Aalto University

PROJECT REPORT

ELEC-E5550 - Statistical Natural Language Processing

Fake News Detection

Author:
Rohan Chauhan

April 25, 2018

Aalto University
School of Science

1 Introduction

Fake news has gained notable attention after US 2016 presidential election. The misinformation spread by fake news divides people into polarized groups on social media. The polarized groups tend to engage in content within their filter bubble and dismiss the opinions of other group. Such conditions can harm democracy and lead to echo chambers which reinforce political extremism.

The spread of misinformation on social media by humans and bots has been studied by several researchers [2] [15]. Although, fake news is more about character of its circulation than content, fake news are biased towards their readership rather than presenting neutral facts. Several research papers have tried to exploit the writing style of fake news article for their detection [12]. Several techniques such as veracity verification [17], stance detection [14], handcrafted features [1] and a combination of semantic, lexical and syntactic information has been used [11]. In this project, we will use word embeddings to predict a given text is "real news" or "fake news".

1.1 Word Embeddings

This project focuses on using word level embeddings to classify and understand fake news. Word embeddings are a language modeling technique where word or phrases are mapped to vectors of real numbers. Firth was the first to propose that a word can be characterized by the company it keeps [3]. Word embeddings can be generated by different techniques such as neural network [10], probabilistic models [5] and other different techniques . There are also a lot of software for training word embeddings such as Word2Vec [10], Glove, fastText, Gensim and deeplearning4j [16]. Word Embeddings have been shown to boost the performance in several NLP tasks.

1.2 Interpretability

In this project, we also consider the notion of interpretability which has gained a lot of importance with General Data Protection Regulation (GDPR). European Union's new GDPR creates a right to explanation whereby a user can ask for an explanation of an algorithmic decision that was made about them [6].Interpretablity methods either explain the model i.e. model transparency or explain the prediction i.e. post-hoc interpretability [8]. In this project, we will use LIME which stands for local interpretable model-agnostic explanations [13]. LIME explains the prediction of any black box model by approximating it locally using a simpler model.

The rest of the report is organized as follows: Section 2 describes the methods used in the project. Section 3 discusses the dataset and experiments followed by results in Section 4. Finally, conclusions are drawn in Section 5.

2 Methods

In this section, we briefly describe the libraries that we used to carry out our experiment.

2.1 Scikit-learn

Scikit-learn is a widely used machine learning library in python that needs no introduction. We used feature extraction and machine learning models from Scikit learn for text classification. We also used other important libraries which are not part of scikit-learn but as important as it such as **pandas**, **numpy** and **matplotlib**.

2.2 fastText

fastText is a library for efficient learning of word representations and sentence classification. fastText was designed due to drawbacks in word2vec. Firstly, word2vec doesn't offer sentence representations. Taking the average of pre-trained word vectors doesn't represent sentence very well. Secondly, word2vec doesn't exploit morphology which means that words with same root don't share the same parameters. For example, disaster and disastrous.

FastText is unified library for text representation and text classification. FastText has a common core of word2vec but uses different pooling strategies i.e. different inputs and outputs. The speed of fastText is significantly faster than state of art DNN models.

2.3 LIME

We used the python version of LIME available on Github [9].The first step of lime is to permute data and then calculate distance between permutations and original observations. The second step is to make predictions on new data using complex model and then pick m features describing complex model outcome from permuted data. The third step is to fit a simple model to permuted data with m features and take the similarity scores as weights.In the end, these feature weights from simple model make explanation for the complex model's local behavior [7].

3 Datasets and Experiments

In this section, first, we describe the dataset and then discuss about the experiment for the project.

3.1 Dataset

We use the fake news dataset collected by George McIntire [4]. This fake news dataset contains 6,335 news articles out of which 3171 are 'real news' and 3164 are 'fake news' . There are 6,335 rows and 4 columns: serial no, title, text and label.

However, fake news itself can be also classified into further categories such as satire, comedy, clickbait, conspiracy, rumor, junk science and extreme bias. But, this dataset can be used only for binary classification. It is also difficult to assure that all labels in this dataset are correct.

3.2 Experiment

Two types of experiments were carried out. The first experiment was using tfidf representation with different machine learning models and also checking their interpretability with LIME. The second experiment was using word2vec with fastText and checking the accuracy with different values of hyper parameters.

3.3 Experiment 1

We tried out different machine learning models for text classification purpose. We compare them on the basis of their accuracy, training time and testing time. We were also able to apply LIME to some of the models which offered prediction probabilities for the classes. These models used term frequency–inverse document frequency (TFIDF). The idea behind TFIDF representation is that word that occur more frequently are less likely to be helpful in distinguishing two classes/documents. Below, we will give a brief explanation about all the machine learning models used and also highlight the ones where LIME could be applied.

1. **Models (with LIME applied)**

    (a) **Naive Bayes classifiers**: In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers "based on applying Bayes' theorem with strong (naive) independence assumptions between the features.Bernoulli Bayes and Multinomial Bayes were the two variants used.

    (b) **K-Nearest Neighbors**: The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression

    (c) **Random Forest**: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

2. **Models (without LIME applied)**

    (a) **Nearest Centroid**: Each class is represented by its centroid, with test samples classified to the class with the nearest centroid.

    (b) **Stochastic Gradient Descent**: Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

    (c) **Support Vector Machines**: A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

(d) **Passive Aggressive**: Passive Aggressive Algorithms are a family of online learning algorithms (for both classification and regression) proposed by Crammer at al.

(e) **Perceptron**: In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers (functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not).

(f) **Ridge regression**: Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value.

3.4 Experiment 2

In this experiment, we used the fastText library for classification. We also tried using different n-grams and learning rate and see how accuracy changes with them.

Finally, we compare the accuracy of all models in Experiment 1 and Experiment 2.

4 Results

The results have been broken down into three sub-sections. In the first section, we discuss the performance of experiment 2 and reflect on the results. In the second section, we compare all the models based on their accuracy followed by discussion on interpretability in third section.

4.1 Word2Vec and Hyper parameters

In Figure 1, we see that the accuracy decreases as n-grams is increased from 1 to 5. Trigrams are said to be state of the art as they are best in modeling short-range dependencies and long-range dependencies. However, with further increase their effectiveness decreases.

In Figure 2, we see that learning rate doesn't play major role in changing accuracy of the model. However, learning rate can play bigger role when the corpus is large enough.

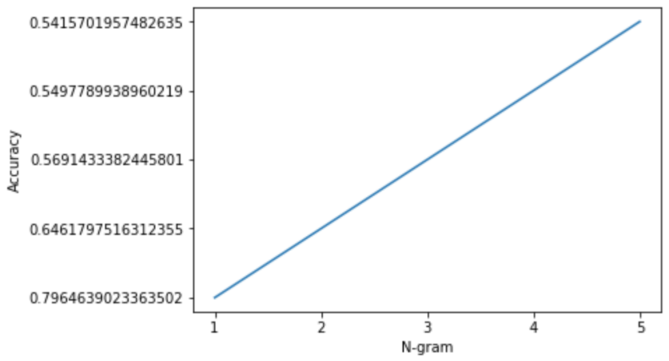


Figure 1: Accuracy vs N-grams

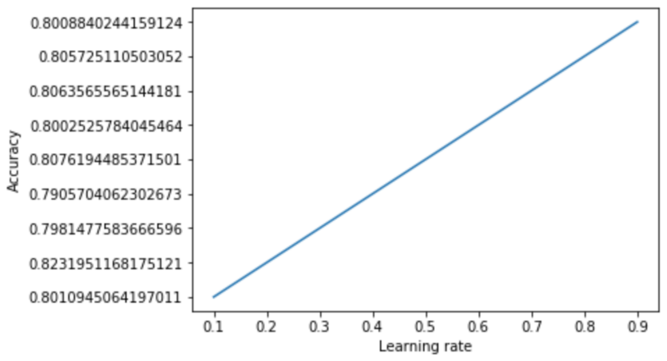


Figure 2: Accuracy vs Learning Rate

4.2 Accuracy of Different Models

The models 'Multinomial Naive Bayes', 'Bernoulli Naive Bayes', 'Random Forest', 'Ridge Classfier', 'Perceptron', 'Passive-Agressive Classfier', 'K-Nearest Neighbours', 'Linear SVC with l1 penalty', 'Linear SVC with l2 penalty', 'SGD with l1 penalty', 'SGD with l2 penalty', 'SGD with Elastic-Net penalty', 'Nearest Centroid', 'FastText' and there accuracy are given as '0.9255050505050505', '0.8743686868686869', '0.9255050505050505', '0.9577020202020202', '0.9343434343434344', '0.9583333333333334', '0.5366161616161617', '0.9501262626262627', '0.9558080808080808', '0.94760101010101', '0.9532828282828283', '0.9583333333333334', '0.8453282828282829', '0.7945695643022521' respectively.

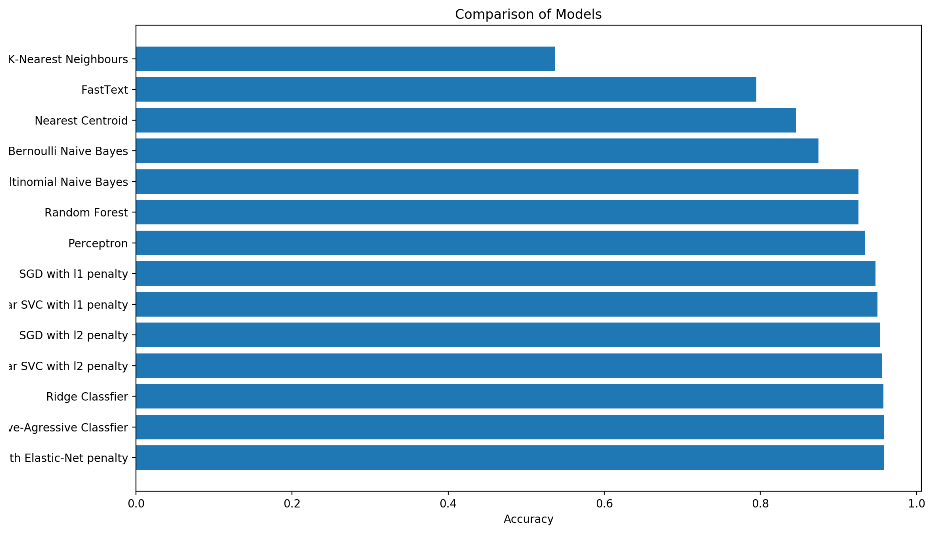
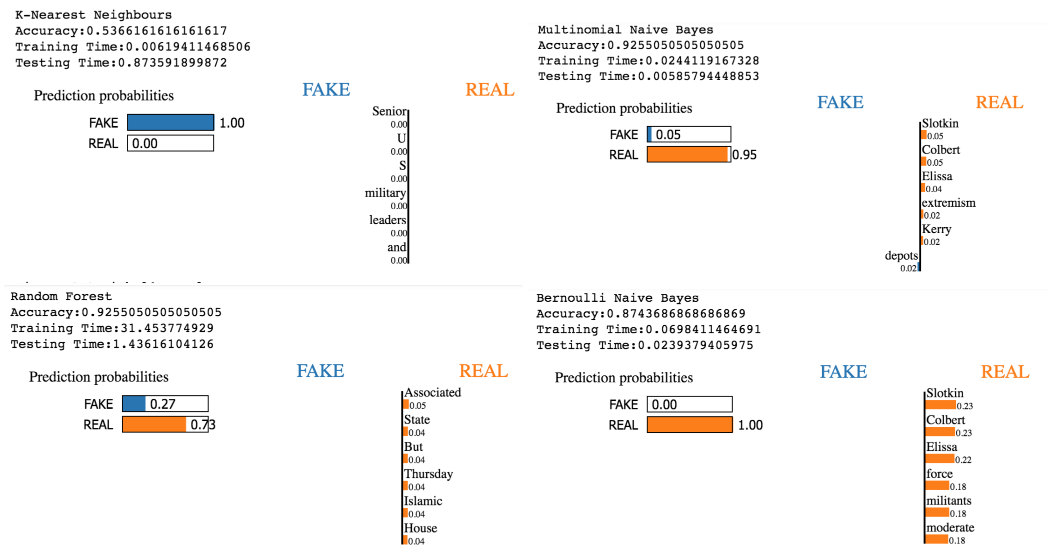


Figure 3: Comparison of different models

The word2vec FastText performs poorly compared to tfidf models. This can happen due to several reasons such as lack of pre-processing or short corpora. We see that random forest, support vector machines and stochastic gradient descent are the top performers.

4.3 Interpretability

We see here that different machine learning models offer different explanation for their prediction in Figure 3. We can verify the model which is working correctly based on its explanation.



```
K-Nearest Neighbours
Accuracy:0.5366161616161617
Training Time:0.00619411468506
Testing Time:0.873591899872
```

```
Multinomial Naive Bayes
Accuracy:0.9255050505050505
Training Time:0.0244119167328
Testing Time:0.00585794448853
```

```
Random Forest
Accuracy:0.9255050505050505
Training Time:31.453774929
Testing Time:1.43616104126
```

```
Bernoulli Naive Bayes
Accuracy:0.8743686868686869
Training Time:0.0698411464691
Testing Time:0.0239379405975
```

Figure 4: Interpretability of different models based on LIME

We can see that k-nearest neighbors is performing most poorly and cannot account for its prediction because it has never encountered those words or either those words were removed due to the tfidf setting.

5 Conclusion

It was interesting to see how different representation can have effect on models accuracy. Interpretability also varies from one model to another and also on the choice of hyper parameters.

The future work for this experiment would be try the experiment with pre-processing. We can also try to use grid-search for hyper parameter optimization for each model.

6 Acknowledgment and Contribution

I would like to thank my family and friends for their support. I would also like to thank Professor Mikko for the course and teaching team for their feedback on project abstract and support during exercise sessions.

As I did the project alone so everything done during this project is my contribution. I would also like to thanks **Wikipedia** and **Scikit documentation** which I haven't been able to include in my references as they have been heavily quoted in this report.

References

[1] Bhatt, G., Sharma, A., Sharma, S., Nagpal, A., Raman, B., and Mittal, A. On the benefit of combining neural, statistical and external features for fake news identification. *arXiv preprint arXiv:1712.03935* (2017).

[2] Bounegru, L., Gray, J., Venturini, T., and Mauri, M. A field guide to'fake news' and other information disorders.

[3] Firth, J. R. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis* (1957).

[4] GeorgeMcIntire. Georgemcintire/fake_real_news_dataset. `https://github.com/GeorgeMcIntire/fake_real_news_dataset`, 2017. [Online; accessed 21-April-2018].

[5] Globerson, A., Chechik, G., Pereira, F., and Tishby, N. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research 8*, Oct (2007), 2265–2295.

[6] Goodman, B., and Flaxman, S. European union regulations on algorithmic decision-making and a" right to explanation". *arXiv preprint arXiv:1606.08813* (2016).

[7] Kasia Kulma. Interpretable machine learning using lime framework - kasia kulma (phd), data scientist, aviva. `https://www.youtube.com/watch?v=CY3t11vuuOM`, 2017. [Online; accessed 21-April-2018].

[8] Lipton, Z. C. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490* (2016).

[9] Marco Tulio Correia Ribeiro. marcotcr/lime. `https://github.com/marcotcr/lime`, 2017. [Online; accessed 21-April-2018].

[10] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[11] Pérez-Rosas, V., Kleinberg, B., Lefevre, A., and Mihalcea, R. Automatic detection of fake news. *arXiv preprint arXiv:1708.07104* (2017).

[12] Potthast, M., Kiesel, J., Reinartz, K., Bevendorff, J., and Stein, B. A stylometric inquiry into hyperpartisan and fake news. *arXiv preprint arXiv:1702.05638* (2017).

[13] Ribeiro, M. T., Singh, S., and Guestrin, C. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), ACM, pp. 1135–1144.

[14] Riedel, B., Augenstein, I., Spithourakis, G. P., and Riedel, S. A simple but tough-to-beat baseline for the fake news challenge stance detection task. *arXiv preprint arXiv:1707.03264* (2017).

[15] Shao, C., Hui, P.-M., Wang, L., Jiang, X., Flammini, A., Menczer, F., and Ciampaglia, G. L. Anatomy of an online misinformation network. *arXiv preprint arXiv:1801.06122* (2018).

[16] Wikipedia contributors. Word embedding — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Word_embedding&oldid=836044700`, 2018. [Online; accessed 20-April-2018].

[17] Zubiaga, A. Learning class-specific word representations for early detection of hoaxes in social media. *arXiv preprint arXiv:1801.07311* (2018).

7 Appendix

7.1 Experiment 1

```
import pandas as pd
import numpy as np
from time import time
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import chi2
from sklearn import metrics

#LIME imports
from lime import lime_text
from lime.lime_text import LimeTextExplainer

#Classifier imports
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
import sklearn.ensemble


#Loading Dataset and Dividing into Test set and Training set
df = pd.read_csv('fake_or_real_news.csv', index_col='UID', names=['UID'
df['label'] = df.label.map({'FAKE':0, 'REAL':1})
X = df['text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=
#Creating Representation using Transform and Fit
vectorizer = sklearn.feature_extraction.text.TfidfVectorizer(lowercase=
train_vectors = vectorizer.fit_transform(X_train)
test_vectors = vectorizer.transform(X_test)

#LIME Setup
class_names = ['FAKE', 'REAL']
explainer = LimeTextExplainer(class_names=class_names)
#Selecting index for example to be explained by LIME
index = 0

#Different classifiers with Explaination
classifier_name=[]
training_time=[]
testing_time=[]
accuracy=[]

#Multinomial Naive Bayes
clf = MultinomialNB(alpha=.01)
print("Multinomial Naive Bayes")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
#For Explaination
clf_pipe = make_pipeline(vectorizer, clf)
exp = explainer.explain_instance(X_test.iloc[index], clf_pipe.predict_pr
exp.show_in_notebook(text=False)
classifier_name.append("Multinomial Naive Bayes")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))
```

```
#Bernoulli Naive Bayes
clf = BernoulliNB(alpha=.01)
print("Bernoulli Naive Bayes")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
#For Explaination
clf_pipe = make_pipeline(vectorizer, clf)
exp = explainer.explain_instance(X_test.iloc[index], clf_pipe.predict_pr
exp.show_in_notebook(text=False)
classifier_name.append("Bernoulli Naive Bayes")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#Random Forest
clf = sklearn.ensemble.RandomForestClassifier(n_estimators=500)
print("Random Forest")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
#For Explaination
clf_pipe = make_pipeline(vectorizer, clf)
```

```
exp = explainer.explain_instance(X_test.iloc[index], clf_pipe.predict_pr
exp.show_in_notebook(text=False)
classifier_name.append("Random Forest")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))


#Ridge Classfier No LIME
clf = RidgeClassifier(tol=1e-2, solver="lsqr")
print("Ridge Classfier")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("Ridge Classfier")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))



#Perceptron No LIME
clf = Perceptron(n_iter=50)
print("Perceptron")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
```

```
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("Perceptron")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))



#Passive-Agressive Classfier No LIME
clf = PassiveAggressiveClassifier(n_iter=50)
print("Passive-Agressive Classfier")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("Passive-Agressive Classfier")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))



#K-Nearest Neighbours
clf = KNeighborsClassifier(n_neighbors=5)
print("K-Nearest Neighbours")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
```

```
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
#For Explaination
clf_pipe = make_pipeline(vectorizer, clf)
exp = explainer.explain_instance(X_test.iloc[index], clf_pipe.predict_pr
exp.show_in_notebook(text=False)
classifier_name.append("K-Nearest Neighbours")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#Linear SVC with l1 penalty No LIME
clf = LinearSVC(penalty="l1", dual=False, tol=1e-3)
print("Linear SVC with l1 penalty")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("Linear SVC with l1 penalty")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))


#Linear SVC with l2 penalty No LIME
clf = LinearSVC(penalty="l2", dual=False, tol=1e-3)
print("Linear SVC with l2 penalty")
start = time()
clf.fit(train_vectors, y_train)
```

```
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("Linear SVC with l2 penalty")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))


#SGD with l1 penalty No LIME
clf = SGDClassifier(alpha=.0001, n_iter=50, penalty="l1")
print("SGD with l1 penalty")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("SGD with l1 penalty")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#SGD with l2 penalty No LIME
clf = SGDClassifier(alpha=.0001, n_iter=50, penalty="l2")
print("SGD with l2 penalty")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
```

```
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
classifier_name.append("SGD with l2 penalty")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#SGD with Elastic-Net penalty No LIME
clf = SGDClassifier(alpha=.0001, n_iter=50, penalty="elasticnet")
print("SGD with Elastic-Net penalty")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training Time:" + str(train_time))
print("Testing Time:" + str(test_time))
#For Explaination
classifier_name.append("SGD with Elastic-Net penalty")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#Nearest Centroid No LIME
clf = NearestCentroid()
print("Nearest Centroid")
start = time()
clf.fit(train_vectors, y_train)
train_time = time() - start
start = time()
```

```
pred = clf.predict(test_vectors)
test_time = time() - start
score = metrics.accuracy_score(y_test, pred)
print("Accuracy:" + str(score))
print("Training␣Time:" + str(train_time))
print("Testing␣Time:" + str(test_time))
#For Explaination
classifier_name.append("Nearest␣Centroid")
training_time.append(str(train_time))
testing_time.append(str(test_time))
accuracy.append(str(score))

#For Plotting
print(classifier_name)
print(training_time)
print(testing_time)
print(accuracy)
```

7.2 Experiment 2

```
import fasttext
import matplotlib.pyplot as plt
import datetime
import numpy as np
from time import time
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import pandas as pd
import os
from sklearn.cross_validation import train_test_split


#Loading Dataset and Dividing into Test set and Training set
df = pd.read_csv('fake_or_real_news.csv', index_col='UID', names=['UID'
#df['label'] = df.label.map({'FAKE':0, 'REAL':1})
X = df['text']
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=
name = 'dataset'

#Writing the training set
with open(str(name)+'_train.txt','w') as file:
    for x in range(0,len(X_train)):
        text=str('__label__'+str(y_train.iloc[x])+'␣'+str(X_train.iloc[
        file.write(text)

#Writing the test set
with open(str(name)+'_test.txt','w') as file:
    for x in range(0,len(X_test)):
        text=str('__label__'+str(y_test.iloc[x])+'␣'+str(X_test.iloc[x])
        file.write(text)

start = time()
classifier = fasttext.supervised('dataset_train.txt', 'model')
train_time = time() - start
start = time()
result = classifier.test('dataset_test.txt')
test_time = time() - start
print 'P@1:', result.precision
print 'R@1:', result.recall
print 'Number␣of␣examples:', result.nexamples
print('Training␣Time:␣'+str(train_time))
print('Testing␣Time:␣'+str(test_time))

predictions = classifier.predict(X_train,1)
print(confusion_matrix(y_train, predictions, labels=["REAL", "FAKE"]))
print("Accuracy␣=␣"+str(accuracy_score(y_train, predictions)))


ngrams_accuracy=[]
ngram=np.arange(1,6)
for x in ngram:
    classifier = fasttext.supervised('dataset_train.txt', 'model', word
    predictions = classifier.predict(X_train,1)
    ngrams_accuracy.append(str(accuracy_score(y_train, predictions)))
plt.plot(ngram, ngrams_accuracy)
plt.xticks(ngram)
plt.xlabel('N-gram')
plt.ylabel('Accuracy')
plt.show()

lr_accuracy=[]
lr=np.arange(0.1,1,0.1)
for x in lr:
    classifier = fasttext.supervised('dataset_train.txt', 'model', lr_up
    predictions = classifier.predict(X_train,1)
    lr_accuracy.append(str(accuracy_score(y_train, predictions)))
plt.plot(lr, lr_accuracy)
plt.xlabel('Learning rate')
plt.ylabel('Accuracy')
plt.show()
```