Capstone 2 Milestone Report

Priya Sathish

Content based News Recommendation System

Problem

Online platforms support so many of our daily activities that we have become dependent on them in our personal and professional lives. We rely on them to buy and sell goods and services, to find information online and to keep in touch with each other. These platforms could help consumers by recommending items as per their interest and preference by just analyzing your past interaction or behavior with the system. From Amazon to LinkedIn, Uber eats to Spotify, Netflix to Facebook, Recommender systems are most extensively used to suggest "Similar items", "Relevant jobs", "preferred foods", "Movies of interest" etc. to their users. Recommender system with appropriate item suggestions helps in boosting sales, increasing revenue, retaining customers and also adds competitive advantage. Recommender systems use a number of different technologies and can be classified into two broad groups as Content based recommendation and Collaborative filtering.

On a day to day basis, the internet has a lot of sources that generate immense amount of daily news diversified in subject matter. There is continuous demand for new information to be available immediately and with ease by the consumers. So, it is crucial that the news is classified and targets the needs and requirements of the user effectively and efficiently. News services have attempted to identify articles of interest to readers based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the millions of blogs available or other sites where content is provided regularly.

This project focuses on content-based recommendation using News category dataset. The goal is to recommend news articles which are similar to the already read article by using attributes like article headline, short description, category, author and publishing date.

Client

News readers, blog readers, news agencies, bloggers, retailers and several online platforms.

Data and approach

https://www.kaggle.com/rmisra/news-category-dataset

This dataset contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost. News in this dataset belongs to 41 different categories. Each news record consists of a headline with a short description in our analysis. In addition, we will combine attributes 'headline' and 'short description' into a single attribute 'text' as the input for classification and proceed with developing a deep learning model to build the recommender system.

Citation: "https://rishabhmisra.github.io/publications/"

Data Wrangling

Acquired the news category dataset from kaggle as a json file and converted it to a pandas dataframe with a shape of (200853, 6) for analysis. When grouped by category we can see that the dataset contains 41 categories of news articles.
'THE WORLDPOST' and 'WORLDPOST' should be the same category, so we change 'THE WORLDPOST' to 'WORLDPOST' and merge them.

After which we have,
Total number of articles: 200853
Total number of authors: 27993
Total number of unique categories: 40

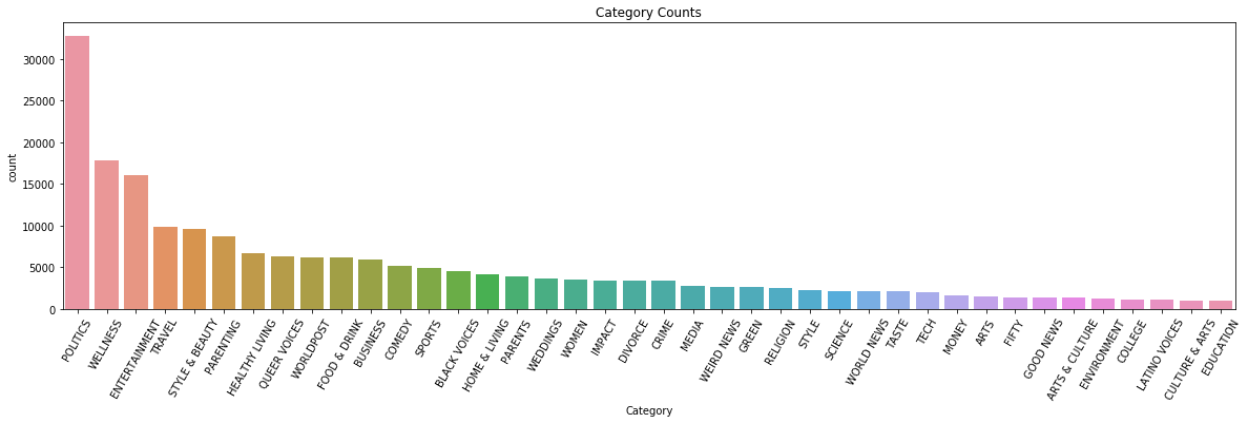
Top 5 categories include:

category	
POLITICS	32739
WELLNESS	17827
ENTERTAINMENT	16058
TRAVEL	9887
STYLE & BEAUTY	9649

Also, we will check for any missing data,

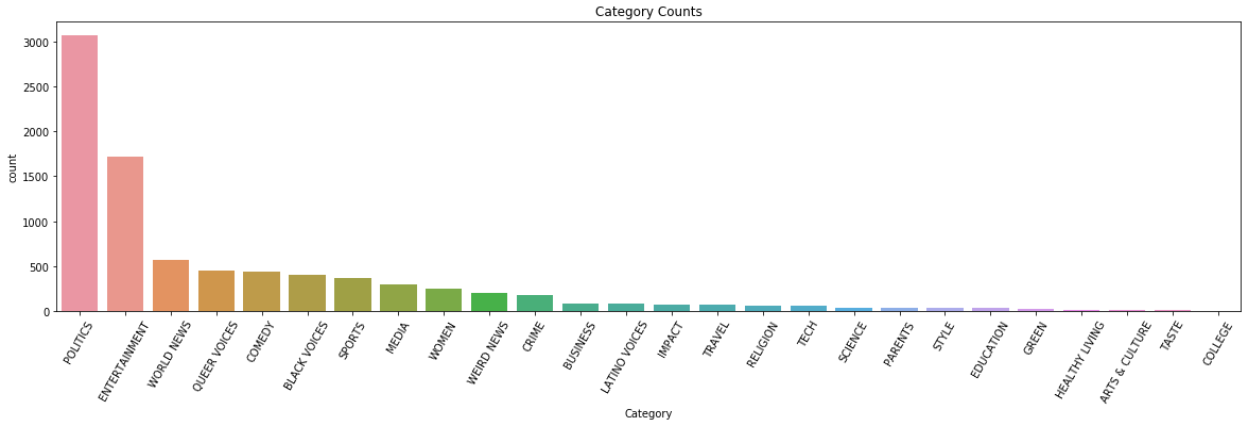
category	0
headline	0
authors	0
link	0
short_description	0

date 0
text 0
words 0
word_length 0



From the plot above we observe that politics, wellness, entertainment, travel and beauty form the top 5 categories of news article headlines during the period of 2014-2018.

We will consider only the latest articles from the year 2018 as the size of the dataset is quite large and processing may consume too much time. So, we filter the dataset to contain only row from the year 2018 and will proceed with text preprocessing.



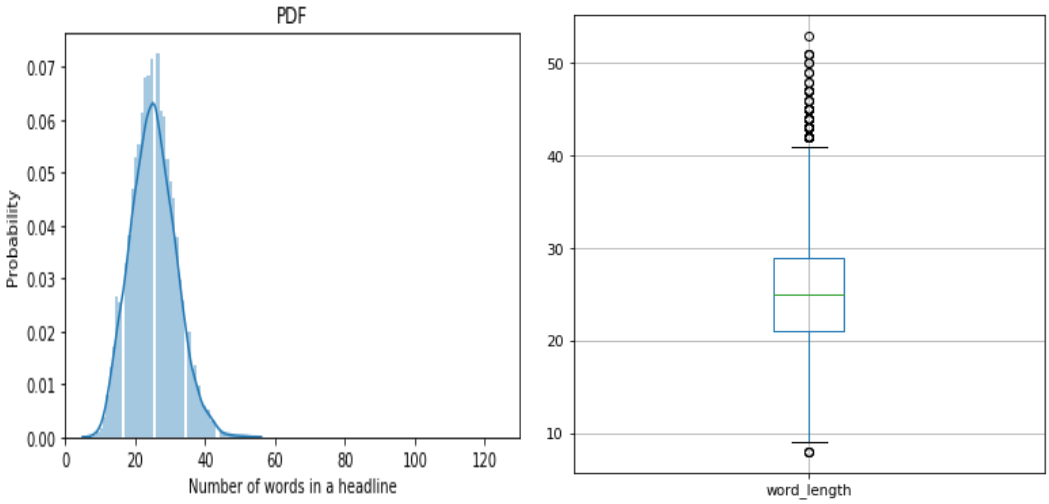
From the plot above we can see that politics, entertainment, world news, queer voices and comedy form the top 5 categories in the year 2018.

We will be using headlines and short description as input X. So, we will combine the short description and headline by concatenation and create a new column.

We tokenize the headlines and then we delete some empty and short data with word length less than 5. We take a look at the word length distribution:

Count 8583.000000
mean 25.176861
std 6.418315
min 8.000000
25% 21.000000
50% 25.000000
75% 29.000000
max 53.000000
Name: word_length, dtype: float64

We create a PDF and box plots to view the word length distribution.



From the plot above we can clearly see that the word length ranges between 8 and 53 with a mean value of 25 and standard deviation of about 6 over 8583 news articles.

Text Preprocessing

We will clean and process the text data so that it is ready for modeling using the Natural Language Toolkit or NLTK Python library.

Tokenize:
We will split the news headline text into tokens based on white space or punctuation. This is considered as a base step for stemming and lemmatization. Once we use word_tokenize() on the headline text we can use the output for stop words removal.

Stop words removal:
Stop words are not much helpful in analysis and also their inclusion consumes much time during processing so let's remove these. We will use the default NLTK corpus to remove the unwanted stop words.

Lemmatize:
Lemmatization is the algorithmic process of finding the lemma of the word depending on the meaning. We will use the WordNetLemmatizer() from the NLTK Python library to remove inflectional endings.

To perform all of the above-mentioned operations on our headline text we will define a function to process:

```
def process_headlines(main_text):
    headlines_without_numbers = re.sub('[^a-zA-Z]', ' ', main_text)
    words = word_tokenize(headlines_without_numbers.lower())
    stop_words_english = set(stopwords.words('english'))
    final_words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words_english]
    return(' '.join(final_words))
```

Bag Of Words – Count Vectorize

To extract features from the text documents and create a vocabulary of all the unique words in all of the news headlines we will use CountVectorizer() from the NLTK Python library. The BOW model only considers if a known word occurs in a document or not. It does not care about meaning, context and order in which they appear. This gives the insight that similar documents will have word counts similar to each other. In other words, the more similar the words in two documents, the more similar the documents can be. However, there are some limitations using this method such as it doesn't take the semantic meaning or context into account and also if the vector size is huge it might result in lot of computation and time.

Preliminary model evaluation using default parameters

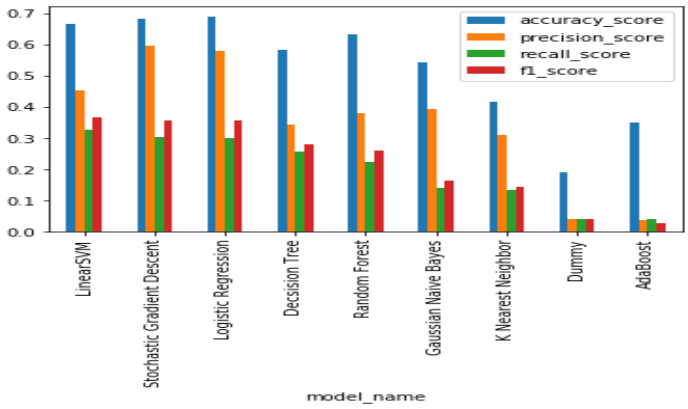
Now that we have preprocessed our text, we will try several kinds of classifiers and compare them with their default parameters. The problem we have here is that the algorithm may not perform well right away but might perform really well with the right set of hyperparameters. To get a preliminary understanding of which types of classifiers will inherently work better we will compare them.

We will take a look at 8 different classifiers along with sklearn's dummy classifier which is just a random baseline.
The classifiers we will compare include:
Dummy Classifier, Stochastic Gradient Descent, RandomForestClassifier, DecisionTreeClassifier, AdaBoostClassifier, Gaussian Naive Bayes, LogisticRegression, LinearSVM and K Nearest Neighbor.

The metrics we will use to evaluate the different classifiers are:
Accuracy – the fraction of samples predicted correctly
Precision – the ratio of true positives to false positives (ability of the classifier not to label a positive as a negative sample)
Recall – the ratio of true positives to false negatives (ability of the classifier to find all the positive samples)
F1 Score – the harmonic average of precision and recall (we will use macro averaging which will compute the average of the F1 scores)

Comparison Results:

model_name	accuracy_score	precision_score	recall_score	f1_score
LinearSVM	0.666078	0.452625	0.328205	0.36622
Stochastic Gradient Descent	0.682316	0.59672	0.305721	0.358697
Logistic Regression	0.689375	0.581914	0.301341	0.356729
Decsision Tree	0.584539	0.345634	0.257183	0.28229
Random Forest	0.633251	0.379824	0.225787	0.259581
Gaussian Naive Bayes	0.545358	0.395019	0.141648	0.164515
K Nearest Neighbor	0.417226	0.311188	0.133848	0.144641
Dummy	0.190964	0.0400646	0.0404579	0.040161



From the above plot and the F1 scores we can see that Linear SVM (0.37), Stochastic Gradient (0.36) Descent and Logistic Regression (0.36) performed better than all other classifiers. The F1 scores for the top three classifiers are also not that great though the real test is how they perform on unseen articles.

The scores were calculated only the news articles from the year 2018. The classifiers might have performed well if we had used the news articles from the years 2014-2018.

Future Work

1. To overcome the limitations of Bag Of Words method, we will implement other embedding techniques like Glove Embedding or Keras Embedding.
2. Will perform hyperparamter tuning on the top 3 classifiers we compared.
3. Implement deep learning models and compare performance.
4. Build a recommendation system with combinations of different features including headline, category, author and published date.