UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Maestría en Ciencia de Datos

Aprendizaje Automático

Alvaro Pequeño Mondragón
1726520

PIA

26 de julio de 2024

Para el proyecto final de la materia se nos pidió encontrar algún modelo de aprendizaje automático que pueda obtener al menos 0.75 en la métrica de ROC AUC haciendo uso de la validación cruzada. El conjunto de datos nos fue proporcionado y incluye 1 columna de id la cual fue removida, 8 columnas con valores numéricos y una ultima columna con valores booleanos (verdadero y falso). La ultima columna es nuestra variable a predecir.

Para empezar, se separó el conjunto de datos de manera que en una variable quedaran los datos de entrada o las “x” u en otra variable los datos de salida o la “y”:

```
#Se elimina la columna de ID ya que no es relevante para los modelos
Datos = Datos.drop(['id'], axis = 1, inplace = False)

#Se separa el conjunto en variables de entrada (X) y variable de salida o repuesta (Y):
datos_x = Datos.drop(['engagement'], axis = 1, inplace = False)
datos_y = Datos['engagement']
```

✓ 0.0s

Después se decidió seguir la estrategia de evaluar varios modelos para ver si alguno puede llegar a cumplir con las características que se requieren, los modelos elegidos son los siguientes:

- Árbol de decisión
- K-vecinos clasificador
- Regresión logística
- Extra tree classifier (Random forest)
- Gradient Boost classifier

```
#Se crean los modelos y sus posibles parametros:
arbol = DecisionTreeClassifier()
arbol_param = [{'criterion': ['gini', 'entropy', 'log_loss'], 'max_depth': list(range(1,60)), 'random_state': [0]}]

kvecinos = KNeighborsClassifier()
kvecinos_param = [{'n_neighbors': list(range(1,10)), 'weights': ['uniform', 'distance'], 'algorithm': ['ball_tree', 'kd_tree', 'brute']}]

reglog = LogisticRegression()
reglog_param = [{'penalty': ['l1', 'l2'], 'solver': ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']}]

bosque = ExtraTreesClassifier()
bosque_param = [{'n_estimators': list(range(20,120,20)), 'max_depth': list(range(1,30)), 'random_state': [0]}]

grad = GradientBoostingClassifier()
grad_param = [{'loss': ['log_loss', 'exponential'], 'n_estimators': list(range(10,110,10)), 'criterion': ['friedman_mse', 'squared_error'], 'random_state': [0]}]
```

✓ 0.0s

Los resultados de los modelos son los siguientes:

```
#Se generan los mejores modelos:
mejor_arbol = GridSearchCV(arbol, arbol_param, cv=5, scoring='roc_auc')
mejor_arbol.fit(datos_x, datos_y)
print('El mejor modelo para arbol de desición:')
print(mejor_arbol.best_params_)
print(mejor_arbol.best_score_)
print('----------------------------------------------------------------------------------------------------')
```

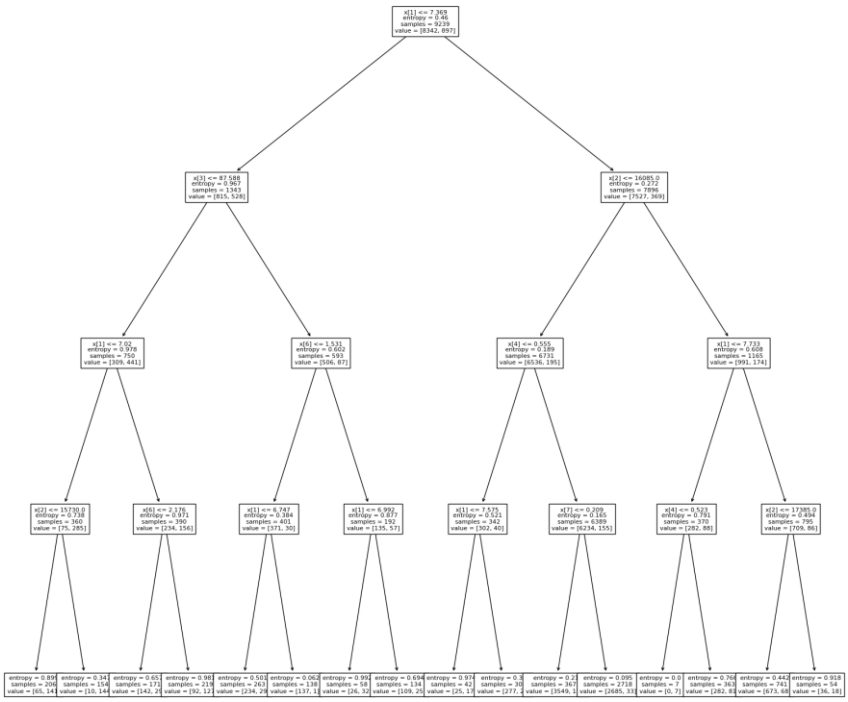
[5] ✓ 1m 14.1s

```
El mejor modelo para arbol de desición:
{'criterion': 'entropy', 'max_depth': 4, 'random_state': 0}
0.8475329559059064
----------------------------------------------------------------------------------------------------
```

Podemos ver que el mejor modelo de árbol de decisión nos da 0.84 para ROC AUC y los parámetros son los siguientes:

- Criterion = entropy
- Max_depth = 4
- Random_state = 0

También se puede observar el árbol del modelo:



El siguiente modelo fue el de k vecinos y los resultados son los siguientes:

```
#Se generan los mejores modelos:
scaler = StandardScaler()
scaler.fit(datos_x)
datos_x_scale = scaler.transform(datos_x)
mejor_kvecinos = GridSearchCV(kvecinos, kvecinos_param, cv=5, scoring='roc_auc')
mejor_kvecinos.fit(datos_x_scale, datos_y)
print('El mejor modelo para kvecinos cercanos:')
print(mejor_kvecinos.best_params_)
print(mejor_kvecinos.best_score_)
print('----------------------------------------------------------------------------------------------------')
```

✓ 25.1s

```
El mejor modelo para kvecinos cercanos:
{'algorithm': 'ball_tree', 'n_neighbors': 9, 'weights': 'distance'}
0.8262453225169306
----------------------------------------------------------------------------------------------------
```

Se observa que el mejor modelo nos da una calificación en ROC AUC de 0.82 y sus parámetros son los siguientes:

- Algorithm = ball_tree
- n_neighbors = 9
- weights = distance

```
#Se generan los mejores modelos:
mejor_reglog = GridSearchCV(reglog, reglog_param, cv=5, scoring='roc_auc')
mejor_reglog.fit(datos_x_scale, datos_y)
print('El mejor modelo para regresion logistica:')
print(mejor_reglog.best_params_)
print(mejor_reglog.best_score_)
print('-----------------------------------------------------------------------------------------------------')
```

✓ 0.7s

```
El mejor modelo para regresion logistica:
{'penalty': 'l2', 'solver': 'liblinear'}
0.8462562615549963
-----------------------------------------------------------------------------------------------------
```

Se observa que el mejor modelo de regresión logística nos da un valor para ROC AUC de 0.84, hay algunos modelos que no se pudieron hacer ya que los parámetros no eran compatibles, pero de los que si se realizaron en la validación cruzada el mencionado fue el mejor y sus parámetros son los siguientes:

- penalty: l2
- solver: liblinear

Después se evaluó el modelo de random forest y estos son los resultados:

```
#Se generan los mejores modelos:
mejor_bosque = GridSearchCV(bosque, bosque_param, cv=5, scoring='roc_auc')
mejor_bosque.fit(datos_x, datos_y)
print('El mejor modelo para random forest:')
print(mejor_bosque.best_params_)
print(mejor_bosque.best_score_)
print('-----------------------------------------------------------------------------------------------------')
```

✓ 2m 38.8s

```
El mejor modelo para random forest:
{'max_depth': 24, 'n_estimators': 100, 'random_state': 0}
0.8954296164765475
-----------------------------------------------------------------------------------------------------
```

El mejor modelo nos da un ROC AUC de 0.89 y sus parámetros son los siguientes:

- max_depth = 24
- n_estimators = 100
- random_state = 0

Por ultimo se evaluó el modelo de gradient boost y este fue el resultado:

```python
#Se generan los mejores modelos:
mejor_grad = GridSearchCV(grad, grad_param, cv=5, scoring='roc_auc')
mejor_grad.fit(datos_x, datos_y)
print('El mejor modelo para gradient boost:')
print(mejor_grad.best_params_)
print(mejor_grad.best_score_)
print('-----------------------------------------------------------------------------------------------')
```

✓ 4m 22.9s

```
El mejor modelo para gradient boost:
{'criterion': 'squared_error', 'loss': 'exponential', 'n_estimators': 100, 'random_state': 0}
0.8884233896894923
-----------------------------------------------------------------------------------------------
```

Podemos ver que el mejor modelo nos da un ROC AUC de 0.88 y los parámetros son los siguientes:

- criterion = squarred_error
- loss = exponential
- n_estimators = 100
- random_state = 0

Podemos ver que en todos los modelos se obtiene un valor de ROC AUC mayor a 0.75 donde los modelos que son ensambles muestran una mejor calificación.