CSE381 Introduction to Machine Learning

Project Documentation

Submitted to:

Prof. Dr. Mahmoud Khalil

Eng. Mahmoud Soheil

Submitted by:

Youssef George Fouad		19P9824
Anthony Amgad Fayek		19P9880
Kerollos Wageeh Youssef		19P3468
CESS	G2 S3	Senior-1

Table of Contents

1. DATA PREPROCESSING

1.1. Importing Needed Libraries

Start by importing 2 given files “util” and samples. In addition, import libraries for classification and results analysis functions.

```
import util, samples
import numpy as np

# For classification
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier

# For results analysis and visualization
from sklearn import metrics
import matplotlib.pyplot as plt
```

1.2. Loading All Datasets

We are performing our classification training and testing tasks on two datasets: digits dataset and faces dataset. Each of them is split into training set on which the classifier model will be trained before being tested on the test set.
In addition to loading the images data itself, we load the labels of each image, which we use to train the model in case of train dataset and calculate the model accuracy when testing it.

```
digits_train_imgs = samples.loadDataFile("digitdata/trainingimages", 5000,28,28)
digits_train_labels = samples.loadLabelsFile("digitdata/traininglabels", 5000)
digits_test_imgs = samples.loadDataFile("digitdata/testimages", 1000,28,28)
digits_test_labels = samples.loadLabelsFile("digitdata/testlabels", 1000)

# Faces Dataset
faces_train_imgs = samples.loadDataFile("facedata/facedatatrain", 451, 60, 70)
faces_train_labels = samples.loadLabelsFile("facedata/facedatatrainlabels", 451)
faces_test_imgs = samples.loadDataFile("facedata/facedatatest", 150, 60, 70)
faces_test_labels = samples.loadLabelsFile("facedata/facedatatestlabels", 150)
```

1.3. Transformation of Data

Images data of both datasets is originally in the form of ",", "+", and "#" characters. Our classification models only know how to deal with numbers, so we use "convertToInteger" function, implemented in "samples" file, which maps each of the symbols mentioned earlier to 0, 1, 2 respectively as its name suggests.

```
# Transform digits images
new_digits_train = np.array(samples.convertToInteger(digits_train_imgs)).reshape(5000, 28*28)
new_digits_test = np.array(samples.convertToInteger(digits_test_imgs)).reshape(1000, 28*28)

# Transform faces images
new_faces_train = np.array(samples.convertToInteger(faces_train_imgs)).reshape(451, 60*70)
new_faces_test = np.array(samples.convertToInteger(faces_test_imgs)).reshape(150, 60*70)
```

2. NAÏVE BAYES CLASSIFIER

2.1. Hyperparameters

In fact, the Naïve Bayes Classifier has no hyperparameters to set prior to the training process.

2.2. Classifying Digits Dataset

```
# Training
gnb = GaussianNB()
gnb_digits_model = gnb.fit(new_digits_train, digits_train_labels)
# Testing
y_pred_digits = gnb_digits_model.predict(new_digits_test)
```

The Naïve Bayes Classifier accuracy is calculated after testing it on the digits test dataset.

```
gnb_digits_accuracy = metrics.accuracy_score(y_pred_digits,digits_test_labels)*100
print(f"The model is {gnb_digits_accuracy}% accurate to digits test data")
```

Conclusion: This classifier only has 50.9% classification accuracy for the digits test set.

```
1  gnb_digits_accuracy = metrics.accuracy_score(y_pred_digits,digits_test_labels)*100
2  print(f"The model is {gnb_digits_accuracy}% accurate to digits test data")
[28]  ✓ 0.2s                                                                Python
...  The model is 50.9% accurate to digits test data
```

The accuracy of Naïve Bayes Classifier is 50.9% on digits test dataset.

2.3. Classifying Faces Dataset

```
# Training
gnb = GaussianNB()
gnb_faces_model = gnb.fit(new_faces_train, faces_train_labels)
# Testing
y_gnb_pred_faces = gnb_faces_model.predict(new_faces_test)
```

The Naïve Bayes Classifier accuracy is calculated after testing it on the faces test dataset.

```
gnb_faces_accuracy = metrics.accuracy_score(y_gnb_pred_faces,faces_test_labels)*100
print(f"The model is {gnb_faces_accuracy}% accurate to test data")
```

Conclusion: This classifier has 88% classification accuracy for the faces test set.

```
1  gnb_faces_accuracy = metrics.accuracy_score(y_gnb_pred_faces,faces_test_labels)*100
2  print(f"The model is {gnb_faces_accuracy}% accurate to test data")
[36]  ✓ 0.2s                                                                Python
...  The model is 88.0% accurate to test data
```

The accuracy of Naïve Bayes Classifier is 88% on faces test dataset.

3. K-NEAREST NEIGHBOURS

3.1. Hyperparameters

The KNN classifier has only 2 hyperparameters. Firstly, the number, K, nearest neighbors, that should be considered on classifying a new point. Secondly, distance calculating method between the points, and we will be considering Euclidean and Manhattan methods.

3.2. Classifying Digits Dataset

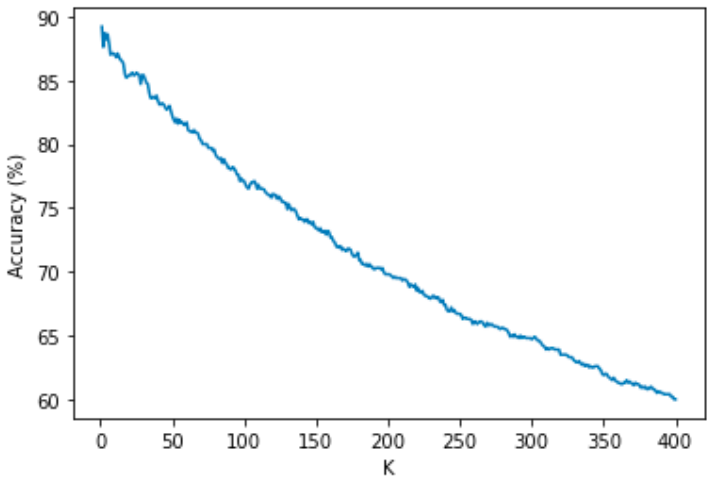
3.2.1. Manhattan Distance

To choose the K that gives us the best accuracy for our digits dataset while using the Manhattan distance, we will try all K values from 1 to 400 and draw a graph to check the best possible value for K.

```
x_digits = list(range(1, 401))
y_digits = []
# Loop for every K value
for i in x_digits:
    # Training
    knn = KNeighborsClassifier(n_neighbors=i, metric='manhattan')
    knn.fit(new_digits_train, digits_train_labels)
    #Testing
    y_knn_pred_digits = knn.predict(new_digits_test)
    # Calculate accuracy for each K
    val = metrics.accuracy_score(y_knn_pred_digits,digits_test_labels)*100
    y_digits.append(val)
# Plot Graph
plt.plot(x_digits, y_digits)
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.show()
```

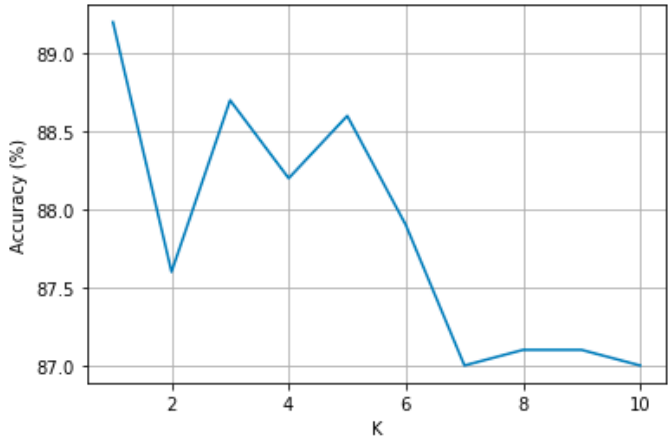
Resulting in:

As we can see: As the number of neighbors taken into consideration increases, the accuracy decreases.



Taking a closer look at K ranging 1:10 as it has the highest accuracy.

```
plt.plot(x_digits[:10], y_digits[:10])
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```



Clearly K = 3 and K = 5 have the highest accuracy percentage, as we ignore K = 1.

```
1  print(f"The model is {y_digits[2]}% accurate to test data at K = 3, using Manhattan distance.")
2  print(f"The model is {y_digits[4]}% accurate to test data at K = 5, using Manhattan distance.")
```

[56] ✓ 0.3s Python

```
The model is 88.7% accurate to test data at K = 3, using Manhattan distance.
The model is 88.6% accurate to test data at K = 5, using Manhattan distance.
```

Conclusion: Using Manhattan distance to classify digits dataset using KNN, K = 3 gives the highest possible accuracy on our test set = 88.7%

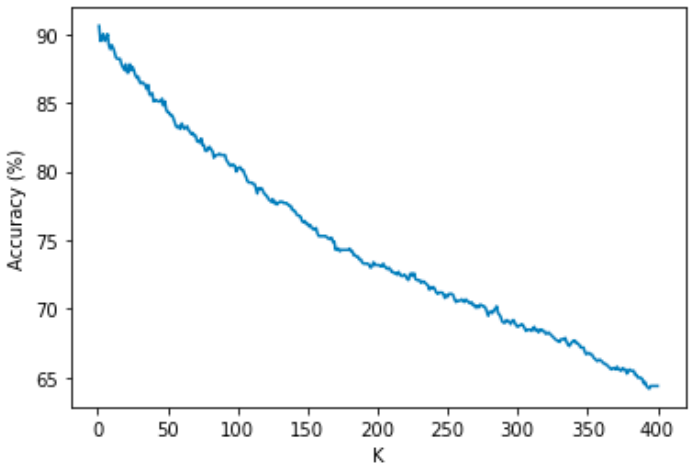
3.2.2. Euclidean Distance

To choose the K that gives us the best accuracy for our digits dataset while using the Euclidean distance, we will try all K values from 1 to 400 and draw a graph to check the best possible value for K.

```
x_digits = list(range(1, 401))
y_digits = []

# Loop for every K value
for i in x_digits:
    # Training
    knn = KNeighborsClassifier(n_neighbors=i, metric='euclidean')
    knn.fit(new_digits_train, digits_train_labels)

    #Testing
    y_knn_pred_digits = knn.predict(new_digits_test)

    # Calculate accuracy for each K
    val = metrics.accuracy_score(y_knn_pred_digits,digits_test_labels)*100
    y_digits.append(val)

# Plot Graph
plt.plot(x_digits, y_digits)
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.show()
```

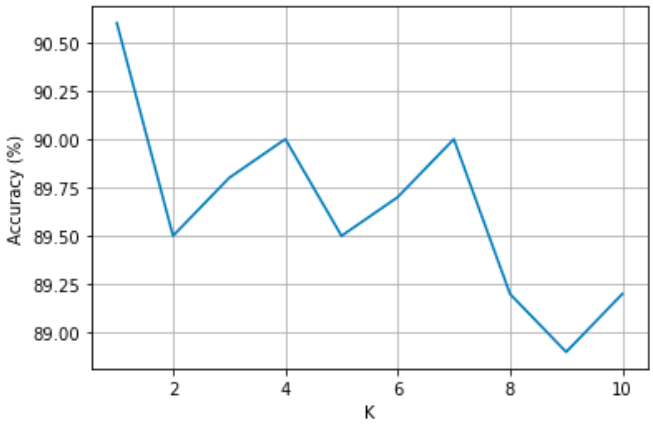
Resulting in:

As we can see: As the number of neighbors taken into consideration increases, the accuracy decreases.



Taking a closer look at K ranging 1:10 as it has the highest accuracy.

```
plt.plot(x_digits[:10], y_digits[:10])
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```



Clearly K = 4 and K = 7 have the highest accuracy percentages, as we ignore K = 1.

```
1  print(f"The model is {y_digits[3]}% accurate to test data at K = 4, using Euclidean distance.")
2  print(f"The model is {y_digits[6]}% accurate to test data at K = 7, using Euclidean distance.")
[64]  ✓ 0.5s                                                                          Python
...  The model is 90.0% accurate to test data at K = 4, using Euclidean distance.
     The model is 90.0% accurate to test data at K = 7, using Euclidean distance.
```

Conclusion: Using Euclidean distance to classify digits dataset using KNN, K = 7 and K = 4 both give the highest possible accuracy on our test set = 90.0%

Conclusion on KNN with digits dataset: To get the best possible accuracy while classifying digits test set with KNN, we shall use Euclidean distance and K = 4 or K = 7.

3.3. Classifying Faces Dataset

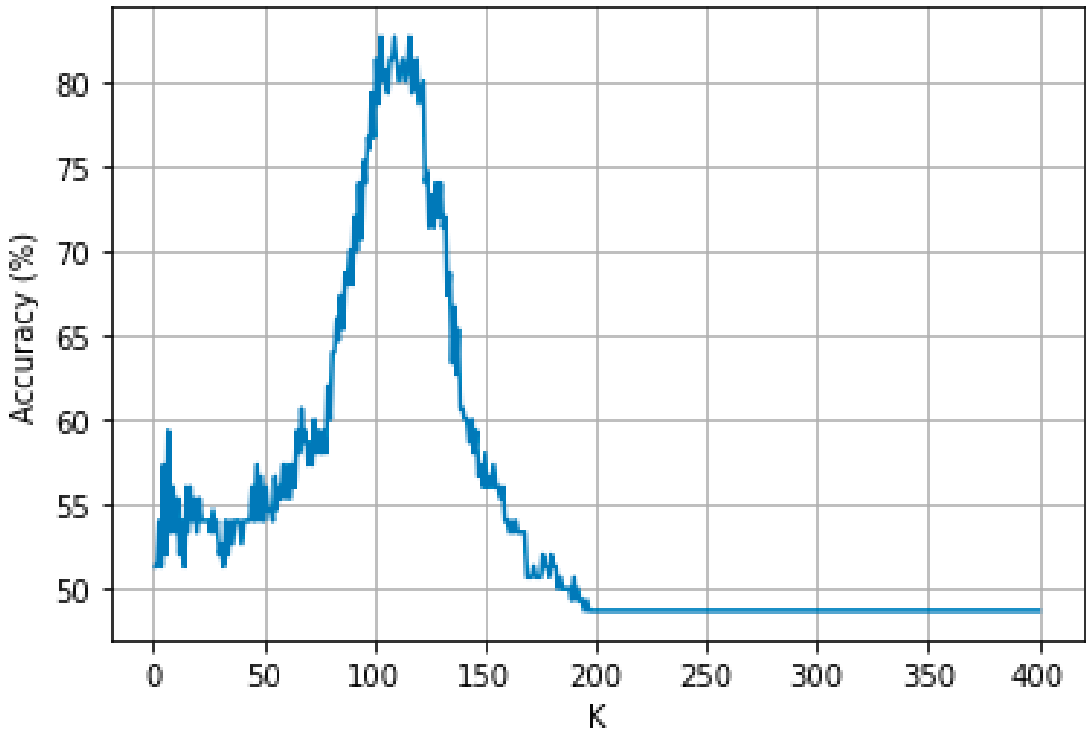
3.3.1. Manhattan Distance

To choose the K that gives us the best accuracy for our faces dataset while using the Manhattan distance, we will try all K values from 1 to 400 and draw a graph to check the best possible value for K.

```
x_faces = list(range(1, 401))
y_faces = []

# Loop for every K value
for i in x_faces:
    # Training
    knn = KNeighborsClassifier(n_neighbors=i, metric='manhattan')
    knn.fit(new_faces_train, faces_train_labels)

    # Testing
    y_knn_pred_faces = knn.predict(new_faces_test)

    # Calculate accuracy for each K
    val = metrics.accuracy_score(y_knn_pred_faces,faces_test_labels)*100
    y_faces.append(val)

plt.plot(x_faces, y_faces)
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```

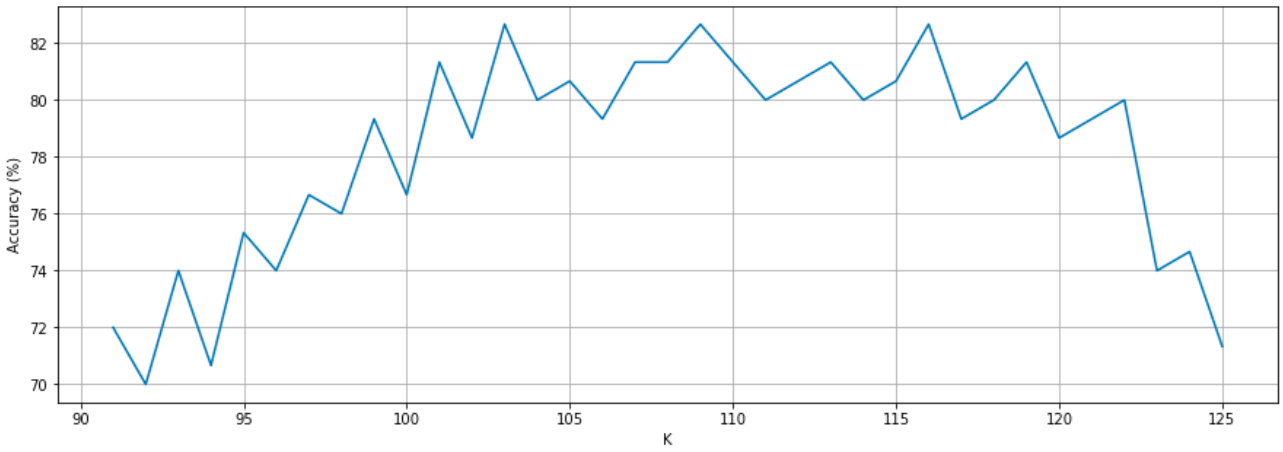
Resulting in:

As we can see, a bell-shaped curve was formed with the peak between K = 90 to K = 125.



Taking a closer look at K ranging 90:125 as it has the highest accuracy.

```
plt.figure(figsize=(15, 5))
plt.plot(x_faces[90:125], y_faces[90:125])
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```



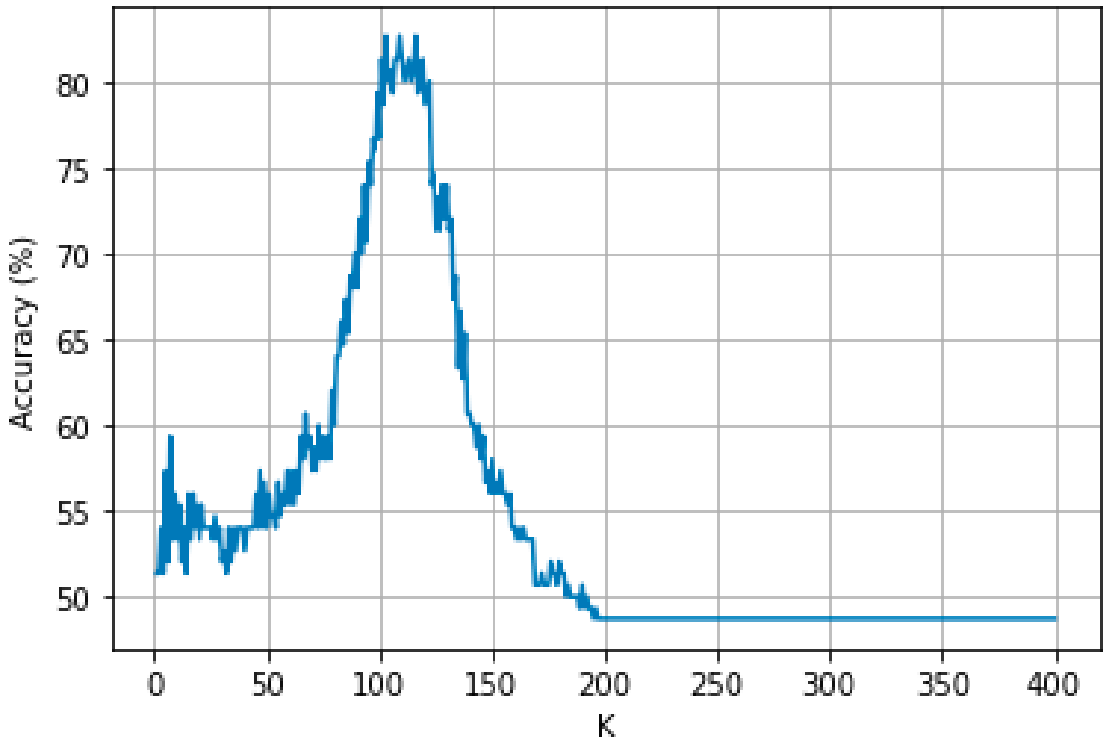
K = 103, K = 109 and K = 116 have the highest accuracy percentages, ignoring K = 1.

```
print(f"The model is {y_faces[102]}% accurate to test data at K = 103, using Manhattan distance.")
print(f"The model is {y_faces[108]}% accurate to test data at K = 109, using Manhattan distance.")
print(f"The model is {y_faces[115]}% accurate to test data at K = 116, using Manhattan distance.")
```

[68] ✓ 0.3s Python

```
The model is 82.66666666666667% accurate to test data at K = 103, using Manhattan distance.
The model is 82.66666666666667% accurate to test data at K = 109, using Manhattan distance.
The model is 82.66666666666667% accurate to test data at K = 116, using Manhattan distance.
```

Conclusion: Using Manhattan distance to classify faces dataset using KNN, K = 103, 109, 116 all give the highest possible accuracy on our test set = 82.67%.



3.3.2. Euclidean Distance

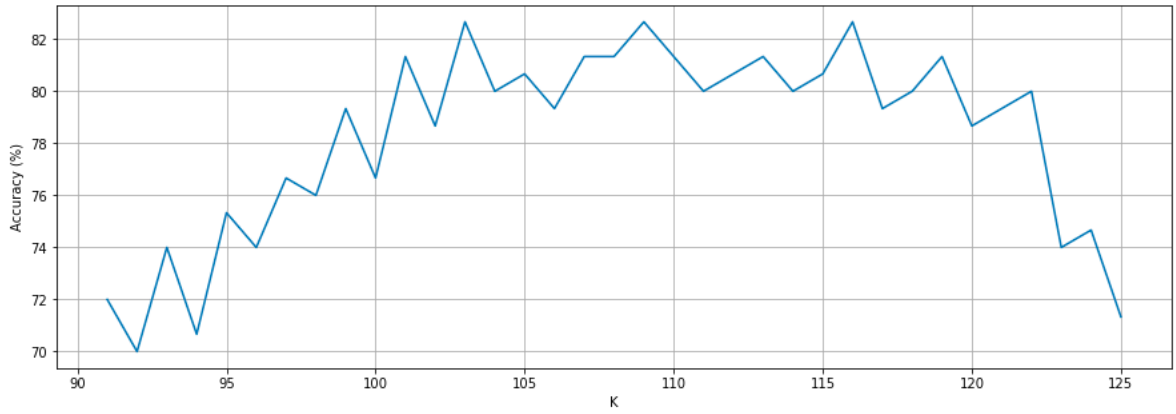
To choose the K that gives us the best accuracy for our faces dataset while using the Euclidean distance, we will try all K values from 1 to 400 and draw a graph to check the best possible value for K.

```
x_faces = list(range(1, 401))
y_faces = []

# Loop for every K value
for i in x_faces:
    # Training
    knn = KNeighborsClassifier(n_neighbors=i, metric='euclidean')
    knn.fit(new_faces_train, faces_train_labels)

    # Testing
    y_knn_pred_faces = knn.predict(new_faces_test)

    # Calculate accuracy for each K
    val = metrics.accuracy_score(y_knn_pred_faces,faces_test_labels)*100
    y_faces.append(val)

plt.plot(x_faces, y_faces)
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```

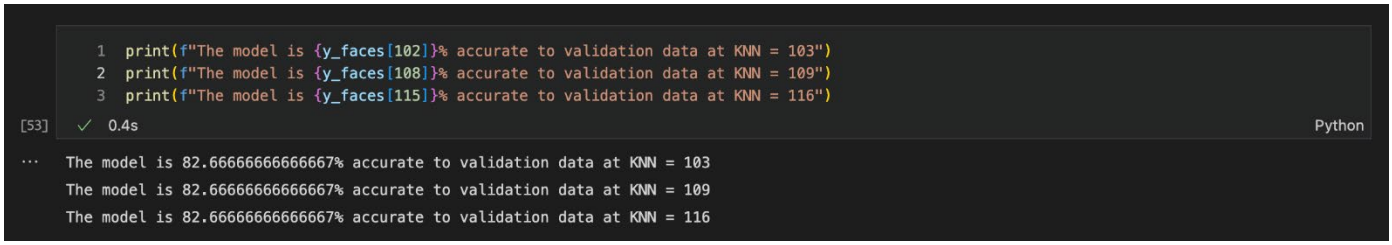
Resulting in:
As we can see, a bell-shaped curve was formed with the peak between K = 90 to K = 120.

Taking a closer look at K ranging 90:120 as they have highest accuracy range.

```
plt.figure(figsize=(15, 5))
plt.plot(x_faces[90:125], y_faces[90:125])
plt.ylabel('Accuracy (%)')
plt.xlabel('K')
plt.grid()
plt.show()
```



Observing the above graph, model results in the highest accuracy percentages at K = 103, K = 109, K = 116.



Conclusion: Using Euclidean distance to classify faces dataset using KNN, K = 103, 109, 116 all give the highest possible accuracy on our test set = 82.67%.

Conclusion on KNN with faces dataset: Both distance calculation methods result in the exact same graphs and accuracy percentages for different K values. The best accuracy 82.67% is obtained at K = 103, 109, 116