Intro to Machine learning

Project Phase 1

Team 8

AbdulRaouf Monir Kamal Mahmoud	19P4442
Abdallah Amin Fahmy	20p5880
Michael Joseph Adeeb	1901075

Included Classifiers:

- SVM
- KNN
- Bayes Classifier

Dataset Description

The titanic dataset is a dataset that contains information of some passengers on the titanic, a ship that sank in 1912 after hitting an iceberg. The dataset can be used to predict whether a passenger survived or not based on variables such as age, class, gender, etc. The description of each column are as follows:

Pclass: Represents the socio-economic class of each passenger on the ship
Survived: Represents whether the passenger has survived or not (1 for survived and 0 for not survived)
Name: The name of the passenger
Sex
Age
SibSp: The number of siblings and spouses of the passenger
Parch: The number of parents and children of the passenger
Ticket: The ticket id
Fare: The amount paid for the ticket
Embarked: The port from which the passenger boarded the titanic
Cabin: The cabin occupied by the passenger

Dataset preprocessing

```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn-whitegrid')

import pandas as pd
```

Here we imported some visualization libraries that will be used later on, and imported pandas to get the csv as dictionary in python.

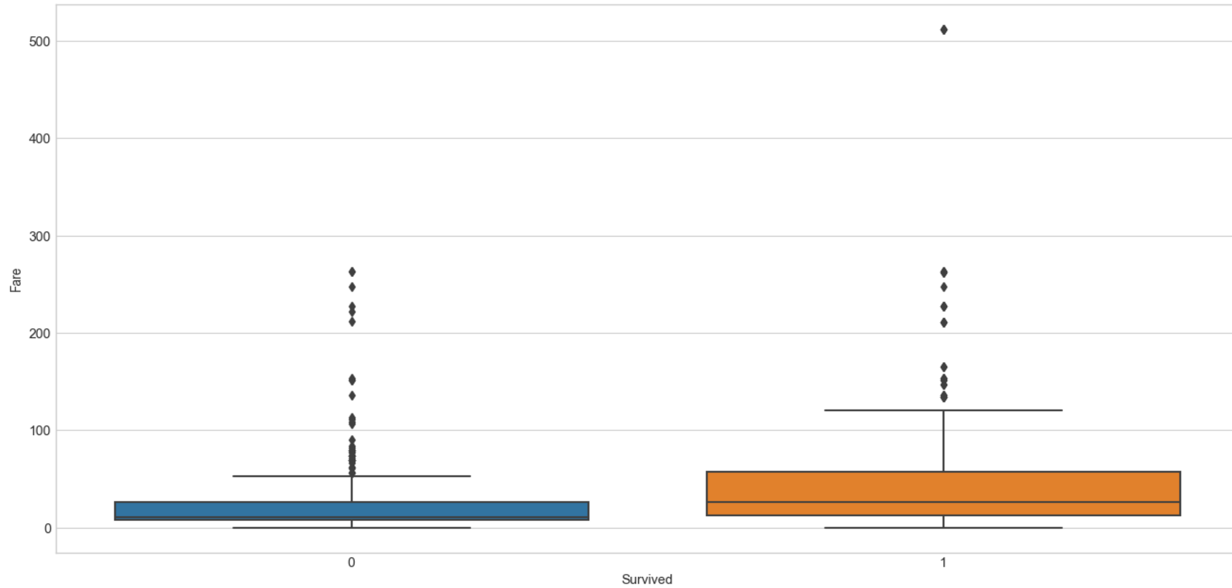
```
train_data = pd.read_csv("./titanic/train.csv")
train_data.info()
```

Then used pd.read_csv to read the csv file and store it in train_data variable, and then we invoked train_data.info to gain some insights about the data (the number of null rows)

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
data = pd.concat([train_data['Survived'], train_data['Fare']], axis=1)
f, ax = plt.subplots(figsize=(16, 8))
fig = sns.boxplot(x=train_data['Survived'], y=train_data['Fare'], data=data)
```

Then we started our data cleaning by gathering out numerical features and searching for the outliers, and we found only one extreme at a fare that is almost equal to 500, as shown below, note that some values we did not consider as outliers as there were too many rows denoted by the box plot, and removing them would reduce out training samples, so we decided to remove only the most extreme point as it was only one




```
train_data[train_data['Fare'] > 500]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
258	259	1	1	Ward, Miss. Anna	female	35.0	0	0	PC 17755	512.3292	NaN	C
679	680	1	1	Cardeza, Mr. Thomas Drake Martinez	male	36.0	0	1	PC 17755	512.3292	B51 B53 B55	C
737	738	1	1	Lesurer, Mr. Gustave J	male	35.0	0	0	PC 17755	512.3292	B101	C

Here we get all passengers who paid fares greater than 500, we get three rows with fares greater than 500, and we replace those fares with the mean of those fares (calculated without the outliers) as shown below

```
# Calculate the mean excluding values greater than 500
mean_without_outliers_train = train_data.loc[train_data['Fare'] <= 500, 'Fare'].mean()

# Replace values greater than 500 with the calculated mean
train_data.loc[train_data['Fare'] > 500, 'Fare'] = mean_without_outliers_train

# Just to make sure that outliers for that column has been removed
train_data[train_data['Fare'] > 500]
```

Now we will check if there is still null rows

```
train_data.isna().sum()
```

[388] ✓ 0.0s

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

As we see there are 177 rows in age column are empty, so we are going to replace the NaN values with the mean age values, and as for the cabin we are going to drop this column, and also drop any other irrelevant feature

```
mean_age_train = int(train_data['Age'].mean())

print(f"Mean Age Train: {mean_age_train}")
```

```
import numpy as np
```

✓ 0.0s

```
train_data['Age'] = train_data['Age'].replace(np.nan, mean_age_train)
```

✓ 0.0s

Do the same with the Fare column

```python
train_data['Fare'] = train_data['Fare'].replace(np.nan, mean_without_outliers_train)
```

892] ✓ 0.0s Python

Data Cleaning (Categorical features)

```python
# Since most of the Cabin column values are filled with NA's, as reported by train_data.info()
# We are going to drop the Cabin column, and remove any other irrelevant feature

train_data = train_data.drop(['PassengerId','Name','Ticket','Embarked','Cabin'], axis=1)
train_data.head()
```

893] ✓ 0.0s Python

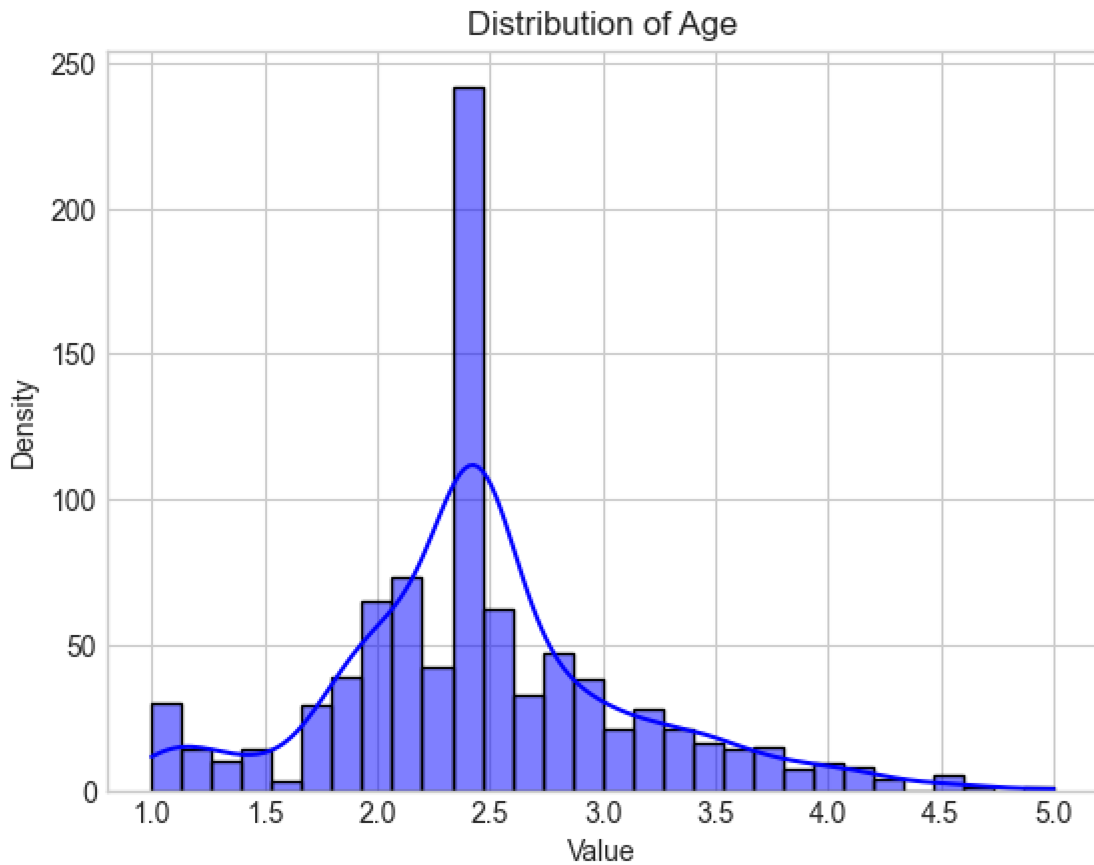
Now we are going to use MinMaxScaler to normalize our columns to be in the range [1,5] since there are many columns with varying ranges so we need to normalize them first

```python
from sklearn.preprocessing import MinMaxScaler

numerical_features = train_data[['Age','Fare','SibSp','Parch','Pclass']]
scaler = MinMaxScaler(feature_range=(1,5))
numerical_features = pd.DataFrame(scaler.fit_transform(numerical_features), columns=numerical_features.columns)

train_data = pd.concat([numerical_features,train_data[['Sex','Survived']]],axis='columns')

train_data
```

	Age	Fare	SibSp	Parch	Pclass	Sex	Survived
0	2.084695	1.110266	1.5	1.000000	5.0	male	0
1	2.888917	2.084157	1.5	1.000000	1.0	female	1
2	2.285750	1.120532	1.0	1.000000	5.0	female	1
3	2.738125	1.807605	1.5	1.000000	1.0	female	1
4	2.738125	1.122433	1.0	1.000000	5.0	male	0
...	...	...	...	...	...	...	...
886	2.336014	1.197719	1.0	1.000000	3.0	male	0
887	1.933903	1.456274	1.0	1.000000	1.0	female	1
888	2.436542	1.356654	1.5	2.333333	5.0	female	0
889	2.285750	1.456274	1.0	1.000000	1.0	male	1
890	2.587334	1.117871	1.0	1.000000	5.0	male	0

Now we will check the distribution of the data, since some models may work best (or not even affected) when the data is normally distributed.

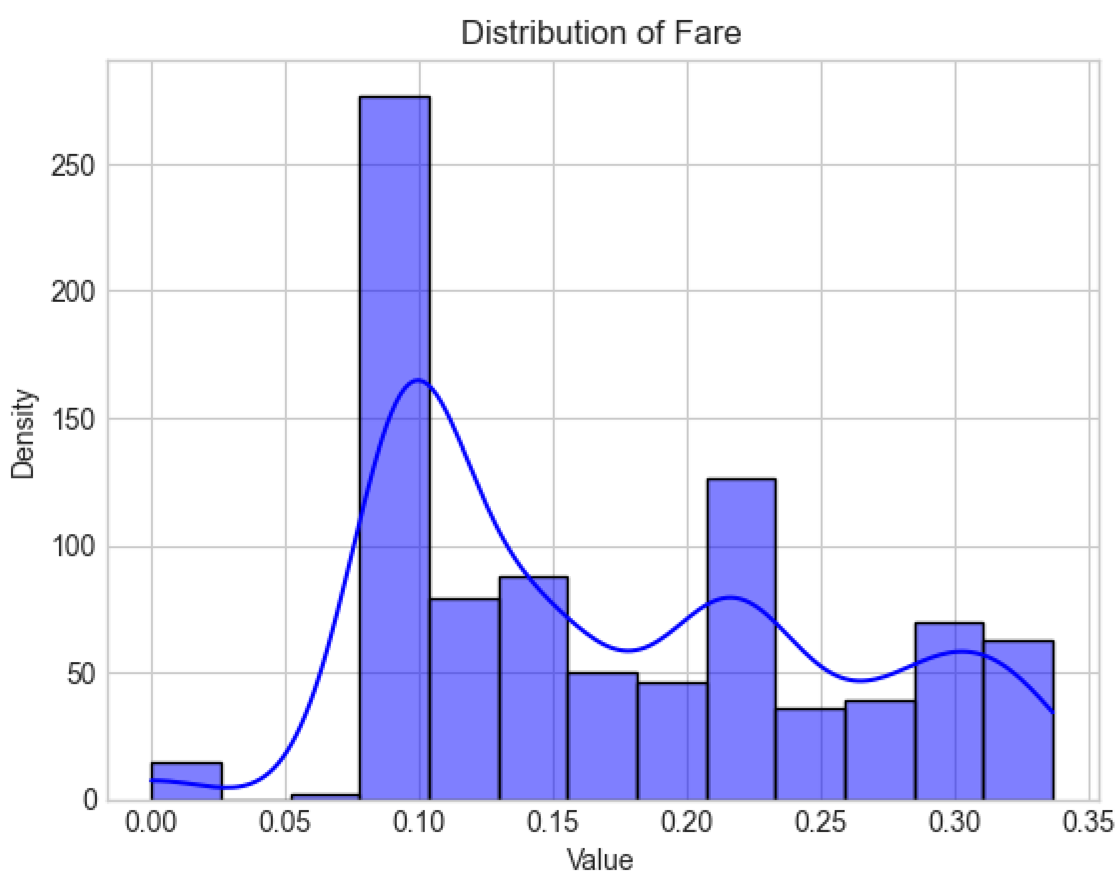
```
# Create a sample DataFrame
data = train_data['Age']
df = pd.DataFrame(data)

# Choose the column for which you want to plot the distribution
column_to_plot = 'Age'

# Create a distribution plot with KDE using Seaborn
sns.histplot(train_data[column_to_plot], kde=True, color='blue', edgecolor='black')

# Add labels and a title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title(f'Distribution of {column_to_plot}')

# Display the plot
plt.show()
```



The Fare was not normally distributed, so we did apply box cox transformation to try to somehow make the data normally distributed, this the result after applying box cox transformation.

```
transformed_feature, lambda_value = boxcox(train_data['Fare'])
transformed_feature
```

```python
# Create a sample DataFrame
data = train_data['Fare']
df = pd.DataFrame(data)

# Choose the column for which you want to plot the distribution
column_to_plot = 'Fare'

# Create a distribution plot with KDE using Seaborn
sns.histplot(train_data[column_to_plot], kde=True, color='blue', edgecolor='black')

# Add labels and a title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title(f'Distribution of {column_to_plot}')

# Display the plot
plt.show()
```



Finally, we applied PCA to the numerical features (Parch, Age, Fare, SibSp) to reduce the dimensions while retaining the variance as much as possible (95% variance retained as shown below)

```
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)
numerical_features = train_data[['Age','Fare','SibSp','Parch']]

numerical_features_pca = pca.fit_transform(numerical_features)
train_data = train_data.drop(numerical_features.columns, axis='columns')
pca_features = pd.DataFrame(data=numerical_features_pca, columns=['PCA1','PCA2','PCA3'])

train_data = pd.concat([train_data,pca_features],axis='columns')
train_data
```

```
train_data.to_csv('updated_train.csv',index=False)
```

We then save the updated data in an updated_train.csv

SVM Classifier

Steps on how to run project.

use sklearn.svm to classifier the data with fit and score function.

And predict the value in y_pred and getting the score before search for optimal hyperparameters and after this use the grid search to find it for (C, Gamma, Kernal) and get the score after this with the optimal hyperparameters.

calculate precesion score and recall score and F1-score using sklearn.metrics to calculate it.

Plot ROC/AUC curves using matplotlip library.

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 numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

[3]

```
train_dataset = pd.read_csv('updated_train.csv')
label_encoder = LabelEncoder()
train_dataset['Sex'] = label_encoder.fit_transform(train_dataset['Sex'])
```

[4]

```
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)
```

[5]

```
classifier = SVC()


classifier.fit(X_train, y_train)


score = classifier.score(X_test, y_test)

print(score)
```

[6]

```
0.8134328358208955
```

```
# Training the model on the training dataset
classifier = SVC(C=1,gamma=0.01,kernel='rbf')

classifier.fit(X_train, y_train)

score = classifier.score(X_test, y_test)

y_pred = classifier.predict(X_test)
print(f"SVM Score before hyperparameter tuning: {accuracy_score(y_test,y_pred)}")

param_grid = {'C':[0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1],'kernel': ['rbf', 'poly', 'sigmoid','linear'],
             }

# gs = GridSearchCV(gnb, grid_params, verbose = 1, cv=20, n_jobs = -1)
gs = GridSearchCV(classifier, param_grid=param_grid,scoring=["accuracy","f1","precision","recall","roc_auc"],cv=10,n_jobs=-1,refit="accuracy")
g_res = gs.fit(X_valid, y_valid)
print(f"SVM after hyperparameter tuning (on validation): {g_res.best_score_}")

y_pred = g_res.predict(X_test)
print(f"SVM Score after hyperparameter tuning (on testing): {accuracy_score(y_test,y_pred)}")
```

```
SVM Score before hyperparameter tuning: 0.8134328358208955
SVM after hyperparameter tuning (on validation): 0.8428571428571427
SVM Score after hyperparameter tuning (on testing): 0.8059701492537313
```

```python
# get the hyperparameters with the best score
gs.best_params_
```
[7]

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

```python
# precesion score
from sklearn.metrics import precision_score
print(precision_score(y_test, y_pred, average='macro'))
```
[8]

```
0.7933832709113608
```

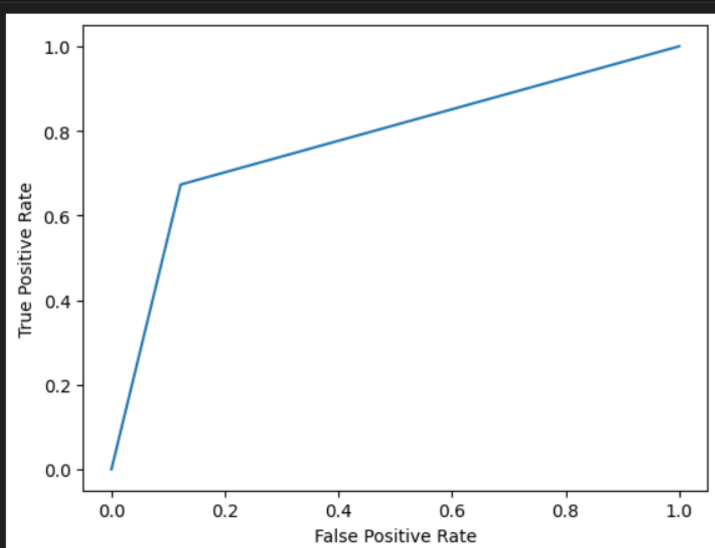
```python
# Recall Score
from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred, average='macro'))
```
[9]

```
0.7755628517823641
```

```python
# F1-Score
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred, average='macro'))
```
[10]

```
0.7818773738469886
```

```python
# ROC/AUC Curves
# fpr, tpr, thresholds = roc_curve(y_test, y_pred, pos_label=2)
plot_roc_curve(y_test, y_pred)
plt.show()
print(f'Bayes Classifier AUC score: {roc_auc_score(y_test, y_pred)}')
```
[12]



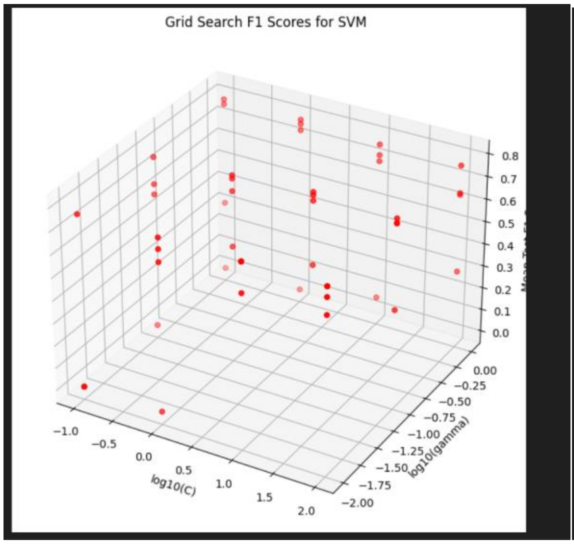

```
Bayes Classifier AUC score: 0.7755628517823641
```

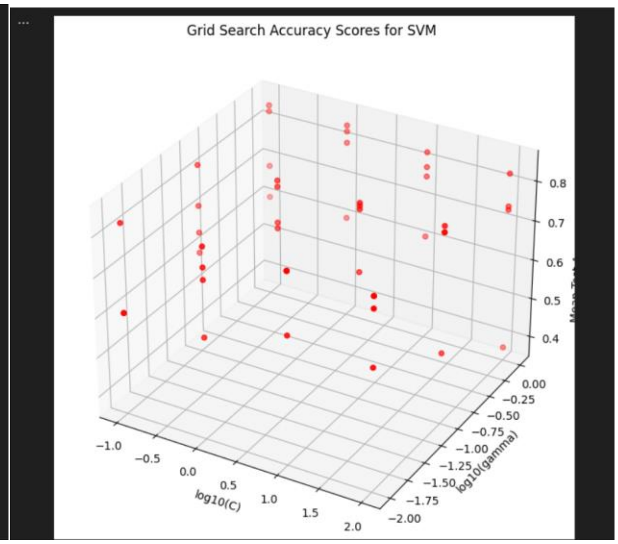
```python
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```
[13]

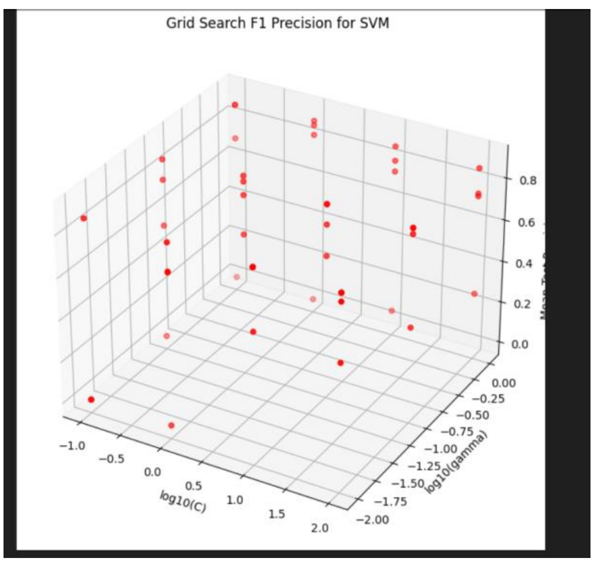
```
Accuracy: 0.7985074626865671
```

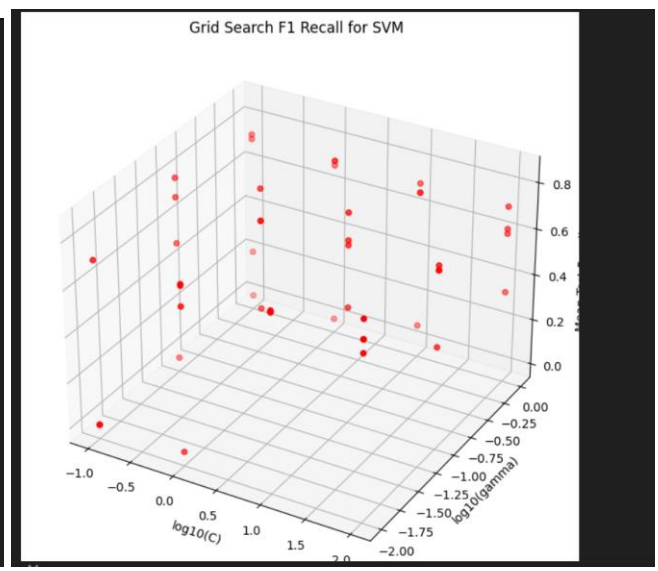
Visualization of Accuracy change

For SVM









Reasons of using final values of hyperparameters

According to the GridSearchCV algorithm the best score produced is

`0.8428571428571427`

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

```python
# Training the model on the training dataset
classifier = SVC(C=1,gamma=0.01,kernel='rbf')

classifier.fit(X_train, y_train)

score = classifier.score(X_test, y_test)

y_pred = classifier.predict(X_test)
print(f"SVM Score before hyperparameter tuning: {accuracy_score(y_test,y_pred)}")

param_grid = {'C':[0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1],'kernel': ['rbf', 'poly', 'sigmoid','linear'],
              }

# gs = GridSearchCV(gnb, grid_params, verbose = 1, cv=20, n_jobs = -1)
gs = GridSearchCV(classifier, param_grid=param_grid,scoring=["accuracy","f1","precision","recall","roc_auc"],cv=10,n_jobs=-1,refit="accuracy")
g_res = gs.fit(X_valid, y_valid)
print(f"SVM after hyperparameter tuning (on validation): {g_res.best_score_}")

y_pred = g_res.predict(X_test)
print(f"SVM Score after hyperparameter tuning (on testing): {accuracy_score(y_test,y_pred)}")
```

```
SVM Score before hyperparameter tuning: 0.8134328358208955
SVM after hyperparameter tuning (on validation): 0.8428571428571427
SVM Score after hyperparameter tuning (on testing): 0.8059701492537313
```

```python
# get the hyperparameters with the best score
gs.best_params_
```

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

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

2) Import GridSearchCV Then import train_valid_test_split and use it to divide the data to sizes according to our needs.
3) We define our KNN Classifier then we start fitting the data and round the numbers
4) Using GridSearchCV we brute search all the possible value of the hyperparameters and get the best hyperparameter value
5) Then we output the hyperparameters with the best score
6) 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)
7) 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

```
In [1]: %pip install fast-ml
```

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.neighbors import KNeighborsClassifier
        import seaborn as sns
        from sklearn.preprocessing import LabelEncoder
```

```
In [3]: train_dataset = pd.read_csv('updated_train.csv')
        label_encoder = LabelEncoder()
        train_dataset['Sex'] = label_encoder.fit_transform(train_dataset['Sex'])
```

```
In [4]: # hyperparameter optemization
        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)
```

```
In [5]: #defining and Fitting KNN model
        classifier = KNeighborsClassifier()
        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_test)
        rounded_KNN = round(classifier.score(X_train, y_train)*100, 2)
        print(rounded_KNN)

        85.07
```

```
In [7]: #Tuning the hyperparameters
        grid_params = { 'n_neighbors' :[i for i in range (1,624,2)],
                       'weights' : ['uniform','distance'],
                       'metric' : ['minkowski','euclidean','manhattan']}
        gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=20, n_jobs = -1 ,scoring=["accuracy","f1","precision","rec

        g_res = gs.fit(X_valid, y_valid)
        g_res.best_score_

        Fitting 20 folds for each of 1872 candidates, totalling 37440 fits

Out[7]: 0.8130952380952381
```

```
In [8]: # get the hyperparameters with the best score
        g_res.best_params_

Out[8]: {'metric': 'manhattan', 'n_neighbors': 51, 'weights': 'distance'}
```

```
In [9]: y_pred = g_res.predict(X_test)
```

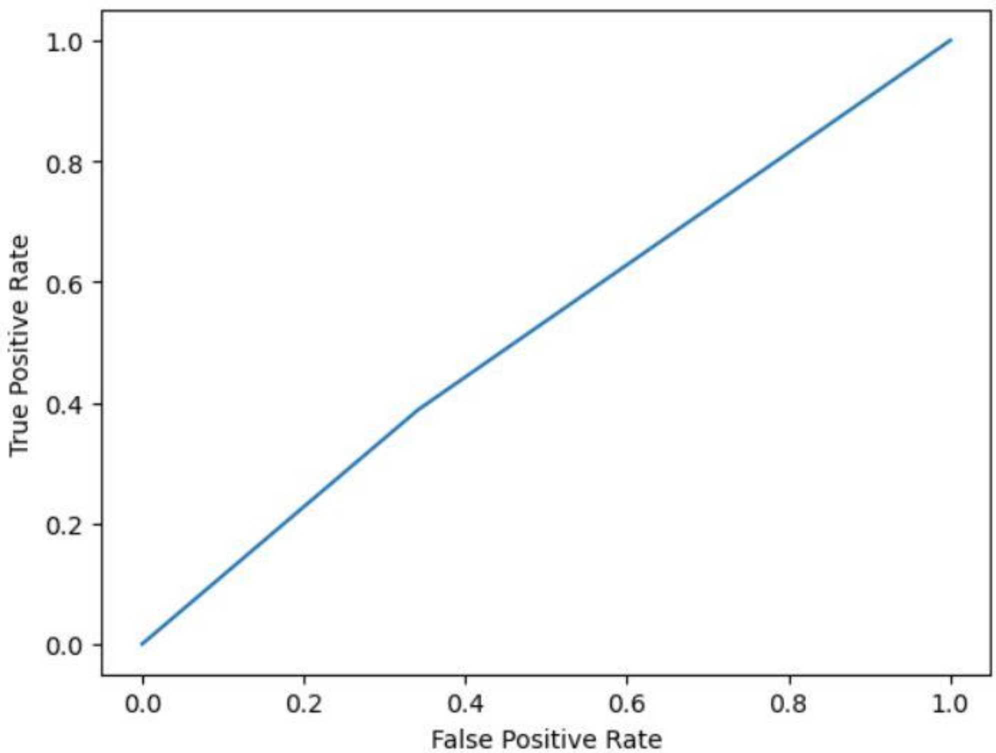
```
In [10]: # precesion score
         from sklearn.metrics import precision_score
         print("Precesion Score:",precision_score(y_test, y_pred, average='macro'))

         Precesion Score: 0.7467595989239423
```

```
In [11]: # Recall Score
         from sklearn.metrics import recall_score
         print("Recall Score:",recall_score(y_test, y_pred, average='macro'))

         Recall Score: 0.7335648148148148
```

In [12]:
```
# F1-Score
from sklearn.metrics import f1_score
print("F1-Score:",f1_score(y_test, y_pred, average='macro'))
```

```
F1-Score: 0.7378312681567558
```

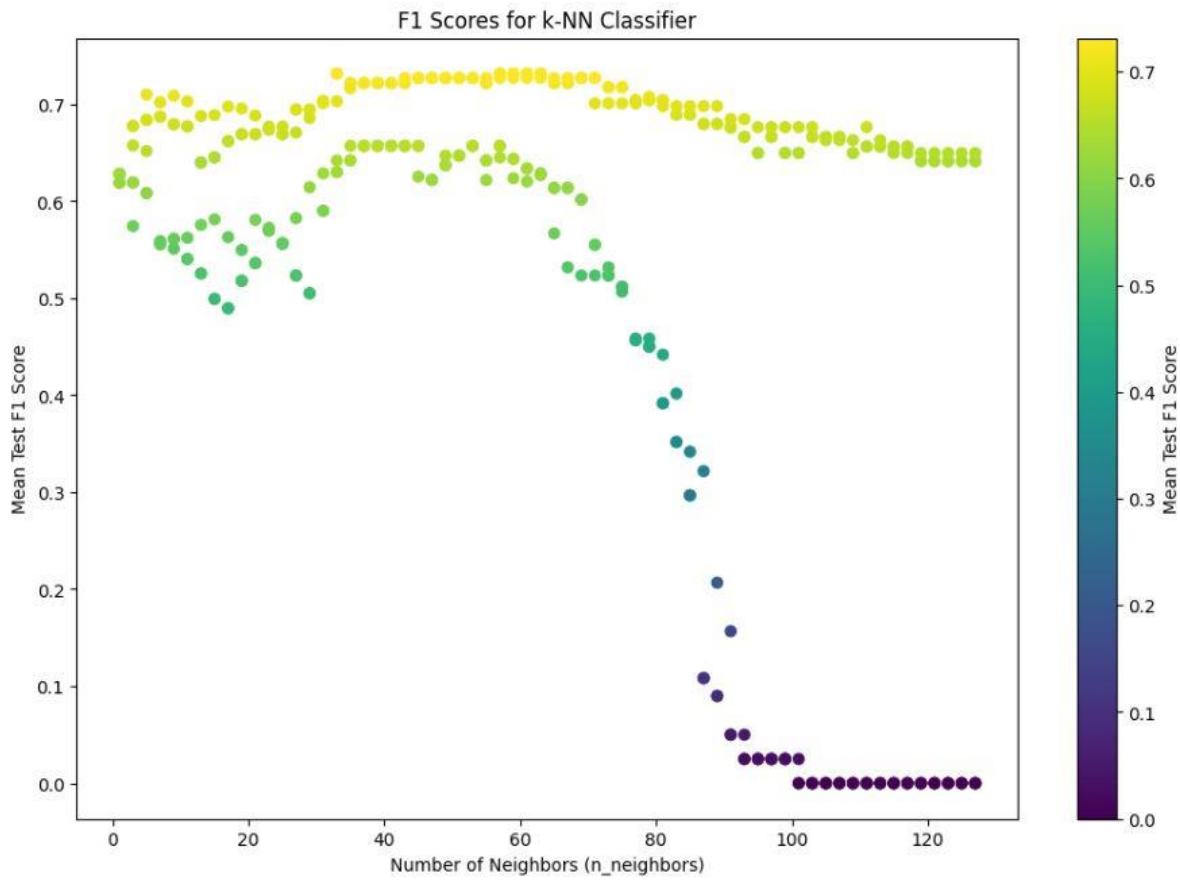
```
# ROC/AUC Curves
%matplotlib inline
from sklearn.metrics import roc_curve, roc_auc_score
def plot_roc_curve(true_y, y_prob):
    fpr, tpr, thresholds = roc_curve(true_y, y_prob)
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
plot_roc_curve(y_test,y_pred)
```



In [13]:
```
# ROC AUC Score
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(y_test, y_pred)
print("ROC-AUC-Score:", r_a_score)
```

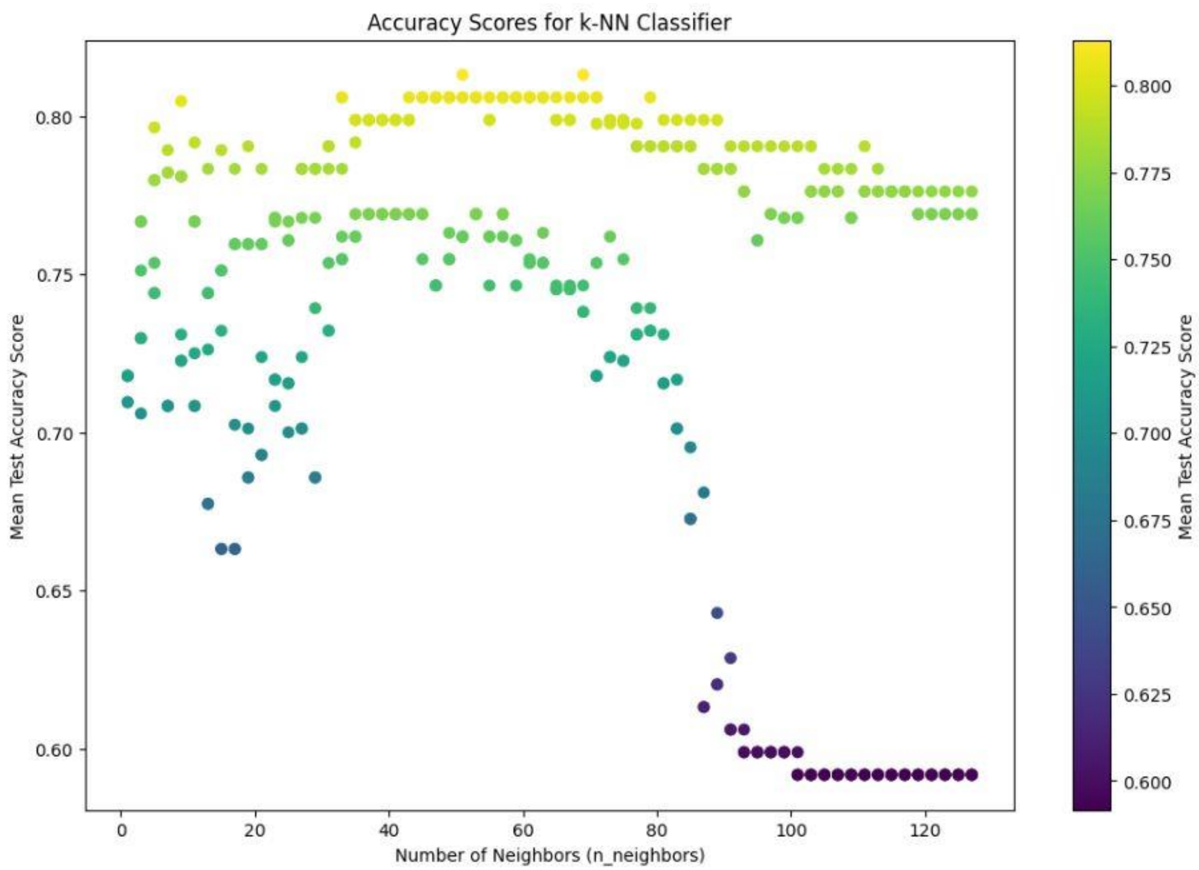
```
ROC-AUC-Score: 0.7335648148148148
```

In [15]:
```python
# Extract the F1 scores and hyperparameters from the cv_results_
f1_scores = g_res.cv_results_['mean_test_f1']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
n_neighbors_values = [param['n_neighbors'] for param in params]

# Create a 2D scatter plot to visualize the F1 scores
plt.figure(figsize=(12, 8))
plt.scatter(n_neighbors_values, f1_scores, c=f1_scores, cmap='viridis', marker='o')

plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Mean Test F1 Score')
plt.title('F1 Scores for k-NN Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test F1 Score')

plt.show()
```



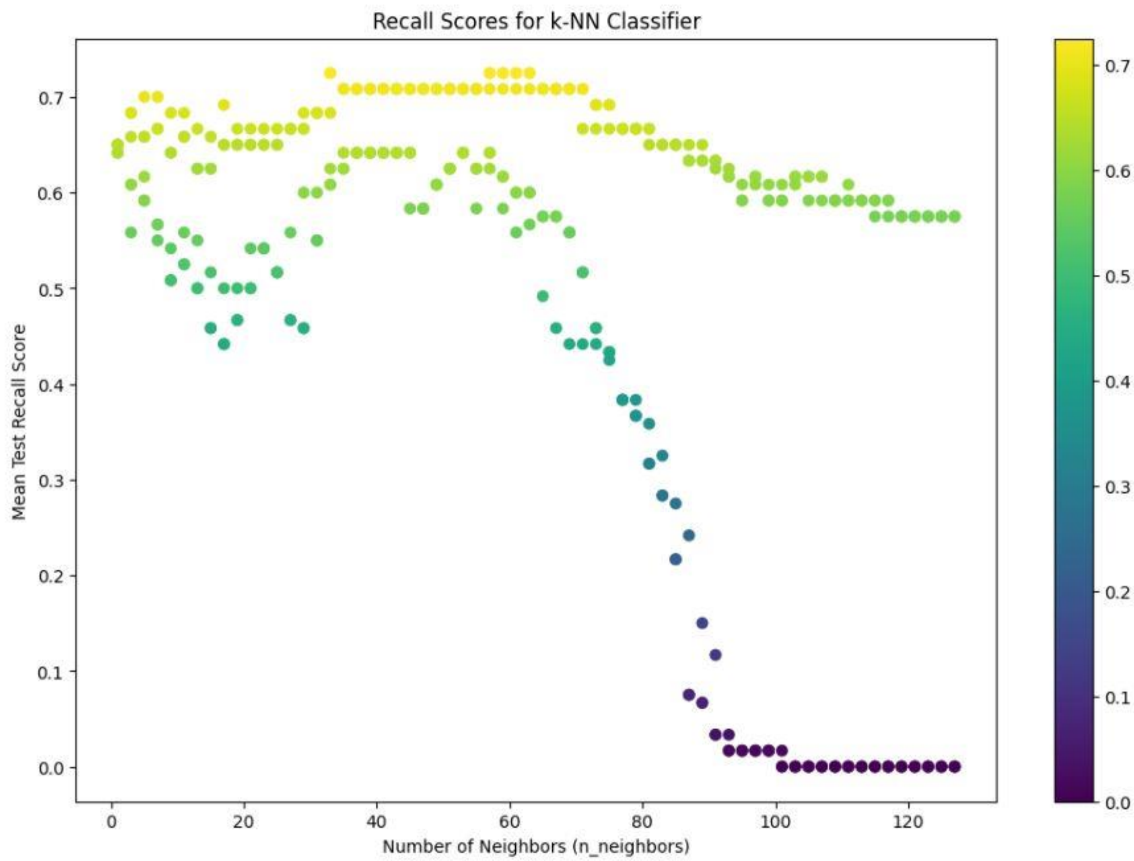
In [17]:

```
# Extract the Accuracy scores and hyperparameters from the cv_results_
f1_scores = g_res.cv_results_['mean_test_accuracy']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
n_neighbors_values = [param['n_neighbors'] for param in params]

# Create a 2D scatter plot to visualize the F1 scores
plt.figure(figsize=(12, 8))
plt.scatter(n_neighbors_values, f1_scores, c=f1_scores, cmap='viridis', marker='o')

plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Mean Test Accuracy Score')
plt.title('Accuracy Scores for k-NN Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test Accuracy Score')

plt.show()
```

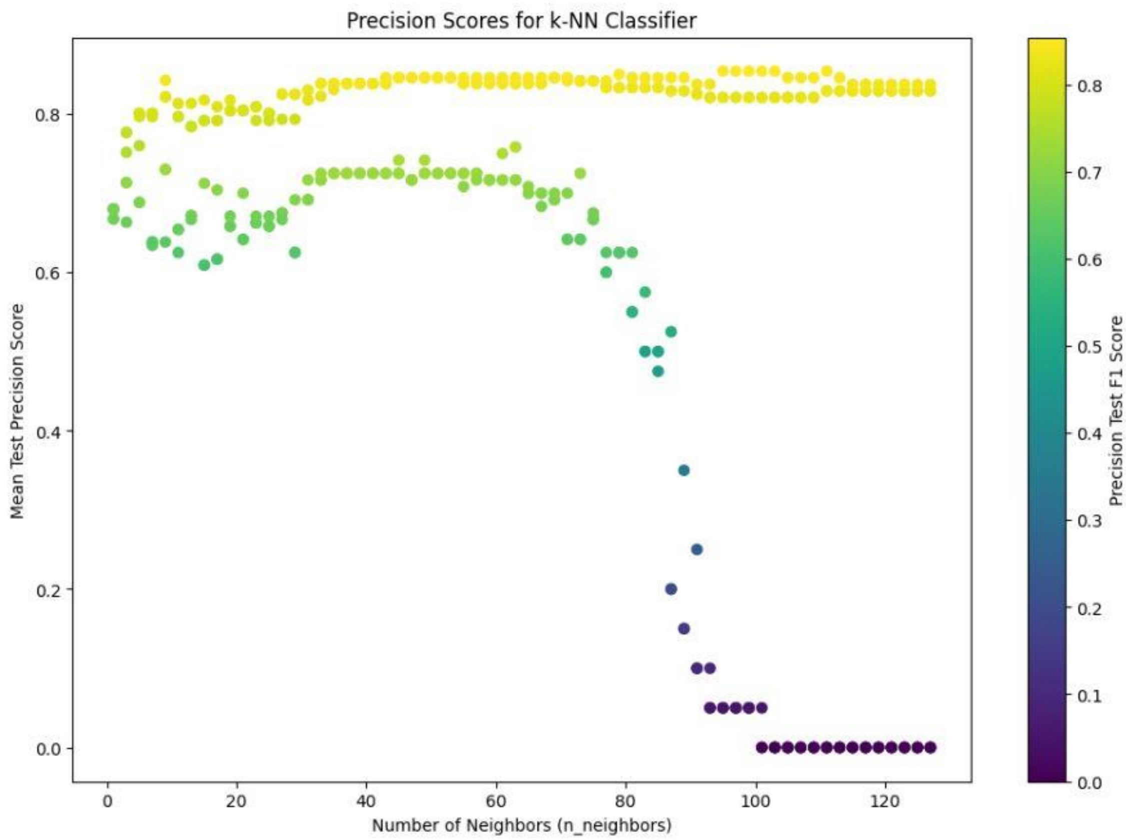


In [18]:

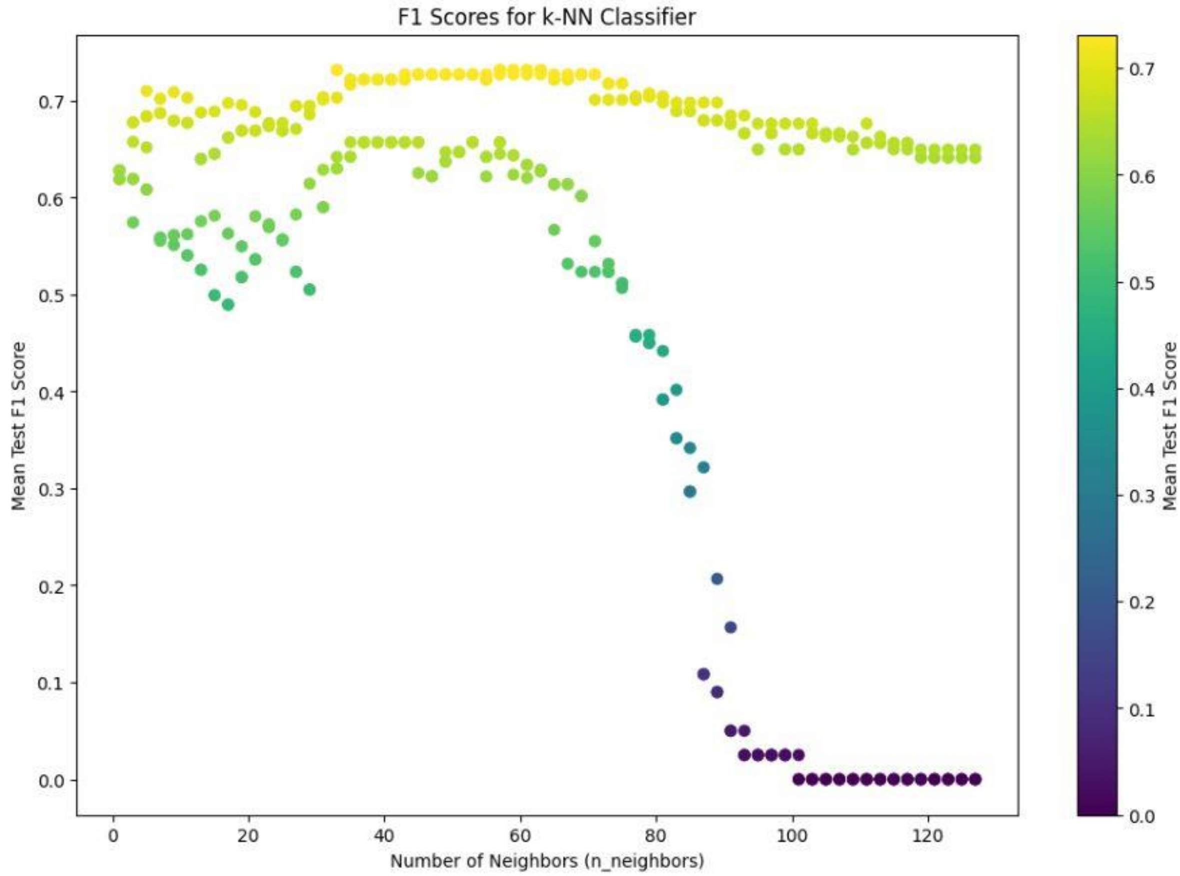
```
# Extract the Recall scores and hyperparameters from the cv_results_
f1_scores = g_res.cv_results_['mean_test_recall']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
n_neighbors_values = [param['n_neighbors'] for param in params]

# Create a 2D scatter plot to visualize the F1 scores
plt.figure(figsize=(12, 8))
plt.scatter(n_neighbors_values, f1_scores, c=f1_scores, cmap='viridis', marker='o')

plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Mean Test Recall Score')
plt.title('Recall Scores for k-NN Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Mean Test Recall Score')

plt.show()
```

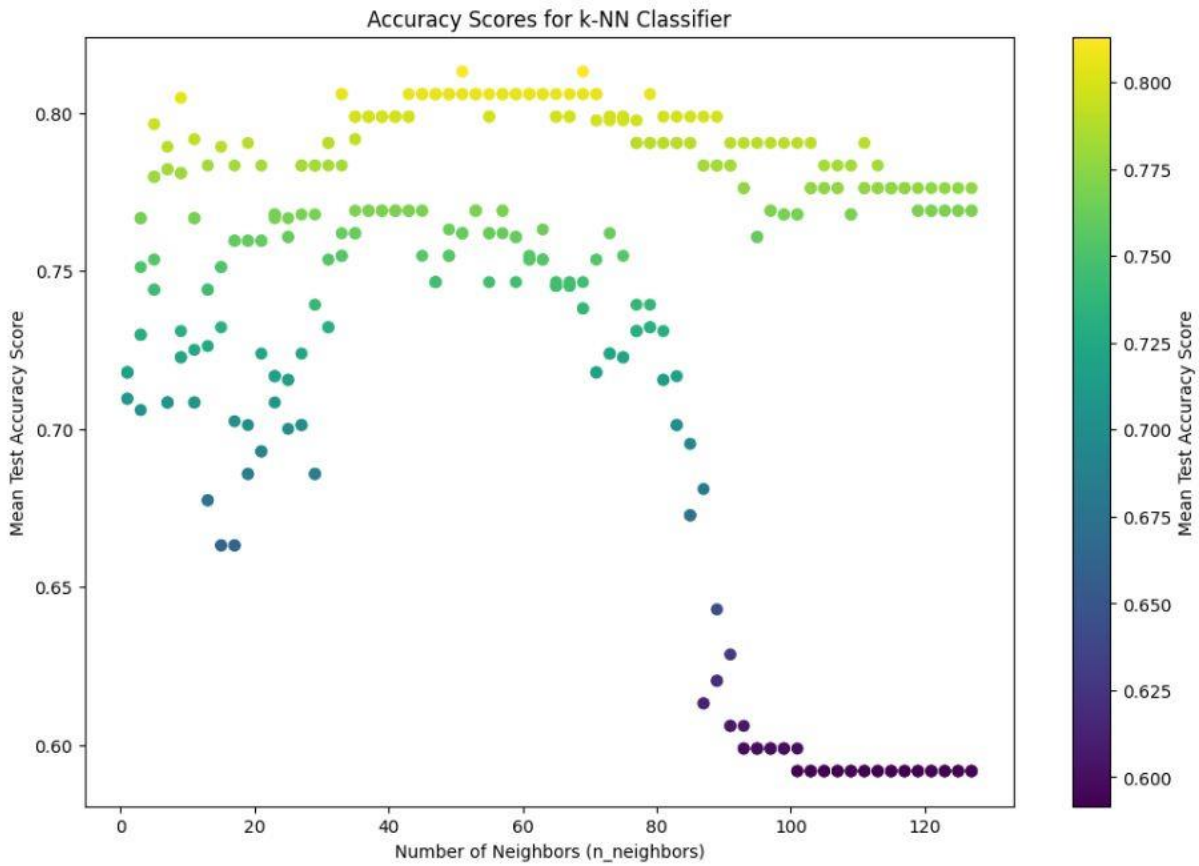


In [19]:
```python
# Extract the Precision scores and hyperparameters from the cv_results_
f1_scores = g_res.cv_results_['mean_test_precision']
params = g_res.cv_results_['params']

# Extract the values for each hyperparameter
n_neighbors_values = [param['n_neighbors'] for param in params]

# Create a 2D scatter plot to visualize the F1 scores
plt.figure(figsize=(12, 8))
plt.scatter(n_neighbors_values, f1_scores, c=f1_scores, cmap='viridis', marker='o')

plt.xlabel('Number of Neighbors (n_neighbors)')
plt.ylabel('Mean Test Precision Score')
plt.title('Precision Scores for k-NN Classifier')

# Add a colorbar to the right of the plot
cbar = plt.colorbar()
cbar.set_label('Precision Test F1 Score')

plt.show()
```

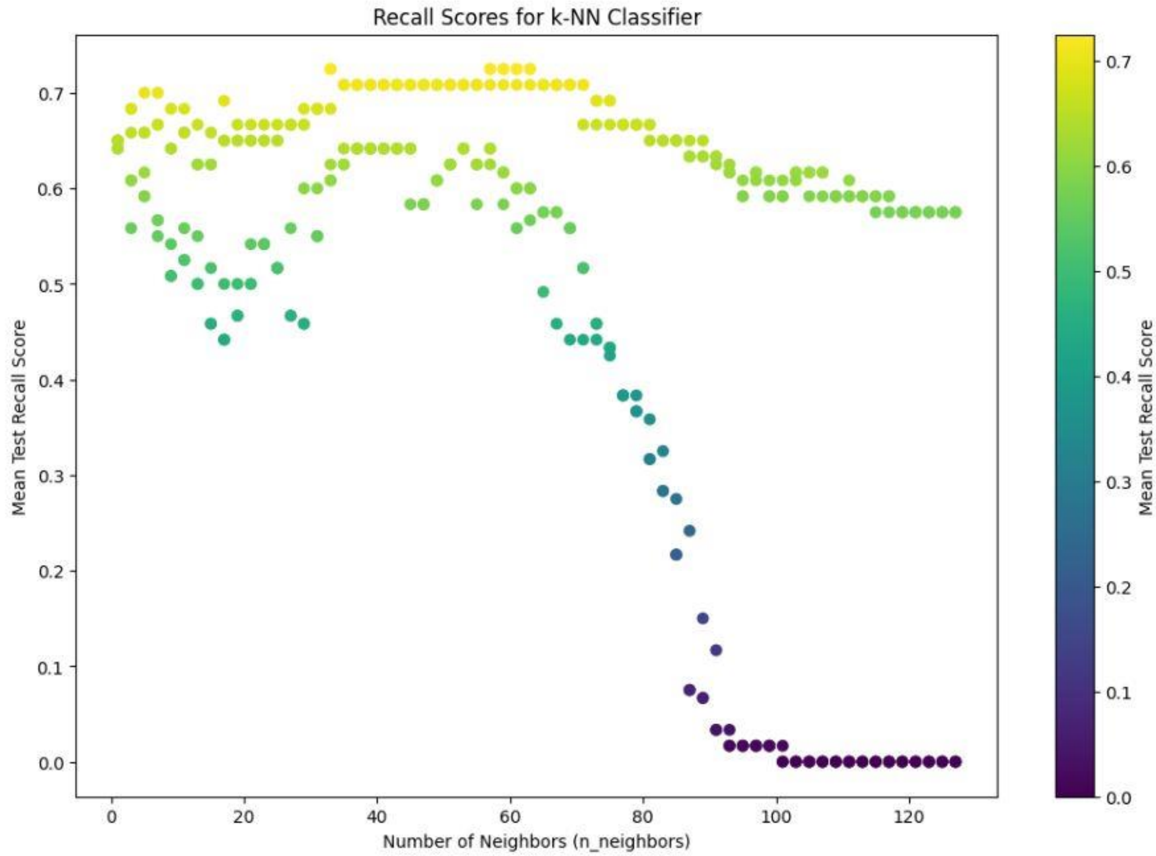


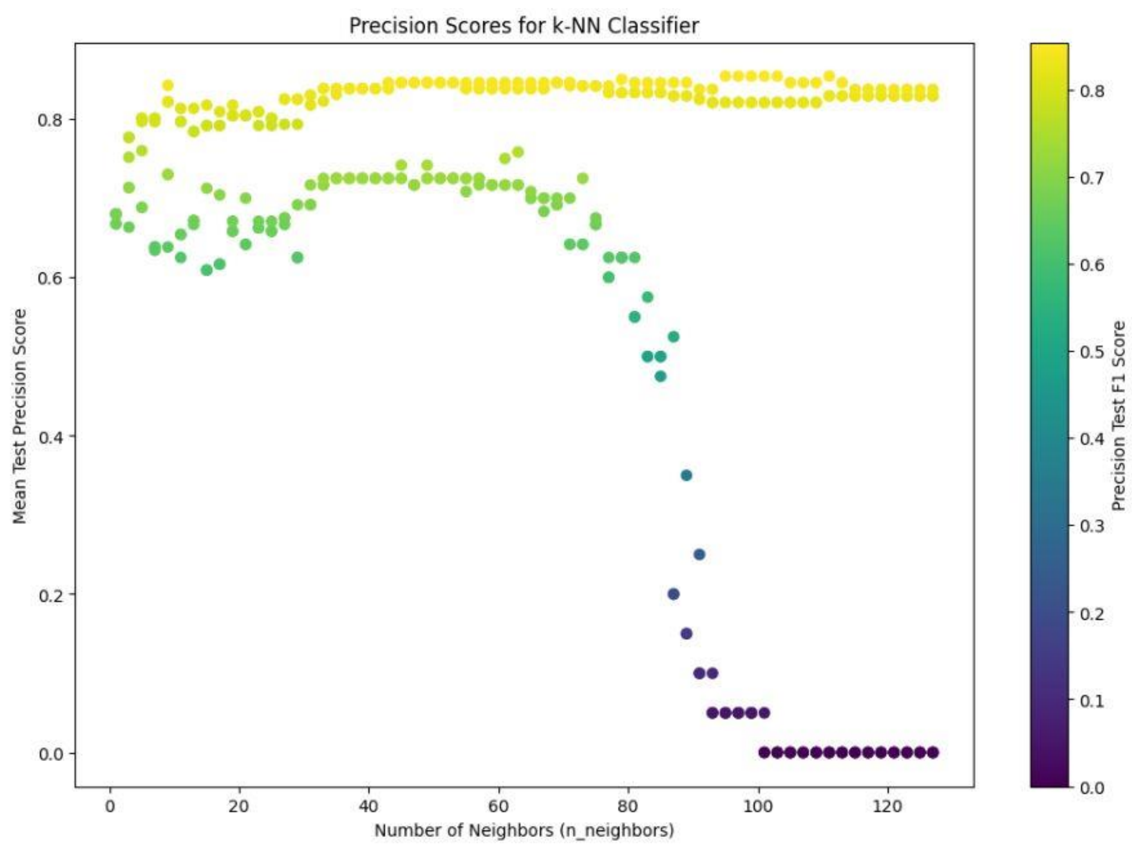
Visualization of Accuracy chang





Accuracy Scores for k-NN Classifier
Mean Test Accuracy Score
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100
120
Number of Neighbors (n_neighbors)
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
Mean Test Accuracy Score


Recall Scores for k-NN Classifier
Mean Test Recall Score
Number of Neighbors (n_neighbors)
Mean Test Recall Score


Precision Scores for k-NN Classifier
Mean Test Precision Score
Number of Neighbors (n_neighbors)
Precision Test F1 Score

Reasons of using final values of hyperparameters

According to the GridSearchCV algorithm the best score produced is

```
0.8130952380952381
```

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

Bayes 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

2) Import GridSearchCV Then import train_valid_test_split and use it to divide the data to sizes according to our needs.
3) We define our KNN Classifier then we start fitting the data and round the numbers
4) Using GridSearchCV we brute search all the possible value of the hyperparameters and get the best hyperparameter value
5) Then we output the hyperparameters with the best score
6) 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)
7) 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

```python
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

```
gs.best_score_
✓ 0.0s

0.7616450216450217
```

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

```python
from sklearn.metrics import accuracy_score
print(f"Naive Bayes Score after hyperparameter tuning (on testing): {accuracy_score(y_test,y_pred)}")
```
✓ 0.0s

```
Naive Bayes Score after hyperparameter tuning (on testing): 0.7835820895522388
```

Accuracy

```python
# precesion score
from sklearn.metrics import precision_score
print(precision_score(y_test, y_pred, average='macro'))
```
✓ 0.0s

```
0.7668926553672316
```

Precision

```python
# Recall Score
from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred, average='macro'))
```
✓ 0.0s

```
0.7982323232323232
```

Recall

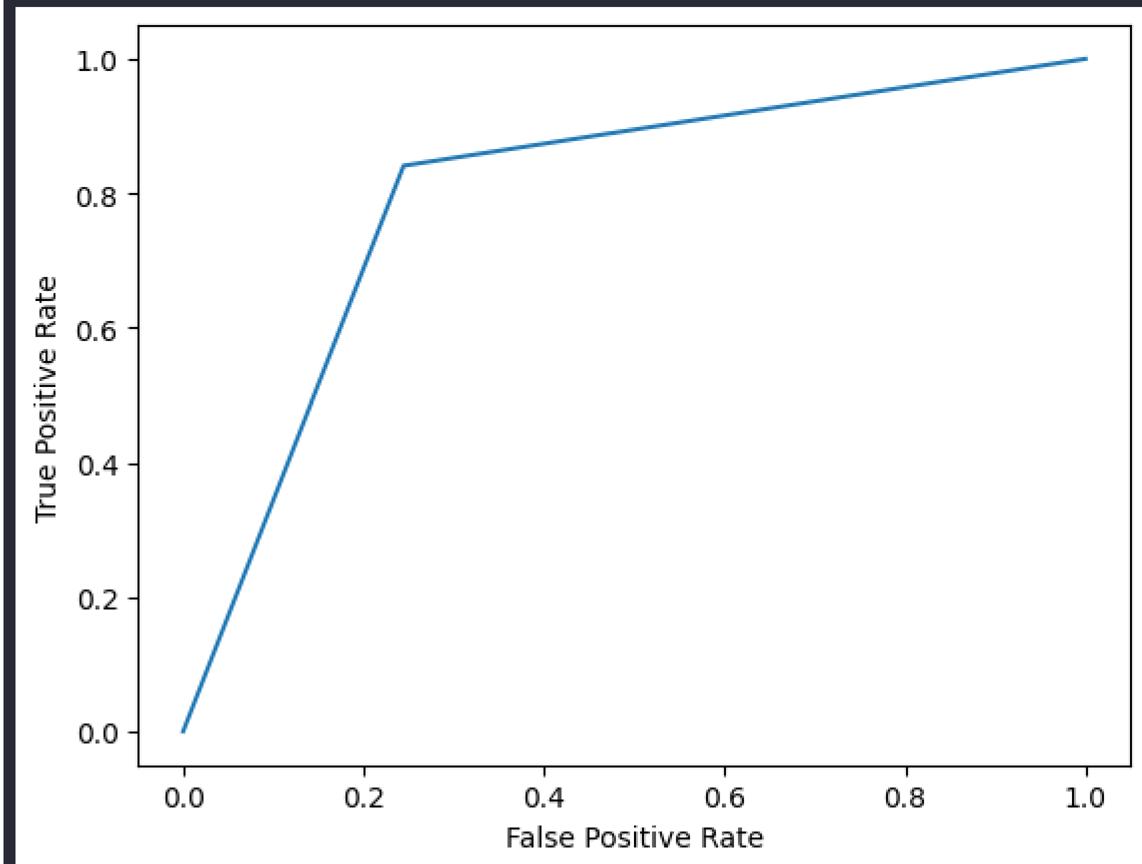
```python
# F1-Score
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred, average='macro'))
```
✓ 0.0s

```
0.7713445130920858
```

F1-Score

```python
plot_roc_curve(y_test, y_pred)
plt.show()
print(f'Bayes Classifier AUC score: {roc_auc_score(y_test, y_pred)}')
```

✓ 0.2s



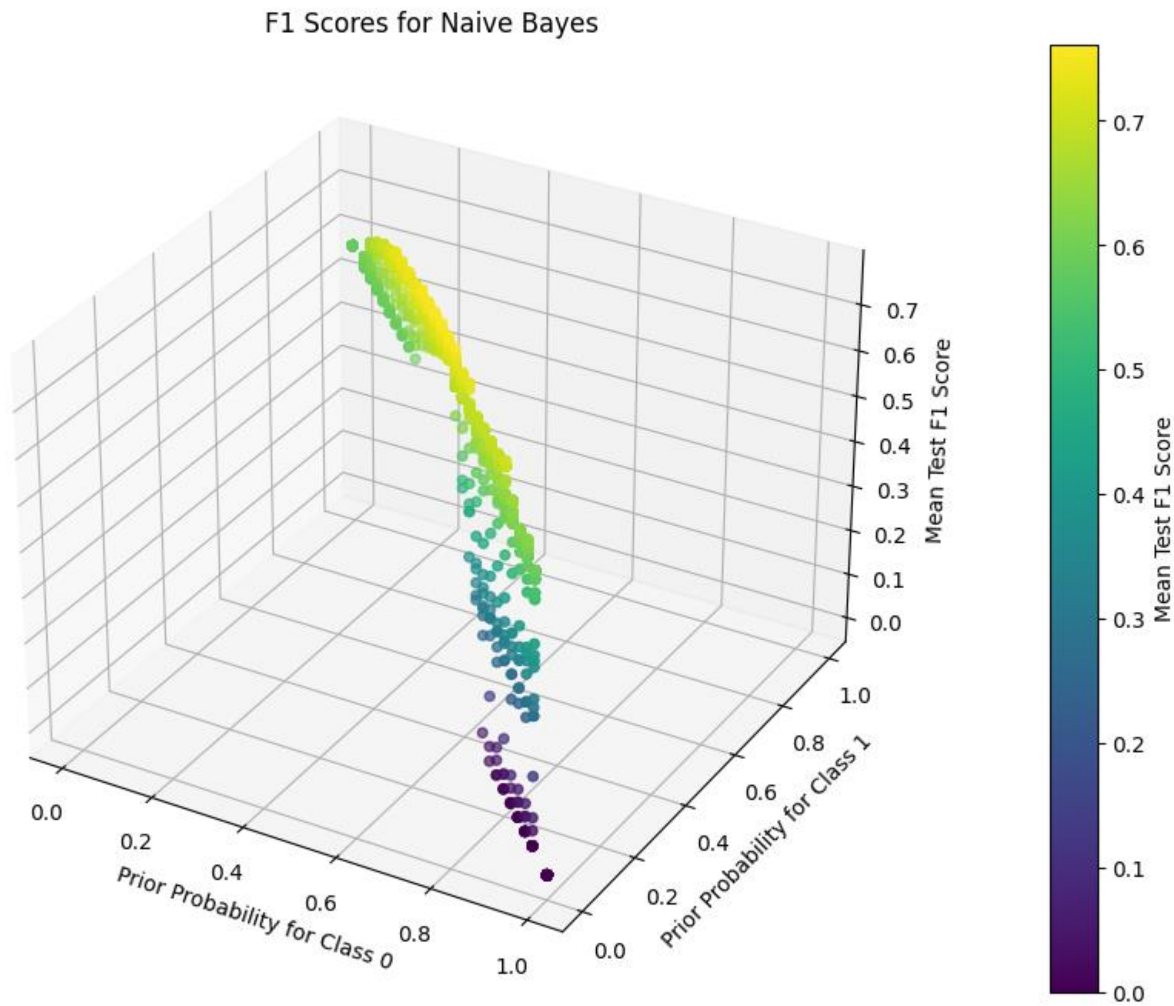

```
Bayes Classifier AUC score: 0.7982323232323232
```

ROC/AUC Curve

Visualization of Accuracy changes

1. F1-Score

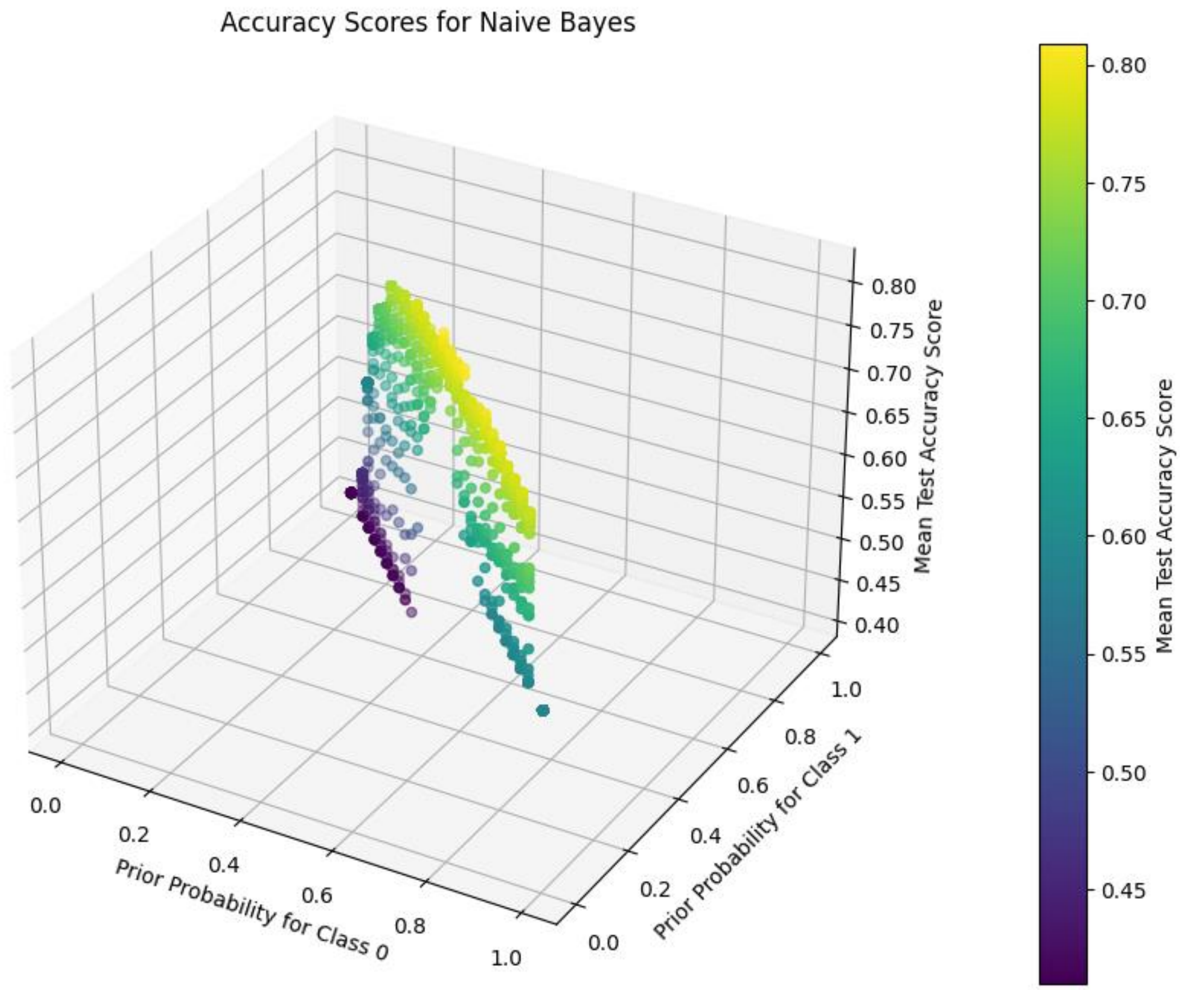
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D


# 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()
```



2. 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
```

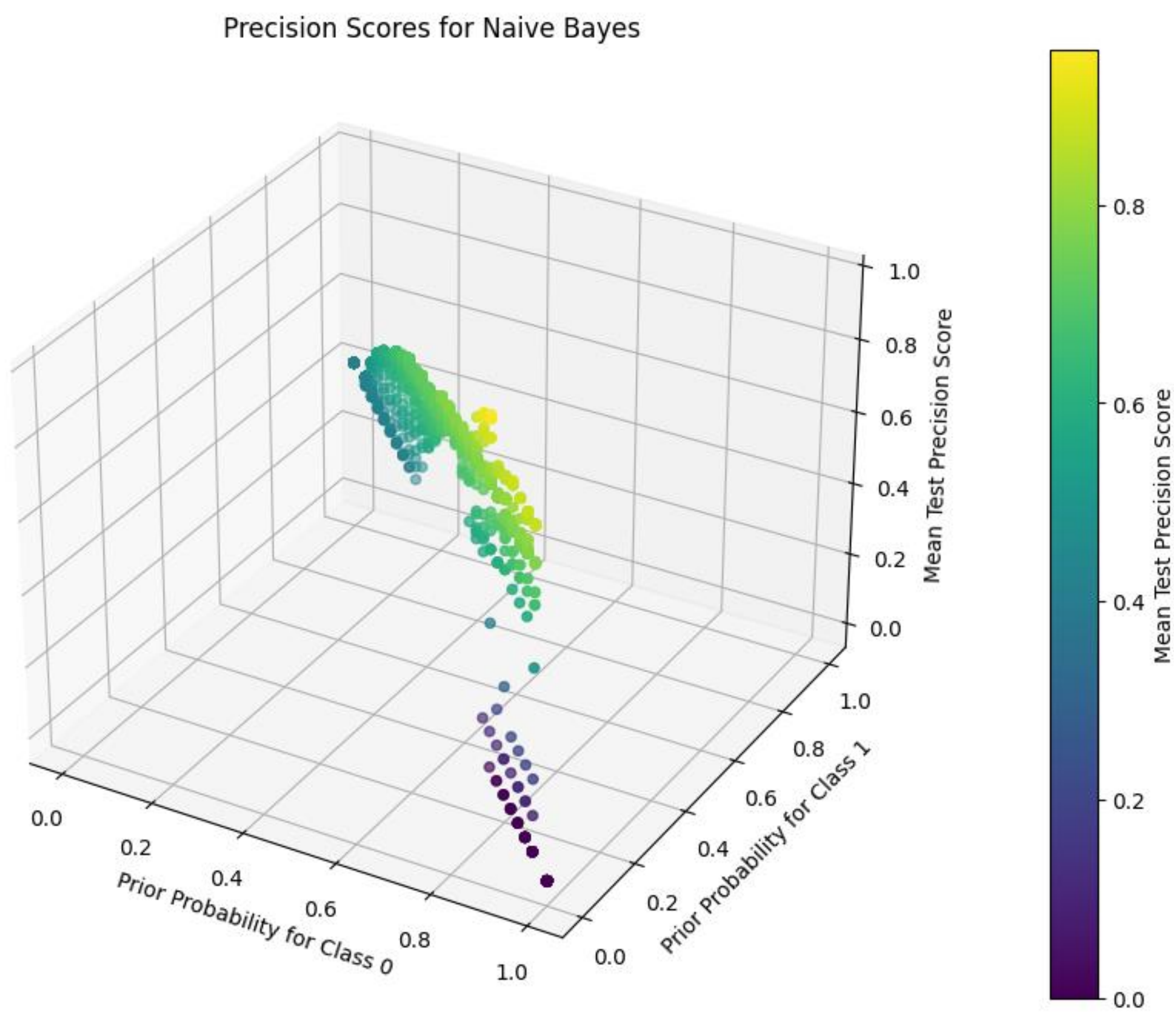
```python
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# 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()
```



3. 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]
```

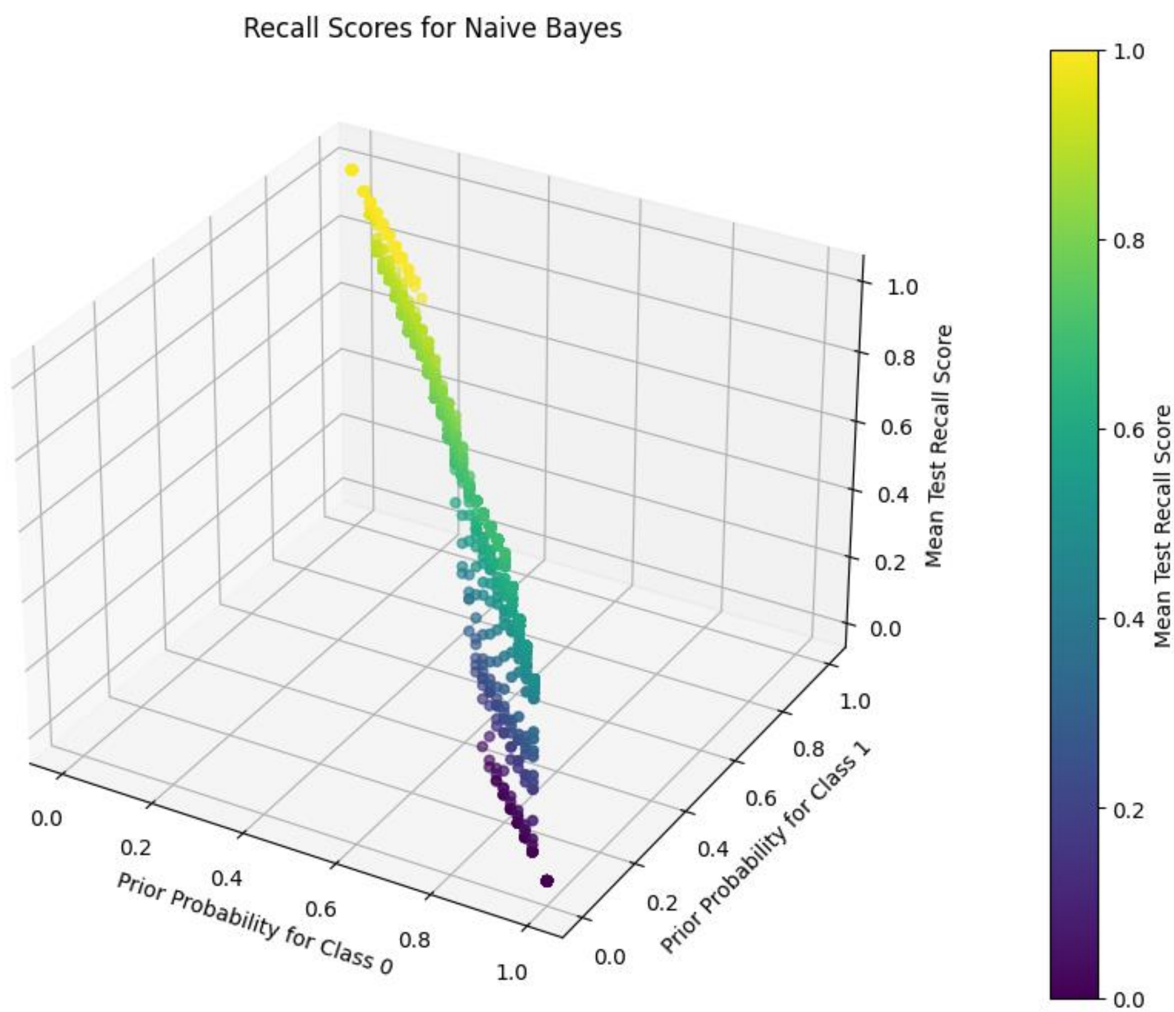
```python
# Create a 3D scatter plot to visualize the F1 scores
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# 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()
```



4. 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
```

```python
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()
```



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.