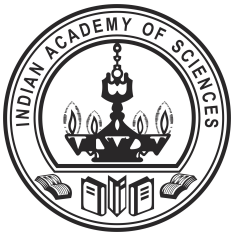
Indian Institute of Science, Bangalore
Supercomputer Education and Research Centre

Application of Machine Learning Techniques for Twitter User Classification

Chirag Nagpal

Guide: Prof. N Balakrishnan

Summer Research Fellowship - 2014
Final Report



Indian Academy of Sciences, Bangalore
Indian National Science Academy, New Delhi
National Academy of Sciences, Allahabad

May - July 2014

CERTIFICATE

This is to certify that,
"Application of Machine Learning Techniques for
Twitter User Classification"
is a bonafide work carried out by
Chirag Nagpal
for the fulfillment of the completion of
The Science Academies'
Summer Research Fellowship, 2014

It is certified that all the corrections and suggestions indicated during the course of two months have been incorporated in the report. The project report has been approved as it satisfies all the requirements.

Prof. N. Balakrishnan
July 2014

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the Indian Academy of Sciences for providing me this opportunity to work at the Indian Institute of Science, Bangalore.

I am deeply indebted to Prof. N. Balakrishnan for his guidance and encouragement.

I would also like to express my gratitude to Ms. Khushboo Singhal and Ms. K. Nagarathna for their help and support.

A special thanks to Prof. SR. Dhore, HoD, Compter Engineering, Army Institute of Technology, University of Pune, without whose support, this fellowship would not have been possible.

Chirag Nagpal

Contents

List of Figures

List of Tables

Introduction 1

1.1 Need of User Classification

Over a period of time, Microblogging services like Twitter have grown in popularity rapidly, with millions of users. As a source of information, these services have been found to be faster than even news channels. These services also allow users to interact not only with their friends and acquaintances, but also with celebrities and organisations. Recently government agencies from various countries, including India too, have started using Twitter as a means to stay in contact with people.

With millions of users, to perform any analysis or information retrieval, a need was felt to classify users, using machine learning algorithms. In this study we propose a machine learning approach to user classification, using only ambient metadata, like profile description, number of followers, number of users followed etc.

1.2 Machine Learning

Machine Learning, is a branch of Artificial Intelligence, concerned with the study of computer systems that can learn from data. Machine Learning algorithms can be broadly divided into supervised, unsupervised and semisupervised.

These algorithms take various different approaches for learning, which

include Decision Trees, Artificial Neural Networks, Bayesian Networks and Clustering.

1.3 Supervised Learning

Supervised learning is the machine learning task of inferring a function from labeled training data.[5] The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen

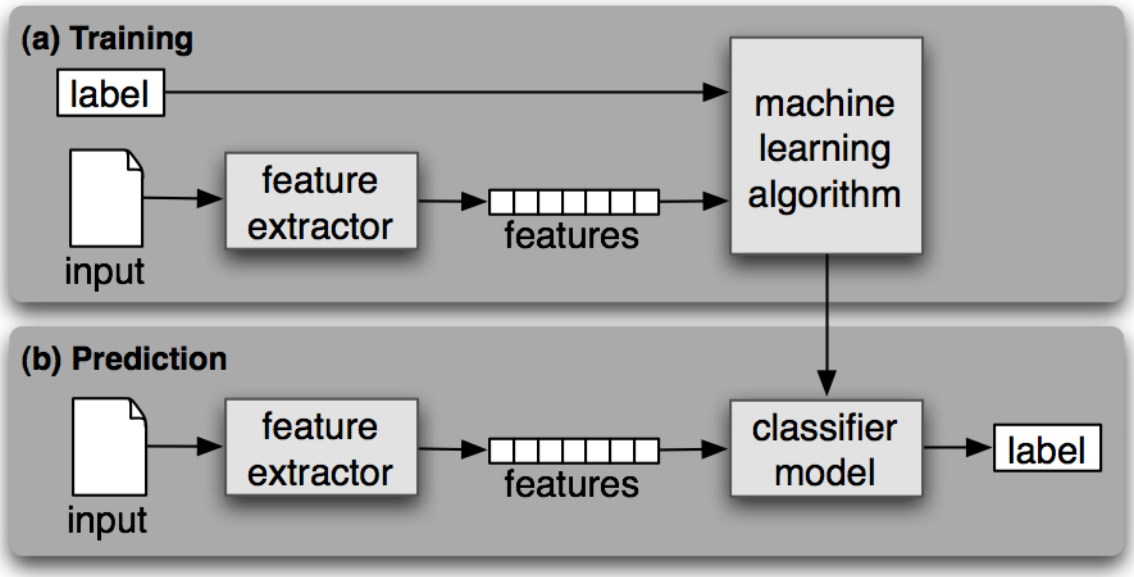


FIGURE 1.1: Supervised Classification.[2]

In figure 1.1, during training (a), a feature extractor is used to convert each input value to a feature set. These feature sets, which capture the basic information about each input that should be used to classify it, are discussed in the next section. Pairs of feature sets and labels are fed into the machine learning algorithm to generate a model. During prediction (b), the same feature extractor is used to convert unseen inputs to feature

sets. These feature sets are then fed into the model, which generates predicted labels.

1.4 Natural Language Processing

NLP is a branch of artificial intelligence which deals with the ability for computers to understand and generate human language. Earlier, NLP utilised hard coded techniques in order to perform lexical analysis, however now, most NLP is performed using statistical Machine Learning techniques, that automatically learn from a corpus of data.

NLP can be utilised to carry out various tasks in lexical analysis like Part of Speech (POS) tagging, Named Entity Recognition and Topical Modelling. Since our user profiles contain lexical data like profile description, our problem can be considered to be an NLP classification task.

Twitter provides application programmers with APIs to handle the Twitter data in their software. These APIs use http for communicating with the twitter stream. Twitter provides basically two such APIs, the REST API and the Streaming API. A combination of both can be utilised by programmers. Twitter requires OAuth to establish authentication with the server. Each user is allocated unique OAuth credentials, for which the user must register as a developer on Twitter.

2.1 Twitter REST API

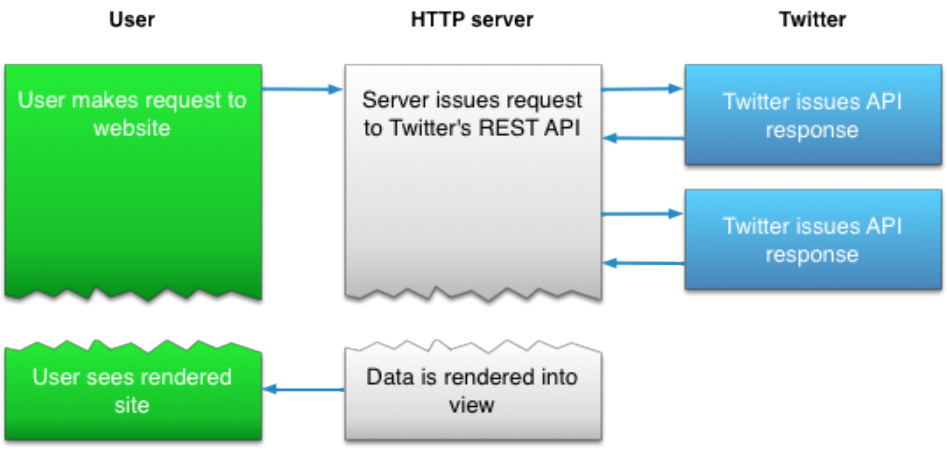


FIGURE 2.1: The Twitter REST API[1]

Representational state transfer (REST) is a method of abstraction on the internet in which a client and server can communicate with ignorance to the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.[3]

Using GET and POST methods, HTTP requests an application with authorisation can interact with Twitter, which includes, posting new tweets, updating user profiles, etc. It can also be used to download ambient metadata associated with user profiles.

Method	Resource	Description
GET	statuses/user_timeline	Returns most recent tweets of a User
GET	search/tweets	Returns relevant tweets
GET	users/show	Returns information of the user
POST	statuses/update	Updates user's tweet
POST	friendships/create	Follows the specified user

Table 2.1: Common REST services

2.2 Twitter Streaming API

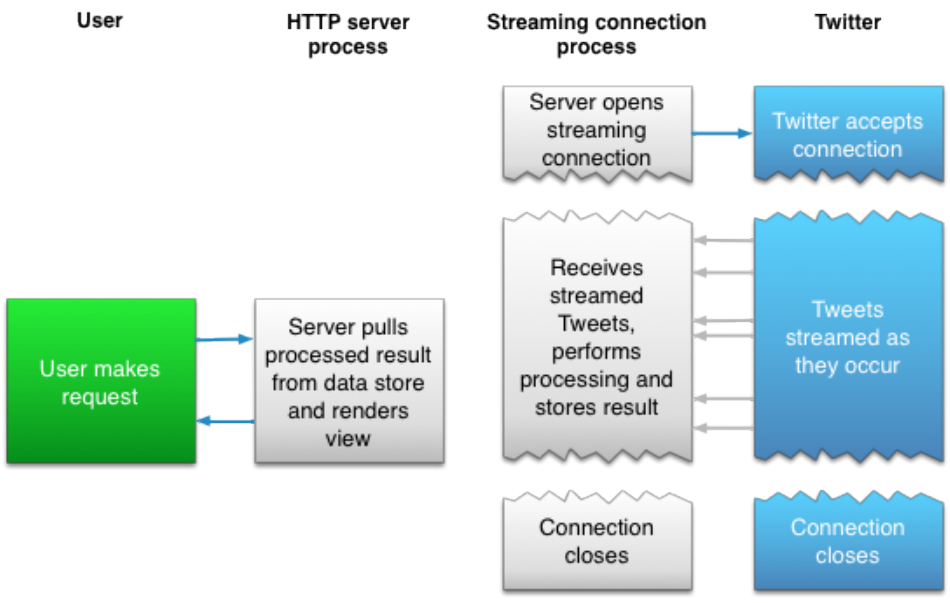


Figure 2.2: The Twitter Streaming API[1]

The streaming API allows developers to access a small subset of the entire Twitter stream. Typically for SPRITZER it is 1% of the total Twitter data. For a streaming API the code to establish a connection in response to the user request would be run in a separate thread than the code used to maintain connection to the stream, though it is more complex, it provides a realtime stream of data, with no overheads relating to the polling of the REST API.

Method	Resource	Description
GET	user	Streams tweets of a User
GET	statuses/sample	Streams sample of tweets
POST	statuses/filter	Streams tweets matching predicate

Table 2.2: Common Streaming API Endpoints

In our study, we trained 3 different classification algorithms to work on our training data, Decision Trees, Support Vector Machines and Naive Bayes.

3.1 Decision Trees

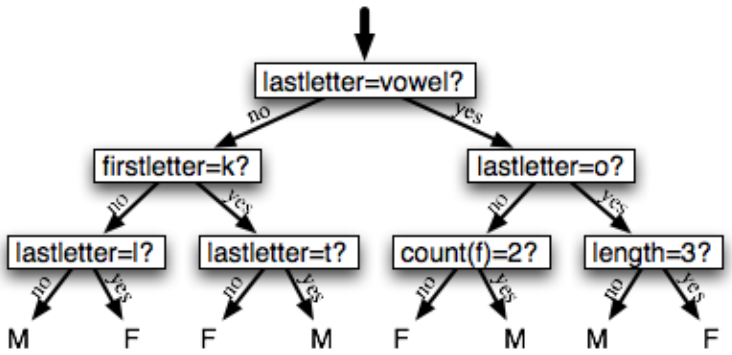


FIGURE 3.1: A Simple Decision Tree[2]

3.1.1 Introduction

A decision tree is a simple flowchart that can be used to output labels for certain features that act as input values. In case of a decision tree, the **decision nodes** are used to check feature values, and **leaf nodes**, are assigned labels. To choose the label for an input value, we begin at the

trees's **root node**. This node contains a condition that checks one of the input value's features, and selects a branch based on that feature's value. We then arrive at a new decision node, with a new condition on the input value's features. We continue following the branch selected by each node's condition, until we arrive at a leaf node, which provides a label for the input value. 3.1 shows an example decision tree model to determine the gender of a person using the features of the name as an input value.

3.1.2 Training

A decision stump is a decision tree with a single node that decides how to classify inputs based on a single feature. It contains one leaf for each possible feature value, specifying the class label that should be assigned to inputs whose features have that value. In order to build a decision stump, we must first decide which feature should be used. The simplest method is to just build a decision stump for each possible feature, and see which one achieves the highest accuracy on the training data, however since this is a complex process we use a technique called **Information Gain**

Information Gain, measures how much more organized the input values become when we divide them up using a given feature. To measure how disorganized the original set of input values are, we calculate entropy of their labels, which will be high if the input values have highly varied labels, and low if many input values all have the same label. In particular, entropy is defined as the sum of the probability of each label times the log probability of that same label.

$$H = -\sum_{1|in|labels} P(l) \times \log_2 P(l)$$

We calculate the entropy of the original set of input values' and then apply the decision stump and calculate the entropy for each of the decision stump's leaves. We then find the mean of the leaf entropy values. The information gain is equal to the original entropy minus the new, reduced entropy. The higher the information gain, the better the decision stump.

3.1.3 Limitations

Since at every decision node, the training data is split, we might reach a node wherein the training data is too small to carry out any classification. This is known as overfitting. This can be limited by stopping the growth of a tree when training data becomes too small or by pruning nodes after validation on a test set.

Decision trees force features to be checked in a particular order, since the number of features are large, many will get repeated in the subsequent branches, thus increasing the space complexity of our decision tree.

3.2 Support Vector Machines

Support Vector Machines (or SVMs) are a vector space based machine learning method where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data (possibly discounting some points as outliers or noise).[4]

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

3.3 Naive Bayes

3.3.1 Introduction

Naive Bayes improves over other machine learning classifiers like decision trees, as it parallelly considers every feature before assigning a label to any input value.

The classifier first calculates the **prior probability** over each label, by calculating the frequency of each label in the training set. The contribution of each label is then taken into consideration to arrive at an estimate for each label. The input value is then assigned the label with the highest calculated probability.

In 3.2 we find that based on the prior probability estimates, the classifier starts closer to the "Automotive" label, however, after considering the word

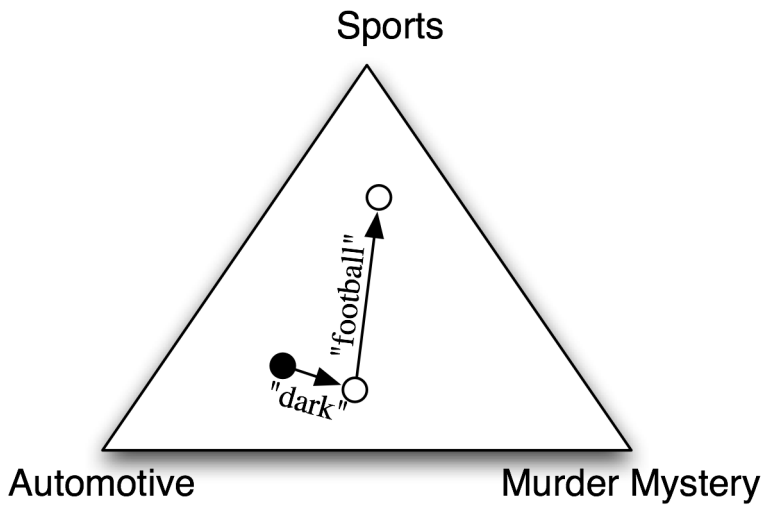


FIGURE 3.2: An Abstract illustration of Naive Bayes[2]

'dark', it moves closer to murder mysteries, further, with the consideration of the word 'football', the classifier assigns the input value the label "sports"

3.3.2 Training

We need to train a classifier that can calculate the probabability of a particualar label given a certain set of features.

$$P(label|features)$$

But, we know that P(features) is same for every choice of label, so P(features, label) is sufficient to calculate the most likely label.

$$P(features, label) = P(label) \times P(features|label)$$

Therefore, after considering multiple features for each label,

$$P(features, label) = P(label) \times \prod_{f|in|features} P(f|label)$$

We can hence calculate the probability of each input value, given a set of features using our training data.

3.3.3 Limitations

In case of Naive Bayes, since it takes into consideration each feature of the given training data, if any feature is not present with a given label, then the probability of that feature is considered to be zero, regardless of how

well the other features fit with the label. Thus, **smoothing** techniques need to be applied to our training data before applying Naive Bayes.

Another limitation of Naive Bayes is that as it considers every feature to be completely independent of the other, whereas, in te real world, every feature has a some dependence on the other.

We consider all features of Naive Bayes to be discrete. However, in case of real world data, features may not be discrete. Such continous numeric features need to be discretised, this is called **binning**. During binning, we might lose certain 'traits' or 'characteristics' of our data.

Approach 4

In order to apply machine learning algorithms on user profiles, we utilised Natural Language ToolKit, NLTK, which is available as a package for Python. We first built a feature extractor that is used to extract relevant features from the Twitter Users' Profiles. For our study, we aimed to study whether the ambient metadata, associated with the user profiles is an effective feature set for accurately classifying user profiles hence, we do **not** consider a particular users' tweet (or status updates)

We then created a feature discretiser to discretise features extracted from our user profiles, since machine learning algorithms require a discrete range of input features rather than continous numerical features.

After successfully having completed the above steps, we applied machine learning algorithms on our training set and validate results using four cross validation.

4.1 Feature Extractor

Any machine learning classifier, requires a set of 'Features'. A feature vector, is a set of features that is used to reduce the dimensinality of data, especially in case of extremely large data.

We are then able to train machine learning algorithms on patterns returned by these machine learning algorithms. For user classification we developed a feature extractor that utilised,

- The Profile Description
- The Number of Users followed by the user.
- The Number of Followers of the user
- The Ratio of the Number of Followers to the Number of Users Followed
- The total Number of Tweets

Previous studies [6] have utilsed the users' latest tweets, for user classification, however, we, inorder to determine, the ability of the 'Profile Description' field, decided to ignore the tweets. Moreover, latest tweets from a particular user may be biased towards a specific topic due to an ongoing event and hence may not be able to give an image of the aggregate interest of a user.

We analysed a corpus of 70,000 user profiles and found that about 86% of the users, provide a profile description, while others decide to keep it blank. Out of the users who do provide, profile description, the average number of characters is 59, while average number of words are about 6.

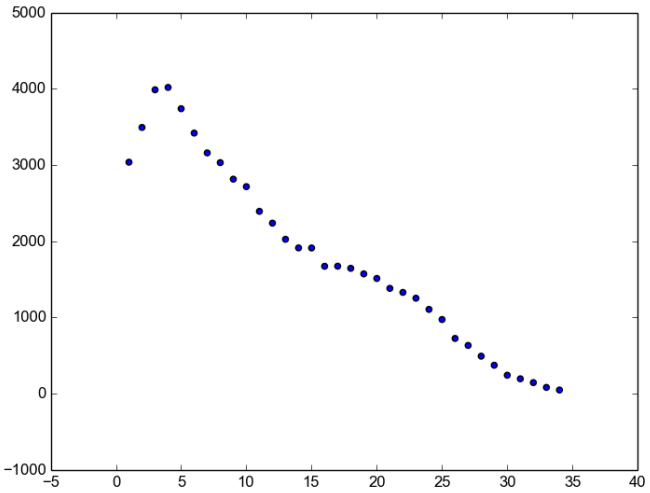


FIGURE 4.1: Number of Users vs. Number of Words

Figure 4.1 gives us an estimate on the how the number of users change with the number of words in the profile description. We find that the maximum number of users correspond to 6 words.

4.2 Binning

The features we used to classify, as given above, need to be first discretised inorder to enable our classifier to be trained on them. While the profile description containes of linguistic elements, other features are large whole numbers.

4.2.1 Profile Description

In order to discretise the linguistic elements, we first aggregate the profile information of the entire corpus of user profiles and find the top 50 most recurrent words. We then develop a feature vector which finds if any of these words is present in a given user profile description. We thus get a binary feature vector.

```
1  import nltk
2
3  for row in l:
4        w = w + row[2]
5
6  all_words = nltk.FreqDist(w)
7  word_features = all_words.keys()[:100]
8
9  def document_features(document):
10     document_words = set(document[2])
11     features = {}
12     for word in word_features:
13         features['contains(%s)' %word] = (word in document_words)
14     return features
```

4.2.2 Numeric Values

Fields such as number of tweets, number of followers, are large whole numbers. Since, these features are sparsely scattered, we decided to reduce the range of these values by a logarthmic scale. For this we used the function,

$$H(n) = [\log_{10} n], \text{ where } [\,] \text{ is greatest integer function}$$

Python allows us to easily calculate the value of the above function for any value 'n' by using the built in function, LEN()

```
1 import nltk
2
3 def document_features(document):
4         features = {}
5         if(document[3])=="":
6                 features['followers'] = 0
7         else:
8                 features['followers'] = len(document[3])
9
10        if document[5]=="":
11                features['following'] = 0
12        else:
13                features['following'] = len(document[5])
14
15        if document[6]=="":
16                features['tweets'] = 0
17        else:
18                features['tweet'] = len(document[6])
19
20        if not (document[3]=="" or document[5]==""):
21                if int(document[3])>int(document[5]):
22                        if int(document[5])!=0:
23                                features['ratio'] = len(str(int(int(
                                    document[3])/int(document[5]))))
24
25                        else:
26
27                                features['ratio'] = 10
28                else:
29
30                        if int(document[3])!=0:
31                                features['ratio'] = 0 - len(str(int(
                                    int(document[5])/int(document[3]))
                                    ))
32
33                        else:
34
35                                features['ratio'] = -10
36
37
38        return features
```

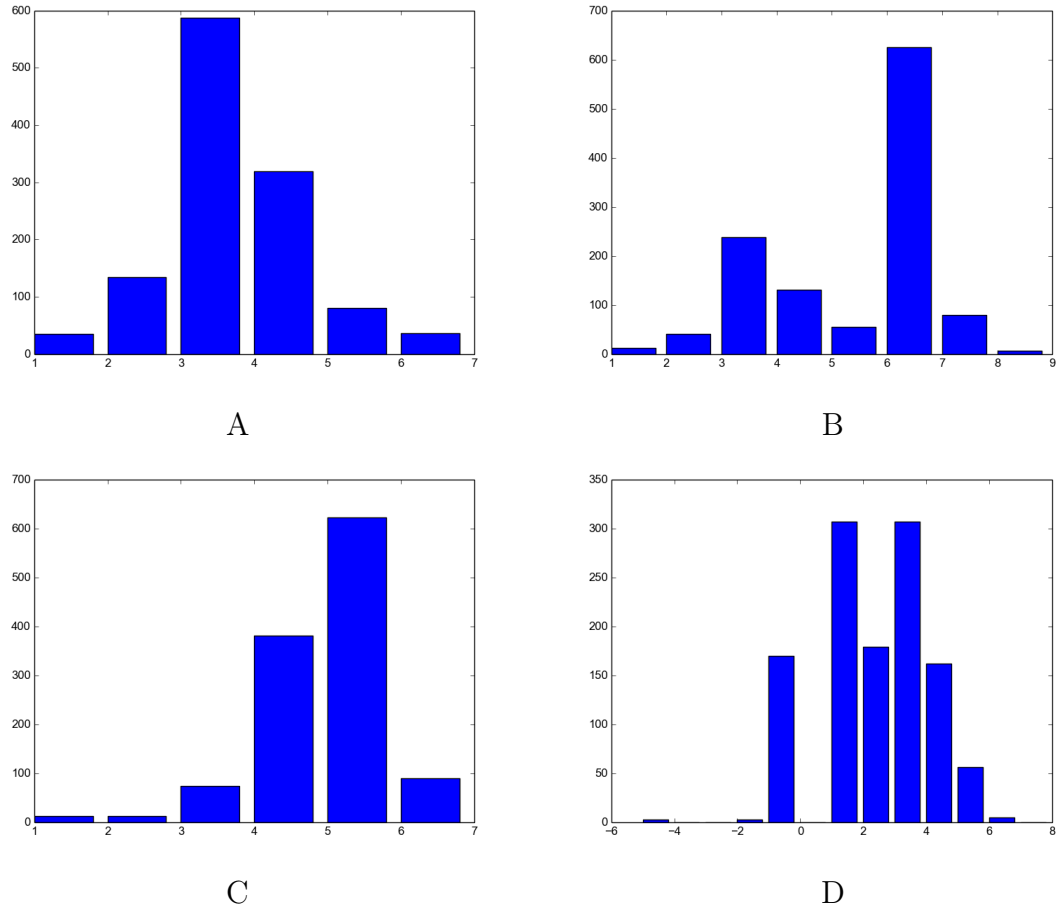


FIGURE 4.2: Features after Binning, A: Following B: Followers C: Tweets D: Ratio

4.3 Training

The Python Natural Language Toolkit or NLTK provides with functions to train classifiers, based on developed feature extractors. For our study we utilise Decision Trees, Naive Bayes' and SVMs for training the classifiers.

As NLTK does not provide libraries for SVM, we utilise the SciKit-Learn package for training classifiers based on SVMs.

Naive Bayes

```
1 import nltk
2 classifierNB = nltk.NaiveBayesClassifier.train(train_set)
```

Decision Trees

```
1 import nltk
2 classifierDT= nltk.DecisionTreeClassifier.train(train_set)
```

Support Vector Machines

```
import nltk
from sklearn.svm import SVC
classifierSVM= nltk.SklearnClassifier(SVC(),sparse=False).train(
    train_set)
```

4.4 Validation

In order to validate our results, we first train our classifier on a training dataset and then check the classification accuracy on a test dataset. This is carried out using **4-cross validation**. Results can be compared in terms of accuracy or **Confusion Matrices**.

4.4.1 4 Cross Validation

This is a technique in which the entire labelled data set is divided into four equal parts, and three parts are used for training while one is used for testing the classifier. This process is iterated over each part, and the mean of the accuracy in all 4 iterations gives us the classifier accuracy.

```
for i in range(4):
        train_set, test_set = feature_sets[:(i*(len_f))/4] +
           feature_sets[((i+1)*(len_f))/4:], feature_sets[(i*(len_f))
           /4:((i+1)*(len_f))/4]
```

4.4.2 Accuracy

Accuracy of a classifier is defined as the ratio of the number of correct labels to the total number of entries labelled.

$$A = \frac{Entries\ correctly\ labelled}{Total\ number\ of\ entries}$$

```
import nltk

acc_temp =  nltk.classify.accuracy(classifierNB, test_set)
```

4.4.3 Confusion Matrix

A Confusion Matrix (contingency table or error matrix) is a table layout that allows visualisation of a supervised learning algorithm.

In case of unbalanced data ie. datasets with an unequal number of labels, accuracy is not a reliable metric for the performance of an algorithm, and can lead to misleading results. A Confusion Matrix on the other hand, provides a more realisitic visualisation of an algorithm. Each column of the matrix represents the instances in a predicted label, while each row represents the instances in an actual label.

```
1 import nltk
2
3 tag =[]
4 gold=[]
5 for item in feature_sets:
6         tag.append(classifierNB.classify(item[0]))
7
8 for item in l:
9         gold.append(item[0])
10
11 cm =  nltk.ConfusionMatrix(gold,tag)
12 print cm.pp(show_percents=True)
```

Results 5

We trained two classifiers to classify users, using different algorithms as described in the previosu chapters. The first is to classify users as Organisations, Celebrities or Users and the second was to classify users on the basis of their interest, Music, Sports, Politics or Others. Results of the classifications have been provided.

5.1 Music, Sports, Politics & Others

5.1.1 Decision Trees

	M	P	S
M	**27.3%**	4.5%	2.6%
P	2.4%	**34.0%**	3.1%
S	2.6%	4.8%	**18.7%**

Accuracy:80%

	M	P	S
M	**29.7%**	2.4%	2.4%
P	6.2%	**30.6%**	2.6%
S	5.0%	1.7%	**19.4%**

Accuracy: 79.7%

	M	P	S
M	**26.3%**	5.3%	2.9%
P	2.9%	**33.7%**	2.9%
S	3.3%	3.6%	**19.1%**

Accuracy: 82.1%

	M	P	S
M	**30.9%**	1.4%	2.2%
P	2.9%	**32.5%**	4.1%
S	3.8%	2.2%	**20.1%**

Accuracy: 83.4%

Table 5.1: Confusion Matrix for Decision Trees

5.1.2 Naive Bayes

	M	P	S
M	**29.8%**	3.8%	4.8%
P	8.7%	**26.9%**	3.8%
S	5.8%	5.8%	10.6%

Accuracy: 67.3%

	M	P	S
M	**25.7%**	1.9%	3.8%
P	9.5%	**19.0%**	9.5%
S	11.4%	7.6%	**11.4%**

Accuracy: 56.1%

	M	P	S
M	**23.1%**	8.7%	1.9%
P	8.7%	**28.8%**	4.8%
S	4.8%	7.7%	**11.5%**

Accuracy: 76.8%

	M	P	S
M	**23.8%**	2.9%	7.6%
P	4.8%	**29.5%**	3.8%
S	10.5%	5.7%	**11.4%**

Accuracy: 76.8%

Table 5.2: Confusion Matrix for Naive Bayes

	Feature	**Label**	**Ratio**
1	contains(music)	m : p	23.4 : 1.0
2	contains(sports)	s : p	10.6 : 1.0
3	contains(news)	p : m	9.4 : 1.0
4	contains(breaking)	p : s	8.6 : 1.0
5	contains(new)	m : p	8.3 : 1.0
6	contains(it)	m : p	6.7 : 1.0
7	contains(father)	s : p	6.2 : 1.0
8	contains(singer)	m : p	5.9 : 1.0
9	contains(india)	m : p	5.9 : 1.0
10	contains(by)	p : m	5.3 : 1.0

Table 5.3: Most Significant Features in Naive Bayes

Features	**DT**	**SVM**	**NB**
numerical	50.5%	*	42.1%
numerical+ratio	53.4%	*	43.1%
numrical+ratio+description	80.1%	*	65.3%

Table 5.4: Average Results for MSP classification

5.2 Users, Organisations & Celebrities

5.2.1 Decision Trees

	U	O	C
U	**36.8**%	1.9%	0.9%
O	7.5%	**16.0%**	15.1%
C	0.9%	1.9%	**18.9%**

Accuracy: 71.7%

	U	O	C
U	**30.5**%	6.7%	.
O	3.8%	**19.0%**	6.7%
C	3.8%	11.4%	**18.1%**

Accuracy: 67.6%

	U	O	C
U	**32.4**%	4.8%	.
O	7.6%	**14.3%**	13.3%
C	.	10.5%	**17.1%**

Accuracy: 63.8%

	U	O	C
U	**32.4**%	4.8%	.
O	7.6%	**14.3%**	13.3%
C	.	10.5%	**17.1%**

Accuracy: 63.8%

Table 5.5: Confusion Matrix for Decision Trees

5.2.2 Support Vector Machines

	U	O	C
U	**36.8**%	2.8%	.
O	7.5%	**17.9%**	13.2%
C	0.9%	.	**20.8%**

Accuracy: 75.5%

	U	O	C
U	**33.3**%	2.9%	1.0%
O	5.7%	**18.1%**	5.7%
C	2.9%	11.4%	**19.0%**

Accuracy: 70.4%

	U	O	C
U	**33.3**%	2.9%	1.0%
O	10.5%	**20.0%**	4.8%
C	.	9.5%	**18.1%**

Accuracy: 71.4%

	U	O	C
U	**34.3**%	3.8%	1.0%
O	5.7%	**19.0%**	4.8%
C	.	11.4%	**20.0%**

Accuracy: 73.3%

Table 5.6: Confusion Matrix for SVM

5.2.3 Naive Bayes

	U	O	C
U	**36.8**%	.	2.8%
O	6.6%	**23.6%**	8.5%
C	0.9%	1.9%	**18.9%**

Accuracy: 79.3%

	U	O	C
U	**34.3**%	1.9%	1.0%
O	5.7%	**18.1%**	5.7%
C	4.8%	7.6%	**21.0%**

Accuracy: 73.4%

	U	O	C
U	**35.2%**	1.9%	.
O	6.7%	**24.8%**	3.8%
C	.	5.7%	**21.9%**

Accuracy: 81.9%

	U	O	C
U	**37.1**%	.	1.9%
O	1.9%	**22.9%**	4.8%
C	1.9%	2.9%	**27.6%**

Accuracy: 87.6%

Table 5.7: Confusion Matrix for Naive Bayes

	Feature	**Label**	**Ratio**
1	followers = 6	c : u	34.1 : 1.0
2	ratio = 4	c : u	26.9 : 1.0
3	ratio = 3	c : u	15.6 : 1.0
4	contains(my)	u : o	14.1 : 1.0
5	followers = 4	u : c	13.6 : 1.0
6	contains(news)	o : u	13.1 : 1.0
7	contains(official)	o : u	13.0 : 1.0
8	contains(from)	o : c	11.2 : 1.0
9	contains(i)	u : o	9.9 : 1.0
10	followers = 3	u : o	8.7 : 1.0

Table 5.8: Most Significant Features in Naive Bayes

Features	**DT**	**SVM**	**NB**
numerical	65.6%	66.7%	64.8%
numerical+ratio	64.6%	65.6%	66.3%
numrical+ratio+description	69.6%	72.9%	80.9%

Table 5.9: Average Results for OUC classification

Conclusion & Future Work 6

We find that ambient metadata of a Twitter user can be an effective feature set for Twitter user classification. Fields like profile description are provided by a majority of users as depicted in Figure 4.1. From Table 5.4 and 5.9 we find an increase both in MPS classifier and OUC classifier, when the profile description is considered as a training feature. While the OUC classifier heavily depends on numerical data like followers count (ref Table 5.8), on the other hand, the MPS classifier, depends more on linguistic elements of the profile description (Table 5.3). Both these classifiers perform well with accuracies of above 80%.

To the best of our knowledge, no previous attempt at Twitter user classification has utilised the profile description feature for training classifiers. This, combined with previous attempts to user classification can lead to development of high accuracy user classifiers. This can also be used to achieve better named entity reccognition in tweets.

Bibliography

[1] Twitter Developer API. `http://dev.twitter.com/`.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[3] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[5] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[6] Marco Pennacchiotti and Ana-Maria Popescu. A machine learning approach to twitter user classification. In Lada A. Adamic, Ricardo A. Baeza-Yates, and Scott Counts, editors, *ICWSM*. The AAAI Press, 2011.

[7] Daniel Ramage, Susan T. Dumais, and Daniel J. Liebling. Characterizing microblogs with topic models. In William W. Cohen and Samuel Gosling, editors, *ICWSM*. The AAAI Press.

[8] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[9] Wayne Xin Zhao, Jing Jiang, Jing He, Yang Song, Palakorn Achananuparp, Ee-Peng Lim, and Xiaoming Li. Topical keyphrase extraction from twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 379–388, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

Appendix (Programs & Scripts)

For the following code examples, you need to install the following python packages:

nltk numpy csv matplotlib

classifier_ouc.py

```
import nltk
import csv
import re
import string
import random
f = open("out3.csv")

reader = csv.reader(f,delimiter='\%', quotechar='|')



l = []
w = []

n_u = 0
n_c = 0
n_o = 0

for row in reader:
        if not (row[2] == "" or row[0] == "" or row[3]== "" or row
            [5]=="" or row[6]=="" or (row[1] in [item[1] for item in l
            ] ) ):

                l.append(row)
                if row[0] == 'u':
                        n_u+=1
                if row[0] == 'c':
                        n_c+=1
                if row[0] == 'o':
                        n_o+=1
```

```
29
30
31
32 print "Total no of items:", len(l), "Users:", n_u, "Celebs:", n_c, "
     Organs:", n_o
33
34
35 random.shuffle(l)
36
37 for item in l:
38         item[2] = item[2].lower()
39
40         for c in string.punctuation:
41                 item[2] = item[2].replace(c, " ")
42
43         item[2] = item[2].split()
44
45 print len(l)
46
47 for row in l:
48 #       print row[2]
49         w = w + row[2]
50
51 all_words = nltk.FreqDist(w)
52 word_features = all_words.keys()[:100]
53
54
55 #print word_features
56
57
58 def document_features(document):
59         document_words = set(document[2])
60         features = {}
61         for word in word_features:
62                 features['contains(\%s)' \%word] = (word in
                       document_words)
63
64         if(document[3])=="":
65                 features['followers'] = 0
66         else:
67                 features['followers'] = len(document[3])
68
69         if document[5]=="":
70                 features['following'] = 0
71         else:
72                 features['following'] = len(document[5])
```

```
73
74          if document[6]=="":
75                  features['tweets'] = 0
76          else:
77                  features['tweet'] = len(document[6])
78
79
80          if not (document[3]=="" or document[5]==""):
81                  if int(document[3])>int(document[5]):
82                          if int(document[5])!=0:
83                                  features['ratio'] = len(str(int(int(
                                        document[3])/int(document[5]))))
84
85                          else:
86
87                                  features['ratio'] = 10
88                  else:
89
90                          if int(document[3])!=0:
91                                  features['ratio'] = 0 - len(str(int(
                                        int(document[5])/int(document[3]))
                                        ))
92
93                          else:
94
95                                  features['ratio'] = -10
96
97
98          return features
99
100
101 feature_sets = []
102
103 for item in l:
104         feature_sets.append([document_features(item),item[0]])
105
106 #print feature_sets[1]
107
108 num_folds = 1
109
110 subset_size = len(feature_sets)/num_folds
111
112 acc = 0
113 acc_temp = 0
114
115 len_f = len(feature_sets)
```

```
116
117
118 for i in range(4):
119
120
121         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
122
123 #       train_set, test_set =feature_sets[:(len_f)/2] ,feature_sets[(
      len_f)/2:]
124
125
126         print """"""==="
127
128         classifierNB = nltk.NaiveBayesClassifier.train(train_set)
129
130
131
132         acc_temp =  nltk.classify.accuracy(classifierNB, test_set)
133
134         print acc_temp
135
136
137
138
139         tag =[]
140         gold=[]
141         for item in test_set:
142
143                 tag.append(classifierNB.classify(item[0]))
144
145
146
147         for item in l[ (i*(len_f))/4:((i+1)*(len_f))/4 ]:
148
149
150                 gold.append(item[0])
151
152
153
154
155         cm =  nltk.ConfusionMatrix(gold,tag)
156
157         print cm.pp(show_percents=True)
158
```

```
159         classifierNB.show_most_informative_features(20)
160 
161         print "==================================="
162 
163         acc += acc_temp
164 
165 
166 
167 print "After 4-Cross Validation Naive Bayes: ", (acc/4)*100
168 
169 from sklearn.naive_bayes import BernoulliNB
170 from sklearn.svm import SVC
171 
172 print "==================================="
173 
174 print "RESULTS FOR SVM:"
175 
176 
177 acc_temp = 0
178 
179 for i in range(4):
180 
181         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
182 
183 
184 
185         classifierSVM= nltk.SklearnClassifier(SVC(),sparse=False).
               train(train_set)
186 
187         tag =[]
188         gold=[]
189         for item in test_set:
190 
191                 tag.append(classifierSVM.classify(item[0]))
192 
193 
194 
195         for item in l[(i*(len_f))/4:((i+1)*(len_f))/4]:
196 
197 
198                 gold.append(item[0])
199 
200 
201 
```

```
202
203         cm =  nltk.ConfusionMatrix(gold,tag)
204
205         print cm.pp(show_percents=True)
206
207
208
209
210         print nltk.classify.accuracy(classifierSVM, test_set)
211
212         acc_temp += nltk.classify.accuracy(classifierSVM, test_set)
213
214         print "==================================="
215         classifierNB.show_most_informative_features(20)
216
217
218 print "After four cross validation in SVM:",  (acc_temp/4)
219
220 acc_temp = 0
221
222
223
224 print "RESULTS FOR DT:"
225
226 for i in range(4):
227
228         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
229
230
231         classifierDT= nltk.DecisionTreeClassifier.train(train_set)
232
233         #print classifierDT.pseudocode(depth=4)
234
235         tag =[]
236         gold=[]
237         for item in test_set:
238
239                 tag.append(classifierDT.classify(item[0]))
240
241
242
243         for item in l[(i*(len_f))/4:((i+1)*(len_f))/4]:
244
245                 gold.append(item[0])
```

```
246
247
248         cm =  nltk.ConfusionMatrix(gold,tag)
249
250         print cm.pp(show_percents=True)
251
252         print nltk.classify.accuracy(classifierDT, test_set)
253
254         acc_temp += nltk.classify.accuracy(classifierDT, test_set)
255
256   #       classifierDT.show_most_informative_features(20)
257
258
259
260 print "After 4 cross Validation in DT:",( acc_temp/4)
```

classifier_mps.py

```
1 import nltk
2 import csv
3 import re
4 import string
5 import random
6 f = open("out3merged.csv")
7
8 reader = csv.reader(f,delimiter='\%', quotechar='|')
9
10
11
12 l = []
13 w = []
14
15 n_m = 0
16
17 for row in reader:
18         if not (row[2] == "" or row[0] == "" or row[3]== "" or row
               [5]=="" or row[6]=="" or (row[1] in [item[1] for item in l
               ] ) ):
19
20                 if row[0] == 'n':
21                         if row[7]=='n':
22                                 row[0] = row[8]
23                         else:
24                                 row[0] = row[7]
25                 if not row[0]=='n':
26                         l.append(row)
```

```
27                 if row[0] == 'm':
28                         n_m+=1
29
30
31 print "Total no of items:", len(l), "Music", n_m
32
33 random.shuffle(l)
34
35 for item in l:
36         item[2] = item[2].lower()
37
38         for c in string.punctuation:
39                 item[2] = item[2].replace(c, " ")
40
41         item[2] = item[2].split()
42
43 print len(l)
44
45 for row in l:
46 #       print row[2]
47         w = w + row[2]
48
49 all_words = nltk.FreqDist(w)
50 word_features = all_words.keys()[:100]
51
52
53 #print word_features
54
55
56 def document_features(document):
57         document_words = set(document[2])
58         features = {}
59         for word in word_features:
60                 features['contains(\%s)' \%word] = (word in
                       document_words)
61
62         if(document[3])=="":
63                 features['followers'] = 0
64         else:
65                 features['followers'] = len(document[3])
66
67         if document[5]=="":
68                 features['following'] = 0
69         else:
70                 features['following'] = len(document[5])
71
```

```
72          if document[6]=="":
73                  features['tweets'] = 0
74          else:
75                  features['tweet'] = len(document[6])
76
77
78          if not (document[3]=="" or document[5]==""):
79                  if int(document[3])>int(document[5]):
80                          if int(document[5])!=0:
81                                  features['ratio'] = len(str(int(int(
                                       document[3])/int(document[5]))))
82
83                          else:
84
85                                  features['ratio'] = 10
86                  else:
87
88                          if int(document[3])!=0:
89                                  features['ratio'] = 0 - len(str(int(
                                       int(document[5])/int(document[3]))
                                       ))
90
91                          else:
92
93                                  features['ratio'] = -10
94
95          return features
96
97
98  feature_sets = []
99
100 for item in l:
101         feature_sets.append([document_features(item),item[0]])
102
103 #print feature_sets[1]
104
105 num_folds = 1
106
107 subset_size = len(feature_sets)/num_folds
108
109 acc = 0
110 acc_temp = 0
111
112 len_f = len(feature_sets)
113
114
```

```
115 for i in range(4):
116
117
118         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
119
120 #       train_set, test_set =feature_sets[:(len_f)/2] ,feature_sets[(
      len_f)/2:]
121
122
123         print "==================================="
124
125         classifierNB = nltk.NaiveBayesClassifier.train(train_set)
126
127
128
129         acc_temp =  nltk.classify.accuracy(classifierNB, test_set)
130
131         print acc_temp
132
133
134
135
136         tag =[]
137         gold=[]
138         for item in test_set:
139
140                 tag.append(classifierNB.classify(item[0]))
141
142
143
144         for item in l[(i*(len_f))/4:((i+1)*(len_f))/4]:
145
146
147                 gold.append(item[0])
148
149
150
151
152         cm =  nltk.ConfusionMatrix(gold,tag)
153
154         print cm.pp(show_percents=True)
155
156         classifierNB.show_most_informative_features(20)
157
```

```
158         print "====================================="
159 
160         acc += acc_temp
161 
162 
163 
164 print "After 4-Cross Validation Naive Bayes: ", (acc/4)*100
165 
166 from sklearn.naive_bayes import BernoulliNB
167 from sklearn.svm import SVC
168 
169 print "====================================="
170 print "RESULTS FOR SVM:"
171 
172 
173 acc_temp = 0
174 
175 for i in range(4):
176 
177         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
178 
179 
180 
181         classifierSVM= nltk.SklearnClassifier(SVC(),sparse=False).
               train(train_set)
182 
183         tag =[]
184         gold=[]
185         for item in test_set:
186 
187                 tag.append(classifierSVM.classify(item[0]))
188 
189 
190 
191         for item in l[(i*(len_f))/4:((i+1)*(len_f))/4]:
192 
193 
194                 gold.append(item[0])
195 
196 
197 
198 
199         cm =  nltk.ConfusionMatrix(gold,tag)
200 
```

```
201         print cm.pp(show_percents=True)
202
203
204
205
206         print nltk.classify.accuracy(classifierSVM, test_set)
207
208         acc_temp += nltk.classify.accuracy(classifierSVM, test_set)
209
210         print "==================================="
211
212
213 print "After four cross validation in SVM:",  (acc_temp/4)
214
215 acc_temp = 0
216
217 print "RESULTS FOR DT:"
218
219 for i in range(4):
220
221         train_set, test_set =feature_sets[:(i*(len_f))/4] +
               feature_sets[((i+1)*(len_f))/4:] ,feature_sets[(i*(len_f))
               /4:((i+1)*(len_f))/4]
222
223
224         classifierDT= nltk.DecisionTreeClassifier.train(train_set)
225
226         #print classifierDT.pseudocode(depth=4)
227
228         tag =[]
229         gold=[]
230         for item in feature_sets:
231
232                 tag.append(classifierDT.classify(item[0]))
233
234
235
236         for item in l[:]:
237
238                 gold.append(item[0])
239
240
241         cm =  nltk.ConfusionMatrix(gold,tag)
242
243         print cm.pp(show_percents=True)
244
```

```
245         print nltk.classify.accuracy(classifierDT, test_set)
246
247         acc_temp += nltk.classify.accuracy(classifierDT, test_set)
248
249
250 print "After 4 cross Validation in DT:",( acc_temp/4)
```

graphing.py

```
 1 import nltk
 2 import csv
 3 import re
 4 import string
 5 import random
 6 import matplotlib.pyplot as plt
 7
 8 f = open("temp.csv")
 9 reader = csv.reader(f,delimiter='%', quotechar='|')
10
11 n = 0
12 c = 0
13 w = [0 for i in range(50)]
14 x = [i for i in range(50)]
15 n_n = 0
16
17 for row in reader:
18         if not (len(row[1])>160):
19 #                print(row[1])
20                 if (row[1]!="(null)"):
21                         c += len(row[1])
22                         for p in string.punctuation:
23                                 row[1] = row[1].replace(p, " ")
24                         w[len(row[1].split())]+=1
25                 else:
26                         n_n+=1
27
28
29                 n+=1
30
31 plt.scatter(x[1:35],w[1:35])
32 plt.show()
33 f.close()
34
35 f = open("out3plusrefined")
36 reader = csv.reader(f,delimiter='%', quotechar='|')
37
```

```
38 w = [(i-5) for i in range(15)]
39 x = [0 for i in range(15)]
40
41 for document in reader:
42         if not (document[3]=="" or document[5]==""):
43                 if int(document[3])>int(document[5]):
44                         if int(document[5])!=0:
45                                 x[5+ len(str(int(int(document[3])/int
                                        (document[5]))))] +=1
46
47                         else:
48
49                                 x[10]+=1
50                 else:
51
52                         if int(document[3])!=0:
53                                 x[ 5 - len(str(int(int(document[5])/
                                        int(document[3]))))]+=1
54
55                         else:
56
57                                 x[0]+=1
58
59 plt.bar(w,x)
60 plt.show()
61
62
63
64 f.close()
65 f = open("out3plusrefined")
66 reader = csv.reader(f,delimiter='%', quotechar='|')
67
68 w = [i for i in range(15)]
69 x = [0 for i in range(15)]
70
71
72
73 for row in reader:
74         print row[5]
75         x[len(row[5])]+=1
76
77 plt.bar(w[1:],x[1:])
78 plt.show()
79 f.close()
80 f = open("out3plusrefined")
81 reader = csv.reader(f,delimiter='%', quotechar='|')
```

```
82
83 w = [i for i in range(15)]
84 x = [0 for i in range(15)]
85
86 for row in reader:
87
88         print row[6]
89         x[len(row[6])]+=1
90
91 plt.bar(w[1:],x[1:])
92 plt.show()
93 f.close()
```