CPT_S 570

Machine Learning, Fall 2020

Homework # 2

Tayyab Munir

11716089

Question 1 – Analytical Part

1. Suppose we have n_+ positive training examples and n− negative training examples. Let C_+ be the center of the positive examples and C_- be the center of the negative examples, i.e., $C_+ = \frac{1}{n_+} \sum_{i:\, y_i=+1} x_i$ and $C_- = \frac{1}{n_-} \sum_{i:\, y_i=-1} x_i$. Consider a simple classifier called CLOSE that classifies a test example x by assigning it to the class whose center is closest.
   - Show that the decision boundary of the CLOSE classifier is a linear hyperplane of the form sign(w· x + b). Compute the values of w and b in terms of C_+ and C_-.

Solution:

We have two points C_+ be center of positive examples and C_- be center of negative examples respectively. We need to show that the boundary of the CLOSE classifier is a linear hyperplane of the form sign(w· x + b).

Here w is the weight vector, which is the distance between the 2 centroids C_+ and C_- which can be written as $w = |C_+ - C_-|$.
By substituting the value of w in sign(w· x + b) we get

$$(C_+ - C_-)\,.\,x + b = 0$$

Consider an example x which has to be classified, let suppose it lies between two clusters such that

$$x = \frac{(C_+ - C_-)}{2}$$

After substituting the value of x we got

$$(C_+ - C_-)\,.\frac{(C_+ - C_-)}{2} + b = 0$$

$$\frac{1}{2}((C_+ - C_-)\,.\,(C_+ - C_-))\ + b = 0$$

$$\frac{1}{2}\ ({C_+}^2 -\ {C_-}^2)\ + b = 0$$

Finally, we can write w and b in terms of C_+ and C_-

$$w = (C_+ - C_-)$$

$$b = -\frac{1}{2}\ ({C_+}^2 -\ {C_-}^2)$$

   - Recall that the weight vector can be written as a linear combination of all the training examples: $w = \sum_{i=1}^{n_+ + n_-} \propto_i\ y_i\, x_i$. Compute the dual weights (α's). How many of the training examples are support vectors?

Solution:

In the above answer we got our weight vector as

$$w = (C_+ - C_-)$$

by substituting values, we get

$$w = (\frac{1}{n_+} \sum\nolimits_{i:\, y_i=+1} x_i \; - \frac{1}{n_-} \sum\nolimits_{i:\, y_i=-1} x_i)$$

- $\boldsymbol{w} = (\sum_{\boldsymbol{i}:\, \boldsymbol{y_i}=+\boldsymbol{1}} \frac{\boldsymbol{1}}{\boldsymbol{n_+}} \boldsymbol{x_i} \; - \; \sum_{\boldsymbol{i}:\, \boldsymbol{y_i}=-\boldsymbol{1}} \frac{\boldsymbol{1}}{\boldsymbol{n_-}} \boldsymbol{x_i})$

Now, we have the weight vector

$$w = \sum_{i=1}^{n_+ n_-} \propto_i \; y_i \, x_i$$

$$w = \sum_{i=1}^{n_+} \propto_i \; y_i \, x_i \sum_{j=1}^{n_-} \propto_j \; y_j \, x_j$$

For positive values lets insert $y_i = 1$and for negative values $y_j = -1$

- $\boldsymbol{w} = \sum_{\boldsymbol{i}=\boldsymbol{1}}^{\boldsymbol{n_+}} \propto_{\boldsymbol{i}} \; \boldsymbol{x_i} \sum_{\boldsymbol{j}=\boldsymbol{1}}^{\boldsymbol{n_-}} \propto_{\boldsymbol{j}} \; \boldsymbol{x_j}$

By comparing both the weights we can get $\propto_i$

$\propto_i = \frac{1}{n_+}$ for positive class points

$\propto_i = \frac{1}{n_-}$ for negative class points

2. Suppose we use the following radial basis function (RBF) kernel: $K(x_i, x_j) = \exp\left(-\frac{1}{2}\left\|x_i - x_j\right\|^2\right)$, which has some implicit unknown mapping φ(x).
   - Prove that the mapping φ(x) corresponding to RBF kernel has infinite dimensions.

Solution:

Radial basis function (RBF) kernel: $K(x_i, x_j) = \exp\left(-\frac{1}{2}\left\|x_i - x_j\right\|^2\right)$

By expanding using $(a+b)^2$

$$= \exp\left(\frac{-1}{2}(\|x_i\|^2 + \left\|x_j\right\|^2 - 2\|x_i\|\left\|x_j\right\|)\right)$$

$$= \exp(-(\|x_i\|^2))\, exp\left(-\left(\left\|x_j\right\|^2\right)\right) exp\ (2\|x_i\|\left\|x_j\right\|)$$

By Taylor Series expansion we got

$$= \exp(-(\|x_i\|^2))\, exp\left(-\left(\left\|x_j\right\|^2\right)\right) \underbrace{\sum_{n=0}^{\infty} \frac{2^n \|x_i\|^n \left\|x_j\right\|^n}{n!}}_{exp\ (2\|x_i\|\left\|x_j\right\|}$$

This equation is the upper bound, hence the RBF has infinite dimensions.

   - Prove that for any two input examples xi and xj , the squared Euclidean distance of their corresponding points in the higher-dimensional space defined by φ is less than 2, i.e., $\left\|φ(x_i) - φ(x_j)\right\|^2 \le 2$.

Solution:

$$\left\|φ(x_i) - φ(x_j)\right\|^2$$

$$= (φ(x_i) - \varphi(x_j)) \cdot (\varphi(x_i) - \varphi(x_j))$$

$$= \varphi(x_i) \cdot \varphi(x_i) + \varphi(x_j) \cdot \varphi(x_j) - 2 \cdot \varphi(x_i) \cdot \varphi(x_j)$$

$$= 2 - 2exp(-^{1} \| x_i - x_j \| ^{2})\ 2$$

$$\left\|φ(x_i) - φ(x_j)\right\|^2 < 2$$

Hence, we can see that the squared Euclidean distance of their corresponding points in the higher dimensional space is less than 2.

3. The decision boundary of a SVM with a kernel function (via implicit feature mapping φ(.)) is defined as follows: $w \,.\, \varphi(\mathrm{x}) + b = \sum_{i \in y_i \alpha_i} K(x_i, x) + b = f(x; \alpha, b)$, where w and b are parameters of the decision boundary in the feature space phi defined by the kernel function K, SV is the set of support vectors, and αi is the dual weight of the ith support vector.

   Let us assume that we use the radial basis function (RBF) kernel $K(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)$; also assume that the training examples are linearly separable in the feature space φ and SVM finds a decision boundary that perfectly separates the training examples. If we choose a testing example x_{far} that is far away from any training instance xi (distance here is measured in the original feature space < d). Prove that $f(x_{far}; \alpha, \mathrm{b}) \approx$ b.

Solution:
We need to prove that $f(x_{far}; \alpha, \mathrm{b}) \approx \mathrm{b}$
Whereas,

$$\|x_{far} - x_i\| \gg 0, \quad \forall\, i \in sv$$

Here SV is the set of support vectors.

$$K(x_{far} - x_i) \approx 0, \quad \forall\, i \in sv$$

$$\sum_{i \in SV} y_i\, \alpha_i\, K(x_{far}, x_i) \approx 0, \forall\, i \in sv$$

So, first part of the function is evaluated to a ‘0’ and we are only left with the other part of the function i.e. 'b'.

$$f(x_{far}; \alpha, \mathrm{b}) \approx \mathrm{b}$$

Hence proved.

4. The function $K(x_i, x_j) = -\langle x_i, x_j \rangle$ is a valid kernel. Prove or Disprove it.

Solution:
The function $K(x_i, x_j) = -\langle x_i, x_j \rangle$ is a valid kernel is a valid kernel if and only if it satisfies the mercer's theorem. In order to make the given function satisfy the mercer's theorem we have to satisfy following condition that

$$K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i, x_j) c_i c_j \geq 0$$

If this condition is true and the value is greater than 0 then the given function is a valid kernel.

After substituting the values we get

$$K(x_i, x_j) = -\langle x_i, x_j \rangle$$

$$K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} -\langle x_i, x_j \rangle\, c_i c_j$$

$$K(x_i, x_j) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \langle x_i, x_j \rangle\, c_i c_j$$

Since we get a negative symbol and a negative is always less than 0 we can say that the kernel $K(x_i, x_j) = -\langle x_i, x_j \rangle$ is not a valid kernel.

5. You are provided with n training examples: $(x_1, y_1), (x_2, y_2), \dots , (x_n, y_n)$, where xi is the input example, yi is the class label (+1 or -1). The teacher gave you some additional information by specifying the costs for different mistakes C_+ and C_-, where C_+ and C_- stand for the cost of misclassifying a positive and negative example respectively.
   a. How will you modify the Soft-margin SVM formulation to be able to leverage this extra information? Please justify your answer.

Solution:

We have $S = \{(x_i, y_i) \mid i = 1,\ 2, \dots ,\ l\} \subset\ R^d\ \times \{+1, -1\}$ be our training sample and our cost of misclassification is C_+ and C_- for making a mistake for positive and negative examples. C_+ is for limiting the influence of positive outliers whereas, C_- is for limiting the influence of negative outliers. For learning from a finite sample, Lets divide S into subsets $S \pm 1$ which contain the indices of all positive and negative examples respectively. The optimization problem can now be formulated as

$$Min\ \frac{1}{2}||w||2\ +\ C\ \sum_{i \in s+1} C + 1\, x_i\ \xi_{i \dots k}$$

$$+\, C \sum_{i \in s+1} C - 1\, x_i\ \xi_{i \dots k}$$

$$s.t\ \ y'(w.\varphi(x_i) + b) >= 1\ -\ \xi_{i \dots k}.\ \ \xi i \geq 0$$

6. You are provided with a set of n training examples: $(x_1, y_1), (x_2, y_2), \dots , (x_n, y_n)$, where xi is the input example, yi is the class label (+1 or -1). Suppose n is very large (say in the order of millions). In this case, standard SVM training algorithms will not scale due to large training set. Tom wants to devise a solution based on "Coarse-to-Fine" framework of problem solving. The basic idea is to cluster the training data; train a SVM classifier based on the clusters (coarse problem); refine the clusters as needed (fine problem); perform training on the finer problem; and repeat until convergence. Suppose we start with k_+ positive clusters and k_- negative clusters to begin with (a cluster is defined as a set of examples). Please specify the mathematical formulation (define all the variables used in your formulation) and concrete algorithm for each of the following steps to instantiate this idea:

   a. How to define the SVM training formulation for a given level of coarseness: a set of k+ positive clusters and a set of k− negative clusters?

Solution:

Given are k+ positive clusters and k- negative clusters to begin with and we need to convert this to a finer approximation to control the complexity of learning. Let us consider the nearest centroid classification so we assume a centroid from each of the clusters. Which means we have k+ positive centroids and k- negative centroids to reduce the complexity we can consider these points as the training examples for the model. After formulating an SVM optimization problem we have

$$Minimize\ \frac{1}{2}\ ||w||^2\ +\ C\ \sum_{i=1}^{k_+\ +\ k_-}\ \xi_i$$

$$s.t.\ y_i(x_i\ \times\ w\ +b) \geq\ 1 -\ \xi_i$$

   b. How to refine the clusters based on the resulting SVM classifier?

Solution:

One way to refine the SVM is to perform training on the clusters that are closer to the decision boundary. The idea is to find the clusters that are close to the decision boundary and break those clusters into smaller clusters and train on those clusters. This will result in a finer classification over the support vectors.

   c. What is the stopping criteria?

Solution:
Accuracy score of the SVM can be stopping criteria. We can compare the accuracies for each iteration and if the difference is low then we can stop the training.

   d. Optional question: For what kind of problems will this solution fail?

Solution:
This solution can only be failed if there are unnecessary or not related training examples.

7. You are provided with a set of n training examples: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where xi is the input example, yi is the class label (+1 or -1). Suppose n is very large (say in the order of millions). In this case, online kernelized Perceptron algorithms will not scale if the number of allowed support vectors are unbounded.
   a. Suppose you have trained using kernelized Perceptron algorithm (without any bounds on support vectors) and got a set of support vectors SV. Tom wants to use this classifier for real-time prediction and cannot afford more than B kernel evaluations for each classification decision. Please give an algorithm to select B support vectors from SV. You need to motivate your design choices in order to convince Tom to use your solution.

Solution:
We are assuming that we have trained using kernelized perceptron algorithm without any bounds which means we have used all the training examples. Now let us suppose that we have X support vectors by training on the given n training examples. Since Tom can only use B kernel evaluations, we need to select B support vectors from X that are useful. By performing a greedy search approach, we can select B support vectors by removing the support vectors from X that contributes less towards the quality of function. Perform the same operation until the size of X is equal to that of B.

   b. Tom wants to train using kernelized Perceptron algorithm but wants to use at most B support vectors during the training process. Please modify the standard kernelized Perceptron training algorithm (from class slides) for this new setting. You need to motivate your design choices in order to convince Tom to use your solution.

Solution:
Since Tom wants to use at most B kernels, so for each training example we will draw a circle in such a way that it comprises only those examples that are of the same class. Count the number of examples in each circle and store it as N. If a training example has a large value of N then ignore the example as it will not be close to the decision boundary otherwise, select only the examples which have the smallest values for N which depicts that they are closer to the decision boundary. Run Kernelized Perceptron over the smaller dataset for only B support vectors. We run the algorithm over only those examples that are closer to the decision boundary and perform only B kernel evaluations which reduces time complexity.

Question 2 – Programming and Empirical Analysis Part

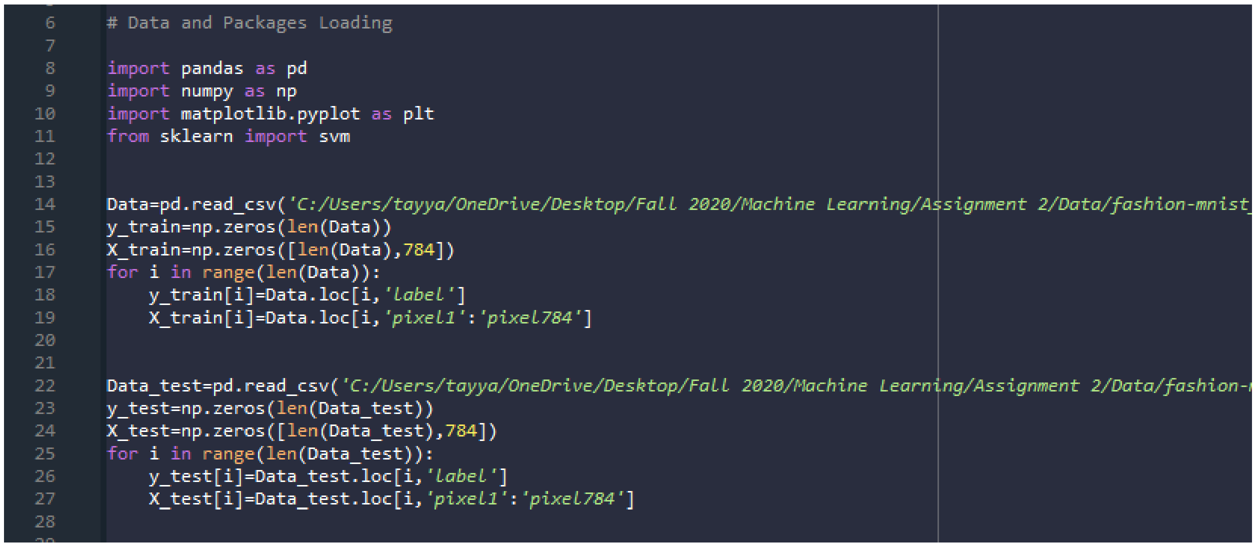
1. Empirical analysis question. You can use a publicly available SVM classifier implementation

(e.g., scikit-learn) for SVM related experiments. scikit-learn (http://scikit-learn.org/stable/modules/svm.html).

You will use the Fashion MNIST data (https://github.com/zalandoresearch/fashion-mnist). There is a fixed training and testing set. From training data, use first 80 percent for \training" and last 20 percent as \validation data".

Each example is a 28x28 grayscale image, associated with a label from 10 classes: 0 Tshirt/top, 1 Trouser, 2 Pullover, 3 Dress, 4 Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot.

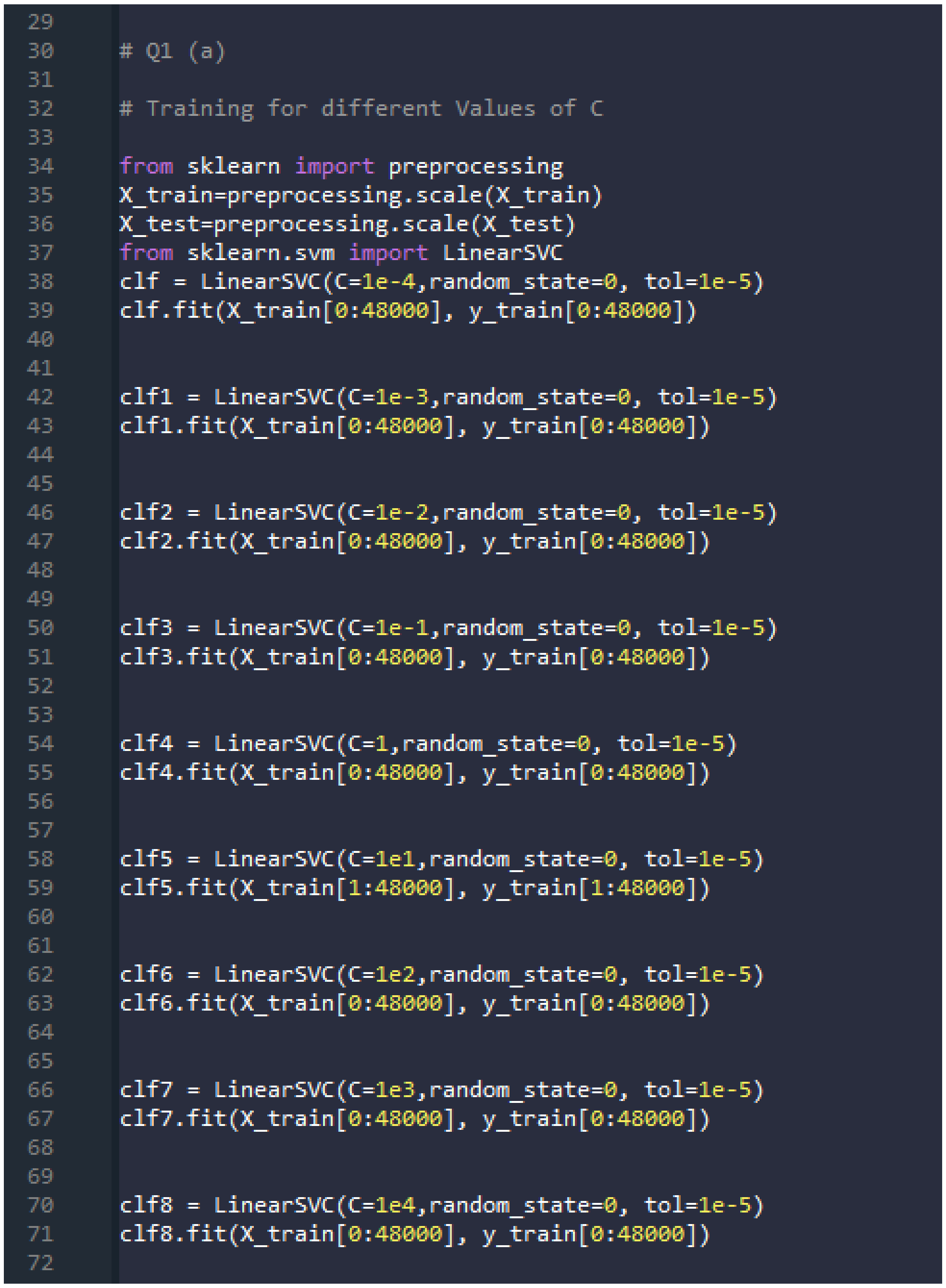
You will use ONLY training data for training and testing data for evaluation.



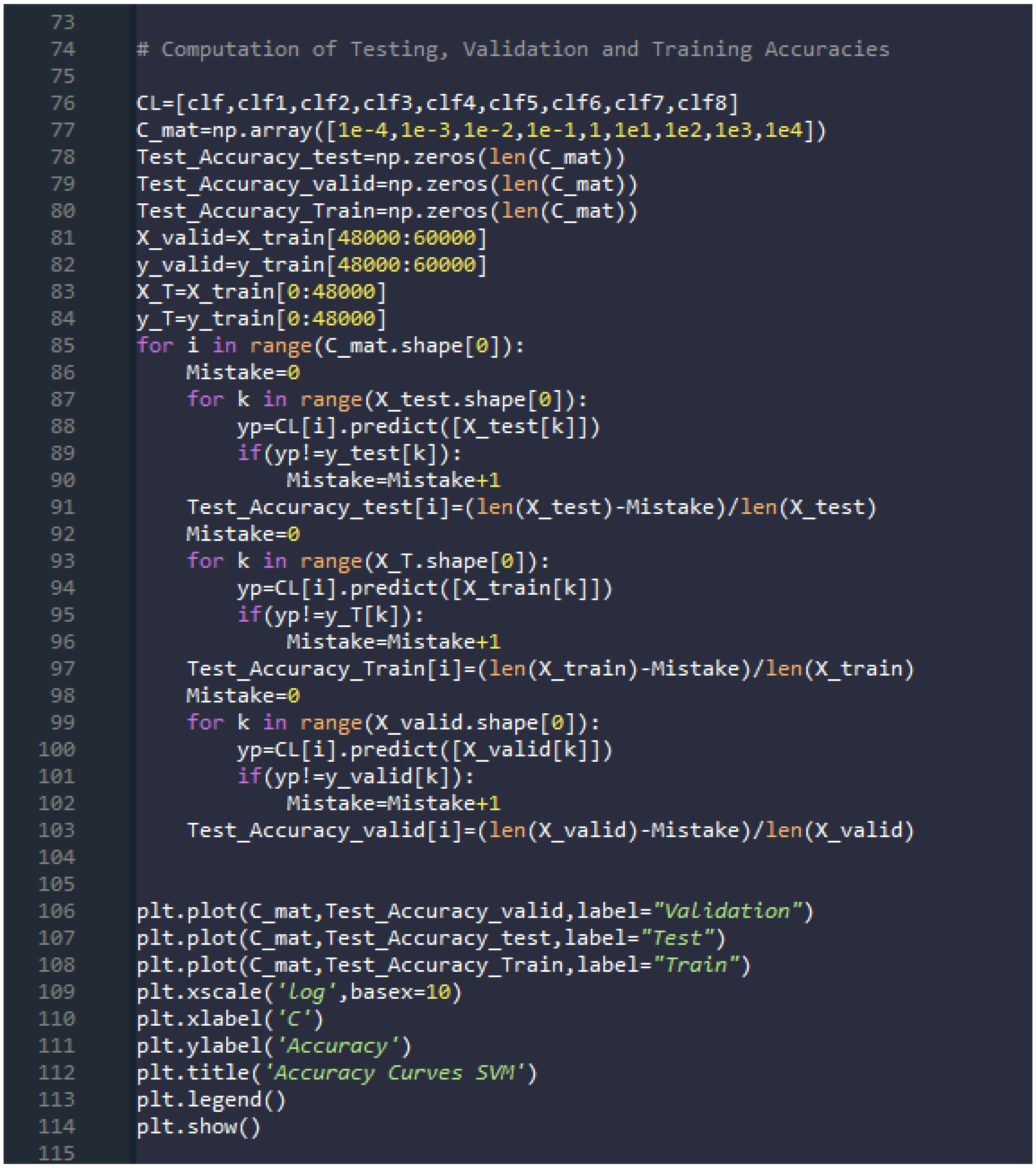
```
# Data and Packages Loading

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm


Data=pd.read_csv('C:/Users/tayya/OneDrive/Desktop/Fall 2020/Machine Learning/Assignment 2/Data/fashion-mnist_
y_train=np.zeros(len(Data))
X_train=np.zeros([len(Data),784])
for i in range(len(Data)):
    y_train[i]=Data.loc[i,'label']
    X_train[i]=Data.loc[i,'pixel1':'pixel784']


Data_test=pd.read_csv('C:/Users/tayya/OneDrive/Desktop/Fall 2020/Machine Learning/Assignment 2/Data/fashion-
y_test=np.zeros(len(Data_test))
X_test=np.zeros([len(Data_test),784])
for i in range(len(Data_test)):
    y_test[i]=Data_test.loc[i,'label']
    X_test[i]=Data_test.loc[i,'pixel1':'pixel784']

```

(a) Using a linear kernel, train the SVM on the training data for different values of *C* parameter: 10-4; 10-3; 10-2; 10-1; 100; 101; 102; 103; 104. Compute the training accuracy, validation accuracy, and testing accuracy for the SVM obtained with different values of the C parameter. Plot the training accuracy, validation accuracy, and testing accuracy as a function of *C* (*C* value on x-axis and Accuracy on y-axis) { one curve each for training, validation, and testing data. Also, plot the number of support vectors (if applicable for the SVM toolkit you are using) as a function of *C*. List your observations.

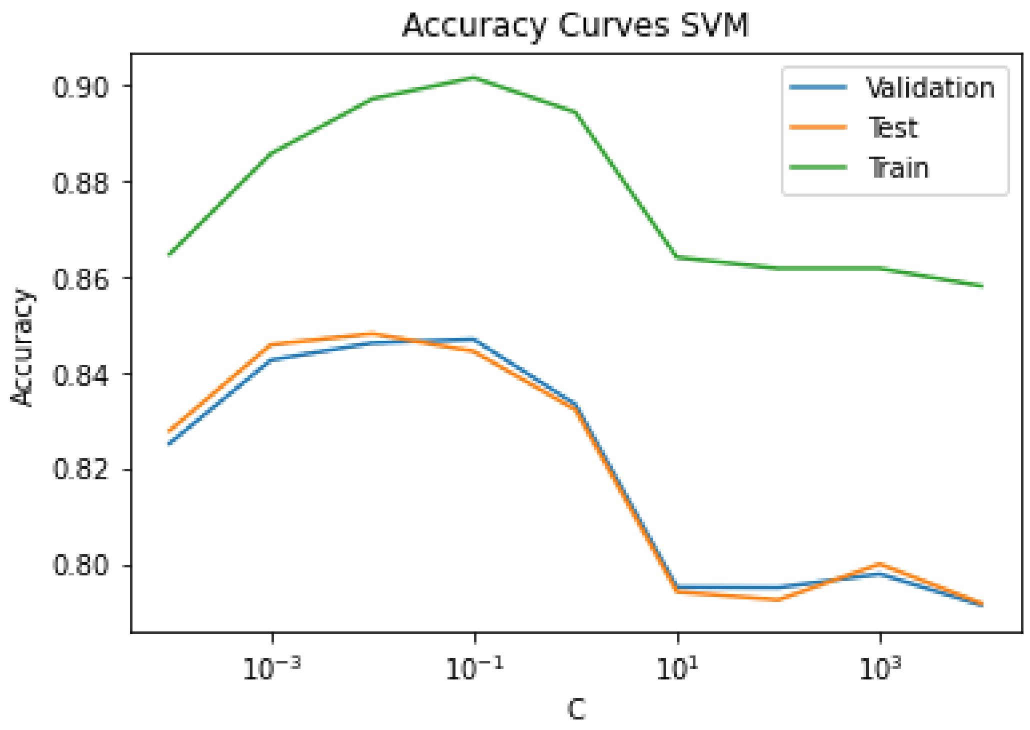
Solution:



```

# Q1 (a)

# Training for different Values of C

from sklearn import preprocessing
X_train=preprocessing.scale(X_train)
X_test=preprocessing.scale(X_test)
from sklearn.svm import LinearSVC
clf = LinearSVC(C=1e-4,random_state=0, tol=1e-5)
clf.fit(X_train[0:48000], y_train[0:48000])


clf1 = LinearSVC(C=1e-3,random_state=0, tol=1e-5)
clf1.fit(X_train[0:48000], y_train[0:48000])


clf2 = LinearSVC(C=1e-2,random_state=0, tol=1e-5)
clf2.fit(X_train[0:48000], y_train[0:48000])


clf3 = LinearSVC(C=1e-1,random_state=0, tol=1e-5)
clf3.fit(X_train[0:48000], y_train[0:48000])


clf4 = LinearSVC(C=1,random_state=0, tol=1e-5)
clf4.fit(X_train[0:48000], y_train[0:48000])


clf5 = LinearSVC(C=1e1,random_state=0, tol=1e-5)
clf5.fit(X_train[1:48000], y_train[1:48000])


clf6 = LinearSVC(C=1e2,random_state=0, tol=1e-5)
clf6.fit(X_train[0:48000], y_train[0:48000])


clf7 = LinearSVC(C=1e3,random_state=0, tol=1e-5)
clf7.fit(X_train[0:48000], y_train[0:48000])


clf8 = LinearSVC(C=1e4,random_state=0, tol=1e-5)
clf8.fit(X_train[0:48000], y_train[0:48000])

```



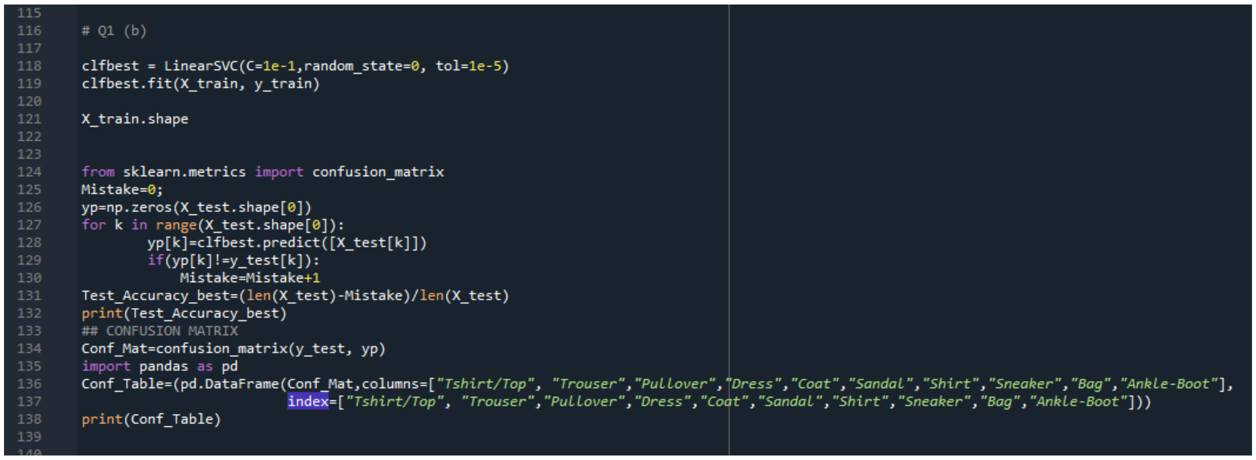
```python

# Computation of Testing, Validation and Training Accuracies

CL=[clf,clf1,clf2,clf3,clf4,clf5,clf6,clf7,clf8]
C_mat=np.array([1e-4,1e-3,1e-2,1e-1,1,1e1,1e2,1e3,1e4])
Test_Accuracy_test=np.zeros(len(C_mat))
Test_Accuracy_valid=np.zeros(len(C_mat))
Test_Accuracy_Train=np.zeros(len(C_mat))
X_valid=X_train[48000:60000]
y_valid=y_train[48000:60000]
X_T=X_train[0:48000]
y_T=y_train[0:48000]
for i in range(C_mat.shape[0]):
    Mistake=0
    for k in range(X_test.shape[0]):
        yp=CL[i].predict([X_test[k]])
        if(yp!=y_test[k]):
            Mistake=Mistake+1
    Test_Accuracy_test[i]=(len(X_test)-Mistake)/len(X_test)
    Mistake=0
    for k in range(X_T.shape[0]):
        yp=CL[i].predict([X_train[k]])
        if(yp!=y_T[k]):
            Mistake=Mistake+1
    Test_Accuracy_Train[i]=(len(X_train)-Mistake)/len(X_train)
    Mistake=0
    for k in range(X_valid.shape[0]):
        yp=CL[i].predict([X_valid[k]])
        if(yp!=y_valid[k]):
            Mistake=Mistake+1
    Test_Accuracy_valid[i]=(len(X_valid)-Mistake)/len(X_valid)


plt.plot(C_mat,Test_Accuracy_valid,label="Validation")
plt.plot(C_mat,Test_Accuracy_test,label="Test")
plt.plot(C_mat,Test_Accuracy_Train,label="Train")
plt.xscale('log',basex=10)
plt.xlabel('C')
plt.ylabel('Accuracy')
plt.title('Accuracy Curves SVM')
plt.legend()
plt.show()

```



As can be seen from graph above, the best value of C based on Validation accuracy is C=1e-1. Also, it is observed that validation accuracy is good approximation of test accuracy or the true error.

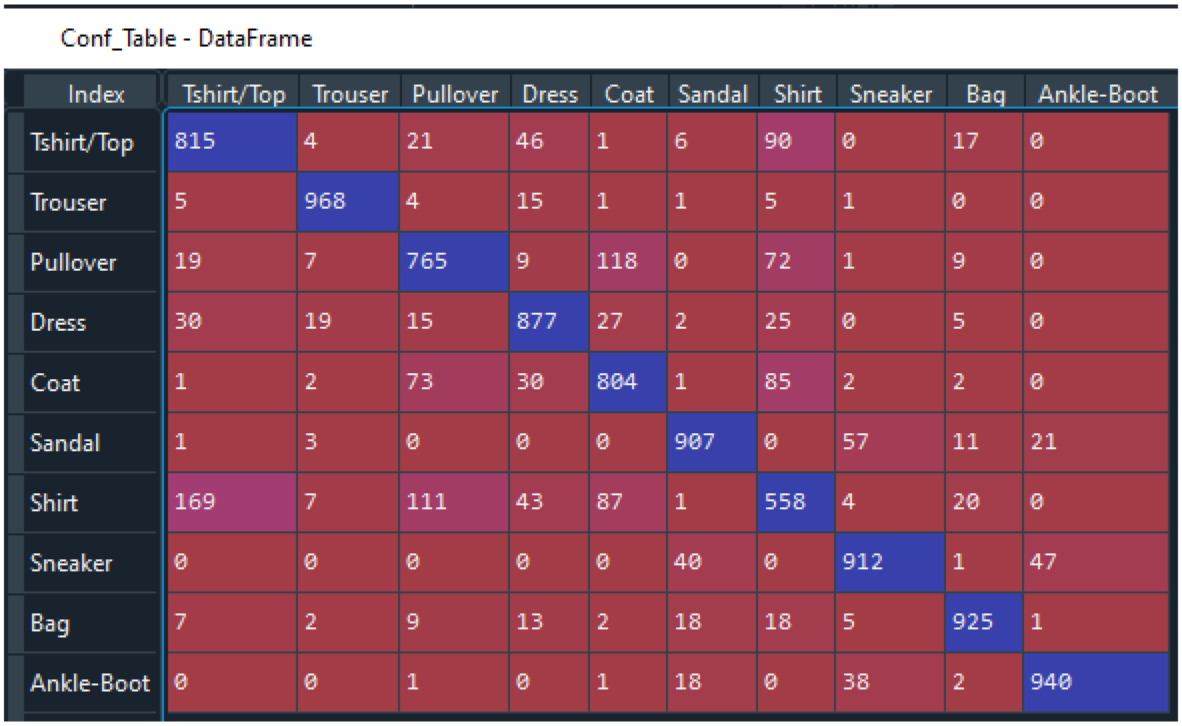
(b) Select the best value of hyper-parameter *C* based on the accuracy on validation set and train a linear SVM on the *combined* set of training and validation examples. Compute the testing accuracy and the corresponding confusion matrix: a 10 × 10 matrix.

Solution:



```python

# Q1 (b)

clfbest = LinearSVC(C=1e-1,random_state=0, tol=1e-5)
clfbest.fit(X_train, y_train)

X_train.shape


from sklearn.metrics import confusion_matrix
Mistake=0;
yp=np.zeros(X_test.shape[0])
for k in range(X_test.shape[0]):
        yp[k]=clfbest.predict([X_test[k]])
        if(yp[k]!=y_test[k]):
            Mistake=Mistake+1
Test_Accuracy_best=(len(X_test)-Mistake)/len(X_test)
print(Test_Accuracy_best)
## CONFUSION MATRIX
Conf_Mat=confusion_matrix(y_test, yp)
import pandas as pd
Conf_Table=(pd.DataFrame(Conf_Mat,columns=["Tshirt/Top", "Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle-Boot"],
                         index=["Tshirt/Top", "Trouser","Pullover","Dress","Coat","Sandal","Shirt","Sneaker","Bag","Ankle-Boot"]))
print(Conf_Table)

```

In [96]: X_train.shape
Out [96]: (60000, 784)

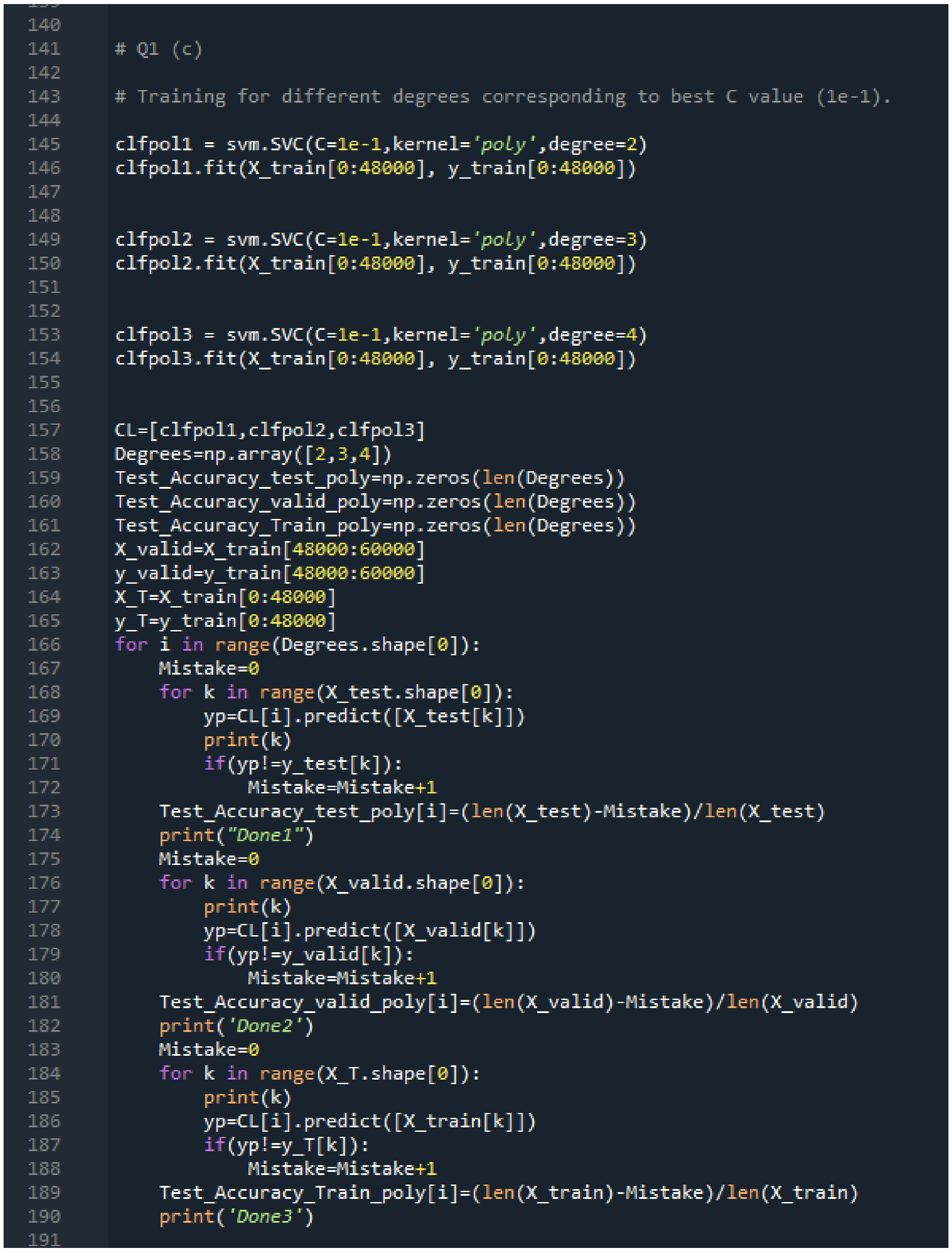


Conf_Table - DataFrame

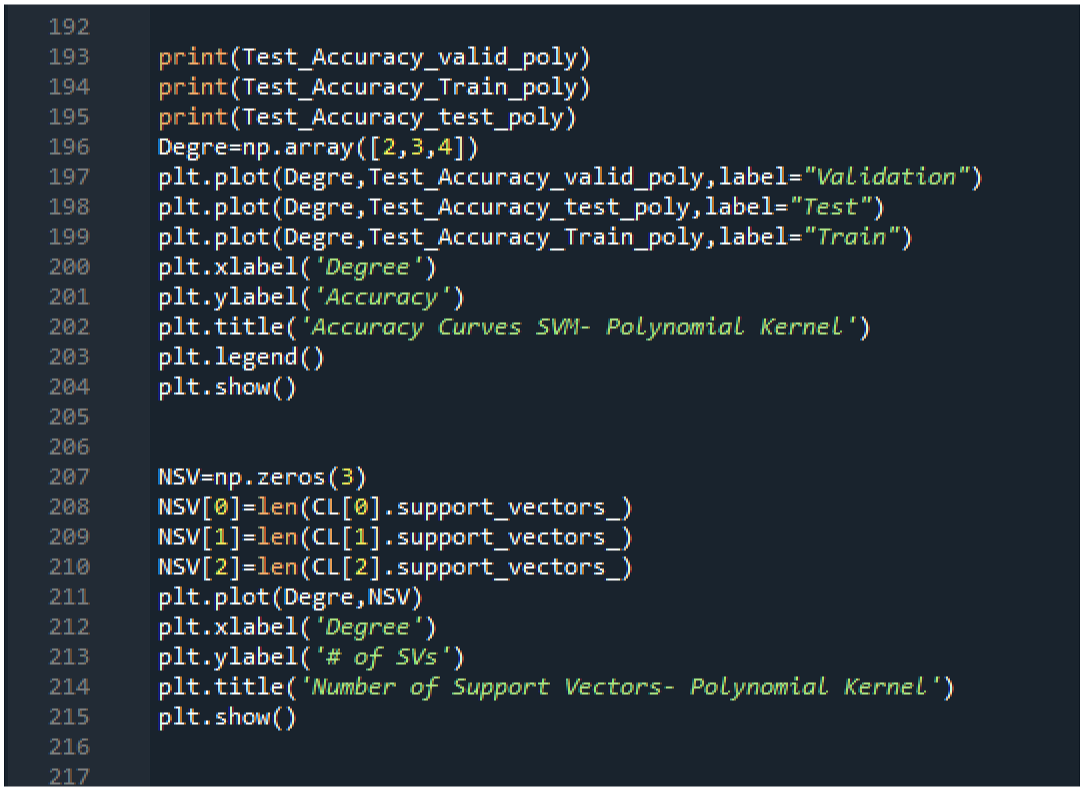
Index	Tshirt/Top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle-Boot
Tshirt/Top	815	4	21	46	1	6	90	0	17	0
Trouser	5	968	4	15	1	1	5	1	0	0
Pullover	19	7	765	9	118	0	72	1	9	0
Dress	30	19	15	877	27	2	25	0	5	0
Coat	1	2	73	30	804	1	85	2	2	0
Sandal	1	3	0	0	0	907	0	57	11	21
Shirt	169	7	111	43	87	1	558	4	20	0
Sneaker	0	0	0	0	0	40	0	912	1	47
Bag	7	2	9	13	2	18	18	5	925	1
Ankle-Boot	0	0	1	0	1	18	0	38	2	940

(c) Repeat the experiment (a) with the best *C* value from (a) with polynomial kernel of degree 2, 3, and 4. Compare the training, validation, testing accuracies, and the number of support vectors for different kernels (linear, polynomial kernel of degree 2, polynomial kernel of degree 3, and polynomial kernel of degree 4). List your observations.

Solution:



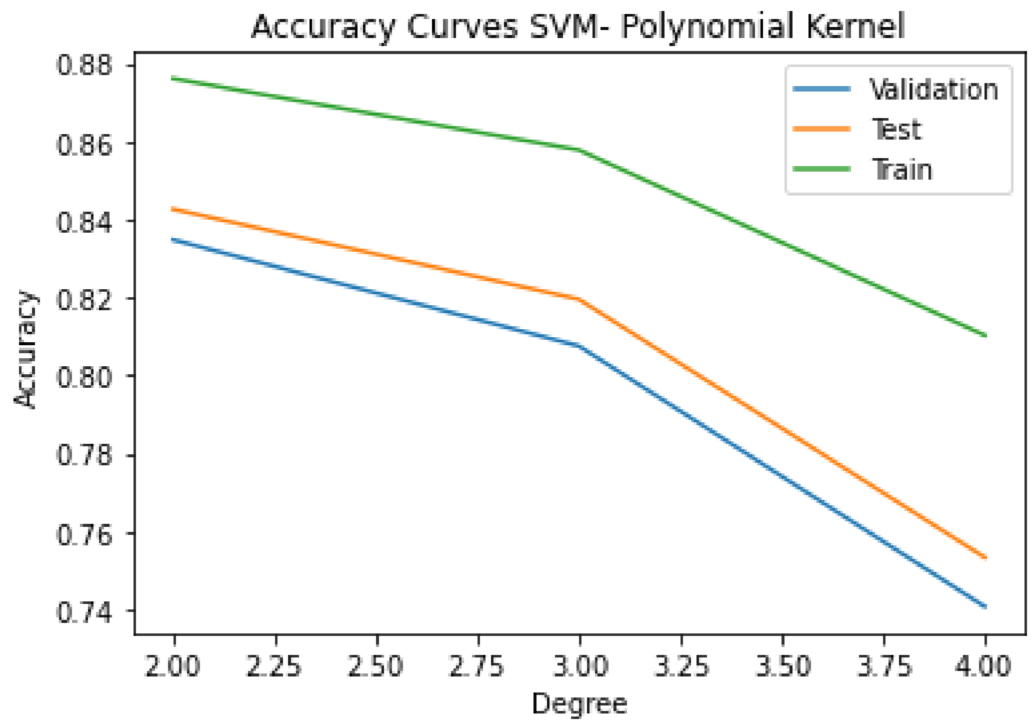
```python
# Q1 (c)

# Training for different degrees corresponding to best C value (1e-1).

clfpol1 = svm.SVC(C=1e-1,kernel='poly',degree=2)
clfpol1.fit(X_train[0:48000], y_train[0:48000])


clfpol2 = svm.SVC(C=1e-1,kernel='poly',degree=3)
clfpol2.fit(X_train[0:48000], y_train[0:48000])


clfpol3 = svm.SVC(C=1e-1,kernel='poly',degree=4)
clfpol3.fit(X_train[0:48000], y_train[0:48000])


CL=[clfpol1,clfpol2,clfpol3]
Degrees=np.array([2,3,4])
Test_Accuracy_test_poly=np.zeros(len(Degrees))
Test_Accuracy_valid_poly=np.zeros(len(Degrees))
Test_Accuracy_Train_poly=np.zeros(len(Degrees))
X_valid=X_train[48000:60000]
y_valid=y_train[48000:60000]
X_T=X_train[0:48000]
y_T=y_train[0:48000]
for i in range(Degrees.shape[0]):
    Mistake=0
    for k in range(X_test.shape[0]):
        yp=CL[i].predict([X_test[k]])
        print(k)
        if(yp!=y_test[k]):
            Mistake=Mistake+1
    Test_Accuracy_test_poly[i]=(len(X_test)-Mistake)/len(X_test)
    print("Done1")
    Mistake=0
    for k in range(X_valid.shape[0]):
        print(k)
        yp=CL[i].predict([X_valid[k]])
        if(yp!=y_valid[k]):
            Mistake=Mistake+1
    Test_Accuracy_valid_poly[i]=(len(X_valid)-Mistake)/len(X_valid)
    print('Done2')
    Mistake=0
    for k in range(X_T.shape[0]):
        print(k)
        yp=CL[i].predict([X_train[k]])
        if(yp!=y_T[k]):
            Mistake=Mistake+1
    Test_Accuracy_Train_poly[i]=(len(X_train)-Mistake)/len(X_train)
    print('Done3')
```



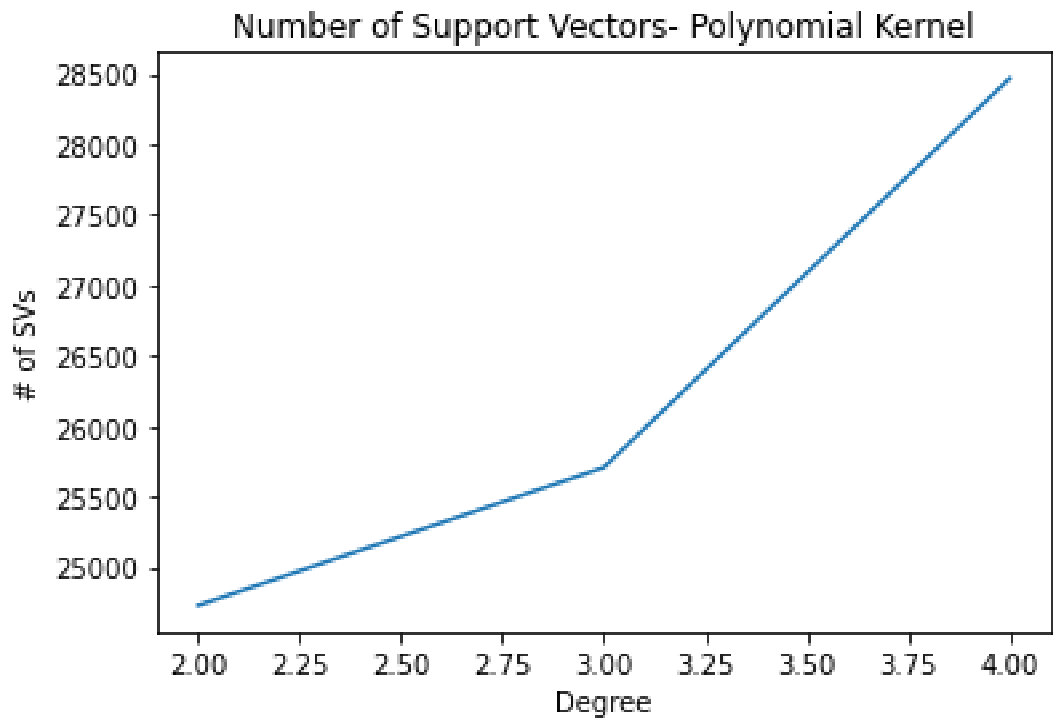
```python

print(Test_Accuracy_valid_poly)
print(Test_Accuracy_Train_poly)
print(Test_Accuracy_test_poly)
Degre=np.array([2,3,4])
plt.plot(Degre,Test_Accuracy_valid_poly,label="Validation")
plt.plot(Degre,Test_Accuracy_test_poly,label="Test")
plt.plot(Degre,Test_Accuracy_Train_poly,label="Train")
plt.xlabel('Degree')
plt.ylabel('Accuracy')
plt.title('Accuracy Curves SVM- Polynomial Kernel')
plt.legend()
plt.show()


NSV=np.zeros(3)
NSV[0]=len(CL[0].support_vectors_)
NSV[1]=len(CL[1].support_vectors_)
NSV[2]=len(CL[2].support_vectors_)
plt.plot(Degre,NSV)
plt.xlabel('Degree')
plt.ylabel('# of SVs')
plt.title('Number of Support Vectors- Polynomial Kernel')
plt.show()

```

Training for different degrees corresponding to best C value (1e-1).

```
[0.83475    0.8075     0.74066667]
[0.87608333     0.85778333      0.81025    ]
[0.8426    0.8195    0.7533]
```



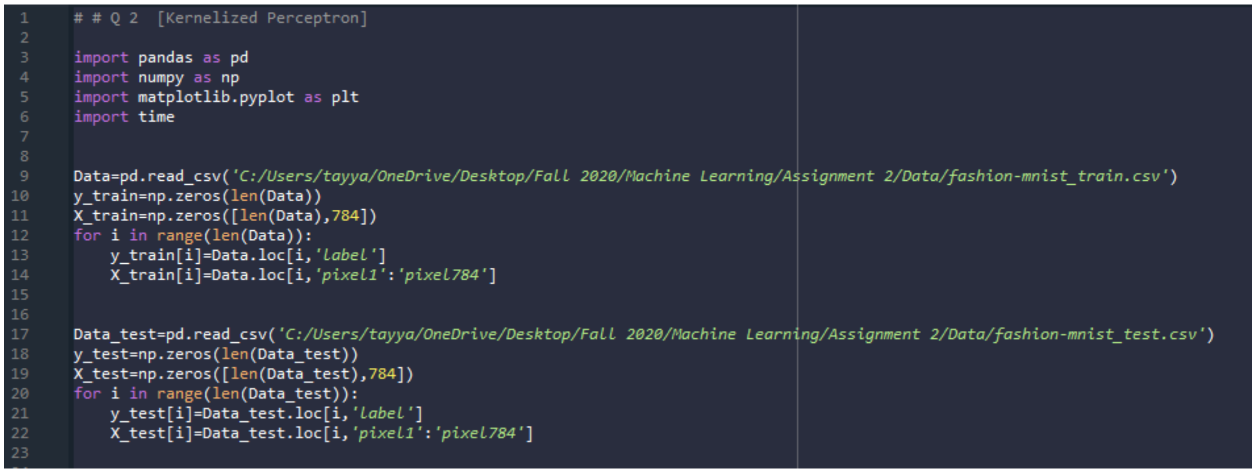
It is observed from the above graph that best degree based on validation accuracy is 2.00.



Degree 2 corresponds to least number of support vectors.

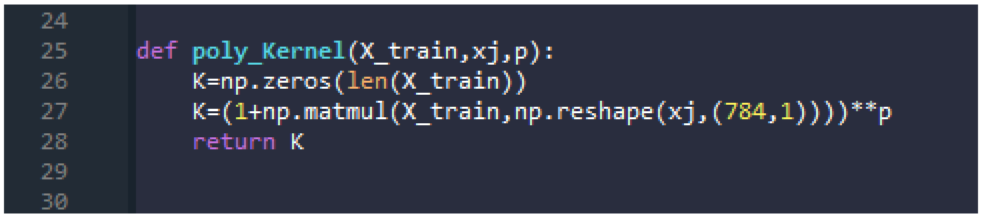
2. Programming question. You will implement the kernelized Perceptron training algorithm (discussed in the class) for *multi-class* classification.

(a) You will use the Fashion MNIST data. Train the kernelized Perceptron classifier for 5 iterations with polynomial kernel (pick the best degree out of 2, 3, and 4 from the above experiment). Plot the number of mistakes as a function of training iterations. Compute the training, validation, and testing accuracy at the end of 5 iterations.



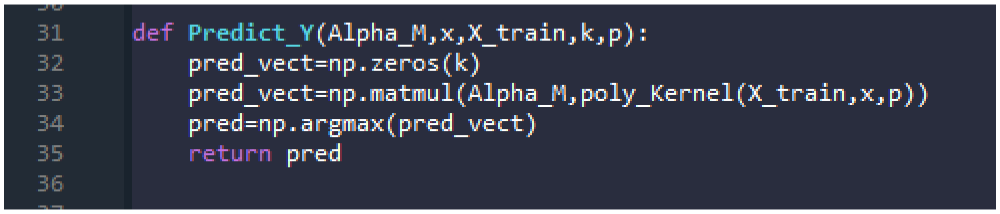
```python
# # Q 2  [Kernelized Perceptron]

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time


Data=pd.read_csv('C:/Users/tayya/OneDrive/Desktop/Fall 2020/Machine Learning/Assignment 2/Data/fashion-mnist_train.csv')
y_train=np.zeros(len(Data))
X_train=np.zeros([len(Data),784])
for i in range(len(Data)):
    y_train[i]=Data.loc[i,'label']
    X_train[i]=Data.loc[i,'pixel1':'pixel784']


Data_test=pd.read_csv('C:/Users/tayya/OneDrive/Desktop/Fall 2020/Machine Learning/Assignment 2/Data/fashion-mnist_test.csv')
y_test=np.zeros(len(Data_test))
X_test=np.zeros([len(Data_test),784])
for i in range(len(Data_test)):
    y_test[i]=Data_test.loc[i,'label']
    X_test[i]=Data_test.loc[i,'pixel1':'pixel784']

```

This function below takes whole X_train as input and computes Kernel of test feature vector with each training example feature. Output is a row vector of length 1xN where N is the number of training examples.



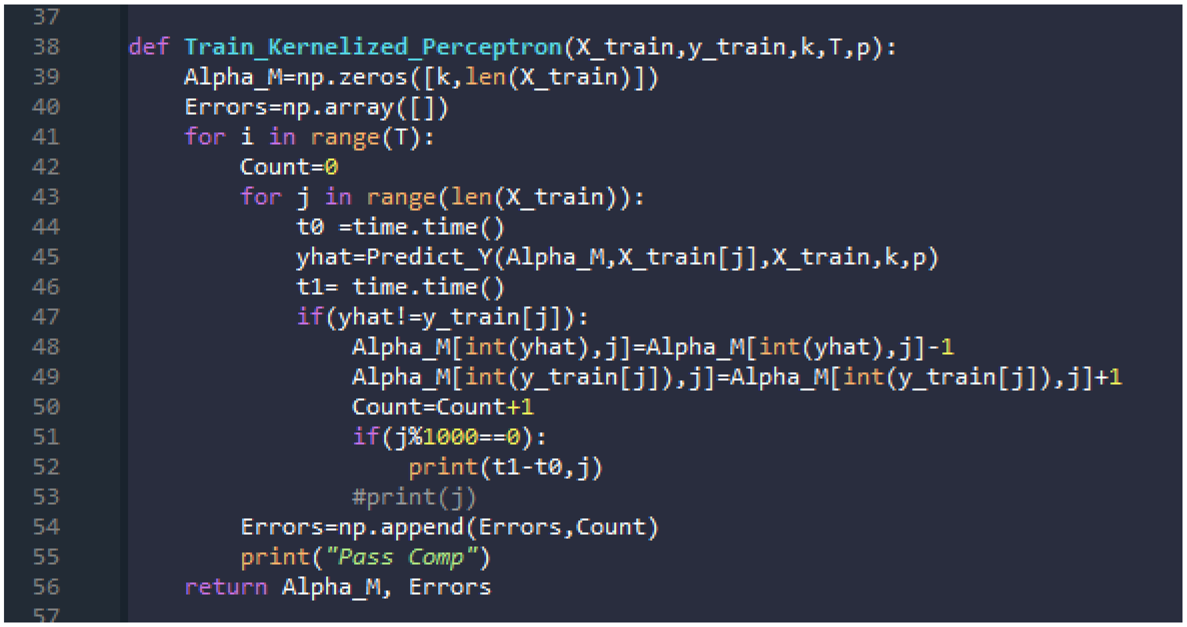
```python

def poly_Kernel(X_train,xj,p):
    K=np.zeros(len(X_train))
    K=(1+np.matmul(X_train,np.reshape(xj,(784,1))))**p
    return K


```

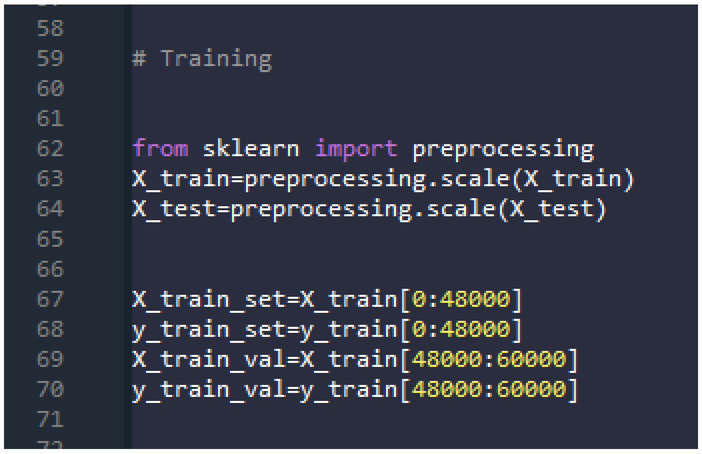
This function below takes Alpha matrix, X_train, number of classes (k) and degree of Polynomial kernel as input and predicts the output for a test feature.

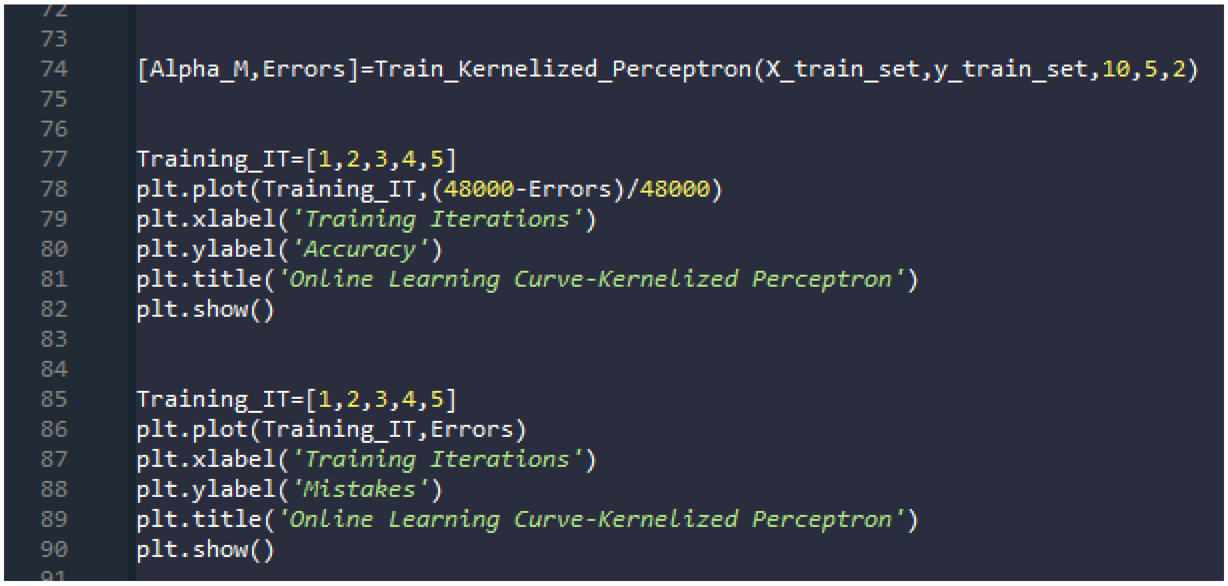


```python
def Predict_Y(Alpha_M,x,X_train,k,p):
    pred_vect=np.zeros(k)
    pred_vect=np.matmul(Alpha_M,poly_Kernel(X_train,x,p))
    pred=np.argmax(pred_vect)
    return pred

```

This function below trains the kernelized perceptron for a polynomial kernel of degree “p”. User also specifies the number of classes (k) and number of training passes (T)


```python

def Train_Kernelized_Perceptron(X_train,y_train,k,T,p):
    Alpha_M=np.zeros([k,len(X_train)])
    Errors=np.array([])
    for i in range(T):
        Count=0
        for j in range(len(X_train)):
            t0 =time.time()
            yhat=Predict_Y(Alpha_M,X_train[j],X_train,k,p)
            t1= time.time()
            if(yhat!=y_train[j]):
                Alpha_M[int(yhat),j]=Alpha_M[int(yhat),j]-1
                Alpha_M[int(y_train[j]),j]=Alpha_M[int(y_train[j]),j]+1
                Count=Count+1
                if(j%1000==0):
                    print(t1-t0,j)
                #print(j)
        Errors=np.append(Errors,Count)
        print("Pass Comp")
    return Alpha_M, Errors

```


```python

# Training


from sklearn import preprocessing
X_train=preprocessing.scale(X_train)
X_test=preprocessing.scale(X_test)


X_train_set=X_train[0:48000]
y_train_set=y_train[0:48000]
X_train_val=X_train[48000:60000]
y_train_val=y_train[48000:60000]

```

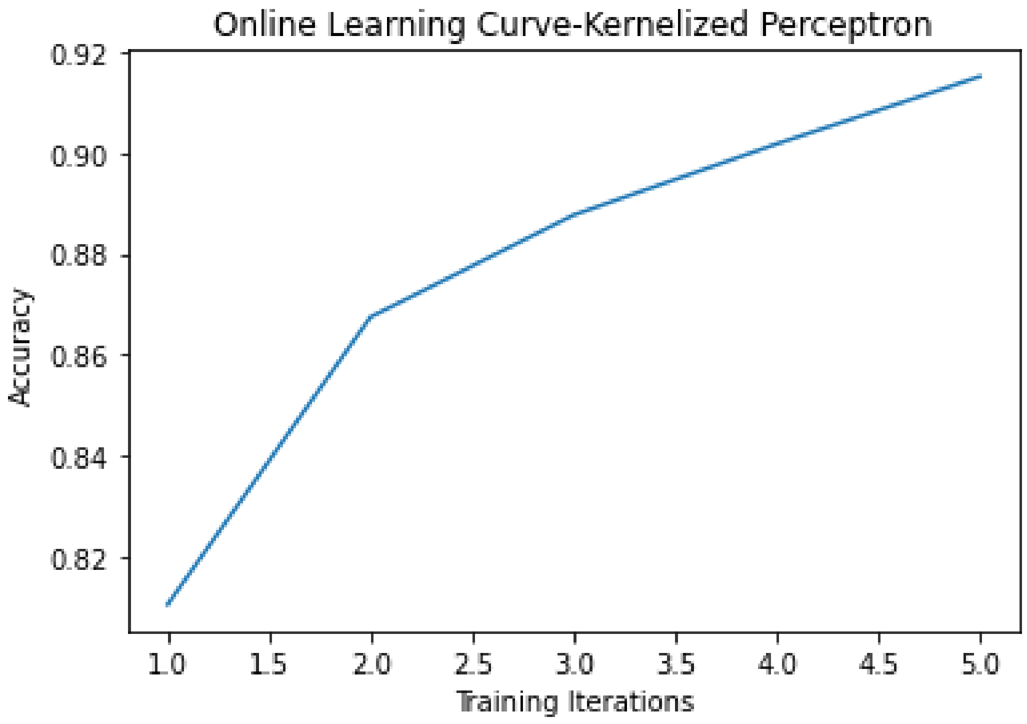

```python
[Alpha_M,Errors]=Train_Kernelized_Perceptron(X_train_set,y_train_set,10,5,2)


Training_IT=[1,2,3,4,5]
plt.plot(Training_IT,(48000-Errors)/48000)
plt.xlabel('Training Iterations')
plt.ylabel('Accuracy')
plt.title('Online Learning Curve-Kernelized Perceptron')
plt.show()


Training_IT=[1,2,3,4,5]
plt.plot(Training_IT,Errors)
plt.xlabel('Training Iterations')
plt.ylabel('Mistakes')
plt.title('Online Learning Curve-Kernelized Perceptron')
plt.show()
```

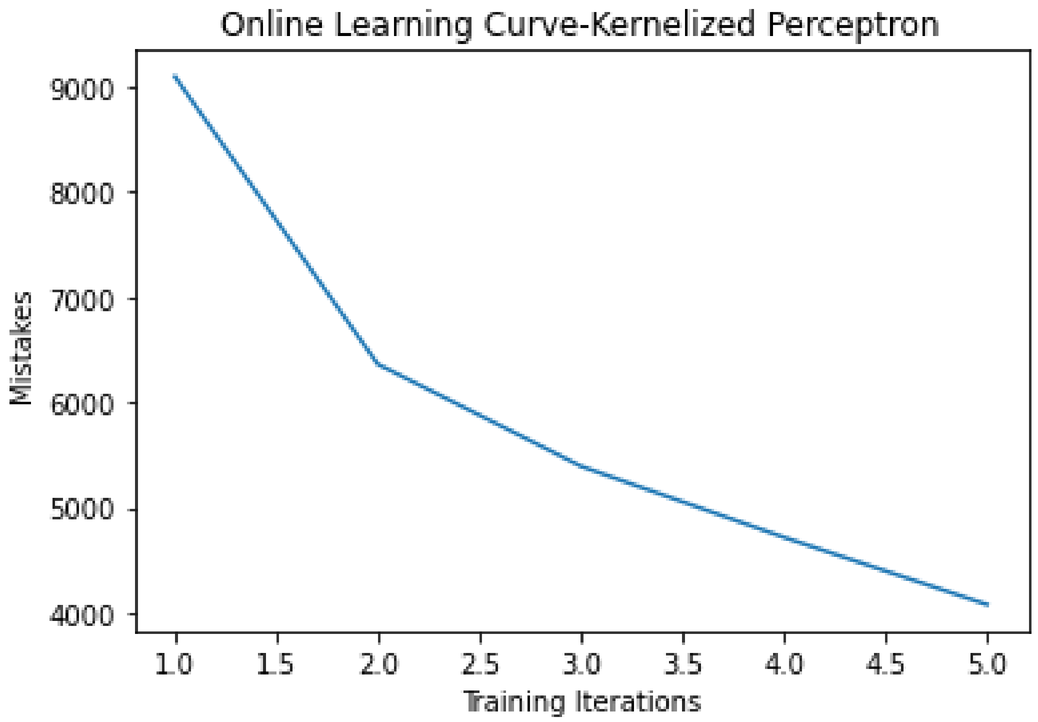
[Alpha_M,Errors]=Train_Kernelized_Perceptron(X_train_set,y_train_set,10,5,2)

Output:

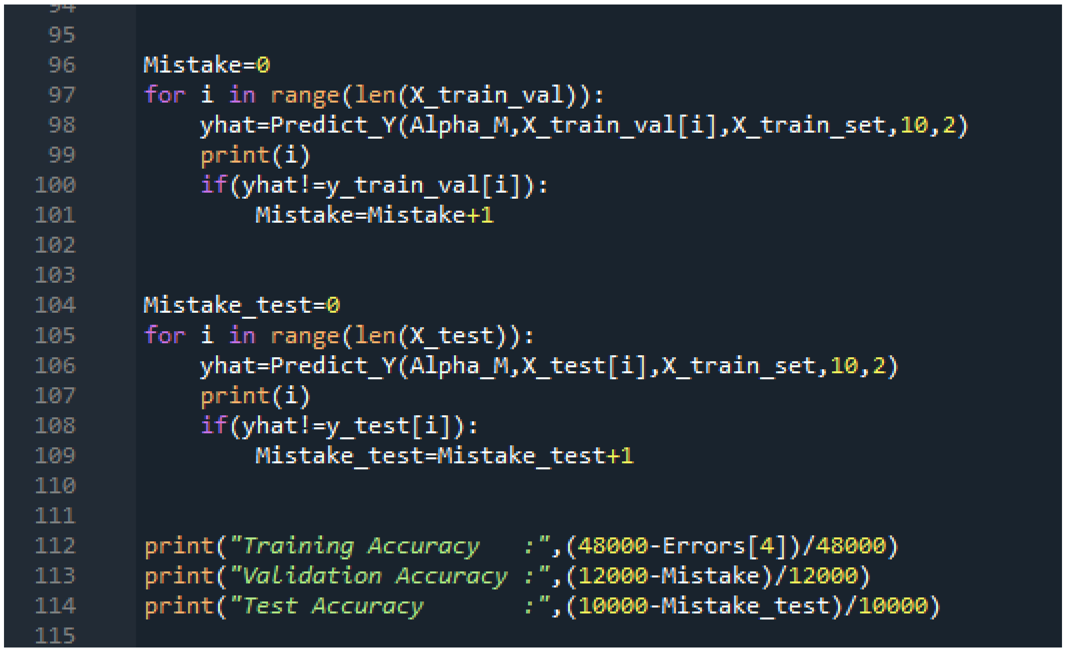
```
0.06251454353333252 0

0.015633344650268555 4000

0.01561427116394043 9000

0.01563096046447754 12000

0.015628576278686523 17000

0.030728578567504883 22000

0.015624046325683594 25000

0.01562666893005371 26000

0.0159912109375 47000

Pass Comp

0.015986919403076172 5000

0.015638351440429688 9000

0.014977693557739258 12000

0.013988733291625977 17000

0.006290674209594726 22000

0.014012813568115234 25000

0.01497650146484375 42000
```

```
0.015624761581420898 47000

Pass Comp

0.01564311981201172 0

0.014961719512939453 8000

0.015614986419677734 17000

0.01599264144897461 25000

0.01499319076538086 47000

Pass Comp

0.015989303588867188 17000

0.014985084533691406 25000

Pass Comp

0.01599287986755371 12000

0.015989303588867188 25000

Pass Comp
```





Comparison of the Training, Test, and validation Accuracy



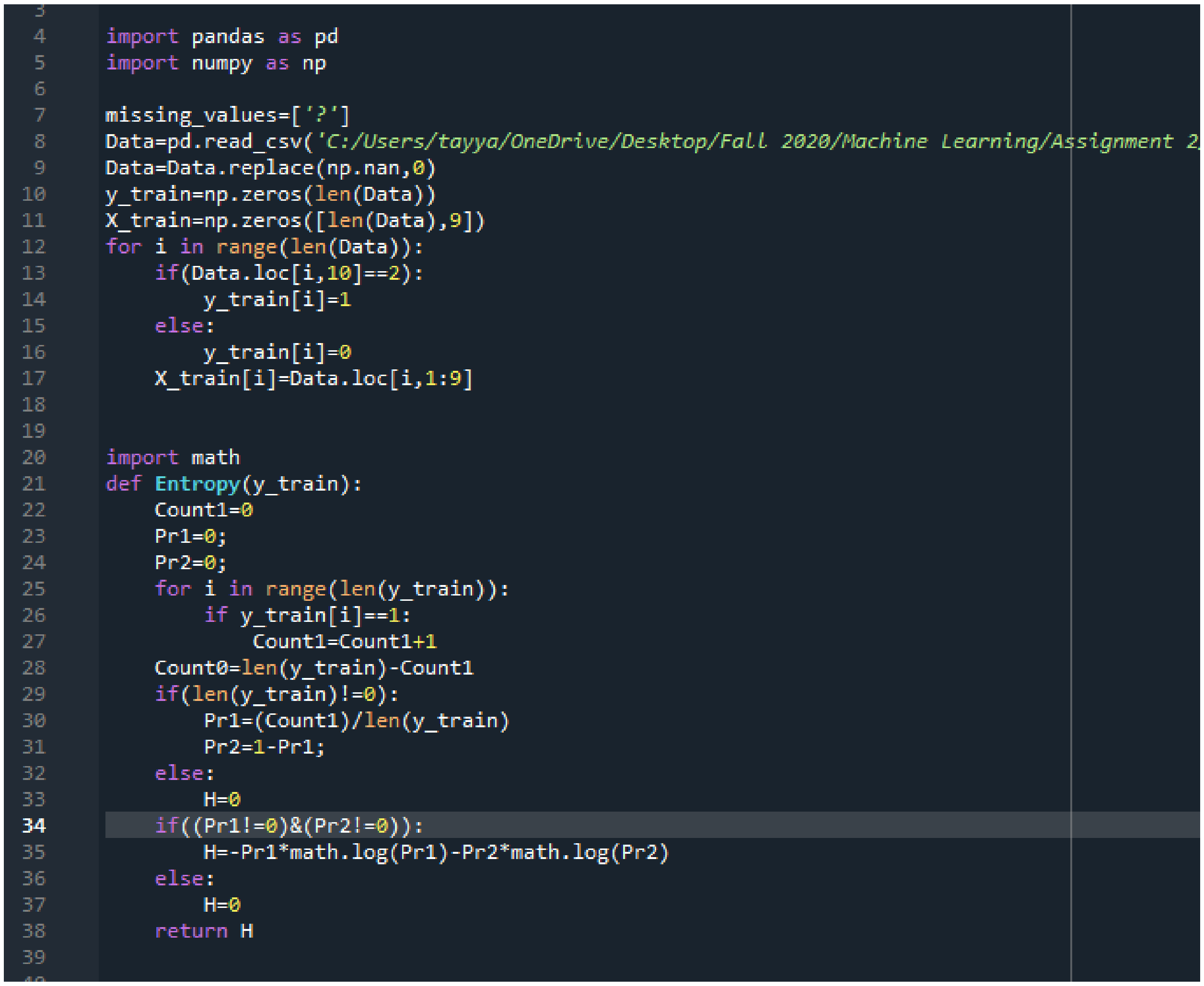
```
Mistake=0
for i in range(len(X_train_val)):
    yhat=Predict_Y(Alpha_M,X_train_val[i],X_train_set,10,2)
    print(i)
    if(yhat!=y_train_val[i]):
        Mistake=Mistake+1


Mistake_test=0
for i in range(len(X_test)):
    yhat=Predict_Y(Alpha_M,X_test[i],X_train_set,10,2)
    print(i)
    if(yhat!=y_test[i]):
        Mistake_test=Mistake_test+1


print("Training Accuracy   :",(48000-Errors[4])/48000)
print("Validation Accuracy :",(12000-Mistake)/12000)
print("Test Accuracy       :",(10000-Mistake_test)/10000)
```

Training Accuracy: 0.9150208333333333

Validation Accuracy: 0.8583333333333333

Test Accuracy: 0.8639

3. Programming question. \Breast Cancer" Classifier using Decision Trees. You will use the following dataset for this question: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29. You will use the first 70 percent examples for training, next 10 percent examples for validation, and last 20 percent examples for testing.

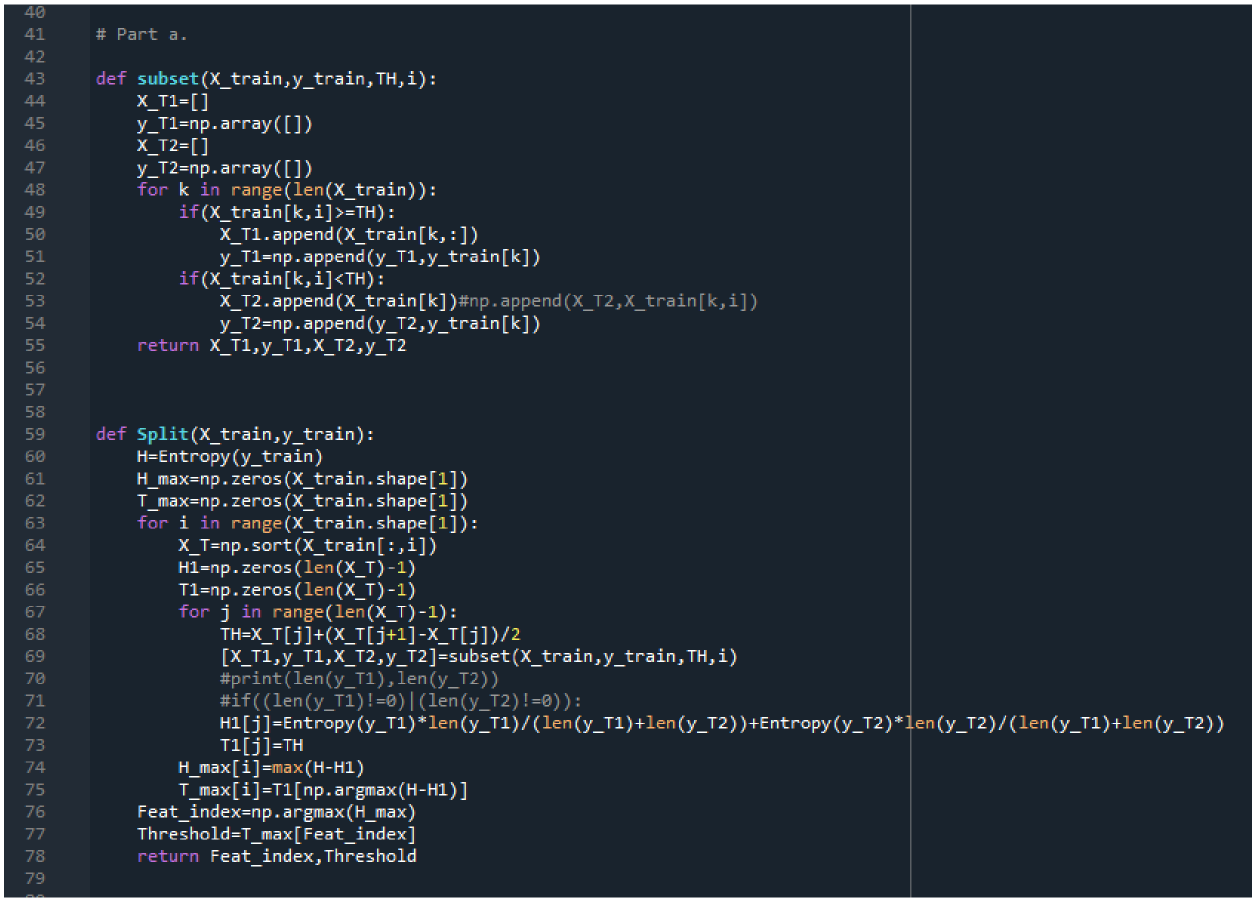


```
import pandas as pd
import numpy as np

missing_values=['?']
Data=pd.read_csv('C:/Users/tayya/OneDrive/Desktop/Fall 2020/Machine Learning/Assignment 2
Data=Data.replace(np.nan,0)
y_train=np.zeros(len(Data))
X_train=np.zeros([len(Data),9])
for i in range(len(Data)):
    if(Data.loc[i,10]==2):
        y_train[i]=1
    else:
        y_train[i]=0
    X_train[i]=Data.loc[i,1:9]


import math
def Entropy(y_train):
    Count1=0
    Pr1=0;
    Pr2=0;
    for i in range(len(y_train)):
        if y_train[i]==1:
            Count1=Count1+1
    Count0=len(y_train)-Count1
    if(len(y_train)!=0):
        Pr1=(Count1)/len(y_train)
        Pr2=1-Pr1;
    else:
        H=0
    if((Pr1!=0)&(Pr2!=0)):
        H=-Pr1*math.log(Pr1)-Pr2*math.log(Pr2)
    else:
        H=0
    return H
```

(a) Implement the ID3 decision tree learning algorithm that we discussed in the class. The key step in the decision tree learning is choosing the next feature to split on. Implement the information gain heuristic for selecting the next feature. Please see lecture notes or https://en.wikipedia.org/wiki/ID3_algorithm for more details. Do the following to select candidate thresholds for continuous features: Sort all candidate values for feature *f* from training data. Suppose *f*1; *f*2; · · · ; *fn* is the sorted list. The candidate thresholds are chosen as *fi* + (*fi*+1 - *fi*)=2 for *i*=1 to *n*.

Solution:

- Here the function "Split" choses the best feature and corresponding threshold based on Maximum Information gain. Output of this function are "Feat_Index (Best Feature)" and corresponding best threshold "Threshold".
- Function split divides the data into two parts by taking data, threshold and index of the feature as input

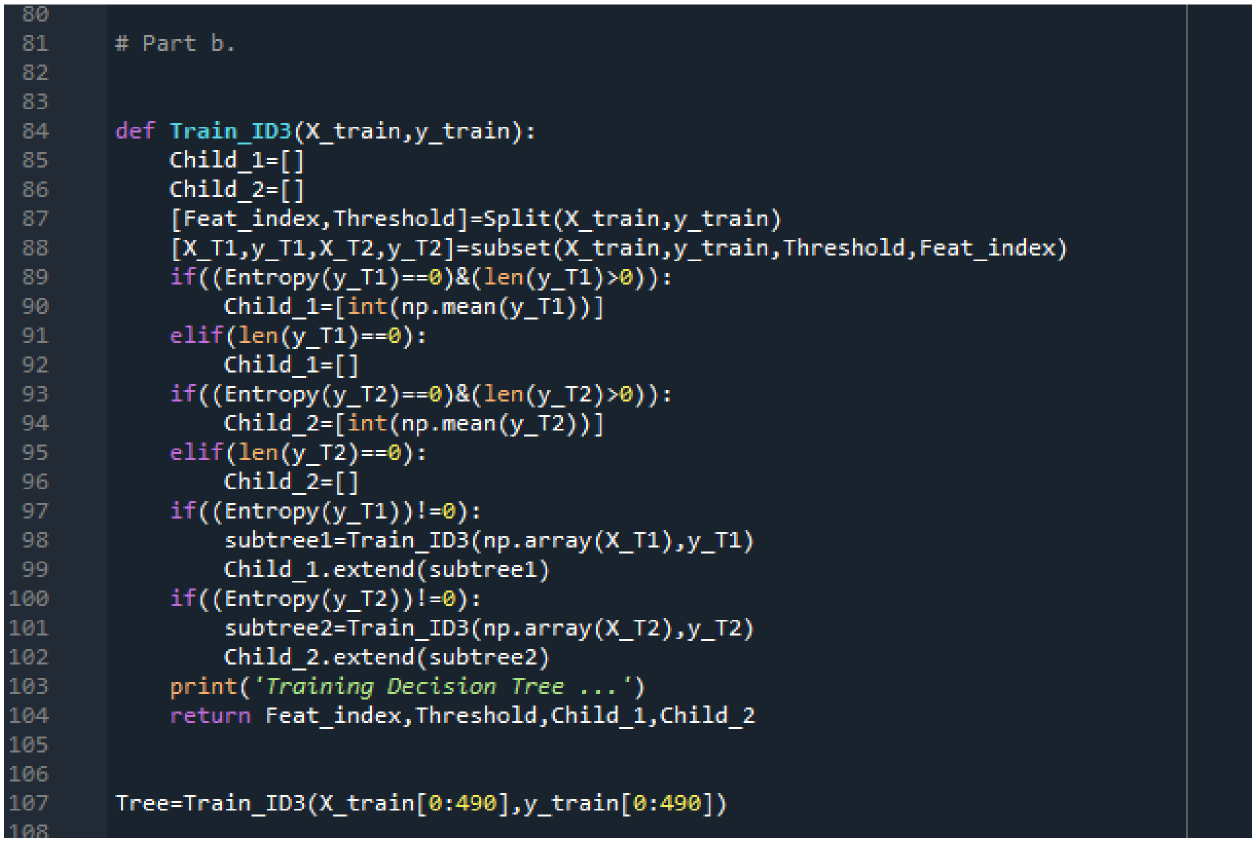


```
# Part a.

def subset(X_train,y_train,TH,i):
    X_T1=[]
    y_T1=np.array([])
    X_T2=[]
    y_T2=np.array([])
    for k in range(len(X_train)):
        if(X_train[k,i]>=TH):
            X_T1.append(X_train[k,:])
            y_T1=np.append(y_T1,y_train[k])
        if(X_train[k,i]<TH):
            X_T2.append(X_train[k])#np.append(X_T2,X_train[k,i])
            y_T2=np.append(y_T2,y_train[k])
    return X_T1,y_T1,X_T2,y_T2



def Split(X_train,y_train):
    H=Entropy(y_train)
    H_max=np.zeros(X_train.shape[1])
    T_max=np.zeros(X_train.shape[1])
    for i in range(X_train.shape[1]):
        X_T=np.sort(X_train[:,i])
        H1=np.zeros(len(X_T)-1)
        T1=np.zeros(len(X_T)-1)
        for j in range(len(X_T)-1):
            TH=X_T[j]+(X_T[j+1]-X_T[j])/2
            [X_T1,y_T1,X_T2,y_T2]=subset(X_train,y_train,TH,i)
            #print(len(y_T1),len(y_T2))
            #if((len(y_T1)!=0)|(len(y_T2)!=0)):
            H1[j]=Entropy(y_T1)*len(y_T1)/(len(y_T1)+len(y_T2))+Entropy(y_T2)*len(y_T2)/(len(y_T1)+len(y_T2))
            T1[j]=TH
        H_max[i]=max(H-H1)
        T_max[i]=T1[np.argmax(H-H1)]
    Feat_index=np.argmax(H_max)
    Threshold=T_max[Feat_index]
    return Feat_index,Threshold
```

(b) Run the decision tree construction algorithm on the training examples. Compute the accuracy on validation examples and testing examples.

Solution:

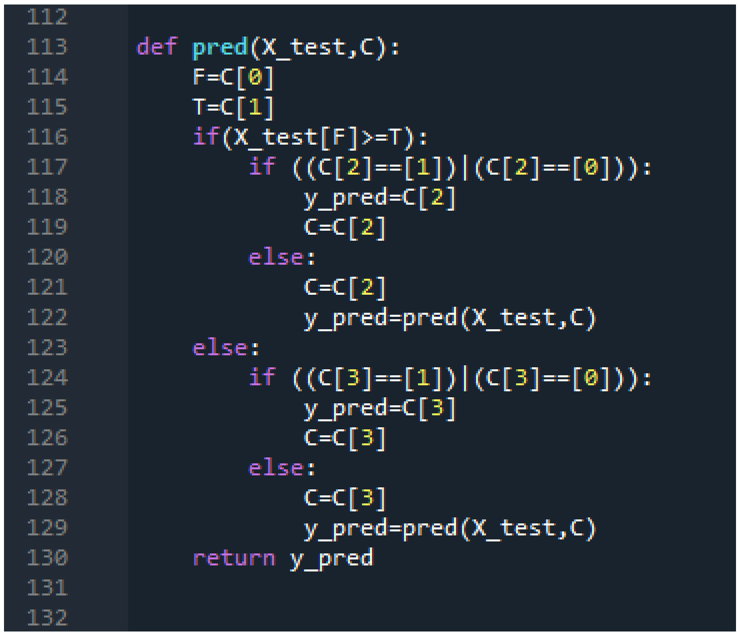
“Train-ID3” trains the decision tree by recursive approach, the format of the decision tree node is “Feat_Index, Threshold, Child_1, Child_2”. Child nodes could be impure or terminal nodes. Each Child also contains same information (“Feat_Index, Threshold, Child_1, Child_2”) if it’s not a leaf node. If Child node is a leaf node it contains “[0]” or “[1]”.



```
# Part b.


def Train_ID3(X_train,y_train):
    Child_1=[]
    Child_2=[]
    [Feat_index,Threshold]=Split(X_train,y_train)
    [X_T1,y_T1,X_T2,y_T2]=subset(X_train,y_train,Threshold,Feat_index)
    if((Entropy(y_T1)==0)&(len(y_T1)>0)):
        Child_1=[int(np.mean(y_T1))]
    elif(len(y_T1)==0):
        Child_1=[]
    if((Entropy(y_T2)==0)&(len(y_T2)>0)):
        Child_2=[int(np.mean(y_T2))]
    elif(len(y_T2)==0):
        Child_2=[]
    if((Entropy(y_T1))!=0):
        subtree1=Train_ID3(np.array(X_T1),y_T1)
        Child_1.extend(subtree1)
    if((Entropy(y_T2))!=0):
        subtree2=Train_ID3(np.array(X_T2),y_T2)
        Child_2.extend(subtree2)
    print('Training Decision Tree ...')
    return Feat_index,Threshold,Child_1,Child_2


Tree=Train_ID3(X_train[0:490],y_train[0:490])
```

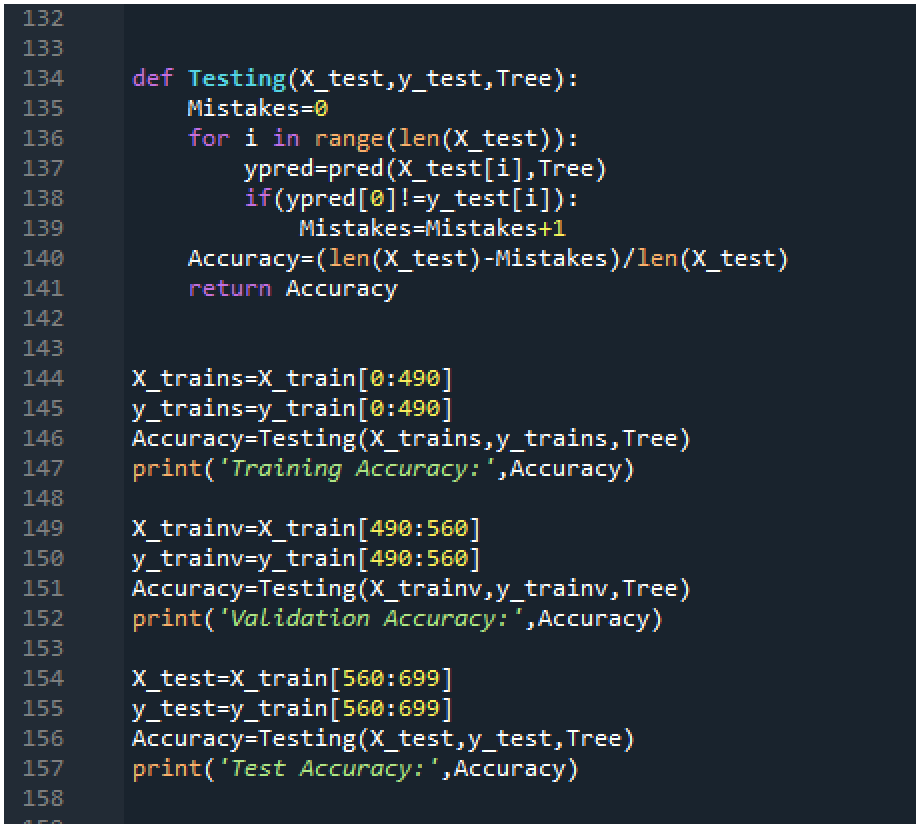
```
In [7]: Tree=Train_ID3(X_train[0:490],y_train[0:490])

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...

Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
Training Decision Tree ...
```

```
Tree
Out[8]:
(2,
 2.5,
 [5,
  2.5,
  [0,
   6.5,
   [2, 4.5, [0], [7, 7.0, [1, 5.0, [0], [1]], [0]]],
   [3,
    5.5,
    [0],
    [7,
     7.5,
     [0],
     [7,
      6.5,
     [1],
     [1],
      3.5,
      [5, 6.0, [6, 5.5, [0], [1, 6.5, [1], [2, 4.5, [0], [1]]]], [1]],
      [0, 5.5, [1], [0]]]]]]],
  [2,
   3.5,
   [0,
    6.5,
    [1, 4.5, [0], [2, 5.5, [0, 7.5, [1], [0]], [0]]],
    [4, 4.5, [5, 0.5, [0], [1]], [1]]],
   [1]]],
 [0, 5.5, [5, 1.5, [0], [1]], [5, 4.5, [0, 3.5, [0], [1]], [1]]])
```



```python

def pred(X_test,C):
    F=C[0]
    T=C[1]
    if(X_test[F]>=T):
        if ((C[2]==[1])|(C[2]==[0])):
            y_pred=C[2]
            C=C[2]
        else:
            C=C[2]
            y_pred=pred(X_test,C)
    else:
        if ((C[3]==[1])|(C[3]==[0])):
            y_pred=C[3]
            C=C[3]
        else:
            C=C[3]
            y_pred=pred(X_test,C)
    return y_pred


```

“Pred” function predicts the output “ypred” for a given value of feature vector.



```
def Testing(X_test,y_test,Tree):
    Mistakes=0
    for i in range(len(X_test)):
        ypred=pred(X_test[i],Tree)
        if(ypred[0]!=y_test[i]):
            Mistakes=Mistakes+1
    Accuracy=(len(X_test)-Mistakes)/len(X_test)
    return Accuracy


X_trains=X_train[0:490]
y_trains=y_train[0:490]
Accuracy=Testing(X_trains,y_trains,Tree)
print('Training Accuracy:',Accuracy)

X_trainv=X_train[490:560]
y_trainv=y_train[490:560]
Accuracy=Testing(X_trainv,y_trainv,Tree)
print('Validation Accuracy:',Accuracy)

X_test=X_train[560:699]
y_test=y_train[560:699]
Accuracy=Testing(X_test,y_test,Tree)
print('Test Accuracy:',Accuracy)
```

“Testing” function checks the testing accuracy using decision tree.

70% of Data is used for training, 10% for Validation and 20% for testing.

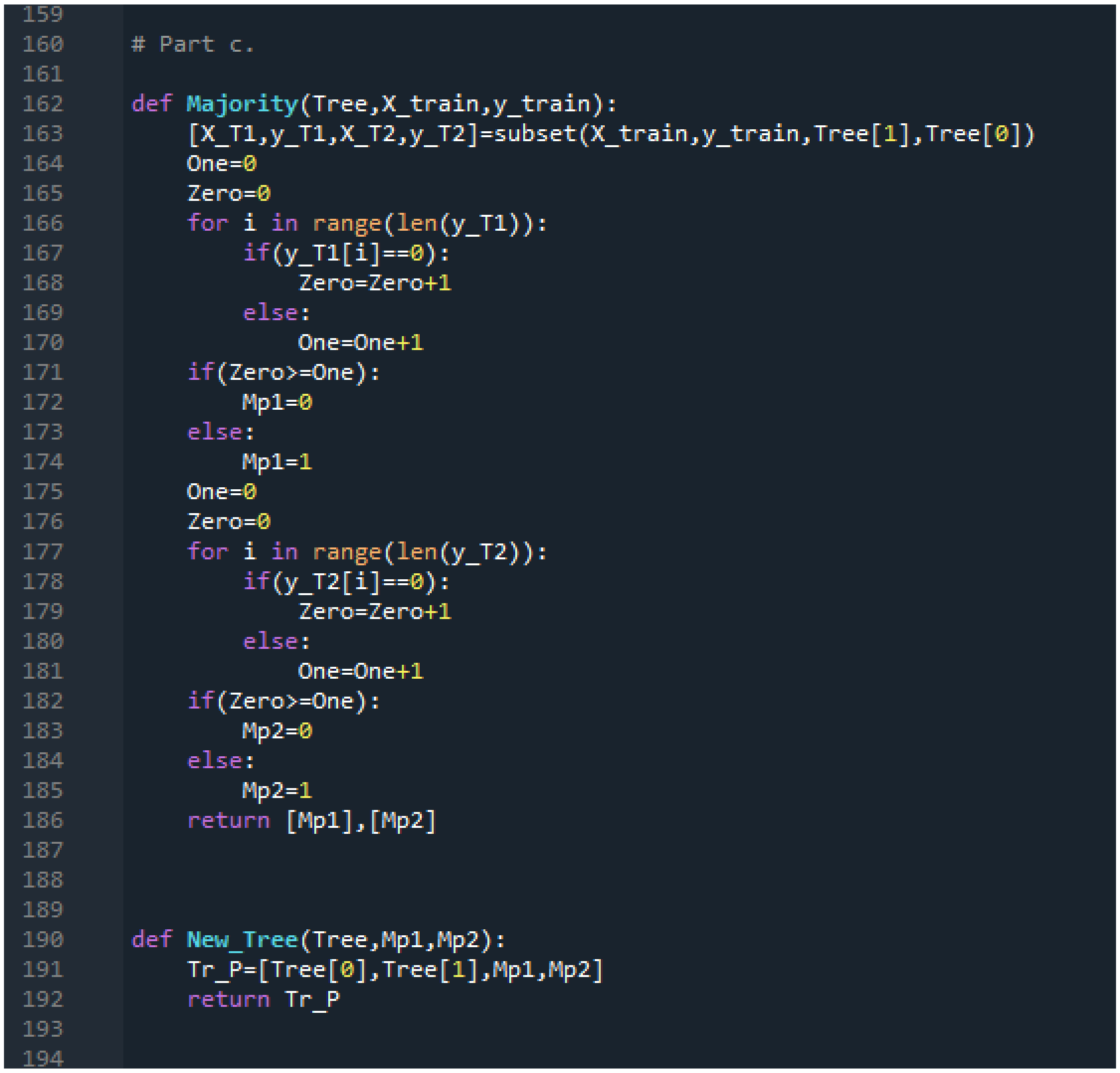
```
Training Accuracy: 1.0
Validation Accuracy: 0.9571428571428572
Test Accuracy: 0.9424460431654677
```

(c) Implement the decision tree pruning algorithm discussed in the class (via validation data).

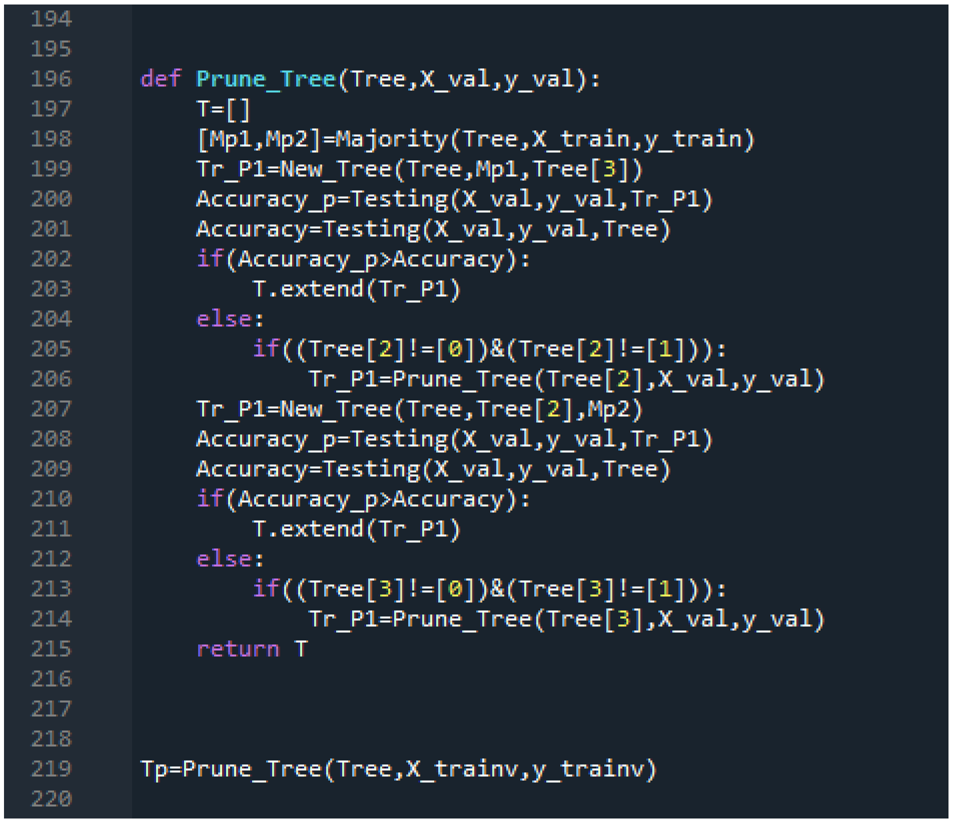
Solution:

In this part pruning algorithm is designed for decision tree.

- “Majority” fuction finds the majority classes of the children in a decision tree node
- “Prune_Tree” prunes the decision tree nodes if validation accuracy increases by placing a majority class at an impure node. Algorithm runs recursively until it has pruned all the nodes if validation accuracy improves



```python

# Part c.

def Majority(Tree,X_train,y_train):
    [X_T1,y_T1,X_T2,y_T2]=subset(X_train,y_train,Tree[1],Tree[0])
    One=0
    Zero=0
    for i in range(len(y_T1)):
        if(y_T1[i]==0):
            Zero=Zero+1
        else:
            One=One+1
    if(Zero>=One):
        Mp1=0
    else:
        Mp1=1
    One=0
    Zero=0
    for i in range(len(y_T2)):
        if(y_T2[i]==0):
            Zero=Zero+1
        else:
            One=One+1
    if(Zero>=One):
        Mp2=0
    else:
        Mp2=1
    return [Mp1],[Mp2]



def New_Tree(Tree,Mp1,Mp2):
    Tr_P=[Tree[0],Tree[1],Mp1,Mp2]
    return Tr_P


```



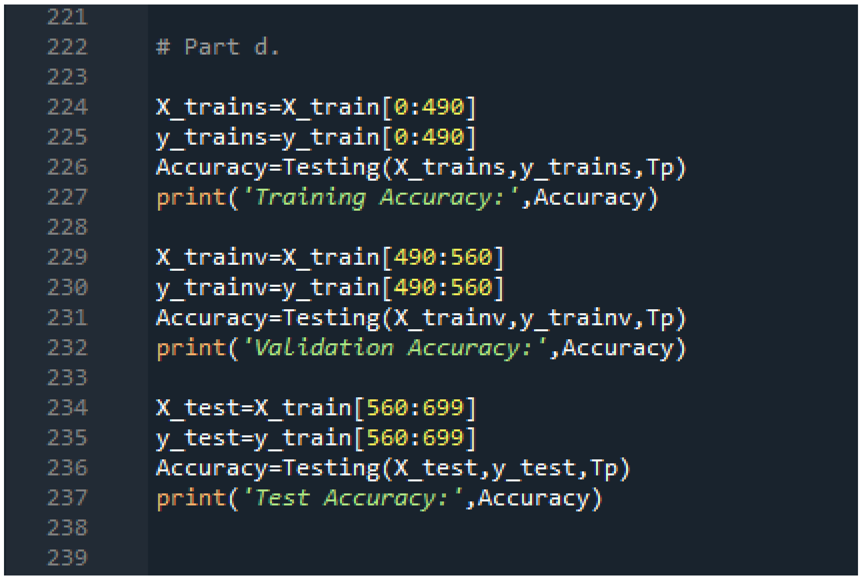
```python


def Prune_Tree(Tree,X_val,y_val):
    T=[]
    [Mp1,Mp2]=Majority(Tree,X_train,y_train)
    Tr_P1=New_Tree(Tree,Mp1,Tree[3])
    Accuracy_p=Testing(X_val,y_val,Tr_P1)
    Accuracy=Testing(X_val,y_val,Tree)
    if(Accuracy_p>Accuracy):
        T.extend(Tr_P1)
    else:
        if((Tree[2]!=[0])&(Tree[2]!=[1])):
            Tr_P1=Prune_Tree(Tree[2],X_val,y_val)
    Tr_P1=New_Tree(Tree,Tree[2],Mp2)
    Accuracy_p=Testing(X_val,y_val,Tr_P1)
    Accuracy=Testing(X_val,y_val,Tree)
    if(Accuracy_p>Accuracy):
        T.extend(Tr_P1)
    else:
        if((Tree[3]!=[0])&(Tree[3]!=[1])):
            Tr_P1=Prune_Tree(Tree[3],X_val,y_val)
    return T



Tp=Prune_Tree(Tree,X_trainv,y_trainv)

```

(d) Run the pruning algorithm on the decision tree constructed using training examples. Compute the accuracy on validation examples and testing examples. List your observations by comparing the performance of decision tree with and without pruning.

To debug and test your implementation, you can employ scikit-learn (http://scikit-learn.org/stable/modules/tree.html).

Solution:

• The validation accuracy improves by pruning but testing accuracy decreases slightly. We can try with a slightly larger validation set to improve testing accuracy.



```

# Part d.

X_trains=X_train[0:490]
y_trains=y_train[0:490]
Accuracy=Testing(X_trains,y_trains,Tp)
print('Training Accuracy:',Accuracy)

X_trainv=X_train[490:560]
y_trainv=y_train[490:560]
Accuracy=Testing(X_trainv,y_trainv,Tp)
print('Validation Accuracy:',Accuracy)

X_test=X_train[560:699]
y_test=y_train[560:699]
Accuracy=Testing(X_test,y_test,Tp)
print('Test Accuracy:',Accuracy)


```

Training Accuracy: 0.9836734693877551

Validation Accuracy: 0.9857142857142858

Test Accuracy: 0.935251798561151