Sentiment Analysis in IMDB reviews dataset

ARPAN POUDEL: 010991022

JARED HARRIS: 010792702

PRIYANKA INGALE: 010986084

Introduction

- Various e-commerce business(for e.g., Amazon, Walmart, IMDB) has reviews and stars ratings of products, available for buyer know about the product.
- With the use of Natural language processing, we can analyze the sentiment of user's review and categorize whether the user is giving a positive review or negative review.
- An accurate description of the product can be depicted by implementing the sentiment analysis on reviews from the user.
- Combining star ratings and review sentiment can better rate the product than alone star rating

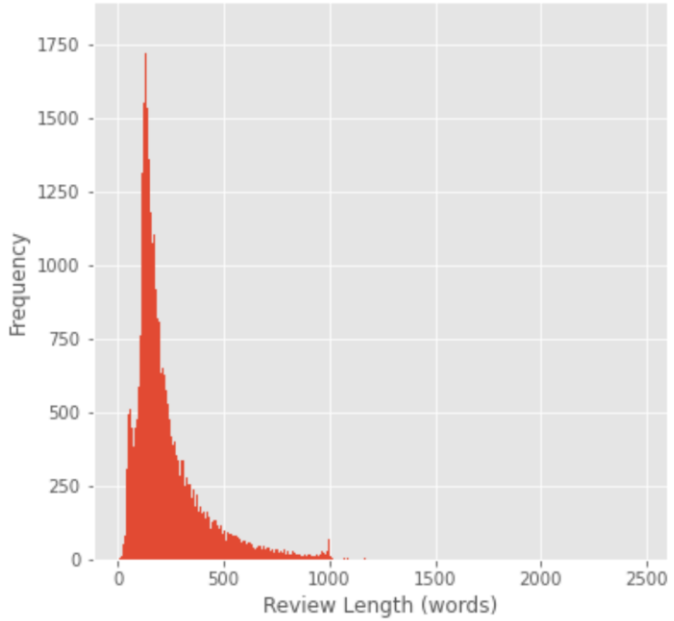
Dataset

- Dataset is taken from Kaggle which consists of 50,000 movie reviews (input variables) and labels as the sentiment (Positive and Negative Review).

```
IMDB_reviews = pd.read_csv('IMDB.csv')
IMDB_reviews.head(3)
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive

- Balanced dataset i.e.. 25000 positive reviews and 25000 negative reviev
- The average words for each review is 231 and the skewness is positive.



Word Cloud Visualization

fig: word cloud for all dataset

Positive Sentiment

fig: word cloud for Positive dataset

Negative Sentiment

fig: word cloud for Negative dataset

Data Preprocessing

- Dataset consists of html codes, emoji, chatwords which doesn't provide sentiments to the model.
- Dataset also consists of punctuation, stop words which needs to be removed.
- Stemming and lemmatization, which is basically the process of bringing words to their word stem is done to avoid redundant word. For example: runner, running, runs converted to run.
- Cleaned dataset is saved for future analysis using machine learning models.

One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exa ctly what happened with me.

The first thing that struck me about Oz was its brutality and unflinching scenes of viol ence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punc hes with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.

It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental sect ion of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.

I would say the main appeal of the show is due to the fact that it goes whe

fig: text before data preprocessing

one reviewer mention watch 1 oz episode hook right exactly happen first thing strike oz brutality unflinche scene violence set right word go trust show faint hearted timid show pull punch regard drug sex violence hardcore classic use word call oz nicknam e give oswald maximum security state penitentary focus mainly emerald city experimental section prison cell glass front face in wards privacy high agenda em city home many aryans muslims gangstas latinos christians italians irish scuffle death stare dodgy dealing shady agreement never far away would say main appeal show due fact go show dare forget pretty picture paint mainstream audience forget charm forget romance oz mess around first episode ever see strike nasty surreal say ready watch develop taste o z got accustom high level graphic violence violence injustice crooked guard sell nickel inmate kill order get away well mannere d middle class inmate turn prison bitch due lack street skill prison experience watch oz may become comfortable uncomfortable v iewing that s get touch dark side

fig: text after data preprocessing

Feature extraction

- Text data are converted higher dimensional containers (matrices) called Vectorization.
- Two types of Vectorization : Vectorize without semantics and Retain Semantic Importance
- Vectorize without semantics includes Count vectorization, TFIDF vectorization.
- Retaining semantic importance uses methods like word embedding and word2vec methods.

TF-IDF vectorization

- TFIDF stands for Term Frequency – Inverse Document Frequency which is the product of term frequency and inverse document frequency.
- Term frequency, *tf(t,d)*, is the relative frequency of term *t* within document d.
- Inverse document frequency is a measure of how much information the word provides

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Model with TF-IDF

Model	Accuracy	Precision	Recall
Support Vector Machine	0.89	0.9	0.88
Random Forest	0.85	0.85	0.85
AdaBoost (n=50, alpha=1.0)	0.80	0.80	0.80
Multinomial Naïve Bayes	0.86	0.86	0.86
Decision Tree (No Max Depth)	0.71	0.71	0.71
Decision Tree (Max Depth = 2)	0.69	0.72	0.69
Decision Tree (Max Depth = 15)	0.74	0.75	0.74

Testing

- Testing with user input as a TF-IDF vector

```
print(text_clf.predict(["This is one of the best i have seen"]))
```

```
[1]
```

```
print(text_clf.predict(["The character are one of the worst actor. Story is terrible"]))
```

```
[0]
```

Implementation and Output

In [52]:
```
#Decision Tree Classifier with Max Depth Set to 15 Using TFIDF Vectorization
#Import sklearn libraries
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
DecisionTreeClf = Pipeline([('tfidf',TfidfVectorizer()),('clf',DecisionTreeClassifier(max_depth=15))])
DecisionTreeClf.fit(X_train,y_train)
DecisionTreePrediction = DecisionTreeClf.predict(X_test)
print("Confusion Matrix")
print(metrics.confusion_matrix(y_test,DecisionTreePrediction))
print("")
print("Classification Report")
print(metrics.classification_report(y_test,DecisionTreePrediction))
print("")
print("Accuracy Score")
print(metrics.accuracy_score(y_test,DecisionTreePrediction))
```

```
Confusion Matrix
[[5322 2886]
 [1402 6890]]

Classification Report
              precision    recall  f1-score   support

           0       0.79      0.65      0.71      8208
           1       0.70      0.83      0.76      8292

    accuracy                           0.74     16500
   macro avg       0.75      0.74      0.74     16500
weighted avg       0.75      0.74      0.74     16500


Accuracy Score
0.7401212121212121
```

Word2vec

- Word embedding maps the word to different dimension vectors.
- It takes an input a very large corpus of text and produces a vector space, each unique word in the corpus assigned equivalent vector in the space.
- We have used a pertained model “globe wiki gigaword” which generates a 200 dimension vector for each word, each dimension representing different features of the word.

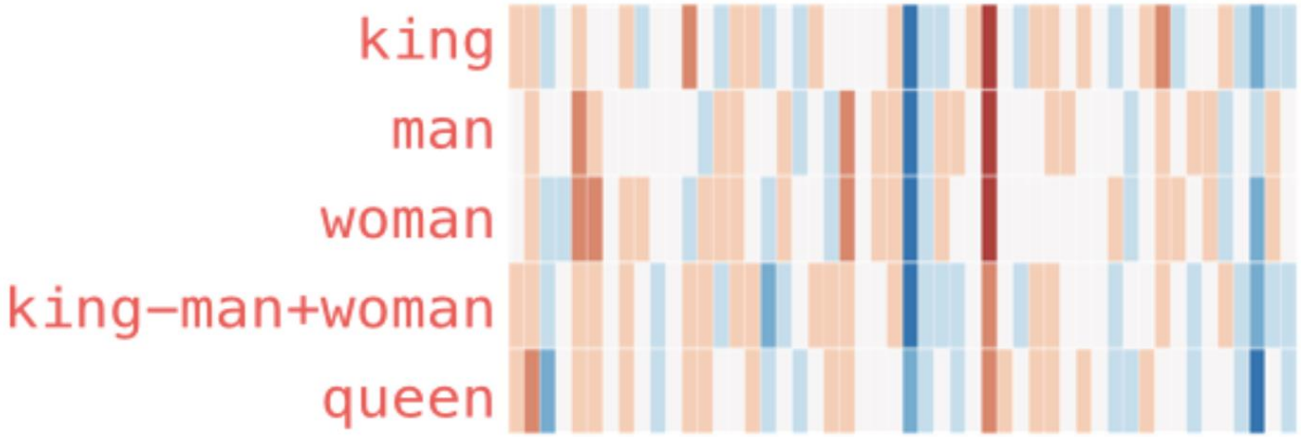


Fig: word embedding for different words

Word2 vec

	0	1	2	3	4	5	6	7	8	9	...	191	192	193	194	
0	0.069268	0.037514	-0.053542	-0.049320	-0.015105	0.033259	-0.130886	0.034476	0.016125	0.065838	...	0.002814	-0.011527	-0.001014	0.028099	-0.024
1	0.073270	0.043425	-0.038058	-0.065127	0.001628	-0.012296	-0.122431	0.043278	0.045777	0.007610	...	0.012166	-0.024579	-0.009322	0.073293	-0.037
2	0.090271	0.012117	-0.082555	-0.074651	-0.001243	-0.002296	-0.124495	0.012805	0.025897	0.054319	...	-0.006658	-0.016111	-0.024595	0.056925	-0.017
3	0.055893	0.070601	-0.089779	-0.074793	-0.007383	-0.007459	-0.076456	-0.007977	0.013484	0.051103	...	-0.057887	-0.055399	-0.049412	0.046975	-0.036
4	0.063769	0.056531	-0.030135	-0.045891	0.011218	0.013602	-0.127557	0.025569	0.030434	0.059601	...	0.002768	-0.039307	-0.005470	0.050983	-0.023
5	0.062888	0.046756	-0.085934	-0.094994	-0.029447	-0.004513	-0.103569	0.003706	0.040635	0.074384	...	0.012208	-0.033047	-0.014406	0.057637	-0.033
6	0.034966	0.077466	-0.046753	-0.040479	0.013124	0.022341	-0.109160	0.014689	0.026323	0.084263	...	0.037809	-0.011513	0.001634	0.009411	-0.023
7	0.026914	0.045093	-0.059812	-0.047826	-0.007287	-0.012112	-0.107862	0.005265	0.042833	0.053549	...	0.026987	-0.047586	0.004863	0.046904	-0.055
8	0.048272	0.046228	-0.076962	-0.062689	0.000329	-0.005977	-0.108111	-0.000688	0.057644	0.062473	...	-0.028314	-0.049712	-0.020739	0.108059	-0.026
9	0.070217	0.020926	-0.090004	-0.048349	-0.002648	0.039678	-0.109528	0.007314	-0.011309	0.072081	...	-0.015354	-0.005348	-0.022155	0.051273	-0.065

Model with Word2vec

Model	Accuracy	Precision	Recall
Logistic Regression	0.813	0.81	0.82
Adaboost (n=50, alpha=0.5)	0.76	0.77	0.77
Adaboost (n=100, alpha=1)	0.78	0.79	0.78

Challenges

- Memory problem while creating a model to generate the word2vec without using pertained model.
- Training time was very high for each models.

Future Works

- Implement LSTM or BERT to the dataset and compare with traditional NLP technique.
- Model selection.
- Check our best model with other reviews dataset.