Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

"Yo, como integrante de la comunidad estudiantil del Tecnológico de Monterrey, soy consciente de que la trampa y el engaño afectan mi dignidad como persona, mi aprendizaje y mi formación, por ello me comprometo a actuar honestamente, respetar y dar crédito al valor y esfuerzo con el que se elaboran las ideas propias, las de los compañeros y de los autores, así como asumir mi responsabilidad en la construcción de un ambiente de aprendizaje justo y confiable"

"Inteligencia artificial avanzada para la ciencia de datos I"

Técnicas de procesamiento de datos para el análisis estadístico y para la construcción de modelos

Equipo:

Frida Cano Falcón A01752953

Jorge Javier Sosa Briseño A01749489

Guillermo Romeo Cepeda Medina A01284015

Daniel Saldaña Rodríguez A00829752

Profesores:

Ivan Mauricio Amaya Contreras

Antonio Carlos Bento

Frumencio Olivas Alvarez

Blanca Rosa Ruiz Hernandez

Hugo Terashima Marín

Fecha de entrega: 28 de agosto de 2023

Introducción

En este documento, se presenta el proceso seguido en el proyecto, centrándose en la generación y comparación de modelos de aprendizaje máquina para abordar el reto planteado. En esta fase, se exploran diferentes tipos y configuraciones de modelos con el objetivo de seleccionar los más efectivos, cuyas decisiones se detallarán en la documentación. Para esto se hizo una investigación acerca de los modelos de clasificación, se encontraron varios modelos que cumplen con las características de nuestros datos para predecir de manera binaria un resultado, en nuestro caso la predicción de la muerte de un individuo en el Titanic. Entre estos modelos, se seleccionaron: KNeighborsClassifier, Support Vector Classifier (SVC), LogisticRegression, DecisionTreeClassifier, GaussianNB, RandomForestClassifier, GradientBoostingClassifier, DecisionTreeClassifier, MLPClassifier.

Entre estos modelos, se tiene la intención de seleccionar sólo tres modelos, por lo que se realizó un proceso para seleccionar el modelo que será explicado en el documento, así mismo se documentó el proceso para la preparación de datos para cada modelo, así como la separación del dataset para entrenar y predecir. A continuación se describe el proceso de la selección e implementación de los modelos.

Etapa 2
Se importan las librerías para implementar los modelos

La continuación del proyecto viene después de la limpieza de datos que realizamos en el entregable pasado, el siguiente paso fue importar las librerías necesarias para utilizar los modelos de predicción que nos interesan, en el siguiente código se evidencia:

Librerías para Modelos de clasificación:

K-Neighbors Classifier: Utiliza el método de vecinos más cercanos para la clasificación. Las predicciones se basan en las clases de los ejemplos cercanos en el espacio de características. Requiere ajustar el número de vecinos y posiblemente otras métricas de distancia.

```
from sklearn.neighbors import KNeighborsClassifier
```

SVC (Support Vector Classifier): Un clasificador que encuentra un hiperplano de separación óptimo entre clases. Puede usar diferentes funciones de kernel para transformar los datos en un espacio de mayor dimensión. Es útil para problemas de clasificación lineal y no lineal.

```
from sklearn.svm import SVC
```

Logistic Regression: Realiza la clasificación utilizando la función logística para modelar la probabilidad de pertenencia a una clase. Es efectivo para problemas de clasificación binaria y multiclase cuando las clases son linealmente separables.

```
from sklearn.linear_model import LogisticRegression
```

Decision Tree Classifier: Construye un árbol de decisión que divide recursivamente el espacio de características en regiones más puras. Es fácilmente interpretable, pero puede ser propenso al sobreajuste si no se controla adecuadamente.

```
from sklearn.tree import DecisionTreeClassifier
```

Gaussian NB: Implementa el clasificador Bayesiano Ingenuo Gaussiano, asumiendo que las características son independientes y distribuidas normalmente. A pesar de su simplicidad, puede funcionar sorprendentemente bien en muchos casos.

```
from sklearn.naive_bayes import GaussianNB
```

Random Forest Classifier: Crea múltiples árboles de decisión y combina sus predicciones para mejorar la robustez y precisión del modelo. Puede manejar características categóricas y numéricas, y es menos propenso al sobreajuste que un solo árbol.

```
from sklearn.ensemble import RandomForestClassifier
```

Gradient Boosting Classifier: Construye una secuencia de modelos que corrigen los errores del modelo anterior. Es útil para problemas de clasificación y regresión, y tiende a producir modelos de alta calidad.

```
from sklearn.ensemble import GradientBoostingClassifier
```

Decision Tree Classifier: Construye un árbol de decisión que divide recursivamente el espacio de características en regiones más puras. Es fácilmente interpretable, pero puede ser propenso al sobreajuste si no se controla adecuadamente.

```
from sklearn.neural_network import MLPClassifier
```

La librería cross_val_score se utiliza en la función que no hemos usado todavía en el proyecto para generalizar los análisis con gráficas y predicciones, así como se importa accuracy_score y metrics para obtener información de nuestros modelos y predicciones

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

Implementación de modelos

Para cada modelo se realizaron diferentes pruebas: utilizando diferentes hiperparámetros y utilizando dos bases de datos diferentes: una en donde eliminamos los pasajeros que contienen datos nulos y otra en donde a través del algoritmo de K-vecinos predecimos los valores nulos para no eliminar dichos datos. Se siguió el siguiente procedimiento:

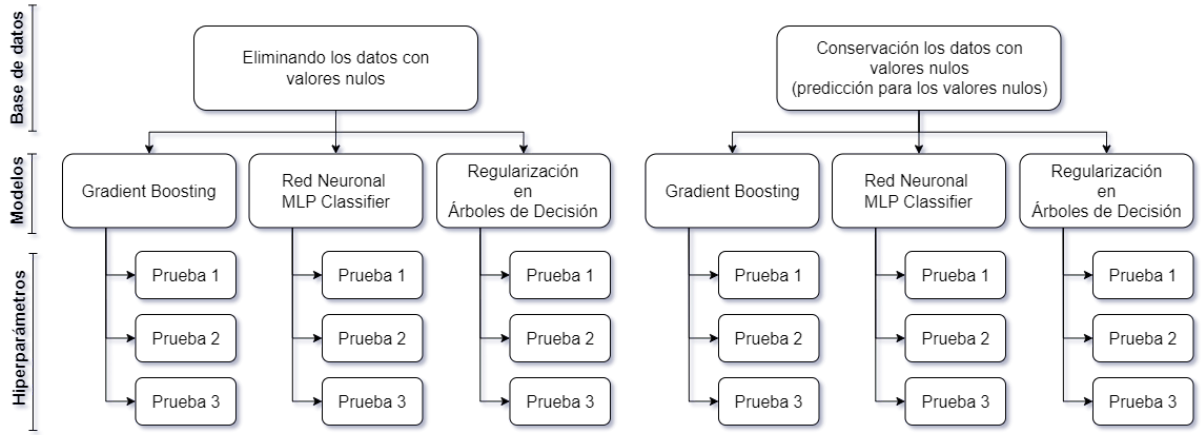


Fig 1. Metodología de pruebas

Para cada prueba se ajustaron los hiper parámetros con el objetivo de conseguir los mejores hiper parámetros y tener una mejor predicción

La siguiente función está diseñada para crear una meta-análisis de múltiples algoritmos para predecir la muerte de una persona en el titanic. Sin embargo, todavía no está implementada para usarla.

```
#Función para implementar meta-algoritmo
def modelfit(alg, dtrain, predictors, performCV=True,
printFeatureImportance=True, cv_folds=5):
#Fit the algorithm on the data
alg.fit(dtrain[predictors], dtrain['Disbursed'])

#Predict training set:
dtrain_predictions = alg.predict(dtrain[predictors])
```

```
dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]

#Perform cross-validation:
if performCV:
cv_score  =  cross_validation.cross_val_score(alg,  dtrain[predictors],
dtrain['Disbursed'], cv=cv_folds, scoring='roc_auc')

#Print model report:
print("\nModel Report")
print("Accuracy                :                %.4g"                %
metrics.accuracy_score(dtrain['Disbursed'].values, dtrain_predictions))
print("AUC           Score           (Train):           %f"           %
metrics.roc_auc_score(dtrain['Disbursed'], dtrain_predprob))

if performCV:
print("CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" %
(np.mean(cv_score),np.std(cv_score),np.min(cv_score),np.max(cv_score)))

#Print Feature Importance:
if printFeatureImportance:
feat_imp             =             pd.Series(alg.feature_importances_,
predictors).sort_values(ascending=False)
feat_imp.plot(kind='bar', title='Feature Importances')
plt.ylabel('Feature Importance Score')
```

En la siguiente sección se realizará un análisis del dataset sin valores nulos:

```
#Separamos tipos de variables en dependientes e independientes
feature_names = ['Age', 'Sex', 'Pclass', 'Embarked', 'Fare']
X_NonNull = df_NonNull[feature_names] # variables predictoras
y_NonNull = df_NonNull['Survived'] # variable de respuesta
#Se divide el dataset en entrenamiento y test, con un ratio de 80-20
X_NonNull_train,  X_NonNull_test,  y_NonNull_train,  y_NonNull_test  =
train_test_split(X_NonNull,  y_NonNull,  test_size=0.2,  random_state=42,
stratify=y_NonNull)
```

Después se crea un array de modelos el cuál tiene la función de almacenar los modelos para utilizar un ciclo para preparar, , entrenar y predecir cada uno de nuestros modelos que importamos, además se hace un análisis de precisión con la librería accuracy score

A continuación presentamos los scores de los modelos:

```
Name Score 0 KNN 0.643357 1 SVC 0.671329 2 LR 0.755245 3 DT 0.713287 4
GNB 0.755245 5 RF 0.769231 6 GB 0.783217 7 MLP 0.769231
```

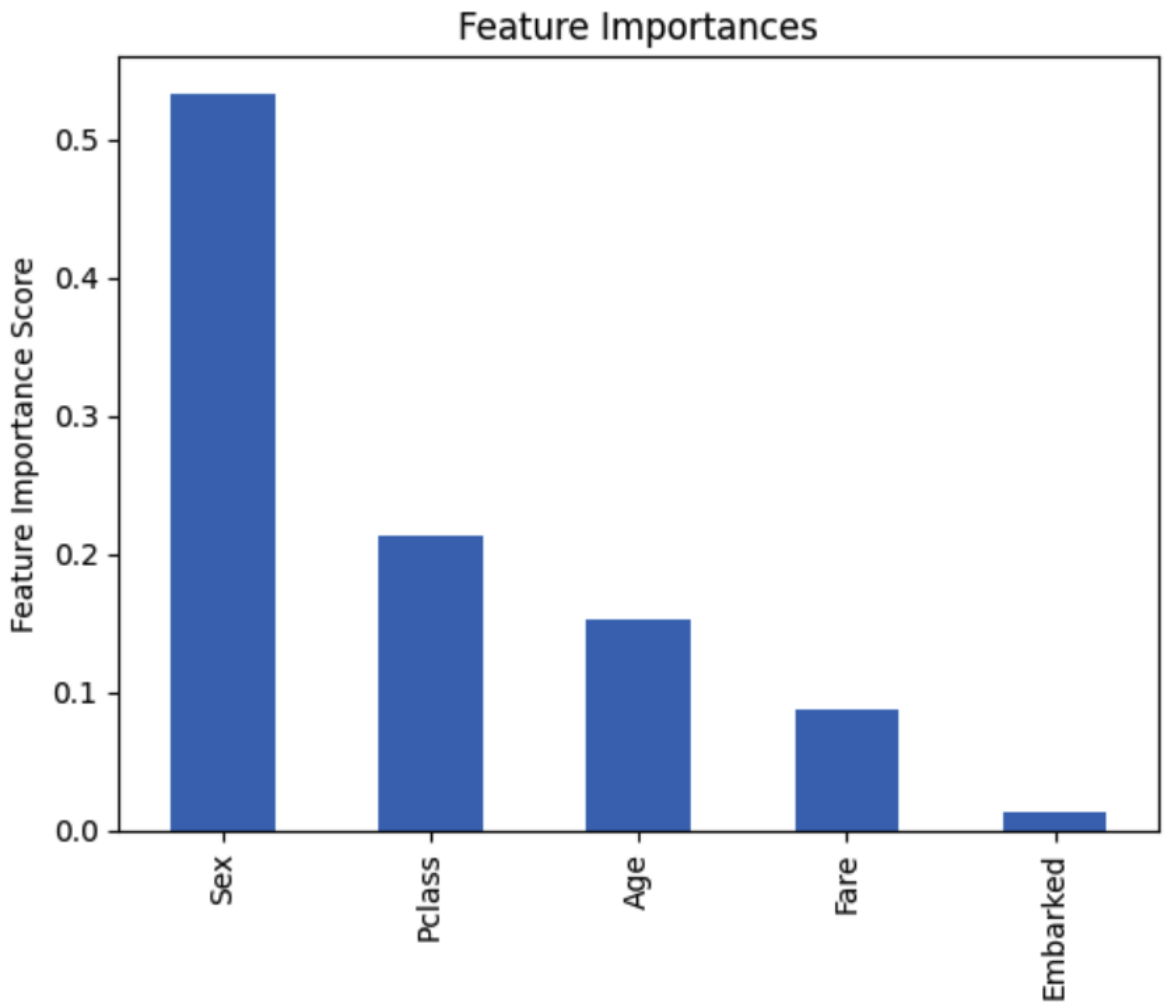
Se puede apreciar que el modelo gradient boosting fue el de mejor resultados entre los modelos probados, por lo que será uno de los modelos que utilizaremos para hacer nuestro análisis y predicción. Sin embargo, haremos algunos cambios en las parámetros para mejorar los resultados de la predicción.

```
Se implementa el modelo completamente
#Se llama la función con un learning_rate de 0.01
GB = GradientBoostingClassifier(learning_rate=0.01, subsample=0.6)
#Se ajustan los datos al modelo
GB.fit(X_NonNull_train, y_NonNull_train)

#Se predice el modelo
y_NonNull_pred = GB.predict(X_NonNull_test)
#Matríz de confusión
confusion_matrix=metrics.confusion_matrix(y_NonNull_test,y_NonNull_pred
)
```

Estos son los resultados que obtuvimos: `ccuracy: 0.8111888111888111 Precision: 0.7735849056603774 Recall: 0.9647058823529412 Fone: 0.8586387434554973`

Se implementa una gráfica de importancia de cada una de las variables para su predicción, podríamos descartar la variable "Embarked" según la gráfica



En las siguientes líneas se hizo el intento de mejorar los falsos positivos del modelo, pero no tuvimos éxito en mejorar el modelo

```
# accuracy_score(y_test, y_pred)
# #GB.predict_proba(X_test)
# y_pred2 = (GB.predict_proba(X_test)[:,1] >= 0.45).astype(bool)
# y_pred2 = y_pred2 * 1
# print(y_pred2)
# accuracy_score(y_test, y_pred2)
```

A continuación se aplicó el modelo de Xtreme Gradient Boosting, este hace un sistema de decisiones basado en un árbol de ramas, o k-tree. Este nos ofrece una variante del método de gradient boosting con una mejor solvencia computacional, ya que es más eficiente que el ya mencionado debido a que realiza computación en paralelo.

```
xGB = xgb.XGBClassifier(learning_rate = 0.1, subsample = 0.2,
min_child_weight = 2, reg_alpha = 1.5)
xGB.fit(X_NonNull_train, y_NonNull_train)
y_NonNull_pred = xGB.predict(X_NonNull_test)
confusion_matrix = metrics.confusion_matrix(y_NonNull_test,
y_NonNull_pred)
accuracy = (confusion_matrix[0,0] + confusion_matrix[1,1]) /
(confusion_matrix[0,0]+confusion_matrix[0,1]+confusion_matrix[1,0]+conf
usion_matrix[1,1])
```

Y se puede apreciar como subieron las métricas de precisión y exactitud en la predicción de resultados.

```
Accuracy: 0.8391608391608392 Precision: 0.8522727272727273 Recall:
0.8823529411764706 Fone: 0.8670520231213872
```

Implementación de red neuronal - Multi-layer Perceptron Classifier (MLP)

Este clasificador implementa el algoritmo de aprendizaje profundo supervisado MLP que se entrena mediante el método de retropropagación (backpropagation) está compuesto por múltiples capas de neuronas: capas de entrada, capas ocultas y capas de salida, cada una conectada a través de conexiones ponderadas (*Neural network models (supervised)*, s. f.). Se escogió este modelo para realizar la predicción de los decesos en el titanic debido a su popularidad actual en la sociedad, puesto que este modelo es la base de las nuevas tecnologías que estamos usando como la nueva

Para realizar la implementación:

```
#Se crea una variable de red neuronal para clasificación, con
parámetros de 100 capas ocultas y una iteración máxima de 10000
mlp_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000)
# Capas y Parámetros
#Se ajustan los datos
mlp_model.fit(X_NonNull_train, y_NonNull_train)
#Se calcula una puntuación de la red neuronal
mlp_score = mlp_model.score(X_NonNull_test, y_NonNull_test)
```

Y se obtuvo un score de MLP Score: 0.7342657342657343

Se crea una variable para regularizar los datos con DesicionTreeClasifier con parámetros de las capas de la red neuronal.

```
regularized_tree = DecisionTreeClassifier(max_depth=5,
min_samples_split=5) # Configura los parámetros
#Se ajustan los datos
```

```
regularized_tree.fit(X_NonNull_train, y_NonNull_train)
regularized_tree_score     =     regularized_tree.score(X_NonNull_test,
y_NonNull_test)
```

Esta regularización mejoró la exactitud del modelo hasta un 0.8181818181818182

Conclusión

A lo largo de esta entrega, se ha llevado acabo un proceso exhaustivo de selección, implementación y comparación de modelos de aprendizaje automático con el fin de lograr el objetivo de predecir la supervivencia de los pasajeros en el Titanic. En el transcurso de esta fase, se estudiaron una variedad de algoritmos de clasificación y se evaluaron en diferentes configuraciones para determinar cuales ofrecen el mejor rendimiento en materia de las métricas de evaluación.

Entre el conjunto de los modelos analizados, destacaron dos enfoques particulares: el algoritmo Xtreme Gradient Boosting (XGBoost) y el uso de redes neuronales a través del Multi-Layer Perceptron Classifier (MLP). Ambos métodos demostraron resultados prometedores en términos de precisión, recall y F1-Score. El algoritmo XGBoost, que implementa un sistema de decisiones basado en árboles, se resaltó por su capacidad para manejar conjuntos de datos complejos y generar predicciones precisas.En contraste, la red neuronal MLP mostró una capacidad para aprender patrones en los datos y adaptarse a relaciones no lineales, lo que contribuyó a su alto rendimiento.

Es importante mencionar que, los mejores rendimientos encontrados en los modelos fueron extremadamente mejores cuando se trabajó con los datos que no contenían valores nulos. La eliminación de registros con datos faltantes permitió a los modelos centrarse en relaciones significativas en los datos disponibles y generar predicciones más precisas. Así mismo, se experimentó con diferentes valores de hiper-parámetros y técnicas de regularización. Las iteraciones constantes permitieron identificar combinaciones óptimas que maximizaron la precisión y la generalización de los modelos.

Finalmente, la conjunción de cada una de las acciones llevadas a cabo previamente, que van desde la selección adecuada de los modelos hasta la exitosa preparación de los datos, arrojó resultados con valores de puntuación superiores al 70%. Además, la implementación de algoritmos como XGBoost y MLP Classifier, junto con la eliminación de valores nulos, condujo a la obtención de resultados sumamente satisfactorios en términos de precisión y rendimiento global lo que nos deja en la dirección idónea para continuar trabajando con estos métodos en la parte final del proyecto.

Referencias

- *Neural network models (supervised)*. (s. f.). Scikit Learn. https://scikit-learn.org/stable/modules/neural_networks_supervised.html