Supervised Learning Assignment Report

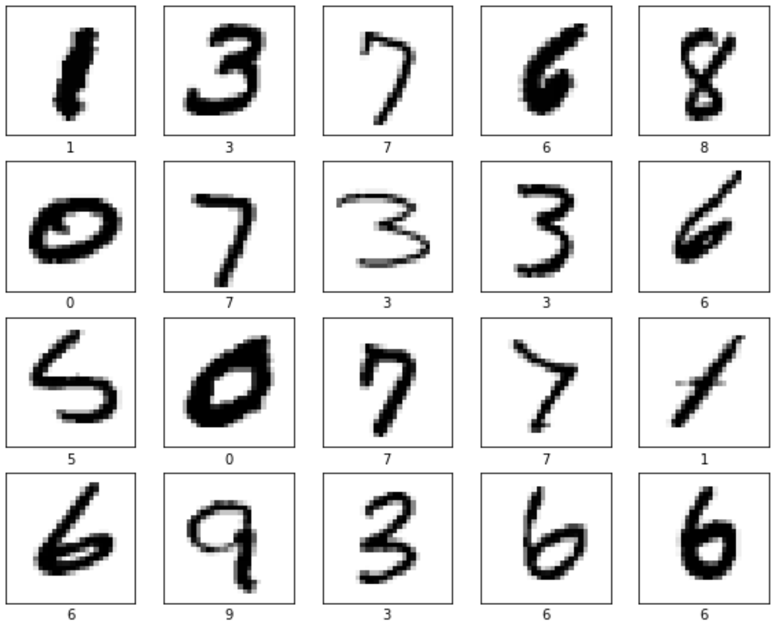
MNIST Dataset Handwritten Digit Recognizer

Spring 2021-2022

Dataset Used:

The dataset which is used in this assignment is mnist dataset that was built in TensorFlow library. Which is composed of images, each image is 28*28 pixel.

The minst dataset has 60,000 training sample. And 10,000 testing sample.



Preprocessing:

The dataset was loaded from keras as:

```
(X_train, Y_train), (X_test, Y_test) = keras.datasets.mnist.load_data()
```

First of all, 10,000 training sample out of 60,000 was taken to train the model, in order to take a balanced number of samples from each class, this function used to count the number of samples of each class in the training samples:

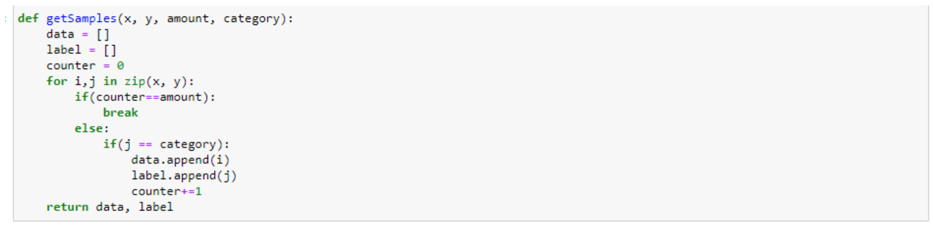
```
def valueCounts(X):
    unique, counts = np.unique(X, return_counts=True)
    return dict(zip(unique, counts))
```

The result is:

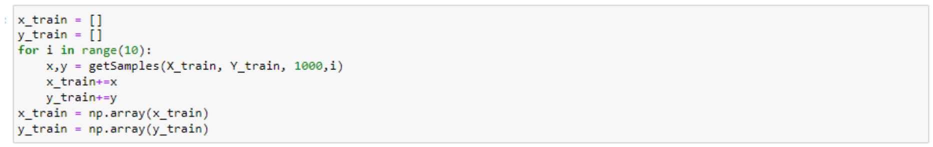
```
valueCounts(Y_train)
```

```
{0: 5923,
 1: 6742,
 2: 5958,
 3: 6131,
 4: 5842,
 5: 5421,
 6: 5918,
 7: 6265,
 8: 5851,
 9: 5949}
```

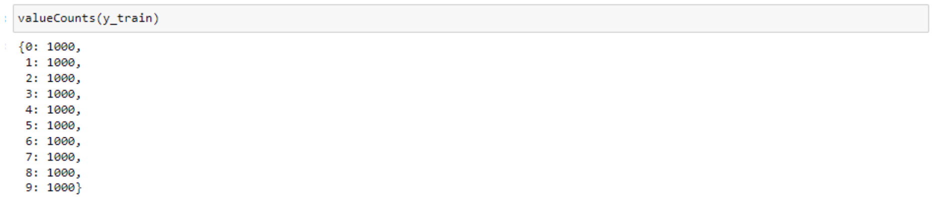
Another function is used to select specific amount of each category. and the result after using it.



```
def getSamples(x, y, amount, category):
    data = []
    label = []
    counter = 0
    for i,j in zip(x, y):
        if(counter==amount):
            break
        else:
            if(j == category):
                data.append(i)
                label.append(j)
                counter+=1
    return data, label
```



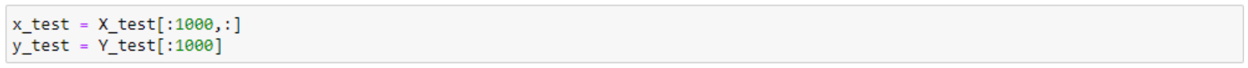
```
x_train = []
y_train = []
for i in range(10):
    x,y = getSamples(X_train, Y_train, 1000,i)
    x_train+=x
    y_train+=y
x_train = np.array(x_train)
y_train = np.array(y_train)
```



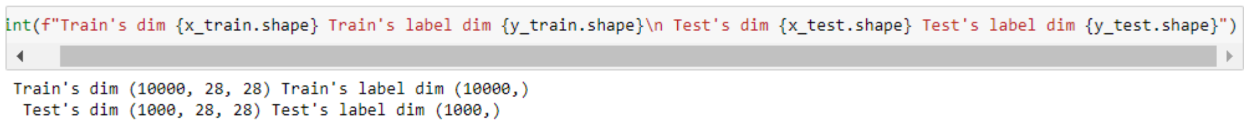
```
valueCounts(y_train)
```

```
{0: 1000,
 1: 1000,
 2: 1000,
 3: 1000,
 4: 1000,
 5: 1000,
 6: 1000,
 7: 1000,
 8: 1000,
 9: 1000}
```

After that, 1,000 testing sample is used to get the accuracy of the model



```
x_test = X_test[:1000,:]
y_test = Y_test[:1000]
```



```
int(f"Train's dim {x_train.shape} Train's label dim {y_train.shape}\n Test's dim {x_test.shape} Test's label dim {y_test.shape}")
```

```
Train's dim (10000, 28, 28) Train's label dim (10000,)
 Test's dim (1000, 28, 28) Test's label dim (1000,)
```

Libraries used:

Tensorflow: To import the dataset.

Numpy: To manipulate tensors as variable in the form of numpy array.

Matplotlib: was used to plot the images and its grids.

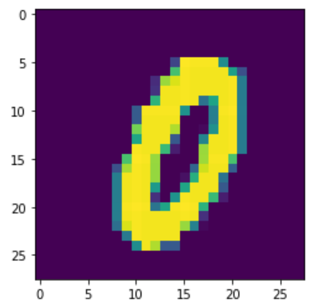
Feature Extraction and functions used:

Feature extraction is done by dividing the image into number of grids and getting a centroid of each grid and getting its feature vector.

This image is a sample which the functions applied on it before the whole data

```
Img = x_train[2]
plt.imshow(Img)
```

```
<matplotlib.image.AxesImage at 0x2090aef8bb0>
```



This function is used to divide the image into grides:

```
def getGrids(img, winRow, winCol):
    grids = []
    imgR, imgC = img.shape
    rowLimitation = imgR-1
    colLimitation = imgC-1

    for r in range(0, rowLimitation, winRow):
        for c in range(0, colLimitation, winCol):
            grid = img[r:r+winRow, c:c+winCol]
            grids.append(grid)

    return np.array(grids)
```

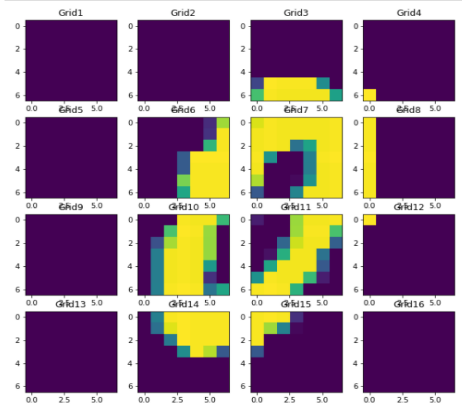
The dimensionality of the sample before and after applying getGrides function on it, it divided into 16 grides each one is 7*7:

```
grids = getGrids(Img, 7, 7)
grids.shape
```

```
(16, 7, 7)
```

To be like this:

```
fig, axs = plt.subplots(4, 4, figsize=(10,10))
col = 0
for r, w in enumerate(grids):
    axs[r//4, col].imshow(w)
    axs[r//4, col].set_title(f'Grid{r+1}')
    col+=1
    col = col % 4
```

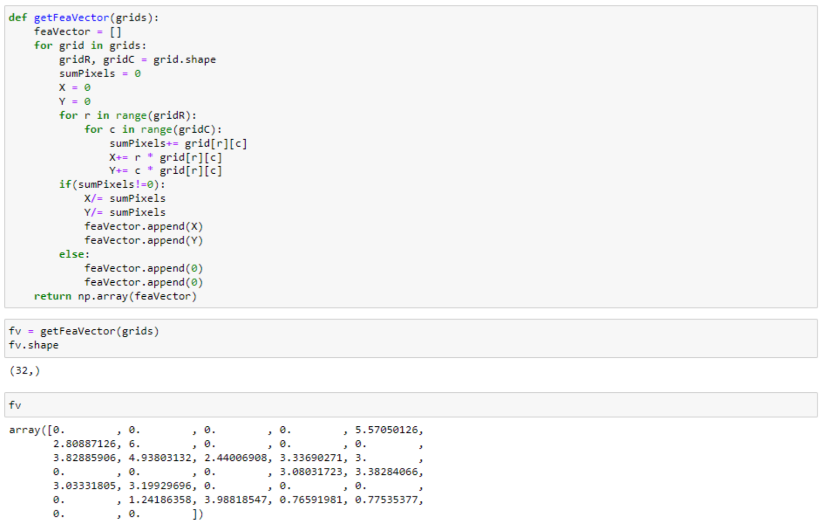


16 grids means it has 32 value in its feature vector respect to the equations used to get the centroid

-X=$\sum_x \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})/\sum_x \sum_y \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$

- Y=$\sum_y \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})/\sum_x \sum_y \boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$

This function is used to get the feature vector from the grides:



```
def getFeaVector(grids):
    feaVector = []
    for grid in grids:
        gridR, gridC = grid.shape
        sumPixels = 0
        X = 0
        Y = 0
        for r in range(gridR):
            for c in range(gridC):
                sumPixels+= grid[r][c]
                X+= r * grid[r][c]
                Y+= c * grid[r][c]
        if(sumPixels!=0):
            X/= sumPixels
            Y/= sumPixels
            feaVector.append(X)
            feaVector.append(Y)
        else:
            feaVector.append(0)
            feaVector.append(0)
    return np.array(feaVector)
```

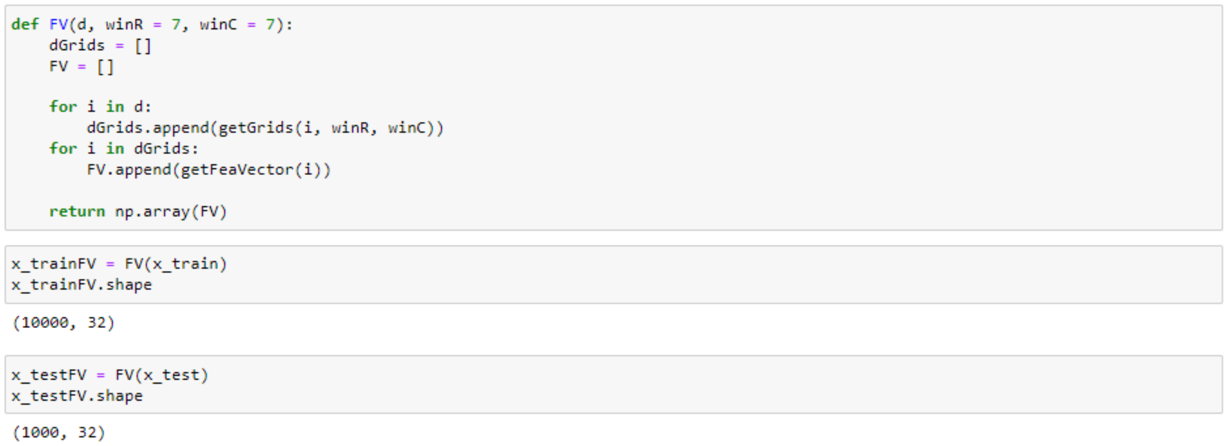
```
fv = getFeaVector(grids)
fv.shape
```

```
(32,)
```

```
fv
```

```
array([0.        , 0.        , 0.        , 0.        , 5.57050126,
       2.80887126, 6.        , 0.        , 0.        , 0.        ,
       3.82885906, 4.93803132, 2.44006908, 3.33690271, 3.        ,
       0.        , 0.        , 0.        , 3.08031723, 3.38284066,
       3.03331805, 3.19929696, 0.        , 0.        , 0.        ,
       0.        , 1.24186358, 3.98818547, 0.76591981, 0.77535377,
       0.        , 0.        ])
```

This function is used to applying the last two functions to features in x_train and y_train:



```
def FV(d, winR = 7, winC = 7):
    dGrids = []
    FV = []

    for i in d:
        dGrids.append(getGrids(i, winR, winC))
    for i in dGrids:
        FV.append(getFeaVector(i))

    return np.array(FV)
```

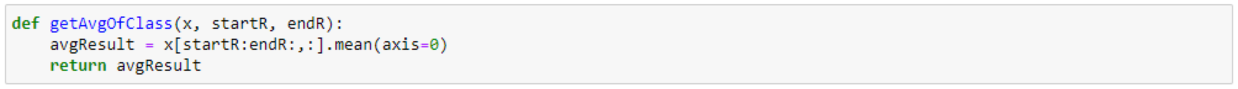
```
x_trainFV = FV(x_train)
x_trainFV.shape
```

```
(10000, 32)
```

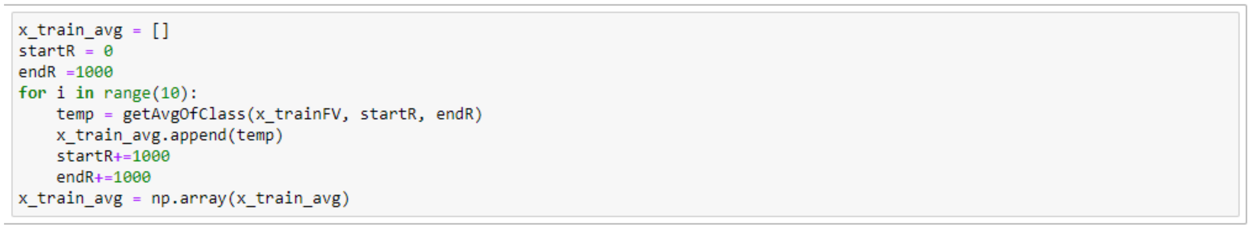
```
x_testFV = FV(x_test)
x_testFV.shape
```

```
(1000, 32)
```

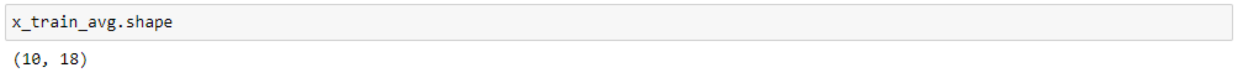
This function is used to get the average per class to reduce the shape of x_train_FV from (10,000 , 18) to (10, 18) one vector represent each class:



```
def getAvgOfClass(x, startR, endR):
    avgResult = x[startR:endR:,:].mean(axis=0)
    return avgResult
```



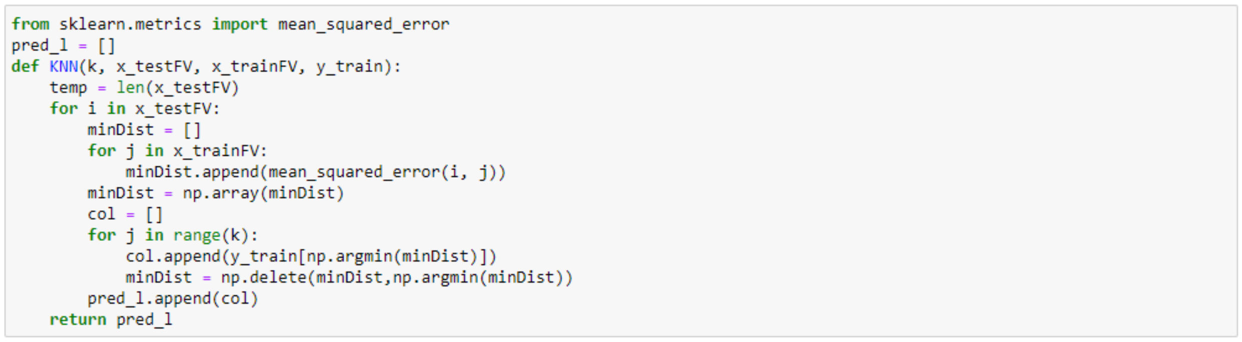
```
x_train_avg = []
startR = 0
endR =1000
for i in range(10):
    temp = getAvgOfClass(x_trainFV, startR, endR)
    x_train_avg.append(temp)
    startR+=1000
    endR+=1000
x_train_avg = np.array(x_train_avg)
```



```
x_train_avg.shape
```

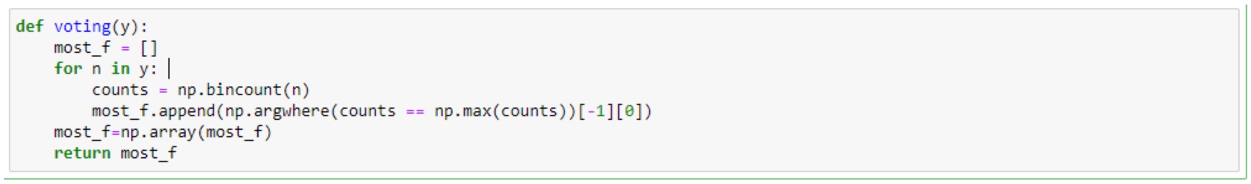
```
(10, 18)
```

This function is used to get k minimum distinct between given value of x_train and y_train:



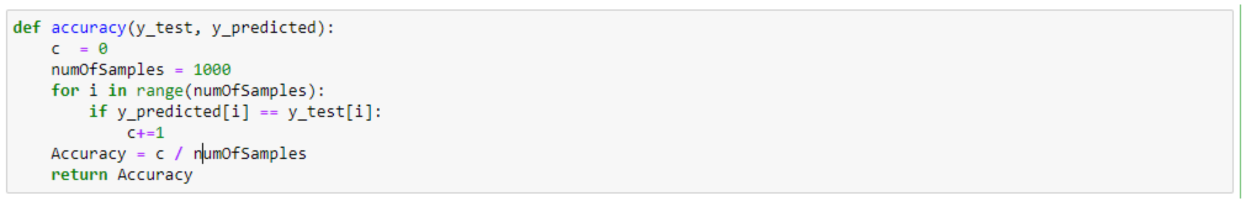
```
from sklearn.metrics import mean_squared_error
pred_l = []
def KNN(k, x_testFV, x_trainFV, y_train):
    temp = len(x_testFV)
    for i in x_testFV:
        minDist = []
        for j in x_trainFV:
            minDist.append(mean_squared_error(i, j))
        minDist = np.array(minDist)
        col = []
        for j in range(k):
            col.append(y_train[np.argmin(minDist)])
            minDist = np.delete(minDist,np.argmin(minDist))
        pred_l.append(col)
    return pred_l
```

This function is used to vote between the k results:



```
def voting(y):
    most_f = []
    for n in y:
        counts = np.bincount(n)
        most_f.append(np.argwhere(counts == np.max(counts))[-1][0])
    most_f=np.array(most_f)
    return most_f
```

This function is used to get accuracy between predicted values and actual values:

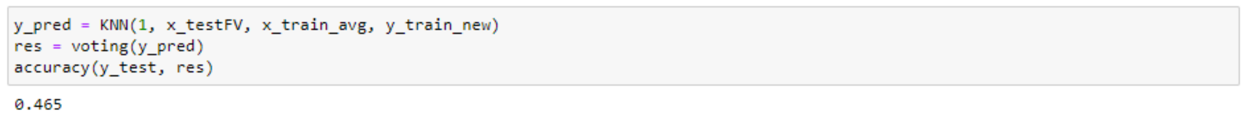


```
def accuracy(y_test, y_predicted):
    c  = 0
    numOfSamples = 1000
    for i in range(numOfSamples):
        if y_predicted[i] == y_test[i]:
            c+=1
    Accuracy = c / numOfSamples
    return Accuracy
```

Comparison between results:

Firstly, when we got average per class with two different window size:

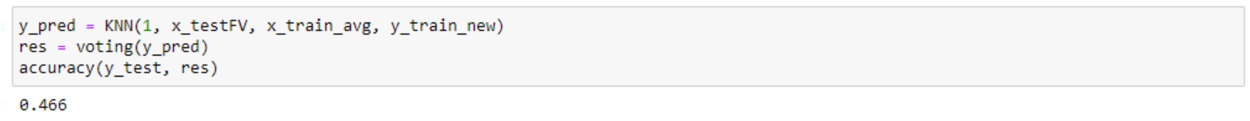
-when the window size was 7*7 the accuracy was 46.5%



```
y_pred = KNN(1, x_testFV, x_train_avg, y_train_new)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.465
```

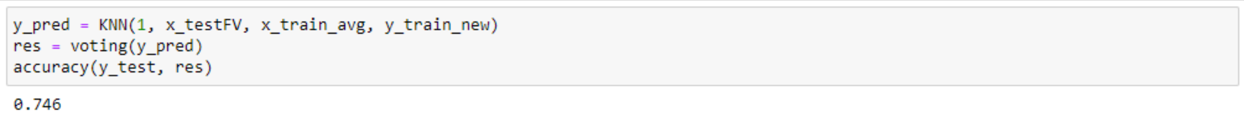
-when the window size was 8*8 the accuracy was 46.6%



```
y_pred = KNN(1, x_testFV, x_train_avg, y_train_new)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.466
```

-when the window size was 3*3 the accuracy was 74.6%

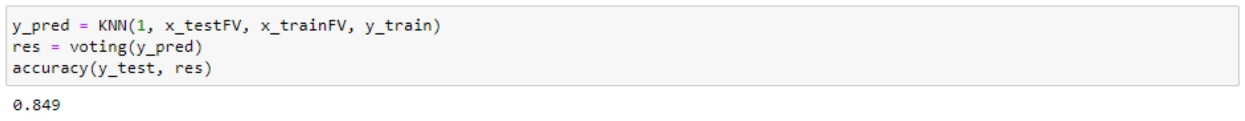


```
y_pred = KNN(1, x_testFV, x_train_avg, y_train_new)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.746
```

Secondly, when we didn't get average per class and use different window size and different value of k:

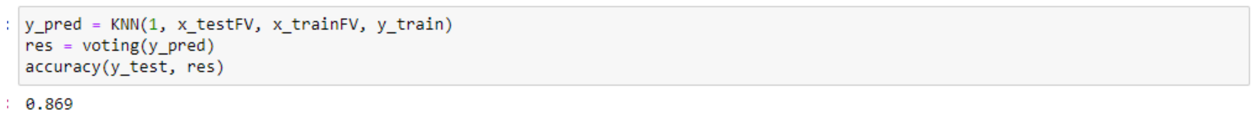
-when the widow size was 7*7 and k=1 the accuracy was 84.9%



```
y_pred = KNN(1, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.849
```

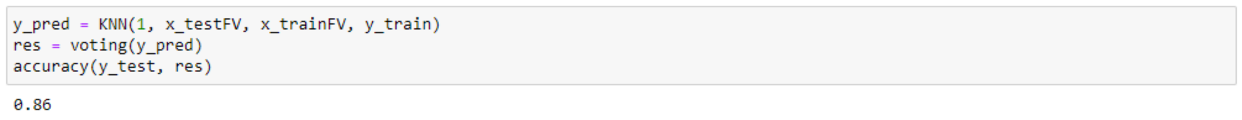
-when the widow size was 8*8 and k=1 the accuracy was 86.9%



```
y_pred = KNN(1, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.869
```

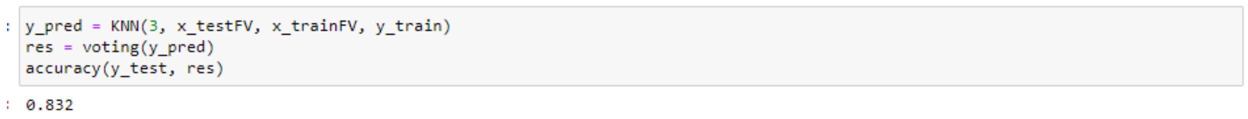
-when the widow size was 3*3 and k=1 the accuracy was 86%



```
y_pred = KNN(1, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.86
```

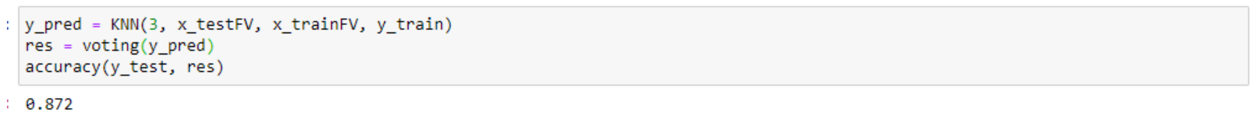
-when the widow size was 7*7 and k=3 the accuracy was 83.2%



```
y_pred = KNN(3, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.832
```

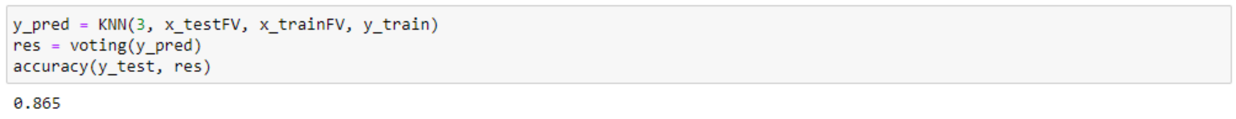
-when the widow size was 8*8 and k=3 the accuracy was 87.2%



```
y_pred = KNN(3, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.872
```

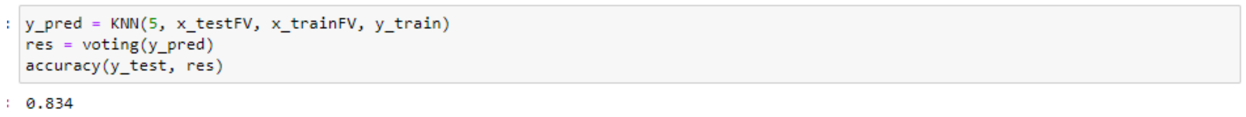
-when the widow size was 3*3 and k=3 the accuracy was 85.5%



```
y_pred = KNN(3, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.865
```

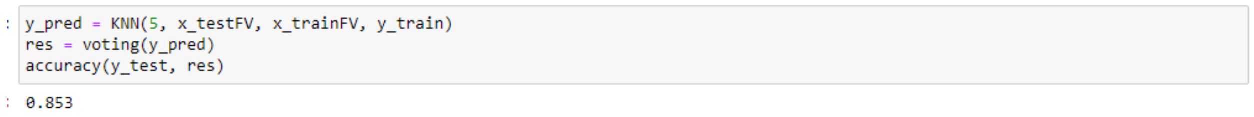
-when the widow size was 7*7 and k=5 the accuracy was 83.4%



```
y_pred = KNN(5, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.834
```

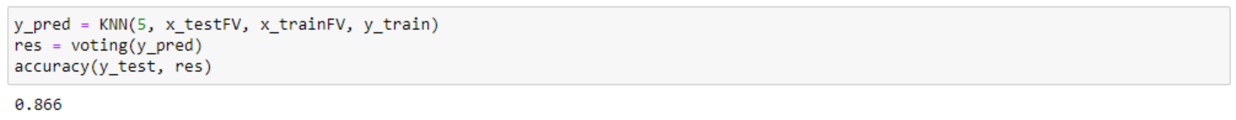
-when the widow size was 8*8 and k=5 the accuracy was 85.3%



```
y_pred = KNN(5, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.853
```

-when the widow size was 3*3 and k=5 the accuracy was 86.6%



```
y_pred = KNN(5, x_testFV, x_trainFV, y_train)
res = voting(y_pred)
accuracy(y_test, res)
```

```
0.866
```

Final analysis and conclusion:

-After comparing results, getting the average per class did not improve the accuracy so we did not have to do that.

-the accuracy I got from different models with different window size and different almost the same.

-the range of the accuracy from the models is from 85.3 to 87.2. my personal comment on this results which is good but not very good that is because the small number of samples I got from training samples and testing samples.

-the best accuracy we got was 87.2% when the window size was 8*8 and k = 3.