Ethan Enrique Verduzco Pérez | A01066955

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.model_selection import KFold
from sklearn.svm import SVR
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
```

```
from sklearn.model_selection import cross_validate
def cross_validation(model, _X, _y, _cv=5):

      _scoring = ['accuracy', 'precision', 'recall', 'f1']
      results = cross_validate(estimator=model,
                               X=_X,
                               y=_y,
                               cv=_cv,
                               scoring=_scoring,
                               return_train_score=True)

      return {"Training Accuracy scores": results['train_accuracy'],
              "Mean Training Accuracy": results['train_accuracy'].mean()*100,
              "Training Precision scores": results['train_precision'],
              "Mean Training Precision": results['train_precision'].mean(),
              "Training Recall scores": results['train_recall'],
              "Mean Training Recall": results['train_recall'].mean(),
              "Training F1 scores": results['train_f1'],
              "Mean Training F1 Score": results['train_f1'].mean(),
              "Validation Accuracy scores": results['test_accuracy'],
              "Mean Validation Accuracy": results['test_accuracy'].mean()*100,
              "Validation Precision scores": results['test_precision'],
              "Mean Validation Precision": results['test_precision'].mean(),
              "Validation Recall scores": results['test_recall'],
              "Mean Validation Recall": results['test_recall'].mean(),
```

```
            "Validation F1 scores": results['test_f1'],
            "Mean Validation F1 Score": results['test_f1'].mean()
            }

# Función obtenida de
# https://www.section.io/engineering-education/how-to-implement-k-fold-cross-validat
```

Datos

```
df = pd.read_csv('/content/water_potability (1).csv')
```

```
df.head(5)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11

```
df.describe()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620

```
df.isnull().sum()
```

```
ph                 491
Hardness             0
Solids               0
```

```
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

Llenamos los valores faltantes con la media de los datos

```
df['ph'].fillna(value=df['ph'].mean(), inplace=True)
df['Sulfate'].fillna(value=df['Sulfate'].mean(), inplace=True)
df['Trihalomethanes'].fillna(value=df['Trihalomethanes'].mean(), inplace=True)
```

```
df.isnull().sum()
```

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

Normalizamos los datos

```
scaler = MinMaxScaler()
scale = scaler.fit_transform(df)
df_scale = pd.DataFrame(scale, columns = df.columns)
df_scale
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_car
0	0.505771	0.571139	0.336096	0.543891	0.680385	0.669439	0.313
1	0.265434	0.297400	0.300611	0.491839	0.581699	0.719411	0.497
2	0.578509	0.641311	0.321619	0.698543	0.581699	0.414652	0.562
3	0.594055	0.605536	0.356244	0.603314	0.647347	0.317880	0.622
4	0.649445	0.484851	0.289922	0.484900	0.514545	0.379337	0.358

Entrena por lo menos 2 algoritmos de clasificación

SVC

3274	0.366197	0.664407	0.191490	0.465860	0.581699	0.387157	0.343

Preprocessing

3276 rows × 10 columns

```
X = df_scale.drop('Potability', axis=1)
y = df_scale['Potability']
```

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
y
```

```
0       0.0
1       0.0
2       0.0
3       0.0
4       0.0
       ...
3271    1.0
3272    1.0
3273    1.0
3274    1.0
3275    1.0
Name: Potability, Length: 3276, dtype: float64
```

Training the Algorithm

To train the kernel SVM, we use the same SVC class of the Scikit-Learn's svm library. The difference lies in the value for the kernel parameter of the SVC class. In the case of the simple SVM we used "linear" as the value for the kernel parameter. However, for kernel SVM you can use Gaussian,

Calibraciones SVC

1. Polynomial Kernel

In the case of polynomial kernel, you also have to pass a value for the degree parameter of the SVC class. This basically is the degree of the polynomial.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=3)
svclassifier.fit(X_train, y_train)
y_pred1 = svclassifier.predict(X_test)
```

Evaluating the Algorithm

```
metrics.accuracy_score(y_test, y_pred1)
```

```
0.6509146341463414
```

```
metrics.accuracy_score(y_test, y_pred1) #porcentaje de prediccion de valores verdader
```

```
0.6509146341463414
```

```
metrics.recall_score(y_test,y_pred1) #indica falsos negativos
```

```
0.2047244094488189
```

```
metrics.precision_score(y_test,y_pred1) #falsos positivos
```

```
0.6582278481012658
```

```
metrics.f1_score(y_test,y_pred1) #media armonica entre recall y presicion
```

```
0.3123123123123123
```

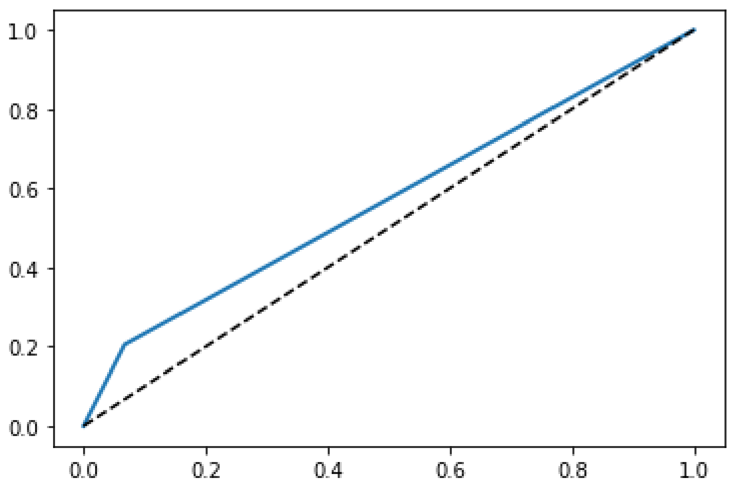
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred1))
print(classification_report(y_test, y_pred1))
```

```
[[375  27]
```

```
 [202  52]]
              precision    recall  f1-score   support

         0.0       0.65      0.93      0.77       402
         1.0       0.66      0.20      0.31       254

    accuracy                           0.65       656
   macro avg       0.65      0.57      0.54       656
weighted avg       0.65      0.65      0.59       656
```

curva ROC y AUC

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred1)
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```



Cross Validation

```
SVC_result = cross_validation(svclassifier, X, y, 5)
print(SVC_result)
```

```
{'Training Accuracy scores': array([0.70267176, 0.69362839, 0.69591759, 0.706600
```

2. Gaussian Kernel

To use Gaussian kernel, you have to specify 'rbf' as value for the Kernel parameter of the SVC class.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
```

Prediction and Evaluation

```
metrics.accuracy_score(y_test, y_pred)
```

```
0.6646341463414634
```

```
metrics.accuracy_score(y_test, y_pred) #porcentaje de prediccion de valores verdadero
```

```
0.6646341463414634
```

```
metrics.recall_score(y_test,y_pred) #indica falsos negativos
```

```
0.20866141732283464
```

```
metrics.precision_score(y_test,y_pred) #falsos positivos
```

```
0.7361111111111112
```

```
metrics.f1_score(y_test,y_pred) #media armonica entre recall y presicion
```

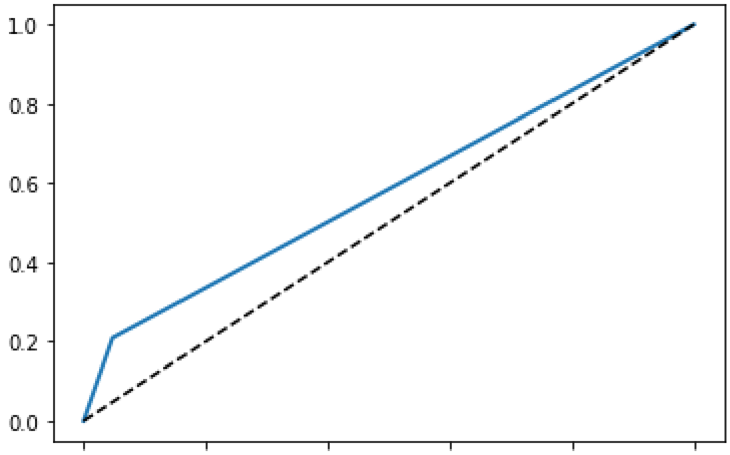
```
0.3251533742331288
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[383  19]
 [201  53]]
              precision    recall  f1-score   support

         0.0       0.66      0.95      0.78       402
         1.0       0.74      0.21      0.33       254

    accuracy                           0.66       656
   macro avg       0.70      0.58      0.55       656
weighted avg       0.69      0.66      0.60       656
```

curva ROC y AUC

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```



Cross Validation

```
SVC_result = cross_validation(svclassifier, X, y, 5)
print(SVC_result)
```

```
{'Training Accuracy scores': array([0.71564885, 0.70431133, 0.71423121, 0.721480
```

3. Sigmoid Kernel

To use the sigmoid kernel, you have to specify 'sigmoid' as value for the kernel parameter of the SVC class.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
```

Prediction and Evaluation

```
metrics.accuracy_score(y_test, y_pred)
```

```
0.6128048780487805
```

```
metrics.accuracy_score(y_test, y_pred) #porcentaje de prediccion de valores verdadero
```

```
0.6128048780487805
```

```
metrics.recall_score(y_test,y_pred) #indica falsos negativos
```

```
0.0
```

```
metrics.precision_score(y_test,y_pred) #falsos positivos
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
0.0
```

```
metrics.f1_score(y_test,y_pred) #media armonica entre recall y presicion
```

```
0.0
```

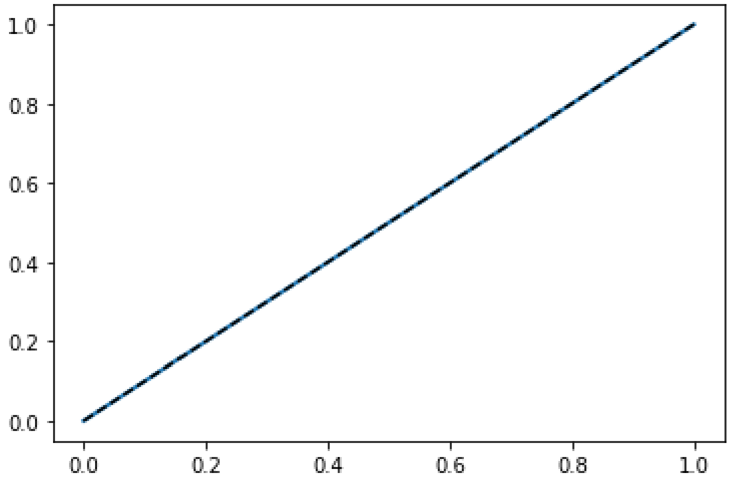
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[402   0]
 [254   0]]
              precision    recall  f1-score   support

         0.0       0.61      1.00      0.76       402
         1.0       0.00      0.00      0.00       254

    accuracy                           0.61       656
   macro avg       0.31      0.50      0.38       656
weighted avg       0.38      0.61      0.47       656

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
```

curva ROC y AUC

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label = None):
```

```
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```



Cross Validation

```
SVC_result = cross_validation(svclassifier, X, y, 5)
print(SVC_result)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
{'Training Accuracy scores': array([0.60992366, 0.60969096, 0.60969096, 0.610072
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318:
  _warn_prf(average, modifier, msg_start, len(result))
```

Decision Tree Classifier

```
columns = ['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity']
X = df[columns]
y = df.Potability
```

División de los datos en entrenamiento y prueba

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
```

Clasificador

```
tree = DecisionTreeClassifier()

#Entrenamiento del modelo
model_tree = tree.fit(X_train, y_train)
```

Predicción de etiquetas

```
y_pred = model_tree.predict(X_test)
```

Evaluación

```
metrics.accuracy_score(y_test, y_pred)
```

```
0.5483214649033571
```

```
metrics.accuracy_score(y_test, y_pred) #porcentaje de prediccion de valores verdadero
```

```
0.5483214649033571
```

```
metrics.recall_score(y_test,y_pred) #indica falsos negativos
```

```
0.44221105527638194
```

```
metrics.precision_score(y_test,y_pred) #falsos positivos
```

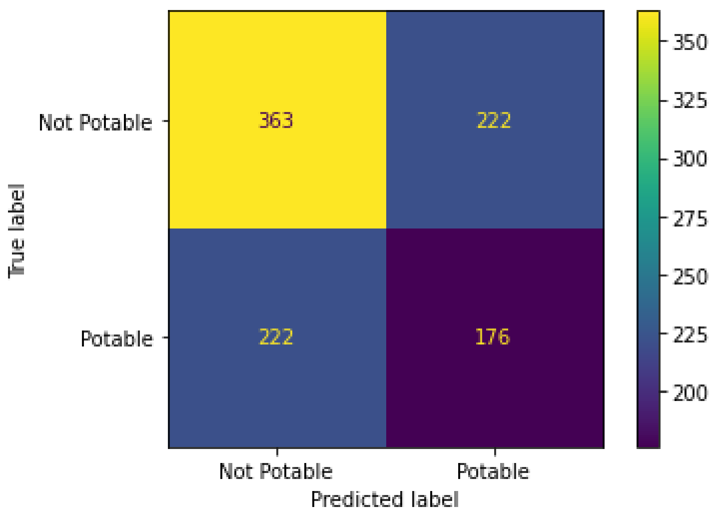
```
0.44221105527638194
```

```
metrics.f1_score(y_test,y_pred) #media armonica entre recall y presicion
```

```
0.44221105527638194
```

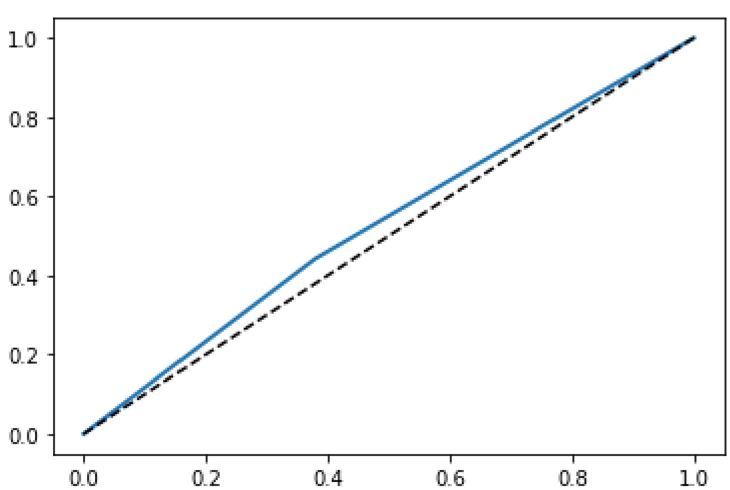
```
matriz = metrics.confusion_matrix(y_test,y_pred)
matriz

     array([[363, 222],
            [222, 176]])
```

```
matriz_plot = metrics.ConfusionMatrixDisplay(confusion_matrix=matriz, display_labels=
matriz_plot.plot()
plt.show()
```



curva ROC y AUC

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```



```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[363 222]
 [222 176]]
              precision    recall  f1-score   support

           0       0.62      0.62      0.62       585
           1       0.44      0.44      0.44       398

    accuracy                           0.55       983
   macro avg       0.53      0.53      0.53       983
weighted avg       0.55      0.55      0.55       983
```

Cross Validation

```
decision_tree_result = cross_validation(tree, X, y, 5)
print(decision_tree_result)
```

```
{'Training Accuracy scores': array([1., 1., 1., 1., 1.]), 'Mean Training Accurac
```

Calibración 2

```
tree = DecisionTreeClassifier(criterion = "entropy", max_depth =3, splitter = "best")

#Entrenamiento del modelo
model_tree = tree.fit(X_train, y_train)


y_pred = model_tree.predict(X_test)


metrics.accuracy_score(y_test, y_pred)
```

```
0.62970498474059
```

```
metrics.accuracy_score(y_test, y_pred) #porcentaje de prediccion de valores verdadero
```

```
0.62970498474059
```

```
metrics.recall_score(y_test,y_pred) #indica falsos negativos
```

```
0.18341708542713567
```

```
metrics.precision_score(y_test,y_pred) #falsos positivos
```

    0.6517857142857143

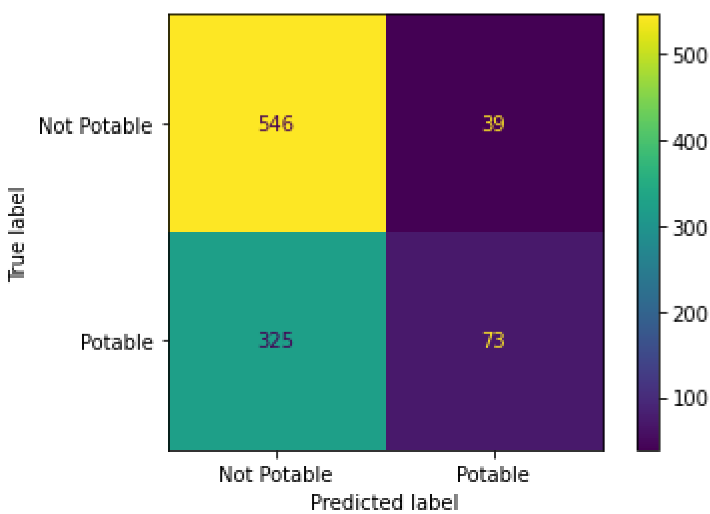
```
metrics.f1_score(y_test,y_pred) #media armonica entre recall y presicion
```

    0.28627450980392155

```
matriz = metrics.confusion_matrix(y_test,y_pred)
matriz
```

    array([[546, 39],
           [325, 73]])

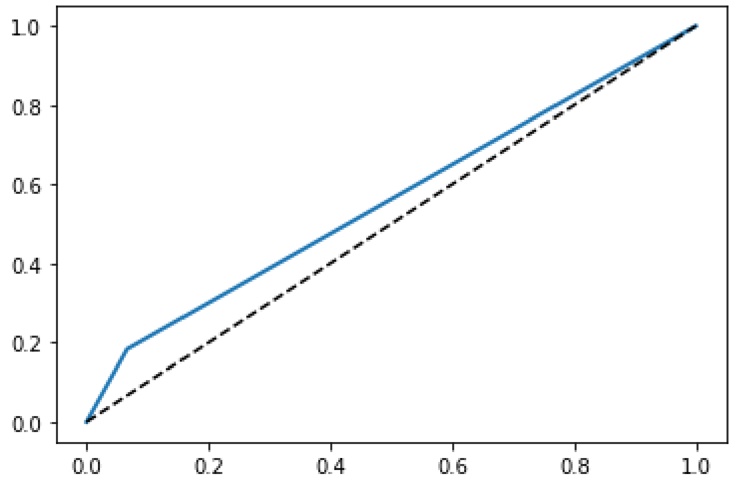
```
matriz_plot = metrics.ConfusionMatrixDisplay(confusion_matrix=matriz, display_labels=
matriz_plot.plot()
plt.show()
```




```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[546  39]
 [325  73]]
              precision    recall  f1-score   support

           0       0.63      0.93      0.75       585
           1       0.65      0.18      0.29       398

    accuracy                           0.63       983
   macro avg       0.64      0.56      0.52       983
weighted avg       0.64      0.63      0.56       983
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```



```
decision_tree_result = cross_validation(tree, X, y, 5)
print(decision_tree_result)
```

```
{'Training Accuracy scores': array([0.62519084, 0.62914918, 0.6474628 , 0.648607
```

Calibración 3

```
tree = DecisionTreeClassifier( max_depth =2, splitter = "best")

#Entrenamiento del modelo
model_tree = tree.fit(X_train, y_train)
```

```
y_pred = model_tree.predict(X_test)
```

```
metrics.accuracy_score(y_test, y_pred)
```

```
0.6042726347914548
```

```
metrics.accuracy_score(y_test, y_pred) #porcentaje de prediccion de valores verdadero
```

```
0.6042726347914548
```

```
metrics.recall_score(y_test,y_pred) #indica falsos negativos
```

```
0.10301507537688442
```

```
metrics.precision_score(y_test,y_pred) #falsos positivos
```

```
0.5616438356164384
```

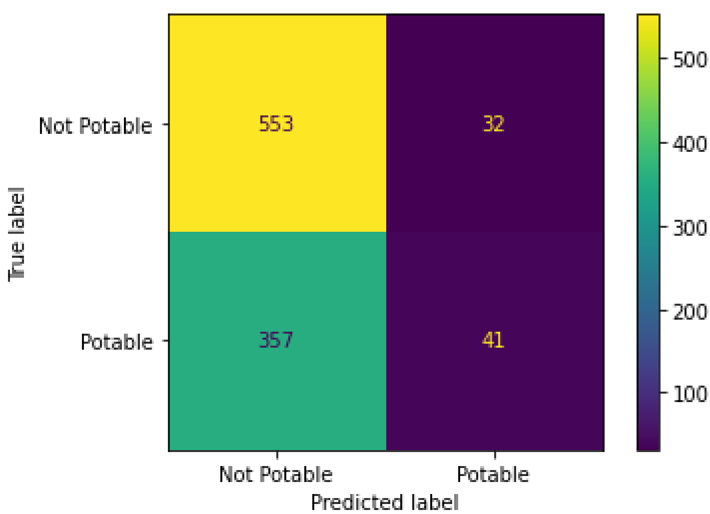
```
metrics.f1_score(y_test,y_pred) #media armonica entre recall y presicion
```

```
0.17409766454352443
```

```
matriz = metrics.confusion_matrix(y_test,y_pred)
matriz
```

```
array([[553,  32],
       [357,  41]])
```

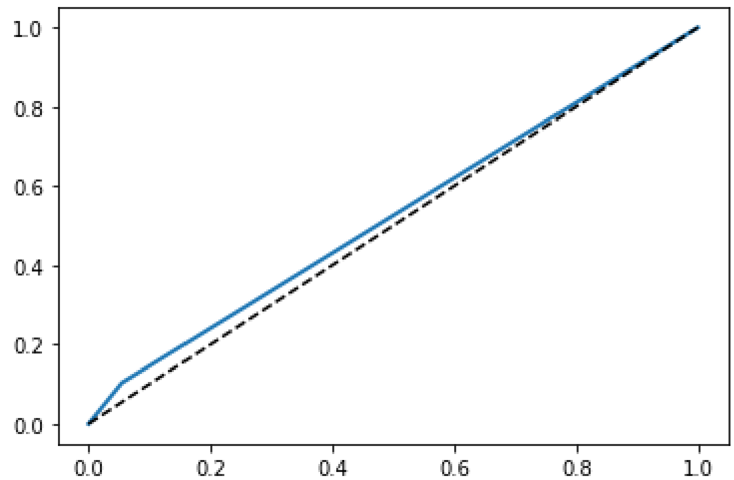
```
matriz_plot = metrics.ConfusionMatrixDisplay(confusion_matrix=matriz, display_labels=
matriz_plot.plot()
plt.show()
```




```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[553  32]
 [357  41]]
              precision    recall  f1-score   support

           0       0.61      0.95      0.74       585
           1       0.56      0.10      0.17       398

    accuracy                           0.60       983
   macro avg       0.58      0.52      0.46       983
weighted avg       0.59      0.60      0.51       983
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth =2, label=label)
    plt.plot([0,1],[0,1],'k--')
plot_roc_curve(fpr, tpr)
plt.show()
```




```
decision_tree_result = cross_validation(tree, X, y, 5)
print(decision_tree_result)
```

```
{'Training Accuracy scores': array([0.62290076, 0.62190004, 0.63487219, 0.646699
```

Análisis de Resultados

Después de analizar llevar a cabo el proceso de normalización de datos, entrenamiento de los modelos, calibración de los hiperparámetros de los modelos, la evaluación y la validación cruzada, fue posible determinar que el método de clasificación que mejor se adapta a los datos es el de Support Vector Classifier, con los siguientes hiperparámetros:

kernel='rbf'

En el que obtuvimos los siguientes resultados:

precision_score de 0.74 y un accuracy mayor a 0.65 de igual manera, siendo el algoritmo de clasificación con los mejores resultados.