Analysis of New York Times Opinion and News Articles

Mikerah Quintyne-Collins

September 16, 2017

0.1 Abstract

In the dawn of fake news, a lot of organizations are trying to filter fake opinions and trying to classify certain articles are legitimate news items. Moreover, they do not want the appearance of censorship. Thus, they also would like to classify opinion articles. The goal of this project is to try to determine which machine learning algorithms are up to the task and to cluster articles in order to find insight into the problem of classifying articles as opinion or news. The results show that with logistic regression, we can obtain an F1 score of 0.97

0.1.1 Introduction

Previous work had been done in this area. The work of Yu and Hatzivassiloglou focused on classifying facts from opinion at both the document and sentence level. Their worked showed that using a naive bayes classifier, they can classify documents correctly with 97% accuracy and sentences with 91% accuracy.

One of the goals of this project is to determine which words distinguish between news articles and opinion articles using unsupervised machine learning techniques.

0.1.2 Description of the Data

The data consists of 10 738 randomly selected New York Times articles from the year 2016. The categories of articles in the dataset are opinion, U.S and World. The opinion category consists of articles that are found on the opinion section of the New York Times website. These include OP-EDs, letters to the editor and editorials. The U.S category consists of news stories based mainly in the United States. The World category consists of news articles based mainly outside of the U.S but their articles that focus on the U.S as well.

0.1.3 Analysis

Data Collection First, the data collection process consisted of writing a spider using the web scraping framework Scrapy. Scrapy is a free and popular web scraping framework for python. It can be installed using pip or conda. You can also obtain it from the official website. The data were stored in a MongoDB database. The code for the web scraper is available in the github repository for this project.

Data Cleaning After collecting the data, we separated the opinion, U.S and World articles into a separate MongoDB collection. Due to the URL structure of articles on the New York Times website, certain articles were in a separate category despite belonging to another. Those articles where sorted in to the appropriate categories. Next, we remove any articles that were not in the English language as these articles do not consists of the majority of the data, they would bias the results. Afterwards, we remove phrases found at the beginning or end of all New York Times articles such as "We're interested in your feedback on this page". Finally, we export the data into a json file for later analysis. The code that was use for data cleaning is available on this project's github repository.

Exploratory Data Analysis Next, we use unsupervised machine learning techniques to gain some insight into our dataset. These techniques are K-means, Latent Dirichlet Allocation(LDA), Non-negative Matrix Factorization (NMF) and Latent Semantic Analysis. We use both a bag-of-words model and tf-idf model. We use K-means for document clustering and both NMF and LDA for topic modeling.

```
In [1]: from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorize
        from sklearn.cluster import KMeans, MiniBatchKMeans
        from sklearn.decomposition import NMF, LatentDirichletAllocation
        from misc import create_data

In [2]: # Initialize count and tf-idf vectorizers
        tf_idf_vectorizer = TfidfVectorizer(stop_words='english')
        count_vectorizer = CountVectorizer(stop_words='english')

In [3]: # Get the data from json file
        data = create_data()
        # Extract text from articles
        articles_text = data['text']

In [4]: # Get count and tf-idf matrices
        tf_idf_matrix = tf_idf_vectorizer.fit_transform(articles_text)
        count_matrix = count_vectorizer.fit_transform(articles_text)

In [5]: # Extract features
        tf_idf_feature_names = tf_idf_vectorizer.get_feature_names()
        count_feature_names = count_vectorizer.get_feature_names()

In [6]: # Number of topics
        n_topics = 3

        # Number of top words per topic
        n_top_words = 20

In [7]: # Run LDA
        lda = LatentDirichletAllocation(n_components=n_topics, learning_method='onl

        # Run NMF
        nmf = NMF(n_components=n_topics, init='nndsvd').fit(tf_idf_matrix)
```

```
In [8]: # Function to display the top words for lda and nmf
        def show_topics(model, feature_names,n_top_words_per_topic):
            """
            Shows the number of of words per topic for each topic
            :param model Scikit learn model
            :param feature_names vector
            :param n_top_words_per_topic int
            """
            for topic_index, topic in enumerate(model.components_):
                print("Topic %d:" %(topic_index))
                print(" ".join([feature_names[i] for i in topic.argsort()[:-n_top_w

        # Run show_topics on lda, nmf and lsa
        print("LDA results:")
        show_topics(lda,count_feature_names,n_top_words)
        print()
        print("NMF results:")
        show_topics(nmf,tf_idf_feature_names,n_top_words)
```

```
LDA results:
Topic 0:
said mr police government people united state states year officials country china c
Topic 1:
people new like said court law years percent world year state time public page stat
Topic 2:
mr trump said clinton campaign republican mrs party president obama election new pe

NMF results:
Topic 0:
trump mr clinton mrs campaign republican said party voters republicans donald presi
Topic 1:
police said people officers court black city law ms department justice mr year new
Topic 2:
mr said united russia turkey government china military syria european islamic state
```

Results of LDA and NMF We used both LDA and NMF to obtain the top 20 words per topic in all of the New York Times articles. We see that we obtain similar terms from both algorithms but are assigned different topic numbers.

For the topics returned from LDA, we see that topic 0 focuses more on World news, topic 1 focuses on opinion articles and topic 2 focus on U.S news. Topic 0 contains terms not directly associated with United States. Topic 1 contains more terms that one might consider more ambiguous. This would be an indication that Topic 1 is probably mostly opinion articles. Topic 2 contains terms that mostly associated with U.S politics and more generally, U.S news.

For the topics returned from NMF, we see that topic 0 focuses more on U.S news, topic 1 focuses more on opinion articles and topic 2 focuses more on World news. Topic 0 has more terms associated with U.S news. Topic 1 contains more terms that are ambiguous and are not particularly related to U.S news or World news. Topic 2 contains more terms related to World news

The results show that both LDA and NMF meaningfully cluster the corpus in accordance to similar terms. However, subjectively, I would say that Non-negative Matrix Factorization modeled the topics better than Latent Dirichlet Allocation. This is because we can easily see clear demarcation, in terms of the words chosen, of each topic. Whereas, for Latent Dirichlet Allocation, topic 0 could be classified as either U.S news or World news, even though it has more terms related to World news.

```
In [9]: # Run k-means on both count_matrix and tf_idf_matrix and lsa versions as we
        km_count = KMeans(n_clusters=n_topics, init='k-means++', max_iter=100, n_ir
        km_tf_idf = KMeans(n_clusters=n_topics, init='k-means++', max_iter=100, n_i

In [10]: def show_clusters(model,feature_names, top_words, topics):
             """
             Shows top words per cluster
             :param model Scikit learn model
             :param feature_names vector
             :param top_words int
             :param topics int
             """
             order_centroids = model.cluster_centers_.argsort()[:, ::-1]
             terms = feature_names

             for i in range(topics):
                 print("Cluster %d: " % i, end='')
                 print(" ".join([terms[i] for i in order_centroids[i , :top_words]]
                 """
                 for ind in order_centroids[i, :top_words]:
                     print(' %s' % terms[ind])
                 """
                 print()

         # Run show_clusters on K-means objects and feature_names vectors
         print("K-means fit on a tf_idf matrix with tf_idf features")
         show_clusters(km_tf_idf, tf_idf_feature_names,n_top_words,n_topics)

         print("K-means fit on a count matrix with count features")
         show_clusters(km_count, count_feature_names, n_top_words,n_topics)

K-means fit on a tf_idf matrix with tf_idf features
Cluster 0: trump mr clinton mrs campaign said republican party voters republicans p

Cluster 1: said mr people new court like state page ms year trump world law years 2

Cluster 2: said mr police government china united state people officers military is

K-means fit on a count matrix with count features
Cluster 0: said mr trump people new state like states government world year address
```

```
Cluster 1: mr trump said clinton campaign mrs republican president new party people

Cluster 2: said mr people police state government ms new year united president like
```

Results of K-means We fit K-means twice, one on a bag of words matrix and the other on a tf-idf matrix. The parameter k was set to 3 as this is the number of topics in our dataset. This was done so that a comparison of both types of models can be done. We obtain the top 20 terms per cluster.

For the clusters returned from the K-means algorithm fitted on a tf-idf model, cluster 0 focuses more on U.S news, cluster 1 focuses more on opinion articles, and cluster 2 focuses more on World news. Cluster 0 contains more terms that would indicate that it has more U.S news articles. Cluster 1 has terms that fit into either U.S news or World news and are more ambiguous. Thus, the logical label for this cluster is opinion. Cluster 2 contains mostly terms focused around World news.

For clusters returned from the K-means algorithm fitter on a bag-of-words model, all the clusters seem to contain terms mostly found in U.S news articles. This is because in a bag-of-words model, we take the frequency of each word as it appear in the corpus. Thus, we do not weight relative to the document it is in.

The results show that running K-means on a tf-idf matrix produces more meaningful clusters, as they correspond more to the actually topics on our dataset. Due to the nature of the bag-of-words model, we got mostly the same terms in each cluster when running K-means on a count matrix.

Classification Now, we use various classifiers to determine which would be better for classifying opinion articles. The classifiers we used were Naive Bayes, Logistic Regression, Random Forests and Decision Trees. We fit each classifier on both a bag-of-words model and tf-idf model. We use K-fold cross validation, where K=10, to compare each classifier and f1 score are our metric of comparison.

```
In [11]: from sklearn.naive_bayes import MultinomialNB
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.pipeline import Pipeline
         from sklearn.model_selection import KFold
         from sklearn.metrics import f1_score

In [12]: # Pipelines for each classifier
         pipeline_multinomial_nb_count = Pipeline([('vectorizer', CountVectorizer()
         pipeline_multinomial_nb_tf_idf = Pipeline([('vectorizer', TfidfVectorizer
         pipeline_logistic_regression_count = Pipeline([('vectorizer', CountVectori
         pipeline_logistic_regression_tf_idf = Pipeline([('vectorizer', TfidfVector
         pipeline_random_forest_count = Pipeline([('vectorizer', CountVectorizer())
         pipeline_random_forest_tf_idf = Pipeline([('vectorizer', TfidfVectorizer()
         pipeline_decision_tree_count = Pipeline([('vectorizer', CountVectorizer())
         pipeline_decision_tree_tf_idf = Pipeline([('vectorizer', CountVectorizer()
```

In [13]:
```
# Use 10-fold cross validation to determine accuracy of each method
k_fold = KFold(n_splits=10)
scores_count = {'Multinomial Naive Bayes':[], 'Logistic Regression':[], 'R
scores_tf_idf = {'Multinomial Naive Bayes':[], 'Logistic Regression':[], '

def compute_scores(pipeline,type_of_classifier,scores):
    """
    Compute the scores for each classifier given a corresponding pipeline
    :param pipeline sklean Pipeline
    :param type_of_classifier str
    :param scores dictionary
    """
    for train_indices, test_indices in k_fold.split(data['text']):
        train_text = data.iloc[train_indices]['text'].values
        train_y = data.iloc[train_indices]['type'].values

        test_text = data.iloc[test_indices]['text'].values
        test_y = data.iloc[test_indices]['type'].values

        pipeline.fit(train_text, train_y)
        predictions = pipeline.predict(test_text)

        score = f1_score(test_y, predictions,average='weighted')
        scores[type_of_classifier].append(score)
    scores[type_of_classifier] = sum(scores[type_of_classifier])/len(score

# Multinomial NB
compute_scores(pipeline_multinomial_nb_count, 'Multinomial Naive Bayes', s
compute_scores(pipeline_multinomial_nb_tf_idf, 'Multinomial Naive Bayes',

# Logistic Regression
compute_scores(pipeline_logistic_regression_count, 'Logistic Regression',
compute_scores(pipeline_logistic_regression_tf_idf, 'Logistic Regression',

# Random Forests
compute_scores(pipeline_random_forest_count, 'Random Forests', scores_cour
compute_scores(pipeline_random_forest_tf_idf, 'Random Forests', scores_tf_

# Decision Trees
compute_scores(pipeline_decision_tree_count, 'Decision Trees', scores_cour
compute_scores(pipeline_decision_tree_tf_idf, 'Decision Trees', scores_tf_

# Display scores
print("Using a count matrix")
print(scores_count)
print("Using a tf idf matrix")
print(scores_tf_idf)
```

```
Using a count matrix
{'Random Forests': 0.93964102968769725, 'Logistic Regression': 0.97204863040889067,
Using a tf idf matrix
{'Random Forests': 0.93698017432361635, 'Logistic Regression': 0.95687111851044349,
```

Results from classifiers The results show that Logistic Regression with a bag-of-words model obtains the highest accuracy out of all the other classifiers and that overall classifiers using the bag-of-words model obtained better results than those that were using the tf-idf model. The only classifier using a tf-idf model that performed better than one using a bag-of-words model was Decision Trees.

0.1.4 Conclusion

We conclude that it is possible to distinguish opinion articles from news articles. Topic modeling using a tf-idf model with Non-negative Matrix Factorization produces meaningful topics. Clustering using K-means with a tf-idf model produces better and more meaningful clusters than K-means with a bag-of-words model. For classification, it is best to use a bag-of-words model instead of a tf-idf model. Logistic Regression is best for classifying between news and opinion articles.