

**AIN SHAMS UNIVERSITY
FACULTY OF
ENGINEERING**



Introduction to Machine Learning

Phase Two

Name	ID
AbdulRahman AlSaeed Ahmed (breast cancer)	19P7808
Seif Eldin Amr Mostafa Hussein (iPhone Purchase)	20P2006
Sherwet Mohamed Khalil Barakat (loan)	20P8105

Contents

Breast cancer Dataset:	4
Introduction:.....	4
Methodology:.....	4
Preprocessing Steps:	4
Results:	5
MLP:	5
SVM:	5
Decision Tree:	5
Loan Dataset:.....	11
MLPC:	11
SVM:.....	13
Decision Trees:	15
Iphone Purchase:	18
Importing libraries:.....	18
Data loading and Preprocessing:	18
MLP Classifier:	19
SVM Classifier:.....	20
Decision Tree Classifier:.....	21
Tuning Hyperparameters:	22
Visualization:	23

Breast cancer Dataset:

Introduction:

In Milestone 2 of our project, we focused on training three advanced machine learning classifiers—MLP Classifier, SVM, and Decision Tree. Our goal was to identify the best performing model by tuning various hyperparameters and evaluating their performance based on accuracy.

Methodology:

Dataset description:

The dataset, `breast_cancer_diagnosis.csv`, comprises features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These features describe characteristics of the cell nuclei present in the image, categorized into two classes: Malignant (M) and Benign (B).

Preprocessing Steps:

The preprocessing involved:

- Imputation: Missing values were imputed using the mean of each feature.
- Standardization: Features were scaled using the `StandardScaler` to bring all values into a similar range.

Classifier Training and Hyperparameter Tuning:

We used `GridSearchCV` to explore various hyperparameters for each classifier:

- MLP Classifier: Tuned parameters included hidden layer sizes, activation functions, solvers, and alpha values.
- SVM Classifier: Explored different Values of C and Kernel Types.
- Decision Tree: Varied the tree depth and the minimum number of samples required to split an internal node.

Results:

MLP:

```
warnings.warn(
MLP best parameters: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'}
MLP accuracy: 0.9883040935672515
MLP classification report:
      precision    recall  f1-score   support

    0       0.99       0.99       0.99        108
    1       0.98       0.98       0.98         63

   accuracy       0.99
  macro avg       0.99
 weighted avg       0.99

Detailed accuracy for each hyperparameter combination:
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'} -> Mean: 0.970, Std: 0.006
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'adam'} -> Mean: 0.972, Std: 0.022
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'} -> Mean: 0.975, Std: 0.011
{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'} -> Mean: 0.965, Std: 0.020
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50,), 'solver': 'sgd'} -> Mean: 0.970, Std: 0.006
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50,), 'solver': 'adam'} -> Mean: 0.972, Std: 0.022
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'solver': 'sgd'} -> Mean: 0.975, Std: 0.011
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'solver': 'adam'} -> Mean: 0.965, Std: 0.019
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'} -> Mean: 0.967, Std: 0.010
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'adam'} -> Mean: 0.972, Std: 0.018
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'} -> Mean: 0.970, Std: 0.013
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'} -> Mean: 0.965, Std: 0.020
{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50,), 'solver': 'sgd'} -> Mean: 0.967, Std: 0.010
{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50,), 'solver': 'adam'} -> Mean: 0.975, Std: 0.014
{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'solver': 'sgd'} -> Mean: 0.970, Std: 0.013
{'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'solver': 'adam'} -> Mean: 0.970, Std: 0.020
SVM best parameters: {'C': 0.1, 'kernel': 'linear'}
```

SVM:

```
SVM best parameters: {'C': 0.1, 'kernel': 'linear'}
SVM accuracy: 0.9824561403508771
SVM classification report:
      precision    recall  f1-score   support

    0       0.98       0.99       0.99        108
    1       0.98       0.97       0.98         63

   accuracy       0.98
  macro avg       0.98
 weighted avg       0.98

Detailed accuracy for each hyperparameter combination:
{'C': 0.1, 'kernel': 'rbf'} -> Mean: 0.945, Std: 0.006
{'C': 0.1, 'kernel': 'linear'} -> Mean: 0.977, Std: 0.005
{'C': 0.1, 'kernel': 'rbf'} -> Mean: 0.965, Std: 0.015
{'C': 1, 'kernel': 'linear'} -> Mean: 0.975, Std: 0.016
{'C': 10, 'kernel': 'rbf'} -> Mean: 0.962, Std: 0.011
{'C': 10, 'kernel': 'linear'} -> Mean: 0.960, Std: 0.023
Decision Tree best parameters: {'max_depth': None, 'min_samples_split': 5}
```

Decision Tree:

```
{ 'C': 10, 'kernel': 'linear' } -> Mean: 0.960, Std: 0.023
Decision Tree best parameters: {'max_depth': None, 'min_samples_split': 5}
Decision Tree accuracy: 0.9298245614035088
Decision Tree classification report:
      precision    recall  f1-score   support

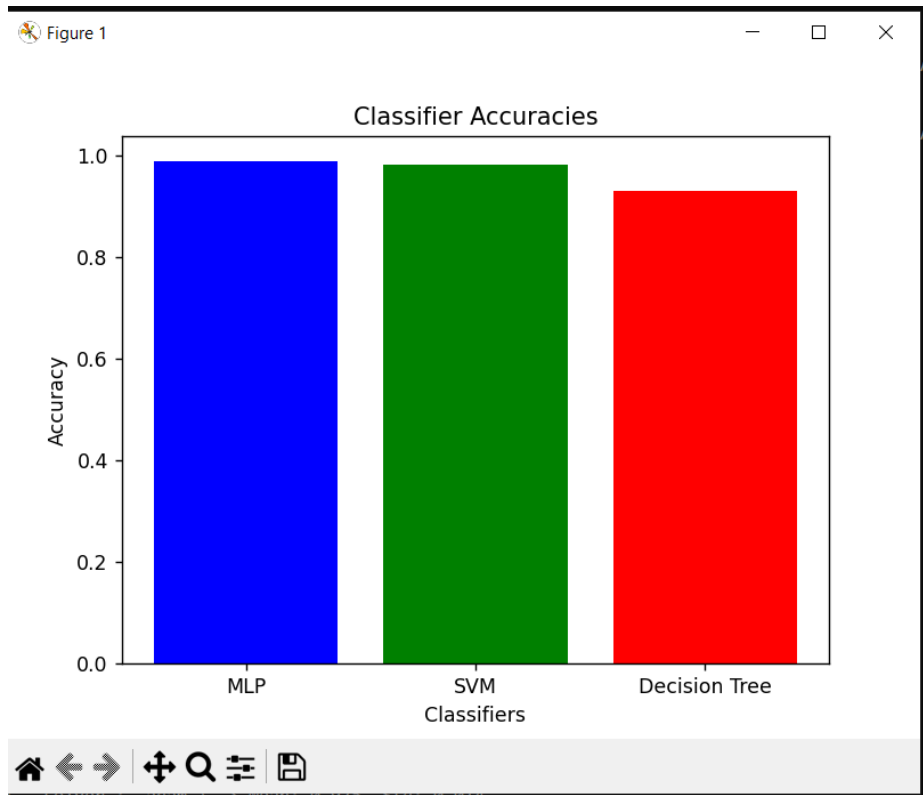
    0       0.96       0.93       0.94        108
    1       0.88       0.94       0.91         63

   accuracy       0.92
  macro avg       0.92
 weighted avg       0.93

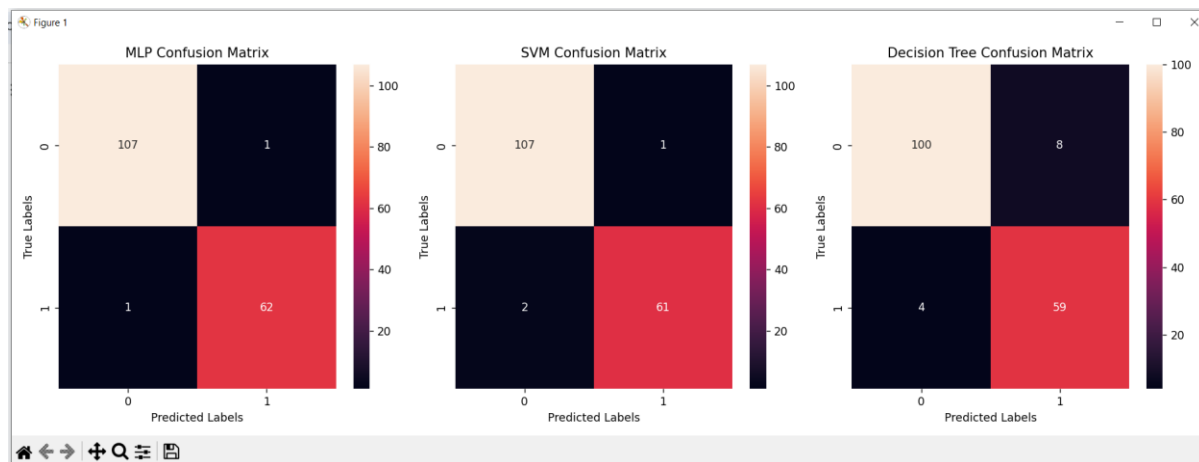
Detailed accuracy for each hyperparameter combination:
{'max_depth': None, 'min_samples_split': 2} -> Mean: 0.920, Std: 0.033
{'max_depth': None, 'min_samples_split': 5} -> Mean: 0.930, Std: 0.019
{'max_depth': None, 'min_samples_split': 10} -> Mean: 0.915, Std: 0.017
{'max_depth': 10, 'min_samples_split': 2} -> Mean: 0.920, Std: 0.033
{'max_depth': 10, 'min_samples_split': 5} -> Mean: 0.930, Std: 0.019
{'max_depth': 10, 'min_samples_split': 10} -> Mean: 0.915, Std: 0.017
{'max_depth': 20, 'min_samples_split': 2} -> Mean: 0.920, Std: 0.033
{'max_depth': 20, 'min_samples_split': 5} -> Mean: 0.930, Std: 0.019
{'max_depth': 20, 'min_samples_split': 10} -> Mean: 0.915, Std: 0.017
{'max_depth': 30, 'min_samples_split': 2} -> Mean: 0.920, Std: 0.033
{'max_depth': 30, 'min_samples_split': 5} -> Mean: 0.930, Std: 0.019
{'max_depth': 30, 'min_samples_split': 10} -> Mean: 0.915, Std: 0.017
```

Accuracy Bar Chart:

We visualized the accuracies of the models to compare their performance easily.



Confusion Matrices:



Discussion:

The MLP Classifier showed considerable flexibility in handling complex patterns, whereas the SVM provided robust performance with different kernels. The Decision Tree was fast and interpretable but slightly less accurate than the other models. The choice of hyperparameters was crucial in balancing overfitting and underfitting, as demonstrated by the grid search results.

Conclusion:

This milestone successfully met its objectives by implementing and evaluating three different classifiers. The findings suggest that while all classifiers performed well, some were more suited to specific types of data distributions and feature scales. The SVM and MLP classifiers in particular showed strong potential for this dataset after fine-tuning their hyperparameters.

Appendices:

Python Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv(r'C:\Users\dell\Downloads\breast_cancer_diagnosis.csv')

# Preprocess the dataset
X = data.drop('diagnosis', axis=1)
y = data['diagnosis'].map({'M': 1, 'B': 0}) # Encode labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

mlp_params = {
    'hidden_layer_sizes': [(50, ), (100,)],
    'activation': ['tanh', 'relu'],
```

```
    'solver': ['sgd', 'adam'],  
    'alpha': [0.0001, 0.05]  
}  
  
svm_params = {  
    'C': [0.1, 1, 10],  
    'kernel': ['rbf', 'linear']  
}  
  
dt_params = {  
    'max_depth': [None, 10, 20, 30],  
    'min_samples_split': [2, 5, 10]
```



```

models = {
    'MLP': (MLPClassifier(max_iter=1000, random_state=1), mlp_params),
    'SVM': (SVC(), svm_params),
    'Decision Tree': (DecisionTreeClassifier(random_state=42), dt_params)
}

results = {}
optimal_params = {}

for model_name, (model, params) in models.items():
    grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    predictions = best_model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    cm = confusion_matrix(y_test, predictions)
    report = classification_report(y_test, predictions)
    results[model_name] = (accuracy, cm, report, best_model)
    optimal_params[model_name] = grid_search.best_params_
    print(f"{model_name} best parameters: {grid_search.best_params_}")
    print(f"{model_name} accuracy: {accuracy}")
    print(f"{model_name} classification report:\n{report}")
    print("Detailed accuracy for each hyperparameter combination:")
    means = grid_search.cv_results_['mean_test_score']
    stds = grid_search.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, grid_search.cv_results_['params']):
        print(f"{params} -> Mean: {mean:.3f}, Std: {std:.3f}")

# Display optimal parameters clearly after all calculations
print("\nOptimal Parameters Summary:")
for model, params in optimal_params.items():
    print(f"{model}: {params}")

# Plotting the accuracies
accuracies = [result[0] for result in results.values()]
labels = list(results.keys())
plt.bar(labels, accuracies, color=['blue', 'green', 'red'])
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.title('Classifier Accuracies')
plt.show()

# Plotting confusion matrices
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

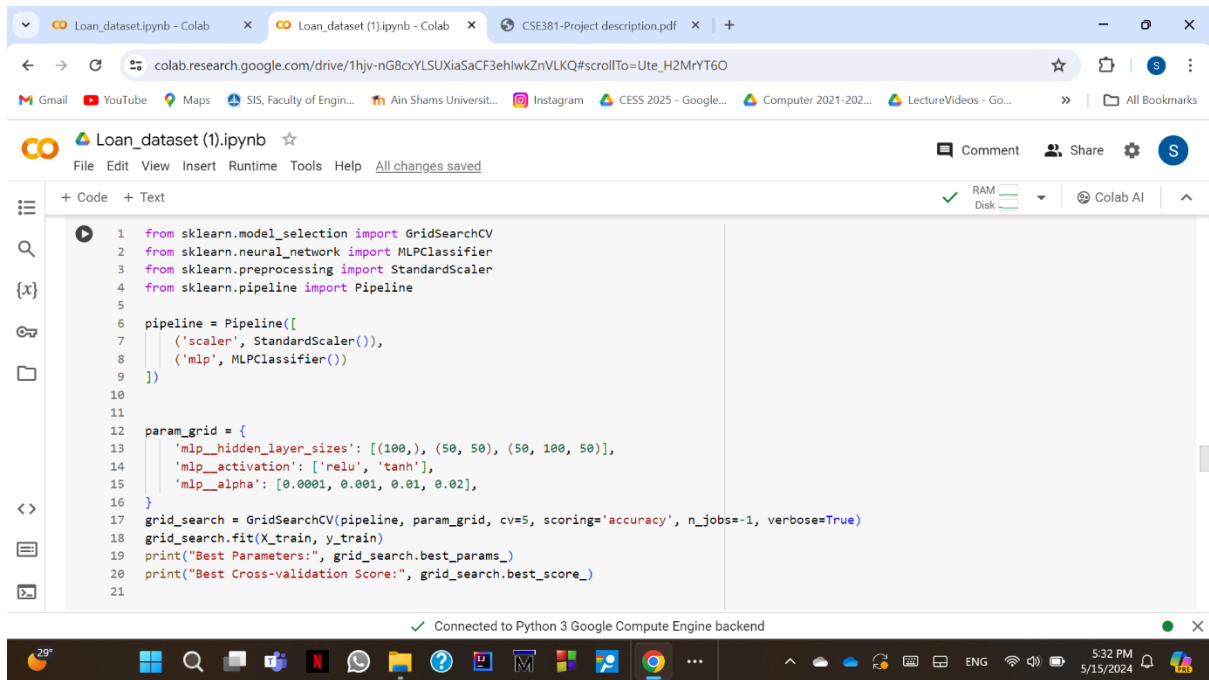
```

```
for ax, (model_name, result) in zip(axes,  
    results.items()): sns.heatmap(result[1],  
    annot=True, fmt="d", ax=ax)  
    ax.set_title(f'{model_name} Confusion Matrix')  
    ax.set_xlabel('Predicted Labels')  
    ax.set_ylabel('True  
Labels') plt.tight_layout()  
plt.show()
```

Loan Dataset:

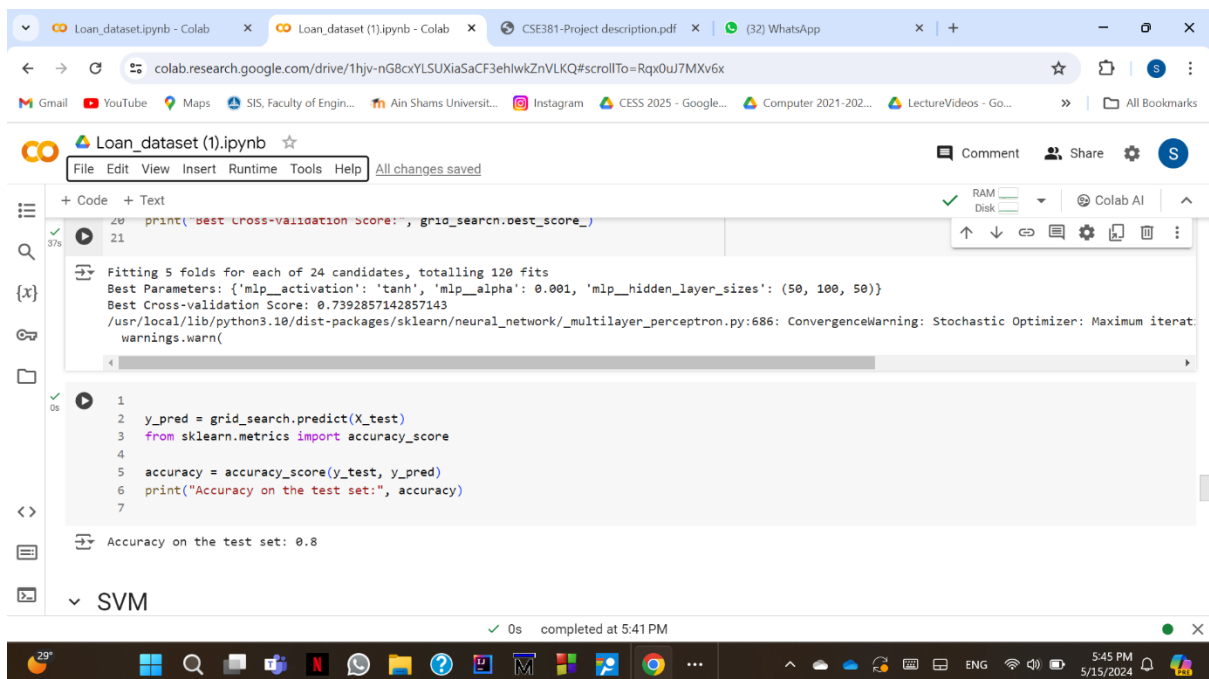
MLPC:

- By determining hidden layers (giving it a bunch of numbers), activation function (tanh) or (relu) and alpha.



```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.neural_network import MLPClassifier
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5
6 pipeline = Pipeline([
7     ('scaler', StandardScaler()),
8     ('mlp', MLPClassifier())
9 ])
10
11
12 param_grid = {
13     'mlp__hidden_layer_sizes': [(100, 50, 50), (50, 100, 50)],
14     'mlp__activation': ['relu', 'tanh'],
15     'mlp__alpha': [0.0001, 0.001, 0.01, 0.02],
16 }
17 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
18 grid_search.fit(X_train, y_train)
19 print("Best Parameters:", grid_search.best_params_)
20 print("Best Cross-validation Score:", grid_search.best_score_)
21
```

- First Hyperparameter by giving hidden layers (100,100), (50,50) and (50,100,50) to see the best cross validation score and accuracy.



```
20 print("Best Cross-validation Score:", grid_search.best_score_)
21
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'mlp__activation': 'tanh', 'mlp__alpha': 0.001, 'mlp__hidden_layer_sizes': (50, 100, 50)}
Best Cross-validation Score: 0.7392857142857143
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations 1000 reached, but optimization didn't converge.
warnings.warn(

1 y_pred = grid_search.predict(X_test)
2
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7
```

Accuracy on the test set: 0.8

SVM

- Trying different hyperparameter by giving hidden layer (100), (50,50) and (50,100,50). This means that the above one has better accuracy than this one.

The screenshot shows a Google Colab notebook titled "Loan_dataset (1).ipynb". The code in the notebook is as follows:

```
18 grid_search.fit(X_train, y_train)
19 print("Best Parameters:", grid_search.best_params_)
20 print("Best Cross-validation Score:", grid_search.best_score_)
21
```

The output of the code is:

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'mlp_activation': 'tanh', 'mlp_alpha': 0.001, 'mlp_hidden_layer_sizes': (50, 100, 50)}
Best Cross-validation Score: 0.7357792207792209
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations 1000 reached, but optimization didn't converge.
warnings.warn(
```

The notebook also shows the accuracy of the model on the test set:

```
1 y_pred = grid_search.predict(X_test)
2 from sklearn.metrics import accuracy_score
3
4 accuracy = accuracy_score(y_test, y_pred)
5 print("Accuracy on the test set:", accuracy)
6
```

The output of the accuracy calculation is:

```
Accuracy on the test set: 0.7857142857142857
```

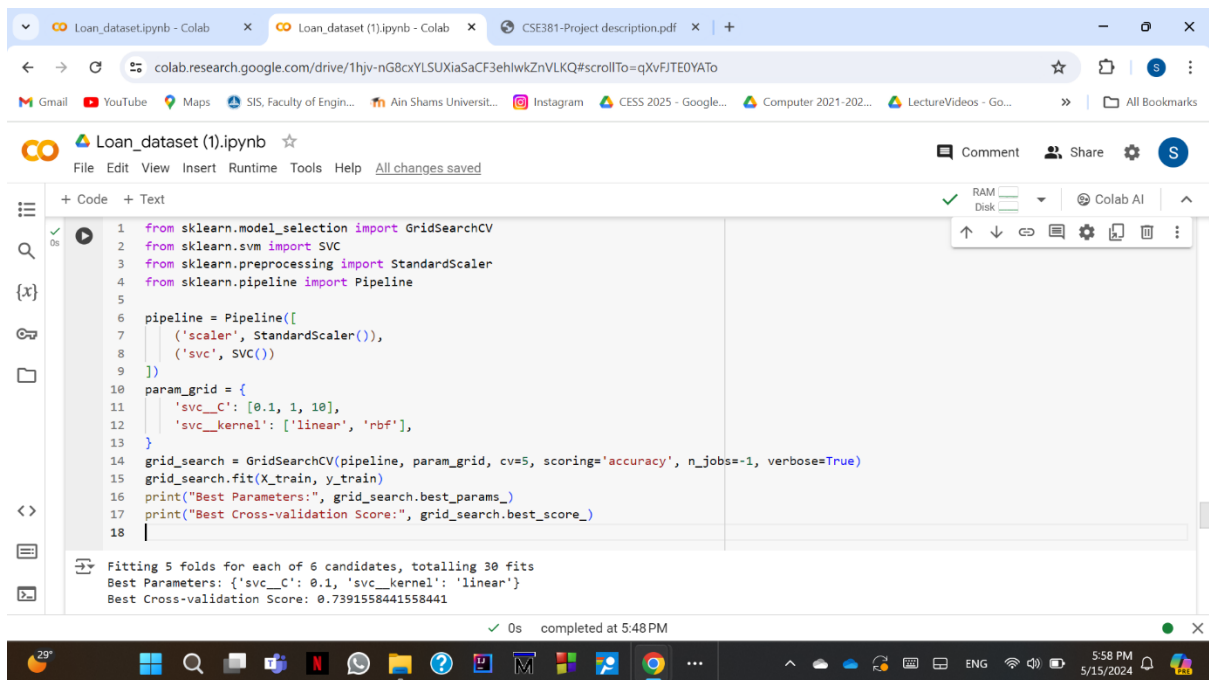
- Accuracy Code of MLPC:

```
[53] 1
      2 y_pred = grid_search.predict(X_test)
      3 from sklearn.metrics import accuracy_score
      4
      5 accuracy = accuracy_score(y_test, y_pred)
      6 print("Accuracy on the test set:", accuracy)
      7
```

➡ Accuracy on the test set: 0.7857142857142857

SVM:

- 'svc__C': This corresponds to the C parameter of the SVC model, which controls the regularization strength. It has three values which are: 0.1, 1, and 10.
- 'svc__kernel': This corresponds to the kernel parameter of the SVC model, which specifies the kernel type to be used in the algorithm. It has two values which are: 'linear' and 'rbf'.
- And from these hyperparameters, they show that the best SVC is to be linear and of strength 0.1 to give best accuracy.



```
1 from sklearn.model_selection import GridSearchCV
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5
6 pipeline = Pipeline([
7     ('scaler', StandardScaler()),
8     ('svc', SVC())
9 ])
10
11 param_grid = {
12     'svc__C': [0.1, 1, 10],
13     'svc__kernel': ['linear', 'rbf'],
14 }
15
16 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
17 grid_search.fit(X_train, y_train)
18 print("Best Parameters:", grid_search.best_params_)
19 print("Best Cross-validation Score:", grid_search.best_score_)
20
```

Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best Parameters: {'svc__C': 0.1, 'svc__kernel': 'linear'}
Best Cross-validation Score: 0.7391558441558441

- Accuracy in this case is 0.8.

```
1
2 y_pred = grid_search.predict(X_test)
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7
```

Accuracy on the test set: 0.8

- If we changed the hyperparameter and tried to add 0.01 for example, it will show that this is the best strength which means that as the strength decreases it gives better cross validation and accuracy.

```

2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5
6 pipeline = Pipeline([
7     ('scaler', StandardScaler()),
8     ('svc', SVC())
9 ])
10 param_grid = {
11     'svc__C': [0.01, 0.1, 1, 10],
12     'svc__kernel': ['linear', 'rbf'],
13 }
14 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
15 grid_search.fit(X_train, y_train)
16 print("Best Parameters:", grid_search.best_params_)
17 print("Best Cross-validation Score:", grid_search.best_score_)
18

```

Fitting 5 folds for each of 8 candidates, totalling 40 fits
 Best Parameters: {'svc__C': 0.01, 'svc__kernel': 'linear'}
 Best Cross-validation Score: 0.7391558441558441

```

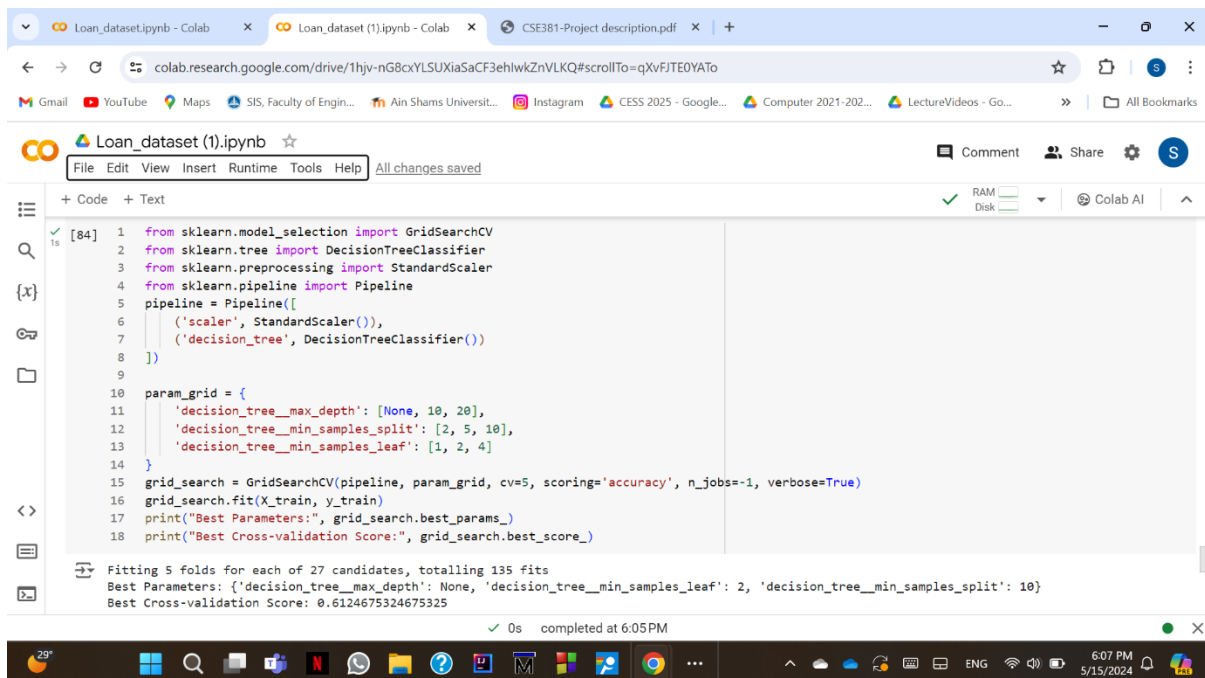
1
2 y_pred = grid_search.predict(X_test)
3 from sklearn.metrics import accuracy_score
4
5 accuracy = accuracy_score(y_test, y_pred)
6 print("Accuracy on the test set:", accuracy)
7

```

➡ Accuracy on the test set: 0.8

Decision Trees:

- `Decision_tree__max_depth'`: This corresponds to the `max_depth` parameter of the `DecisionTreeClassifier`, which controls the maximum depth of the tree. It has three values which are: None, 10, and 20.
- `Decision_tree__min_samples_split'`: This corresponds to the `min_samples_split` parameter, which specifies the minimum number of samples required to split an internal node. It has three values which are: 2, 5, and 10.
- `Decision_tree__min_samples_leaf'`: This corresponds to the `min_samples_leaf` parameter, which specifies the minimum number of samples required to be at a leaf node. It has three values which are: 1, 2, and 4.
- The best hyperparameter in this example is that the max depth to be None, min sample is 2 and min sample split is 10.



```
[84] 1 from sklearn.model_selection import GridSearchCV
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5 pipeline = Pipeline([
6     ('scaler', StandardScaler()),
7     ('decision_tree', DecisionTreeClassifier())
8 ])
9
10 param_grid = {
11     'decision_tree__max_depth': [None, 10, 20],
12     'decision_tree__min_samples_split': [2, 5, 10],
13     'decision_tree__min_samples_leaf': [1, 2, 4]
14 }
15 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
16 grid_search.fit(X_train, y_train)
17 print("Best Parameters:", grid_search.best_params_)
18 print("Best Cross-validation Score:", grid_search.best_score_)

Fitting 5 folds for each of 27 candidates, totalling 135 fits
Best Parameters: {'decision_tree__max_depth': None, 'decision_tree__min_samples_leaf': 2, 'decision_tree__min_samples_split': 10}
Best Cross-validation Score: 0.6124675324675325

0s completed at 6:05PM
```

- Accuracy:

```

1
2 y_pred = grid_search.predict(X_test)
3
4 from sklearn.metrics import accuracy_score
5
6 accuracy = accuracy_score(y_test, y_pred)
7 print("Accuracy on the test set:", accuracy)
8

```

Accuracy on the test set: 0.7

- By changing Hyperparameters by adding 20 to min sample split and 3 to min sample leaf. We'll find out that the best min sample split is 20 which made the cross validation better and the accuracy is way better.

```

1 from sklearn.model_selection import GridSearchCV
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.preprocessing import StandardScaler
4 from sklearn.pipeline import Pipeline
5 pipeline = Pipeline([
6     ('scaler', StandardScaler()),
7     ('decision_tree', DecisionTreeClassifier())
8 ])
9
10 param_grid = {
11     'decision_tree__max_depth': [None, 10, 20],
12     'decision_tree__min_samples_split': [2, 5, 10, 20],
13     'decision_tree__min_samples_leaf': [1, 2, 3, 4]
14 }
15 grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=True)
16 grid_search.fit(X_train, y_train)
17 print("Best Parameters:", grid_search.best_params_)
18 print("Best Cross-validation Score:", grid_search.best_score_)

```

Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best Parameters: {'decision_tree__max_depth': None, 'decision_tree__min_samples_split': 2, 'decision_tree__min_samples_leaf': 20}
Best Cross-validation Score: 0.6708454545454545

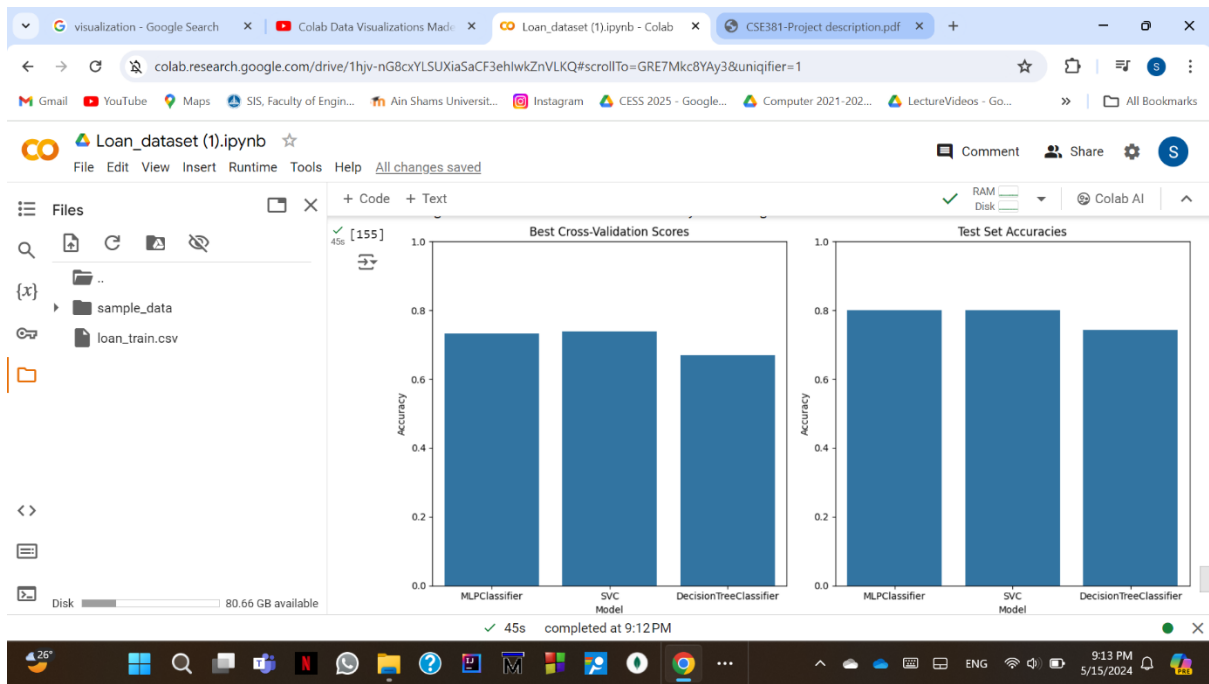
```

1
2 y_pred = grid_search.predict(X_test)
3
4 from sklearn.metrics import accuracy_score
5
6 accuracy = accuracy_score(y_test, y_pred)
7 print("Accuracy on the test set:", accuracy)
8

```

→ Accuracy on the test set: 0.7428571428571429

Visualization:



Iphone Purchase:

Importing libraries:

Importing Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

Data loading and Preprocessing:

Loading and Preprocessing

```
# Load the dataset
df = pd.read_csv('Iphone_purchase.csv')

# Drop the 'User ID' column
df.drop('User ID', axis=1, inplace=True)

# Preprocess the data
# Encoding categorical column 'Gender'
labelEncoder = LabelEncoder()
labelEncoder.fit(df["Gender"])
df["Gender"] = labelEncoder.transform(df["Gender"])

# Splitting the data into features (X) and target variable (y)
X = df[['Gender', 'Age', 'EstimatedSalary']]
y = df['Purchased']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

```
# Scaling using MinMaxScaler, normalizing balanced classes gives same accuracies
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train_smote)
X_test_normalized = scaler.transform(X_test)

# Standardize the features using StandardScaler for comparison, standardizing balanced classes gives lower accuracies
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# scaling balanced classes
scaler_smote = StandardScaler()
X_train_scaled_smote = scaler_smote.fit_transform(X_train_smote)
X_test_scaled_smote = scaler_smote.transform(X_test)
```

MLP Classifier:

Using MinMaxScaler():

```
# Define the MLPClassifier
mlp = MLPClassifier(max_iter=1000)

# Define the parameter grid for hyperparameter tuning
parameter_space = [
    'hidden_layer_sizes': [(10,10), (50,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
]

# Perform GridSearchCV to find the best hyperparameters
clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=5)
clf.fit(X_train_normalized, y_train_smote)

# Best parameters and best score
best_params_MLP = clf.best_params_
best_score = clf.best_score_

# Test the best MLP Classifier
mlp_best = clf.best_estimator_
mlp_predictions = mlp_best.predict(X_test_normalized)
mlp1_accuracy = accuracy_score(y_test, mlp_predictions)

print("Best MLP Classifier Accuracy (Normalized Data):", mlp1_accuracy * 100)
print("Hyperparameters for Best MLP Classifier:")
print(" - hidden_layer_sizes:", best_params_MLP['hidden_layer_sizes'])
print(" - activation:", best_params_MLP['activation'])
print(" - solver:", best_params_MLP['solver'])
print(" - alpha:", best_params_MLP['alpha'])
print(" - learning_rate:", best_params_MLP['learning_rate'])
print("Best MLP Classifier Predictions on Testing Set (Normalized Data):")
print(mlp_predictions)
```

Using standardscaler():

```
# Perform GridSearchCV with standardized data
clf.fit(X_train_scaled, y_train)

# Best parameters and best score
best_params = clf.best_params_
best_score = clf.best_score_

# Test the best MLP Classifier
mlp_best = clf.best_estimator_
mlp_predictions = mlp_best.predict(X_test_scaled)
mlp_accuracy = accuracy_score(y_test, mlp_predictions)

print("Best MLP Classifier Accuracy (Standardized Data):", mlp_accuracy * 100)
print("Hyperparameters for Best MLP Classifier:")
print(" - hidden_layer_sizes:", best_params['hidden_layer_sizes'])
print(" - activation:", best_params['activation'])
print(" - solver:", best_params['solver'])
print(" - alpha:", best_params['alpha'])
print(" - learning_rate:", best_params['learning_rate'])
print("Best MLP Classifier Predictions on Testing Set (Standardized Data):")
print(mlp_predictions)
```

Output:

```
Best MLP Classifier Accuracy (Normalized Data): 92.5
Hyperparameters for Best MLP Classifier:
- hidden_layer_sizes: (50, 50)
- activation: relu
- solver: adam
- alpha: 0.0001
- learning_rate: adaptive
Best MLP Classifier Predictions on Testing Set (Normalized Data):
[1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1 1 0 0 0
 1 0 1 1 0 0]
Best MLP Classifier Accuracy (Standardized Data): 92.5
Hyperparameters for Best MLP Classifier:
- hidden_layer_sizes: (10, 10)
- activation: tanh
- solver: adam
- alpha: 0.0001
- learning_rate: adaptive
Best MLP Classifier Predictions on Testing Set (Standardized Data):
[1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0
 1 0 1 1 0 0]
```

SVM Classifier:

```
# Define the SVM Classifier
svm = SVC()

# Define the parameter grid for hyperparameter tuning
parameter_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['linear', 'rbf']
}

# Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(svm, parameter_grid, refit=True, verbose=2, cv=5, n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Test the best SVM Classifier
svm_best = grid_search.best_estimator_
svm_predictions = svm_best.predict(X_test_scaled)
svm_accuracy = accuracy_score(y_test, svm_predictions)

print("Best SVM Classifier Accuracy:", svm_accuracy * 100)
print("Hyperparameters for Best SVM Classifier:")
print(" - C:", best_params['C'])
print(" - gamma:", best_params['gamma'])
print(" - kernel:", best_params['kernel'])
print("Best SVM Classifier Predictions on Testing Set:")
print(svm_predictions)
```

Output:

```
Fit Fitting 5 folds for each of 32 candidates, totalling 160 fits
Best SVM Classifier Accuracy: 92.5
Hyperparameters for Best SVM Classifier:
- C: 1
- gamma: 1
- kernel: rbf
Best SVM Classifier Predictions on Testing Set:
[1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0
 1 0 1 1 0 0]
```

Decision Tree Classifier:

Decision Tree Classifier

```
[20] # Define the Decision Tree Classifier
decision_tree = DecisionTreeClassifier()

# Define the parameter grid for hyperparameter tuning
parameter_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 4, 6, 8, 10, 20, 30],
    'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'auto', 'sqrt', 'log2'],
}

# Perform GridSearchCV to find the best hyperparameters
grid_search = GridSearchCV(decision_tree, parameter_grid, refit=True, verbose=2, cv=5, n_jobs=-1)
# Decision tree performance is not affected by scaling differences so, no need to use scaled
grid_search.fit(X_train_smote, y_train_smote)

# Best parameters and best score
best_params_DT = grid_search.best_params_
best_score = grid_search.best_score_

# Test the best Decision Tree Classifier
decision_tree_best = grid_search.best_estimator_
decision_tree_predictions = decision_tree_best.predict(X_test)
decision_tree_accuracy = accuracy_score(y_test, decision_tree_predictions)
```

```
print("Best Decision Tree Classifier Accuracy:", decision_tree_accuracy * 100)
print("Hyperparameters for Best Decision Tree Classifier:")
print(" - criterion:", best_params_DT['criterion'])
print(" - max_depth:", best_params_DT['max_depth'])
print(" - min_samples_split:", best_params_DT['min_samples_split'])
print(" - min_samples_leaf:", best_params_DT['min_samples_leaf'])
print(" - max_features:", best_params_DT['max_features'])
print("Best Decision Tree Classifier Predictions on Testing Set:")
print(decision_tree_predictions)
```

Output:

```
⚙ Fitting 5 folds for each of 1920 candidates, totalling 9600 fits
Best Decision Tree Classifier Accuracy: 86.25
Hyperparameters for Best Decision Tree Classifier:
- criterion: entropy
- max_depth: 30
- min_samples_split: 7
- min_samples_leaf: 4
- max_features: auto
Best Decision Tree Classifier Predictions on Testing Set:
[1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0
 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0
 1 0 1 1 0 0]
```

Tuning Hyperparameters:

We use GridSearch():

MLP:

```
# Define the parameter grid for hyperparameter tuning
parameter_space = {
    'hidden_layer_sizes': [(10,10), (50,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],}
```

SVM:

```
# Define the parameter grid for hyperparameter tuning
parameter_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['linear', 'rbf']
}
```

Decision Tree:

```
# Define the parameter grid for hyperparameter tuning
parameter_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 4, 6, 8, 10, 20, 30],
    'min_samples_split': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'auto', 'sqrt', 'log2'],
}
```

Visualization:

Tuning max_depths of decision tree

```
# Define a range of max depth values to test
max_depth_values = list(range(1, 11))

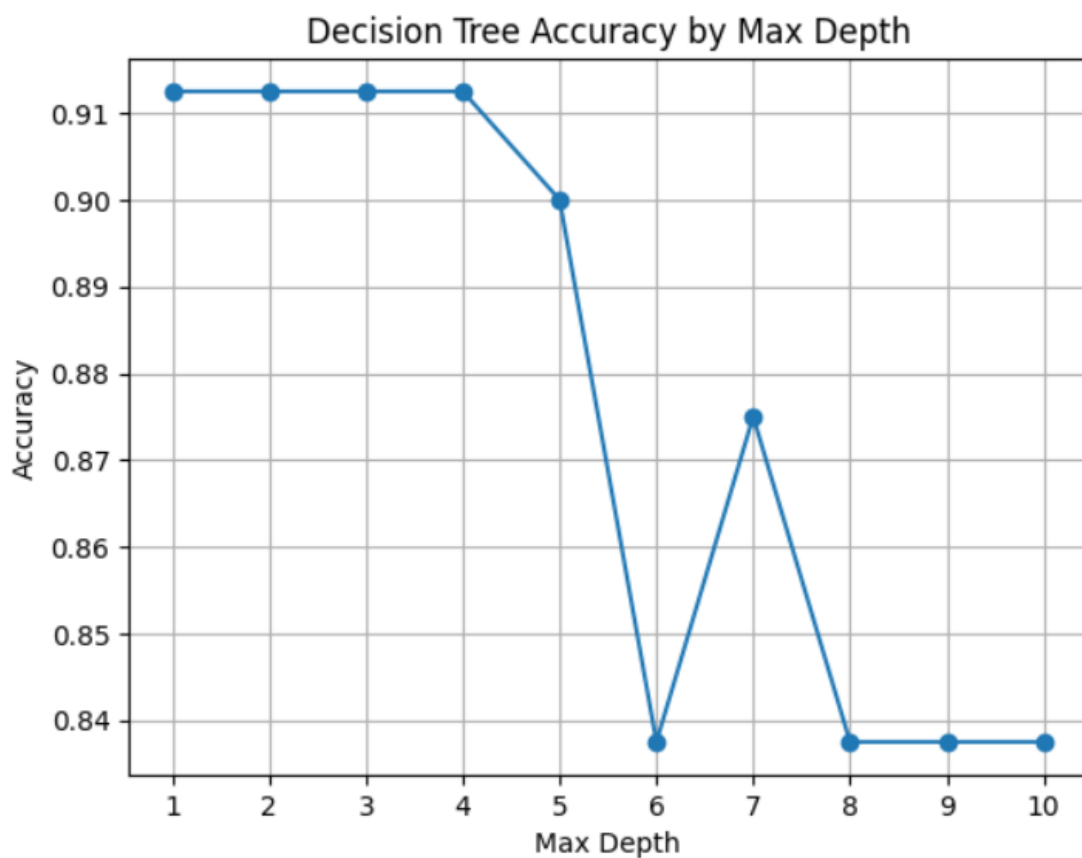
# Initialize an empty list to store accuracy values
accuracy_values = []

# Train decision tree classifiers with different max depth values
for max_depth in max_depth_values:
    # Initialize and train decision tree classifier
    decision_tree = DecisionTreeClassifier(max_depth=max_depth, random_state=42)
    decision_tree.fit(X_train_smote, y_train_smote)

    # Make predictions on the test set
    decision_tree_predictions = decision_tree.predict(X_test)

    # Calculate accuracy and append to list
    accuracy = accuracy_score(y_test, decision_tree_predictions)
    accuracy_values.append(accuracy)

# Plot the relationship between max depth and accuracy
plt.plot(max_depth_values, accuracy_values, marker='o')
plt.title('Decision Tree Accuracy by Max Depth')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.xticks(max_depth_values)
plt.grid(True)
plt.show()
```



Plotting a graph to visualize accuracies of each classifier:

```
classifiers = ['MLP', 'SVM', 'Decision Tree']  
accuracies = [mlp_accuracy, svm_accuracy, decision_tree_accuracy]  
  
plt.bar(classifiers, accuracies, color=['blue', 'red', 'yellow'])  
plt.xlabel('Classifier')  
plt.ylabel('Accuracy')  
plt.title('Comparison of Classifier Accuracies')  
plt.ylim(0, 1) # Set y-axis limit to 0-1 for accuracy percentage  
plt.show()
```

