

# Aprendizaje Automático

## Práctica 1

### 1) Análisis exploratorio de los datos

Importa los datos del fichero `dataset.csv` y realiza el análisis exploratorio de los datos. Describe en el informe los resultados de este análisis y deposita el código Python en Aula Virtual en el fichero `answer1.ipynb`

### 2) Clasificación usando todas las características de los datos

Usando todas las características, implementa los métodos

- Logistic Regression,
- SVM, y
- Random Trees

para clasificar los datos. Describe en el informe los parámetros usados y los resultados obtenidos con los distintos métodos y deposita el código Python en Aula Virtual en el fichero `answer2.ipynb`

### 3) Clasificación usando 4 características de los datos

Selecciona 4 características de los datos. Usando estas características, implementa los métodos

- Logistic Regression,
- SVM, y
- Random Trees

para clasificar los datos. Describe en el informe los parámetros usados y los resultados obtenidos con los distintos métodos y deposita el código Python en Aula Virtual en el fichero `answer3.ipynb`

### 4) Clasificación usando 2 características de los datos

Selecciona 2 características de los datos. Usando estas características, implementa los métodos

- Logistic Regression,
- SVM, y
- Random Trees

para clasificar los datos. Describe en el informe los parámetros usados y los resultados obtenidos con los distintos métodos y deposita el código Python en Aula Virtual en el fichero `answer4.ipynb`.

Redacta un informe detallado respondiendo a cada pregunta en una sección diferente. Motiva cada elección realizada durante el proceso de diseño de los clasificadores y describe cada resultado obtenido. Compara los resultados obtenidos con distintos números de características. Incluye el código Python en el apartado correspondiente del informe.

# 1 Análisis exploratorio de los datos

## 1.1 Importacion del dataset

El primer paso de este analisis consiste en la importacion del dataset, utilizando la libreria pandas para leer el fichero dataset.csv. A partir de ahí, se configura la visualizacion del dataset para poder observar todas las columnas disponibles.

Este paso es importante para obtener una vista inicial del tamaño y las características del dataset, además de confirmar que los datos han sido leídos correctamente.

```
dataset = pd.read_csv("dataset.csv")
pd.set_option('display.max_columns', len(dataset.columns))
```

## 1.2 Inspeccion del dataset

A continuacion, se evalua la estructura del dataset observando las dimensiones (dataset.shape), las primeras filas (dataset.head(1)), la informacion de cada columna y otras estadísticas descriptivas como la media, la desviacion estandar, minimos, maximos y cuartiles, además de otros valores atípicos y distribuciones inusuales (dataset.describe()).

Todo esto permite conocer cuantas filas y columnas contiene el dataset (dataset.columns), el tipo de datos que maneja (dataset.info()), la aparicion de valores nulos o la presencia de inconsistencia en algun dato.

```
dataset.columns
dataset.shape
dataset.head(1)
dataset.info()
dataset.describe()
```

## 1.3 Anonimizacion de los datos y analisis de correlacion

Para poder analizar los datos, se ha creado un nuevo conjunto de datos anonimizado, en el que se ha eliminado la columna Target del dataset. Además de esto, se realiza un analisis de la correlacion entre todas las columnas/variables.

Es importante realizar este paso para detectar altas correlaciones, las cuales suelen indicar multicolinealidad, algo que afecta a los modelos predictivos.

```
dataset_anonymized = dataset.drop(["Target"], axis=1)
dataset_anonymized.to_csv('dataset_anonymized.csv', index=False)
dataset_anonymized.corr()
```

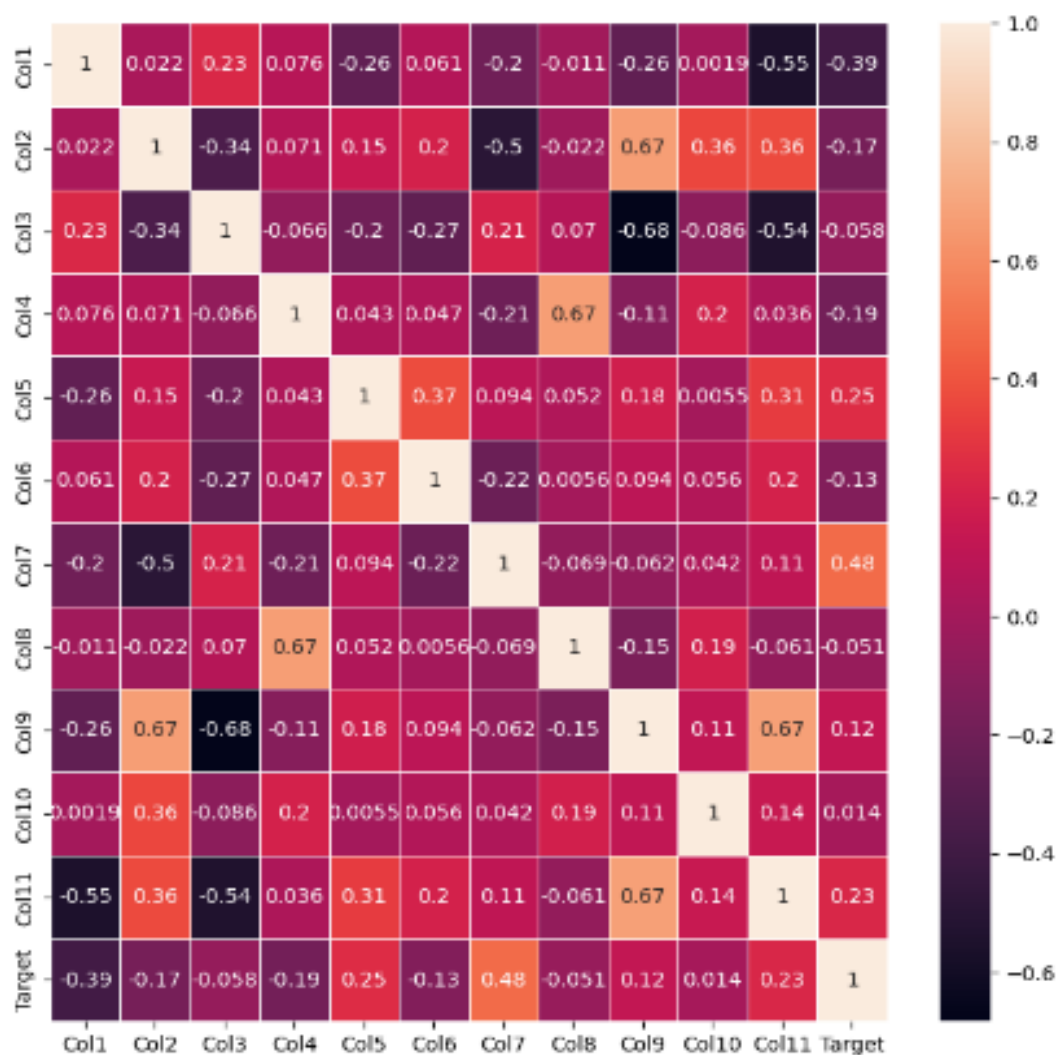
El analisis de una correlacion esta identificada con las demas, donde un coeficiente de correlacion cercano a 1 o -1 indica una correlacion fuerte, mientras que un valor cercano a 0 indica una correlacion practicamente inexistente.

## 1.4 Visualizacion de la matriz de correlacion

La matriz de correlacion se visualiza mediante un mapa de calor utilizando la libreria seaborn, la cual facilita la identificacion de relaciones positivas o negativas entre las variables.

```
fig, ax = plt.subplots(figsize=(9,9))
sb.heatmap(dataset.corr(), linewidth=0.5, annot=True)
```

El grafico resultante proporciona una vision mas clara de las relaciones mas fuertes entre las variables, donde los colores oscuros indican una correlacion alta, mientras que los colores mas claros indican una correlacion baja.

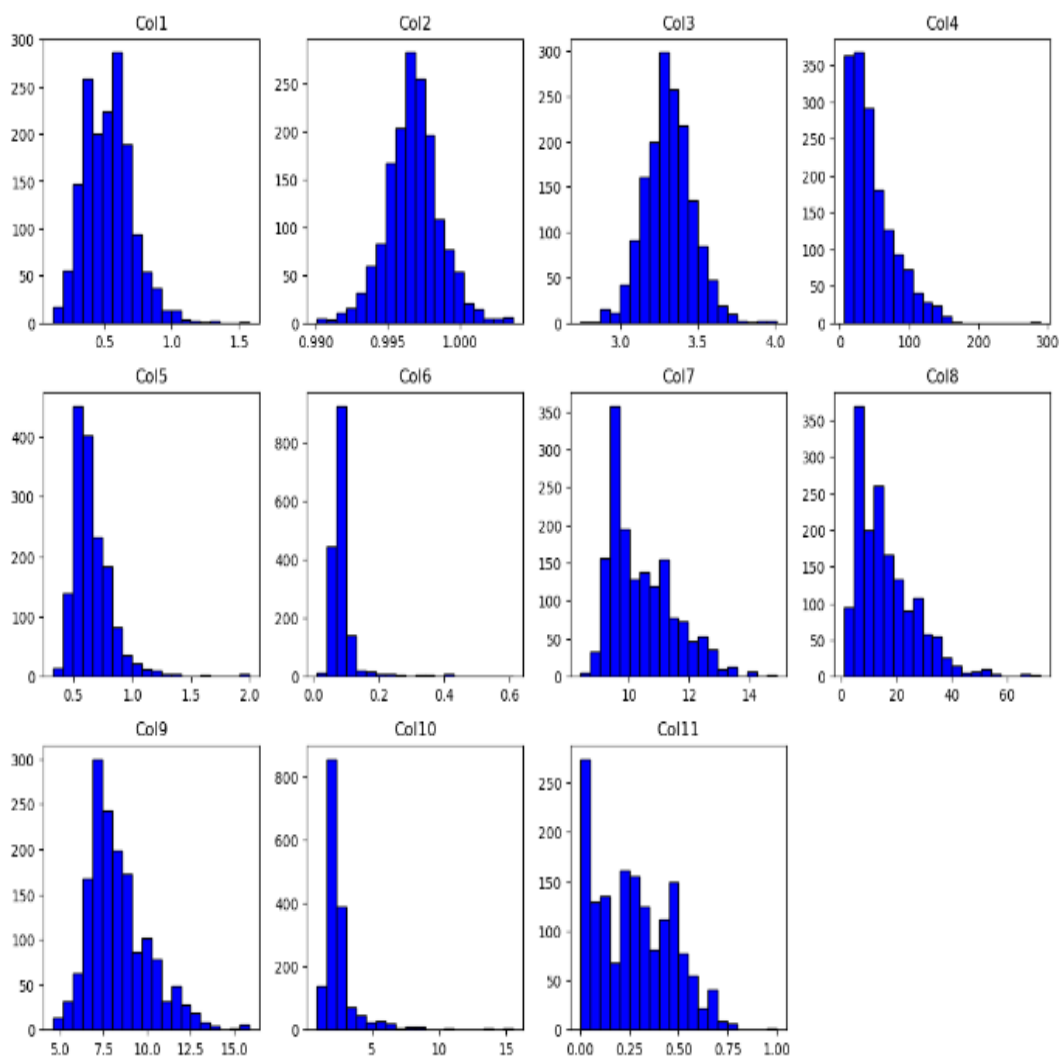


## 1.5 Visualizacion de distribuciones

Para observar la distribucion de cada una de las variables en el dataset anonimizado, se generan histogramas que permiten identificar patrones y verificar si las variables estan distribuidas de manera normal, poseen sesgos o son valores atipicos.

```
columns = dataset_anonymized.columns
fig = plt.figure(figsize=(12,12))
for i in range(0,11):
    ax = plt.subplot(4,4,i+1)
    ax.hist(dataset_anonymized[columns[i]], bins=20, color='blue', edgecolor='black')
    ax.set_title(dataset_anonymized.head(0)[columns[i]].name)
plt.tight_layout()
plt.show()
```

Estos histogramas muestran la frecuencia de los valores dentro de ciertos intervalos, algo que ayuda a entender mejor la distribucion de los datos.



## 1.6 Separacion de características y etiquetas

Una vez hecho esto, se separan las características (X) y las etiquetas (y) del dataset, lo cual permite preparar los datos para los modelos de machine learning.

En este caso, X contiene la informacion de todas las columnas, exceptuando la columna Target, mientras que y contiene los valores de la variable objetivo Target.

```
X = dataset_anonymized
y = dataset.get("Target")
print('Class labels:', np.unique(y))

# RESULTADO
Class labels: [3 4 5 6 7 8]
```

## 1.7 Division del dataset en entrenamiento y prueba

El dataset se divide en dos conjuntos: Uno de entrenamiento (0.75) y otro de prueba (0.25). Esta division se hace de tal forma que la proporcion de clases en ambos conjuntos sea similar, evitando asi posibles desbalances.

Es necesario realizar este paso para poder entrenar los modelos con una parte del dataset y luego probar su rendimiento con datos que no han sido vistos previamente.

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size=0.25,
                                                    random_state=1, stratify=y)
```

## 1.8 Estandarizacion de los datos

La estandarizacion de los datos asegura que todas las características se encuentren en la misma escala, algo que es considerado de gran importancia en modelos de regresion logistica, SVM o redes neuronales, los cuales se veran mas adelante.

Para este caso, se utiliza StandardScaler() para normalizar los datos, los cuales se ajustan a las características de los mismos para que tengan media 0 y desviacion estandar 1, lo que se traduce en una mejora de rendimiento a la hora de utilizar los algoritmos mencionados anteriormente.

```
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

## 1.9 Verificacion del balance de clases

Y por ultimo, se evalua el balance de las clases tanto en los datos de entrenamiento como en los datos de prueba para verificar si existe algun tipo de desbalance en las etiquetas, ya que de darse este caso, se podria producir un desbalance significativo que puede afectar a la capacidad predictiva del modelo.

```
print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))

# RESULTADO
Labels counts in y:      [ 0  0  0 10 53 681 638 199 18 ]
Labels counts in y_train: [ 0  0  0  8 40 511 478 149 13 ]
Labels counts in y_test:  [ 0  0  0  2 13 170 160  50  5 ]
```

Es importante mencionar que el balance de clases da una idea de si existe predominancia de una clase sobre otra, algo que podria requerir tecnicas de remuestreo para mitigar el sesgo.

## 2 Clasificación usando todas las características de los datos

### 2.1 Introduccion

En este apartado se presentan los resultados de la clasificacion del conjunto de datos del fichero dataset.csv utilizando 3 metodos de aprendizaje supervisado: Regresion Logistica (Logistic Regression), Maquinas de Soporte Vectorial (SVM) y Arboles de Decision (Random Trees).

Cada uno de estos clasificadores ha sido entrenado usando todas las características del dataset. A continuacion, se iran describiendo los parametros utilizados para cada modelo, presentando los resultados obtenidos y discutiendo el rendimiento de cada clasificador.

### 2.2 Carga y preparacion de los datos

Antes de entrenar los modelos, primero se debe cargar el dataset, la anonimizacion de las características y la separacion de las características (X) respecto a la etiqueta objetivo (y).

Ademas de esto, tambien se realiza una estandarizacion de las características, algo que es muy importante realizar para modelos como la Regresion Logistica (Logistic Regression) y SVM (Support Vector Machines), que son sensibles a las escalas de las variables.

```
dataset = pd.read_csv("dataset.csv")
dataset_all_characteristics = dataset.drop(["Target"], axis=1)
X = dataset_all_characteristics
y = dataset.get("Target")

X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size=0.25,
                                                    random_state=1, stratify=y)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Por otro lado, el dataset se divide en dos conjuntos: Uno de entrenamiento (0.75) y otro de prueba (0.25). Esta division se hace de tal forma que la proporcion de clases en ambos conjuntos sea similar, evitando asi posibles desbalanceos.

### 2.3 Regresion Logistica (Logistic Regression)

La Regresion Logistica es un modelo lineal, por lo que es mas adecuado cuando las características tienen relaciones lineales con las etiquetas. En este caso, el modelo muestra un buen rendimiento general, aunque la precision podria mejorar utilizando tecnicas de seleccion de características o ajustando mas los parametros.

A continuacion, se explican brevemente los parametros utilizados:

- `C = 100.0`: El parametro de regularizacion C controla la penalizacion aplicada a los errores. Un valor alto de C indica que se permite una menor penalizacion, por lo que el modelo intenta ajustar mas los datos.
- `solver = 'lbfgs'`: El solver 'lbfgs' es un optimizador recomendado para solucionar problemas pequeños y medianos.
- `multi-class = 'ovr'`: Procedente de las siglas OnevsRest (OVR), significa que para la clasificacion multiclase, el modelo entrenara un clasificador independiente para cada clase.

```
lr = LogisticRegression(C=100.0, solver='lbfgs', multi_class='ovr')
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % lr.score(X_test_std, y_test))

# RESULTADO
Misclassification samples: 165
Accuracy: 0.588
```

Se han obtenido como resultado 165 muestras mal clasificadas, un numero relativamente bajo para este modelo, y una precision de 0.588, por lo que esa proporcion de predicciones del modelo es correcta.

## 2.4 Maquinas de Soporte Vectorial (SVM)

SVM con kernel RBF es potente en la captura de relaciones no lineales entre las características. El modelo muestra un rendimiento sólido y, aunque los resultados dependen mucho de la selección de los parámetros gamma y C, el ajuste da buenos resultados.

A continuacion, se explican brevemente los parametros utilizados:

- kernel = 'rbf': Se utiliza el kernel radial base (RBF), que es adecuado para problemas no lineales.
- gamma = 0.7: Controla el grado de influencia de los puntos individuales. Un valor bajo significa que el área de influencia de cada punto es alta, mientras que un valor alto restringe el área.
- C = 30.0: Controla el grado de penalización aplicado a los errores de clasificación. Un valor más alto de C tiende a reducir los errores de clasificación en el conjunto de entrenamiento.

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.7, C=30.0)
svm.fit(X_train_std, y_train)
y_pred = svm.predict(X_test_std)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % svm.score(X_test_std, y_test))

# RESULTADO
Misclassification samples: 157
Accuracy: 0.608
```

Se han obtenido como resultado 157 muestras mal clasificadas, un numero relativamente bajo para este modelo, y una precision de 0.608, una proporcion ligeramente superior a la obtenida con la Regresion Logistica.

## 2.5 Arboles de Decision (Random Trees)

Los árboles de decisión tienden a sobreajustar los datos si no se limitan adecuadamente. En este caso, al limitar la profundidad a 4, se puede evitar el sobreajuste y el modelo puede ser generalizado adecuadamente. Sin embargo, en comparación con los otros modelos, el rendimiento es ligeramente inferior, posiblemente porque el modelo no pudo capturar todas las complejidades de los datos con solo 4 niveles de profundidad.

A continuacion, se explican brevemente los parametros utilizados:

- criterion = 'gini': Utiliza el índice de Gini para medir la pureza de los nodos.
- max-depth = 4: La profundidad máxima del árbol se fija en 4 para evitar el sobreajuste.

```
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
tree_model.fit(X_train, y_train)
y_pred = tree_model.predict(X_test)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % tree_model.score(X_test, y_test))

# Visualización del árbol de decisión
tree.plot_tree(tree_model, feature_names=['Col1', 'Col2', 'Col3', 'Col4', 'Col5', 'Col6', 'Col7',
'Col8', 'Col9', 'Col10', 'Col11'], filled=True)
plt.show()

# RESULTADO
Misclassification samples: 172
Accuracy: 0.570
```

Se han obtenido como resultado 172 muestras mal clasificadas, un numero similar a los otros dos modelos, y una precision de 0.570, por lo que esa proporcion de predicciones del modelo es correcta.

## 3 Clasificación usando 4 características de los datos

### 3.1 Introduccion

En este apartado se presentan los resultados de la clasificación de 4 de las columnas del fichero dataset.csv utilizando 3 métodos de aprendizaje supervisado: Regresión Logística (Logistic Regression), Máquinas de Soporte Vectorial (SVM) y Árboles de Decisión (Random Trees).

El objetivo de esto es comparar el rendimiento de estos modelos al reducir el número de variables y analizar el impacto de esta reducción en la precisión de los clasificadores.

### 3.2 Selección de las 4 características

En primer lugar, se realiza un análisis de correlación entre las variables del dataset completo para identificar las características más relevantes. Para ello, se han seleccionado las características Col1, Col5, Col7 y Col11.

Las variables seleccionadas son aquellas que mejor coeficiente de correlación poseen, además de poder aportar mucha relevancia para su clasificación, a fin de evitar multicolinealidad y maximizar la capacidad predictiva del modelo.

```
dataset_4_characteristics = dataset_anonymized.drop(["Col2", "Col3", "Col4", "Col6",  
"Col8", "Col9", "Col10"], axis=1)  
dataset_4_characteristics.to_csv('dataset_4_characteristics.csv', index=False)
```

### 3.3 Preparación de los datos

Una vez hecho esto, el conjunto de datos se ha dividido en 2 subconjuntos: Entrenamiento (0.75) y prueba (0.25), lo que asegura que la distribución de las clases sea equilibrada.

Además, los datos han sido estandarizados mediante StandardScaler(), algo importante en clasificadores como la Regresión Logística (Logistic Regression) y SVM (Support Vector Machines), que son sensibles a la escala de los datos.

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size=0.25,  
random_state=1, stratify=y)  
sc = StandardScaler()  
sc.fit(X_train)  
X_train_std = sc.transform(X_train)  
X_test_std = sc.transform(X_test)
```

### 3.4 Regresión Logística (Logistic Regression)

La Regresión Logística muestra un rendimiento satisfactorio utilizando solo 4 características, lo que sugiere que las variables seleccionadas tienen una relación significativa con la etiqueta objetivo. A pesar, también se puede suponer que la precisión obtenida se ha visto limitada por la naturaleza lineal del modelo.

A continuación, se explican brevemente los parámetros utilizados:

- $C = 100.0$ : El parámetro de regularización  $C$  controla la penalización aplicada a los errores. Un valor alto de  $C$  indica que se permite una menor penalización, por lo que el modelo intenta ajustar más los datos.
- `solver = 'lbfgs'`: El solver 'lbfgs' es un optimizador recomendado para solucionar problemas pequeños y medianos.
- `multi-class = 'ovr'`: Procedente de las siglas One-vs-Rest (OVR), significa que para la clasificación multiclase, el modelo entrenará un clasificador independiente para cada clase.

```
lr = LogisticRegression(C=100.0, solver='lbfgs', multi_class='ovr')  
lr.fit(X_train_std, y_train)  
y_pred = lr.predict(X_test_std)  
print('Misclassification samples: %d' % (y_test != y_pred).sum())  
print('Accuracy: %.3f' % lr.score(X_test_std, y_test))  
  
# RESULTADO  
Misclassification samples: 164  
Accuracy: 0.590
```

Se han obtenido como resultado 164 muestras mal clasificadas y una precisión de 0.590.



### 3.5 Maquinas de Soporte Vectorial (SVM)

El uso de un kernel RBF permite capturar relaciones no lineales entre las características, por lo que el modelo SVM sigue siendo altamente efectivo incluso teniendo unicamente 4 características.

Sin embargo, al reducir el numero de características, el modelo podria quedar limitado a la hora de identificar patrones mas complejos.

A continuacion, se explican brevemente los parametros utilizados:

- kernel = 'rbf': Se utiliza el kernel radial base (RBF), que es adecuado para problemas no lineales.
- gamma = 0.7: Controla el grado de influencia de los puntos individuales. Un valor bajo significa que el área de influencia de cada punto es alta, mientras que un valor alto restringe el área.
- C = 30.0: Controla el grado de penalización aplicado a los errores de clasificación. Un valor más alto de C tiende a reducir los errores de clasificación en el conjunto de entrenamiento.

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.7, C=30.0)
svm.fit(X_train_std, y_train)
y_pred = svm.predict(X_test_std)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % svm.score(X_test_std, y_test))

# RESULTADO
Misclassification samples: 174
Accuracy. 0.565
```

Se han obtenido como resultado 174 muestras mal clasificadas y una precision de 0.565.

### 3.6 Arboles de Decision (Random Trees)

El arbol de decision tiene un rendimiento adecuado, aunque presenta una ligera tendencia al sobreajuste al tener solo 4 características, algo que se ha solucionado limitando la profundidad, quedando a expensas de la precision en algunas ocasiones.

A continuacion, se explican brevemente los parametros utilizados:

- criterion = 'gini': Utiliza el índice de Gini para medir la pureza de los nodos.
- max-depth = 4: La profundidad máxima del árbol se fija en 4 para evitar el sobreajuste.

```
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
tree_model.fit(X_train, y_train)
tree.plot_tree(tree_model, feature_names=['Col1', 'Col5', 'Col7', 'Col11'], filled=True)
plt.show()
y_pred = tree_model.predict(X_test)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % tree_model.score(X_test, y_test))

# RESULTADO
Misclassification samples: 174
Accuracy. 0.565
```

Se han obtenido como resultado 174 muestras mal clasificadas y una precision de 0.565.

## 4 Clasificación usando 2 características de los datos

### 4.1 Introduccion

En este apartado se presentan los resultados de la clasificacion de 2 de las columnas del fichero dataset.csv utilizando 3 metodos de aprendizaje supervisado: Regresion Logistica (Logistic Regression), Maquinas de Soporte Vectorial (SVM) y Arboles de Decision (Random Trees).

El objetivo de esto es comparar el rendimiento de estos modelos al reducir el numero de variables y analizar el impacto de esta reduccion en la precision de los clasificadores.

### 4.2 Seleccion de las 2 características

En primer lugar, se realiza un análisis de correlación entre las variables del dataset completo para identificar las características más relevantes. Para ello, se han seleccionado las características Col1 y Col7.

Las variables seleccionadas son aquellas que mejor coeficiente de correlacion poseen, ademas de poder aportar mucha relevancia para su clasificación, a fin de evitar multicolinealidad y maximizar la capacidad predictiva del modelo.

```
dataset_2_characteristics = dataset_anonymized.drop(["Col2", "Col3", "Col4", "Col5",  
"Col6", "Col8", "Col9", "Col10", "Col11"], axis=1)  
dataset_2_characteristics.to_csv('dataset_2_characteristics.csv', index=False)
```

### 4.3 Preparacion de los datos

Una vez hecho esto, el conjunto de datos se ha dividido en 2 subconjuntos: Entrenamiento (0.75) y prueba (0.25), lo que asegura que la distribucion de las clases sea equilibrada.

Ademas, los datos han sido estandarizados mediante StandardScaler(), algo importante en clasificadores como la Regresion Logistica (Logistic Regression) y SVM (Support Vector Machines), que son sensibles a la escala de los datos.

```
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size=0.25,  
random_state=1, stratify=y)  
sc = StandardScaler()  
sc.fit(X_train)  
X_train_std = sc.transform(X_train)  
X_test_std = sc.transform(X_test)
```

#### 4.4 Regresión Logística (Logistic Regression)

La Regresión Logística muestra un rendimiento aceptable a pesar de la reducción a solo 2 características, lo que indica que las variables seleccionadas aportan información relevante para la clasificación.

Sin embargo, la reducción a solo 2 variables limita la capacidad del modelo para capturar relaciones más complejas.

A continuación, se explican brevemente los parámetros utilizados:

- $C = 100.0$ : El parámetro de regularización  $C$  controla la penalización aplicada a los errores. Un valor alto de  $C$  indica que se permite una menor penalización, por lo que el modelo intenta ajustar más los datos.
- `solver = 'lbfgs'`: El solver 'lbfgs' es un optimizador recomendado para solucionar problemas pequeños y medianos.
- `multi-class = 'ovr'`: Procedente de las siglas OnevsRest (OVR), significa que para la clasificación multiclase, el modelo entrenará un clasificador independiente para cada clase.

```
lr = LogisticRegression(C=100.0, solver='lbfgs', multi_class='ovr')
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test_std)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % lr.score(X_test_std, y_test))

# RESULTADO
Misclassification samples: 167
Accuracy: 0.583
```

Se han obtenido como resultado 167 muestras mal clasificadas y una precisión de 0.583.

#### 4.5 Máquinas de Soporte Vectorial (SVM)

El SVM con kernel RBF demuestra un rendimiento superior al capturar relaciones no lineales entre las dos características seleccionadas.

Sin embargo, en este caso, el clasificador se ha beneficiado de uso de un kernel no lineal, lo que sugiere que aunque el número de características sea bajo, el modelo puede seguir identificando patrones complejos en los datos.

A continuación, se explican brevemente los parámetros utilizados:

- `kernel = 'rbf'`: Se utiliza el kernel radial base (RBF), que es adecuado para problemas no lineales.
- `gamma = 0.7`: Controla el grado de influencia de los puntos individuales. Un valor bajo significa que el área de influencia de cada punto es alta, mientras que un valor alto restringe el área.
- $C = 30.0$ : Controla el grado de penalización aplicado a los errores de clasificación. Un valor más alto de  $C$  tiende a reducir los errores de clasificación en el conjunto de entrenamiento.

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.7, C=30.0)
svm.fit(X_train_std, y_train)
y_pred = svm.predict(X_test_std)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % svm.score(X_test_std, y_test))

# RESULTADO
Misclassification samples: 167
Accuracy: 0.583
```

Se han obtenido como resultado 167 muestras mal clasificadas y una precisión de 0.583.

#### 4.6 Árboles de Decision (Random Trees)

El Arbol de Decision logra un rendimiento decente, aunque la falta de complejidad en el modelo y al estar trabajando con solo 2 características, es algo que podría haber limitado su capacidad predictiva.

Por otro lado, el uso de una profundidad limitada evita que el modelo se sobreajuste a los datos de entrenamiento, algo que se considera muy importante cuando se está trabajando con un número tan reducido de características.

A continuación, se explican brevemente los parámetros utilizados:

- criterion = 'gini': Utiliza el índice de Gini para medir la pureza de los nodos.
- max-depth = 4: La profundidad máxima del árbol se fija en 4 para evitar el sobreajuste.

```
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=1)
tree_model.fit(X_train, y_train)
tree.plot_tree(tree_model, feature_names=['Col1', 'Col7'], filled=True)
plt.show()
y_pred = tree_model.predict(X_test)
print('Misclassification samples: %d' % (y_test != y_pred).sum())
print('Accuracy: %.3f' % tree_model.score(X_test, y_test))

# RESULTADO
Misclassification samples: 178
Accuracy: 0.555
```

Se han obtenido como resultado 178 muestras mal clasificadas y una precisión de 0.555.

## 5 Comparación de los resultados y conclusiones

### 5.1 Comparacion de los resultados

Características	Regresion Logistica	SVM	Arboles de Decision
Todas	0.588	0.608	0.570
4	0.590	0.565	0.565
2	0.583	0.583	0.555

El SVM con kernel RBF ha sido el modelo más consistente en cuanto a rendimiento en todos los casos.

Por otro lado, la Regresión Logística ha ido mostrando un rendimiento sólido pero decreciente a medida que se han ido reduciendo las características.

Y por ultimo, los Árboles de Decisión han sido los más afectados por la reducción de información, lo que sugiere que su capacidad para construir reglas de decisión eficaces depende más de la cantidad de características disponibles.

### 5.2 Conclusiones

El rendimiento de los clasificadores ha ido dependiendo significativamente del número de características utilizadas. A continuacion se muestran algunos de los aspectos mas importantes a tener en cuenta en cada modelo:

- SVM (Support Vector Machines): ha sido el más robusto frente a la reducción de características, adaptándose bien incluso en el caso en el que se han tenido solo 2 características.
- Regresion Logistica (Logistic Regression): Ha ido mostrando un rendimiento decreciente cuando se han ido reduciendo la cantidad de características, destacando su dependencia en información más completa para realizar predicciones precisas.
- Árboles de Decisión (Random Trees): Han necesitado más características para crear reglas complejas y precisas. Sin embargo, con menos variables, su capacidad para capturar patrones importantes se ha visto disminuida.

En general, reducir el número de características impacta el rendimiento de los modelos, pero algunos, como el SVM (Support Vector Machines), pueden manejar esta reducción mejor que otros.