Northeastern University

NEWS CLASSIFICATION AND SENTIMENT ANALYSIS

Amandeep Singh[1], Eklavya Saxena[2], Ankur Jain[3]

[1]COE, Northeastern University, Boston
singh.ama@husky.neu.edu
[2]COE, Northeastern University, Boston
saxena.e@husky.neu.edu
[3]COE, Northeastern University, Boston
jain.anku@husky.neu.edu

Research Paper

CSYE – 7245

Big-Data Systems & Intelligence Analytics

Prof. Nicholas W. Brown[4]

[4]COE, Northeastern University, Boston
ni.brown@northeastern.edu

April 28, 2018

1. Abstract - This study is to show the classification of news articles by sentiment and topic. The vision is to create the capability to track how sentiment on a topic has evolved over time, how different news outlets cover the same topic and, in the limit, to be able to predict future behavior through sentiment trends. The most challenging task was to scrap the news article from websites and to overcome this issue we have used python's packages and libraries. We used classification models such as Naïve Bayes, Multi-Layer Perceptron and XG Boost. Upon completion of the project we found the XG Boost model is preferable because other models are not giving accurate results on the unseen data and it is not overfitted.

2. Introduction - Most work on opinion mining has been carried out on subjective text types such as blogs and product reviews. Authors of such text types typically express their opinion freely. The situation is different in news articles: many newspapers at least want to give an impression of objectivity so that journalists will often refrain from using clearly positive or negative vocabulary. Automatically identifying sentiment that is not expressed lexically is rather difficult, but lexically expressed opinion can be found in news texts, even if it is less frequent than in product or film reviews.

we have focused in our recent opinion mining experiments, presented here, on considering smaller and larger word windows around entities, and we have attempted to separate positive and negative sentiment from good and bad news.

3. Methodologies/Approach – Below is the high-level design of the process:

Fig. 1 – Process of pipeline

We approached the problem using incrementally powerful algorithms. This allowed us to compare the performance as well as to comment on the advantages and disadvantages of each algorithm.

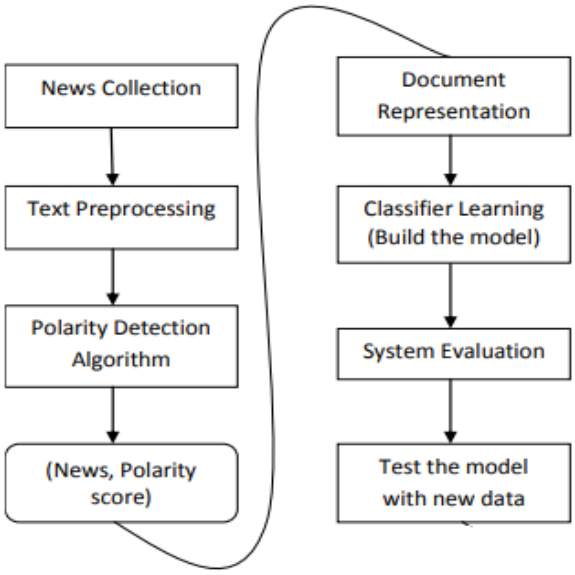

Fig. 2 Step by Step process

3.1 News collection/Data - To predict the sentiments and category of news article, we scraped the data from the web. For that, we have passed the website links to a JSON file and that file was used to scrap the data. We have used python's Article library which gives article's author, publish date, content, video links etc. We have stored data into a separate CSV file which was used for further process.

3.2 Text preprocessing - Text data is unstructured data. So, we cannot provide raw test data to classifier as an input. Firstly, we need to tokenize the document into words to operate on word level. Text data contains more noisy words which are not contributing towards classification. So, we need to drop those words. In addition, text data may contain numbers, more white spaces, tabs, punctuation characters, stop words etc. We also need to clean data by removing all those words.
We have used sentence segmentation, tokenization, lemmatization, stop words and vectorization techniques.

3.3 Polarity/Sentiment Detection Algorithm - Once the above techniques were applied, we have used python TextBlob library to predict the polarity and sentiment value. We have categorized the articles based on the following conditions –

1. If sentiment value > 0 then it is considered as a positive sentiment

2. If sentiment value < 0 then it is considered as a negative sentiment
3. If sentiment value = 0 then it is considered as a neutral sentiment

3.4 Classifier Learning - As most of the research shows that SVM and Naïve Bayes classification algorithms performs good in text classification. So, we are considering these two and additionally we used XG Boost algorithms to classify the text and check each algorithm's accuracy. We can compare all the results such as accuracy, precision, recall and other model evaluation methods. All three classification algorithms are implemented and tested.

3.4 System Evaluation - We divided the data into train and test set. We evaluated all three classifiers performance by checking each one's accuracy, precision, recall. The results are as given in the next section.

3.5 Testing with new data - When comparing results of all classifiers, XG Boost classifier performs well for unknown data. Despite of less accuracy as compared to other two models, it is best because other models are overfitted.

3.6 Model Deployment –

3.6.1 Pickle – It is the standard way of serializing objects in Python. We used pickle operation to serialize our machine learning algorithms and save the serialized format to a file. Later we can load this file to de-serialize our model and use it to make new predictions.

3.6.2 Amazon S3 – To store the pickle file on cloud, we have used python's boto library which is a python interface to Amazon Web Services.

3.6.3 Docker – Docker image automates the task and makes it OS independent. We have Dockerized our code to make it accessible from anywhere to anyone. The image can be downloaded from the below link. https://hub.docker.com/r/ankkur13/bigdatadockerimage/4.

4. Code with Documentation -

4.1 Web Scraping – To scrap the news article, JSON file is the input which has all the required links.

```
# Set the limit for number of articles to download
LIMIT = 1000000000
articles_array = []

data = {}
data['newspapers'] = {}

# Loads the JSON files with news sites
with open('NewsPapers.json') as data_file:
    companies = json.load(data_file)
```

Limit is used to fetch as many number of articles from the web.

Fig. 3 – Reading JSON file

Once the file is loaded, then we have used python's Article library to fetch the author name, title, content, published data, video links etc.

```
article['title'] = content.title
article['text'] = content.text
article['authors'] = content.authors
article['top_image'] =  content.top_image
article['movies'] = content.movies
newsPaper['articles'].append(article)
articles_array.append(article)
```

Fig. 4 – Article's information

Once the articles are fetched we have created another CSV file to store data.

```
#Finally it saves the articles as a CSV-file.
try:
    f = csv.writer(open('Scraped_data_news_output.csv', 'w', encoding='utf-8'))
    f.writerow(['Title', 'Authors','Text','Image','Videos','Link','Published_Date'])
    #print(article)
    for artist_name in articles_array:
        title = artist_name['title']
        authors=artist_name['authors']
        text=artist_name['text']
        image=artist_name['top_image']
        video=artist_name['movies']
        link=artist_name['link']
        publish_date=artist_name['published']
        # Add each artist's name and associated link to a row
        f.writerow([title, authors, text, image, video, link, publish_date])
except Exception as e: print(e)
```

Fig. 5 – Creating CSV file

4.2 Data Preprocessing - In Fig. 6, Stemming, Tokenization and Lemmatization has been implemented for text processing. First, we created the tokens then we pass them through lemmatization to get the root words and then used stemming to reduce inflected words.

```
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()
ps = PorterStemmer()


def lemmatize_text(text):
    return [lemmatizer.lemmatize(w) for w in w_tokenizer.tokenize(text)]

df['Title_lemmatized'] = df.TITLE.apply(lemmatize_text)
df['Content_lemmatized'] = df.Content.apply(lemmatize_text)

lemmatization_title=df['Title_lemmatized']
lemmatization_content=df['Content_lemmatized']

def title_stemming(text):
    l = []
    for i in text:
        for j in i:
            #converts list item to lower case
            p=j.lower()
            # removes punctuation,numbers and returns list of words
            q=re.sub('[^A-Za-z]+', ' ', p)
            l.append(ps.stem(q))
    return l
```

Fig. 6 - Stemming and Lemmatization

4.3 Count Vs No of Authors – The Fig. 7 shows the number of publications posted by an author.

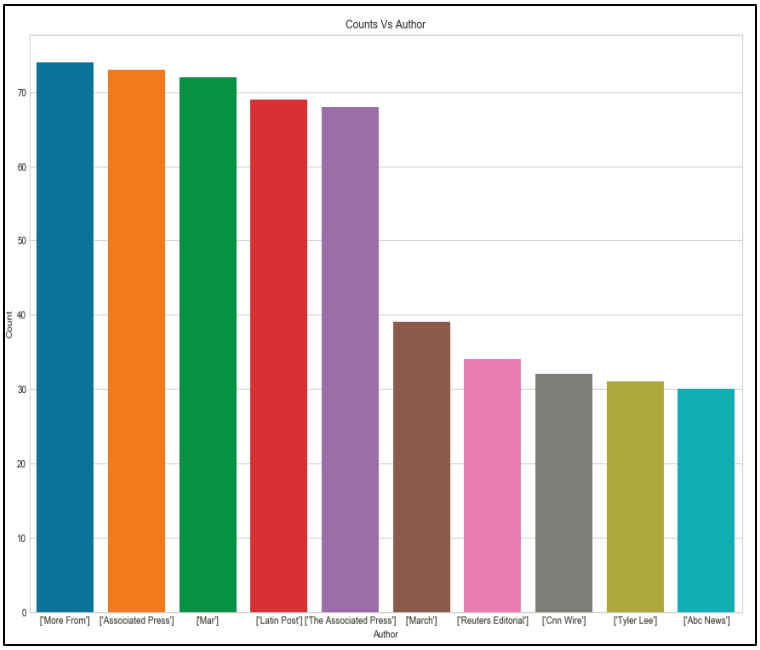

Fig. 7 – Articles published by an Author

4.4 Count Vs Category – The Fig. 8 shows the count of articles in a category. It can be concluded that there is more entertainment type news in our dataset as compared to other categories i.e. Health, Science & Tech. and Business.

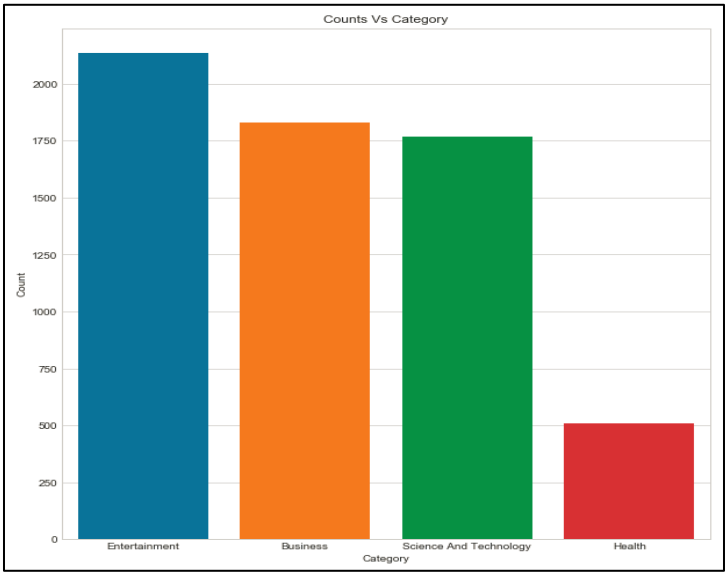

Fig. 8 – No. of Articles in a Category

4.5 Most occurring word in Title column – The Fig. 9 shows the most frequent words in Title column.

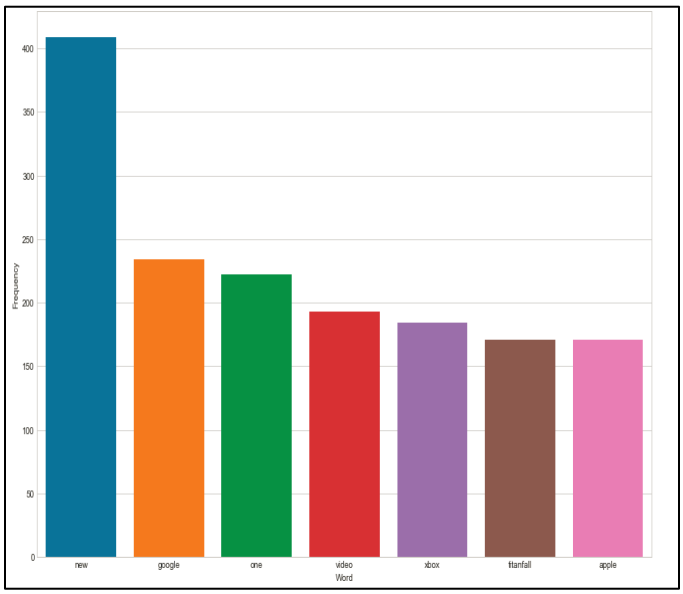

Fig. 9 – Frequent words in Title

4.6 Categorize Title into sentiments – The below code will categorize the title into sentiments based on the sentiment value.

```
#!pip install textblob

bloblist_desc = list()

df_review_str=df['TITLE'].astype(str)
for row in df_review_str:
    blob = TextBlob(row)
    bloblist_desc.append((row,blob.sentiment.polarity, blob.sentiment.subjectivity))
    df_polarity_desc = pd.DataFrame(bloblist_desc, columns = ['TITLE','sentiment','polarity

def f(df_polarity_desc):
    if df_polarity_desc['sentiment'] > 0:
        val = "Positive Title"
    elif df_polarity_desc['sentiment'] == 0:
        val = "Neutral Title"
    else:
        val = "Negative Title"
    return val
```

Fig. 10 - Title Sentiment code

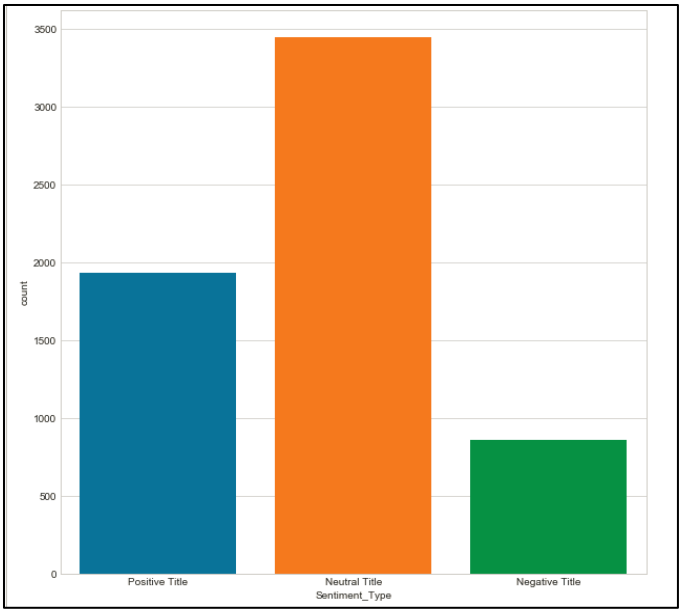

Fig. 11 – Category of sentiments of Title Column

The Fig. 11 shows Neutral Title are more in number as compared to positive and negative title.

4.7 Categorize content into sentiments - The Fig. 12 will categorize the content into sentiments based on the value.

```
bloblist_desc_content = list()

df_review_str_content=df['Content'].astype(str)
for row in df_review_str_content:
    blob_content = TextBlob(row)
    bloblist_desc_content.append((row,blob_content.sentiment.polarity, blob_content.sentiment.subjectivity
))
    df_polarity_desc_content = pd.DataFrame(bloblist_desc_content, columns = ['Content','sentiment','polar
ity'])

def f(df_polarity_desc_content):
    if df_polarity_desc_content['sentiment'] > 0:
        val = "Positive Content"
    elif df_polarity_desc_content['sentiment'] == 0:
        val = "Neutral Content"
    else:
        val = "Negative Content"
    return val

df_polarity_desc_content['Sentiment_Type'] = df_polarity_desc_content.apply(f, axis=1)
df_polarity_desc_content.head()
```

Fig. 12 – Categorizing content into different categories

4.8 Algorithms – We have used Naïve Bayes, Multi-Layer Perceptron and XG Boost classifier to classify the articles into sentiments.

4.8.1 Naïve Bayes Algorithm -

```
nb = MultinomialNB()
nb.fit(X_train, y_train)

predict_train = nb.predict(X_train)
predict_test = nb.predict(X_test)

recall_train_nb=f1_score(y_train, predict_train)
recall_test_nb=f1_score(y_test, predict_test)
```

Fig. 13 – Naïve Bayes Classifier

4.8.2 Multi-Layer Perceptron -

```
# Import Multi-Layer Perceptron Classifier Model
mlp = MLPClassifier(hidden_layer_sizes=(37,37,37))
mlp.fit(X_train,y_train)

predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)

recall_train_mlp=f1_score(y_train, predict_train)
recall_test_mlp=f1_score(y_test, predict_test)
```

Fig. 14 - ML Perceptron Classifier

4.8.3 XG Boost -

```
xgb=XGBClassifier()

xgb_fit=xgb.fit(X_train, y_train)

predict_train = xgb.predict(X_train)
predict_test = xgb.predict(X_test)

recall_train_xgb = f1_score(y_train, predict_train)
recall_test_xgb = f1_score(y_test, predict_test)
```

Fig. 15 – XG Boost Classifier

5. Results – After implementing and testing the above models to predict the sentiments below is the accuracy metrics. Our best model is XG Boost because another model is overfitted. The difference between the training data and testing data accuracy is more in Naïve Bayes and ML Perceptron.

	Naive Bayes_Model	MLP_Model	XGB_Model
Metrics_Train	0.97316	1.00000	0.86847
Metrics_Test	0.85101	0.85938	0.82984

Fig. 16 – Accuracy Metrics

6. Conclusion – As shown above, the presented study showed the three classification models i.e. Naïve Bayes, ML Perceptron and XG Boost. We achieved 83% accuracy of predicting the sentiment of a news article. There is still scope of improvement to learn the model based on the context so that it can predict more accurately for ex. 'the river bank' and 'the money bank'. In this example the bank word is same but the context is different.

7. Acknowledgments – We'd like to thank our professor, teaching assistants, family and friends who supported in the completion of this research project. Appreciating everyone who helped us knowingly or unknowingly for this project.

8. References –

1. http://newspaper.readthedocs.io/en/latest/
2. https://www.geeksforgeeks.org/newspaper-article-scraping-curation-python/
3. https://www.smallsurething.com/web-scraping-article-extraction-and-sentiment-analysiswith-scrapy-goose-and-textblob/
4. https://github.com/nikbearbrown
5. http://newspaper.readthedocs.io/en/latest/ http://textblob.readthedocs.io/en/dev/quickstart.html