ST451 Project

Candidate Number: 14565

Introduction

Text processing is a valuable tool to be able to provide crucial insights for businesses, governments and the public. Whether it be reviews, news headlines or social media posts, these sources contain vital information that when analysed with the proper tools can provide crucial insights.

The goal of this project is to apply a Bayesian perspective to text analysis. The project is divided into two parts, the first part focuses on topic modelling, which divides the data into unobserved groups. The second part of the project relates to sentiment analysis, which uses natural language processing to estimate the polarity of text. We apply our algorithms to real world data and then discuss their efficacy.

Part 1

I. Introduction to Topic Modelling

Topic modelling can be used to discover hidden frameworks from a set of text documents. It is an unsupervised machine learning technique that reads a corpus of documents, detects patterns of words between them and then clusters the words into topics that best describe them. Latent Dirichlet allocation (LDA) is a popular Bayesian topic modelling technique that has been used for topic modelling in bio-medicine and natural language applications. In this project we apply LDA to model analyse financial news headlines for Google stocks. The data set is collected from Kaggle and has approximately 170,000 financial news headlines related to google between the period of July 2018 and June 2020. In the upcoming sections, we provide a brief introduction to the LDA model, walk through the data cleaning and preprocessing steps and finally apply the model and discuss results.

II. Introduction to LDA

Latent Dirichlet Allocation for machine learning applications was first introduced by Blei and Jordan (2003). In their introductory paper, they described LDA as a generative probabilistic model which models each term in a text corpus as a finite mixture over an underlying set of topics. This means that the goal of LDA is to generate a set of topics from a given test corpora where we don't know how many or which topics the documents contain.

To give a brief explanation of how the model works, the general idea is that each document in the corpus can be described as a random mixture of unknown topics. Each unknown topic is represented by a set of words. So essentially, LDA solves a clustering problem. The topics are clustered onto a set of words and the documents are clustered onto a set of topics.

To start off, each document in the corpus is assumed to be generated by LDA's generative process. Thus, for document w in the set of documents D the following takes place:

- Choose $\theta \sim Dir(\alpha)$. θ represents the distribution over topics for all our documents. It is dependent on parameter α.
- For each word N in document w:
    - Assign a random topic to the word. The random topic is given by $z_n \sim Multinomial(\theta)$.
    - Choose a random word w_n from the corresponding topic z_n. The word is chosen from the multinomial conditional probability $p(w_n|z_n, \beta)$. Here, β is a $k\ X\ V$ matrix that represents the distribution over vocabulary for all the k topics. β has Dirichlet distribution dependent on parameter η.

An important assumption is that the total number of topics k is known and constant. We also assume that each document in the corpus has N words. Before proceeding it is important to discuss why the Dirichlet Distribution is used in the algorithm.

The Dirichlet distribution is a multivariate generalisation of the Beta distribution. Its probability density function is given by:

$$f(x_1, \dots, x_K | a_1, \dots, a_K) = \frac{1}{\beta(\alpha)} \prod_{i=1}^{K} x_i^{a_i - 1}$$

$$where \sum x_i = 1 \, , x_i \ \geq 0$$

$$\beta(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}$$

The Dirichlet distribution is often used as a prior in Bayesian statistics because it serves as a conjugate prior to the Multinomial distribution. This means that when the observations have a Multinomial distribution and Dirichlet is used as the prior for the observation's parameters, then the posterior will follow a Dirichlet distribution as well. This gives us a closed form expression for the posterior and makes the computation easier by avoiding numerical integration.

Back to LDA, given the α and β parameters, we obtain the following join posterior probability distribution:

$$p(\theta, z, w | \alpha, \beta) = p(\theta | \ \alpha) \prod_{n=1}^{N} p(z_n | \theta) p(w_n | z_n, \mathrm{B})$$

Where θ is the topic distribution over documents, z is the set of N topics and w is the set of N words obtained in the previous step. By summing over z and then integrating out θ we get the marginal distribution of a single document. And then by taking the product marginal distributions of all single documents we obtain the probability over the set of all documents.

$$p(D|\alpha,\beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\prod_{n=1}^{N_d} \sum_{z_{d_n}} p\left(z_{d_n}|\theta_d\right) p(w_{d_n}|z_{d_n},\beta))\, d\theta_d$$

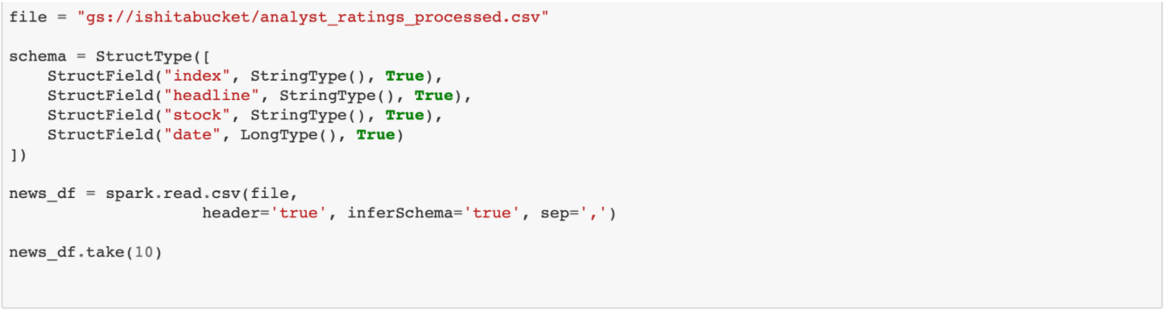
What sets LDA apart is that for each document the topic node is sampled again and again. This means that each document can be associated with multiple topics. However, this posterior is intractable and so approximation techniques have to be used to solve this problem. The most common way to solve this problem for LDA is through Variational Bayes inference. Variational Bayes inference approximates the posterior to a simpler parametric distribution. The approximate posterior is selected dependent on minimizing the Kullback-Leibler divergence. However, the variational bayes or "batch" inference has been shown to be particularly costly when topic modelling on large datasets (Hoffman, Bach & Blei, 2010). To reduce computation time on large datasets a new technique called Online Variational Bayes has been developed. Online Variational Bayes, makes batches of the data and only feeds some of the data to the model at each pass. We will use both Batch Variational Bayes and Online Variational Bayes during our implementation.

III. Data Processing

Data preprocessing consists of three parts. First, we load the data into Spark. We then clean the text from the news headlines. The cleaned news headlines are then converted into vectors.

i. Loading the data

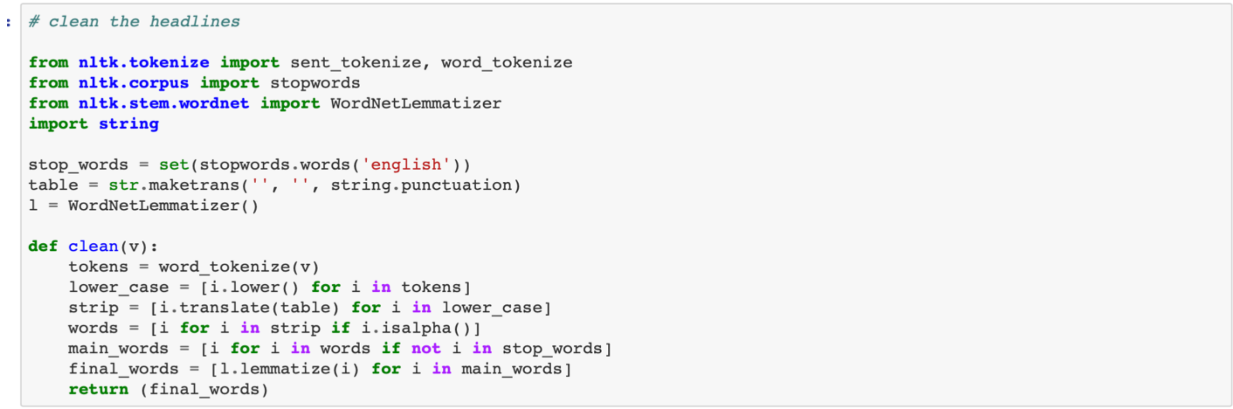
Given the complexity of the model and the size of the dataset we conduct our analysis in Pyspark and start by loading the data as a Spark Dataframe. We filter to the news headlines that relate only to the Google stock.



```
file = "gs://ishitabucket/analyst_ratings_processed.csv"

schema = StructType([
    StructField("index", StringType(), True),
    StructField("headline", StringType(), True),
    StructField("stock", StringType(), True),
    StructField("date", LongType(), True)
])

news_df = spark.read.csv(file,
                    header='true', inferSchema='true', sep=',')

news_df.take(10)
```

ii. Clean the Data

Cleaning the text data is essential so that the LDA model can focus on key words for its analysis. We use Python's NLTK library for our text cleaning. This is because the NLTK library contains many important functions that make text cleaning easier. We start out by using “word_tokenize” to divide each news headline into vectors of individual words. Each word is now called a token. This base step is essential to be able to lemmatize the words. We talk more about lemmatization up ahead. Next, we make all tokens lowercase and strip punctuation from the tokens by making sure that the “string.maketrans()” method maps all punctuation to none. We then remove all tokens where all characters are not alphabet letters using the “isalpha” method. We then move on to removing “stopwords” from our tokens. Stopwords are commonly used words such as “the” or “a” that contribute no value to the text analysis. The NLTK library already contains a list of common English stopwords that make it easier to remove them from our headlines. The final step in our data cleaning is lemmatization. Lemmatization groups together the inflected forms of a word so that they are analysed as a single word by LDA. For example, after lemmatization the words “studies”, “studying” and “study” would all be classed by their lemma “study”. We define a function named “clean” that takes a news headline as input and applies to it all the modifications discussed above. We then convert our news datadrame to an RDD and map each row of the RDD to the “clean” function. This cleans and tokenizes all the news headlines.

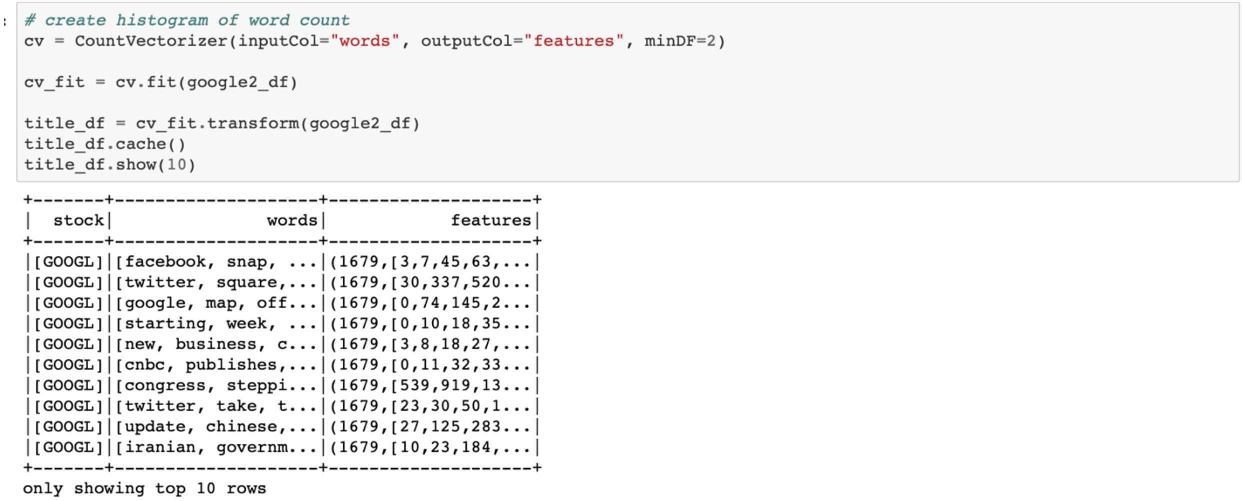


```
# clean the headlines

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string

stop_words = set(stopwords.words('english'))
table = str.maketrans('', '', string.punctuation)
l = WordNetLemmatizer()

def clean(v):
    tokens = word_tokenize(v)
    lower_case = [i.lower() for i in tokens]
    strip = [i.translate(table) for i in lower_case]
    words = [i for i in strip if i.isalpha()]
    main_words = [i for i in words if not i in stop_words]
    final_words = [l.lemmatize(i) for i in main_words]
    return (final_words)
```

iii. Vectorization

Since the LDA model does not understand textual data, we first need to vectorize our tokens before feeding them into the model. This means that tokenized headlines need to be converted into fixed length vectors.

We do this using the “CountVectorizer” function from sklearn which functions based on a Bag-of-Words model. The Bag-of-Words model works by assigning a unique number to each word in the text corpus. Then a tokenized headline can be encoded as a fixed length numerical vector. Here the length of the vector is equal to the length of the vocabulary from the text corpus. The value of each position in the vector represents the frequency with which the corresponding word occurs in the headline. We first create an instance of the count vectorizer class and then fit it to the data in order to build a vocabulary of words from the news headlines. We then use the “transform” function to encode each headline as a vector. In all, there are 1679 unique words in the news headlines. This means that each news headline is encoded as a sparse vector of length 1679. We also print out the first 50 words from the vocabulary to see which words are left after the cleaning process. These vectors are now ready to be fed into the LDA model.



```
# create histogram of word count
cv = CountVectorizer(inputCol="words", outputCol="features", minDF=2)

cv_fit = cv.fit(google2_df)

title_df = cv_fit.transform(google2_df)
title_df.cache()
title_df.show(10)
```

```
+-------+--------------------+--------------------+
|  stock|               words|            features|
+-------+--------------------+--------------------+
|[GOOGL]|[facebook, snap, ...|(1679,[3,7,45,63,...|
|[GOOGL]|[twitter, square,...|(1679,[30,337,520...|
|[GOOGL]|[google, map, off...|(1679,[0,74,145,2...|
|[GOOGL]|[starting, week, ...|(1679,[0,10,18,35...|
|[GOOGL]|[new, business, c...|(1679,[3,8,18,27,...|
|[GOOGL]|[cnbc, publishes,...|(1679,[0,11,32,33...|
|[GOOGL]|[congress, steppi...|(1679,[539,919,13...|
|[GOOGL]|[twitter, take, t...|(1679,[23,30,50,1...|
|[GOOGL]|[update, chinese,...|(1679,[27,125,283...|
|[GOOGL]|[iranian, governm...|(1679,[10,23,184,...|
+-------+--------------------+--------------------+
only showing top 10 rows
```

iv. Applying LDA

For this project we use two different variations of LDA – Batch LDA and Online LDA. As discussed before, Batch LDA passes the entire dataset through each iteration whereas online LDA only passes some of the data through each iteration. Applying the LDA algorithm is particularly straightforward using the LDA function from Pyspark's ML library. We classify the headlines into 20 different topics and set the maximum number of iterations to 10. We switch between Batch LDA and Online LDA using the "optimizer" parameter. Online LDA is performed when optimizer is set to "online" and Batch LDA is performed when it is left as default.

We also define a log perplexity function to compare the performance of the Batch LDA algorithm against its Online competitor. The function outputs two graphs – the first one details the relationship between training data size and test perplexity and the second one focuses on training data size and computation time.

IV. Results

We print out the 5 highest weighted words from each category. For both variations of the algorithm, we see that LDA is able to distinguish clearly between at least seven different types of headlines. However, out of the 20 topics modelled, we see that multiple topics related to stock price and industry headlines. This makes sense considering that we only fed financial news headlines to the model.

Batch LDA

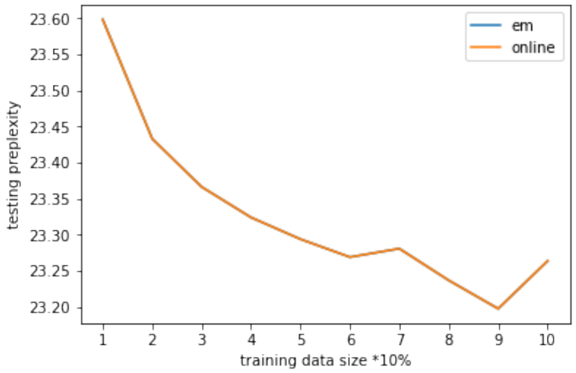
Topic	Top 5 Words
Stock Price Topic	target, alphabet, price, maintains, raise
Smartphone Topic	apple, wwdc, huawei, smartphone, ban
Smart Car Topic	waymo, autonomous, grocery, walmart, vehicle
Industry Topic	google, alphabet, share, market, company
Manufacturing Topic	supply, glass, worker, cryptocurrency, focused
Political Topic	connect, ground, minister, service, european
Marketing Topic	pricing, advertiser, duty, positively, march

Online LDA

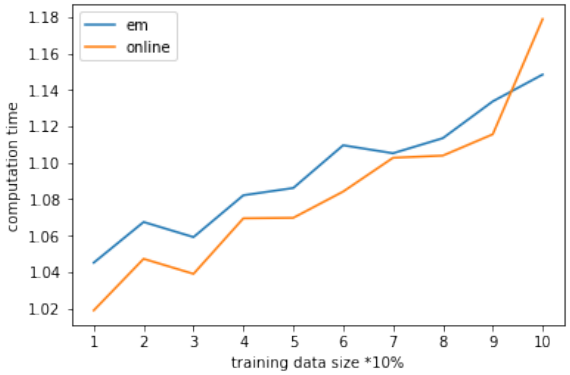
Topic	Top 5 Words
Google Play Store Topic	google, store, platform, play, app
Coronavirus Topic	coronavirus, due, google, cancel, conference
Cryptocurrency Topic	regulation, year, cryptocurrency, bitcoin, hacker
Legal Topic	amount, record, charge, co, attorney
Board of Directors Topic	board, effective, changing, larry, expect
Instagram Topic	long, take, identify, Instagram, damage
Voice Assistant Topic	siri, alexa, trend, hill, say
Stock Price Topic	alphabet, share, company, lower, trading
Industry Topic	portfolio, einhorn, others, buffett, adjusted

The perplexity of both variations of LDA is the same. Perplexity measures how well the model predicts the data. Usually, lower perplexity indicates that the particular variation is better at prediction of the data. In our case, we can see that both the models are equally good at inferring latent topics. However, we observe that in general the online version of the LDA takes less computational time than the batch variation.

Testing Perplexity



Computation Time



Part 2

I. Introduction to Sentiment Analysis

Sentiment analysis is a classification problem that predicts the polarity of a text through computational analysis. It has many real-world applications in marketing, forecasting and to track public sentiment about a product. Over time different algorithms have been used to classify sentiment, some rule-based such as the Naive Bayes algorithm and others automatic, like Deep Learning. In this project however, we focus on applying different variations of the Naive Bayes algorithm to classify sentiment. We apply the Naive Bayes algorithm to classify sentiment of Yelp reviews. The dataset is acquired from Kaggle and contains 560,000 unique yelp reviews. Each review is assigned a polarity score of 1 if it is a negative review and 2 if it is a positive review. In this empirical exercise, we train a Naive Bayes model on the training data and analyse its accuracy on testing data.

II. Naive Bayes Model

Before proceeding further let's go over a brief overview of the Naive Bayes model. The Naive Bayes algorithm is perhaps one of the oldest machine learning algorithms. It is a supervised learning algorithm that is based on the Bayes theorem and an assumption of conditional independence between the features of the dataset. This is to say that the algorithm assumes that the occurrence of one word in a text document does not affect the probability of seeing any other word. The Naive Bayes classifier ignores all grammar rules and the order of words, and classifies text based on a Bag-of-Words model. Even though the model is computationally easy to implement it has been shown to perform classification tasks surprising well.

For the purposes of text classification, we will refer to the words in our Yelp reviews as features and the sentiment as a category. The bayes theorem states that the probability of the review belonging to a certain category given certain features is:

$$P(C|features) = \frac{P(features|C)P(C)}{P(features)}$$

Here $P(C)$ is the prior, $P(features|C)$ is the likelihood and $P(features)$ is the evidence. Since we are trying to classify the reviews into one of two categories – negative or positive, one way to classify would be to calculate the posterior:

$$\frac{P(C_1|features)}{P(C_2|features)} = \frac{P(features|C_1)P(C_1)}{P(features|C_2)P(C_2)}$$

To compute $P(features|C_1)$ and $P(features|C_2)$ we need to specify a generative model. This is where the "naive" part of Naive Bayes comes in. Due to assumption of conditional independence between features, we can say that:

$$P(features|C_i) = P(feature_1|C_i) * P(feature_2|C_i) * \ldots * P(feature_n|C_i)$$

Where each feature is a word. There are different kinds of generative models that can be used to specify the generative distribution for each category. For the purpose of this project, we will focus on three assumptions about how the data for each category is generated:

- Multinomial Naive Bayes:
- Bernoulli Naive Bayes
- Complement Naive Bayes

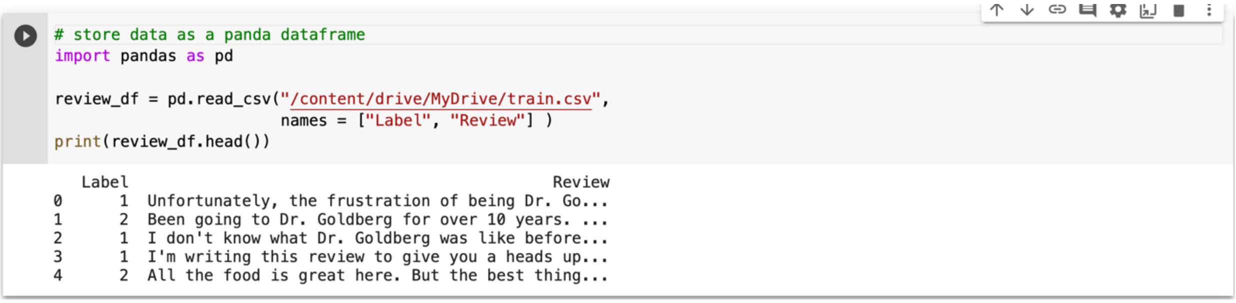
For multinomial Naive Bayes, we assumed that the features are generated from a multinomial distribution. This means that the probability $P(feature_i|C_i)$ is based on word frequency. However, when calculating the likelihood this way we encounter a problem. Say that feature 1 does not appear for category 1. Then the frequency-based probability $(feature_1|C_1)$ will be equal to zero. This means that the likelihood will be zero and this will discount the contribution of all other features towards the likelihood. To prevent this from happening, Naive Bayes usually incorporates a pseudo count to all word frequencies.

The Bernoulli Naive Bayes model has its own classification rules. It estimates $P(feature_i|C_i)$ as the fraction of reviews from category C_i that contain the feature $feature_i$. So, the Bernoulli model ignores the number of occurences of $feature_i$ in each review and instead uses binary occurrence information.

Complement Naive Bayes is similar to Multinomial Naive Bayes but it is designed to work on imbalanced datasets. It estimates feature generation probabilities using the probability of $feature_i$ not belonging to class C_i. The review is assigned to the category with the lowest complement match. The creators of Complement Naive Bayes have shown that it's estimates are more stable than those from Multinomial Naive Bayes.

III. Data Preprocessing

We start by loading our data as a pandas dataframe. The label column specifies the sentiment of the review – 1 for negative and 2 for positive. The review column contains the text of the review.

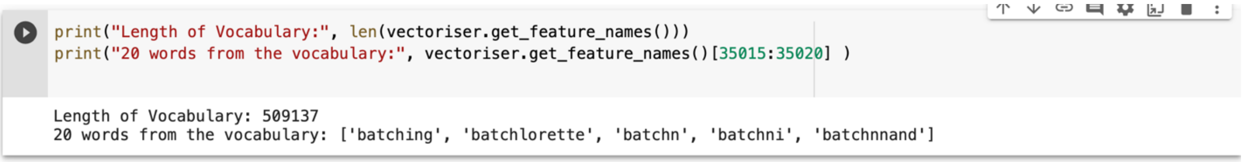


```
# store data as a panda dataframe
import pandas as pd

review_df = pd.read_csv("/content/drive/MyDrive/train.csv",
                        names = ["Label", "Review"] )
print(review_df.head())
```

```
   Label                                             Review
0      1  Unfortunately, the frustration of being Dr. Go...
1      2  Been going to Dr. Goldberg for over 10 years. ...
2      1  I don't know what Dr. Goldberg was like before...
3      1  I'm writing this review to give you a heads up...
4      2  All the food is great here. But the best thing...
```

We then clean the reviews using the NLTK library. The cleaning process remains exactly the same as for part 1 of the project. We make all the letters lowercase, strip all punctuation, remove non-alphabetic terms, remove common words and lemmatize inflected words. This is done to make sure that the Naive Bayes model is able to focus on the key words in the reviews to output the best results.

We then convert our reviews into sparse vectors using the count vectorizer. It converts the text into fixed length numerical vectors. We have 509137 unique words in our vocabulary. Thus each review is converted to a sparse vector of length 509137. We then split our data into training and testing. Exactly 420,000 reviews are used for training and 140,000 are separated for testing.



```
print("Length of Vocabulary:", len(vectoriser.get_feature_names()))
print("20 words from the vocabulary:", vectoriser.get_feature_names()[35015:35020] )
```

```
Length of Vocabulary: 509137
20 words from the vocabulary: ['batching', 'batchlorette', 'batchn', 'batchni', 'batchnnand']
```

IV. Applying Naive Bayes and Results

We fit three different types of Naive Bayes models using the sklearn library. We first fit the model to training data. Then we predict the labels of test data and obtain the test accuracy by comparing the predicted labels to actual labels.

Classifier	Test Accuracy
Multinomial Naive Bayes	87.061
Bernoulli Naive Bayes	79.286
Complement Naive Bayes	87.065

In our analysis we find that Multinomial Naive Bayes and Complement Naive Bayes provide us with the best test accuracy of approximately 87%. This similar performance of MNB and CNB is expected given that we trained the models on a balanced dataset- there were an equal number of positive and negative reviews. Bernoulli Naive Bayes had a test accuracy of 79% which is much lower than that of MNB and CNB. This might be because BNB focuses only on whether a word occurs in a document or not and it disregards the frequency with which that word occurs, this makes it much more susceptible to misclassifying negative reviews that contain a few positive words and vice-versa.

Bibliography

- Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. the Journal of machine Learning research, 3, 993-1022.

- Hoffman, M., Bach, F. R., & Blei, D. M. (2010). Online learning for latent dirichlet allocation. In advances in neural information processing systems

- Islam, M. J., Wu, Q. J., Ahmadi, M., & Sid-Ahmed, M. A. (2007, November). Investigating the performance of naive-bayes classifiers and k-nearest neighbor classifiers. In 2007 International Conference on Convergence Information Technology (ICCIT 2007)). IEEE.

- Daily Financial News for 6000 Stocks. (n.d.). Kaggle. Retrieved April 10, 2021, from https://www.kaggle.com/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests

- Yelp Review Sentiment Dataset. (n.d.). Kaggle. Retrieved April 10, 2021, from https://www.kaggle.com/ilhamfp31/yelp-review-dataset