Milestone -1

Team members :-

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

Contents

1. KNN ALGORITHM

1.1 STEPS

- **Import necessary libraries:**

```
from samples import *
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
from importlib import reload
```

- **Load digit data (training ,test):**
- **Producedata() in samples.py load digit data by default**
- **The provided data has 5000 training samples and 1000 test samples**

```
training_samples = 5000
tst_samples = 1000
features, targets = producedata(training_samples)

tst_features_path = "data/digitdata/testimages"
tst_target_path = "data/digitdata/testlabels"


tst_features, tst_targets = producedata(tst_samples,tst_features_path,tst_target_path)
```

- **Define 2 functions to:**
- **train a model given certain K as a hyperparameter**

```
from sklearn import metrics

def train_knn_classifier(features, target, k=5):
    model = KNeighborsClassifier(n_neighbors=k)  # choosing the hyper parameter K = xxx
    # print(k)

    # Train the model using the training sets
    model.fit(features, target)  # feeding data to the model
    return model
```

- **Predict the output of given test data**

```
def predict_output(model, tst_data=[[0, 2]]):
    # Predict Output
    predicted = model.predict(tst_data)


    return predicted
```

- **Define a function to try different values of K ranging from**

(3 - $\sqrt{\boldsymbol{no\ of\ samples}}$) and measure the corresponding accuracy

```
def all_the_work(i=3,title=None,xlbl=None,ylbl=None):

    x_points = []
    y_points = []

    while i <= (training_samples ** 0.5):                                      # no of k neighbours should be odd

        trained_model = train_knn_classifier(features, targets, i)          # train model and predict output
        predicted_targets = predict_output(trained_model, tst_features)     #predict the output
        acc = metrics.accuracy_score(tst_targets, predicted_targets)        # Model Accuracy, how often is the classifier correct?
        #visualization
        x_points.append(i)
        y_points.append(acc)
        print("k :",i,"\tAccuracy:",acc)
        i+=2

    print(list(zip(x_points,y_points)))
    plt.plot(x_points, y_points,marker='o')
    plt.title(title,fontsize = 20)
    plt.xlabel(xlbl,fontsize = 20)
    plt.ylabel(ylbl,fontsize = 20)
    plt.show()
all_the_work(title='knn',xlbl='k-neighbours',ylbl='accuracy')
"""KNN DOESNOT PERFORM PROPERLY WITH LARGE NUMBER OF FEATURES"""
```

1.2 OUTPUT AND VISUALISATION OF 1ST DATASET

```
k : 17   Accuracy: 0.878
k : 19   Accuracy: 0.874
k : 21   Accuracy: 0.872
k : 23   Accuracy: 0.878
k : 25   Accuracy: 0.876
k : 27   Accuracy: 0.87
k : 29   Accuracy: 0.869
k : 31   Accuracy: 0.864
k : 33   Accuracy: 0.864
k : 35   Accuracy: 0.86
k : 37   Accuracy: 0.856
k : 39   Accuracy: 0.857
k : 41   Accuracy: 0.852
k : 43   Accuracy: 0.851
k : 45   Accuracy: 0.851
k : 47   Accuracy: 0.848
k : 49   Accuracy: 0.845
k : 51   Accuracy: 0.843
```

Figure 1:knn on digits

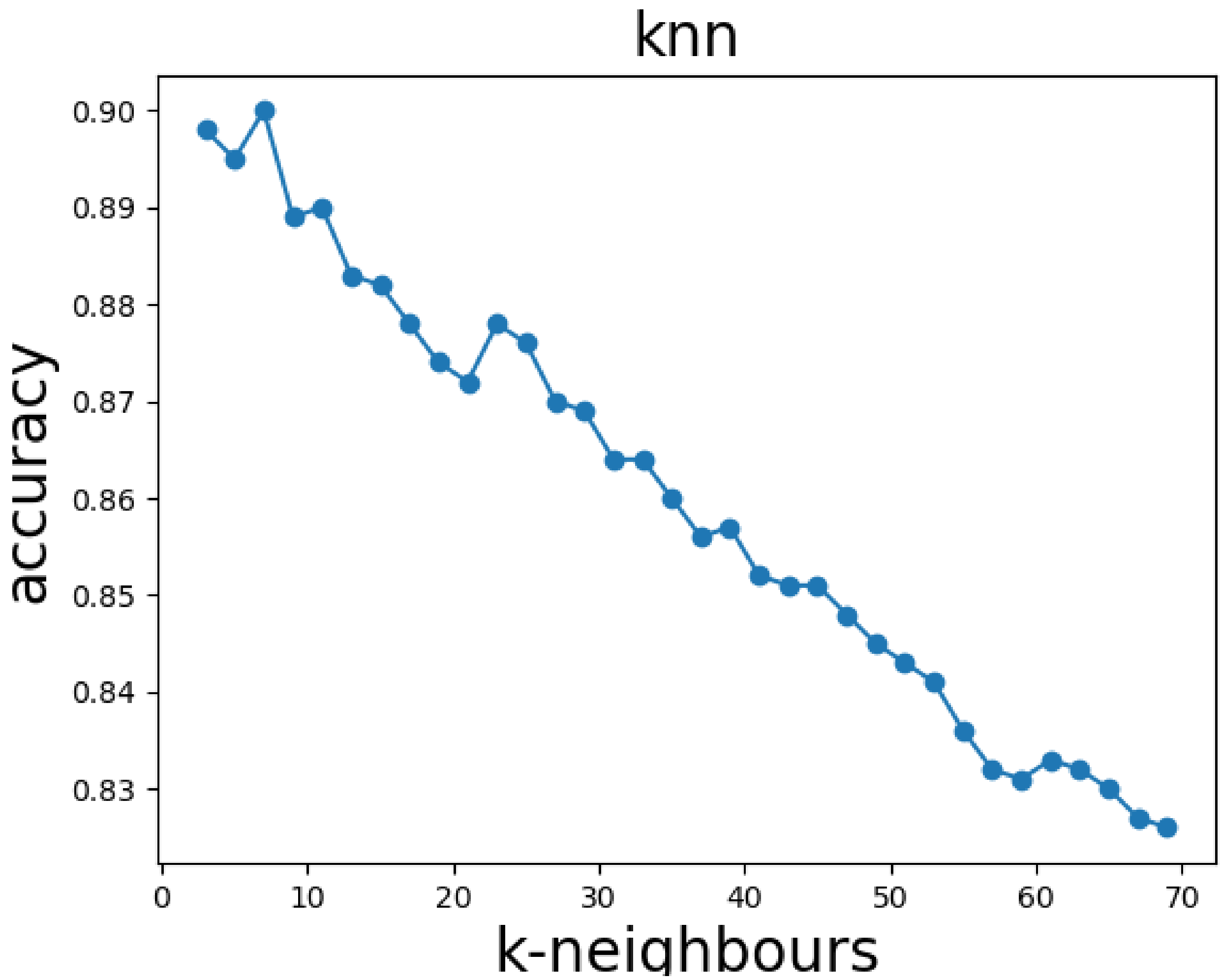

knn
accuracy
0.90
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0
10
20
30
40
50
60
70
k-neighbours

Figure 2

- **Repeat the same process on face data images**
- **We need only to change number of training and test samples and file paths**

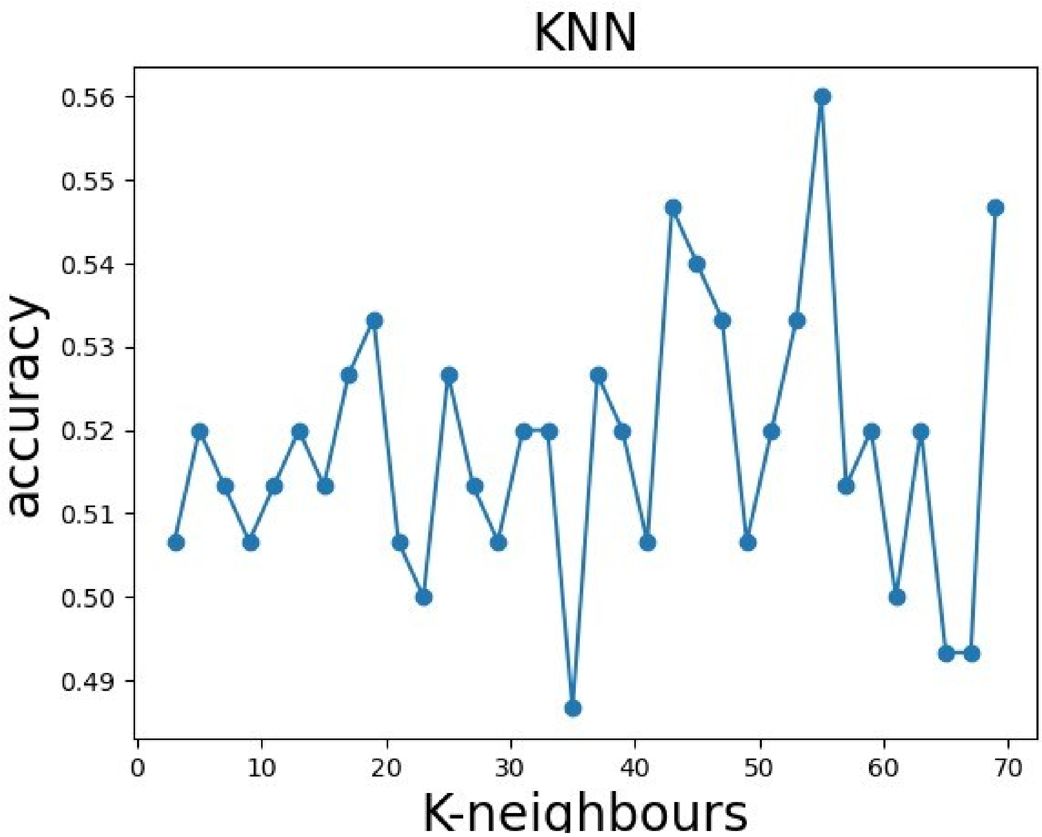
```
features,targets = producedata(451,"data/facedata/facedatatrain","data/facedata/facedatatrainlabels",60)      #loading training data

tst_features,tst_targets = producedata(150,"data/facedata/facedatatest","data/facedata/facedatatestlabels",60)
print("\n---------------------------------")
all_the_work()
```

1.3 OUTPUT AND VISUALISATION OF 2ND DATASET

```
k : 11  Accuracy: 0.5133333333333333
k : 13  Accuracy: 0.52
k : 15  Accuracy: 0.5133333333333333
k : 17  Accuracy: 0.5266666666666666
k : 19  Accuracy: 0.5333333333333333
k : 21  Accuracy: 0.5066666666666667
k : 23  Accuracy: 0.5
k : 25  Accuracy: 0.5266666666666666
k : 27  Accuracy: 0.5133333333333333
k : 29  Accuracy: 0.5066666666666667
k : 31  Accuracy: 0.52
k : 33  Accuracy: 0.52
k : 35  Accuracy: 0.4866666666666667
k : 37  Accuracy: 0.5266666666666666
k : 39  Accuracy: 0.52
k : 41  Accuracy: 0.5066666666666667
k : 43  Accuracy: 0.5466666666666666
k : 45  Accuracy: 0.54
k : 47  Accuracy: 0.5333333333333333
k : 49  Accuracy: 0.5066666666666667
k : 51  Accuracy: 0.52
```

Figure 3: knn on faces

Figure 4



- **We choose the k-value of the highest accuracy**

(7) in digits classification
(55) in face classification

2. NAÏVE BAYES ALGORITHM

2.1 STEPS

- **Import significant modules**

```
from samples import *
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
import matplotlib.pyplot as plt
```

- **Load digit data (training ,test):**
- **Producedata() in samples.py load digit data by default**
- **The provided data has 5000 training samples and 1000 test samples**

```
training_samples = 5000
tst_samples = 1000
features, targets = producedata(training_samples)


#----------------------                     ------
```

```
tst_features_path = "data/digitdata/testimages"
tst_target_path =  "data/digitdata/testlabels"


tst_features, tst_targets  = producedata(tst_samples,tst_features_path,tst_target_path)
```

- **Define a function to:**
- **train a gaussian naïve bayes model with a certain hyperparameter "variance smoothing"**
- **predict the output of test data**
- **measure the accuracy**

- **Visualize the accuracy for different values of the hyperparameter**

```python
def all_work(var=0.01):

    x_points= []
    y_points = []
    norm = 1/var
    for i in range(int(var*norm),30,1):


        gnb = GaussianNB(var_smoothing=(i/norm))
        gnb.fit(features, targets)
        y_pred = gnb.predict(tst_features)

        x_points.append(i/norm)
        y_points.append(metrics.accuracy_score(tst_targets, y_pred) * 100)
        print("var = ",i/100,"\tGaussian Naive Bayes model accuracy(in %):", y_points[i-1])
    print(x_points,"\n",y_points)



    plt.title("naive bayes", fontsize=20)
    plt.xlabel('sigma', fontsize=20)
    plt.ylabel('accuracy', fontsize=20)

    plt.plot(x_points,y_points,marker='o')

    plt.show()
```

- **Repeat the same process on face data images**
- **We need only to change number of training and test samples and file paths**

```
all_work()

# comparing actual response values (y_test) with predicted response values (y_pred)
face_training_samples=451
face_tst_samples=150

features,targets = producedata(face_training_samples,"data/facedata/facedatatrain","data/facedata/facedatatrainlabels",60)      #loading training data

tst_features,tst_targets = producedata(face_tst_samples,"data/facedata/facedatatest","data/facedata/facedatatestlabels",60)
print("\n-----------------------------------")
all_work(0.5)
```

2.2 OUTPUT AND VISUALISATION OF 1ST DATASET

```
var =  0.01     Gaussian Naive Bayes model accuracy(in %): 69.8
var =  0.02     Gaussian Naive Bayes model accuracy(in %): 71.8
var =  0.03     Gaussian Naive Bayes model accuracy(in %): 72.3
var =  0.04     Gaussian Naive Bayes model accuracy(in %): 72.7
var =  0.05     Gaussian Naive Bayes model accuracy(in %): 72.8
var =  0.06     Gaussian Naive Bayes model accuracy(in %): 72.6
var =  0.07     Gaussian Naive Bayes model accuracy(in %): 72.6
var =  0.08     Gaussian Naive Bayes model accuracy(in %): 72.39999999999999
var =  0.09     Gaussian Naive Bayes model accuracy(in %): 72.3
var =  0.1  Gaussian Naive Bayes model accuracy(in %): 71.5
var =  0.11     Gaussian Naive Bayes model accuracy(in %): 71.39999999999999
var =  0.12     Gaussian Naive Bayes model accuracy(in %): 71.5
var =  0.13     Gaussian Naive Bayes model accuracy(in %): 71.39999999999999
var =  0.14     Gaussian Naive Bayes model accuracy(in %): 71.39999999999999
var =  0.15     Gaussian Naive Bayes model accuracy(in %): 71.1
var =  0.16     Gaussian Naive Bayes model accuracy(in %): 70.8
var =  0.17     Gaussian Naive Bayes model accuracy(in %): 70.39999999999999
var =  0.18     Gaussian Naive Bayes model accuracy(in %): 70.3
var =  0.19     Gaussian Naive Bayes model accuracy(in %): 70.3
var =  0.2  Gaussian Naive Bayes model accuracy(in %): 70.3
var =  0.21     Gaussian Naive Bayes model accuracy(in %): 70.3
var =  0.22     Gaussian Naive Bayes model accuracy(in %): 70.3
var =  0.23     Gaussian Naive Bayes model accuracy(in %): 70.1
```

Figure 5: NB on digits

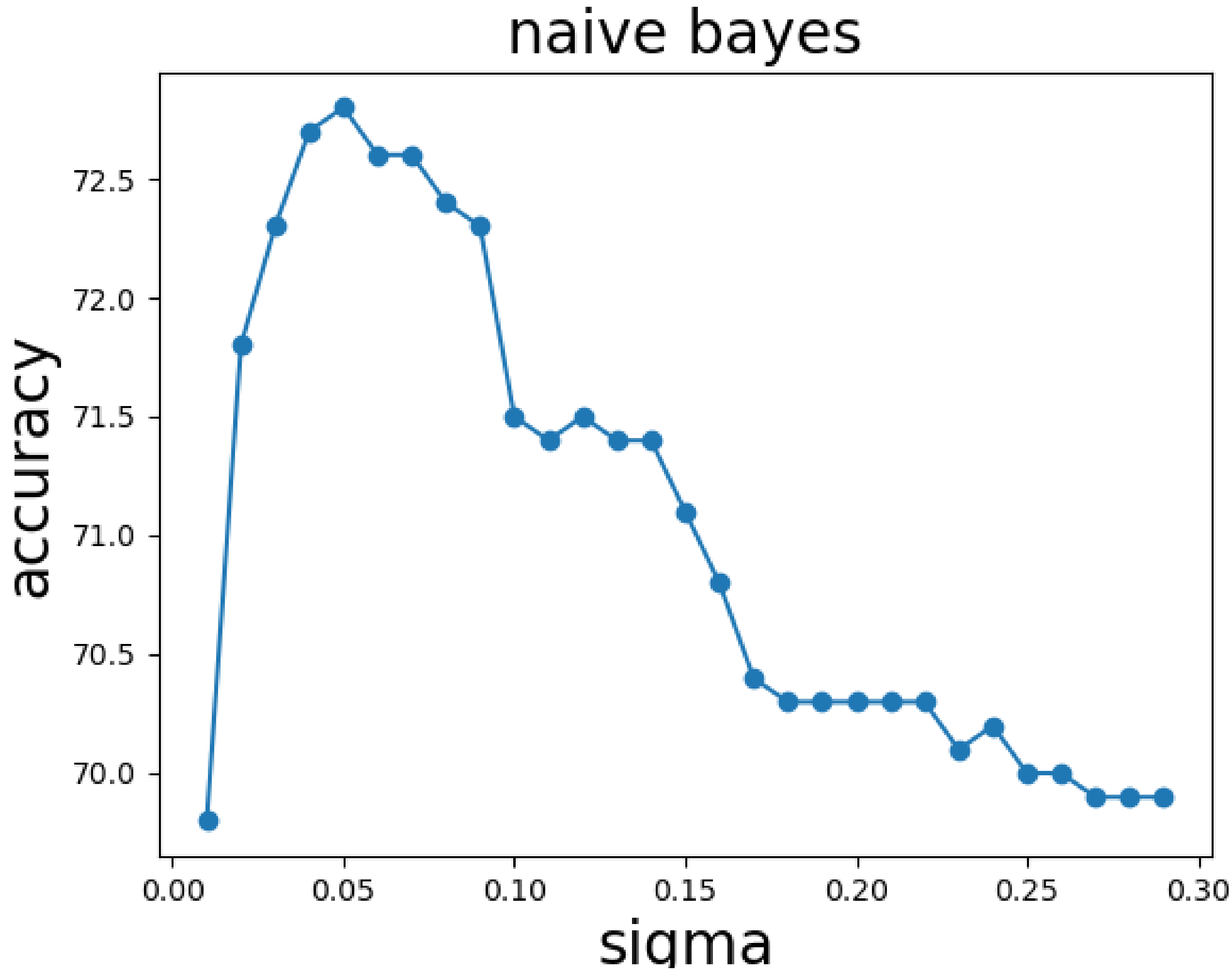

naive bayes
accuracy
72.5
72.0
71.5
71.0
70.5
70.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
sigma

Figure 6

2.3 OUTPUT AND VISUALISATION OF 2ND DATASET

```
var =  0.01      Gaussian Naive Bayes model accuracy(in %): 50.66666666666667
var =  0.02      Gaussian Naive Bayes model accuracy(in %): 49.333333333333336
var =  0.03      Gaussian Naive Bayes model accuracy(in %): 50.66666666666667
var =  0.04      Gaussian Naive Bayes model accuracy(in %): 50.66666666666667
var =  0.05      Gaussian Naive Bayes model accuracy(in %): 52.0
var =  0.06      Gaussian Naive Bayes model accuracy(in %): 52.0
var =  0.07      Gaussian Naive Bayes model accuracy(in %): 52.0
var =  0.08      Gaussian Naive Bayes model accuracy(in %): 51.33333333333333
var =  0.09      Gaussian Naive Bayes model accuracy(in %): 51.33333333333333
var =  0.1  Gaussian Naive Bayes model accuracy(in %): 52.666666666666664
var =  0.11      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.12      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.13      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.14      Gaussian Naive Bayes model accuracy(in %): 54.0
var =  0.15      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.16      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.17      Gaussian Naive Bayes model accuracy(in %): 53.333333333333336
var =  0.18      Gaussian Naive Bayes model accuracy(in %): 54.0
var =  0.19      Gaussian Naive Bayes model accuracy(in %): 52.666666666666664
var =  0.2  Gaussian Naive Bayes model accuracy(in %): 52.666666666666664
```

Figure 7: NB on faces

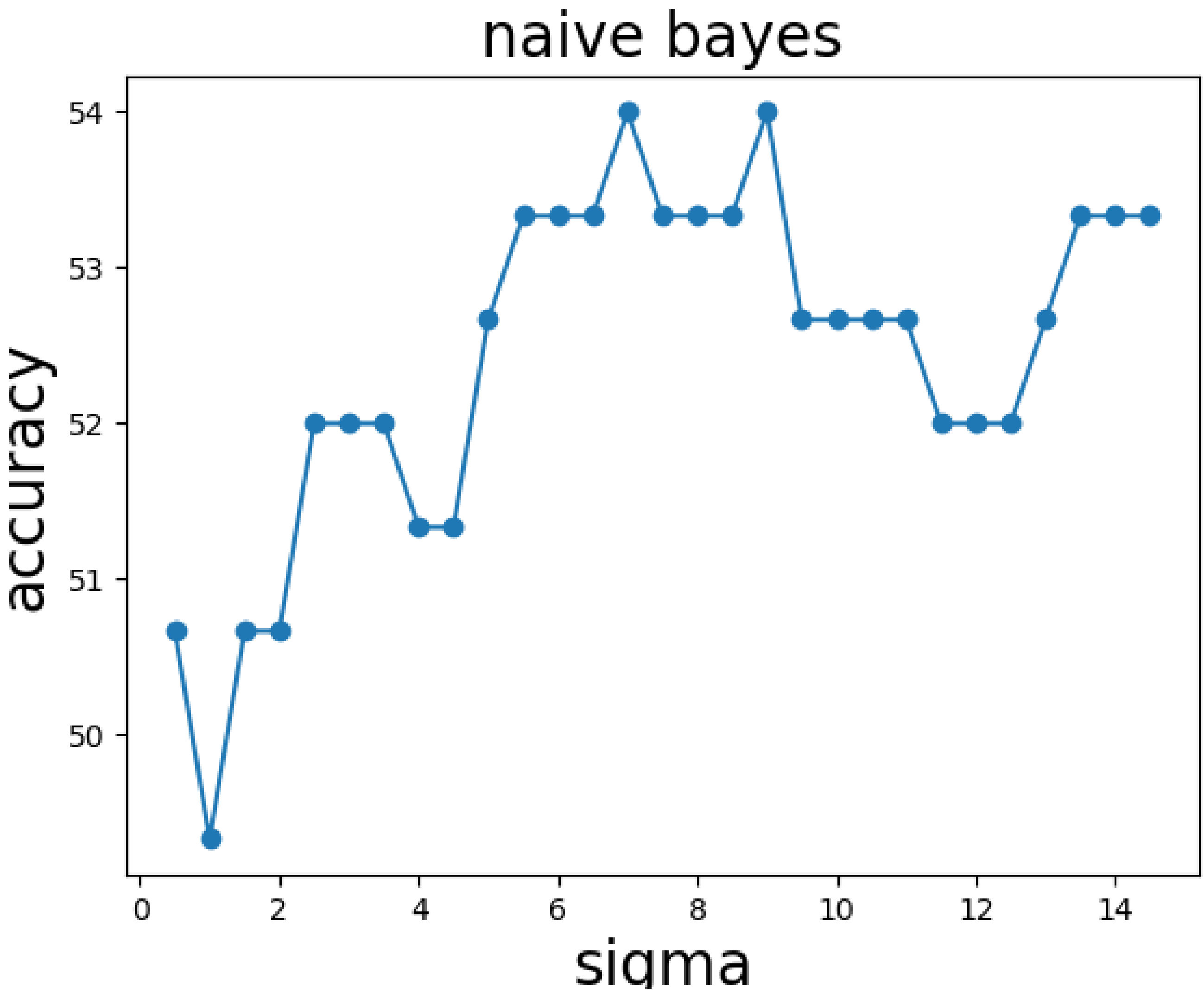


Figure 8

- **We choose variance smoothing values of highest accuracy as a hyperparameter**

(0.05) in digits

(7 or 9) in faces