**CENG 420 | ELEC 569A**

**Question 1.**

As we know, the K-Nearest Neighbours approach works for binary classification problems by default. A binary learning i.e. a two-class approach for classification problems considers both positive and negative classes. Examples are provided for the two-classes to a learning algorithm, in order to build a classifier that will be used to discriminate between those classes. This comes under supervised learning where the labels of the two-classes are known in advance.

The K-NN approach for binary classification can be extended for one-class classification problems by making the algorithm learn from positive target class or positive examples only.

A classifier should be developed which is able to recognise the examples belonging to its target and rejecting others as outliers.

**Pseudo Code for K-NN one-class classification**

Let’s take target Data and test sample X as input parameters.

Required output is the class label of test sample X.

Classifying an input test sample X, whether it belongs to a target class or not following steps can be performed:

1. A threshold value is decided in advance. Label it as TH. Let’s take the value of TH as ‘1.0’.

1. K neighbors are selected for classification.

2. Now, find the nearest neighbor for sample X in the target class and label it as B.

3. Compute the distance between X and Y and label it as DXY.

4. Now, compute the k number of nearest neighbors for Y in the target class.

5. Calculate the average distance of all the k-nearest neighbors. Label it as DAVG.

6. If DXY / DAVG > TH,

6.1. Reject X as a target class, else

6.2. Accept X as a target class

One and two class methods can both give useful classification accuracies when the negative class is well characterized. The advantage of one class methods is that it eliminates guessing at the optimal features for the negative class when they are not well defined. In these cases, one-class methods can be superior to two-class methods when the features which are chosen as representative of that positive class are well defined.

**K-NN Implementation for iris dataset**

def getAccuracy(testSet, predictions):

correct = 0

for x in range(len(testSet)):

if testSet[x][-1] == predictions[x]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

# prepare data

trainingSet=[]

testSet=[]

split = 0.67

loadDataset('irisdata.csv', split, trainingSet, testSet)

print('Train set: ' + repr(len(trainingSet)))

print('Test set: ' + repr(len(testSet)))

# generate predictions

predictions=[]

k = 3

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x], k)

result = getResponse(neighbors)

predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) + '%')

main()

Output:

Train set: 106

Test set: 43

…

…

> predicted='Iris-versicolor', actual='Iris-versicolor'

> predicted='Iris-versicolor', actual='Iris-versicolor'

> predicted='Iris-versicolor', actual='Iris-versicolor'

> predicted='Iris-versicolor', actual='Iris-versicolor'

> predicted='Iris-versicolor', actual='Iris-versicolor'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-versicolor', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

> predicted='Iris-virginica', actual='Iris-virginica'

Accuracy: 95.34883720930233%

**Question 2.**

1. King Arthur
2. American Pie
3. Daredevil
4. Batman Vs Superman
5. Cinderella
6. Enchanted

Target minimum support: 60%: 60x4/100: 2.4~ 2

Target minimum confidence: 80%: 80x4/100:3.2~3

|  |  |
| --- | --- |
| **TID** | **Items** |
| 01 | A, B, C, D |
| 02 | E, B, D, F |
| 03 | C, B, E, F, D |
| 04 | D, B, C |

Step 1: Scan D for count of each candidate. The candidate list is {A, B, C, D, E, F}

C1=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| A | 01 |
| B | 04 |
| C | 03 |
| D | 04 |
| E | 02 |
| F | 02 |

Step 2: Compare candidate support count with minimum support count (i.e.2)

L1=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| B | 04 |
| C | 03 |
| D | 04 |
| E | 02 |
| F | 02 |

Step 3: Generate candidate C2 from L1 and find the support.

C2=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| B, C | 03 |
| B, D | 04 |
| B, E | 02 |
| B, F | 02 |
| C, D | 03 |
| C, E | 01 |
| C, F | 01 |
| D, E | 02 |
| D, F | 02 |
| E, F | 02 |

Step 4: Compare candidate (C2) support count with the minimum support count

L2=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| B, C | 03 |
| B, D | 04 |
| B, E | 02 |
| B, F | 02 |
| C, D | 03 |
| D, E | 02 |
| D, F | 02 |
| E, F | 02 |

Step 5: Generate candidate C3 from L2 and find the support.

C3=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| B, C, D | 03 |
| B, C, E | 01 |
| B, C, F | 01 |
| C, D, E | 01 |
| C, D, F | 01 |
| D, E, B | 02 |
| D, E, F | 02 |
| E, F, C | 01 |

Step 6: Compare candidate (C3) support count with the minimum support count

L3=

|  |  |
| --- | --- |
| **Item set** | **Support Count** |
| B, C, D | 03 |
| D, E, B | 02 |
| D, E, F | 02 |

Step 7: So the data contains the frequent item sets: {B, C, D}

Generate the Association rule from frequent item sets with the support and confidence.

F = {{B}, {C}, {D}, {B, C}, {B, D}, {C, D}}

|  |  |  |  |
| --- | --- | --- | --- |
| **Association Rule** | **Support** | **Confidence** | **Confidence%** |
| BC🡪D | 03 | 3/3 | 100% |
| BD🡪C | 03 | ¾ | 75% |
| CD🡪B | 03 | 3/3 | 100% |
| B🡪CD | 03 | ¾ | 75% |
| C🡪BD | 03 | 3/3 | 100% |
| D🡪BC | 03 | ¾ | 75% |

Given minimum confidence threshold is 80%. So only BC => D, CD => B, C => BD rules are output.

Final rules are:

Rule 1: BC => D: American Pie, Daredevil => Batman Vs Superman

Rule 2: CD => B: Daredevil, Batman Vs Superman => American Pie

Rule 3: C => BD: Daredevil => American Pie, Batman Vs Superman

**Question 3. Unsupervised Learning Problem**

Creating a YouTube video and posting it online is a practice that has been taken up by literally millions of people worldwide.



However, while some videos enjoy up to millions of views, others languish in lonely areas of cyberspace.

What makes some videos more popular than others?

It’s time to find out why people hit that ‘like’ button!

Using a web-based crawler, data has been collected from YouTube over a 14-month period from 2010 to 2011, to try to answer this question. The data set, youtube.csv, is attached as well. The data set consists of multiple session ids over which data was recorded, however you should only analyse the data corresponding to your randomly selected id from a provided list of valid ids (see below). For further information about the variables, see the Data Sets page at the end of this document.

The valid session IDs are: 20, 25, 40, 41, 44

1. Principal Component Analysis: For the randomly selected ID, conduct a principal component analysis on the data, excluding the first column, which contains the session id number. In your analysis include:
2. Correlation output. Given the size of the data set, this time, printing the correlation matrix will not tell us anything useful, so think about how you would like to present the correlation information (e.g. as an image, the top X most highly correlated … the choice is yours and the possibilities are up to you).
3. Conduct a Principal Component Analysis using prcomp(). Should you use the option scale in prcomp(), to standardise your data? Explain your decision (1-2 sentences). Hint: with large data sets, you can determine if you need to scale by randomly sampling the data set and then e.g. producing boxplots of this random sample.

Try to explain the first four principal components (if they are not sensible, note this).

1. Produce a plot of the first two principal components and discuss any easily visible trends. If you find the output difficult to read in parts, this also tells us something about the popularity of videos on YouTube! If there are no discernible trends, note this and refer the reader to your discussion in part c.

**Data Set**

youtube.csv: The You Tube data can be summarised as follows

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Additional Information** |
| Session ID | Unique session id number | **R** |
| Keywords | Number of keywords associated with this video | log transformed |
| Video view count | Number of views of the video | log transformed |
| Video favourite count | Number of times this video was listed as a favourite | log transformed |
| Video rating count | Number of ratings | log transformed |
| Video rating average | Average rating of video | - |
| Video comments count | Number of comments made about the video | log transformed |
| Like count | Number of likes attracted by video | log transformed |
| Dislike count | Number of dislikes attracted by video | log transformed |
| Uploader follower count | Number of followers of uploader | log transformed |
| Uploader video count | Number of videos uploaded by the uploader | log transformed |
| Uploader friends | Number of friends of uploader | log transformed |
| Uploader views count | Number of views of any video of uploader | log transformed |
| Uploader Age | The age of the uploader | Measured in years |
| Max Quality of video | The best available video quality |  |
| Age of video | The age of the video | log transformed |