CSCE633 Homework 01

Lu Sun

September 2020

Question 1

(i) Plot $\vec{x}_1, \vec{x}_2, \vec{x}_3$ in 2D space. Plot the line of $\omega(0)$ and the corresponding direction.

Answer:

$\omega(0) = [2, -1, 1]^T \longrightarrow 2 - x + y = 0$ The figure is shown as follow:

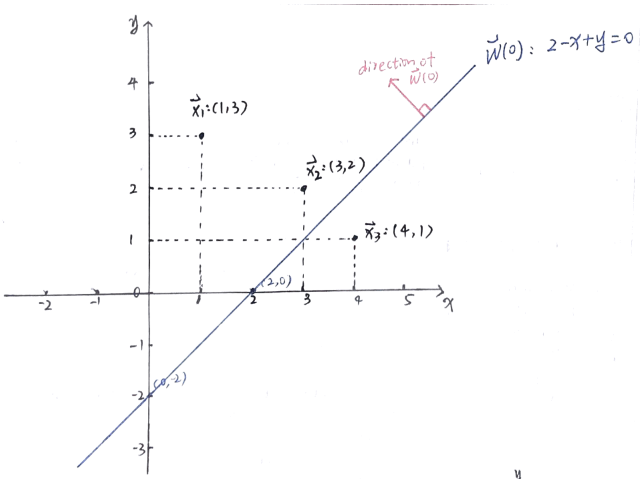

Figure 1: Q1.(i)

(ii) Indicate the class of $\vec{x}_1, \vec{x}_2, \vec{x}_3$ by $\omega(0)$. Point out the incorrectly classified samples.

Answer:

(a)The results are shown in the following table:

(b) From the above table, the incorrectly classified sample is $\vec{x}_2$.

samples	$sign\left(\omega(0)^T x_n\right)$	predict class	true class (y_n)
$\vec{x}_1 : (1, 3)$	$sign(2 - 1 + 3) = sign(4) = +1$	Class 1	Class 1 (+1)
$\vec{x}_2 : (3, 2)$	$sign(2 - 3 + 2) = sign(1) = +1$	Class 1	Class 2 (-1)
$\vec{x}_3 : (4, 1)$	$sign(2 - 4 + 1) = sign(-1) = -1$	Class 2	Class 2 (-1)

Table 1: Q1.(ii)

(iii) Find the value of the new weight $\omega(1)$ with LPA method and above misclassified sample. Find the new line corresponding to (1). Plot $\omega(1)$ and its direction. Indicate which samples are correctly classified and which are not.

Answer:

(a) In (ii), only sample $\vec{x}_2$ is misclassified. So, in process to get $\omega(1)$ with PLA method, only $\vec{x}_2 : (3, 2)$, $y_2 : -1$ and $\omega(0) : [2, -1, 1]^T$ are needed. Finally, $\omega(1)$ is got through the following formula:

$$\omega(1) = \omega(0) + y_2 \cdot [1, \vec{x}_2]^T = [2, -1, 1]^T - 1 \cdot [1, 3, 2]^T = [1, -4, -1]^T$$

(b) The new line corresponding to $\omega(1) : [1, -4, -1]^T$ is $1 - 4x - y = 0$

(c) The plot is shown as follow:

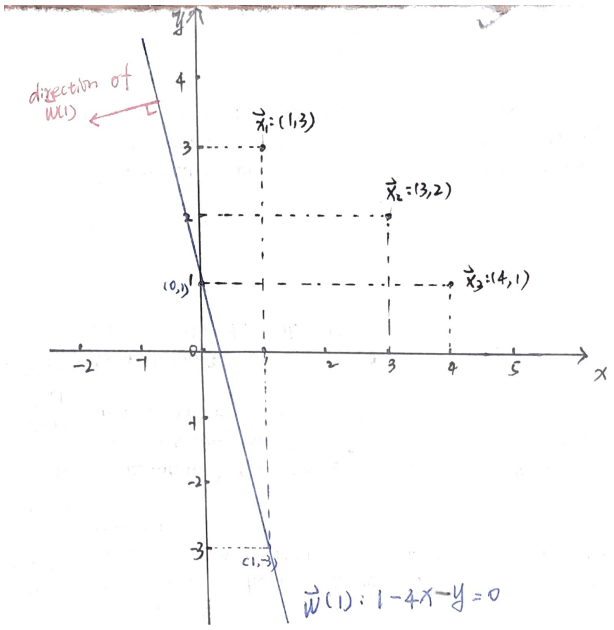

Figure 2: Q1.(iii)

(d) From the above plot, following results can be get:

samples	$sign\Big(\omega(1)^T x_n\Big)$	predict class	true class (y_n)	classified
$\vec{x}_1 : (1,3)$	$sign(1-4-3) = sign(-6) = -1$	Class 2	Class 1 (+1)	Incorrect
$\vec{x}_2 : (3,2)$	$sign(1-12-2) = sign(-13) = -1$	Class 2	Class 2 (-1)	Correct
$\vec{x}_3 : (4,1)$	$sign(1-16-1) = sign(-16) = -1$	Class 2	Class 2 (-1)	Correct

Table 2: Q1.(iii)

Namely, sample $\vec{x}_2, \vec{x}_3$ are correctly classified, while $\vec{x}_1$ is not.

(iv) Find the value of the new weight $\omega(2)$ with LPA method and above misclassified sample. Find the new line corresponding to (2). Plot $\omega(2)$ and its direction. Indicate which samples are correctly classified and which are not.

Answer:

(a) In (iii), only sample $\vec{x}_1$ is misclassified. So, in process to get $\omega(2)$ with PLA method, only $\vec{x}_1 : (1,3)$, $y_1 : +1$ and $\omega(1) : [1,-4,-1]^T$ are needed. Finally, $\omega(2)$ is got through the following formula:

$$\omega(2) = \omega(1) + y_1 \cdot [1, \vec{x}_1]^T = [1,-4,-1]^T + 1 \cdot [1,1,3]^T = [2,-3,2]^T$$

(b) The new line corresponding to $\omega(2) : [2,-3,2]^T$ is $2 - 3x + 2y = 0$

(c) The plot is shown as follow:

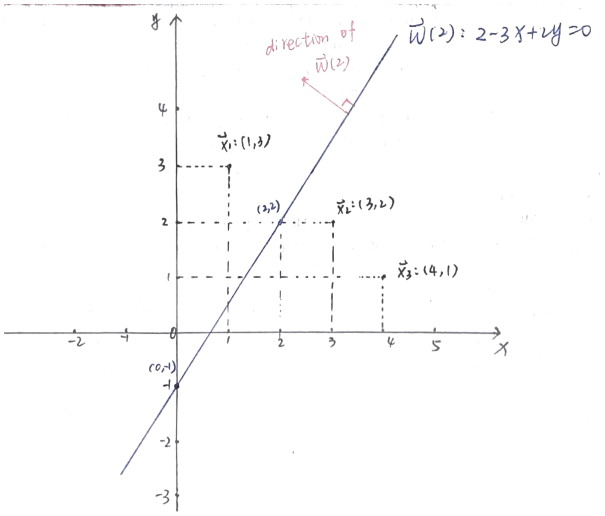

Figure 3: Q1.(iii)

(d) From the above plot, following results can be get:

samples	$sign\Big(\omega(1)^T x_n\Big)$	predict class	true class (y_n)	classified
$\vec{x}_1 : (1, 3)$	$sign(2 - 3 + 6) = sign(5) = +1$	Class 1	Class 1 (+1)	Correct
$\vec{x}_2 : (3, 2)$	$sign(2 - 9 + 4) = sign(-3) = -1$	Class 2	Class 2 (-1)	Correct
$\vec{x}_3 : (4, 1)$	$sign(2 - 12 + 2) = sign(-8) = -1$	Class 2	Class 2 (-1)	Correct

Table 3: Q1.(iv)

Namely, sample $\vec{x}_1, \vec{x}_2, \vec{x}_3$ are all correctly classified and none of the samples is incorrectly classified.

Question 2

(a.i)Compute the number of samples belong to each class. Are the classes equally distributed?

Answer:

The code for computing the number of samples is as follow: (in the code , 'Iris-setosa': 0, 'Iris-versicolor': 1,'Iris-virginica':2)

```
Lg = len(datasetTrain)
for i in range(3):
  print(len([1 for j in range(Lg) if datasetTrain[j].label==i]))
```

Finally, the following results are shown:

- The numbers of samples belong to 'Iris-setosa': 0, 'Iris-versicolor': 1,'Iris-virginica':2 are 30,30,30 respectively
- The classes equally distributed.

(a.ii) Plot the histogram of each feature. How are the features distributed?

Answer:

The code for plots is as follow:

```
import matplotlib.pyplot as plt
sl = [datasetTrain[i].sepal_length for i in range(Lg)]
sw = [datasetTrain[i].sepal_width  for i in range(Lg)]
pl = [datasetTrain[i].petal_length for i in range(Lg)]
pw = [datasetTrain[i].petal_width  for i in range(Lg)]
mapList = { 0:[sl,'sepal_length'], 1:[sw,'sepal_width'], ...
    2:[pl,'petal_length'],  3:[pw,'petal_width']}

plt.style.use('ggplot')
#plt.grid(True)
for i in range(4):
  plt.hist(mapList[i][0], bins=20, color = 'w',ec = ...
      'darkblue',lw=1.5)
  plt.xlabel(mapList[i][1])
  plt.ylabel('Number of train data')
  plt.show()
```

The following plots can be shown:
From the below figures, we can know that

- Features sepal length and sepal width are unimodal, uniform distributions
- Features petal length and petal width are bimodal distributions

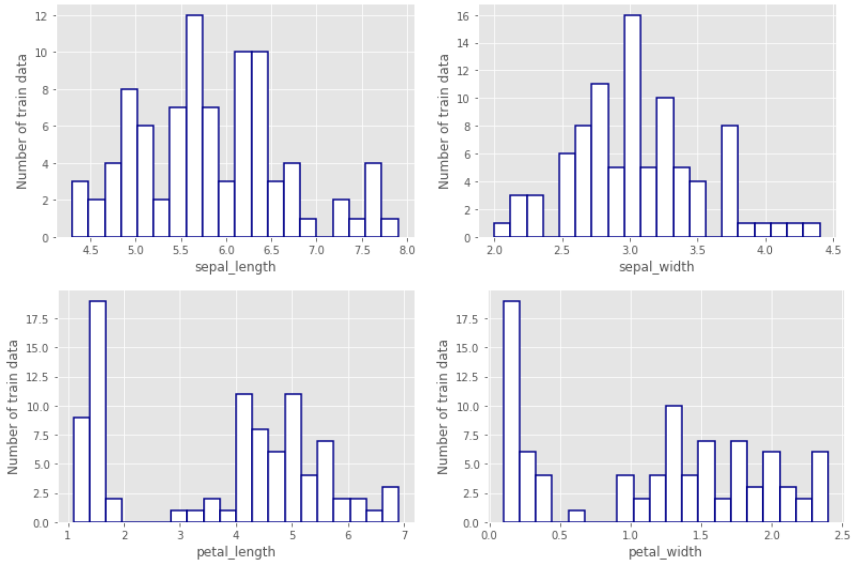

Figure 4: The histogram of each feature

(a.iii) Plot scatter plots. Indicate the class on the plots. What do you observe? How separable do the classes look? Are there feature combinations for which the three classes are more separable?

Answer:

The code for plots is as follow:

```
def plot_data(xdata,ydata,xlabl,ylabl):
  #'Iris-setosa': 0, 'Iris-versicolor': 1,'Iris-virginica':2
  SeVeVi_x = []
  SeVeVi_y = []
  for j in range(3):
      SeVeVi_x.append([xdata[i] for i in range(Lg) if ...
          datasetTrain[i].label == j])  # List: heights of cats
      SeVeVi_y.append([ydata[i] for i in range(Lg) if ...
          datasetTrain[i].label == j])  # List: weights of cats

  plt.scatter(SeVeVi_x[0], SeVeVi_y[0], c='b', marker='x', ...
      label='Iris-setosa')  # put points corresponding to cats, ...
      b: blue colored, x: marked with 'x'
  plt.scatter(SeVeVi_x[1], SeVeVi_y[1], c='r', marker='+', ...
      label='Iris-versicolor')  # put points corresponding to ...
      cats, b: blue colored, x: marked with 'x'
  plt.scatter(SeVeVi_x[2], SeVeVi_y[2], c='g', marker='1', ...
      label='Iris-virginica')  # put points corresponding to ...
      cats, b: blue colored, x: marked with 'x'
```

```
12
13    plt.xlabel(xlabl)
14    plt.ylabel(ylabl)
15    plt.legend()
16    plt.show()
17
18  #sl : "sepal_length", sw : "sepal_width", pl : "petal_length", ...
        pw : "petal_width"
19  #plot_data(sl,sw,'sepal_length','sepal_width')
20
21  import itertools
22  for i, j in itertools.product(range(4),range(4)):
23    if i<j:
24      plot_data(mapList[i][0],mapList[j][0],mapList[i][1],mapList[j][1])
```

The following plots can be shown:

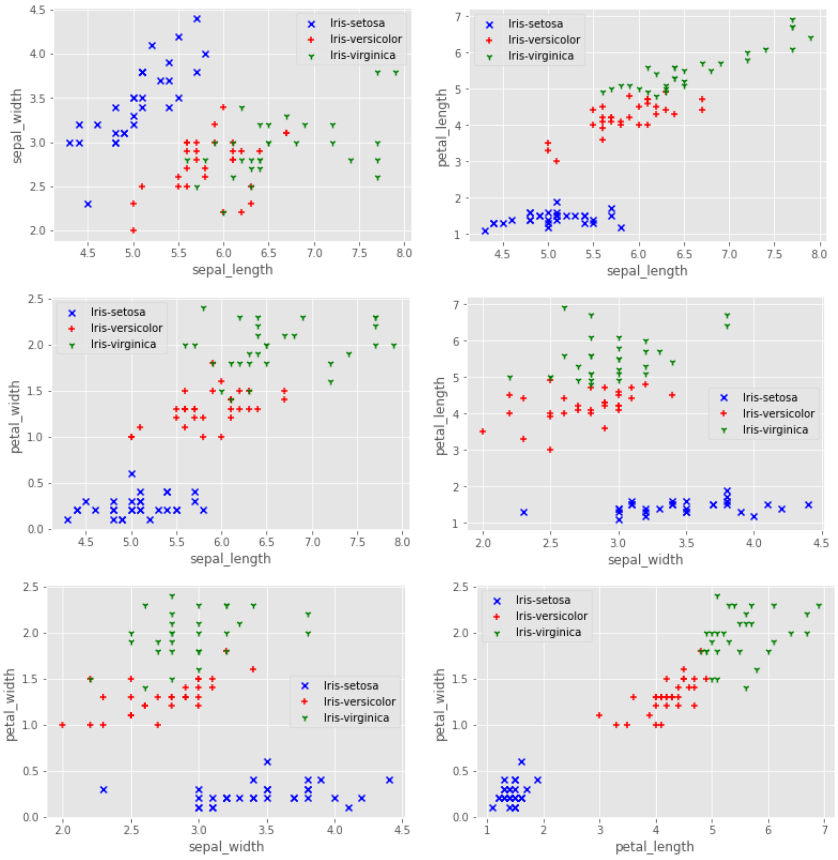

Figure 5: The scatter of combined two features

From the above figures, we can know that:

- Some combination of features can separate the three classes, while some can not.
- Combination of (sepal length, sepal width) can't separate class versicolor and virginica. While all the other five combinations look good in separating the three classes.
- Combination (petal length, petal width) is more separable for the three classes both in x-axis and y-axis.

(b.i) Implement a KNN using l2-norm

Answer:

The pseudocode is shown as follow:

```
Knn( test_data, train_data,k):
    distance = ||test_data - train_data||
    indx_tep = k smallest points' order in distance list
    countx = train_data_label[indx_tep]
    number of label 0 = count(countx==0)
    number of label 1 = count(countx==1)
    number of label 2 = count(countx==2)
    return the largest number's label
```

Here, we don't care about the order of k-th points. Namely, for example, [9,9,1,2,3]'s 1-th largest point is 9, at order 1. Although order 2 is also 9, the largest number, we don't care about that order. Then the code for KNN method is shwon as follow:

Listing 1: KNN

```
###############Define Norm Function####################
def L0Fun(xtest,datatrain):
  x_y = datatrain-np.array(xtest)
  #return sum([1 for i in range(len(x_y)) if x_y[i]!=0])
  return np.count_nonzero(x_y,axis = 1)
def L1Fun(xtest,datatrain):
  x_y = datatrain-np.array(xtest)
  #return sum(abs(x_y))
  return np.sum(abs(x_y),axis=1)
def L2Fun(xtest,datatrain):
  x_y = datatrain-np.array(xtest)
  #return np.sqrt(x_y.dot(x_y))
  return np.sqrt(np.sum(x_y * x_y, axis=1))
def CsFun(xtest,datatrain):
  x = np.array(xtest)
  #y = np.array(y)
  #return x.dot(y)/(np.sqrt(x.dot(x))*np.sqrt(y.dot(y)))
  xnorm = np.sqrt(x.dot(x))
  ynorm = np.sqrt(np.sum(datatrain * datatrain, axis=1))
  return np.sum(datatrain * x,axis = 1)/ (xnorm * ynorm)
mapNorm = {0:L0Fun,1:L1Fun,2:L2Fun,3:CsFun}
```

```
22
23
24 ################Define KNN Function####################
25 def KnnFun(xtest, intK, xtrain = datasetTrain, normch = 2):
26   datatrain = np.reshape([xtrain[i].full_vector() for i in
27   range(len(xtrain))],[-1,5])
28
29   #select norm
30   normFun = mapNorm[normch]
31
32   #compute all the distance
33   dist = normFun(xtest,datatrain[0::,0:4])          #total dist list
34
35   #select top k-th nearest point regardless the order
36   indx_tep = np.argpartition(-dist,-intK)[-intK:]   #index
37   countx = [int(datatrain[i,-1]) for i in indx_tep] #K-th lable list
38
39   esti_tep = np.array([countx.count(i) for i in range(3)])
40   esti = np.argmax(esti_tep)
41
42   return esti
```

(b.ii) Set $K = 1, 3, 5, 7, ..., 19$, train the model and compute the classification accuracy. Plot the ACC with K. Report the best K^*

Answer:

First, the following code for ACC is implemented.

```
1 def AccFun(estiLabel,trueLabel):
2     x_y = np.array(estiLabel)-np.array(trueLabel)
3     Lg_xy = len(x_y)
4     #return sum([1 for i in range(Lg_xy) if x_y[i]==0]) / Lg_xy
5     return np.count_nonzero(x_y==0,axis=0) / Lg_xy, x_y
```

Then, the code for training the model and compute ACC on development data is shown as follow:

```
1 datasetDev = parse_dataset('/content/drive/My ...
      Drive/2020_FALL/CSCE633/HomeWork/HW01/iris_dev.csv')
2
3 intK = np.arange(1,21,2)
4 LgK = len(intK)
5 LgDev = len(datasetDev)
6 estiLabel = np.zeros([LgDev,LgK],dtype = int)
7 trueLabel = np.zeros([LgDev,LgK],dtype = int)
8 for i,k in itertools.product(range(LgDev),range(LgK)):
9   xdev = datasetDev[i].feature_vector()
10  lab = KnnFun(xdev,intK[k])
11  estiLabel[i][k] = lab
12  trueLabel[i][k] = datasetDev[i].label
13 print(accRes)
```

The results of ACC is shown in the following table:

K	1	3	5	7	9
ACC	0.96666667	0.93333333	0.93333333	0.93333333	0.93333333
K	11	13	15	17	19
ACC	0.93333333	0.93333333	0.93333333	0.93333333	0.93333333

Table 4: ACC

The code for plot is shown as follow:

```
plt.plot(intK,accRes,'-+')
plt.xlabel('choice of K')
plt.ylabel('Accuracy Rate')
```

The plot is shown as follow:

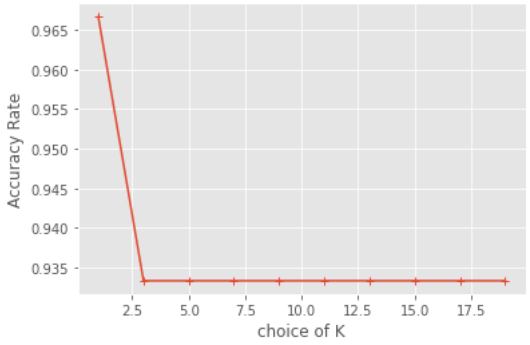

Figure 6: ACC

- From the above table and plot, the best $K^* = 1$ whose ACC is the highest 0.96666667

(b.iii) Report the ACC on test set with K^*. What do you observe?

Answer:

The code is as follow:

```
datasetTest = parse_dataset('/content/drive/My ...
    Drive/2020_FALL/CSCE633/HomeWork/HW01/iris_test.csv')  # ...
    Returns the list of datapoints

Kstar = 1
LgTest = len(datasetTest)
estiLabel = np.zeros([LgTest,1],dtype = int)
```

```
trueLabel = np.zeros([LgTest,1],dtype = int)
for i in range(LgTest):
  xdev = datasetTest[i].feature_vector()
  lab= KnnFun(xdev, Kstar)
  estiLabel[i] = lab
  trueLabel[i] = datasetTest[i].label
print(accRes)
```

Finally, we get:

- Acc = 0.96666667
- From the observation, the ACC for development set is equal to that of test set with $K^* = 1$.

(b.iv) Experiment with different types of distances. Report your findings

The plots are shown as follow:

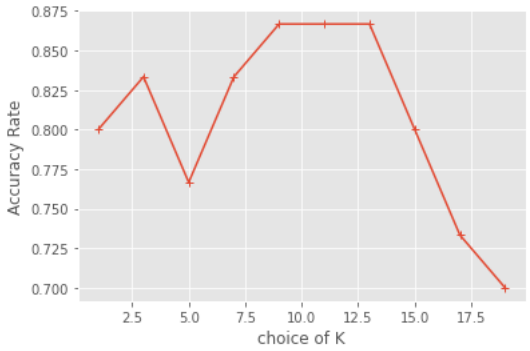

Figure 7: l0-norm

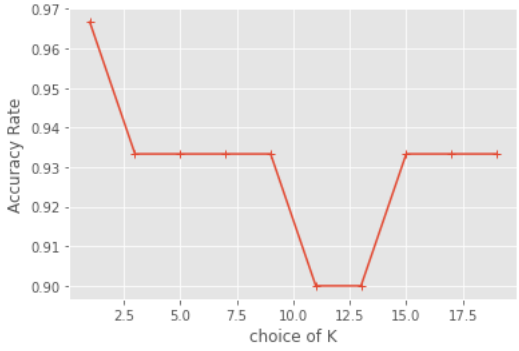

Figure 8: l1-norm

From the above plots, we can know:

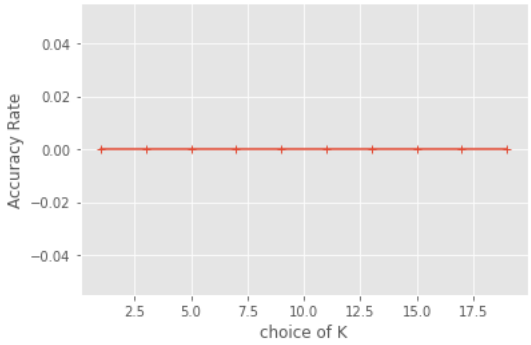

Figure 9: cosine similarity

- l0-norm's best K^* is 9,11,13 which is larger than that of l2-norm, while the accuracy is smaller.
- l1-norm has the same best K^* and ACC as that of l2-norm.
- Cosine similarity can't get good results in this exmaple, whose ACC is always 0.

Appendix

(i) Code for upload data

```
import numpy as np

class DataPoint(object):

  def __init__(self, feats):
    """
    Return a Datapoint object whose attributes are given by "feats"
    """
    self.sepal_length= feats['sepal_length']
    self.sepal_width = feats['sepal_width']
    self.petal_length= feats['petal_length']
    self.petal_width = feats['petal_width']
    self.label = feats['label']

  def feature_vector(self):
    """
    Return feature vector as a numpy array
    """
    return np.array([self.sepal_length, self.sepal_width, ...
        self.petal_length, self.petal_width])

  def feature_vector_with_bias(self):
    """
    Return features with 1 appended at the end
    We 'absorb' bias (b) into the feature vector w by adding one ...
        additional constant dimension
    y = wx + b becomes y = w'x' where x' = [x; 1] and w' = [w; b]
    """
    return np.array([self.sepal_width, self.sepal_width, ...
        self.petal_length, self.petal_width, 1.])

  def full_vector(self):
    """
    Return feature vector as a numpy array
    """
    return np.array([self.sepal_length, self.sepal_width, ...
        self.petal_length, self.petal_width,self.label])

  def __str__(self):
    """
    print(object) uses this method
    Define it in the way you want to see the object (e.g. ...
        Height:27.0, Weight:50.0, Label:1)
    """
    return "sepal_length:{}, sepal_width:{}, petal_length:{}, ...
        petal_width:{}, Label:{}".format(self.sepal_length, ...
        self.sepal_width, self.petal_length, self.petal_width, ...
        self.label)

def parse_dataset(filename):
  data_file = open(filename, 'r')  # Open File "to read"
  dataset = []  # List to hold Datapoint objects
```

```
45    label_map = {'Iris-setosa': 0, 'Iris-versicolor': ...
          1,'Iris-virginica':2}  # define labels as 0/1/2.
46
47    for index, line in enumerate(data_file):
48      if index == 0:  # First line: "Height   Weight  Label" ...
            describes the datapoint, it's not an actual datapoint,
49        continue  # do nothing, it will skip all the following lines
50      #print(line)
51      sepal_length, sepal_width, petal_length,petal_width,label = ...
            line.strip().split(',')#line.strip().split('\t')  # ...
            strip() removes '\n', and split('\t') splits the line at ...
            tabs
52      dataset.append(DataPoint({'sepal_length':float(sepal_length), ...
            'sepal_width':float(sepal_width),
53                                'petal_length':float(petal_length), ...
                                    'petal_width':float(petal_width),
54                                'label':label_map[label]}))  # ...
                                    Create DataPoint object for ...
                                    the given data
55
56    print("Total Number of Data Points: {0}".format(len(dataset)))
57    print("Examples: 1 - {0} , 2 - {1}".format(dataset[0], ...
          dataset[-1]))
58    return dataset
59
60  datasetTrain = parse_dataset('/content/drive/My ...
        Drive/2020_FALL/CSCE633/HomeWork/HW01/iris_train.csv')  # ...
        Returns the list of datapoints
```

(ii) Code for bonus

```
1   ##################l0 norm####################
2   intK = np.arange(1,21,2)
3   LgK = len(intK)
4   LgDev = len(datasetDev)
5   estiLabel = np.zeros([LgDev,LgK],dtype = int)
6   trueLabel = np.zeros([LgDev,LgK],dtype = int)
7   for i,k in itertools.product(range(LgDev),range(LgK)):
8     xdev = datasetDev[i].feature_vector()
9     lab = KnnFun(xdev, intK[k],normch = 0)
10    estiLabel[i][k] = lab
11    trueLabel[i][k] = datasetDev[i].label
12  #estiLabel = np.reshape(estiLabel,[-1,1])
13  #trueLabel = np.reshape(trueLabel,[-1,1])
14  #print(estiLabel,trueLabel)
15  accRes,_ =AccFun(estiLabel,trueLabel)
16  print(accRes)
17
18  plt.plot(intK,accRes,'-+')
19  plt.xlabel('choice of K')
20  plt.ylabel('Accuracy Rate')
21  plt.show()
22  ##################l1 norm####################
23  intK = np.arange(1,21,2)
24  LgK = len(intK)
25  LgDev = len(datasetDev)
```

```
estiLabel = np.zeros([LgDev,LgK],dtype = int)
trueLabel = np.zeros([LgDev,LgK],dtype = int)
for i,k in itertools.product(range(LgDev),range(LgK)):
  xdev = datasetDev[i].feature_vector()
  lab = KnnFun(xdev, intK[k],normch=1)
  estiLabel[i][k] = lab
  trueLabel[i][k] = datasetDev[i].label
#estiLabel = np.reshape(estiLabel,[-1,1])
#trueLabel = np.reshape(trueLabel,[-1,1])
#print(estiLabel,trueLabel)
accRes,_ =AccFun(estiLabel,trueLabel)
print(accRes)

plt.plot(intK,accRes,'-+')
plt.xlabel('choice of K')
plt.ylabel('Accuracy Rate')
plt.show()
##################cosine similarity###########
intK = np.arange(1,21,2)
LgK = len(intK)
LgDev = len(datasetDev)
estiLabel = np.zeros([LgDev,LgK],dtype = int)
trueLabel = np.zeros([LgDev,LgK],dtype = int)
for i,k in itertools.product(range(LgDev),range(LgK)):
  xdev = datasetDev[i].feature_vector()
  lab = KnnFun(xdev, intK[k],normch=3)
  estiLabel[i][k] = lab
  trueLabel[i][k] = datasetDev[i].label
#estiLabel = np.reshape(estiLabel,[-1,1])
#trueLabel = np.reshape(trueLabel,[-1,1])
#print(estiLabel,trueLabel)
accRes,_ =AccFun(estiLabel,trueLabel)
print(accRes)

plt.plot(intK,accRes,'-+')
plt.xlabel('choice of K')
plt.ylabel('Accuracy Rate')
plt.show()
```

(iii) Code for KNN with order considered

```
def KnnFun(xtest, intK, xtrain = datasetTrain, normch = 2):
  datatrain = np.reshape([xtrain[i].full_vector() for i in ...
      range(len(xtrain))],[-1,5])
  normFun = mapNorm[normch]

  #compute all the distance
  dist = normFun(xtest,datatrain[0::,0:4])                    ...
      #total dist list

  #select top k-th nearest point regardless equal nearst points' ...
      order
  indx_tep = np.argpartition(-dist,-intK)[-intK:]             #index
  #indx = dist[indx_tep]                                      ...
      #intK-th dist list
  #print(dist,'\n',indx)                                      ...
```

```
        #check top intK-th
12  countx = [int(datatrain[i,-1]) for i in indx_tep]          ...
        #intK-th lable list
13  #print(countx)
14
15  esti_tep = np.array([countx.count(i) for i in range(3)])
16  esti = np.argmax(esti_tep)
17  #print(esti_tep,esti)                                       ...
        #check lable selection
18
19  """
20  #select top k-th nearest point and record possible equal ...
        nearest top k-th points
21  indx = []
22  indist = []
23  while (len(indx) <intK):
24    indx_tep = np.where(dist==min(dist))[0]
25    #if len(indx_tep) == 1:
26    #  indx.append(datatrain[indx_tep][0])
27    #else:
28    for i in range(len(indx_tep)):
29      indx.append(datatrain[indx_tep[i]])
30      indist.append(dist[indx_tep[i]])
31    datatrain = np.delete(datatrain,indx_tep,axis = 0)
32    dist = np.delete(dist,indx_tep,axis = 0)
33
34  if len(indx) > intK:
35    print("# nearest points is {0} larger than ...
          {1}".format(len(indx),intK))
36
37  countx = [indx[i][4] for i in range(len(indx))]
38  esti_tep = np.array([countx.count(i) for i in range(3)])
39  esti = np.where(esti_tep == max(esti_tep))[0]
40  if len(esti) > 1:
41    indistnew = np.where(indist!=indist[-1])[0]
42    indx = [indx[i] for i in indistnew]
43    countx = [indx[i][4] for i in range(len(indx))]
44    esti_tep = np.array([countx.count(i) for i in range(3)])
45    esti = np.where(esti_tep == max(esti_tep))[0]
46    print("Eorro: More than one opt estimator")
47  return esti ,indx, countx
48  """
49  return esti
```