**KNN Classifier**

**Steps on how to run project**

1. Load the train data set then use LabelEncoder() to convert

‘Sex’ feature from string to numerical value

1. Import GridSearchCV Then import train\_valid\_test\_split and use it to divide the data to sizes according to our needs.
2. We define our KNN Classifier then we start fitting the data and round the numbers
3. Using GridSearchCV we brute search all the possible value of the hyperparameters and get the best hyperparameter value
4. Then we output the hyperparameters with the best score
5. Then we start calculating the different metrics for our classifer using their imported libraries.

Those metrics include (precesion score, Recall Score, F1-Score, ROC/AUC Curves and ROC AUC Score)

1. We then start visualizing our metrics by:

* Extracting each score and the hyperparameters from the result
* Extracting the value for each hyperparameter
* Creating a 2d scatter plot to visualize the scores

**Screenshots of the Code including the output**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

train\_dataset = pd.read\_csv('updated\_train.csv')

label\_encoder = LabelEncoder()

train\_dataset['Sex'] = label\_encoder.fit\_transform(train\_dataset['Sex'])

Here we imported pandas and label encoder in order to transform our ‘Sex’ column into numerical values using label encoding

import numpy as np

# hyperparameter optimization

from sklearn.model\_selection import GridSearchCV

from fast\_ml.model\_development import train\_valid\_test\_split

X\_train, y\_train, X\_valid, y\_valid, X\_test, y\_test = train\_valid\_test\_split(train\_dataset, *target* = 'Survived',

*train\_size*=0.7, *valid\_size*=0.15, *test\_size*=0.15)

Then we import numpy, GridSearchCV and train\_valid\_tes\_split, to be used in the process of training of the model and fine tuning the hyperparameters to get the best hyperparameter values for the model

from itertools import product

def generate\_prior\_combinations(*num\_combinations*=10):

    """

    Generate combinations of prior probabilities for a binary classification problem.

    Parameters:

    - num\_combinations (int): Number of combinations to generate.

    Returns:

    - List of tuples, where each tuple represents a combination of prior probabilities.

    """

    # Ensure that num\_combinations is a positive integer

    num\_combinations = max(1, *int*(num\_combinations))

    # Generate all possible combinations of prior probabilities

    prior\_combinations = *list*(product(np.linspace(0, 1, num\_combinations), *repeat*=2))

    # Filter out combinations where the sum is not 1 (valid probabilities)

    prior\_combinations = [prior for prior in prior\_combinations if sum(prior) == 1]

    return prior\_combinations

Here we define a function to generate combinations of the bayes classifier hyperparameters (priori probabilities)

from sklearn.naive\_bayes import GaussianNB

# Training the model on the training dataset

gnb = GaussianNB()

gnb.fit(X\_train,y\_train)

y\_pred = gnb.predict(X\_test)

# Fine tuning the model on the validation dataset

grid\_params = {

    "var\_smoothing": np.logspace(0,-9,*num*=100),

    "priors": generate\_prior\_combinations(30)

    }

gs = GridSearchCV(gnb, *param\_grid*=grid\_params, *scoring*=["accuracy","f1","precision","recall","roc\_auc"],*cv*=10,*n\_jobs*=-1,*refit*="f1" )

gs.fit(X\_valid, y\_valid)

y\_pred = gs.predict(X\_test)

The above function (GridSearchCV) gets the best hyperparameters based on the scoring metric f1-score. And below is the best score achieved by the Bayes classifier

A screenshot of a computer

Description automatically generated

Now we will show the results for each of the four metrics (accuracy, f1-score,recall,precision)

A screen shot of a computer

Description automatically generated

Accuracy

A screen shot of a computer program

Description automatically generated

Precision

A screen shot of a computer program

Description automatically generated

Recall

A screen shot of a computer program

Description automatically generated

F1-Score

A screen shot of a graph

Description automatically generated

ROC/AUC Curve

**Visualization of Accuracy changes**

1. F1-Score

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

# Extract the F1 scores, prior probabilities, and hyperparameters from the cv\_results\_

f1\_scores = gs.cv\_results\_['mean\_test\_f1']

params = gs.cv\_results\_['params']

# Extract the values for each hyperparameter and prior probabilities

prior\_class\_0\_values = [param['priors'][0] for param in params]

prior\_class\_1\_values = [param['priors'][1] for param in params]

# Create a 3D scatter plot to visualize the F1 scores

fig = plt.figure(*figsize*=(12, 8))

ax = fig.add\_subplot(111, *projection*='3d')

# Scatter plot for F1 scores as a function of prior probabilities

sc = ax.scatter(prior\_class\_0\_values, prior\_class\_1\_values, f1\_scores, *c*=f1\_scores, *cmap*='viridis', *marker*='o')

ax.set\_xlabel('Prior Probability for Class 0')

ax.set\_ylabel('Prior Probability for Class 1')

ax.set\_zlabel('Mean Test F1 Score')

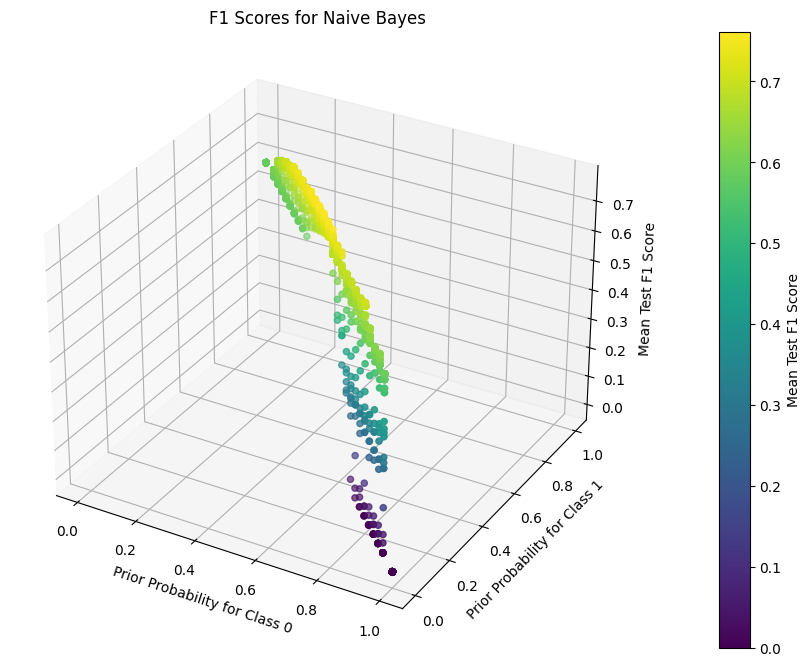
plt.title('F1 Scores for Naive Bayes')

# Add a colorbar to the right of the plot

cbar = fig.colorbar(sc, *ax*=ax, *pad*=0.1, *aspect*=20)

cbar.set\_label('Mean Test F1 Score')

plt.show()



1. Accuracy:

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

# Extract the F1 scores, prior probabilities, and hyperparameters from the cv\_results\_

f1\_scores = gs.cv\_results\_['mean\_test\_accuracy']

params = gs.cv\_results\_['params']

# Extract the values for each hyperparameter and prior probabilities

prior\_class\_0\_values = [param['priors'][0] for param in params]

prior\_class\_1\_values = [param['priors'][1] for param in params]

# Create a 3D scatter plot to visualize the F1 scores

fig = plt.figure(*figsize*=(12, 8))

ax = fig.add\_subplot(111, *projection*='3d')

# Scatter plot for F1 scores as a function of prior probabilities

sc = ax.scatter(prior\_class\_0\_values, prior\_class\_1\_values, f1\_scores, *c*=f1\_scores, *cmap*='viridis', *marker*='o')

ax.set\_xlabel('Prior Probability for Class 0')

ax.set\_ylabel('Prior Probability for Class 1')

ax.set\_zlabel('Mean Test Accuracy Score')

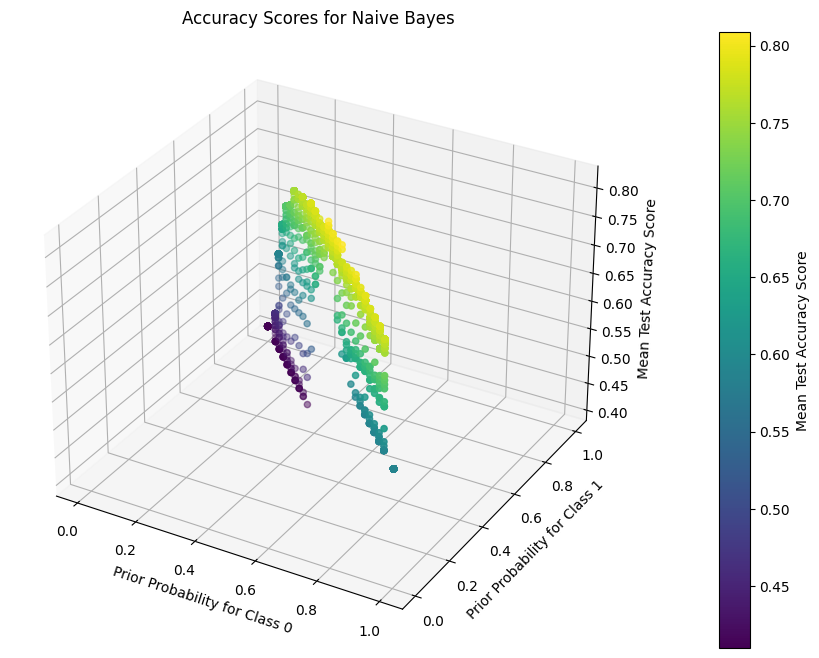
plt.title('Accuracy Scores for Naive Bayes')

# Add a colorbar to the right of the plot

cbar = fig.colorbar(sc, *ax*=ax, *pad*=0.1, *aspect*=20)

cbar.set\_label('Mean Test Accuracy Score')

plt.show()



1. Precision:

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

# Extract the F1 scores, prior probabilities, and hyperparameters from the cv\_results\_

f1\_scores = gs.cv\_results\_['mean\_test\_precision']

params = gs.cv\_results\_['params']

# Extract the values for each hyperparameter and prior probabilities

prior\_class\_0\_values = [param['priors'][0] for param in params]

prior\_class\_1\_values = [param['priors'][1] for param in params]

# Create a 3D scatter plot to visualize the F1 scores

fig = plt.figure(*figsize*=(12, 8))

ax = fig.add\_subplot(111, *projection*='3d')

# Scatter plot for F1 scores as a function of prior probabilities

sc = ax.scatter(prior\_class\_0\_values, prior\_class\_1\_values, f1\_scores, *c*=f1\_scores, *cmap*='viridis', *marker*='o')

ax.set\_xlabel('Prior Probability for Class 0')

ax.set\_ylabel('Prior Probability for Class 1')

ax.set\_zlabel('Mean Test Precision Score')

plt.title('Precision Scores for Naive Bayes')

# Add a colorbar to the right of the plot

cbar = fig.colorbar(sc, *ax*=ax, *pad*=0.1, *aspect*=20)

cbar.set\_label('Mean Test Precision Score')

plt.show()

A graph of a graph showing a graph of a graph

Description automatically generated with medium confidence

1. Recall:

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

# Extract the F1 scores, prior probabilities, and hyperparameters from the cv\_results\_

f1\_scores = gs.cv\_results\_['mean\_test\_recall']

params = gs.cv\_results\_['params']

# Extract the values for each hyperparameter and prior probabilities

prior\_class\_0\_values = [param['priors'][0] for param in params]

prior\_class\_1\_values = [param['priors'][1] for param in params]

# Create a 3D scatter plot to visualize the F1 scores

fig = plt.figure(*figsize*=(12, 8))

ax = fig.add\_subplot(111, *projection*='3d')

# Scatter plot for F1 scores as a function of prior probabilities

sc = ax.scatter(prior\_class\_0\_values, prior\_class\_1\_values, f1\_scores, *c*=f1\_scores, *cmap*='viridis', *marker*='o')

ax.set\_xlabel('Prior Probability for Class 0')

ax.set\_ylabel('Prior Probability for Class 1')

ax.set\_zlabel('Mean Test Recall Score')

plt.title('Recall Scores for Naive Bayes')

# Add a colorbar to the right of the plot

cbar = fig.colorbar(sc, *ax*=ax, *pad*=0.1, *aspect*=20)

cbar.set\_label('Mean Test Recall Score')

plt.show()

A graph of a graph with a rainbow line

Description automatically generated with medium confidence

**Reasons of using final values of hyperparameters**

According to the GridSearchCV algorithm the best score produced is

0.7616450216450217

The reason for choosing this value as the best hypermeter is according to the highest F1-Score value produced by this hyperparameter value.