Instituto Tecnológico y de Estudios Superiores de Monterrey.

Campus Estado de México

Momento de Retroalimentación: Módulo 2 Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución.

ID3 Con Framework

Profesor

Jorge Adolfo Ramírez Uresti

Integrantes

Sebastián Espinoza Farías……. ITC | A01750311

Fecha de entrega: 10-09-2015

Para la segunda entrega del portafolio de implementación se desarrolló nuevamente el algoritmo ID3, sobre árboles de decisión. Pero con la diferencia, en que esta vez se desarrolló utilizando frameworks como SKLearn para el desarrollo del modelo y otros como matplotlib para la visualización de la matriz de confusión.

El algoritmo esta vez fue probado con cuatro datasets distintos:

Tennis: Es el mismo dataset que el visto en clase y el que fue utilizado para la versión del algoritmo sin framework. Este no fue el dataset final de esta entrega debido a que los resultados obtenidos eran extraños. A diferencia del algoritmo hecho a mano que presenta overfitting. Este en el momento de entrar a los tests, presentaba resultados con overfitting o muy alejados a algo deseable. Por lo que se tiene la hipótesis de que existe algo dentro del algoritmo de la librería que impide que se comporte de manera exitosa en ese dataset tan pequeño.

Mushrooms: Es el mismo dataset para identificar si un hongo puede o no ser consumido. A diferencia del algoritmo hecho a mano que parecía presentar overfitting, este muestra resultados de 99% de accuracy, que a pesar de ser muy elevado el porcentaje, me parece más lógico que la implementación a mano que alcanzaba el 100%. Sin embargo, al pensar que se podría tratar de overfitting, se probó con dos datasets más.

Cars: Este fue el mismo dataset con el que se aprobó definitivamente el algoritmo anterior, el cual ya no presentó síntomas de overfitting. Sin embargo, al utilizar el framework, se obtuvieron resultados muy elevados aún. Por lo que a pesar de que fuera un buen dataset para la implementación anterior, se buscó probar con uno diferente esta vez.

Heart: El dataset final, este dataset cuenta con filas mucho más variadas, lo que lo convierte en un problema un poco más complejo de resolver. Al probar el algoritmo, ya no se obtienen resultados de excelencia como los 99%, sin embargo, se obtienen resultados de 85-87%, esto me parece un resultado coherente y con el cual quedé satisfecho al saber que no realiza overfitting y es un modelo fuerte con otros datasets.

El algoritmo realiza de manera automática la división del dataset en entrenamiento y prueba, reservando un 20% de los datos para evaluación y un 80% para entrenamiento.

Una vez hecha la división, el programa genera dos archivos CSV:

- <dataset>_train.csv: contiene los datos utilizados para entrenar el árbol.
- <dataset>_test.csv: contiene los datos reservados para probar el modelo.

Esta funcionalidad permite visualizar claramente qué datos fueron usados como train y test.

Resultados

Cuando el algoritmo finaliza su ejecución, se imprime en la terminal información de utilidad para saber qué fue lo que el algoritmo realizó:

```
Saved splits:
  Train -> heart_train.csv  (rows: 734)
  Test  -> heart_test.csv   (rows: 184)

Dataset: heart.csv
Train class counts:
 HeartDisease
1    406
0    328
Name: count, dtype: int64
Test class counts:
 HeartDisease
1    102
0     82
Name: count, dtype: int64
```

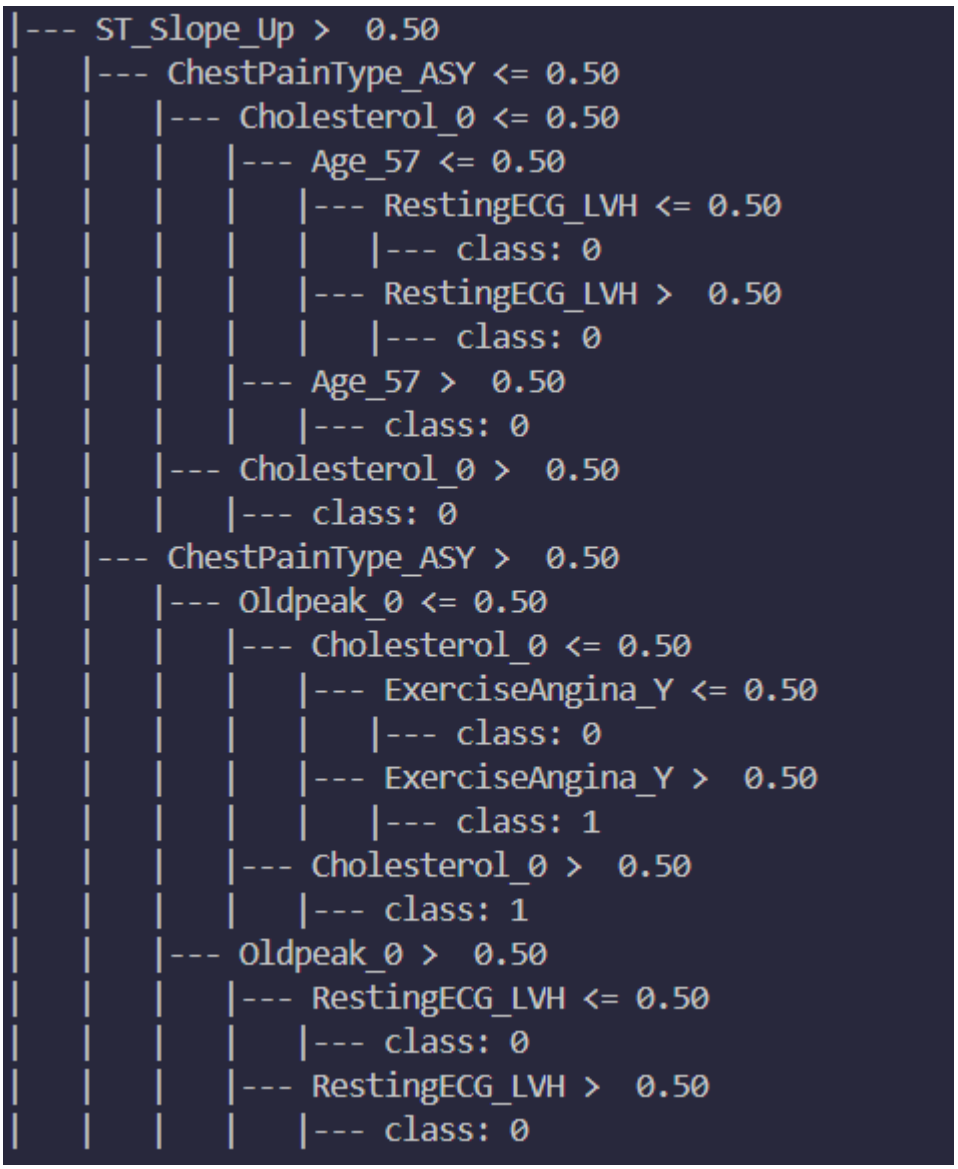
La primera parte del resultado, al igual que el modelo anterior, muestra cual fue el dataset utilizado, el tamaño de las divisiones realizadas entre el dataset de train y de test. Así como las cuentas de clases en cada uno de los datasets. Lo cual es útil para saber que se está trabajando con buenos datos y no existe un desbalance entre las cantidades y features.

```
GridSearchCV used StratifiedKFold(n_splits=5)
Best params from GridSearchCV: {'model__ccp_alpha': 0.005, 'model__max_depth': 5, 'model__min_samples_leaf': 5}
Best CV score: 0.853
```

La siguiente parte del resultado muestra cómo se aplicó GridSearchCV sobre un pipeline compuesto por el One-Hot Encoding y un Decision Tree Classifier con criterio de entropía. El grid search explora de forma sistemática combinaciones de hiperparámetros y las evalúa mediante validación cruzada estratificada de 5 pliegues, preservando la proporción de clases

en cada división. La mejor combinación se selecciona por el mayor promedio de exactitud entre pliegues (Best CV score) y luego el pipeline se reentrena con todo el conjunto de entrenamiento usando esos valores óptimos. Este proceso busca reducir el posible overfitting y mejorar la capacidad de generalización del modelo antes de su evaluación final. En este caso podemos ver que el mejor score que se obtuvo para el dataset de heart fue de 0.853, el cual es bastante bueno.

```
Pretty tree (train):
|--- ST_Slope_Up <= 0.50
|   |--- ChestPainType_ASY <= 0.50
|   |   |--- Sex_F <= 0.50
|   |   |   |--- Oldpeak_0 <= 0.50
|   |   |   |   |--- RestingBP_130 <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- RestingBP_130 >  0.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- Oldpeak_0 >  0.50
|   |   |   |   |--- RestingBP_120 <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- RestingBP_120 >  0.50
|   |   |   |   |   |--- class: 1
|   |   |--- Sex_F >  0.50
|   |   |   |--- class: 0
|   |--- ChestPainType_ASY >  0.50
|   |   |--- Cholesterol_0 <= 0.50
|   |   |   |--- Sex_M <= 0.50
|   |   |   |   |--- RestingBP_130 <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- RestingBP_130 >  0.50
|   |   |   |   |   |--- class: 0
|   |   |   |--- Sex_M >  0.50
|   |   |   |   |--- Oldpeak_0.5 <= 0.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Oldpeak_0.5 >  0.50
|   |   |   |   |   |--- class: 1
|   |   |--- Cholesterol_0 >  0.50
|   |   |   |--- class: 1
```

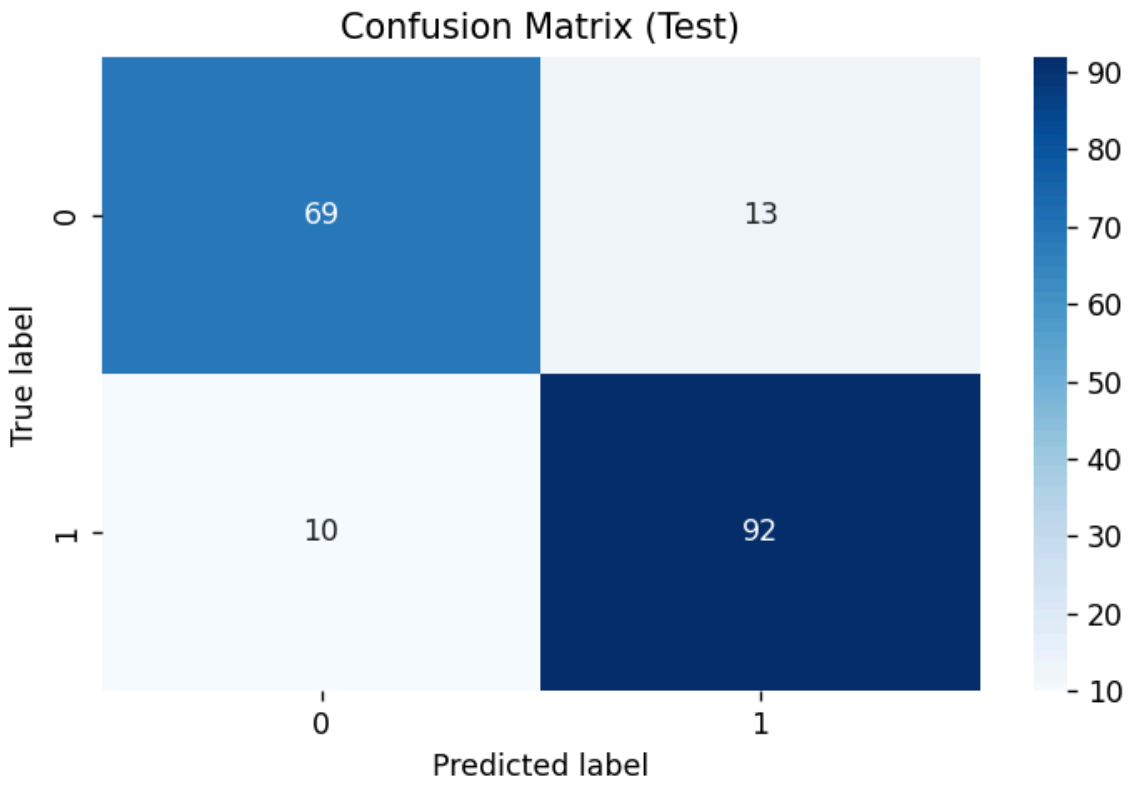


La siguiente parte del resultado muestra el árbol entrenado que genera export_text de scikit-learn. Como las variables se codificaron con One-Hot Encoding, cada columna es binaria (0/1), y por eso los cortes aparecen como <= 0.50 y > 0.50: el primero indica que la categoría no está presente (0) y el segundo que sí está presente (1). Cada nivel de sangría corresponde a una profundidad del árbol y cada camino desde la raíz hasta un leaf, define una regla de decisión.

```
Accuracy
  Train: 0.8773841961852861
  Test : 0.875

Confusion Matrix (Test)
        pred_0  pred_1
true_0      69      13
true_1      10      92
```

Posteriormente se muestra el accuracy de train y test, con lo que podemos evaluar rápidamente el desempeño del modelo. Después vemos la matriz de confusión, que nos ayuda a conocer que fue realmente lo que sucedió. Al poder realizar la matriz con uso de frameworks, también se graficó:



```
Per-class Precision, Recall, F1 (Test)
              precision    recall  f1-score   support

           0      0.873     0.841     0.857        82
           1      0.876     0.902     0.889       102

    accuracy                          0.875       184
   macro avg      0.875     0.872     0.873       184
weighted avg      0.875     0.875     0.875       184
```

Para finalizar, se imprime el classification_report de scikit-learn, equivalente a las métricas calculadas manualmente en la entrega anterior. El reporte muestra, por clase, precision, recall y F1-score: precision indica la proporción de predicciones positivas correctas, recall indica la proporción de ejemplos de esa clase correctamente identificados y F1-score es la media entre precision y recall. Esta vez también incluye support, que es el número de instancias de cada clase en el conjunto de prueba, la accuracy global y dos promedios, macro avg y weighted avg. Macro avg promedia todas las clases por igual. Weighted avg promedia ponderando por el support. Estas métricas ofrecen una visión más completa del desempeño del modelo y permiten evaluar casos con desbalance de clases.

Análisis y conclusión final

Después de la implementación del mismo algoritmo, pero uno hecho a mano y el otro utilizando frameworks, me quedó mucho más claro la utilidad y las maneras en las que se debe implementar.
Me parece muy interesante que con el dataset de tennis, el algoritmo hecho a mano haya obtenido resultados excelentes todo el tiempo, mientras que el que utiliza frameworks, le costó mucho más trabajo. Al hacer una reflexión sobre esto, pienso que la implementación pasada si sufría mucho de overfitting. Pero ha sido de mucha utilidad ver y hacer la comparación para entender que se puede mejorar. En este caso también se agrega la parte del one-hot-encoding y Grid Search, lo que hace que sea un modelo mucho más robusto y me genera más confianza.

El dataset de heart, para encontrar si se presenta o no una enfermedad en un paciente me parece también algo increíble. Por un lado quedé satisfecho con los resultados obtenidos al probar con este, sin embargo, me imagino un modelo que sí pueda ser utilizado en un escenario real médico, el modelo debe ser mucho más preciso y robusto para que pueda

utilizarse de una mejor manera, por lo que creo que aún existe un área de mejora para mi y poder desarrollar mejores modelos para mejores soluciones.

Por último, me parecieron dos grandes prácticas para no solo conocer cómo es la implementación en código de un árbol de decisión, si no para realmente comprender su utilidad y cómo estos pueden ser evaluados para su uso en un escenario real.

Dataset utilizado para el entrenamiento usando heart.csv

https://github.com/Sebastian-Espinoza-25/Machine-Learning-Implementation-With-Framework/blob/main/heart_train.csv

Dataset utilizado para la evaluación usando heart.csv

https://github.com/Sebastian-Espinoza-25/Machine-Learning-Implementation-With-Framework/blob/main/heart_test.csv