# Automatic text classification of sports blog data

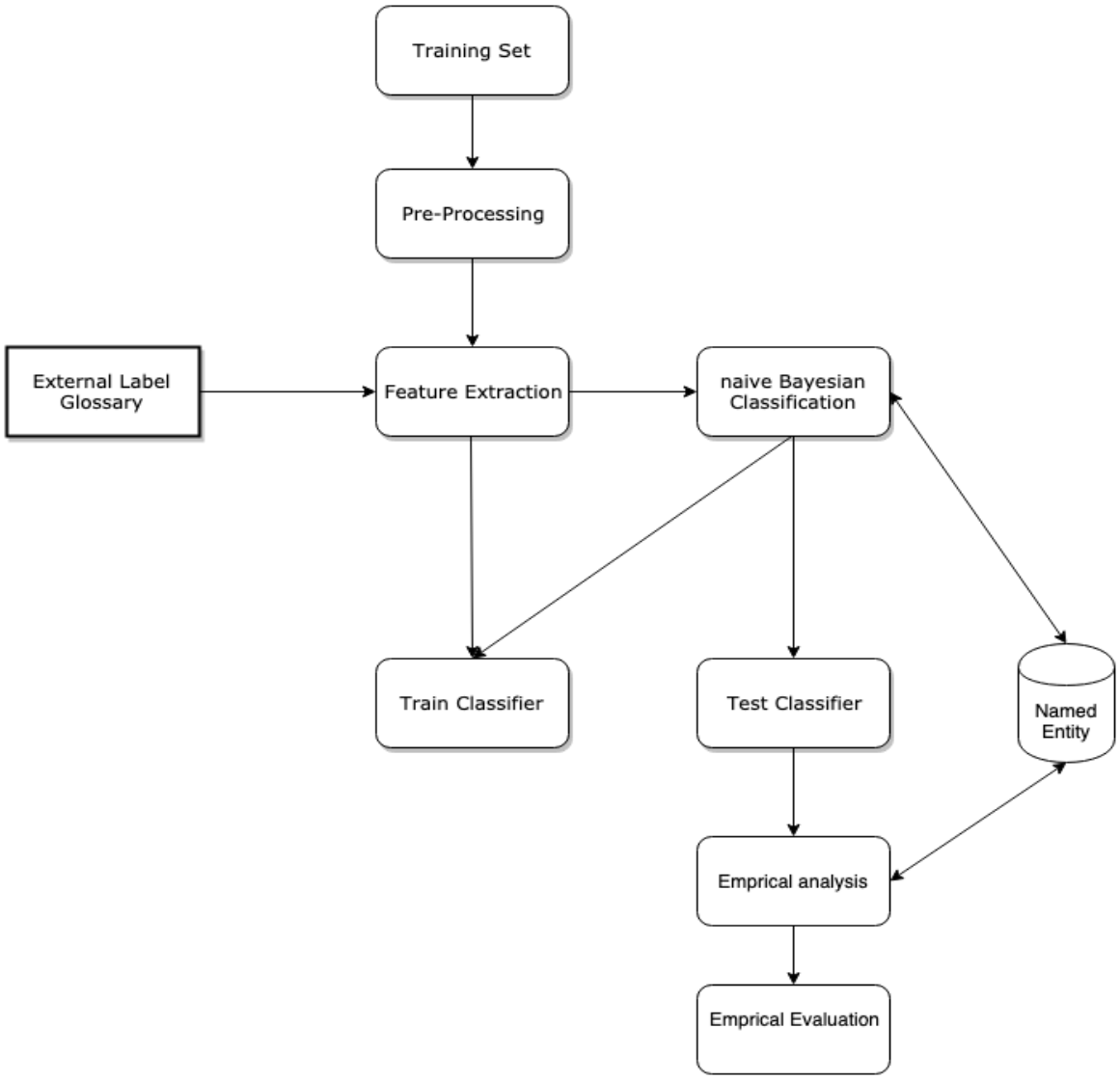
Ben Stewart S 2016103513
Siva Kailash S 2016103593

Github https://github.com/SivaK18/Semantic-Analysis

Automatic Text Classification is a semi-supervised machine learning task that automatically assigns a given text document to a set of predefined categories based on the features extracted from its textual content. We will be attempting to automatically classify the textual entries made by bloggers on various topics, to the appropriate category by following steps like preprocessing, feature extraction and naïve **Bayesian classification**. **Empirical evaluation** must result in a very high accuracy. In addition to classifying the textual entries of blogs, it is proposed that the extracted features themselves be further classified under more meaningful heads which results in generation of a semantic resource that lends greater understanding to the classification task. This semantic resource can be used for data mining requirements that arise in the future.

We have taken the paper that classifies text of sports blog data that works only with sports data. It is limited to very small tagset. Here our aim is to study general blog data and classify them. The process we are going to do is given in the following Block diagram.

## BLOCK DIAGRAM :



## DATASETS :

The training datasets are taken from https://www.kaggle.com/rtatman/blog-authorship-corpus.

The Corpus consists of the collected posts of 19,320 bloggers gathered from blogger.com in August 2004. The corpus incorporates a total of 681,288 posts and over 140 million words - or approximately 35 posts and 7250 words per person.

All bloggers included in the corpus fall into one of three age groups:

- 8240 "10s" blogs (ages 13-17),
- 8086 "20s" blogs(ages 23-27)
- 2994 "30s" blogs (ages 33-47).

For each age group there are an equal number of male and female bloggers.

## Modules :

**Pre-processing :** This involves collecting data from the dataset, loading the data, removing duplicate tuples, white spaces and plotting the length of each post (blog post).

**Feature Extraction :** This involves figuring out how we can actually train the data. It contains finding the count of bloggers and post and the age of the users and count of unique labels (i.e the unique topics of the post) that is feeded into the kernel.

**Adding External label Glossaries :** After finding the labels, we need to feed those into our kernel as a basis for classification.

**Naive Bayesian Classification** : This involves loading the data, and training the model on the split dataset so that it can be tested on untrained data. This takes the labels that are generated in the previous step as a basis / an input for classification.(This involves stemming as well).

**Train Classifier :** This involves training our split data based on naive Bayesian classification. As a result of this we get a functional model that is ready to take the test .

**Test Classifier :** Once the model is ready we need to test the model on untrained data so that we could get the efficiency of the model. On training the data we got a accuracy of 35 %, which is 12x greater than random prediction.

**Named Entities :** Now we have a model that produces 35% which is a bit low (considering the huge size of the dataset). However this can be reduced by reducing the no of dimensions of the dataset or by converting the linear dimensions to multi dimensional non-linear dimensions (i.e. instead of going to a label of 1*40 we can go to a label consisting of dimensions like x*y (where x * y = 40)). This may increase the complexity of the model but also increases the accuracy of it as well. Once label is ready we need to feed it again to the model, train and test it.We reduced No.of the dimensions from 40 to 3 . Now we got a accuracy of about 54% which is a significant improvement from the previous one.We couldn't achieve more than this because of the factor that the blog contains Native slangs (other language texts ) written in English which is difficult to interpret and classify.

**Empirical analysis :** Once the models are ready we need to find which model is highly accurate so that we stop the training just in time to avoid overfitting problem.

**Empirical Evaluation :** This is the final step / module where the model is evaluated for its correctness and accuracy (a final checking of the model before conclusion).A custom text is given as an input and is check it for correctness.

**Evaluation results** description of each module is given above **:** (Code)

1. Pre processing :
df = pd.read_csv("../Semantic-Analysis/blogtext.csv")
df.drop_duplicates(subset="text",inplace=True)
df.text.str.len().describe()
Screenshot : Preprocessing

2. Feature Extraction :
df.text.str.len().plot()
df.age.value_counts()
list = df.topic.unique().tolist()
Screenshot : Features

3.External Glossary :
dictOfWords = { list[i] : i for i in range(0, len(list) ) }
dictOfWords
Screenshot : Labels

4.Naive Bayesian Classifier :
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

```
from sklearn import decomposition, ensemble
data = pd.read_csv(
    "../Semantic-Analysis/blogtext.csv",
    usecols=['topic', 'text']   # Only load the three columns
specified.
)
print(data.shape)
data.head()
```
Screenshot : classifier

5.Stemming :
```
import nltk
df['text'] = df['text'].apply(nltk.word_tokenize)
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
```
Screenshot : Stemming

6.train and testing :
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(counts, df['topic'],
test_size=0.33, random_state=0)
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB().fit(X_train, y_train)
predicted = model.predict(X_test)
print(np.mean(predicted == y_test))
```
Screenshot : Confmat1

7. Named Entity Generation :
(Reduction of dimension)
```
data1['topic'] = data1['topic'].replace(dictOfWords1)
data1.head()
import nltk
data1['text'] = data1['text'].apply(nltk.word_tokenize)
```

data1.head()
(the labelling part is done manually).
Screenshot : NE

8. Confusion Matrix :

```
from sklearn.metrics import confusion_matrix
import pylab as pl
cm = confusion_matrix(y_test, predicted)
pl.matshow(cm)
pl.title('Confusion matrix of the classifier')
pl.colorbar()
pl.show()
```

Screenshot : Conf matrix

9. Test against custom data:

```
predicted = model.predict('custom text')
```

Predicted ;

Screenshots can be found here

**Conclusion :**

The dimensions are reduced and the model got about 54% accuracy (considering the size of the data as well as the native slang of the users ). The model is verified for classifying custom text.

Some of the applications of the system are

- Content management of blogs.
- Analyse the users and the blogs.
- Targeted advertisements based on the content.