In this homework, I implemented the one-versus-all support vector classification algorithm

Part 2)

Read Data from

hw06_images.csv

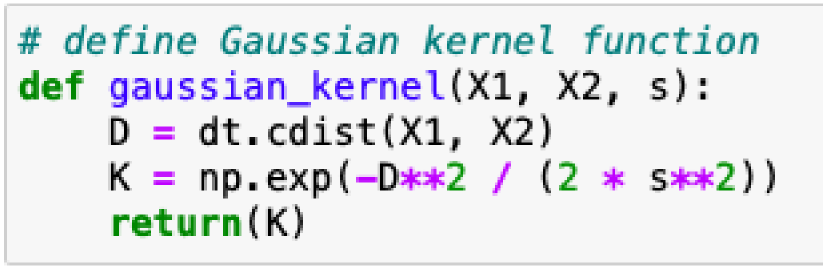
hw06_labels.csv

Part 3)

Data is divided into train and test set. first 1000 images to the training set and the remaining 4000 images to the test set.

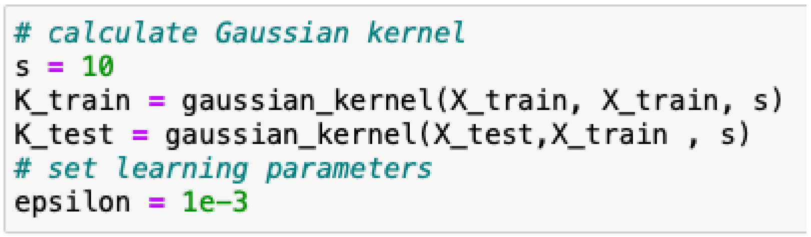
Part 4)

I used lab08 code to implement one-versus-all support vector classification algorithm. First took Gaussian Kernel Function from lab08.



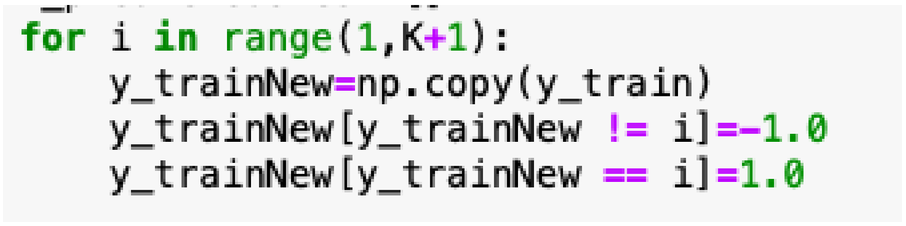
```python
# define Gaussian kernel function
def gaussian_kernel(X1, X2, s):
    D = dt.cdist(X1, X2)
    K = np.exp(-D**2 / (2 * s**2))
    return(K)
```

Then calculated Gaussian Kernels for train and Test data.



```python
# calculate Gaussian kernel
s = 10
K_train = gaussian_kernel(X_train, X_train, s)
K_test = gaussian_kernel(X_test,X_train , s)
# set learning parameters
epsilon = 1e-3
```

I defined OVA(c, s=10) function to find predictions for training and test sets. This code works similarly with lab08 code. However, to make it one-vs-all I iterated K times. At each step I set one label to 1, and others as -1. Then trained one- versus-all support vector classification algorithm to calculate train and test labels.



```python
for i in range(1,K+1):
    y_trainNew=np.copy(y_train)
    y_trainNew[y_trainNew != i]=-1.0
    y_trainNew[y_trainNew == i]=1.0
```

get label scores using given formula in the class. Which is used also in lab08.

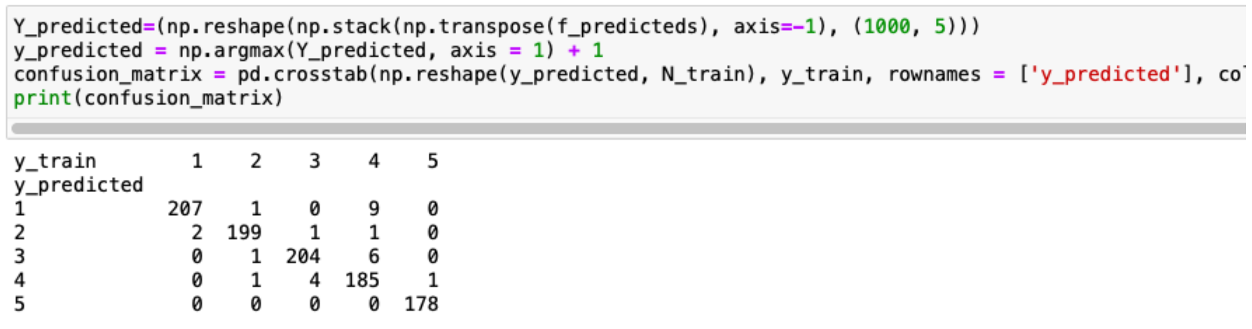
$$\begin{aligned} f(z) &= w^T . z + w_0 \\ &= w^T . \Phi(x) + w_0 \\ &= \sum_{i=1}^{N} \alpha_i y_i \underbrace{\Phi(x_i)^T . \Phi(x)}_{k(x_i, x)} + w_0 \end{aligned}$$

```
# calculate predictions on training samples
f_predicted = np.matmul(K_train, y_trainNew[:,None] * alpha[:,None]) + w0
f_predicteds.append(f_predicted)

# calculate predictions on training samples
f_predictedTest = np.matmul(K_test , y_trainNew[:,None]* alpha[:,None]) + w0
f_predictedsTest.append(f_predictedTest)
```

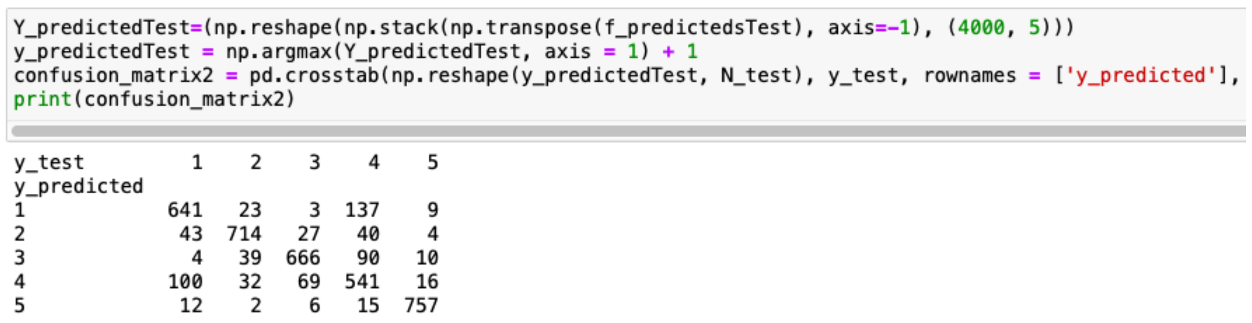
Part 5-6)

After getting predict labels scores as train=(5,1000) test=(5,4000) array. Get predicted label using argmax. Finally printed confusion matrix using pd.crosstab.



```
Y_predicted=(np.reshape(np.stack(np.transpose(f_predicteds), axis=-1), (1000, 5)))
y_predicted = np.argmax(Y_predicted, axis = 1) + 1
confusion_matrix = pd.crosstab(np.reshape(y_predicted, N_train), y_train, rownames = ['y_predicted'], co
print(confusion_matrix)
```

```
y_train        1    2    3    4    5
y_predicted
1            207    1    0    9    0
2              2  199    1    1    0
3              0    1  204    6    0
4              0    1    4  185    1
5              0    0    0    0  178
```

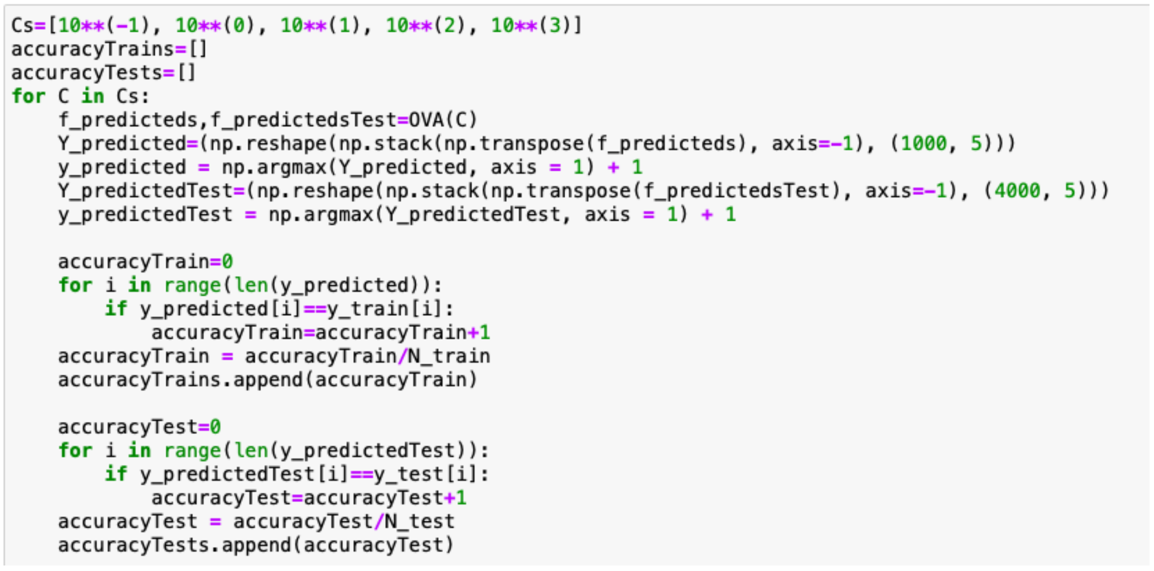


```
Y_predictedTest=(np.reshape(np.stack(np.transpose(f_predictedsTest), axis=-1), (4000, 5)))
y_predictedTest = np.argmax(Y_predictedTest, axis = 1) + 1
confusion_matrix2 = pd.crosstab(np.reshape(y_predictedTest, N_test), y_test, rownames = ['y_predicted'],
print(confusion_matrix2)
```

```
y_test         1    2    3    4    5
y_predicted
1            641   23    3  137    9
2             43  714   27   40    4
3              4   39  666   90   10
4            100   32   69  541   16
5             12    2    6   15  757
```

Part 7)

I this part, I draw the classification accuracy for training and set data points as a function of C(0.1, 1, 10, 100 and 1000) with the kernel width parameter s to 10. For each parameter I run Ova function then calculated accuracy errors from these predictions for train and test sets.

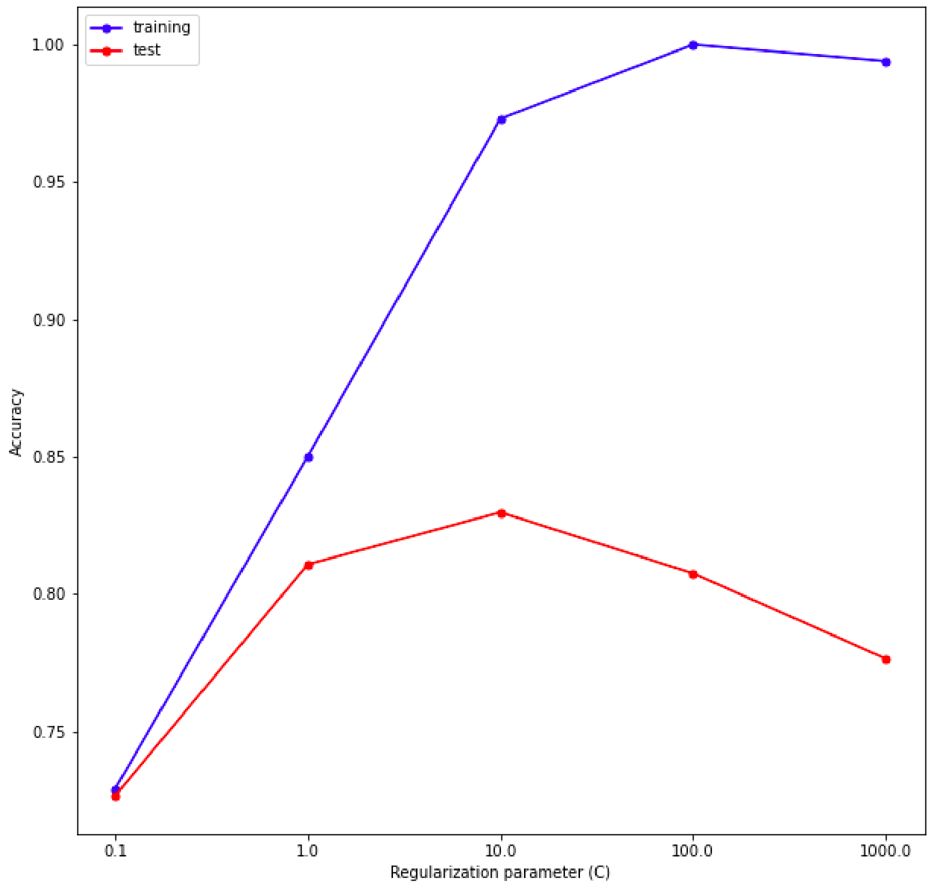


```
Cs=[10**(-1), 10**(0), 10**(1), 10**(2), 10**(3)]
accuracyTrains=[]
accuracyTests=[]
for C in Cs:
    f_predicteds,f_predictedsTest=OVA(C)
    Y_predicted=(np.reshape(np.stack(np.transpose(f_predicteds), axis=-1), (1000, 5)))
    y_predicted = np.argmax(Y_predicted, axis = 1) + 1
    Y_predictedTest=(np.reshape(np.stack(np.transpose(f_predictedsTest), axis=-1), (4000, 5)))
    y_predictedTest = np.argmax(Y_predictedTest, axis = 1) + 1

    accuracyTrain=0
    for i in range(len(y_predicted)):
        if y_predicted[i]==y_train[i]:
            accuracyTrain=accuracyTrain+1
    accuracyTrain = accuracyTrain/N_train
    accuracyTrains.append(accuracyTrain)

    accuracyTest=0
    for i in range(len(y_predictedTest)):
        if y_predictedTest[i]==y_test[i]:
            accuracyTest=accuracyTest+1
    accuracyTest = accuracyTest/N_test
    accuracyTests.append(accuracyTest)
```

Then plotted accuracy values of train and test sets for different Regularization C parameters. Increasing regularization parameter(C), gives better results in terms of accuracy for train set. Increasing C, we increase penalization of mispredicted labels. That's why it is going to be overfitting. That causes decrease in test set after parameter C = 10.


training
test
Accuracy
Regularization parameter (C)