Name: Vipin Singh
Email: vipsingh@iu.edu

Applied Data Science

Natural Language Processing, Naïve Bayes, Support Vector Machines and Neural Networks

Assignment 9

1. Part 1

Naïve Bayes

- Build a Naive Bayes classier for the dataset given below and find the probability of it raining given that the temperate is High, it is windy and the Humidity is high.

Temperature	Windy	Humidity	Rain
High	Yes	Low	No
High	Yes	Moderate	Yes
Low	Yes	High	Yes
Low	Yes	Low	No
High	No	Low	No
High	No	Moderate	Yes
Low	No	Low	No
Low	No	Moderate	No

To answer this question, we need to build the following table of attribute, value and probability. The first column is ATTRIBUTE, which Stores the distinct value of each attribute, for example for Humidity all the three distinct values – Low, Moderate and High are shown below. Similarly, the ATTRIBUTE column is built for all the attributes except the class attribute.

Name: Vipin Singh
Email: vipsingh@iu.edu

The Column VALUE, has two sub columns as the distinct values of class attribute i.e. rain. The Possible value of Rain are Yes and No. The VALUE column stores the count of the attribute values satisfying the class value for example when Humidity is Low there are 4 instances of Rain as No. The Probability column stores the probability for class value Yes and No for example when Humidity is Low, the probability of having no rain is 4/5. The number 4/5 is calculated by checking the total instances of No rain for all the values of Humidity which are 5 in this case.

ATTRIBUTE		VALUE		PROBABILITY	
		Yes	No	Yes	No
Humidity	Low	0	4	0	4/5
	Moderate	2	1	2/3	1/5
	High	1	0	1/3	0
Windy	Yes	2	2	2/3	2/5
	No	1	3	1/3	3/5
Temperature	High	2	2	2/3	2/5
	Low	1	3	1/3	3/5

The below table shows the probability of class variable Rain. There are total 8 instances of Rain as yes and No. The 3 are yes and 5 are No. This table shows the probability as 3.8 and 5/8 for yes and No respectively.

Class Variable	Probability	
Rain	Yes	No
	3/8	5/8

We need to predict the class of the following new record using naïve bayes classification. Consider the new record as V tuple.
V = Temperature = High, Windy = Yes, Humidity = High, Rain =?

Let's assume Rain = No, and calculate the probability of Rain = No using the naïve bayes classification.

P(No/V) = P(V/No) * P(Rain/No)
= Probability of High Temp for No Rain * Probability of Windy Yes for No Rain
*Probability of High Humidity when No Rain * Probability of No Rain
= 2/5*2/5*0*5/8 (Check columns shown in Yellow above)
= 0

Name: Vipin Singh
Email: vipsingh@iu.edu

This is one of the drawback of naïve Bayes, that if one probability of the event is 0, the whole probability becomes 0. We can apply some smoothing in this type of scenarios. P(No/V) = 2/5*2/5*0.1*5/8 = 0.01. Hence the probability is 0.01 after smoothing for Rain = No

Let's assume Rain = Yes, and calculate the probability of Rain = Yes using the naïve bayes classification.

P(Yes/V) = P(V/Yes) * P(Rain/Yes)
= Probability of High Temp for Rain as Yes * Probability of Windy Yes for Rain as Yes*Probability of High Humidity for Rain as Yes * Probability of Rain as Yes
= 2/3*2/3*1/3*3/8 (Check columns shown in Blue above)
= 4/72
= 0.056

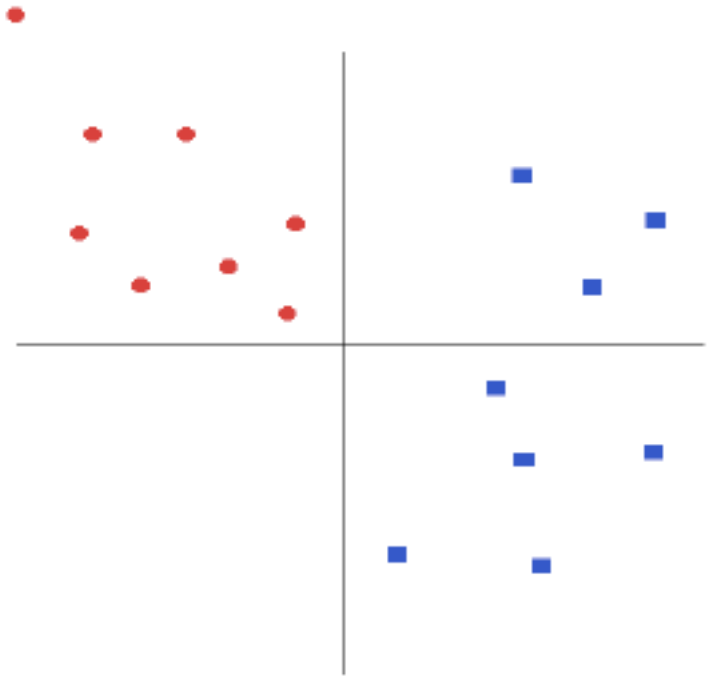
- What is the basic assumption of the Naive Bayes classier and why is it a drawback when it comes to solving real life problems?

The basic assumption of Naïve bayes classifier is to consider each attribute as independent, hence it multiplies the probability of attributes to calculate the final probability. In a real-life problem, this will not always be true, the events or the attributes may be dependent on each other. Hence the output of the naïve bayes may not be fully accurate. This algorithm is called Naïve because of this Naïve assumption of considering all the events as independent.

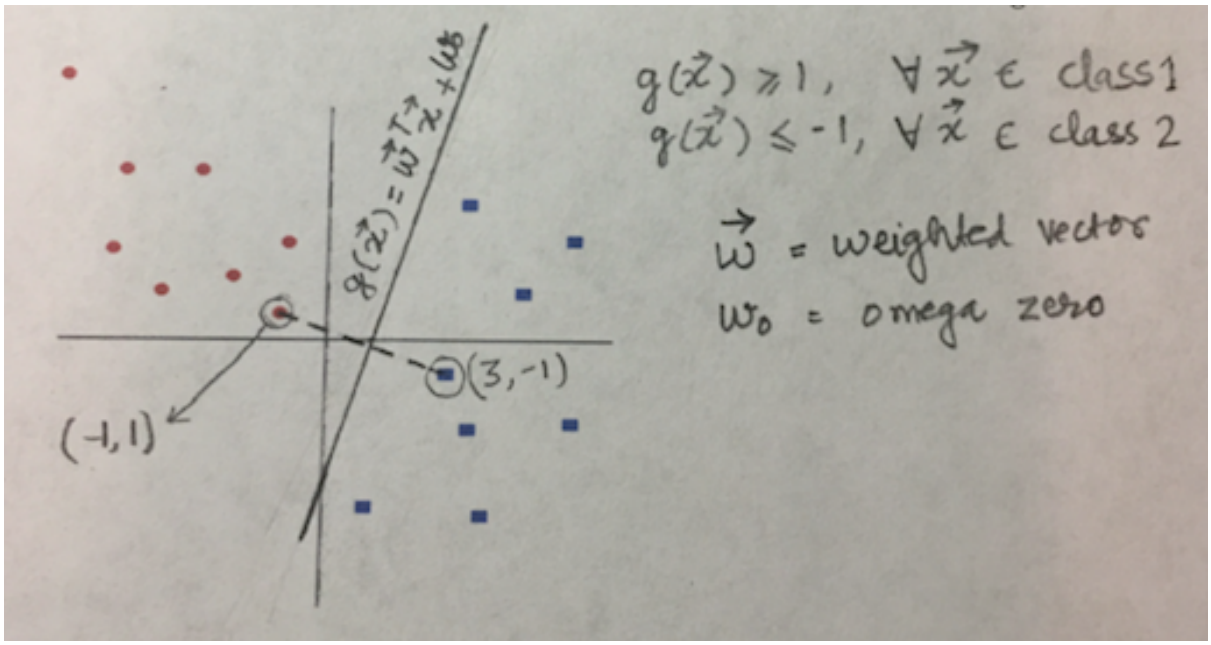
Support Vector Machines

- For the classification space given below, identify the support vectors that help define the margin of separation:

Name: Vipin Singh
Email: vipsingh@iu.edu



The below picture shows the hyperplane separating the two data sets classs 1 and class2. The equation of the hyperplane is show below. In next few steps, we will solve the equation of the hyperplane drawn below. The weighted vector is calculated by checking the distance between the two nearest points in each class. Once the weighted vector is identified the constant a and omega zero are calculated to derive the final equation of the hyperplane.



Name: Vipin Singh
Email: vipsingh@iu.edu

Step 1 – Calculating the weighted vector and Omega zero (w_0). Considering two points as shown on graph

Weighted vector = (3, -1) – (-1,1) = (4a, -2a)

Step2 - Hyperplane equation based on the weighted vectors for two points of Class1 and Class2. For more see references

Using Point (3, -1) => $12a + 2a + w_0 = 1$ -- Class 1
Using Point (-1, 1) => $4a - 2a + w_0 = -1$ -- Class 2

Step 3 - Using the above two equation calculate the value of "a" to determine weighted vector and Omega Zero w_0
Solving the two equations as shown below
$w_0 = 6a - 1$ ----------Equation 1
$w_0 = 1 - 14a$ ----------Equation 2

Substituting the w_0 from Equation 2 to Equation 1

$a = 1/10$

Solving Equation 1 for w_0

$w_0 = -4/10$

Step 4 – Weighted vector = (4/10, -2/10) = (2/5, -1/5)
Substituting the values of Weighted Vector and w_0 will give the final equation for hyperplane.

$g(x) = 2x_1/5 - x_2/5 - 4/10$

Name: Vipin Singh
Email: vipsingh@iu.edu

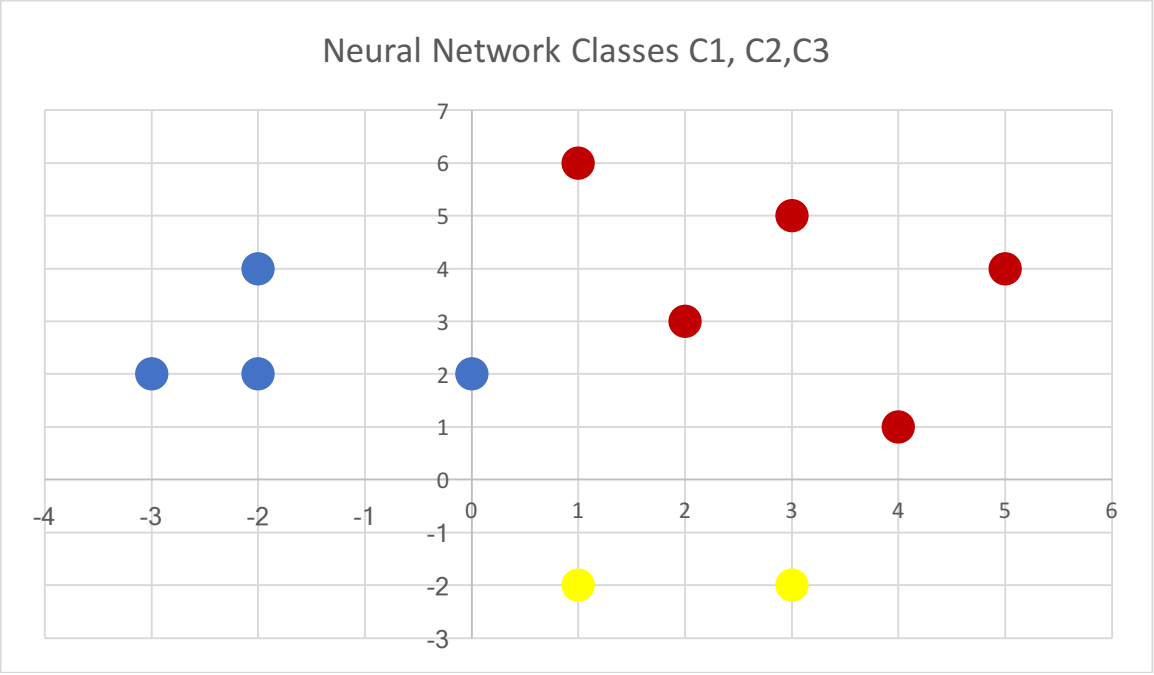
- What is the underlying assumption of SVMs. Explain briefly how SVMs can be applied in non-linearly separable spaces.

  I think the underlying assumption of SVM is that it assumes that the sample data is independent and identically distributed.
  SVM's can be applied in non-linearly spaces by using kernel trick functions. These functions take low dimensional input space and transform it to a higher dimensional space i.e. it converts the non-separable problem to separable problem. These functions perform extremely complex data transformation, then find out the process to separate the data based on the labels defined.

Neural Networks

- Given the following 3-class classication problem:
  C1: (4,1), (2,3), (3,5), (5,4), (1,6)
  C2: (0,2), (-2,2), (-3,2), (-2,4)
  C3: (1, -2), (3, -2)

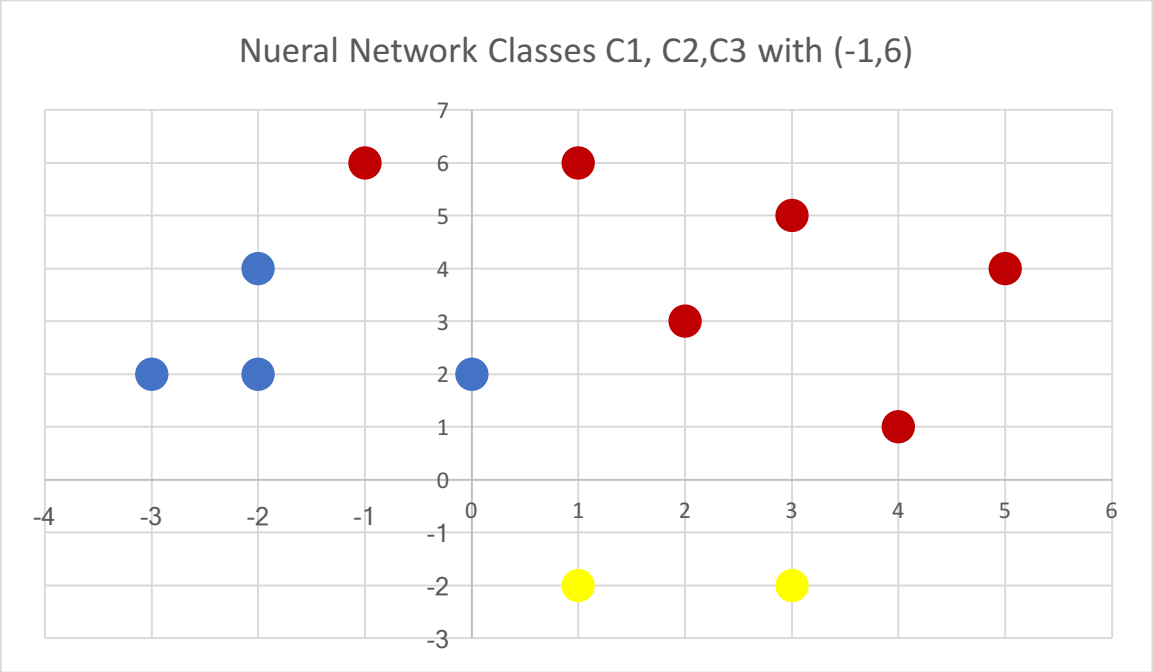
  a) Will a single layer neural network be sufficient to predict the class of the input data? Justify your answer.



Name: Vipin Singh
Email: vipsingh@iu.edu

The C1, C2 and C3 classes are plotted as Red, Blue and Yellow respectively. If we observe the graph above it seems that a single layer network will not be sufficient to predict the class of the input data, since the classes cannot be separated by single line. This classification need to be solved by more complex network using hidden layers.

b) Add the sample (-1,6) to C1. Repeat part (a).



Adding the point (-1,6) don't change the scenario much, still the classes cannot be separated by single layer neural network using the single lines. This classification need more complex network using hidden layers.

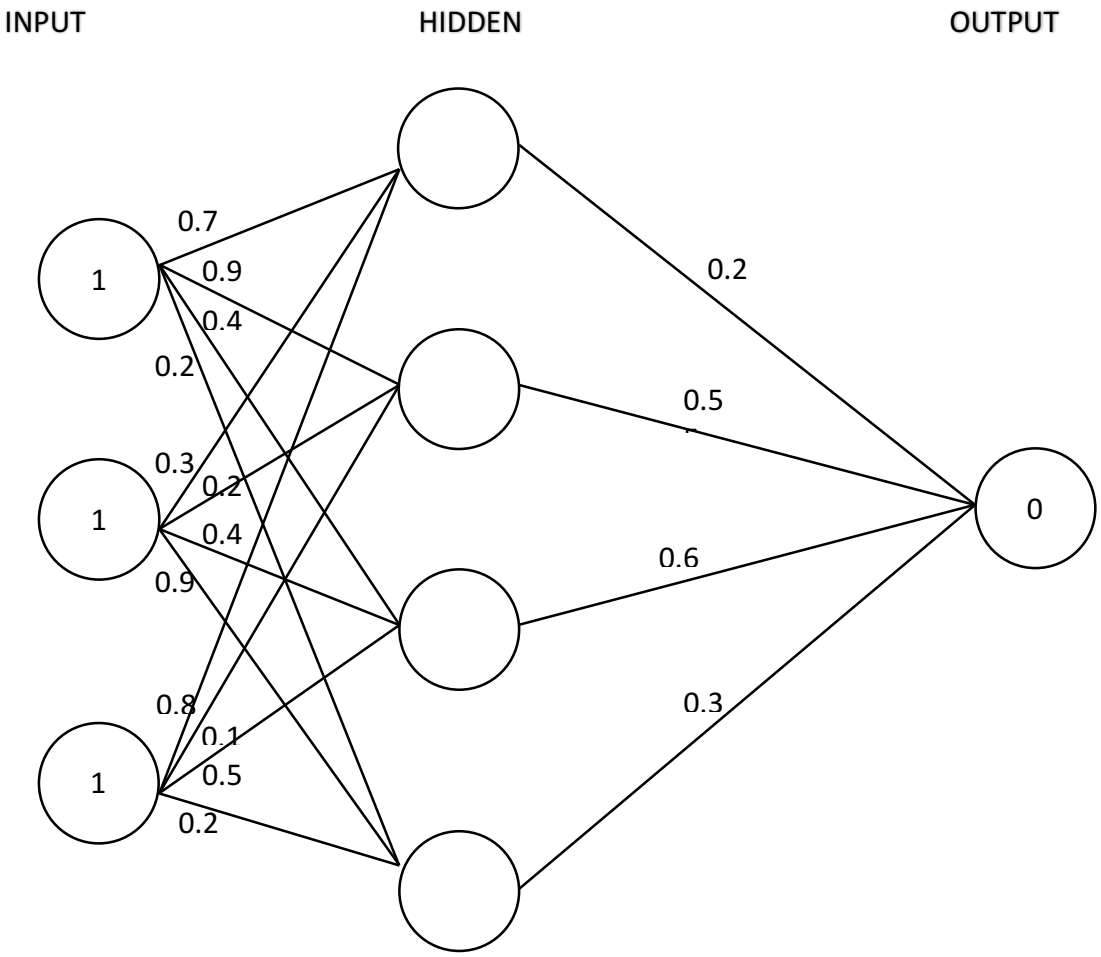
- Design a neural network to determine output 'd' based on the inputs a, b and c. Choose the appropriate activation function and number of layers to solve the neural net problem.

a	b	c	d
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Name: Vipin Singh
Email: vipsingh@iu.edu

To design the neural network of the above table, we need to consider one row form the table, draw a network with input, hidden and output layer, choose the random weights between 0 and 1 for each synapse in the network. Choose the activation function and do the forward propagation with randomly chosen weights. Once the output is generated, it is compared with the expected output to get the margin of error. The backward propagation is done to adjusts the weights at each synapse to bring the output near to the expected. In this way, many times this process of forward propagation and backward propagation is repeated to train the model for the expected output. The model output can be compared by using different activation functions, the best fitting activation function is chosen.

Let's take the last row of the above table, a=1, b=1, c= 1 and d = 0. Draw a following neural network with single hidden layer with four neurons. Assign the random weights to the synapses.

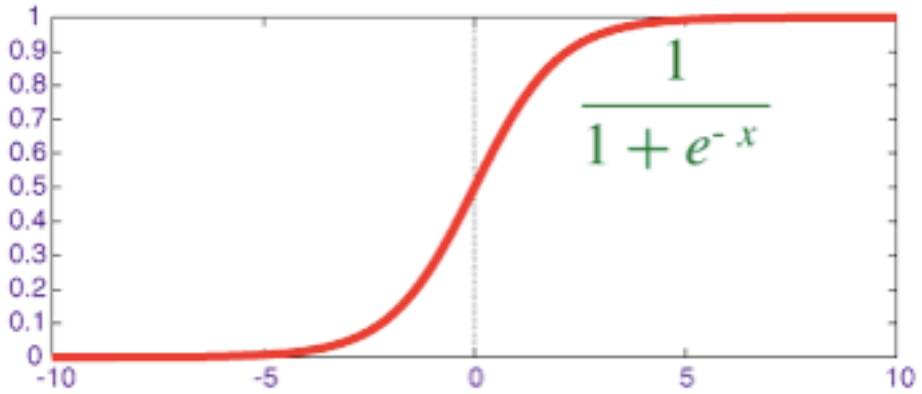


Name: Vipin Singh
Email: vipsingh@iu.edu

Compute the values of the hidden layer by multiplying with weights and summing up as follows

1*0.7 + 1*0.3 + 1*0.8 = 1.8
1*0.9 + 1*0.2 + 1*0.1 = 1.2
1*0.4 + 1*0.4 + 1*0.1 = 0.9
1*0.2 + 1*0.9 + 1*0.2 = 1.3

Let's choose the activation function as Sigmoid Function.



Applying Sigmoid function to the four neurons of the hidden layer

f (1.8) = 1 / (1 + $e^{-1.8}$) = 0.858
f (1.2) = 1 / (1 + $e^{-1.2}$) = 0.768
f (0.9) = 1 / (1 + $e^{-0.9}$) = 0.711
f (1.3) = 1 / (1 + $e^{-1.3}$) = 0.785

Next step is to sum the output of the hidden layer with next set of weights

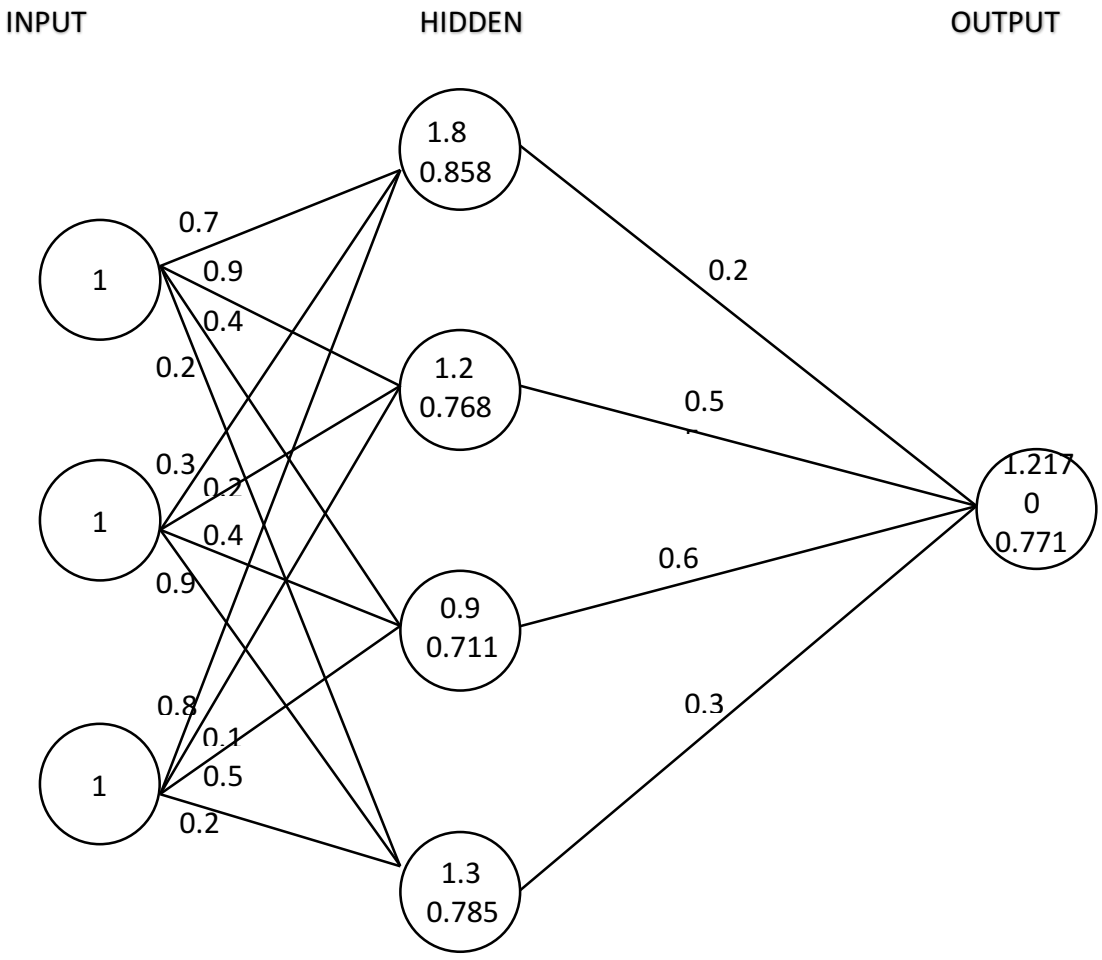
0.858*0.2 + 0.768*0.5 + 0.711*0.6 + 0.785*0.3 = 1.217

Apply the activation function (Sigmoid)

f (1.217) = 1 / (1 + $e^{-1.217}$) = 0.771

Name: Vipin Singh
Email: vipsingh@iu.edu

Showing the neural network with the above calculated values.



Target Output is 0, but by using the random weights and activation function it is 0.771 which is the result of first forward propagation. Hence, margin of error = target – calculated = 0 – 0.771 = -0.771.

This margin of error is used in the backward propagation to adjust the weight at each synapse to achieve the target output, then again forward propagation is done with new weights to achieve the target value. This process of forward propagation and backward propagation is repeated multiple times to reduce the margin error and improve the model. Once the output is near to the target out the neural network model is ready to classify the new input data.

Name: Vipin Singh
Email: vipsingh@iu.edu

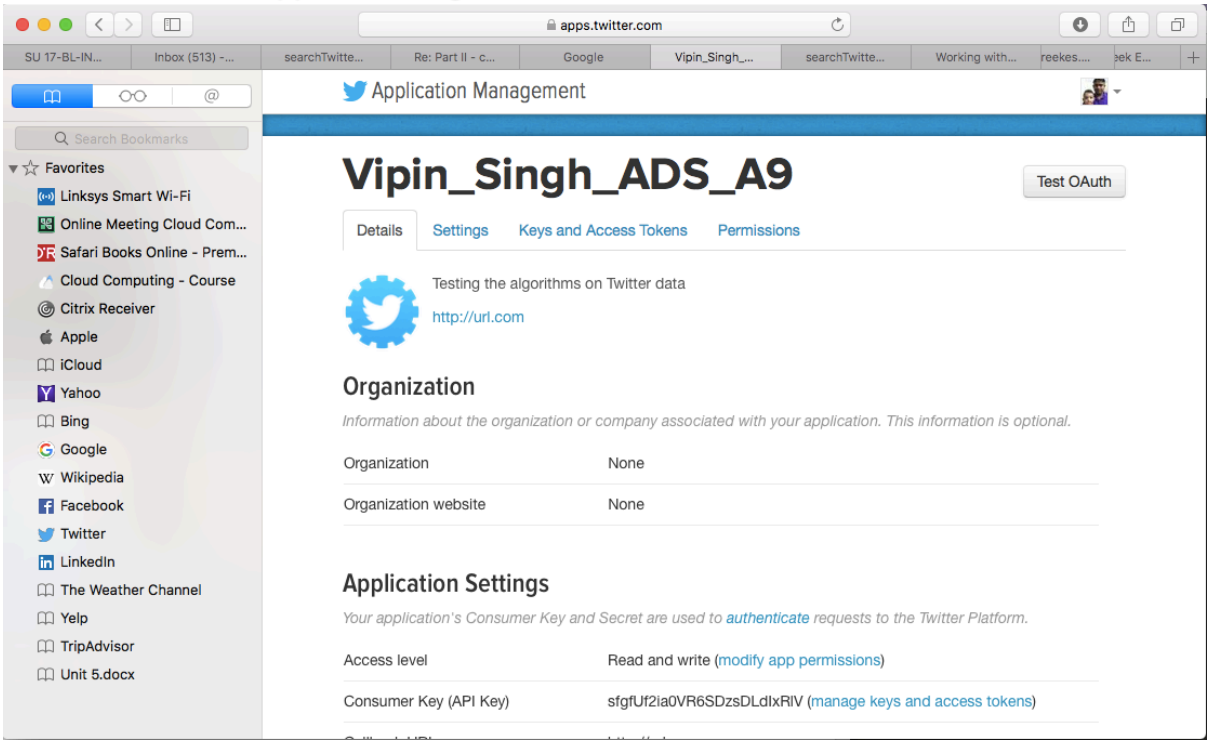
2. Part 2

Extract tweets using the Twitter API as demonstrated in the tutorial for two specific hashtags: '#dog' and '#cat'. Collect equal number of tweets for each of the hashtags and it is not necessary to collect more than 5000 tweets in total. A given tweet belongs to class 'dog' if it contains '#dog'(Even if the tweet contains both '#dog' and '#cat' it will belong to class 'dog') and belongs to class 'cat' otherwise.

In this part of the assignment, you will be using the dataset created above to develop classification models. Create appropriate features and develop models using Naive Bayes, SVMs and Neural nets and compare their accuracies. Make sure to split the data into 80-20 for training and validation.

You DO NOT have to submit the dataset you used for the analysis. Provide detailed explanation on how you created the model. Reflect on the performance of each of the models in correctly predicting the class of the tweet.

Created the twitter application to get the tweets.



Name: Vipin Singh
Email: vipsingh@iu.edu

Modified the code to collect around 5000 tweets by passing the value of n= 5000

```
def searchTwitter(query,api=api,n=5000):
    r = []
    qs = 0
    if len(r)==0:
        r.extend([t for t in api.request("search/tweets",{'q':query,'count':n})])
#        print('length of r',len(r),'Value of n',n)
        qs +=1
#        print('qs-1',qs)
        while len(r) < n:
            print("Querrying twitter for {}. {}/{} gathered.".format(query,len(r),n))
            last =  r[-1]['id']
#            print('last',last)
            r.extend([t for t in api.request("search/tweets",{'q':query,'count':n,'max_id':last})])
            qs += 1
#            print('qs',qs)
            if qs > 180:
                time.sleep(840)
                qs   =  0
    return r[:n]
```

searchTwitter

The below code snippet calls the searchTwitter and collect the tweets for cats as the tweets with tag '#cats - #dogs' and for dogs '#dogs - #cats'. These tags will ensure that if the #cats are in the beginning it will be considered as cat tweet, else dogs. The last two line in the below snippet are added to remove the duplicates in collected data.

```
#Get JSON from Twitter
cats = searchTwitter('#cats -#dogs')
dogs = searchTwitter('#dogs -#cats')
#
## Convert the json returned by Twitter into a dataframe
cats_d = pd.read_json(json.dumps(cats))
dogs_d = pd.read_json(json.dumps(dogs))
#


cats_d['text'] = cats_d["text"].apply(preprocess)
cats_d['text'] = cats_d["text"].apply(normalize)

dogs_d['text'] = dogs_d["text"].apply(preprocess)
dogs_d['text'] = dogs_d["text"].apply(normalize)



print('dogs_lentgh',len(dogs_d))
print('cats_length', len(cats_d))

cats_d=cats_d.drop_duplicates(subset=['text'], keep='first')
dogs_d=dogs_d.drop_duplicates(subset=['text'], keep='first')
```

Name: Vipin Singh
Email: vipsingh@iu.edu

preprocess

Created a function 'preprocess', to clean and preprocess the data. The preprocess function is converting the tweet in to lowercase, replacing the regular expression of a url with URL, converting @ to AT_USER, removing additional white spaces, removing quotes, removing citation, removing tickers, removing numbers and trimming the tweets.

```
#start process_tweet
def preprocess(tweet):
    # process the tweets
    #Convert to lower case
    tweet = tweet.lower()
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Remove quotes
    tweet = re.sub('&amp;quot;|&amp;amp', '', tweet)
    #Remove citations
    tweet = re.sub('@[a-zA-Z0-9]*', '', tweet)
    #Remove tickers
    tweet = re.sub('\$[a-zA-Z0-9]*', '', tweet)
    #Remove numbers
    tweet = re.sub('[0-9]*','',tweet)
    #trimming
    tweet = tweet.strip('\'"')

    return tweet
```

Classification models and features

Total tweets retrieved are 5000, but after removing the duplicates dogs tweets are 2799 and cats 2770 as per the below output screenshot.

```
dogs_lentgh 5000
cats_length 5000
Duplicates Removed
dogs_lentgh 2799
cats_length 2770
```

The vectorizer created the sparse matrix for cats and dogs. The total number of features created were 10485 as per the below screenshot.

Name: Vipin Singh
Email: vipsingh@iu.edu

```
>>> cats_tdm.shape
(2770, 10485)
>>> dogs_tdm.shape
(2799, 10485)
```

The catsdogs matrix is as shown below. This data is then split in to 80:20 train and test. The ravel command changed the Y train and test in to vector. The data is accurately split, 80% of 5569 is 4455.2, hence 4455 for train and remaining for test.

```
>>> catsdogs.shape
(5569, 10485)
```

```
>>> trainX.shape
(4455, 10485)
>>> testX.shape
(1114, 10485)
>>> trainY.shape
(4455,)
>>> testY.shape
(1114,)
```

The models are evaluated on three parameters – Precision, Recall and F-Score.

Precision is the fraction of the relevant instances among the retrieved instances, in other words if there is a set of 12 animals (cats and dogs) to classify, and the algorithm predicted that there are 8 dogs and 4 cats. This classification is compared with the actual or real classification to understand the True Positive, False Positive, True negative and False negative predicted by algorithm. In our example if out of the 8 dogs predicted, only 5 are really dogs and 3 are cats, then the precision will 5/8.

Recall is the fraction of relevant instances that have been retrieved over total relevant instances. In the above example the recall is 5/12. The algorithm can recall or retrieve 5/12 from the complete data set.

F-Score is the harmonic-mean of the precision and recall.

Naïve Bayes

The model for Naïve bayes is created by importing the GaussianNB (Gaussian naïve Bayes) from Scikit-learn library (sklearn.naive_bayes). The Naive Bayes class is created to fit the trainX and trainY dataset. A score function is called to evaluate the performance of the model. The precision and Recall are 0.873 and 0.759 respectively. Our model predicts well and gives a F-Score of 0.812. Please see the below screen shot for details.

Name: Vipin Singh
Email: vipsingh@iu.edu

```
Naive Bayes Performance

Precision:   0.873
Recall:      0.759
F-Score:     0.812
```

Support Vector Machine

The model for SVM is created by importing the SVC (C –Support Vector Classification) from Scikit-learn library (sklearn.svm). SVC fits the trainX and trainY dataset. A score function is called to evaluate the performance of the model. The precision and Recall are 0.946 and 0.993 respectively. Our model predicts well and gives a F-Score of 0. 969.This model shows better performance compared to Naïve Bayes if we look at the F-Score of naïve bayes and SVM. Please see the below screen shot for details.

```
SVM performance

Precision:   0.946
Recall:      0.993
F-Score:     0.969
```

Neural Networks

The model for neural network is created by importing the MLPClassifier from Scikit-learn library (sklearn.neural_network). MLPClassifier fits the trainX and trainY dataset. A score function is called to evaluate the performance of the model. The precision and Recall are 0.978 and 0.976 respectively. Our model predicts well and gives a F-Score of 0. 977.This model shows better performance compared to SVM if we look at the F-score of SVM and Neural Network. Please see the below screen shot for details.

```
Neural Network Performance

Precision:   0.978
Recall:      0.976
F-Score:     0.977
```

Name: Vipin Singh
Email: vipsingh@iu.edu

3. Part 3

For each question, determine which algorithm is best. Discuss why you chose the algorithm you did and why you did not select the others.

- You have 100 observations of users top 5 favorite songs on a popular music streaming site, what algorithm is best to predict the favorite song of 20 users based on their other four favorite songs

  The best algorithm in this scenario is Naïve Bayes since this algorithm is good in prediction for small data set. This algorithm uses the conditional probability which suits best to this problem. Neural network is not the best choices here because less data set and no deep machine learning is required. SVM is also good for large data sets, hence not a good choice in this scenario.

- You have 5 billion observations with 300 million variables from your most recent self-driving car testing, but you're missing 5% of your data because of an intermittent sensor failure. Which algorithm can help you complete your dataset?

  In this scenario, the best algorithm is Neural Network because it is very good with large data sets and it works well even if there is missing data. SVM is not a good choice here because of the missing 5% data, and Naïve Bayes as explained earlier is not a best choice for large data sets.

- You have 5 billion observations with 300 million variables from your recent self-driving car testing. Each observation has been labeled as either “hazard" or “safe", depending on the safety of the situation. Which algorithm can help you predict whether new observations are occurring in hazardous or safe situations?

  Support Vector Machine is the best algorithm in this scenario, it works well with the large data sets and it is a true classifier. As explained earlier neural network can even work with missing data sets and it is not a true classifier hence not the best fit in this scenario. Naïve Bayes is not the best here due to huge data set.

Name: Vipin Singh
Email: vipsingh@iu.edu

References:

https://www.youtube.com/watch?v=llVINQDk4o8
https://www.youtube.com/watch?v=1NxnPkZM9bc&t=317s
https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/
https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/
https://en.wikipedia.org/wiki/Precision_and_recall

Appendix

```
# Twitter Python Script

import json
import pandas as pd
import numpy as np
import sklearn
import re
from sklearn.feature_extraction import text as sk_fe_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, f1_score, recall_score
from TwitterAPI import TwitterAPI
from nltk import PorterStemmer, WordNetLemmatizer
from stopwords import STOPWORDS
import time

#Set up the variables for your 'application'
consumerkey = 'sfgfUf2ia0VR6SDzsDLdIxRlV'
consumersecret = 'J5B46ysBCXpQRT0yvEZk1idoEyu22tGihUJe0msnhilXVTX0wU'

#Setup your API key
api = TwitterAPI(consumerkey,consumersecret,auth_type='oAuth2')

def score(true,pred):
  return (precision_score(true,pred),
        recall_score(true,pred),
        f1_score(true,pred))

def print_score(s):
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
 print("""
Precision:  {:0.3}
Recall:     {:0.3}
F-Score:    {:0.3}
""".format(*s))

#def searchTwitter(query,maximumID,feed="search/tweets",api=api,n=100):
#  return [t for t in api.request(feed, {'q':query,'count':n, 'max_id':maximumID})]


def searchTwitter(query,api=api,n=5000):
   r = []
   qs = 0
   if len(r)==0:
      r.extend([t for t in api.request("search/tweets",{'q':query,'count':n})])
#       print('length of r',len(r),'Value of n',n)
      qs +=1
#       print('qs-1',qs)
      while len(r) < n:
         print("Querrying twitter for {}. {}/{} gathered.".format(query,len(r),n))
         last =  r[-1]['id']
#          print('last',last)
         r.extend([t for t in api.request("search/tweets",{'q':query,'count':n,'max_id':last})])
         qs += 1
#          print('qs',qs)
         if qs > 180:
            time.sleep(840)
            qs   =  0
   return r[:n]

def normalize(x,style="PorterStemmer"):
 """This normalizer will remove stopwords and normalize text"""
 x = [t.lower() for t in x.split() if t not in STOPWORDS]
 if style == "PorterStemmer":
  return " ".join(list(map(PorterStemmer().stem,x)))
 elif style == "Lemma":
  try: return " ".join(list(map(WordNetLemmatizer.lemmatize,x)))
  except: return " ".join(x)
 elif style == "CharGram":
  cs = " ".join(x)
  return " ".join([cs[i:i+3] for i in range(0,len(cs),3)])
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
  else:
    return " ".join(x)

#start process_tweet
def preprocess(tweet):
    # process the tweets
    #Convert to lower case
    tweet = tweet.lower()
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Remove quotes
    tweet = re.sub('&amp;quot;|&amp;amp', '', tweet)
    #Remove citations
    tweet = re.sub('@[a-zA-Z0-9]*', '', tweet)
    #Remove tickers
    tweet = re.sub('\$[a-zA-Z0-9]*', '', tweet)
    #Remove numbers
    tweet = re.sub('[0-9]*','',tweet)
    #trim
    tweet = tweet.strip('\'"')
    return tweet



#Get JSON from Twitter
cats = searchTwitter('#cats -#dogs')
dogs = searchTwitter('#dogs -#cats')
#
## Convert the json returned by Twitter into a dataframe
cats_d = pd.read_json(json.dumps(cats))
dogs_d = pd.read_json(json.dumps(dogs))
#



cats_d['text'] = cats_d["text"].apply(preprocess)
cats_d['text'] = cats_d["text"].apply(normalize)
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
dogs_d['text'] = dogs_d["text"].apply(preprocess)
dogs_d['text'] = dogs_d["text"].apply(normalize)



print('dogs_lentgh',len(dogs_d))
print('cats_length', len(cats_d))

cats_d=cats_d.drop_duplicates(subset=['text'], keep='first')
dogs_d=dogs_d.drop_duplicates(subset=['text'], keep='first')

print('Duplicates Removed')

# print(cats_d.duplicated(['text']))

print('dogs_lentgh',len(dogs_d))
print('cats_length', len(cats_d))

#print('cats ----',cats_d['text'])
#
#print('--------')
#
#print('dogs - ',dogs_d['text'])

## If you would like to look at the full data frame
## cats_d.info()
## dogs_d.info()
#
##Get text only and replace hashtags with blanks
##If you want to use the normalizer, import it above and pass x.replace() to the noramlizer function
#
cats_text = [x.replace('#cats',"") for x in cats_d['text']]
dogs_text = [x.replace('#dogs',"") for x in dogs_d['text']]
#
#Create features and return sparse matricies
vectorizer = sk_fe_text.CountVectorizer(cats_text+dogs_text)
vectorizer.fit(cats_text+dogs_text)
cats_tdm = vectorizer.transform(cats_text).toarray()
dogs_tdm = vectorizer.transform(dogs_text).toarray()
```

Name: Vipin Singh
Email: vipsingh@iu.edu

```
#Create visible matricies and combine
zeros = np.zeros((len(cats_text),1))
ones = np.ones((len(dogs_text),1))
catsdogs = np.concatenate((cats_tdm,dogs_tdm),axis=0)
y = np.ravel(np.concatenate((zeros,ones),axis=0))

#Create train/test split for modeling
trainX,testX,trainY,testY = train_test_split(catsdogs,y,test_size=.20)

#Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(trainX,trainY)

print("\n\nNaive Bayes Performance")
s = score(testY,nb.predict(testX))
print_score(s)

#SVM
from sklearn.svm import SVC
svm = SVC()
svm.fit(trainX,trainY)

print("\n\nSVM performance")
s = score(testY,svm.predict(testX))
print_score(s)

#Neural Network
from sklearn.neural_network import MLPClassifier
nn = MLPClassifier()
nn.fit(trainX,trainY)

print("\n\nNeural Network Performance")
s = score(testY,nn.predict(testX))
print_score(s)
```

Name: Vipin Singh
Email: vipsingh@iu.edu

Output of the Program

```
>>> runfile('/Users/vipinsingh/Desktop/ADS/Assignment9/Twitter.py',
wdir='/Users/vipinsingh/Desktop/ADS/Assignment9')

Querrying twitter for #cats -#dogs. 89/5000 gathered.
Querrying twitter for #cats -#dogs. 176/5000 gathered.
Querrying twitter for #cats -#dogs. 259/5000 gathered.
Querrying twitter for #cats -#dogs. 341/5000 gathered.
Querrying twitter for #cats -#dogs. 429/5000 gathered.
Querrying twitter for #cats -#dogs. 498/5000 gathered.
Querrying twitter for #cats -#dogs. 575/5000 gathered.
Querrying twitter for #cats -#dogs. 640/5000 gathered.
Querrying twitter for #cats -#dogs. 723/5000 gathered.
Querrying twitter for #cats -#dogs. 823/5000 gathered.
Querrying twitter for #cats -#dogs. 911/5000 gathered.
Querrying twitter for #cats -#dogs. 1011/5000 gathered.
Querrying twitter for #cats -#dogs. 1098/5000 gathered.
Querrying twitter for #cats -#dogs. 1181/5000 gathered.
Querrying twitter for #cats -#dogs. 1266/5000 gathered.
Querrying twitter for #cats -#dogs. 1342/5000 gathered.
Querrying twitter for #cats -#dogs. 1436/5000 gathered.
Querrying twitter for #cats -#dogs. 1515/5000 gathered.
Querrying twitter for #cats -#dogs. 1589/5000 gathered.
Querrying twitter for #cats -#dogs. 1675/5000 gathered.
Querrying twitter for #cats -#dogs. 1767/5000 gathered.
Querrying twitter for #cats -#dogs. 1856/5000 gathered.
Querrying twitter for #cats -#dogs. 1943/5000 gathered.
Querrying twitter for #cats -#dogs. 2026/5000 gathered.
Querrying twitter for #cats -#dogs. 2114/5000 gathered.
Querrying twitter for #cats -#dogs. 2189/5000 gathered.
Querrying twitter for #cats -#dogs. 2273/5000 gathered.
Querrying twitter for #cats -#dogs. 2342/5000 gathered.
Querrying twitter for #cats -#dogs. 2415/5000 gathered.
Querrying twitter for #cats -#dogs. 2494/5000 gathered.
Querrying twitter for #cats -#dogs. 2581/5000 gathered.
Querrying twitter for #cats -#dogs. 2670/5000 gathered.
Querrying twitter for #cats -#dogs. 2755/5000 gathered.
Querrying twitter for #cats -#dogs. 2832/5000 gathered.
Querrying twitter for #cats -#dogs. 2880/5000 gathered.
Querrying twitter for #cats -#dogs. 2964/5000 gathered.
Querrying twitter for #cats -#dogs. 3051/5000 gathered.
Querrying twitter for #cats -#dogs. 3124/5000 gathered.
Querrying twitter for #cats -#dogs. 3187/5000 gathered.
Querrying twitter for #cats -#dogs. 3276/5000 gathered.
Querrying twitter for #cats -#dogs. 3376/5000 gathered.
Querrying twitter for #cats -#dogs. 3476/5000 gathered.
Querrying twitter for #cats -#dogs. 3576/5000 gathered.
Querrying twitter for #cats -#dogs. 3669/5000 gathered.
Querrying twitter for #cats -#dogs. 3768/5000 gathered.
Querrying twitter for #cats -#dogs. 3867/5000 gathered.
Querrying twitter for #cats -#dogs. 3967/5000 gathered.
Querrying twitter for #cats -#dogs. 4059/5000 gathered.
Querrying twitter for #cats -#dogs. 4159/5000 gathered.
Querrying twitter for #cats -#dogs. 4256/5000 gathered.
Querrying twitter for #cats -#dogs. 4355/5000 gathered.
Querrying twitter for #cats -#dogs. 4455/5000 gathered.
Querrying twitter for #cats -#dogs. 4555/5000 gathered.
Querrying twitter for #cats -#dogs. 4655/5000 gathered.
Querrying twitter for #cats -#dogs. 4753/5000 gathered.
Querrying twitter for #cats -#dogs. 4844/5000 gathered.
```

Name: Vipin Singh
Email: vipsingh@iu.edu

Querrying twitter for #cats -#dogs. 4944/5000 gathered.
Querrying twitter for #dogs -#cats. 97/5000 gathered.
Querrying twitter for #dogs -#cats. 197/5000 gathered.
Querrying twitter for #dogs -#cats. 296/5000 gathered.
Querrying twitter for #dogs -#cats. 394/5000 gathered.
Querrying twitter for #dogs -#cats. 491/5000 gathered.
Querrying twitter for #dogs -#cats. 575/5000 gathered.
Querrying twitter for #dogs -#cats. 664/5000 gathered.
Querrying twitter for #dogs -#cats. 762/5000 gathered.
Querrying twitter for #dogs -#cats. 858/5000 gathered.
Querrying twitter for #dogs -#cats. 942/5000 gathered.
Querrying twitter for #dogs -#cats. 1041/5000 gathered.
Querrying twitter for #dogs -#cats. 1135/5000 gathered.
Querrying twitter for #dogs -#cats. 1219/5000 gathered.
Querrying twitter for #dogs -#cats. 1319/5000 gathered.
Querrying twitter for #dogs -#cats. 1393/5000 gathered.
Querrying twitter for #dogs -#cats. 1488/5000 gathered.
Querrying twitter for #dogs -#cats. 1588/5000 gathered.
Querrying twitter for #dogs -#cats. 1688/5000 gathered.
Querrying twitter for #dogs -#cats. 1775/5000 gathered.
Querrying twitter for #dogs -#cats. 1869/5000 gathered.
Querrying twitter for #dogs -#cats. 1969/5000 gathered.
Querrying twitter for #dogs -#cats. 2065/5000 gathered.
Querrying twitter for #dogs -#cats. 2158/5000 gathered.
Querrying twitter for #dogs -#cats. 2254/5000 gathered.
Querrying twitter for #dogs -#cats. 2346/5000 gathered.
Querrying twitter for #dogs -#cats. 2423/5000 gathered.
Querrying twitter for #dogs -#cats. 2502/5000 gathered.
Querrying twitter for #dogs -#cats. 2595/5000 gathered.
Querrying twitter for #dogs -#cats. 2694/5000 gathered.
Querrying twitter for #dogs -#cats. 2773/5000 gathered.
Querrying twitter for #dogs -#cats. 2835/5000 gathered.
Querrying twitter for #dogs -#cats. 2899/5000 gathered.
Querrying twitter for #dogs -#cats. 2967/5000 gathered.
Querrying twitter for #dogs -#cats. 3041/5000 gathered.
Querrying twitter for #dogs -#cats. 3133/5000 gathered.
Querrying twitter for #dogs -#cats. 3207/5000 gathered.
Querrying twitter for #dogs -#cats. 3289/5000 gathered.
Querrying twitter for #dogs -#cats. 3389/5000 gathered.
Querrying twitter for #dogs -#cats. 3483/5000 gathered.
Querrying twitter for #dogs -#cats. 3566/5000 gathered.
Querrying twitter for #dogs -#cats. 3666/5000 gathered.
Querrying twitter for #dogs -#cats. 3764/5000 gathered.
Querrying twitter for #dogs -#cats. 3857/5000 gathered.
Querrying twitter for #dogs -#cats. 3949/5000 gathered.
Querrying twitter for #dogs -#cats. 4041/5000 gathered.
Querrying twitter for #dogs -#cats. 4141/5000 gathered.
Querrying twitter for #dogs -#cats. 4241/5000 gathered.
Querrying twitter for #dogs -#cats. 4341/5000 gathered.
Querrying twitter for #dogs -#cats. 4440/5000 gathered.
Querrying twitter for #dogs -#cats. 4540/5000 gathered.
Querrying twitter for #dogs -#cats. 4640/5000 gathered.
Querrying twitter for #dogs -#cats. 4740/5000 gathered.
Querrying twitter for #dogs -#cats. 4834/5000 gathered.
Querrying twitter for #dogs -#cats. 4934/5000 gathered.
dogs_lentgh 5000
cats_length 5000
Duplicates Removed
dogs_lentgh 2799
cats_length 2770

Name: Vipin Singh
Email: vipsingh@iu.edu

Naive Bayes Performance

Precision: 0.873
Recall: 0.759
F-Score: 0.812

SVM performance

Precision: 0.946
Recall: 0.993
F-Score: 0.969

Neural Network Performance

Precision: 0.978
Recall: 0.976
F-Score: 0.977