

Project 2 – Text Mining

SENTIMENTAL ANALYSIS ON IMDB MOVIE REVIEW

SENTIMENT ANALYSIS



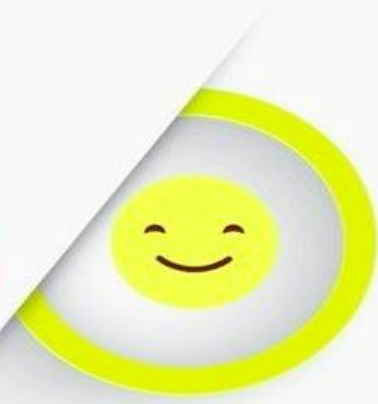
NEGATIVE

Totally dissatisfied with the service. Worst customer care ever.



NEUTRAL

Good Job but I will expect a lot more in future.



POSITIVE

Brilliant effort guys! Loved Your Work.

Group 3

Andreas Koumanakos, Dwayne Simms, Manan Alam,
Khushboo Singhal, Tsegaye Mekonnen

Table of Contents

	0
ABSTRACT	3
PROBLEM STATEMENT	4
ETHICAL ML CONSIDERATIONS	5
WHAT IS SENTIMENTAL ANALYSIS	6
But why do we need sentiment analysis?	6
How does sentiment analysis work?	6
UNDERSTANDING & LOADING DATASET	8
Missing data:	8
Removing duplicates:	9
Pre - Processing Text	9
HTML Parsing:	10
Special Characters:	10
Stemming words with NLTK:	10
Stop word removal:	11
Word vectorization	12
TF-IDF	12
Bag of Words:	13
DATA EXPLORATION	15
PROPOSED APPROACH	19
Algorithms	19
Pre-processing methods	19
Exporting methods	19
FEATURE ENGINEERING	20
TRAINING MODELS	21
Logistic Regression:	21
Decision Tree Classifier	22
Random Forest Classifier	23
Naive Bayes Classifier	24

Summary:	24
<i>GENERATE THE PICKLE</i>	26
<i>DEPLOYMENT</i>	27
App Url: https://moviesentiment2.herokuapp.com/	28
<i>SUMMARY</i>	30

ABSTRACT

IMDb is one of the most popular online databases for movies and personalities. It's a platform where millions of users, and professional critics read and write movie reviews. This provides a large and diverse dataset for sentiment analysis. In this project, we were tasked to implement different classification models to predict the sentiment of IMDb reviews, either as positive or negative, using only text each review contains. The goal is to find the model with the highest F1 score and best generalization. We trained different models using multiple combinations of text features and hyper-parameter settings for both the classifiers and the features, which we found could potentially impact the performance significantly. Every model was evaluated by k-fold cross validation to ensure the consistency of their performance. We found that our best performing model was the Logistic Regression with a bag of words, which reported an accuracy score of 89% on the final test set.

PROBLEM STATEMENT

IMDB/Amazon want to collect catalogs of movies that users will want to watch. By collecting reviews data we are able to indicate which movie viewers associate positivity and negativity. This would be a strategic and data driven process when purchasing rights to new movies. Identifying sentiment of already available reviews help IMDB to further tasks such as recommendation engine, movie listing and general acceptability.

With the help of Text Mining Techniques we want to perform Sentiment Analysis on an IMDB dataset. Our main goal here is to estimate the sentiment of movie reviews for analytical purposes so backend management can enhance customer satisfaction and business decisions.

Everyday a large segment of people visit IMDB to look for the reviews and decide if a movie is worth watching or not. Sometimes people let their inner movie critic loose and post long reviews for the movies they watch. With that said, people who look for reviews usually read only the first 2-3 lines and then decide to watch a movie or not. Due to this fact, we want to solve a very generic problem where a user can determine the sentiment of the movie review and make a quick decision rather than reading those long movie reviews.

ETHICAL ML CONSIDERATIONS

With the creation of this report, the deployment of our app, and models we made sure we follow the Aletheia Framework. We are certain that the Social Impact, Accuracy and Trust, and finally Data Governance pillars of the previously mentioned framework were followed to a great extent.

Our models use pre classified reviews as positive or negative to train our models, so to minimize profanity, and inappropriate language we recommend pre proceeding of text and neutralization. We take no pre-processing of such a task since our dataset has come from a trusted source preprocessed and listed on Kaggle.

On the other hand our system can “understand” the impact of words, sentiment analysis of review will not be directly viable on the website. Reviews will be more accessible for backend clients, there is easier filtration of reviews and follow up decisions to a movie that has the most negative sentiments. As far as the Accuracy and Trust is concerned, we made sure that we eliminate any sort of bias by selecting random texts for machine learning algorithms. IMDB dataset is transparent and traceable. The algorithms and training of the model is clearly stated.

Users write reviews on their own recognisance, on a public site with informed consent that once posted that data would be used for which way the site owners chooses. No harm or repercussions during the collection or storage of data, outside of the standards site posting requirements. The Data collected holds no personal data to the user, names, age or ethnicity providing the highest level of anonymity. Which allows no bias when data is run through the model. Parameters were set during word vectorization and algorithm fitting to define reviews as negative or positive based on standard dictionary meaning allowing no input for biases. The algorithmic process used to define sentiments positivity and negativity value didn't allow for any added decision-making. The operation was purely assigning to a value the treatment was basic.

Last but not least we have Data Governance. As mentioned prior, we are using reviews that are posted on IMDb, hence the data we are using are internal. There is no confidential information. With that stated, our findings, reports and models follow every pillar of the Aletheia framework.

WHAT IS SENTIMENTAL ANALYSIS

Sentiment analysis or opinion mining is a simple task of understanding the emotions of the writer of a particular text. What was the intent of the writer when writing a certain thing?

We use various natural language processing (NLP) and text analysis tools to figure out what could be subjective information. We need to identify, extract and quantify such details from the text for easier classification and working with the data.

But why do we need sentiment analysis?

Sentiment analysis serves as a fundamental aspect of dealing with customers on online portals and websites for the companies. They do this all the time to classify a comment as a query, complaint, suggestion, opinion, or just love for a product. This way they can easily sort through the comments or questions and prioritize what they need to handle first and even order them in a way that looks better. Companies sometimes even try to delete content that has a negative sentiment attached to it.

It is an easy way to understand and analyze public reception and perception of different ideas and concepts, or a newly launched product, maybe an event or a government policy.

Emotion understanding and sentiment analysis play a huge role in collaborative filtering-based recommendation systems. Grouping together people who have similar reactions to a certain product and showing them related products. Like recommending movies to people by grouping them with others that have similar perceptions for a certain show or movie.

Lastly, they are also used for spam filtering and removing unwanted content.

How does sentiment analysis work?

NLP or natural language processing is the basic concept on which sentiment analysis is built upon. Natural language processing is a superclass of sentiment analysis that deals with understanding all kinds of things from a piece of text.

NLP is the branch of AI dealing with texts, giving machines the ability to understand and derive from the text. For tasks such as virtual assistant, query solving, creating and maintaining human-like conversations, summarizing texts, spam detection, sentiment analysis, etc. it includes

everything from counting the number of words to a machine writing a story, indistinguishable from human texts.

Sentiment analysis can be classified into various categories based on various criteria. Depending upon the scope it can be classified into document-level sentiment analysis, sentence level sentiment analysis, and sub sentence level or phrase level sentiment analysis.

Also, a very common classification is based on what needs to be done with the data or the reason for sentiment analysis. Examples of which are

- Simple classification of text into positive, negative or neutral. It may also advance into fine grained answers like very positive or moderately positive.
- Aspect-based sentiment analysis- where we figure out the sentiment along with a specific aspect it is related to. Like identifying sentiments regarding various aspects or parts of a car in user reviews, identifying what feature or aspect was appreciated or disliked.
- The sentiment along with an action associated with it. Like mails written to customer support. Understanding if it is a query or complaint or suggestion etc

Based on what needs to be done and what kind of data we need to work with there are two major methods of tackling this problem.

- Matching rules based sentiment analysis: There is a predefined list of words for each type of sentiment needed and then the text or document is matched with the lists. The algorithm then determines which type of words or which sentiment is more prevalent in it. This type of rule based sentiment analysis is easy to implement, but lacks flexibility and does not account for context.
- *Automatic sentiment analysis*: They are mostly based on supervised machine learning algorithms and are actually very useful in understanding complicated texts. Algorithms in this category include support vector machine, linear regression, rnn, and its types. This is what we are gonna explore and learn more about.

UNDERSTANDING & LOADING DATASET

The IMDB Movie Ratings Sentiment Analysis dataset was published by M Yasser H

(<https://www.kaggle.com/yasserh>) on Kaggle for public use. It is 52.72 MB in size with 40,000 records and 2 columns with the review text and sentiment level as 0 - for negative and 1 - positive reviews in a csv file.

```
#Importing the dataset
df = pd.read_csv('movie.csv', header=0)
target = 'label'
df.reset_index(drop=True, inplace=True)
original_df = df.copy(deep=True)
display(df.head(6))

print('\n\033[1mInference:\033[0m The Dataset consists of {} features & {} samples.'.format(df.shape[1], df.shape[0]))
```

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1
5	A terrible movie as everyone has said. What ma...	0

Inference: The Dataset consists of 2 features & 40000 samples.

The data has almost the same number of reviews for Negative and Positive reviews.

```
df.label.value_counts()
```

```
0    20019
1    19981
Name: label, dtype: int64
```

Missing data:

There are no missing labels in the dataset which makes it easier to just get started working the dataset easier.

But as we can see on the screenshot below we can see a few duplicate reviews in the dataset.

```
#Checking the stats of all the columns
```

```
display(df.describe(include='all'))
```

	text	label
count	40000	40000.000000
unique	39723	NaN
top	Hilarious, clean, light-hearted, and quote-wor...	NaN
freq	4	NaN
mean	NaN	0.499525
std	NaN	0.500006
min	NaN	0.000000
25%	NaN	0.000000
50%	NaN	0.000000
75%	NaN	1.000000
max	NaN	1.000000

```
#Check for empty elements
```

```
print(df.isnull().sum())
```

```
print('\n\033[1mInference:\033[0m The dataset doesn\'t have any null elements')
```

```
text      0
label     0
dtype: int64
```

Inference: The dataset doesn't have any null elements

Removing duplicates:

```
#Removal of any Duplicate rows (if any)
```

```
counter = 0
```

```
r,c = original_df.shape
```

```
df1 = df.drop_duplicates()
```

```
df1.reset_index(drop=True, inplace=True)
```

```
if df1.shape==(r,c):
```

```
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
```

```
else:
```

```
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed ---> {r-df1.shape[0]}')
```

Inference: Number of duplicates dropped/fixed ---> 277

We have dropped 277 records so that our machine learning model is not biased due to duplicate information.

Pre - Processing Text

Our data has a lot of jargon text since it was collected from the web. We created a custom function to filter the text among html tags and remove html tags in the data (e.g.

 <p> Good movie </p>). There are open and closed brackets which we have removed using regular expressions. BeautifulSoup is a python

package which helps to parse html content and export the specified text only. Regular expressions are the primary tools for text processing at a high level introduced in Datacamp introduction to NLP content.

HTML Parsing:

```
# Pre Processing the text to remove unnecessary content
# e.g. <br> <br /> <p></p>

def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^\]]*\]', '', text)

#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text

#Apply function on review column
df1['text']=df1['text'].apply(denoise_text)
```

Special Characters:

```
#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'^a-zA-z0-9\s'
    text=re.sub(pattern,'',text)
    return text

#Apply function on review column
df1['text']=df1['text'].apply(remove_special_characters)
```

Stemming words with NLTK:

Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words “chocolates”, “chocolatey”, “choco” to the root word, “chocolate” and “retrieval”, “retrieved”, “retrieves” reduce to the stem “retrieve”.

After lowercasing each word we use NLTK Snowballstemmer, which is an API interface used to remove morphological affixes from words, leaving only the word stem. This stemming algorithms aim to remove those affixes required for eg. grammatical role, tense, derivational morphology leaving only the stem of the word.

E.g. Some more example of stemming for root word "like" include:

-> "likes"

-> "liked"

-> "likely"

-> "liking"

Here is the preprocessing function defined to do stemming and stop word removal.

```
: #Stemming the text

from nltk.stem import SnowballStemmer

def simple_stemmer(text):
    snowball = SnowballStemmer(language='english')
    text= ' '.join([snowball.stem(word) for word in text.split()])
    return text
#Apply function on review column
df1['text']=df1['text'].apply(simple_stemmer)
```

Stop word removal:

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words. These stopwords are commonly used words (such as “the”, “a”, “an”, “in”) that do not affect the sentiment of the review text.

Stop word remove

```
#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
df1['text'] =df1['text'].apply(remove_stopwords)

{'itself', 'down', 'there', 'with', 'through', 'against', 'and', 'mightn', 'am', 'hadn't', 'for', 'how', 'shan't', 'theirs', 'the', 'mustn't', 'during', 'ain', 'at', 'again', 'that'll', 'here', 'if', 'between', 'wasn't', 's', 'herself', 've', 'don't', 'no', 'you'll', 're', 'being', 'mightn't', 'haven't', 'has', 'hadn', 'you', 'was n', 'd', 'you'd', 'out', 'into', 'only', 'can', 'some', 'be', 'doesn', 'is', 'hasn', 'should've', 'an', 'couldn't', 'yourselves', 'had', 'did', 'but', 'won', 'who', 'each', 'himself', 'not', 'y', 'this', 'too', 'to', 'few', 'do', 'your', 'their', 'our', 'after', 'below', 'once', 'isn', 'she', 'didn', 'by', 'she's', 'all', 'what', 'a', 'have', 't', 'm', 'doing', 'you're', 'that', 'because', 'was', 'we', 'him', 'own', 'll', 'where', 'then', 'most', 'having', 'shouldn't', 'off', 'over', 'furthe r', 'when', 'about', 'haven', 'which', 'he', 'above', 'more', 'very', 'does', 'hers', 'before', 'from', 'aren't', 'now', 'couldn', 'shouldn', 'doesn't', 'these', 'do n', 'ours', 'on', 'so', 'of', 'in', 'both', 'it', 'won't', 'been', 'hasn't', 'o', 'mustn', 'until', 'as', 'ourselves', 'same', 'other', 'or', 'those', 'should', 'ar e', 'wouldn', 'just', 'will', 'up', 'weren', 'they', 'themselves', 'while', 'its', 'any', 'yours', 'her', 'i', 'shan', 'myself', 'aren', 'it's', 'my', 'such', 'unde r', 'needn't', 'yourself', 'why', 'whom', 'isn't', 'ma', 'his', 'needn', 'didn't', 'you've', 'weren't', 'me', 'them', 'wouldn't', 'nor', 'were', 'than'}
```

As we can see in the above diagram we cross checked each word within NLTK's english list of stop words so that it won't take up space in our text corpus, or take up valuable processing time.

Word vectorization

Tokenizing converts all of the sentences/phrases/etc into a series of words, and then it might also include converting it into a series of numbers - math stuff only works with numbers, not words. Each of the resulting small units are called tokens. There are numerous uses of doing this. We can use this tokenized form to:

- Count the number of words in the text
- Count the frequency of the word, that is, the number of times a particular word is present

And so on. We can extract a lot more information which we'll discuss in detail in future articles. For now, it's time to dive into the meat of this article – the different methods of performing tokenization in NLP.

TF-IDF

Term frequency-inverse document frequency is a text vectorizer that transforms the text into a usable vector. It combines 2 concepts, Term Frequency (TF) and Document Frequency (DF).

The term frequency is the number of occurrences of a specific term in a document. Term frequency indicates how important a specific term in a document. Term frequency represents every text from the data as a matrix whose rows are the number of documents and columns are the number of distinct terms throughout all documents.

Document frequency is the number of documents containing a specific term. Document frequency indicates how common the term is.

Inverse document frequency (IDF) is the weight of a term, it aims to reduce the weight of a term if the term's occurrences are scattered throughout all the documents. IDF can be calculated as follow:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

Where idf_i is the IDF score for term i , df_i is the number of documents containing term i , and n is the total number of documents. The higher the DF of a term, the lower the IDF for the term. When the number of DF

is equal to n which means that the term appears in all documents, the IDF will be zero, since $\log(1)$ is zero, when in doubt just put this term in the stopword list because it doesn't provide much information.

The TF-IDF score as the name suggests is just a multiplication of the term frequency matrix with its IDF, it can be calculated as follow:

$$w_{i,j} = tf_{i,j} \times idf_i$$

Where w_{ij} is TF-IDF score for term i in document j , tf_{ij} is term frequency for term i in document j , and idf_i is IDF score for term i .

It is included within the sklearn feature extraction :

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf=TfidfVectorizer(strip_accents=None,lowercase=False,preprocessor=None,use_idf=True,norm='l2',smooth_idf=True)
tfidf.fit_transform(df.text)

y=df.label.values
x=tfidf.fit_transform(df.text)
```

```
df.text
```

```
0      grew b watch love thunderbird mate school watc...
1      put thi movi dvd player sat coke chip expect w...
2      whi peopl know particular time past wa like fe...
3      even though great interest biblic movi wa bore...
4      im die hard dad armi fan noth ever chang got t...
...
39718  western union someth forgotten classic western...
39719  thi movi incred piec work explor everi nook cr...
39720  wife watch thi movi becaus plan visit sicili s...
39721  first watch flatlin wa amaz necessari featur g...
39722  whi would thi film good onli gross estim award...
Name: text, Length: 39723, dtype: object
```

```
pickle.dump(tfidf,open('models/tfidfvector_model_2.pkl','wb'))
```

Bag of Words:

As an alternative to TFIDF we also use a bag of word vectorization to compare the best representation. The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model has also been used for computer vision.

The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier.

Bag of Words

```
vect = CountVectorizer(max_features = 11000)
```

```
X_train_dm = vect.fit_transform(X_train)
```

```
X_test_dm = vect.transform(X_test)
```

```
X_train_dm.shape
```

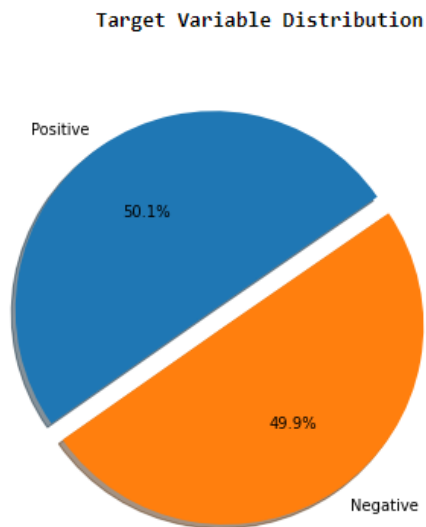
```
(31778, 11000)
```

DATA EXPLORATION

Our dataset is composed of two columns. Those are “text” which denotes the reviews that a user left for a movie, and “label” which denoted by 0 or 1. A negative review is 0 and a positive review is 1. Our target

```
In [66]: #Let us first analyze the distribution of the target variable

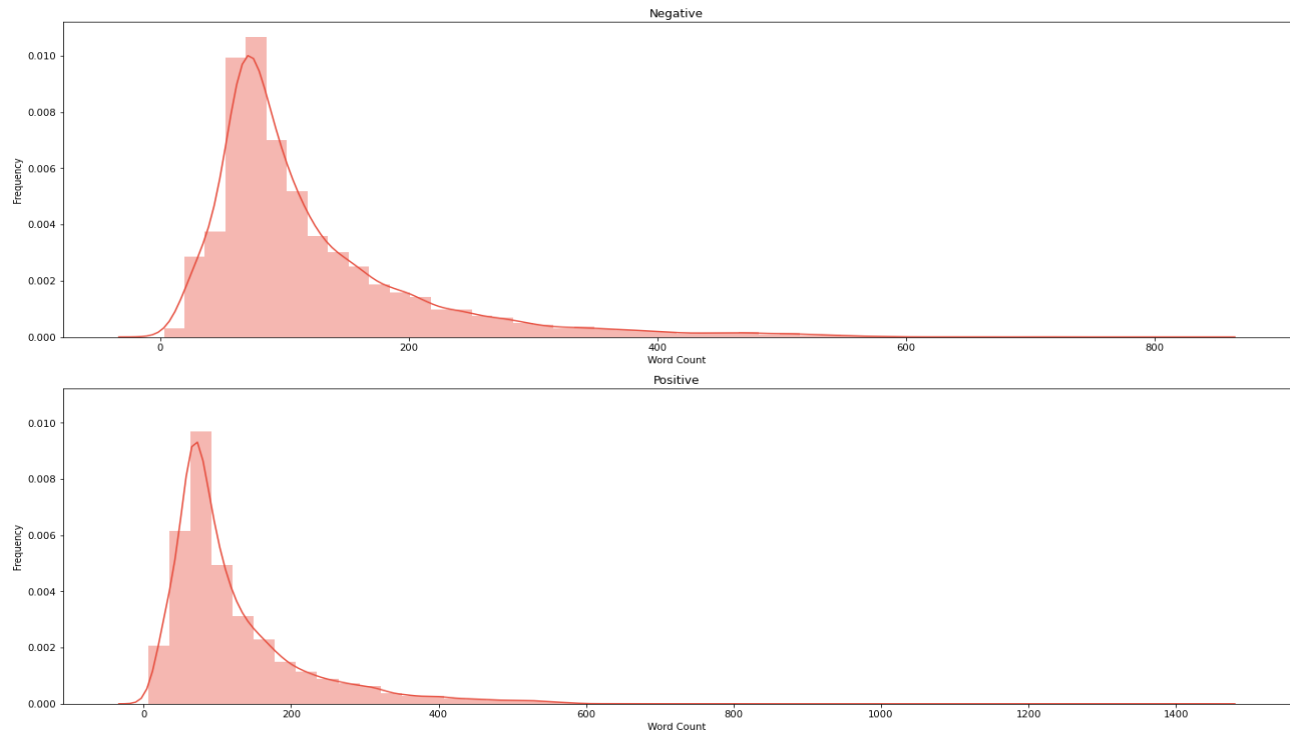
print('\033[1mTarget Variable Distribution'.center(55))
plt.pie(df[target].value_counts(), labels=['Positive','Negative'], counterclock=False, shadow=True,
        explode=[0,0.1], autopct='%1.1f%%', radius=1, startangle=215)
plt.show()
```



variable in this case is the ‘label’ column. From the pie chart you can see the distribution of reviews we have in this dataset; 50.1% of the responses are positive, and 49.9% are negative.

Another interesting fact is the text sequence length. The more words you have in a review, the better our model will be. There are so many words in the English language and of course we cannot include every single one of them. With that said, if we have a good amount of words we can create a pretty accurate model. From the graphs below we can see that when it comes to negative reviews, usually the reviews have a length of about 800 characters and for positive reviews is a bit higher, around 1400 at max.

Words Per text



```
: #Plot word number function

def plot_word_number_histogram(textno, textye):

    """A function for comparing word counts"""

    fig, axes = plt.subplots(ncols=1, nrows=2, figsize=(18, 12), sharey=True)
    sns.distplot(textno.str.split().map(lambda x: len(x)), ax=axes[0], color='#e74c3c')
    sns.distplot(textye.str.split().map(lambda x: len(x)), ax=axes[1], color='#e74c3c')

    axes[0].set_xlabel('Word Count')
    axes[0].set_ylabel('Frequency')
    axes[0].set_title('Negative')

    axes[1].set_xlabel('Word Count')
    axes[1].set_ylabel('Frequency')
    axes[1].set_title('Positive')

    fig.suptitle('Words Per text', fontsize=24, va='baseline')

    fig.tight_layout()

: plot_word_number_histogram(df[df['label'] == 0]['text'],
                             df[df['label'] == 1]['text'],
                             )
```

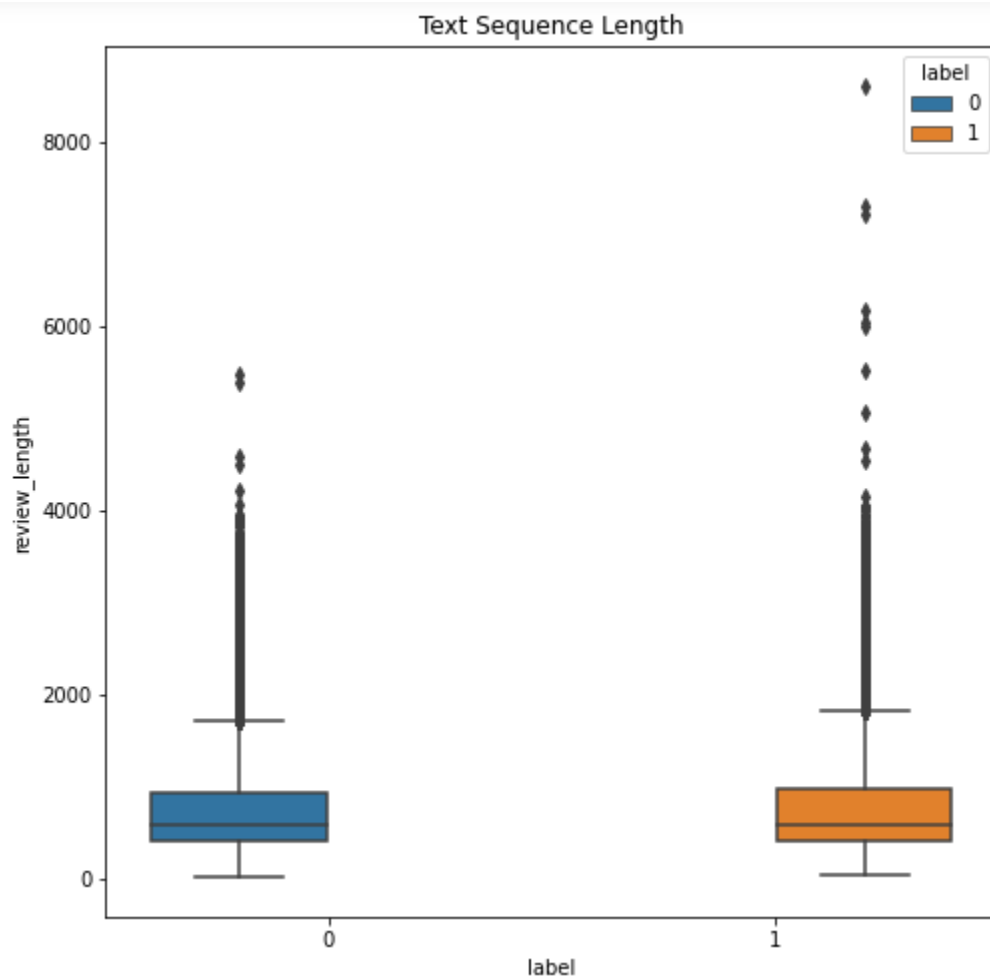
```
In [67]: #Visualising the average text sequence length

df2 = df.copy()
df2['review_length'] = 0

for i in tqdm(range(df.shape[0])):
    df2['review_length'][i] = len(df2['text'][i])

plt.figure(figsize=[8,8])
sns.boxplot(x='label',y='review_length', data=df2, hue='label')
plt.title('Text Sequence Length')
plt.show()

100%|██████████| 39723/39723 [00:09<00:00, 4074.82it/s]
```



Since we have all the words tokenized, it's pretty easy to understand which words are being used most often. We can separate those words into two different "graphs". One for positive reviews and one for negative reviews. In the graphs below, you can see what words are being used most often. Size here plays an important role. The bigger the word, the most frequent it is.

In [68]: `#Visualising the`

```
positivedata = df[df['label']== 1]
positivedata =positivedata['text']
negdata = df[df['label']== 0]
negdata= negdata['text']

def wordcloud_draw(data, color, s):
    words = ' '.join(data)
    cleaned_word = " ".join([word for word in words.split() if(word!='movie' and word!='film')])
    wordcloud = WordCloud(stopwords=stopwords.words('english'),background_color=color,width=2500,height=2000).generate(cleaned_word)
    plt.imshow(wordcloud)
    plt.title(s)
    plt.axis('off')

plt.figure(figsize=[20,10])
plt.subplot(1,2,1)
wordcloud_draw(positivedata, 'white', 'Most-common Positive words')

plt.subplot(1,2,2)
wordcloud_draw(negdata, 'white', 'Most-common Negative words')
plt.show()
```



PROPOSED APPROACH

The following two lists contain the foundations of our approach. The algorithms column lists the algorithms used for the classification task and the methods column lists the basic mechanisms for feature engineering. We obtained acceptable results with common algorithms such as logistic regression (LR), Decision Tree Classifier (DT), Naive Bayes(MNB), Random Forest (DT), and support vector machines (SVM).

Algorithms

- Logistic Regression (LR)
- Decision Tree Classifier
- Naive Bayes Classifier(MNB)
- Random Forest Classifier
- Linear Support Vector Machine(SVM)

Pre-processing methods

- TF-IDF
- Bag of Words (BoW)

Exporting methods

- Binarization

FEATURE ENGINEERING

Most of the NLP tasks for feature engineering are done at text preprocessing and vectorization steps discussed in the above sections. Here we just split the 80% of the data for training and the remaining 20% for testing and evaluation.

```
In [69]: #Splitting the data into training & testing sets

X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(x, y, train_size=0.8, test_size=0.2, random_state=0)

print('Original set ---> ',X.shape,Y.shape,'\nTraining set ---> ',Train_X.shape,Train_Y.shape,'\nTesting set ---> ', Test
<----->

Original set ---> (39723, 1) (39723,)
Training set ---> (31778, 119540) (31778,)
Testing set ---> (7945, 119540) (7945,)
```

TRAINING MODELS

Logistic Regression:

```
# Building Logistic Regression Classifier
```

```
LR_model = LogisticRegression()  
LR = LR_model.fit(Train_X, Train_Y)  
pred = LR.predict(Test_X)  
pred_prob = LR.predict_proba(Test_X)  
Classification_Summary(pred,pred_prob,0)
```

Evaluation

<<<----- Evaluating Logistic Regression (LR) ----->>>

Accuracy = 88.7%

F1 Score = 88.6%

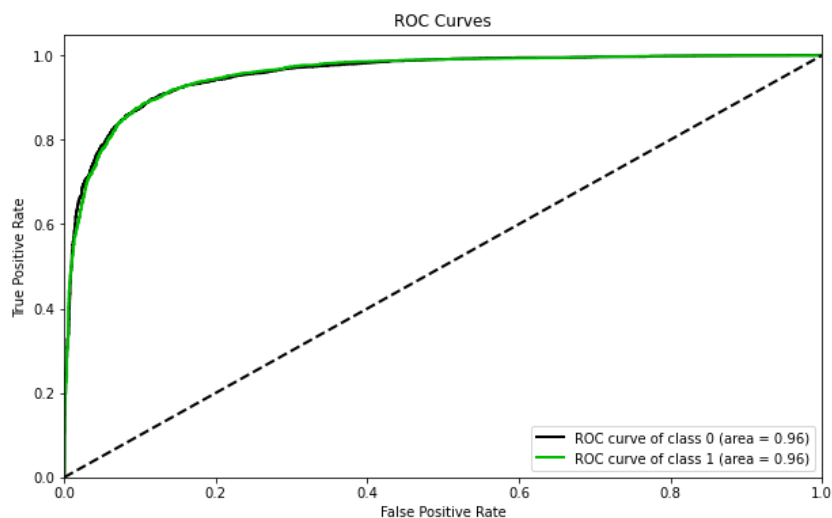
Confusion Matrix:

[[3570 514]

[382 3479]]

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.87	0.89	4084
1	0.87	0.90	0.89	3861
accuracy			0.89	7945
macro avg	0.89	0.89	0.89	7945
weighted avg	0.89	0.89	0.89	7945



Decision Tree Classifier

```
# Building Decision Tree Classifier

DT_model = DecisionTreeClassifier()
DT = DT_model.fit(Train_X, Train_Y)
pred = DT.predict(Test_X)
pred_prob = DT.predict_proba(Test_X)
Classification_Summary(pred,pred_prob,1)
```

Evaluation:

<<<----- Evaluating Decision Tree Classifier (DT) ----->>>

Accuracy = 71.0%

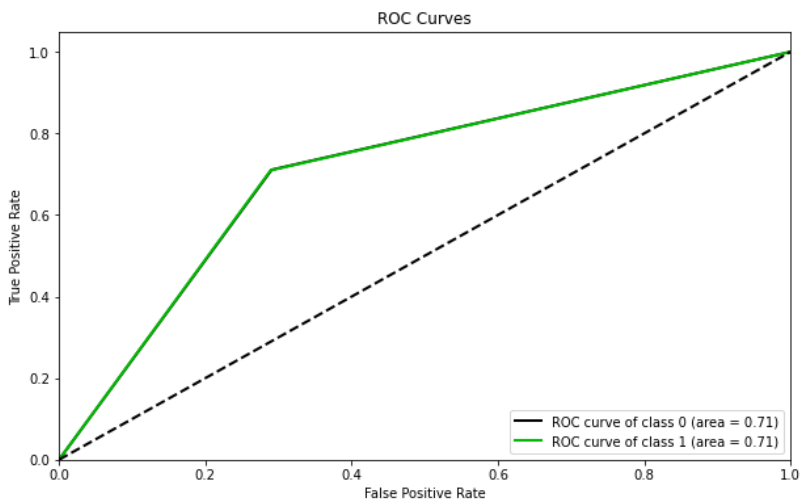
F1 Score = 70.39999999999999%

Confusion Matrix:

```
[[2903 1181]
 [1122 2739]]
```

Classification Report:

	precision	recall	f1-score	support
0	0.72	0.71	0.72	4084
1	0.70	0.71	0.70	3861
accuracy			0.71	7945
macro avg	0.71	0.71	0.71	7945
weighted avg	0.71	0.71	0.71	7945



Random Forest Classifier

```
# Building Random Forest Classifier

RF_model = RandomForestClassifier()
RF = RF_model.fit(Train_X, Train_Y)
pred = RF.predict(Test_X)
pred_prob = RF.predict_proba(Test_X)
Classification_Summary(pred,pred_prob,2)
```

Evaluation:

<<<----- Evaluating Random Forest Classifier (RF) ----->>>

Accuracy = 84.6%

F1 Score = 84.3%

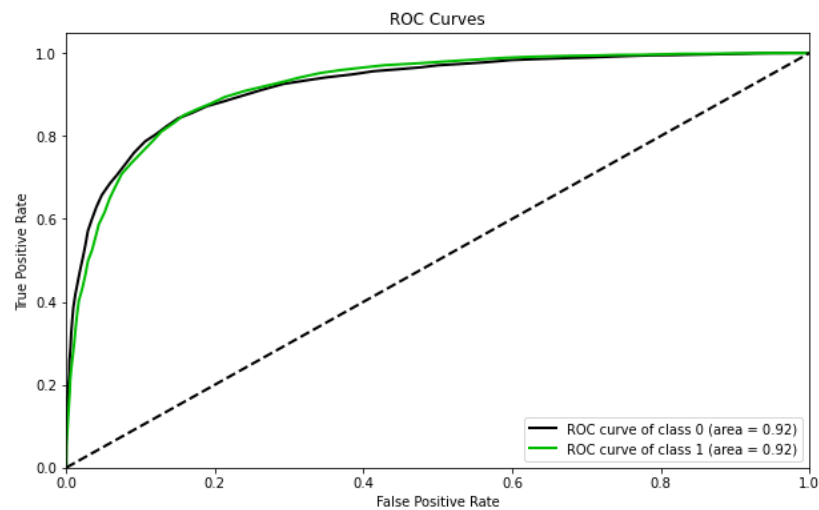
Confusion Matrix:

[[3440 644]

[581 3280]]

Classification Report:

	precision	recall	f1-score	support
0	0.86	0.84	0.85	4084
1	0.84	0.85	0.84	3861
accuracy			0.85	7945
macro avg	0.85	0.85	0.85	7945
weighted avg	0.85	0.85	0.85	7945



Naive Bayes Classifier

```
# Building Naive Bayes Classifier

NB_model = BernoulliNB()
NB = NB_model.fit(Train_X, Train_Y)
pred = NB.predict(Test_X)
pred_prob = NB.predict_proba(Test_X)
Classification_Summary(pred,pred_prob,3)
```

Evaluation:

<<<----- Evaluating Naïve Bayes Classifier (NB) ----->>>

Accuracy = 84.5%

F1 Score = 83.6%

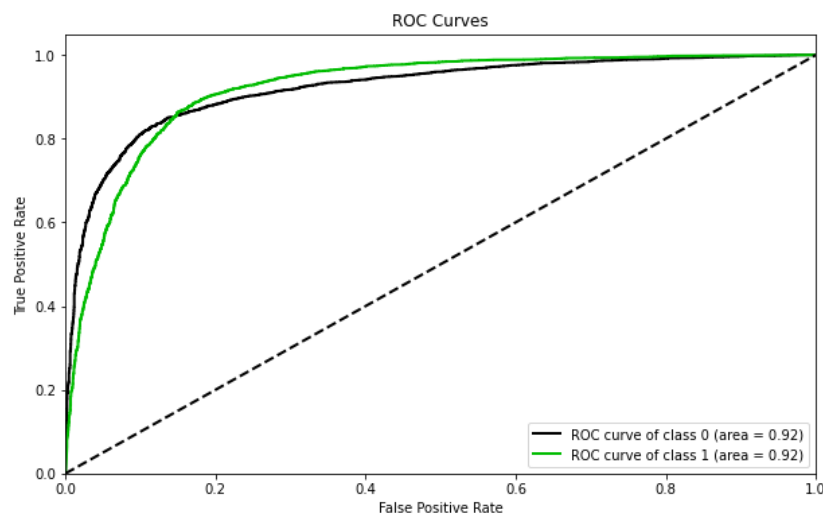
Confusion Matrix:

[[3578 506]

[723 3138]]

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.88	0.85	4084
1	0.86	0.81	0.84	3861
accuracy			0.85	7945
macro avg	0.85	0.84	0.84	7945
weighted avg	0.85	0.85	0.85	7945



Summary:

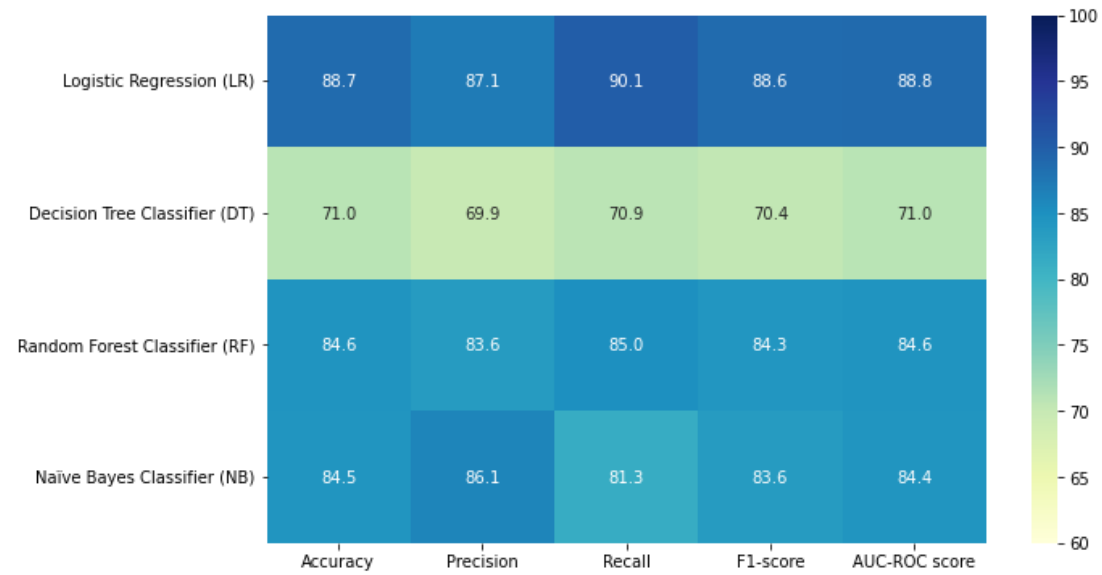
Using TF IDF vectorization we can see on the figure below which show the comparisons of different algorithms, which Logistic regression has the best score in F1-score of 88.6%. all the algorithms were evaluated using the same training set and test set.

We can conclude after running the Logistic Regression (LR), Decision Tree Classifier (DT), Random Forest Classifier (RF) and Naive Bayes Classifier (NB) modules.

```
#Comparing all the models Scores
```

```
#plt.figure(figsize=[12,5])
```

```
sns.heatmap(Evaluation_Results, annot=True, vmin=60, vmax=100.0, cmap='YlGnBu', fmt='.1f')  
plt.show()
```



GENERATE THE PICKLE

To use models and word vectors externality to app deployment we need to binarize the python object using pickle python package.

```
import pickle|
pickle.dump(vect,open('models/bagg_word_final.pkl','wb'))
pickle.dump(svm_classifier,open('models/model_final.pkl','wb'))
```

DEPLOYMENT

Our Prototype App can be accessed using any web browser, by entering url <https://moviesentiment2.herokuapp.com/> :

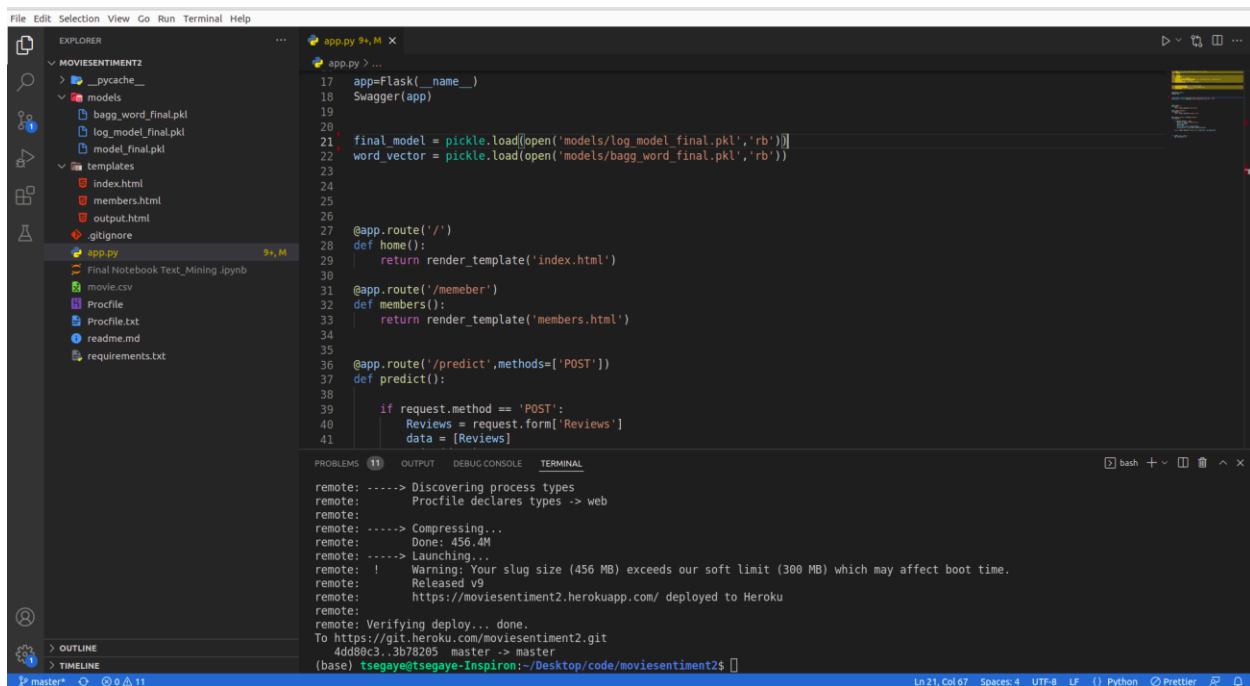
It is with help of the following technologies and tools:

- pickled model
- pickled word vectorization
- heroku cloud app deployment - Python stack
- Flask
- HTML for user input

Flask is used for developing web applications using python, implemented on Werkzeug and Jinja2.

Advantages of using Flask framework are: There is a built-in development server and a fast debugger provided.

Here is our file structure and organization:



The screenshot displays a VS Code editor interface. On the left, the 'EXPLORER' sidebar shows the project structure for 'MOVIESENTIMENT2'. The files listed include: `__pycache__`, `models` (containing `bagg_word_final.pkl`, `log_model_final.pkl`, and `model_final.pkl`), `templates` (containing `index.html`, `members.html`, and `output.html`), `gitignore`, `app.py` (selected), `Final Notebook Text_Mining.ipynb`, `movie.csv`, `Profile`, `Profile.txt`, `readme.md`, and `requirements.txt`. The main editor window shows the content of `app.py`, which is a Flask application. The code includes imports for `Flask`, `Swagger`, `pickle`, and `request`. It defines a `home` route, a `members` route, and a `predict` route that handles POST requests. The `predict` route loads pickled models and performs sentiment analysis on the input reviews. The bottom panel shows the 'TERMINAL' output, which displays the Heroku deployment process, including discovering process types, compressing the slug, and launching the application. The final output shows the application is deployed to `https://moviesentiment2.herokuapp.com/`.

```
17 app=Flask(__name__)
18 Swagger(app)
19
20
21 final_model = pickle.load(open('models/log_model_final.pkl','rb'))
22 word_vector = pickle.load(open('models/bagg_word_final.pkl','rb'))
23
24
25
26
27 @app.route('/')
28 def home():
29     return render_template('index.html')
30
31 @app.route('/member')
32 def members():
33     return render_template('members.html')
34
35
36 @app.route('/predict',methods=['POST'])
37 def predict():
38
39     if request.method == 'POST':
40         Reviews = request.form['Reviews']
41         data = [Reviews]
```

remote: -----> Discovering process types
remote: Profile declares types -> web
remote: -----> Compressing...
remote: Done: 456.4M
remote: -----> Launching...
remote: ! Warning: Your slug size (456 MB) exceeds our soft limit (300 MB) which may affect boot time.
remote: Released v9
remote: https://moviesentiment2.herokuapp.com/ deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/moviesentiment2.git
4dd88c3..3b78205 master -> master
(base) tsegaye@tsegaye-Inspiron:~/Desktop/code/moviesentiment2\$

You can see the app being deployed to Heroku and generating a URL but this URL might not be available after a seven days trial since we don't have a paid subscription. In that case anyone can request a new URL from our team or run Flask app by typing:

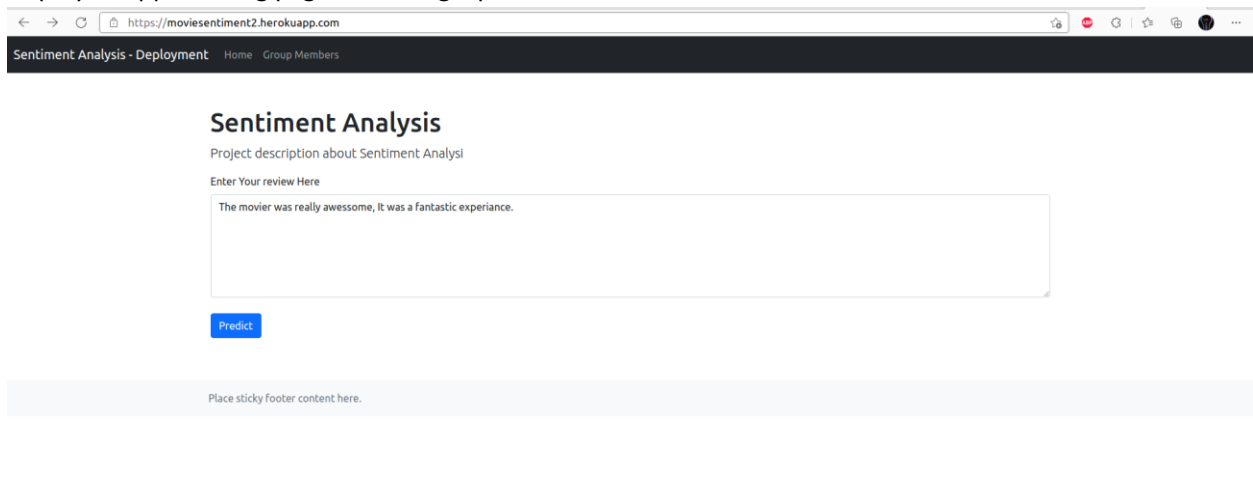
- python app.py or
- flask run

in the main directory after installing necessary packages with the following command

- pip install -r requirements.txt

App Url: <https://moviesentiment2.herokuapp.com/>

Deployed Apps landing page - inserting a positive review



Sentiment Analysis - Deployment Home Group Members

Sentiment Analysis

Project description about Sentiment Analysis

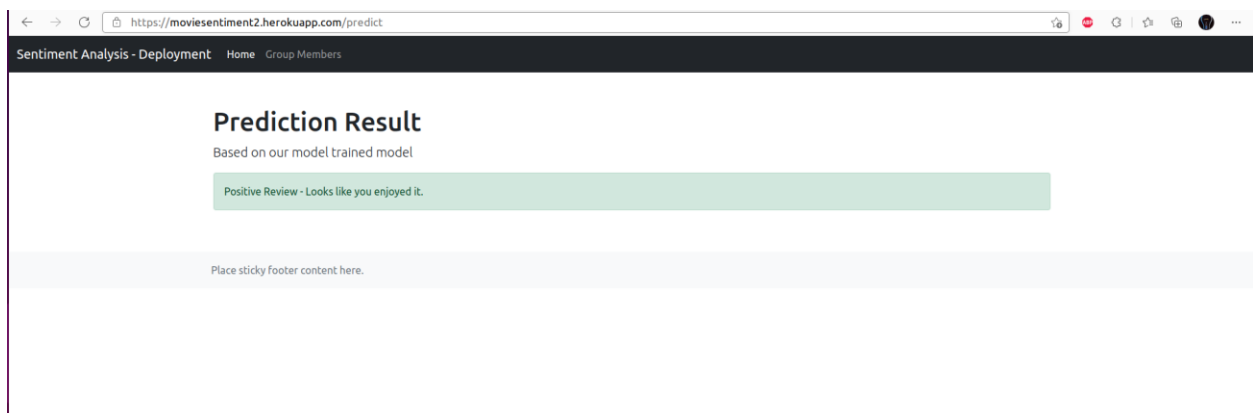
Enter Your review Here

The movier was really awessome, It was a fantastic experience.

Predict

Place sticky footer content here.

Deployed Apps landing page - positive



Sentiment Analysis - Deployment Home Group Members

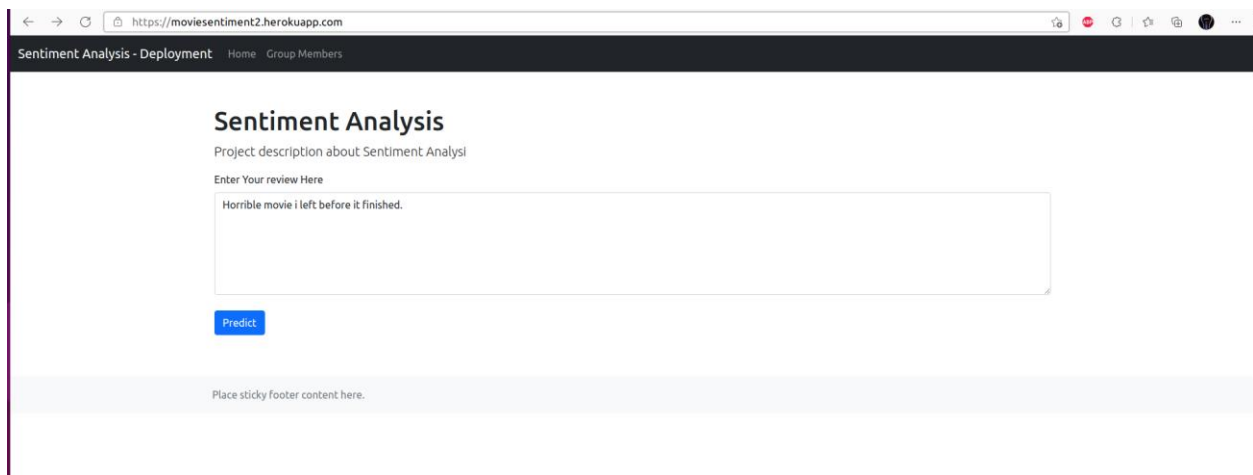
Prediction Result

Based on our model trained model

Positive Review - Looks like you enjoyed it.

Place sticky footer content here.

Deployed Apps landing page - inserting a negative review



The screenshot shows a web browser at the URL `https://moviesentiment2.herokuapp.com`. The page has a dark header with the title "Sentiment Analysis - Deployment" and navigation links for "Home" and "Group Members". The main content area is titled "Sentiment Analysis" with a subtitle "Project description about Sentiment Analysis!". Below this is a form labeled "Enter Your review Here" containing the text "Horrible movie I left before it finished." and a blue "Predict" button. A light gray footer area contains the text "Place sticky footer content here."

Sentiment Analysis - Deployment Home Group Members

Sentiment Analysis

Project description about Sentiment Analysis!

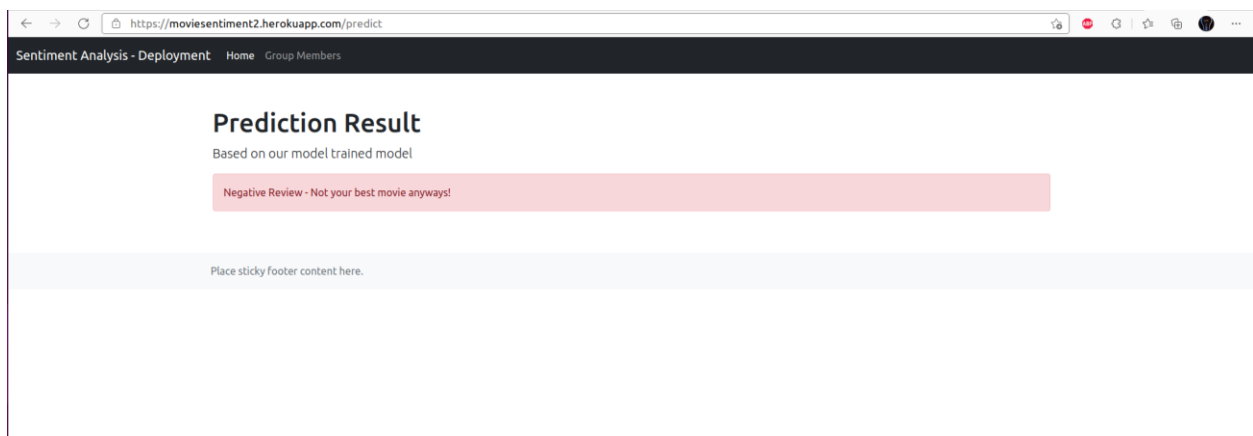
Enter Your review Here

Horrible movie I left before it finished.

Predict

Place sticky footer content here.

Deployed Apps landing page - Negative result



The screenshot shows the same web browser at the URL `https://moviesentiment2.herokuapp.com/predict`. The header is identical. The main content area is titled "Prediction Result" with a subtitle "Based on our model trained model". Below this is a red box containing the text "Negative Review - Not your best movie anyways!". The footer area is the same as the previous screenshot.

Sentiment Analysis - Deployment Home Group Members

Prediction Result

Based on our model trained model

Negative Review - Not your best movie anyways!

Place sticky footer content here.

SUMMARY

We can conclude after running the Logistic Regression (LR), Decision Tree Classifier (DT), Random Forest Classifier (RF) and Naive Bayes Classifier (NB) modules, using Logistic Regression and TFIDF tokenization has the highest accuracy for sentiment analysis.

This model is far from accurate since it is trained with a limited amount of data, but it shows how to get started and which algorithms to choose from for such a problem. In this project, we classified movie reviews from the IMDb dataset as positive or negative. We used common pre-processing steps such as HTML and non-word removal and tokenization.

We learned that not only is it important to know the mechanisms of the algorithms, but that feature construction requires creativity and understanding of the problem. We also learned that maintaining good practices while building models is important because it allows reproducibility, and sound model comparison and model selection.

Finally we have built a working prototype which can be accessed from the URL to test if a given review is positive or not. <https://moviesentiment2.herokuapp.com/> .