Análisis y Reporte sobre el desempeño del modelo.

Maximiliano Martinez Marquez A01251527

In [3]:

```
from google.colab import drive

drive.mount("/content/gdrive")
!pwd  # show current path
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
call drive.mount("/content/gdrive", force_remount=True).
/content
```

In [4]:

```
%cd "/content/gdrive/MyDrive/AD2022"
!ls  # show current directory
/content/gdrive/MyDrive/AD2022
Datos_Titanic.README   iris.data     PlayDataset.csv  wine.data
ds_salaries.csv        iris.names    test.csv         wine.names
gender_submission.csv  mercurio.csv  train.csv
winequality_red.csv
```

In [5]:

```
!pip install mlxtend --upgrade --no-deps
Looking in indexes: https://pypi.org/simple, https://us-
python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.7/dist-
packages (0.14.0)
Collecting mlxtend
  Downloading mlxtend-0.20.0-py2.py3-none-any.whl (1.3 MB)
     |████████████████████████████████| 1.3 MB 7.8 MB/s
Installing collected packages: mlxtend
  Attempting uninstall: mlxtend
    Found existing installation: mlxtend 0.14.0
    Uninstalling mlxtend-0.14.0:
      Successfully uninstalled mlxtend-0.14.0
Successfully installed mlxtend-0.20.0
```

In [16]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, learning_curve,
GridSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics
from sklearn.metrics import roc_curve
```

```
from matplotlib import pyplot as plt

from mlxtend.evaluate import bias_variance_decomp
```

In [5]:

```
red_wine = pd.read_csv('winequality_red.csv', header = 0)
y = red_wine['class']
X = red_wine.drop('class', axis=1)
```

Separación y evaluación del modelo con un conjunto de prueba y un conjunto de validación (Train/Test/Validation)

In [6]:

```
'''
Split para obtener conjunto de prueba y de entrenamiento.

:Prueba: = (20%)
'''
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
'''
Split para obtener conjunto de validación.

:Validación: = (20%)
:Entrenamiento: = (60%)
'''
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.25, random_state=1)
```

In [7]:

```
lr = LogisticRegression(random_state=1)
```

Comportamiento esperado de Regresión Logística:

- Sesgo alto
- Varianza baja

In [ ]:

```
lr.fit(X_train, y_train)
```

In [14]:

```
print(f"{lr.score(X_train, y_train):.3f}")
```

```
0.749
```

In [ ]:

```
print(f"{lr.score(X_val, y_val):.3f}")
```

```
0.709
```

Al realizar una comparación en los scores del set de entrenamiento y el set de validación, se puede decir que el modelo no está overfit ya que el valor score de entrenamiento no se encuentra muy alejado del valor de score del set de validación.

In [ ]:

```
#set up plotting area
```

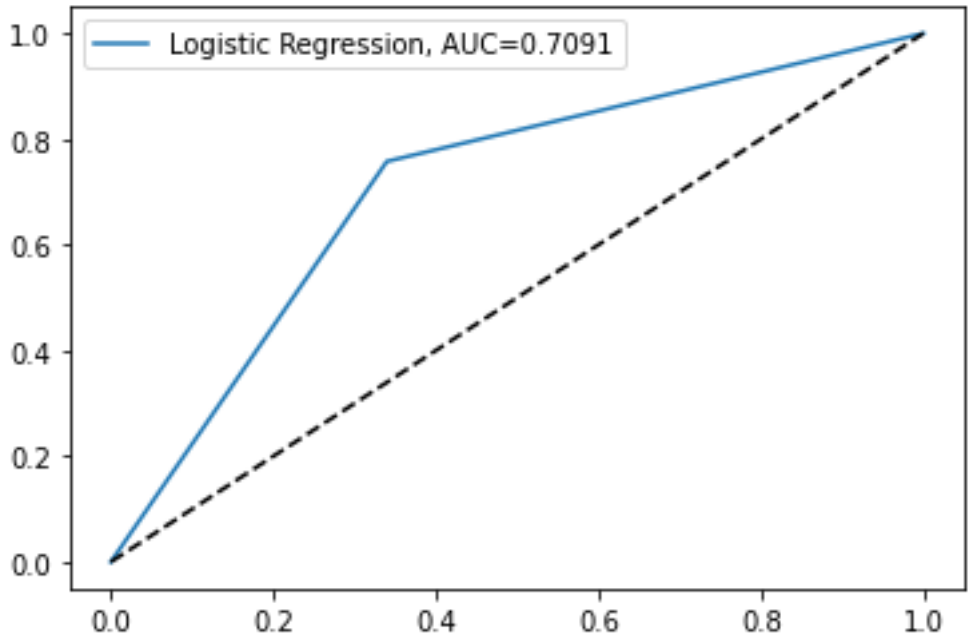
```
plt.figure(0).clf()

#fit logistic regression model and plot ROC curve
y_pred = lr.predict(X_val)
fpr, tpr, _ = metrics.roc_curve(y_val, y_pred)
auc = round(metrics.roc_auc_score(y_val, y_pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))
plt.plot([0,1],[0,1],color="black",linestyle="--")

#add legend
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7f198a35fc10>
```



El siguiente paso es encontrar el valor del sesgo y de variación del modelo.

Luego, para poder evaluar nuestro modelo, se comparó con otro modelo, Árbol de decisión.

In [ ]:

```
train_sizes, train_scores, valid_scores = learning_curve(
    lr, X_train, y_train
)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

valid_mean = np.mean(valid_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)
```

In [11]:

```
plt.subplots(1, figsize=(10,10))
```

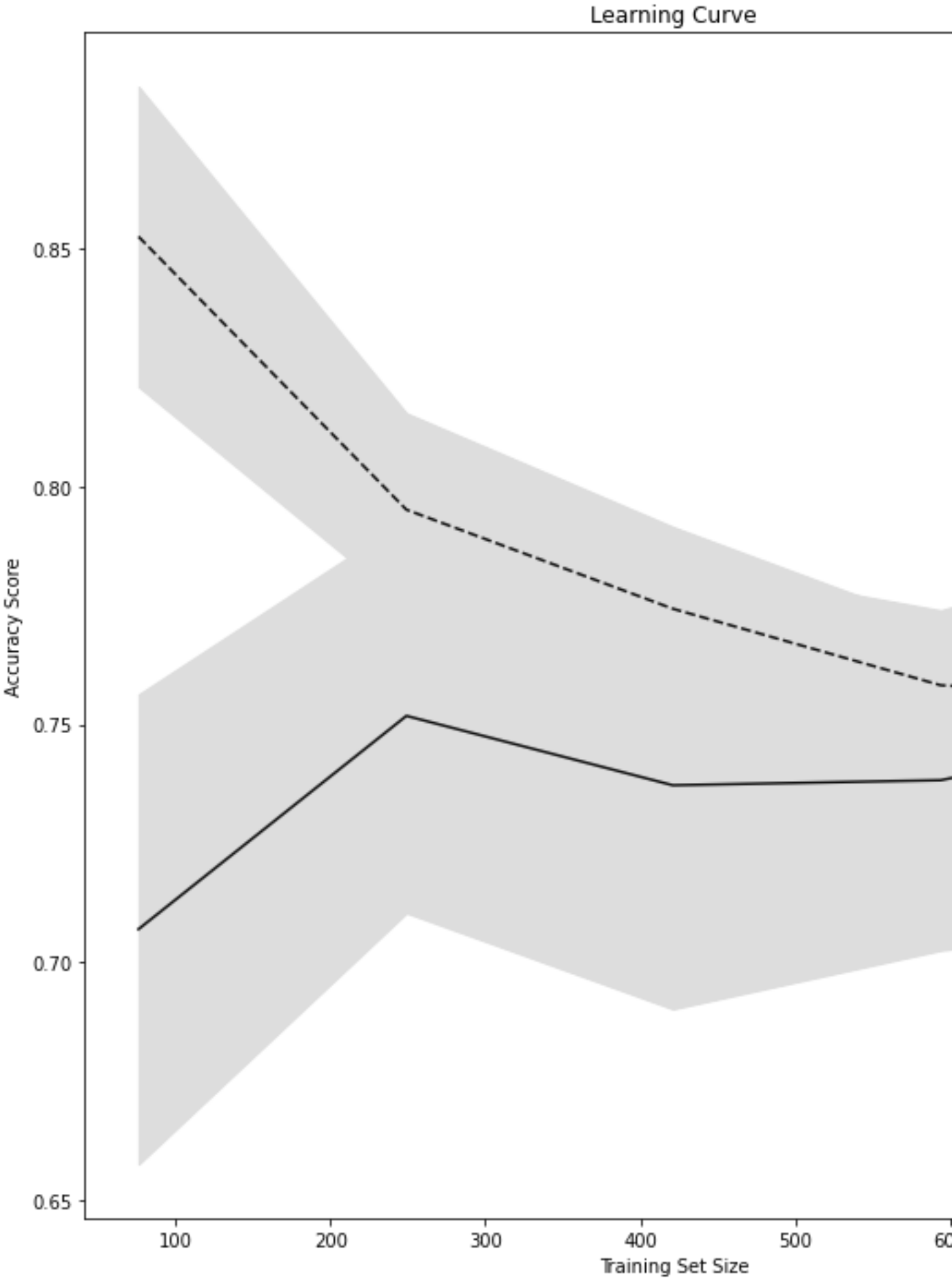
```
plt.plot(train_sizes, train_mean, '--', color="#111111", label="Training
score")
plt.plot(train_sizes, valid_mean, color="#111111", label="Cross-
Validation score")

plt.fill_between(train_sizes, train_mean - train_std, train_mean +
train_std, color="#DDDDDD")
plt.fill_between(train_sizes, valid_mean - valid_std, valid_mean +
valid_std, color="#DDDDDD")

plt.title("Learning Curve")
plt.xlabel("Training Set Size"), plt.ylabel("Accuracy Score"),
plt.legend(loc="best")
plt.tight_layout()
plt.show
```

Out[11]:

```
<function matplotlib.pyplot.show(*args, **kw)>
```


Learning Curve
Accuracy Score
0.85
0.80
0.75
0.70
0.65
100
200
300
400
500
Training Set Size

A partir de la curva de aprendizaje anterior, se puede observar que que modelo cuenta con underfitting y tambien se observa que despues al llegar a la mitad de los datos, la precisión disminuye.

In [ ]:

```
X_train_v = X_train.values
y_train_v = y_train.values

X_val_v = X_val.values
y_val_v = y_val.values

avg_expected_loss_lr, avg_bias_lr, avg_var_lr = bias_variance_decomp(
        lr, X_train_v, y_train_v, X_val_v, y_val_v,
        loss='mse',
        random_seed=1)
```

In [15]:

```
print('Average expected loss: %.3f' % avg_expected_loss_lr)
print('Average bias: %.3f' % avg_bias_lr)
print('Average variance: %.3f' % avg_var_lr)
Average expected loss: 0.288
Average bias: 0.251
Average variance: 0.036
```

A partir de los resultados anteriores, se puede observar que el sesgo es mucho mayor que la varianza, el cual se habia comentado que era el comportamiento esperado.

In [ ]:

```
dtr = DecisionTreeClassifier(random_state=1)

avg_expected_loss_dtr, avg_bias_dtr, avg_var_dtr = bias_variance_decomp(
        dtr, X_train_v, y_train_v, X_val_v, y_val_v,
        loss='mse',
        random_seed=1)
```

In [ ]:

```
print('Average expected loss: %.3f' % avg_expected_loss_dtr)
print('Average bias: %.3f' % avg_bias_dtr)
print('Average variance: %.3f' % avg_var_dtr)
Average expected loss: 0.293
Average bias: 0.143
Average variance: 0.149
```

En el caso de Árbol de Decisión, se puede observar que el sesgo y la varianza se encuentran balanceados.

A partir de esto, y comparando con los valores de sesgo y varianza de regresión logística, se puede concluir, que el modelo cuenta con un sesgo entre alto y varianza baja.

Entonces, para reducir el sesgo se puede realizar una

In [26]:

```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
```

```
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

In [27]:

```
new_lr = GridSearchCV(lr, param_grid = param_grid, cv = 3, verbose=True,
n_jobs=-1)
```

In [ ]:

```
best_lr = new_lr.fit(X_train, y_train)
```

In [34]:

```
best_lr.best_estimator_.get_params()
```

Out[34]:

```
{'C': 0.23357214690901212,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': 1,
 'solver': 'newton-cg',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [33]:

```
print(f"{best_lr.score(X_val, y_val):.3f}")
0.725
```

Ahora que está optimizado el modelo y se intentó arreglar el sesgo, se realizará una predicción sobre el set de prueba.

In [35]:

```
print(f"{best_lr.score(X_test, y_test):.3f}")
0.731
```