Exploring Fake News with Logistic Regression and LSA

University of Washington, Seattle, Autumn Quarter 2017, Aman Arya

December 6, 2017

1 Introduction

Prior to October 2016, the term "Fake News" remained a relative unknown to both the American public and major American media corporations. Today however, one would themselves hard-pressed to find a political news article that has not been lambasted by one side or another as "Fake News".

The New York Times defines "Fake News" as a "made up story with the intention to deceive". While there are countless news stories out there currently that support one side or another, these would not be considered as "Fake News" unless it was proved that the author of the article wrote with the intention to deceive their readers. The ambiguity and difficulty of the situation is of course, to define what exactly it means to "deceive" and to prove the intentionality of the writer to "deceive".

During the divisive American 2016 election, a large number of fake news articles had spread across many popular social media platforms such as Facebook and Twitter. This may go without saying, but the spread of deceptive ideas upon popular social media platforms has the ability to sway public opinion greatly on many issues, especially the most important humanitarian, political, international and environmental issues we face today as a civilization and greatly effect our responses to these issues and how we deal with them in the coming months and years. To combat the spread of "Fake News", Facebook and other social media platforms have begun initiatives to use AI in order to classify which news articles are "Fake" and which are "Real".

In this paper, I attempt to tackle this problem with the use of Logistic Regression and using LSA with a K-Nearest Neighbors Classification. I discuss the performance of my models, and possible reasons Logistic Regression outperformed LSA with KNN as well as possible next steps from here.

2 Formulation

My formulation of this problem is identical on George McIntire's formulation in his blog post "On Building a "Fake News" Classification Model" in ODSC.

The acquisition of the "Fake News" proved largely simple for McIntire as Kaggle had recently published a collection of 13,000 news articles that had been published during the 2016 election cycle and been labeled as "Fake" by the popular Chrome Extension BS Detector.

Acquiring "Real News" was more difficult. To accomplish this McIntire scraped news from "All Sides", a news website which hosts articles from across the political spectrum.

In this problem I mostly focused on the actual text of the article as I believed that it would be extremely difficult for any classifier to detect "Fake" or "Real" based simply on the headline. Looking at the full text also gives the classifiers more features to work with, which I believed would greatly improve my performance.

Along with the full text of the Article, I have a label given with "FAKE" or "REAL" for the article.

3 Techniques Used

With my solutions I decided to go with Logistic Regression and LSA with K-means clustering. I began with Logistic Regression as I believed it to be a strong baseline in terms of basic classification techniques and this problem has been thoroughly explored using Naive Bayes classifiers quite

extensively already. I used LSA with KNN neighbors, as I wanted to see if using LSA would improve the overall feature space against the tf-idf features. I also believed that by calculating the singular values of various tf-idf matrices would prove effective in finding certain keywords indicated "Fake" or "Real", which might also give us clues of what kind of articles we should be consuming.

3.1 Logistic Regression

Logistic Regression is a linear model that is used to predict a discrete variable's class. In this case that discrete variable is the text document itself and the class would be either real or fake.

With Logistic Regression the overall idea is to calculate a probability that the document belongs to one class (y) or another given a list of features x. To calculate this probability we use the sigmoid function as shown below.

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + exp(-\theta^T x} \tag{1}$$

$$P(y = 0|x) = 1 - h_{theta}(x) \tag{2}$$

θ is a vector of weights that is learned by the Logistic Regression algorithm that is used to effectively make the classification. The learning is done usually through gradient descent on a cost function as shown below.

Logistic Regression functions with on the basis of independent variables, if features given are not independent, the predictions may be faulty. Logistic Regression has a computational advantage on LSA, however it can be prone to over-fitting if the gradient descent is handled incorrectly with a poor learning rate or the learning rate does not change over time.

3.2 LSA with KNN

LSA (Latent Semantic Analysis) is a process which was designed to answer the problem of finding keywords in documents for the purposes of document similarity and search. The difficulty in the problem lies not in simply finding words that occur in documents over and over again, but in understanding the underlying concepts behind the words. LSA solves this problem through a tf-idf representation of the documents and computing the singular values of the this tf-idf matrix.

The process of computing the LSA begins with coming up with a tf-idf matrix for the corpus. In this matrix the rows are the words that are appearing in at least 3 of the documents and removing terms that occur in more than 60% of the documents. The columns are the document titles and the values are the tf-idf values for each word in that document.

Once this matrix is computed the SVD is performed on the matrix. This dimensionality reduction assists with finding the strongest possible relationships within the data.

These singular values from the SVD make a "concept" space in which it may possible to perform KNN to classify various articles by looking at its neigbors within the concept space.

Drawbacks of the LSA include the fact that is a bag-of-words model that may not include information about context and word order. SVD is also computationally expensive.

K-nearest neighbors is a classification technique that on top of the concept space created by the LSA. KNN works by representing the components of the LSA as a vector and computing a pre-defined distance between that vector and the other vectors in the input. The pre-defined distance metric used here was cosine similarity. Once we find the "neighbors" in the concept space, depending on their classification and if k or more neighbors are of that classification, we compute the current vector's classification.

K-nearest neighbors often has a high computational cost with the number of metrics we need to compute between that and the neighbors and it requires knowledge of what k should be. However, it is not prone to over-fitting and is robust to noise in the dataset.

4 Training and Testing Data Used

As stated above in the Formulation section, the data used here is identical to George McIntire's data collected and used in his blog post "On Building a “Fake News” Classification Model" in ODSC.

Data is aggregated from many different news sources including New York Times, WSJ, Bloomberg, NPR, and the Guardian. Labeling of the data was done through both the BS Detector Chrome extension and using data from "All Sides". Overall the dataset included 10558 articles with a 50/50 split on both Real and Fake news.

With the Training and Testing Data aggregated I used sklearn's TfidfVectorizer to find those text values which appeared in 3 or more of the documents but not in 60% of the documents and those values that are not stop words (the, he, it, she, am, I, etc.).

It should be noted that this is a Bag-of-words formulation of the problem and is a poor formulation due the fact that it does not take into account ideas like Word Ordering and Context, which is extremely important for this problem. Further formulations and solutions to this problem should take this into account.

5 Experiments

For all of the the experiments we simplified the text bodies through TfidfVectorizer as provided by sklearn. I selected only those text values that appeared in 3 or more documents and did not occur in more than 60% of the documents. Stop words were removed as well.

We split the data on an 80/20 split, with 80% of the data going to the train side and 20% going to the test side. Additionally, the data went through a 5-fold cross-validation to check for over-fitting.

Logistic Regression was done with an l2-norm. The SVD performed in the LSA was done with 100 components and a Normalizer which normalized each sample to the unit norm and the normalization was done with an l2-norm.

KNN was done using 5 neighbors. A further exploration of this parameter is necessary.

Grid-search was not performed in this case due to a lack of time.

6 Results

For results I looked at the overall F1-scores, accuracies, avg 5-fold Cross Validation and the confusion matrices.

6.1 Logistic Regression Results

F1 score: 92.5%
Accuracy: 92.25%
Avg 5-fold Cross Validation: 92.3137%

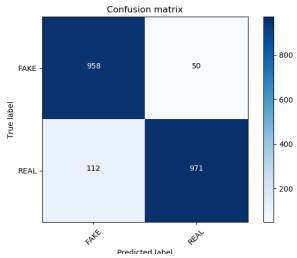

Figure 1: Logistic Regression Confusion Matrix

6.2 LSA+KNN results

F1 score: 89.09%
Accuracy: 89.1%
Avg 5-fold Cross Validation: 89.2681%

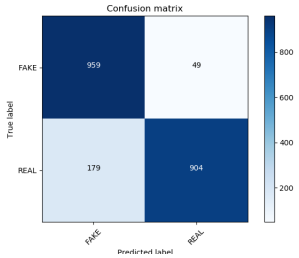

Figure 2: LSA+KNN Confusion Matrix

7 Discussion

Many of problems that arise with Fake News classification come from the overall formulation of what "Fake" and "Real" news actually is. These terms are largely very subjective and open to interpretation from multiple sides. As such the formulation of such a problem is difficult, as with many so-called "Wicked Problems". Additionally, by simply trying to find certain keywords to identify a document is problematic as mere keywords do not identify "Fake" or "Real" news. Even within all the articles of "Fake" and "Real", these "keywords" will most likely be words such as "Trump" or "Clinton", which do not truly help us with identifying the classification.

That being said, both classifiers worked surprisingly well for this problem. Logistic Regression, being a classifier with only tf-idf features given as training input managed to outperform the KNN classifier with LSA features. We can observe why this might be the case by looking at some of the components and their weights that were given by LSA.

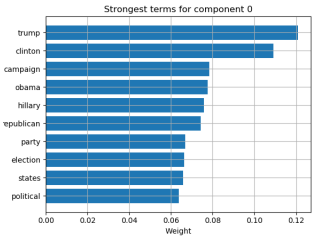

Figure 3: Keywords

As you can see in Figure 3, most of the features that were considered highly were simply words such as "Trump", "Clinton", "campaign", "Obama" which are not very interesting features that were calculated here. It seems that LSA did not make much difference in the end and the features that were given to both the Logistic Regression and KNN were largely similar. Therefore, between the two classifiers Logistic Regression out-competed KNN, due to it being more advanced and able to learn the feature weights much better.

A few next steps that can be done at this point are perform Grid-search on the current two classifiers to compute the best possible F1-scores and accuracy as well as try out different kinds of classifiers for both the tf-idf and LSA feature sets. Possible classifiers that can be used from here are XG-Boost or Random-Forests.

I believe that finding keywords through a Bag-of-words model as was performed here is really not a great idea for this purpose. Finding out how words are ordered and the context of how the text is formulated overall is far more indicative of the label of "Fake" and "Real" news. The problem is not so simple that we can look for a few keywords and create a classifier based on these keywords. We do not do this in real life when evaluating news articles and to have our classifiers use this poor method will always ensure suboptimal performance.

Such models that can take both word-orders and context into account may include deep neural networks and other such techniques.

8 Personal Retrospective

From doing this assignment I learned more about the problem of "Fake News" within our society and ways that we can possible combat it. While this remains a very important topic from both a political, social, and technological perspective I believe a more concrete formulation of the problem is necessary, where there are clearly defined guidelines of what is "Fake" and what is "Real". It is only then that we can better understand and tackle the problem as a society. There is much more work that needs to be done here and we are only at the beginning.

While this initial attempts are not formulated correctly, they are important in guiding ways that we can formulate this complex problem correctly. To begin solving such a problem, we have to start somewhere and these basic explorations of such datasets are an excellent way to get started.

There is also a discussion to be made here on the risks of false positives and false negatives. What effect would it have if our classifier marked "Fake" news as "Real" and "Real" news as "Fake"? While this paper is not a suitable place for an in-depth discussion like this, I believe that this discussion needs to happen among those solving this problem. However, I think we can all agree that certain ideas can be dangerous especially if put into the hands of those that lead us or the majority of the population. Many historical catastrophes have occurred due to misinformation and the data science community, due to the nature of the work we do, has a moral obligation to combat such misinformation.

9 References

Alvarez, Miguel M. "How Can Machine Learning and AI Help Solving the Fake News Problem?" The Practical Academic, 21 Nov. 2017, miguelmalvarez.com/2017/03/23/how-can-machine-learning-and-ai-help-solving-the-fake-news-problem/.

McCormick, Chris. "Latent Semantic Analysis (LSA) for Text Classification Tutorial." Mcormickl, mccormickml.com/2016/03/25/lsa-for-text-classification-tutorial/.

McIntire, George. "On Building a 'Fake News' Classification Model *Update." Open Data Science, ODSC, opendatascience.com/blog/how-to-build-a-fake-news-classification-model/.

Genes, Yunus. "Detecting Fake News With NLP – Yunus Genes – Medium." Medium, Medium, 23 May 2017, medium.com/@Genyunus/detecting-fake-news-with-nlp-c893ec31dee8.

Jarmal, Katharine. "Detecting Fake News with Scikit-Learn." DataCamp, DataCamp, www.datacamp.com/community/tutorials/scikit-learn-fake-news.

"Logistic Regression." Unsupervised Feature Learning and Deep Learning Tutorial, Stanford, ufldl.stanford.edu/tutorial/supervised/LogisticRegression/.

10 Appendix A

```
# coding: utf-8

# In[2]:

"""
Aman Arya
CSE 415
1535134
A7: Part B
Fake News Classification via Naive Bayes and LSA
"""
import itertools
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from sklearn.cluster import KMeans
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier


# In[3]:

# Import ‘fake_or_real_news.csv‘
df = pd.read_csv
("https://s3.amazonaws.com/assets.datacamp.com/blog_assets /fake_ or_real_news.csv")


# In[6]:

df = df.set_index("Unnamed: 0")
df.head()


# In[7]:

y = df.label
df.drop("label", axis=1)
X_train, X_test, y_train, y_test =
train_test_split(df['text'], y, test_size=0.33, random_state=53)


# In[8]:

# Initialize the ‘tfidf_vectorizer‘
tfidf_vectorizer = TfidfVectorizer
(stop_words='english', max_df=0.6, min_df=3, use_idf=True, norm='l2',
sublinear_tf=True, ngram_range=(1, 2))

# Fit and transform the training data
tfidf_train = tfidf_vectorizer.fit_transform(X_train)

# Transform the test set
tfidf_test = tfidf_vectorizer.transform(X_test)


# In[9]:

def plot_confusion_matrix(cm, classes, name,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
```

```
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.clf()
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(name)


# In[10]:
clf = LogisticRegression(penalty="l2")

clf.fit(tfidf_train, y_train)
pred = clf.predict(tfidf_test)
print "Logistic Regression F1 and Accuracy Scores : \n"
print ( "F1 score {:.4}%".format( metrics.f1_score(y_test, pred, average='macro')*100 ) )
print ( "Accuracy score {:.4}%".format(metrics.accuracy_score(y_test, pred)*100) )
cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'], name="LogReg.png")

# In[11]:
scores = cross_val_score(clf, tfidf_train, y_train, cv=5)
print(scores)


# In[12]:

lsa = TruncatedSVD(n_components=100, n_iter=10)


# In[13]:

X = lsa.fit_transform(tfidf_train)
t = lsa.transform(tfidf_test)
print("LSA fit and transformed")
```

```
# In[14]:
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski')
knn.fit(X, y_train)

# In[15]:
pred_km = knn.predict(t)

# In[15]
print "LSA + KNN F1 and Accuracy Scores : \n"
print ( "F1 score {:.4}%".format( metrics.f1_score(y_test, pred_km, average='macro')*100 ) )
print ( "Accuracy score {:.4}%".format(metrics.accuracy_score(y_test, pred_km)*100) )
cm = metrics.confusion_matrix(y_test, pred_km, labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'], name="LSA.png")

scores = cross_val_score(knn, X, y_train, cv=5)
print(scores)

#In[16]
feat_names = tfidf_vectorizer.get_feature_names()

for compNum in range(0, 2):
    plt.clf()
    comp = lsa.components_[compNum]

    indeces = np.argsort(comp).tolist()

    indeces.reverse()

    terms = [feat_names[weightIndex] for weightIndex in indeces[0:10]]
    weights = [comp[weightIndex] for weightIndex in indeces[0:10]]

    terms.reverse()
    weights.reverse()
    positions = np.arange(10) + .5

    plt.figure(compNum)
    plt.barh(positions, weights, align='center')
    plt.yticks(positions, terms)
    plt.xlabel('Weight')
    plt.title('Strongest terms for component %d' % (compNum))
    plt.grid(True)
    plt.show()
```