BLOQUE 2: ÁRBOLES Y SVM

Realizado por Nikos Loizou Fernández

Decision Tree

Ejercicio 3

```
from sklearn . tree import DecisionTreeClassifier
arbol = DecisionTreeClassifier (criterion ='gini', max_depth =5, min_samples_split=10, min_samples_leaf=10)
```

Usando esta configuración, el resultado del classification_report fue el siguiente:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.93      0.77      0.84       102
           1       0.96      0.99      0.97       546

    accuracy                           0.96       648
   macro avg       0.94      0.88      0.91       648
weighted avg       0.95      0.96      0.95       648
```

Al cambiar los hiper parámetros, usando max_depth = 8, min_samples_split = 7, min_samples_leaf = 7, obtenmos el siguiente classification_report:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.84      0.77      0.81       102
           1       0.96      0.97      0.97       546

    accuracy                           0.94       648
   macro avg       0.90      0.87      0.89       648
weighted avg       0.94      0.94      0.94       648
```

Al compararlas podemos comprobar que hacer el árbol más profundo, con más capacidad de aprendizaje ha reducido la precisión general en la clase 0 (clase minoritaria), mientras que en la clase 1 se mantiene más. Esto solo demuestra que hacer el árbol más profundo no siempre será la mejor opción, pudiendo llegar a crear desajuste y descuidar la clase minoritaria.

Al utilizar los mismos parámetros que al comenzar, pero usando entropy, obtenemos los siguientes resultados:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.90      0.77      0.83       102
           1       0.96      0.98      0.97       546

    accuracy                           0.95       648
   macro avg       0.93      0.88      0.90       648
weighted avg       0.95      0.95      0.95       648
```

Los resultados son bastante semejantes, esto debido a que las clases están bien separadas y balanceadas, por tanto, las métricas tienden a seleccionar los mismos splits en ambos casos.

Ejercicio 4

Usando los hiper parámetros dados en el ejercicio, obtenemos el siguiente classification report:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.90      0.76      0.83       102
           1       0.96      0.98      0.97       546

    accuracy                           0.95       648
   macro avg       0.93      0.87      0.90       648
weighted avg       0.95      0.95      0.95       648
```

Comparado con los resultados anteriores, podemos observar que la salida es muy parecida. Esto se debe a que cada árbol individual que crea Bagging, usa Gini o Entropy para decidir los splits. Aunque usa muchos árboles, al tener la distribución de clases y características muy clara, los datos serán similares.

Ejercicio 5

Usando los hiper parámetros dados en el ejercicio, obtenemos el siguiente classification report:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.93      0.76      0.84       102
           1       0.96      0.99      0.97       546

    accuracy                           0.95       648
   macro avg       0.94      0.88      0.91       648
weighted avg       0.95      0.95      0.95       648
```

En este podemos ver una muy pequeña mejora frente a los anteriores, esto se debe a la aleatorización de Random Forest, que reduce la correlación. Cómo los datos están ya bastante bien separados y balanceados, un solo árbol ya tiene un rendimiento cercano al máximo posible. Si hubiese más ruido en los datos, la diferencia sería más significativa.

Ejercicio 6

Usando los hiper parámetros dados en el ejercicio, obtenemos el siguiente classification report:

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.90	0.77	0.83	102
1	0.96	0.98	0.97	546
accuracy			0.95	648
macro avg	0.93	0.88	0.90	648
weighted avg	0.95	0.95	0.95	648

Como podemos observar los resultados son muy parecidos a los anteriores, el motivo sigue siendo la buena separación y el balance de los datos, de esta forma hay pocos errores que corregir en cada árbol. Además, el sesgo es muy bajo que casi no se reduce. Sería más notable la diferencia habiendo mucho más ruido en los datos.

SVM

Antes de empezar a usar SVM es necesario que escalemos los datos, esto para que las variables más grandes no dominen la distancia y el modelo no ignore las variables más pequeñas.

Ejercicio 3

Tras entrenar el modelo con clasificador SVM lineal, y hacer las predicciones, imprimimos el classification report para poder evaluar el rendimiento de este:

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.77	0.77	0.77	107
1	0.81	0.81	0.81	133
accuracy			0.79	240
macro avg	0.79	0.79	0.79	240
weighted avg	0.79	0.79	0.79	240

Podemos decir que el rendimiento de este clasificador es sólido y bueno, con aproximadamente el 80% de predicciones correctas. Teniendo un resultado parecido en el Recall y f1 score, por tanto, podríamos argumentar que no existe un sesgo fuerte hacia una clase.

Ejercicio 4

Probando los diferentes Kernels, y ejecutando sus classification reports, obtenemos las siguientes tablas:

Reporte de Clasificación RBF:

	precision	recall	f1-score	support
0	0.81	0.75	0.78	107
1	0.81	0.86	0.83	133
accuracy			0.81	240
macro avg	0.81	0.80	0.80	240
weighted avg	0.81	0.81	0.81	240

Reporte de Clasificación Polynomial:

	precision	recall	f1-score	support
0	0.71	0.67	0.69	107
1	0.75	0.77	0.76	133
accuracy			0.73	240
macro avg	0.73	0.72	0.72	240
weighted avg	0.73	0.73	0.73	240

Reporte de Clasificación Sigmoid:

	precision	recall	f1-score	support
0	0.75	0.78	0.76	107
1	0.81	0.79	0.80	133
accuracy			0.78	240
macro avg	0.78	0.78	0.78	240
weighted avg	0.78	0.78	0.78	240

Comparando y evaluando el rendimiento de estos 3 kernels, deducimos que:

- RBF es el que ha obtenido mejores resultados, aunque la diferencia no es demasiado notable. Esto nos indica que hay ciertas interacciones no lineales que son captadas por RBF, mejorando la detección de defectos, de ahí su mejora en accuracy.
- Comprobando los datos del csv podemos ver que no hay complejidad extrema en ellos, de ahí que Polynomial no mejore. De hecho, no funciona tan bien debido a que transforma los datos a un espacio muy grande, y ya que hay 17 variables, pero pocas muestras, el modelo no generaliza bien, pudiendo haber subajuste en el test.
- Sigmoid da unos resultados muy parecidos a Linear ya que su frontera de decisión es limitada y no tan flexible, RBF consigue un mejor rendimiento debido a que si que puede crear curvas más complejas que separen mejor los datos que tienen interacciones no lineales.

En resumen, comprobamos un mejor un funcionamiento en RBF debido a su capacidad de crear curvas más complejas, aún así no es muy grade debido a que hay pocas interacciones no lineales que son captadas por RBF.

Ejercicio 6

En el ejercicio 5 hemos aplicado PCA a los datos de entrenamiento para reducir la dimensionalidad. Luego, hemos entrenado a un modelo RBF ya que fue el que mejor desempeño tuvo anteriormente.

Hemos probado con varias dimensiones, en este caso resaltamos estas 3:

- 5 dimensiones:

	precision	recall	f1-score	support
0	0.74	0.65	0.69	107
1	0.74	0.81	0.78	133
accuracy			0.74	240
macro avg	0.74	0.73	0.74	240
weighted avg	0.74	0.74	0.74	240

- 10 dimensiones:

	precision	recall	f1-score	support
0	0.73	0.66	0.70	107
1	0.75	0.80	0.78	133
accuracy			0.74	240
macro avg	0.74	0.73	0.74	240
weighted avg	0.74	0.74	0.74	240

- 16 dimensiones:

	precision	recall	f1-score	support
0	0.81	0.75	0.78	107
1	0.81	0.86	0.83	133
accuracy			0.81	240
macro avg	0.81	0.80	0.80	240
weighted avg	0.81	0.81	0.81	240

Evaluando las 3 y conociendo el funcionamiento del clasificador SVM RBF, podemos concluir que:

El modelo con 5 dimensiones pierde información importante de las 17 variables originales, debido a eso el accuracy disminuye hasta un 74 %, teniendo f1 clase 0 = 0.69 y de clase 1 = 0.78, esto debido a que la reducción elimina relaciones no lineales críticas. Con 10 dimensiones se captura algo más de información, aunque no existe una mejora, los resultados son los mismos. Mientras que, con 16 dimensiones, prácticamente toda la información original, se logra una gran mejora, logrando unos resultados parecidos a los obtenidos sin PCA. Por tanto, el kernel RBF funciona mejor cuanta más información tenga, ya que podrá modelar curvas lineales más complejas.