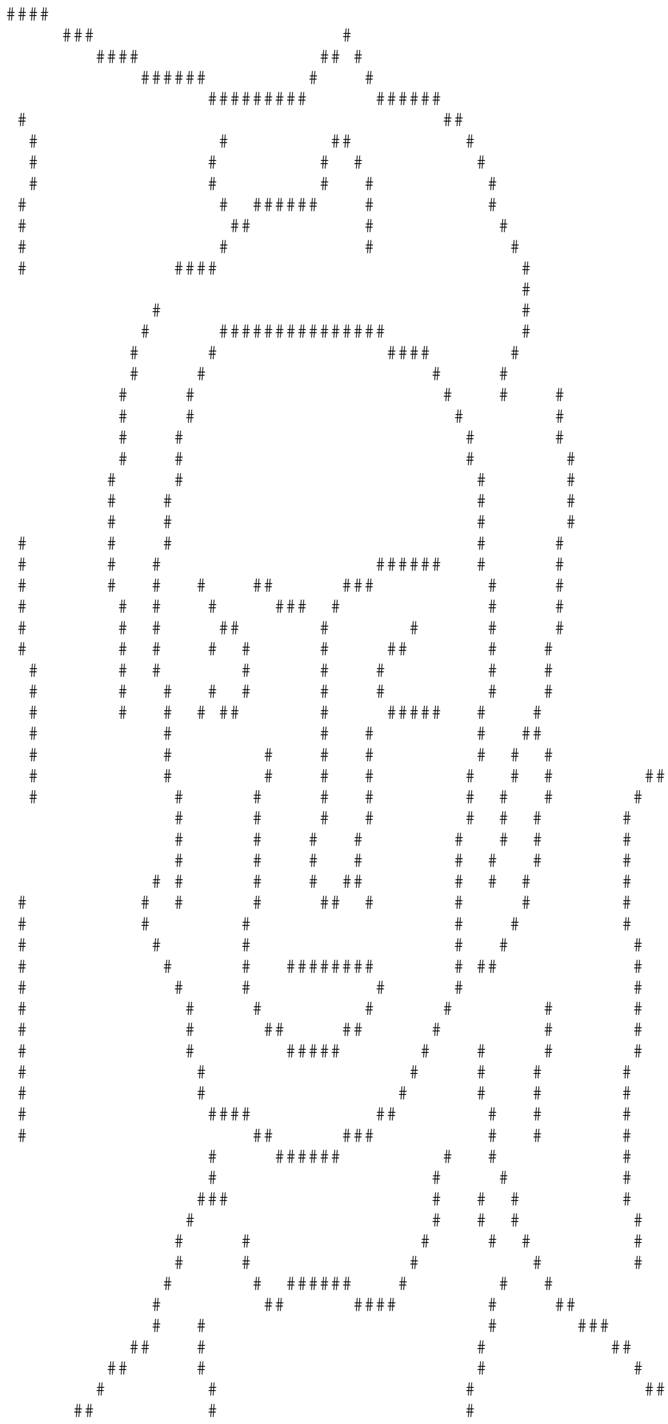
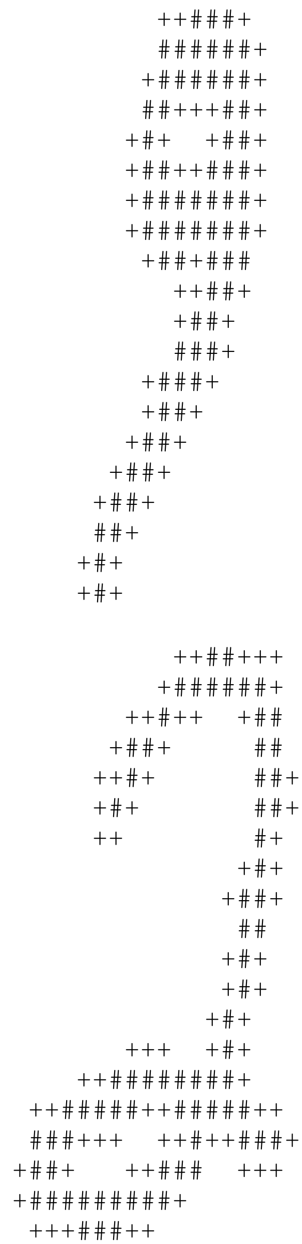
Milestone -2

Team members :-

Boules Emad	**19P9291**
Sohayla Hamed	**19P7343**
Salma Hamed	**19P8794**





Contents

1. MLP ALGORITHM

1.1 STEPS

1) Import necessary libraries:

```
import warnings
import random
import numpy as np
import matplotlib.pyplot as plt
# loading datasets

# handwritten numerical face
ocr_training_samples = 5000
ocr_test_samples = 1000
digits_train_data, digits_train_labels = producedata(ocr_training_samples, "data/digitdata/trainingim
digits_test_data, digits_test_labels  = producedata(ocr_test_samples, "data/digitdata/testimages", "d

# edge image: classify as face or no face
face_training_samples=451
face_test_samples=150
face_train_data, face_train_labels = producedata(face_training_samples,"data/facedata/facedatatrain",
face_test_data, face_test_labels = producedata(face_test_samples,"data/facedata/facedatatest", "data/
```

2) Tune for best hyperparameters and visualize the scores

Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes. Typical examples include C, kernel and gamma for Support Vector Classifier.

The grid search provided by GridSearchCV exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter.

Using GridSearchCV, we get the best accuracy and train the data using these hyperparameters. Then, we predict labels for the testing data and output the accuracy. We also visualize random hyperparameters combinations vs their score using matplotlib.

Note: The accuracy of the test data when using the optimal hyperparameters will be slightly less than the score using train data. This is because we use the train data to get the hyperparameters. Note also that the accuracy for face is less than digits all the time.

```
#mlp classifier

def classify_mlp(train_data, test_data, train_labels, test_labels):

    # changing hyperparameters
    # first, we get the best hyperparamters using exhaustive search
    warnings.filterwarnings("ignore")
    cv = ShuffleSplit(n_splits=1, test_size=0.2, random_state=1)
    param_grid = {
    'hidden_layer_sizes': [5], #'hidden_layer_sizes': [5,10,15,(5,5),(5,10)],
    'activation': ['identity','logistic','tanh','relu'],
    'solver': ['lbfgs','sgd','adam'],
    'max_iter': [1000],
    'random_state': [5] }

    gridSearch = GridSearchCV(MLPClassifier(), param_grid, cv=cv,
                          refit=True,verbose=2, return_train_score=True,
                          n_jobs=-1)
    gridSearch.fit(train_data, train_labels)
    print("cv_results_: ", gridSearch.cv_results_)
    print('Score: ', gridSearch.best_score_)
    print('Parameters: ', gridSearch.best_params_)
    # then, we plot the parameter vs the score (uncomment the next line)
    #plot_grid_search(gridSearch.cv_results_)
```

3) Train and test data using best hyperparameters

```
# train, test accuracy
h = gridSearch.best_params_['hidden_layer_sizes']
a = gridSearch.best_params_['activation']
s = gridSearch.best_params_['solver']
mlp = MLPClassifier(hidden_layer_sizes=h,
                    activation=a,
                    solver=s,
                    random_state=5, max_iter=1000)
mlp.fit(train_data, train_labels)
#print(mlp.score(train_data, train_labels)) = 1.0 when converge

predictions_test = mlp.predict(test_data)
test_score = accuracy_score(predictions_test, test_labels) # accuracy of
print("\naccuracy score on test data: ", test_score)
rand = random.randint(0, 100) #output any random row to see
print("example of predicted output: ", predictions_test[:rand]) # predict
print("whereas actual output: ", test_labels[:rand]) #actual label
```

1.2 OUTPUT

1.2.1 GridSearchCV

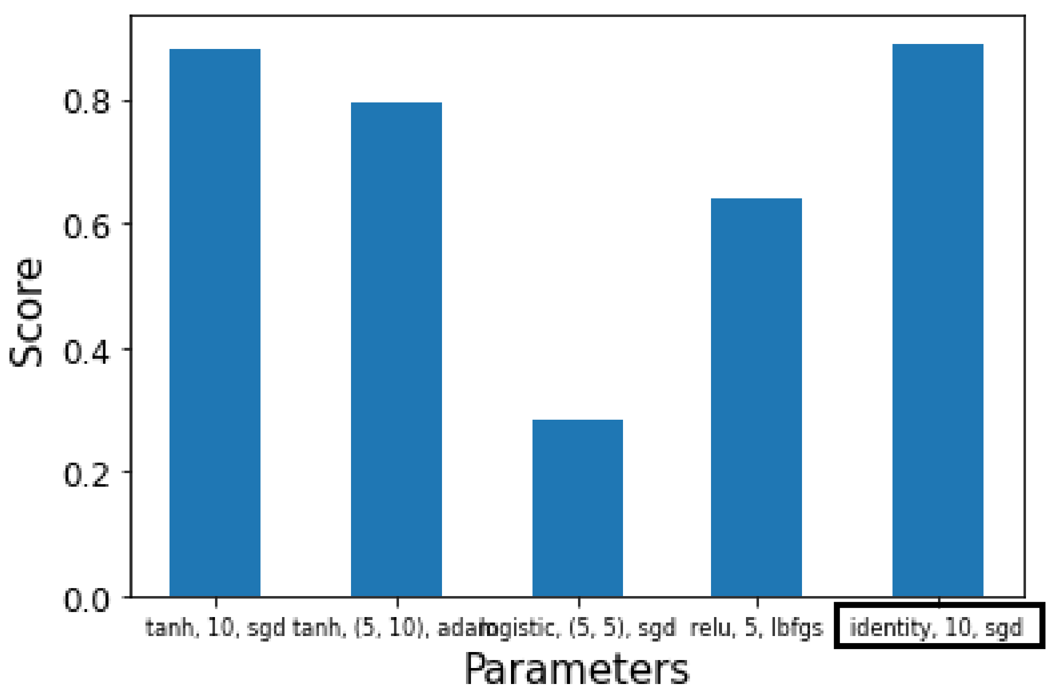
```
accuracy score on test data:  0.912
example of predicted output:  [9 0 2 5 1 9 7 8 1 0 4 1 7 9 0 4 2 6 8 1]
whereas actual output:  [9, 0, 2, 5, 1, 9, 7, 8, 1, 0, 4, 1, 7, 9, 6, 4, 2, 6, 8, 1]
Fitting 1 folds for each of 12 candidates, totalling 12 fits
[CV] END activation=identity, hidden_layer_sizes=5, solver=lbfgs; total time=   4.3s
[CV] END activation=identity, hidden_layer_sizes=5, solver=sgd; total time=  10.4s
[CV] END activation=identity, hidden_layer_sizes=5, solver=adam; total time=  10.8s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=lbfgs; total time=   4.9s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=sgd; total time=  10.4s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=adam; total time=  10.1s
[CV] END activation=tanh, hidden_layer_sizes=5, solver=lbfgs; total time=   5.1s
[CV] END ..activation=tanh, hidden_layer_sizes=5, solver=sgd; total time=   9.8s
[CV] END .activation=tanh, hidden_layer_sizes=5, solver=adam; total time=  10.2s
[CV] END activation=relu, hidden_layer_sizes=5, solver=lbfgs; total time=   5.1s
[CV] END ..activation=relu, hidden_layer_sizes=5, solver=sgd; total time=   9.8s
[CV] END .activation=relu, hidden_layer_sizes=5, solver=adam; total time=  10.6s
Score:  nan
Parameters:  {'activation': 'identity', 'hidden_layer_sizes': 5, 'solver': 'lbfgs'}

accuracy score on test data:  0.5133333333333333
example of predicted output:  [0 1 0 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 0]
whereas actual output:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
Fitting 1 folds for each of 12 candidates, totalling 12 fits
[CV] END activation=identity, hidden_layer_sizes=5, solver=lbfgs; total time=   0.1s
[CV] END activation=identity, hidden_layer_sizes=5, solver=sgd; total time=   2.6s
[CV] END activation=identity, hidden_layer_sizes=5, solver=adam; total time=   1.5s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=lbfgs; total time=   0.3s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=sgd; total time=   2.6s
[CV] END activation=logistic, hidden_layer_sizes=5, solver=adam; total time=   2.7s
[CV] END activation=tanh, hidden_layer_sizes=5, solver=lbfgs; total time=   0.1s
[CV] END ..activation=tanh, hidden_layer_sizes=5, solver=sgd; total time=   2.5s
[CV] END .activation=tanh, hidden_layer_sizes=5, solver=adam; total time=   2.4s
[CV] END activation=relu, hidden_layer_sizes=5, solver=lbfgs; total time=   0.1s
[CV] END ..activation=relu, hidden_layer_sizes=5, solver=sgd; total time=   2.7s
[CV] END .activation=relu, hidden_layer_sizes=5, solver=adam; total time=   1.3s
Score:  0.6407766990291263
Parameters:  {'activation': 'logistic', 'hidden_layer_sizes': 5, 'solver': 'lbfgs'}
```

1.2.2 MLP PREDICTION EXAMPLE

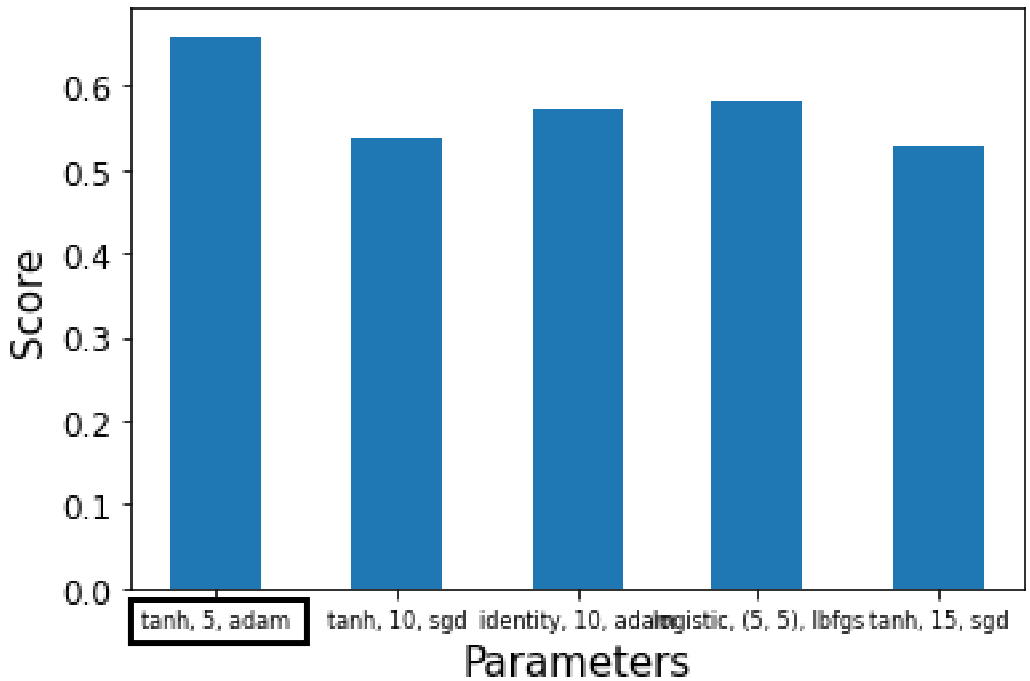
```
Score:  0.901
Parameters:  {'activation': 'tanh', 'hidden_layer_sizes': 15, 'solver': 'sgd'}
x_values:  ['tanh, 10, sgd', 'tanh, (5, 10), adam', 'logistic, (5, 5), sgd', 'relu, 5, lbfgs', 'identity, 10, sgd']
y_values:  [0.88, 0.793, 0.282, 0.641, 0.89]

accuracy score on test data:  0.86
example of predicted output:  [9 0 2 5 1 9 7 8 1 0 4]
whereas actual output:  [9, 0, 2, 5, 1, 9, 7, 8, 1, 0, 4]
```

```
Score:  0.6593406593406593
Parameters:  {'activation': 'tanh', 'hidden_layer_sizes': 5, 'solver': 'adam'}
x_values:  ['tanh, 5, adam', 'tanh, 10, sgd', 'identity, 10, adam', 'logistic, (5, 5), lbfgs', 'tanh, 15, sgd']
y_values:  [0.6593406593406593, 0.5384615384615384, 0.5714285714285714, 0.5824175824175825, 0.5274725274725275]

accuracy score on test data:  0.5133333333333333
example of predicted output:  [0 1 0 1 0 1 1 0 1 0 0 1 0 0]
whereas actual output:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0]
```

1.3 VISUALISATION OF ACCURACY RANGE OF 1ST DATASET



1.4 VISUALISATION OF ACCURACY RANGE OF 2ND DATASET



2. SVM ALGORITHM

2.1 STEPS

1) Import libraries and dataset

```
from samples import producedata
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, ShuffleSplit
import warnings
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import PredefinedSplit
# loading datasets

# handwritten numerical face
ocr_training_samples = 5000
ocr_test_samples = 1000
n_validation = 1000
digits_train_data, digits_train_labels = producedata(ocr_training_samples, "data/digitdata/trainingimages","data/di
digits_test_data, digits_test_labels  = producedata(ocr_test_samples, "data/digitdata/testimages", "data/digitdata/
digits_val_data, digits_val_labels = producedata(n_validation, "data/digitdata/validationimages", "data/digitdata/v


# edge image: classify as face or no face
face_training_samples=451
face_test_samples=150
face_val_samples=200
face_train_data, face_train_labels = producedata(face_training_samples,"data/facedata/facedatatrain", "data/facedat
face_test_data, face_test_labels = producedata(face_test_samples,"data/facedata/facedatatest", "data/facedata/faced
face_val_data, face_val_labels = producedata(face_val_samples, "data/facedata/facedatavalidation", "data/facedata/f
#print("lucy in the sky: ", face_val_data)
#print("with diamonds: ", face_val_labels)
```

2) Classify SVM (main)

3) Changing hyperparameters

Hyperparameters, C, kernel, and gamma were changed using GridSearchCV where we custom split the test and validation dataset with the ones provided in ‘data’ folder. Alternatively, we could use the ShuffleSplit defined in MLP. After plotting the score results, we use the best parameters to train and test the data, and finally we output an example prediction and label.

```python
    param_grid = {
    "C":[0.1, 1, 10, 100],
    "kernel":['rbf','linear','poly','sigmoid'],
    "gamma":[0.1,'auto','scale'] }
    svc = svm.SVC()
    gridSearch = GridSearchCV(svc, param_grid, cv=cv, #splits
                              refit = True, verbose=3, return_train_score=True,
                              n_jobs=-1)
    gridSearch.fit(train_data, train_labels) #all data
    #print("cv_results_: ", gridSearch.cv_results_)

    print('Score: ', gridSearch.best_score_)
    print('Parameters: ', gridSearch.best_params_)
    # then, we plot the parameter vs the score (uncomment the next line)
    plot_grid_search(gridSearch.cv_results_)


    # train, test accuracy
    c = gridSearch.best_params_['C']
    k = gridSearch.best_params_['kernel']
    g = gridSearch.best_params_['gamma']
    svc = SVC(C=c, kernel=k, gamma=g,
              random_state=1)
    fit = svc.fit(train_data, train_labels)
    labels_pred = fit.predict(test_data)
    #print("svc score: ", svc.score(test_data, test_labels))
    print("\naccuracy score on test data: ", accuracy_score(test_labels, labels_pred))

    rand = random.randint(0, 100) #output any random row to see
    print("example of predicted output: ", labels_pred[:rand]) # predicted digits or faces OUTPUT
    print("whereas actual output: ", test_labels[:rand], "\n") #actual label
```

```python
classify_svm(digits_train_data, digits_test_data, digits_train_labels, digits_test_labels
             digits_val_data, digits_val_labels)
classify_svm(face_train_data, face_test_data, face_train_labels, face_test_labels,
             face_val_data, face_val_labels)
```

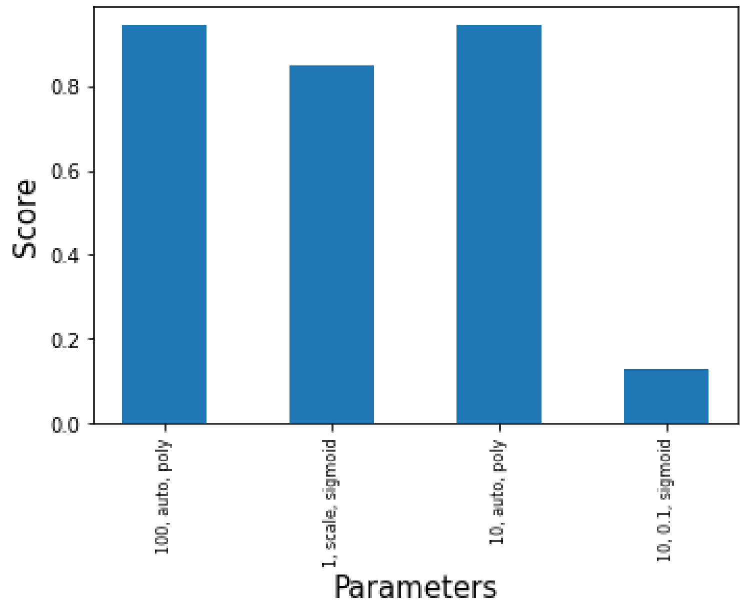
2.2 OUTPUT OF 1ST DATASET

```
Fitting 1 folds for each of 48 candidates, totalling 48 fits
Score:  0.964
Parameters:  {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
x_values:  ['100, auto, poly', '1, scale, sigmoid', '10, auto, poly', '10, 0.1, sigmoid', '1, scale,
sigmoid']
y_values:  [0.943, 0.85, 0.944, 0.127, 0.85]

accuracy score on test data:  0.942
example of predicted output:  [9 0 2 5 1 9 7 8 1 0 4 1]
whereas actual output:  [9, 0, 2, 5, 1, 9, 7, 8, 1, 0, 4, 1]
```

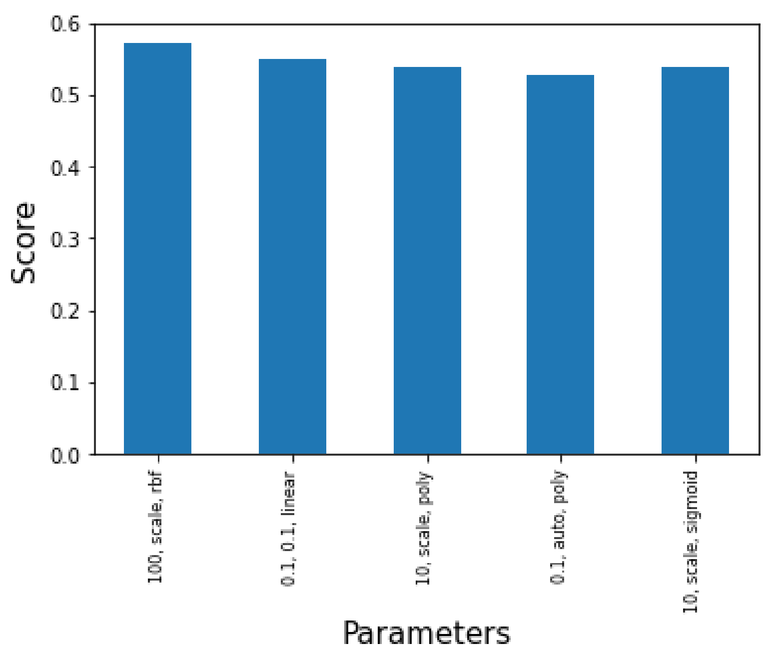
2.3 OUTPUT OF 2ND DATASET

```
Fitting 1 folds for each of 48 candidates, totalling 48 fits
Score:  0.5824175824175825
Parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
x_values:  ['100, scale, rbf', '0.1, 0.1, linear', '10, scale, poly', '0.1, auto, poly', '10, scale,
sigmoid']
y_values:  [0.5714285714285714, 0.5494505494505495, 0.5384615384615384, 0.5274725274725275,
0.5384615384615384]

accuracy score on test data:  0.5333333333333333
example of predicted output:  [0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
whereas actual output:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1,
1, 0, 0]
```

2.4 VISUALISATION OF ACCURACY RANGE OF 1ST DATASET

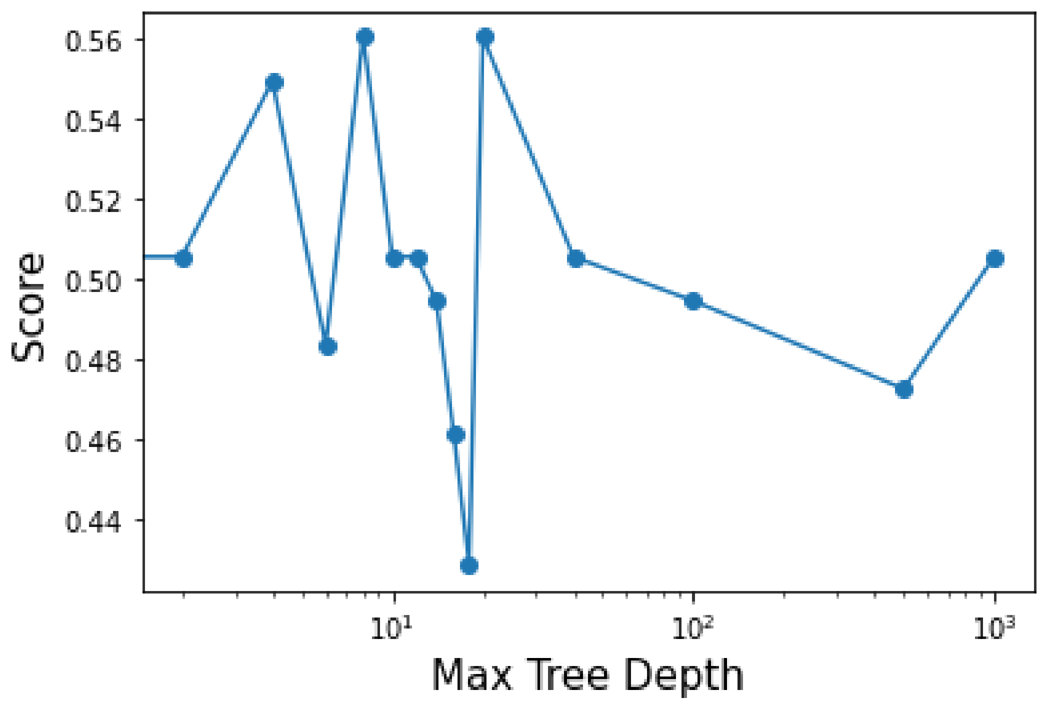


2.5 VISUALISATION OF ACCURACY RANGE OF 2ND DATASET



3. DECISION TREES

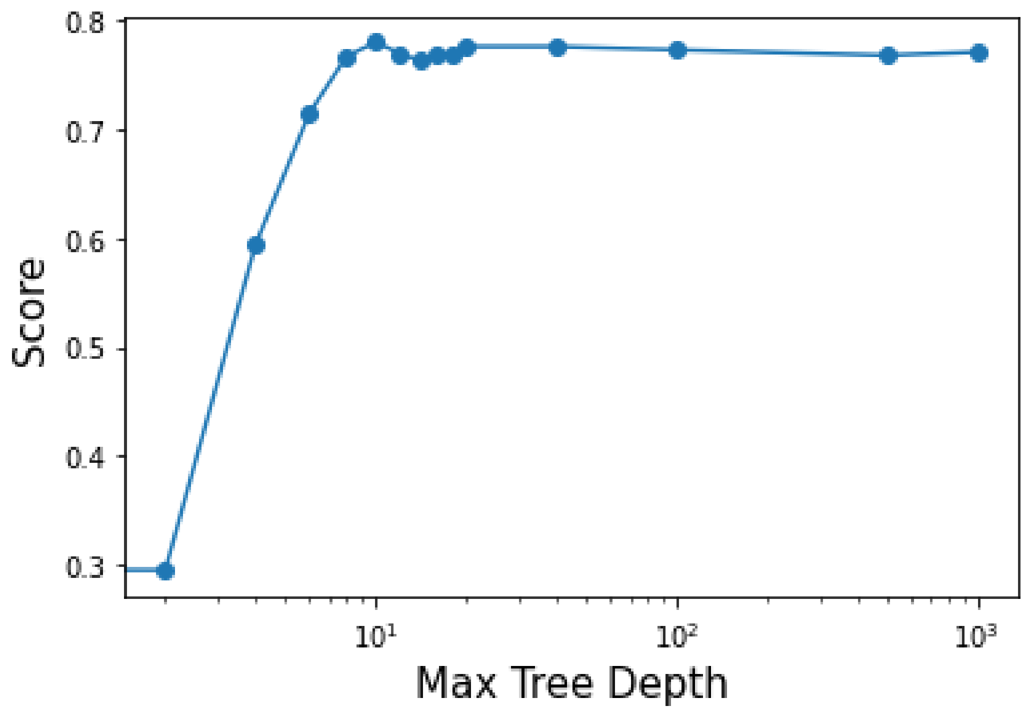
3.1 FACES




```
Fitting 1 folds for each of 15 candidates, totalling 15 fits
Score:  0.5604395604395604
Parameters:  {'max_depth': 8}
plot x:  [-1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 40, 100, 500, 1000]
plot y:  [0.49450549 0.50549451 0.54945055 0.48351648 0.56043956 0.50549451
 0.50549451 0.49450549 0.46153846 0.42857143 0.56043956 0.50549451
 0.49450549 0.47252747 0.50549451]

prediction accuracy on test data:  50.66666666666667 %
example of predicted output:  [0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 1
 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 1 1]
whereas actual output:  [1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1,
1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 1, 0, 0]
```

```
|--- feature_1439 <= 1.00
|   |--- feature_2241 <= 1.00
|   |   |--- feature_885 <= 1.00
|   |   |   |--- feature_1239 <= 1.00
|   |   |   |   |--- feature_1920 <= 1.00
|   |   |   |   |   |--- feature_3442 <= 1.00
|   |   |   |   |   |   |--- feature_1191 <= 1.00
|   |   |   |   |   |   |   |--- feature_2779 <= 1.00
|   |   |   |   |   |   |   |   |--- feature_1506 <= 1.00
|   |   |   |   |   |   |   |   |   |--- feature_3088 <= 1.00
|   |   |   |   |   |   |   |   |   |   |--- feature_2041 <= 1.00
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 10
|   |   |   |   |   |   |   |   |   |   |--- feature_2041 >  1.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_3088 >  1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- feature_1506 >  1.00
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- feature_2779 >  1.00
|   |   |   |   |   |   |   |   |--- feature_2516 <= 1.00
|   |   |   |   |   |   |   |   |   |--- feature_3032 <= 1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_3032 >  1.00
|   |   |   |   |   |   |   |   |   |   |--- feature_1572 <= 1.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |   |--- feature_1572 >  1.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- feature_2516 >  1.00
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_1191 >  1.00
|   |   |   |   |   |   |   |--- feature_3027 <= 1.00
|   |   |   |   |   |   |   |   |--- feature_1740 <= 1.00
|   |   |   |   |   |   |   |   |   |--- feature_522 <= 1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_522 >  1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- feature_1740 >  1.00
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- feature_3027 >  1.00
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_3442 >  1.00
|   |   |   |   |   |   |--- feature_2325 <= 1.00
|   |   |   |   |   |   |   |--- feature_209 <= 1.00
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_209 >  1.00
```
fitting

3.2 DIGITS




```
Fitting 1 folds for each of 15 candidates, totalling 15 fits
Score:  0.78
Parameters:  {'max_depth': 10}
plot x:  [-1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 40, 100, 500, 1000]
plot y:  [0.762 0.295 0.595 0.714 0.767 0.78  0.77  0.763 0.77  0.769 0.776 0.776
 0.773 0.768 0.771]

prediction accuracy on test data:  72.7 %
example of predicted output:  [9 0 2 3 1 9 7 8 1 0 4 1 9 9 0 1 2 6 8 1 3 7 5 3 4 1 8 1 5 8 1 7 0 6 0 6 2
 1 1 7 1 5 5 4 6 5 5 5]
whereas actual output:  [9, 0, 2, 5, 1, 9, 7, 8, 1, 0, 4, 1, 7, 9, 6, 4, 2, 6, 8, 1, 3, 7, 5, 4, 4, 1, 8,
1, 3, 8, 1, 2, 5, 8, 0, 6, 2, 1, 1, 7, 1, 5, 3, 4, 6, 9, 5, 0]
```

```
|   |   |   |--- feature_359 <= 0.50
|   |   |   |   |--- feature_513 <= 0.50
|   |   |   |   |   |--- feature_293 <= 0.50
|   |   |   |   |   |   |--- feature_479 <= 0.50
|   |   |   |   |   |   |   |--- feature_214 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_440 <= 1.50
|   |   |   |   |   |   |   |   |   |--- class: 8
|   |   |   |   |   |   |   |   |--- feature_440 >  1.50
|   |   |   |   |   |   |   |   |   |--- feature_520 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- feature_520 >  0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_489 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 7
|   |   |   |   |   |   |   |   |   |   |--- feature_489 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |   |--- feature_214 >  0.50
|   |   |   |   |   |   |   |   |--- feature_511 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_605 <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |   |--- feature_605 >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 6
|   |   |   |   |   |   |   |   |--- feature_511 >  0.50
|   |   |   |   |   |   |   |   |   |--- feature_297 <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- feature_412 <= 1.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 9
|   |   |   |   |   |   |   |   |   |   |--- feature_412 >  1.00
|   |   |   |   |   |   |   |   |   |   |   |--- class: 5
|   |   |   |   |   |   |   |   |   |--- feature_297 >  1.50
|   |   |   |   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_479 >  0.50
|   |   |   |   |   |   |   |--- feature_458 <= 1.00
|   |   |   |   |   |   |   |   |--- class: 6
|   |   |   |   |   |   |   |--- feature_458 >  1.00
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_293 >  0.50
|   |   |   |   |   |   |--- feature_413 <= 0.50
|   |   |   |   |   |   |   |--- feature_452 <= 0.50
|   |   |   |   |   |   |   |   |--- feature_580 <= 0.50
|   |   |   |   |   |   |   |   |   |--- feature_551 <= 1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 4
|   |   |   |   |   |   |   |   |   |--- feature_551 >  1.00
|   |   |   |   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |   |   |--- feature_580 >  0.50
|   |   |   |   |   |   |   |   |   |--- feature_488 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 5
|   |   |   |   |   |   |   |   |   |--- feature_488 >  0.50
```