The Influence of Sentiment Polarity on News Popularity

Kate Chen & Christopher Wolff
Duke University

ISS110
Patrick Herron
January 16, 2018

Abstract

We conducted a quantitative study to illustrate the relationship between the sentiment polarity of a news article and its popularity. A collection of articles from the New York Times was aggregated and labeled to train a multinomial naive Bayes classifier and a support vector machine. The results were compared with an existing model and showed an improvement in classification accuracy from 58% to 71%. We also show that the output of a specific classifier sees an increase in standard deviation for sparse input vectors. Additionally, we created a Python framework that provides automated API querying, web scraping, data analysis, and visualization tools for future research. In contrast to our initial hypothesis, we found evidence that there may be a positive correlation between positive sentiment polarity and popularity, through a variety of third factors must also be taken into consideration.

Introduction

In Orson Welles' classic film *Citizen Kane*, Charles Kane revives a dying newspaper and subsequently builds a news empire through yellow journalism often solely focused on scandal or other negative events. While Welles' depiction of corruption and fraudulent information in the news industry is exaggerated, his inspiration is not mislead - the saying "bad news sells" often propels news sources and journalists to focus disproportionately negatively on current events rather than maintain a neutral approach. Psychologists suspect that readers' attraction to negative news is the cause of an innate negativity bias, but past studies have primarily determined this phenomenon through psychological reasoning rather than quantitative evidence (Stafford, 2014). Therefore, the goal of our research was to create a software application that not only determines a sentiment score for an article based solely on its text, but also examine the general correlation of an article's popularity with its sentiment. Our hypothesis was that there will be a positive correlation between the negativity of an article and its popularity. In other words, we hypothesize to see an average trend of the most popular news being the most negative, and subsequent articles increasing in average positivity as the article popularity decreases. Using our computed sentiment scores and other data, we aimed to make a conclusion based on quantitative evidence.

Background

Previous Work

Sentiment analysis has been applied in a large range of domains. For example, it has been used to gauge the negativity or positivity of news coverage focused on a specific person or topic (Godbole, Srinivasaiah, & Skiena, 2007). The majority of this study focused on the effectiveness of the sentiment analysis model through analyzing coverage sentiment in news and blogs. In addition, news and blog sentiments over time about specific topics were compared, and top positive and top negative subjects were recorded. Studies have also been conducted to analyze the correlation between news coverage of businesses and their performances in financial stock markets (Atreya, Cohen, & Zhai, 2011; Kalyani, Bharathi, & Jyothi, 2016). This sentiment model was able to predict with 70 percent accuracy movements in the stock market based on preceding news and their sentiments. In addition, sentiment scores were confirmed through comparisons to manual human sentiment scoring. However, while many studies involving sentiment analysis exist, research regarding the validity of the claim that negativity in news from accredited news organizations results in higher view counts is not prominent, if existing. Nonetheless, many processes in sentiment analysis are similar regardless of the research objective, and therefore, these studies served as useful resources for our approach.

Word Vector Representations of Documents

One of the key challenges in building a sentiment analyzer is generating a numerical representation of any given article, also referred to as a vector space representation (Salton, Wong, & Yang, 1975). The two most common representations are the bag of words model and the term frequency - inverse document frequency (tf-idf) model. Both represent the article as a fixed-size vector, which is necessary so that articles with different lengths can be used equivalently for computation. The bag of words model first tokenizes the document string, meaning it splits all of the words contained in it, and stores them in a list. It then counts how often each of the words occur and converts these frequencies into a vector $\vec{x}$. The vector has a size corresponding to the total number of words in the article collection. The words are always arranged in the same order, so that the i^{th} element x_i always corresponds to the frequency of the i^{th} word. Since the capitalization of a word usually does not change its semantics, all words are often converted to lower case beforehand. Optionally, the resulting frequencies are normalized by the document length so that longer documents do not result in a word vector with a larger overall magnitude.

One problem with the bag of words model is that common words such as ”the” or ”a” tend to occur much more often in the English language, and will therefore carry a large weight in the word vectors. However, these words should have little to no effect on the overall sentiment score. The tf-idf model takes the overall occurrence of each term in the entire document collection into account as well and assigns a larger weight for rarer terms. The tf-idf score for some term t is defined as the product of the term frequency TF(t) and the inverse document frequency IDF(t):

$$\mathrm{TFIDF}(t) = \mathrm{TF}(t) \cdot \mathrm{IDF}(t) \tag{1}$$

with

$$\mathrm{IDF}(t) = \ln\left(\frac{n_d}{1+\mathrm{DF}(t)}\right) \tag{2}$$

where n_d denotes the total number of documents and DF(t) is the number of documents containing the term t. The 1 is added to the denominator so that it will never be zero. As a result of the inverse document frequency factor, terms will be weighted according to their ”rarity” within the document collection in addition to their raw frequency. In some models the resulting tf-idf scores are normalized, typically by the Euclidean norm:

$$\vec{x}_{norm} = \frac{\vec{x}}{||\vec{x}||_2} = \frac{\vec{x}}{\sqrt{\sum x_i^2}} \tag{3}$$

Naive Bayes Classification

A naive Bayes sentiment classifier can be thought of as a function that takes a feature vector, such as one of the two described above, as an input, and assigns probabilities to each possible outcome. In our case, the outcomes were chosen to be binary labels that correspond to the articles sentiment – positive and negative. The model is based on Bayes’ theorem (Berkson, 1930), which gives the relationship

$$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)} \tag{4}$$

$P(y|x_1,...x_n)$ denotes the conditional probability of y given the set of features $x_1,...,x_n$; in other words, the likelihood of y occurring given that $x_1,...,x_n$ are true. In our case, y might be the probability that a given article has negative sentiment and the input probabilities x_i are true if the i^{th} word in the feature vector is present in the article. This term is also called the posterior probability, because it represents the desired output of the system. $P(y)$ is the prior probability that a given class occurs. For instance, if 60% of all articles have a negative sentiment, $P(y)$ is 0.6, regardless of the article. Similarly, $P(x)$ is the prior probability that a given term x occurs in any given document. The term $P(x_1,...,x_n|y)$ is the likelihood that every term in the article occurs in the input vector given the output class y. At first, it might seem that this term is difficult to compute, and that converting the challenge of finding the posterior probability to finding its ”inverse” has not made it significantly easier. However, that is where the main idea behind the naive Bayes model comes in:

Assumption 1. *The conditional probabilities $P(x_i|y)$ are independent of each other.*

That is also the origin of the word ”naive” in naive Bayes, because the assumption is usually just false, especially in news articles. For example, the probabilities of the terms ”white house” and ”politics” occurring in an article are clearly dependent on each other. Nonetheless, the independence assumption proves useful for computational purposes, because it lets us compute $P(x_1,...,x_n|y)$ as

$$P(x_1,...,x_n|y) = \prod P(x_i|y) \tag{5}$$

It simply turns into the product of all individual conditional probabilities, which are easy to compute. Lastly, a multinomial naive Bayes classifier is a special version of naive Bayes designed to work well with tf-idf weighting.

Support Vector Machines

A support vector machine (SVM) is another model that can be used for classification. An SVM creates a plane of separation between the two output classes in the n-dimensional feature space of the input vector. If we imagine that the input vector only has two features, which would correspond to having only two distinct words in the document collection, then a SVM would separate the positive and negative labels using a line.

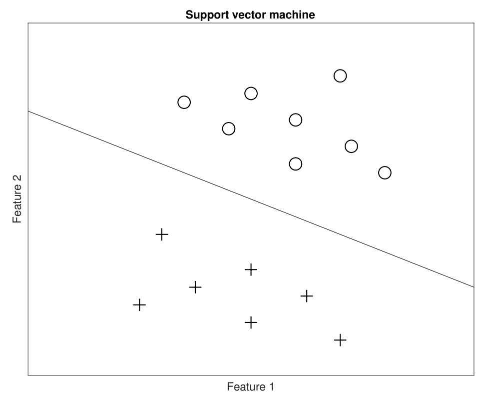

Figure 1. Two-dimensional SVM example (created with MATLAB^{TM})

Here, each point represents an article with two features. The numerical value of each feature corresponds to each articles weight in its word vector, so either its frequency count or tf-idf weight. The plus signs and circles correspond to positive and negative sentiment labels, respectively. Once this line has been "learned," any new article can easily be classified as positive if it lies below the line and negative if it lies above the line. However, an input word vector usually has about 10,000 different features, so the data can not simply be separated by a one-dimensional object. Instead, an SVM creates an (n-1)-dimensional hyperplane through the n-dimensional space that the word vectors are in. For instance, if there were three total features, the separator would be a plane.

Procedure

The procedure of our study included two main tasks. The first one was collecting an archive of news articles along with relevant information: title, abstract, full story text, and view count. We predicted that the title and abstract would most likely have the largest impact on an article's popularity, because they are directly exposed to the reader. The second component was determining a sentiment value for each of the articles. We experimented with the Python `TextBlob` library to assign sentiment values. However, for our final results, we trained our own sentiment analysis model using hand-labeled training data.

Defining Sentiment and Popularity

Before gathering any quantifiable measures of sentiment and popularity, we needed to define these two attributes first. The term sentiment can span a multitude of dimensions, such as opinion, emotions, and affect (Munezero, Montero, Sutinen, & Pajunen, 2014). However, we were mostly interested in sentiment polarity - whether a given article is positive, negative, or neutral. We defined a sentiment score attribute of an article, which is a continuous number ranging from -1 to +1, where scores of -1, 0, and +1 correspond to extremely negative polarity, neutral polarity, and extremely positive polarity, respectively. In our own classifier, we used discrete binary labels because that choice synergized best with our mathematical models. We defined popularity as the number of views a given article received over its lifespan. While other measures, such as the number of times an article was shared via online media or the total revenue it generated, were also viable measures, using the view count was convenient because it was directly available in the generated dataset. One downside to this form of measuring popularity is that older articles will likely have a higher view count. However, we reasoned that articles are generally featured for a relatively constant time period online, after which they are less likely to be read and view count stagnates. Hence, the extra views that arise from the additional time beyond the time period in which the article was featured would not significantly contribute to the total.

Data Collection

To approach collecting articles for the data, multiple factors were taken into consideration. In addition to the title and abstract of the article, popularity and time of publication needed to be taken into consideration. In the initial research phase, we investigated the ”trending” pages of news sources such as Facebook or Mashable. However, upon consideration, the New York Times offered an open source API that provided the necessary features we were looking for. In addition, the New York Times is an accredited news source with the second largest distribution in the United States and was therefore appropriate to be chosen as the data source for this project.

The New York Times Most Popular API offers three measures of popularity: most emailed, most shared and most viewed. Because the research question is focused on the popularity of an article in the context of how many people are attracted to read it, the view count was selected as the feature of choice. The API requested two inputs: a String representing the article section, and an integer denoting a time period during which the article was live. The section refers to the category within the newspaper, such as Education, Fashion, Food, Health, U.S, or World. The option to query all sections was also available. The time period values were limited to 1, 7, and 30, corresponding to the past day, week, and month of news. Surprisingly, when querying a time period value of 30, a few of the articles returned had a `published_date` older than one month. We assumed that these articles were originally published earlier, but were still available on the New York Times website. Additionally, the API allowed an offset parameter k, which specified that the k most viewed articles should be ignored and only articles starting from k+1 should be returned.

Using the inputs given by the user, the API returned a JSON file containing 20 articles. For each article, the API returns its URL, column, section, byline, title, abstract, published date, and source. The articles were listed in order of popularity. Therefore, the first article returned in the array would be the most popular article in the specified time period, defined by view count. Additional documentation about the New York Times Most Popular API can be found at New York Times Developer Website (Times, n.d.). Within the returned JSON file and features of each article, only title, and abstract were given, but the full document text was missing. We therefore used the web scraping tool `BeautifulSoup4` (Nair, 2014) to automatically extract the story content of each article from its given URL by parsing the websites HTML source code. In the end, a total of 1908 articles from all sections were collected from a period of 30 days, ranging from October 25, 2017 to November 24, 2017 and taking up 14.2 MB of memory storage space.

Sentiment Analysis

The sentiment analysis of the retrieved articles was initially done using the Python `TextBlob` library. For each article, the title, abstract, and full text were separately fed through the `TextBlob` sentiment analyzer. Then, each article was plotted against its relative popularity rank using `matplotlib` (Hunter, 2007). We used `numpy` (van der Walt, Colbert, & Varoquaux, 2011) to calculate the minimum, maximum, mean, and standard deviation for each of the three datasets.

Next, we trained our own sentiment classifier by hand-labeling over 600 news articles with binary sentiment labels: positive or negative, using a user interface (UI) we developed. The UI showed us each article's title and abstract and enabled us to enter a corresponding label, relabel previously labeled articles, or assign random labels to all articles for testing purposes. When labeling the articles, topic of the article in addition to any noticeable tone in the title or abstract were primarily taken into consideration; however, some cases were difficult to label as purely negative or purely positive. In these scenarios, guidelines for how these articles would be handled were established so that these articles would be judged equally and fairly. For example, an article about a positive aspect of a negative topic, such as a public figure releasing an apology after sexual harassment accusations, was judged as a negative article because of the core issue at hand, which would generally evoke negative emotion. In addition, because articles could not be labeled as neutral, or have a sentiment score of 0, seemingly neutral news was labeled as positive. An example of this would be an article including photographs and captions of Trump's activities in Asia, with no subjective information. Furthermore, political viewpoints were not taken into consideration - while the New York Times is generally viewed as more liberal, an article involving conversative topics did not automatically result in a negative sentiment score.

The articles were first preprocessed by filtering out stopwords and converting all terms to lowercase. We separately used both the bag of words representation as well as the tf-idf weights for the full story text of each article as input vectors. The support vector classifier and the multinomial naive Bayes classifier from the `scikit-learn` machine learning library (Pedregosa et al., 2011) were both utilized. Prior to training, the labeled articles were randomly split into 75% used for training and 25% used for testing. The testing data aided as a tool for validating our models performance to see how well they perform on unseen data. We recorded the resulting accuracy and stored the trained models using Python's object serialization tool `pickle`. Then, we compared the performance of our own models with the `TextBlob` baseline classifier. Lastly, we repeated the initial experiment of plotting sentiment against popularity but with our own classifiers.

Results and Discussion

The plots for the title, abstract, and story sentiment obtained from the `TextBlob` classifier can be seen in Figure 2.

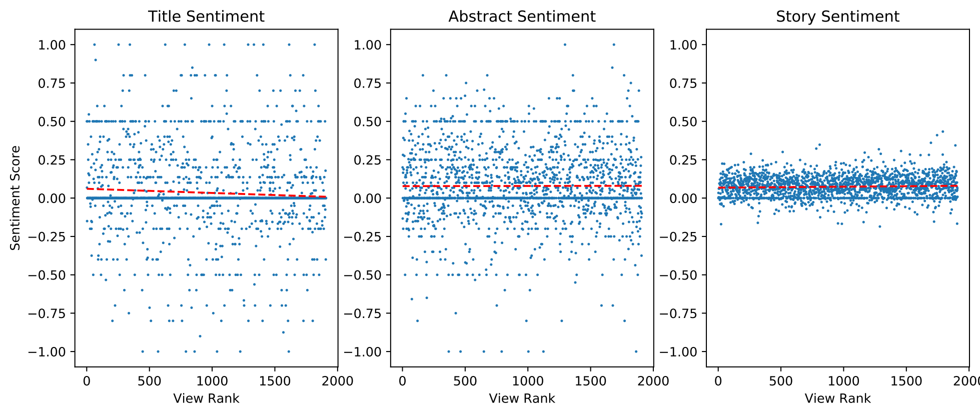

Figure 2. `TextBlob` sentiment score results (created with `matplotlib`)

The dotted red lines represent a linear fit for each of the datasets, created using `numpy`. From the flat slope of these trend lines, it is clear that there is no obvious pattern or correlation. Table 1 shows the general properties of each dataset.

	Title	Abstract	Story
min	-1	-1	-.19
max	1	1	.71
mean	.14	.22	.23
std	.30	.30	.19

Table 1

Minimum, Maximum, Mean, and Standard Deviation of TextBlob Sentiment Scores

We noticed that both the range and standard deviation were much lower when the whole story was used as an input vector. This lead us to believe that the outputs for the `TextBlob` classifier tend to regress towards a mean score as the number of input features gets bigger. Our prediction is that `TextBlob` uses a bag of words model and only takes the existence of each term into account without regard for its frequency, so that its word vector representation is simply a multi-hot vector over all possible terms. This would imply that sparse input vectors have a much higher similarity in comparison.

Upon further investigation, we realized that `TextBlob` is based on the `PatternAnalyzer` from the `nltk` library (Loper & Bird, 2002), which was trained using movie reviews. The fact that the model was trained on a completely different type of text data lead us to believe that it might be mislabeling the news articles. An example of this is the phrase ”Saudi Arabia Arrests 11 Princes, Including Billionaire Alwaleed bin Talal” with the abstract ”The sweeping campaign of arrests appears to be the latest move to consolidate the power of Crown Prince Mohammed bin Salman, the favorite son and top adviser of King Salman,” which received positive sentiment scores of 0.997 and 0.988, respectively. While this specific news is certainly negative, the sentiment analyzer return an extremely high score. We believe that this occurs because words like favorite, prince, power, or billionaire would return high sentiment scores, potentially outweighing the more negative words, such as arrest.

Labels like the one described compromised the data, which warranted and necessitated a new way to obtain the sentiment data. Further research showed that many other sentiment analyzers were also trained on movie reviews or otherwise inadequate for the purposes of this study. Therefore, in order to achieve the most ideal results for the specific needs of the sentiment analysis in this study, we created our own model. The following snippet shows our final results for combinations of different word vector representations (bag of words and tf-idf) with different classifiers (naive Bayes and SVM).

```
>>> train_model()
Found 622 labeled articles
221 +, 401 -
Vectorizing using bag of words...
Naive Bayes: 67.31% accuracy
SVM: 64.10% accuracy
Vectorizing using tfidf...
Naive Bayes: 69.87% accuracy
SVM: 71.15% accuracy
```

The `TextBlob` classifier was used on the full set of labeled articles. Since the returned labels from `TextBlob` were non-binary, we rounded all positive scores to +1 and all negative scores to -1. For scores of 0, the training example was disregarded and not counted towards the overall accuracy. The `TextBlob` classifier achieved a total accuracy of **57.76%**, which is significantly less than our model.

The plots below show the relationship between view count and sentiment score based on the new sentiment scores computed with our models.

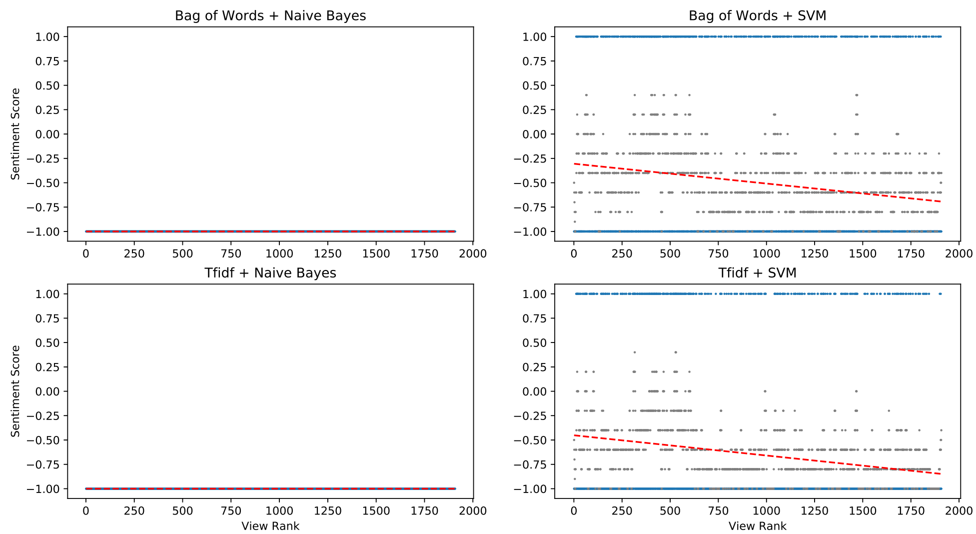

Figure 3. Our sentiment score results (created with `matplotlib`)

The blue data points are exact predictions made by the classifiers. The red line is a linear fit for each graph, representing the general trend of the data. The orange scatter plot is a ten-point moving-average of the data. This makes it easier to visualize the progression of the sentiment values over time. An unexpected observation is that the naive Bayes classifier guessed a negative sentiment for all data points. We predict that this relates to the fact that a majority of labeled articles (401) were negative, compared to the 221 positive articles. When computing the class probability $P(\text{negative}|t_1,...,t_n)$, the prior $P(y)$ dominates all other terms. When the final guess it computed using argmax{P(negative), P(positive)}, the positive probability will never be high enough, regardless of the word vectorizer chosen. The trend line for the SVM data shows a clear negative trend, contrary to our initial hypothesis. As the popularity of the articles decreases, the negativity increases. However, the hypothesis was supported in regards to the negativity of news articles, in that a clear majority of the articles returned and ranked were negative as opposed to positive. Therefore, the data supports that negative news is indeed popularly read and more so than positive news.

There are multiple third factors that may have influenced this data. For example, the conclusion may be influenced by a simple possibility that, in general, there is more negative news available to be read than positive news. It may also be possible that in general, negative news is more commonly relevant to readers' lives, as they may cause feelings of fear or urgency that may make them more important to read than positive news. Negative news may also lead to more complex, controversial, or nuanced issues that require the reader to click on the article and view it to further understand the article rather than being able to understand the jist from reading the title, and not opening the article to read it. Regardless, the data presents a conclusion that indicates a more complex relationship between sentiment and article popularity beyond the simple assumption that negative news is more popular.

Conclusions

Though the results of this study did not strictly support nor refute the hypothesis and further research would be necessary to confirm our results or otherwise discover other correlations, the surprising outcome of the data raises an interesting question into the assumption that negative news is more popular than positive news. While the data does seem to support that humans naturally gravitate more towards negative news, the true correlation between sentiment and popularity does not seem to be as simple or clear. In addition, third variables such as the numerical prominence of negative news or cultural context in current events may heavily influence the data. However, the conclusion of this study does present a different perspective and possibly contradict a long-held common belief in the news industry. Therefore, further investigation may uncover facts and trends that could bring insight into the true impact of negativity in news popularty or otherwise call industry changes to action.

Due to the time and resource limitations of this project, there are a number of ways that this research can be expanded upon, confirmed, or disputed. Further investigation with larger data pools, more detailed sentiment analysis and potential introduction of other factors such as changes over time can help researchers form a more nuanced and accurate hypothesis. In addition, as discovered in the initial sentiment analysis model, the standard deviation of the `TextBlob` sentiment scores was lower when the entire story was analyzed for sentiment. Therefore, a potential extension to this study would be to conduct the sentiment analysis on the entire articles rather than just the titles or abstracts. However, the methods and models used in this study can provide a solid foundation upon which further research can be conducted to reach a stable and replicatable conclusion regarding the correlation between sentiment and news popularity.

References

Atreya, A., Cohen, N., & Zhai, J. (2011). Sentiment analysis of news articles for financial signal prediction. doi: https://nlp.stanford.edu/courses/cs224n/2011/reports/nccohen-aatreya-jameszjj.pdf

Berkson, J. (1930, 02). Bayes' theorem. *Ann. Math. Statist.*, *1*(1), 42–56. Retrieved from `https://doi.org/10.1214/aoms/1177733259` doi: 10.1214/aoms/1177733259

Godbole, N., Srinivasaiah, M., & Skiena, S. (2007). Large-scale sentiment analysis for news and blogs. *International Conference on Weblogs and Social Media*. doi: http://icwsm.org/papers/3–Godbole-Srinivasaiah-Skiena.pdf

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90-95. Retrieved from `http://aip.scitation.org/doi/abs/10.1109/MCSE.2007.55` doi: 10.1109/MCSE.2007.55

Kalyani, J., Bharathi, H., & Jyothi, R. (2016). Stock trend prediction using news sentiment analysis. doi: https://arxiv.org/abs/1607.01958

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. In *In proceedings of the acl workshop on effective tools and methodologies for teaching natural language processing and computational linguistics. philadelphia: Association for computational linguistics.*

Munezero, M. D., Montero, C. S., Sutinen, E., & Pajunen, J. (2014, April). Are they different? affect, feeling, emotion, sentiment, and opinion detection in text. *IEEE Transactions on Affective Computing*, *5*(2), 101-111. doi: 10.1109/TAFFC.2014.2317187

Nair, V. G. (2014). *Getting started with beautiful soup*. Packt Publishing.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Salton, G., Wong, A., & Yang, C. S. (1975, November). A vector space model for automatic indexing. *Commun. ACM*, *18*(11), 613–620. Retrieved from `http://doi.acm.org/10.1145/361219.361220` doi: 10.1145/361219.361220

Stafford, T. (2014). Psychology: Why bad news dominates the headlines. *BBC - Future*. doi: http://www.bbc.com/future/story/20140728-why-is-all-the-news-bad

Times, N. Y. (n.d.). The new york times developer network. doi: https://www.developer.nytimes.com

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22-30. Retrieved from `http://aip.scitation.org/doi/abs/10.1109/MCSE.2011.37` doi: 10.1109/MCSE.2011.37

Appendix

Source Code

This appendix contains the main source code used for this project. Additional files as well as a detailed README that explains how to run this project can be found at https://github.com/christopher-wolff/news-analysis.

__main__.py

```python
"""Sample use of analyzer module."""

from analyzer import query
from analyzer import scrape_stories
from analyzer import label_articles
from analyzer import train_model
from analyzer import analyze
from analyzer import visualize

if __name__ == '__main__':
    print('Requesting articles from NYT API')
    print('================================')
    query(num_queries=1)
    print()

    print('Scraping full article texts from NYT website')
    print('============================================')
    scrape_stories()
    print()

    print('Labeling articles')
    print('=================')
    label_articles(reset=True, rand_labels=True)
    print()

    print('Training classifiers')
    print('====================')
    train_model()
    print()

    print('Analyzing data')
    print('====================')
    analyze()
    print()

    print('Visualize results')
    print('=================')
    visualize()
```

analyzer.py

```
"""Main module."""


__authors__ = 'Christopher Wolff, Kate Chen'
__version__ = '1.0'
__date__ = '9/10/2017'


import json
import os.path
import pickle
import random
import urllib

from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn import naive_bayes
from sklearn import svm
from sklearn.metrics import accuracy_score
from textblob import TextBlob
import matplotlib.pyplot as plt
import requests
import numpy as np


SETTINGS_PATH = 'settings.json'
RAW_PATH = 'data/raw.json'
STORIES_PATH = 'data/with_stories.json'
LABELS_PATH = 'data/with_labels.json'
SENTIMENTS_PATH = 'data/with_sentiments.json'
MNB_PATH = 'models/mnb.pkl'
SVM_PATH = 'models/svm.pkl'
COUNT_VECT_PATH = 'models/count_vect.pkl'
TFIDF_VECT_PATH = 'models/tfidf_vect.pkl'

BASE_URI = 'http://api.nytimes.com/svc/mostpopular/v2'
TYPE = 'mostviewed'
SECTION = 'all-sections'
TIME_PERIOD = '1'
RESPONSE_FORMAT = 'json'
```

```python


def query(num_queries=1):
    """Request data from NYT and store it as a json file.

    Args:
        num_queries (int): The number of queries
    """
    # Load API key
    settings = json.load(open(SETTINGS_PATH))
    API_KEY = settings['API_KEY']

    # Send requests
    URI = f'{BASE_URI}/{TYPE}/{SECTION}/{TIME_PERIOD}.{RESPONSE_FORMAT}'
    articles = []
    for k in range(num_queries):
        print(f'Running query {k+1}...')
        offset = k * 20
        payload = {'api_key': API_KEY, 'offset': offset}
        response = requests.get(URI, params=payload)
        articles += response.json()['results']

    # Save to file
    with open(RAW_PATH, 'w') as output_file:
        json.dump(articles, output_file)


def scrape_stories():
    """Get full document texts from urls."""
    # Load articles
    articles = json.load(open(RAW_PATH))

    # Submit GET request and parse response content
    for k, article in enumerate(articles):
        print(f'Scraping article {k+1}...')
        url = article['url']
        f = urllib.request.urlopen(url)
        soup = BeautifulSoup(f, 'html5lib')
        story = ''
        for par in soup.find_all('p', class_='story-body-text \
                                              story-content'):
            if par.string:
                story += ' ' + par.string
        article.update({'story': story})

```

```
    # Save articles
    with open(STORIES_PATH, 'w') as output_file:
        json.dump(articles, output_file)


def label_articles(reset=False, relabel=False, start=0, rand_labels=False):
    """Run UI for sentiment labeling.

    Loads all articles and presents those without a label.

    Args:
        reset (boolean): Delete all labels
        relabel (boolean): Allow option to override existing labels
        start (int): Article number to start from
        rand_labels (boolean): Assign all random labels
    """
    # Load articles
    if reset or not os.path.isfile(LABELS_PATH):
        articles = json.load(open(STORIES_PATH))
    else:
        articles = json.load(open(LABELS_PATH))
    if start >= len(articles):
        raise ValueError(f'Invalid starting point: {start}')

    # Label articles
    sentiments = [-1, 1]
    print(f'Available sentiments: {sentiments}')
    for k, article in enumerate(articles[start:]):
        if not relabel and 'sentiment' in article:
            continue
        print(f'Article: {k+start+1}')
        print(f"Title: {article['title']}")
        print(f"Abstract: {article['abstract']}")
        if rand_labels:
            sent = random.choice(sentiments)
        else:
            try:
                sent = int(input('Label: '))
            except ValueError:
                break
            if sent not in sentiments:
                break
        article.update({'sentiment': sent})
        print('----------------------------')

```

```
    # Save articles
    with open(LABELS_PATH, ’w’) as output_file:
        json.dump(articles, output_file)


def train_model(random_state=None):
    """Train a sentiment analyzer model.

    Args:
        random_state (int): Random seed for train_test_split used by numpy
    """
    # Load articles
    articles = json.load(open(LABELS_PATH))
    # Extract data
    articles = [article for article in articles if ’sentiment’ in article]
    stopset = set(stopwords.words(’english’))
    titles = [article[’title’] for article in articles]
    labels = [article[’sentiment’] for article in articles]

    # Vectorize data
    count_vect = CountVectorizer(lowercase=True,
                                 strip_accents=’ascii’,
                                 stop_words=stopset,
                                 decode_error=’replace’)
    tfidf_vect = TfidfVectorizer(use_idf=True,
                                 lowercase=True,
                                 strip_accents=’ascii’,
                                 stop_words=stopset,
                                 decode_error=’replace’)

    # Analyze and display relevant information
    num_total = len(articles)
    num_pos = sum(article[’sentiment’] == 1 for article in articles)
    num_neg = sum(article[’sentiment’] == -1 for article in articles)
    print(f’Found {num_total} labeled articles’)
    print(f’{num_pos} +, {num_neg} -’)

    # Train using count vectorizer
    print(’Vectorizing using bag of words...’)
    x = count_vect.fit_transform(titles)
    y = labels
    if random_state is not None:
        x_train, x_test, y_train, y_test = train_test_split(
                x, y, random_state=random_state)
    else:
```

```
        x_train, x_test, y_train, y_test = train_test_split(x, y)

    mnb_clf = naive_bayes.MultinomialNB()
    mnb_clf.fit(x_train, y_train)
    y_pred = mnb_clf.predict(x_test)
    mnb_acc = accuracy_score(y_test, y_pred) * 100
    print('Naive Bayes: %.2f%% accuracy' % mnb_acc)

    svm_clf = svm.SVC(probability=True)
    svm_clf.fit(x_train, y_train)
    y_pred = svm_clf.predict(x_test)
    svm_acc = accuracy_score(y_test, y_pred) * 100
    print('SVM: %.2f%% accuracy' % svm_acc)

    # Train using tfidf vectorizer
    print('Vectorizing using tfidf...')
    x = tfidf_vect.fit_transform(titles)
    y = labels
    if random_state is not None:
        x_train, x_test, y_train, y_test = train_test_split(
                x, y, random_state=random_state)
    else:
        x_train, x_test, y_train, y_test = train_test_split(x, y)

    mnb_clf = naive_bayes.MultinomialNB()
    mnb_clf.fit(x_train, y_train)
    y_pred = mnb_clf.predict(x_test)
    mnb_acc = accuracy_score(y_test, y_pred) * 100
    print('Naive Bayes: %.2f%% accuracy' % mnb_acc)

    svm_clf = svm.SVC(probability=True)
    svm_clf.fit(x_train, y_train)
    y_pred = svm_clf.predict(x_test)
    svm_acc = accuracy_score(y_test, y_pred) * 100
    print('SVM: %.2f%% accuracy' % svm_acc)

    # Store vectorizers and trained classifiers
    with open(SVM_PATH, 'wb') as output_file:
        pickle.dump(mnb_clf, output_file)
    with open(MNB_PATH, 'wb') as output_file:
        pickle.dump(svm_clf, output_file)
    with open(COUNT_VECT_PATH, 'wb') as output_file:
        pickle.dump(count_vect.vocabulary_, output_file)
    with open(TFIDF_VECT_PATH, 'wb') as output_file:
        pickle.dump(tfidf_vect.vocabulary_, output_file)
```

```


def analyze():
    """Analyze article data."""
    # Calculate sentiment scores
    articles = json.load(open(LABELS_PATH))
    mnb_clf = pickle.load(open(MNB_PATH, 'rb'))
    svm_clf = pickle.load(open(SVM_PATH, 'rb'))
    count_vocabulary = pickle.load(open(COUNT_VECT_PATH, 'rb'))
    tfidf_vocabulary = pickle.load(open(TFIDF_VECT_PATH, 'rb'))
    stopset = set(stopwords.words('english'))
    count_vect = CountVectorizer(lowercase=True,
                                 strip_accents='ascii',
                                 stop_words=stopset,
                                 decode_error='replace',
                                 vocabulary=count_vocabulary)
    tfidf_vect = TfidfVectorizer(use_idf=True,
                                 lowercase=True,
                                 strip_accents='ascii',
                                 stop_words=stopset,
                                 decode_error='replace',
                                 vocabulary=tfidf_vocabulary)
    for k, article in enumerate(articles):
        title = article['title']
        abstract = article['abstract']
        story = article['story']
        print(f'{k+1}: {title}')
        title_sent = TextBlob(title).sentiment
        abstract_sent = TextBlob(abstract).sentiment
        story_sent = TextBlob(story).sentiment
        article.update({'title_sent': title_sent,
                        'abstract_sent': abstract_sent,
                        'story_sent': story_sent})
        print(f'{title_sent} {abstract_sent} {story_sent}')

        count = count_vect.fit_transform([title])
        tfidf = tfidf_vect.fit_transform([title])
        article.update({'count_mnb_sent': mnb_clf.predict(count).item(0),
                        'count_svm_sent': svm_clf.predict(count).item(0),
                        'tfidf_mnb_sent': mnb_clf.predict(tfidf).item(0),
                        'tfidf_svm_sent': svm_clf.predict(tfidf).item(0)})

    # Test TextBlob performance
    num_total = 0
    num_correct = 0
```

```
    for article in articles:
        if 'sentiment' not in article:
            continue
        title_sent = article['title_sent'].polarity
        true_sent = article['sentiment']
        if title_sent == 0:
            continue
        if _sign(title_sent) == true_sent:
            num_correct += 1
        num_total += 1
    acc = num_correct / num_total * 100
    print('========================')
    print('TextBlob accuracy: %.2f' % acc)
    print('========================')

    # Determine min, max, mean, and std
    title_sents = np.array([a['title_sent'] for a in articles])
    abstract_sents = np.array([a['abstract_sent'] for a in articles])
    story_sents = np.array([a['story_sent'] for a in articles])

    print('Title Sentiments')
    print('----------------')
    print(f'min: {np.min(title_sents)}')
    print(f'max: {np.max(title_sents)}')
    print(f'mean: {np.mean(title_sents)}')
    print(f'std: {np.std(title_sents)}')
    print()

    print('Abstract Sentiments')
    print('-------------------')
    print(f'min: {np.min(abstract_sents)}')
    print(f'max: {np.max(abstract_sents)}')
    print(f'mean: {np.mean(abstract_sents)}')
    print(f'std: {np.std(abstract_sents)}')
    print()

    print('Story Sentiments')
    print('----------------')
    print(f'min: {np.min(story_sents)}')
    print(f'max: {np.max(story_sents)}')
    print(f'mean: {np.mean(story_sents)}')
    print(f'std: {np.std(story_sents)}')
    print()

    # Save to file
```

```
    with open(SENTIMENTS_PATH, 'w') as output_file:
        json.dump(articles, output_file)


def visualize():
    """Visualize the data."""
    # Load data
    articles = json.load(open(SENTIMENTS_PATH))
    title_sents = [article['title_sent'][0] for article in articles]
    abstract_sents = [article['abstract_sent'][0] for article in articles]
    story_sents = [article['story_sent'][0] for article in articles]
    count_mnb_sents = [article['count_mnb_sent'] for article in articles]
    count_svm_sents = [article['count_svm_sent'] for article in articles]
    tfidf_mnb_sents = [article['tfidf_mnb_sent'] for article in articles]
    tfidf_svm_sents = [article['tfidf_svm_sent'] for article in articles]

    view_rank = range(1, len(articles) + 1)

    # Calculate trendlines
    z1 = np.polyfit(view_rank, title_sents, 1)
    p1 = np.poly1d(z1)
    z2 = np.polyfit(view_rank, abstract_sents, 1)
    p2 = np.poly1d(z2)
    z3 = np.polyfit(view_rank, story_sents, 1)
    p3 = np.poly1d(z3)

    z4 = np.polyfit(view_rank, count_mnb_sents, 1)
    p4 = np.poly1d(z4)
    z5 = np.polyfit(view_rank, count_svm_sents, 1)
    p5 = np.poly1d(z5)
    z6 = np.polyfit(view_rank, tfidf_mnb_sents, 1)
    p6 = np.poly1d(z6)
    z7 = np.polyfit(view_rank, tfidf_svm_sents, 1)
    p7 = np.poly1d(z7)

    # Compute moving average
    window_size = 10
    window = np.ones(int(window_size))/float(window_size)
    count_svm_sents_ma = np.convolve(count_svm_sents, window, 'same')
    tfidf_svm_sents_ma = np.convolve(tfidf_svm_sents, window, 'same')

    # Plot sentiment versus view rank
    # TextBlob
    plt.figure(1)
    plt.subplot(1, 3, 1)
```

```
    plt.scatter(view_rank, title_sents, s=5)
    plt.plot(view_rank, p1(view_rank), 'r--')
    plt.title('Title Sentiment')
    plt.xlabel('View Rank')
    plt.ylabel('Sentiment Score')
    plt.ylim(-1.1, 1.1)

    plt.subplot(1, 3, 2)
    plt.scatter(view_rank, abstract_sents, s=5)
    plt.plot(view_rank, p2(view_rank), 'r--')
    plt.title('Abstract Sentiment')
    plt.xlabel('View Rank')
    plt.ylim(-1.1, 1.1)

    plt.subplot(1, 3, 3)
    plt.scatter(view_rank, story_sents, s=5)
    plt.plot(view_rank, p3(view_rank), 'r--')
    plt.title('Story Sentiment')
    plt.xlabel('View Rank')
    plt.ylim(-1.1, 1.1)

    # sklearn classifiers
    plt.figure(2)
    plt.subplot(2, 2, 1)
    plt.scatter(view_rank, count_mnb_sents, s=5)
    plt.plot(view_rank, p4(view_rank), 'r--')
    plt.title('Bag of Words + Naive Bayes')
    plt.ylabel('Sentiment Score')
    plt.ylim(-1.1, 1.1)

    plt.subplot(2, 2, 2)
    plt.scatter(view_rank, count_svm_sents, s=5)
    plt.scatter(view_rank, count_svm_sents_ma, s=5, facecolor='0.5')
    plt.plot(view_rank, p5(view_rank), 'r--')
    plt.title('Bag of Words + SVM')
    plt.ylim(-1.1, 1.1)

    plt.subplot(2, 2, 3)
    plt.scatter(view_rank, tfidf_mnb_sents, s=5)
    plt.plot(view_rank, p6(view_rank), 'r--')
    plt.title('Tfidf + Naive Bayes')
    plt.xlabel('View Rank')
    plt.ylabel('Sentiment Score')
    plt.ylim(-1.1, 1.1)
```

```
    plt.subplot(2, 2, 4)
    plt.scatter(view_rank, tfidf_svm_sents, s=5)
    plt.scatter(view_rank, tfidf_svm_sents_ma, s=5, facecolor='0.5')
    plt.plot(view_rank, p7(view_rank), 'r--')
    plt.title('Tfidf + SVM')
    plt.xlabel('View Rank')
    plt.ylim(-1.1, 1.1)

    plt.show()


def _sign(x):
    if x < 0:
        return -1
    elif x > 0:
        return 1
    else:
        return 0
```