resultados	JdM
Resolución del problema	JdM
Código Python	JdM