

Sentiment analysis of IMDB movie reviews

CM3060 Natural Language Processing

I Introduction.....	3
Problem area.....	3
Objectives.....	3
Dataset.....	4
Evaluation methodology.....	6
Accuracy score.....	6
Precision.....	6
Recall.....	7
Evaluation cutoffs.....	7
II Implementation.....	7
Baseline.....	8
Exploratory data analysis.....	9
Summary of the dataset.....	9
Sentiment distribution.....	12
Following Lakshmipathi N's approach.....	12
Pre-processing.....	12
Remove HTML and junk text.....	13
Remove special characters.....	13
Remove spaces.....	14
Text stemming.....	14
Remove stopwords.....	15
Classification approach.....	16
Feature extraction.....	16
Testing and training data split.....	17
Model training.....	17
Multinomial Naive Bayes classification.....	17
Model evaluation.....	18
Refinements and iteration.....	19
Preprocessing.....	20
Classification approach.....	22
Test and training data split.....	22
Feature extraction.....	22
Model training and predictions.....	23
Model evaluation.....	23
Accuracy score.....	23

Classification report.....	23
III Outcome.....	25
Performance.....	25
Summary.....	26
Resources and bibliography.....	27

I Introduction

Problem area

User-generated data like reviews can be a huge source of information that can be used to understand how people feel about products and services. One area where there is an abundance of reviews available to the public is the film industry. Whether it's reviews written by individual movie critics for their publications, review aggregator sites like Metacritic or rotten tomatoes, or average movie-goers writing a review on IMDB or Google Play there is a treasure trove of data available.

By analysing movie reviews filmmakers and production companies can gain insights into how their movies are perceived by the audience, and make data-driven decisions about their marketing strategies, potential improvements or even how sequels should be developed.

From a movie-goer's perspective, it could be useful to have access to aggregated sentiments from various sources (such as user reviews and critic reviews) to help them choose which movies to watch.

In academic research this data can be used to identify trends, patterns, and factors that impact public reception of films that can all contribute to the field of film studies.

A human could easily read a few reviews and understand whether it is good or bad. But what about fifty thousand reviews? It wouldn't really be feasible to have a human read so many reviews. It would take ages to get any insights. Computers are ideal to analyse such large datasets but they don't understand human language and the complex ways we can combine words to express opinions, right?

That is where the field of Natural Language Processing (NLP) comes in. We can use NLP techniques and machine learning algorithms to help us analyse large amounts of text data. When we want to go through a bunch of reviews and understand whether they are good or bad, one technique we can apply is sentiment analysis.

Sentiment analysis is a type of text categorization task used to determine the sentiment or subjective opinion expressed in a given piece of text computationally (Jurafsky and Martin, 2023).

When we analyse the sentiment of text data, we are interested in attitudes. According to the Scherer typology of affective states, attitudes are enduring, affectively coloured beliefs or dispositions towards people or objects, like loving, hating, enjoying, disliking, etc.

Objectives

In this project we will create a text classifier that can analyse the sentiment of large numbers of user-generated movie reviews. We will be using some natural language processing techniques to preprocess the dataset, extract features, and train a Multinomial Naive Bayes

model from the Scikit Learn library to predict whether the sentiment of a movie review is positive or negative.

We will use an Kaggle user's notebook Sentiment Analysis of IMDB Movie Reviews (N., Lakshmipathi 2020) as a point of departure, following a similar initial approach to preprocessing data and extracting features using the Bag of Words approach. We will use the accuracy score and classification report from this notebook as the baseline for comparing our implementation of the model's performance.

Once we have done an initial round of prediction with the model, we will note the performance, and adjust the processing techniques and feature extraction methods to see if we can improve the performance.

In summary, this project has the following objectives:

1. Do an initial round of preprocessing similar to the steps taken in the Sentiment Analysis of IMDB Movie Reviews Kaggle notebook.
2. Train a Multinomial Naive Bayes model to classify movie reviews into one of two categories: positive or negative.
3. Test the classifier on unseen data and compare the performance to the baseline from the Kaggle notebook.
4. Perform a second round of preprocessing and note any changes to performance.

Dataset

Having decided on building a classifier to do sentiment analysis on movie reviews the next step was to find an appropriate dataset.

The IMDB Dataset of 50K Movie Reviews is a large movie review dataset available on the Kaggle platform. The dataset was compiled for the proceedings of the 49th annual meeting of the Association for Computational Linguistics: Human Language Technologies (Maas, Daly, Pham, Huang, Ng, and Potts; 2011)

According to Maas et al. (2011), the main dataset contains 50,000 reviews split into training and test sets of 25,000 reviews each. The distribution of labels (positive or negative) is balanced.

To avoid issues with correlated ratings, no more than 30 reviews for a given movie is included in the collection. Moreover, the train and test sets consist of disjoint sets of movies to prevent any advantage gained from memorizing movie-specific terms and their association with labels (Maas et al., 2011).

The labels for the train and test sets were awarded as follows:

- Reviews with a score of 4 or less out of ten: Negative
- Reviews with a score of 7 or more out of ten: Positive

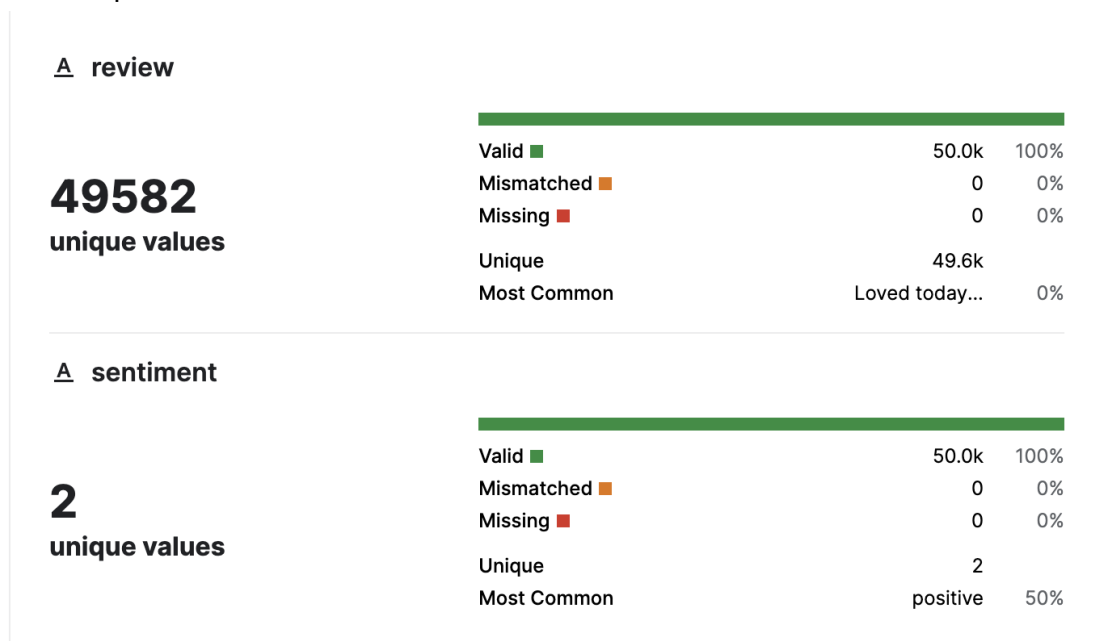
Neutral reviews (5 or 6 out of 10) were left out of the dataset.

The dataset also has an additional 50,000 reviews that are not labeled and can therefore be used for unsupervised learning. This set also contains reviews with a more neutral review. We will not use this part of the dataset for this project.

Inspecting the dataset.csv file, we see two columns. Column A contains the text review, and Column B contains the sentiment, listed as either “positive” or “negative”.

A review	A sentiment
One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. The...	positive
A wonderful little production. The filming technique is very unassuming-very old-time-B...	positive
I thought this	positive

The screenshot below shows that there are approximately fifty thousand unique text reviews, and two unique values found in the sentiment column.



Evaluation methodology

We will use the following attributes to measure model performance:

- Accuracy score
- Precision
- Recall

Accuracy score

In simple terms, the accuracy score is a percentage of correct predictions made by a model. We can informally state this in mathematics terms as:

```
Accuracy = Number of correct predictions / Total number of predictions
```

Parashar (2011) argues that it is used widely to understand the performance of a classification model because it is relatively simple to calculate, and because it conveniently quantifies the performance of the model in one number.

According to Ashar (n.d.) the `accuracy_score` function of the `sklearn.metrics` package can calculate the accuracy score for a set of predicted labels by checking it against the true labels in the test data. When applied it will return a fraction of the correct predictions, for example, `accuracy_score = 0.863`.

While it is useful to summarise model performance simply, accuracy score doesn't work well with unbalanced data. It also doesn't provide any useful information on mistakes in the model (Parashar, 2011). Ideally we want to use it in conjunction with some other metrics.

Precision

A model might incorrectly label a movie review as being of positive sentiment when it isn't. Precision can tell us what proportion of reviews we classified as positive sentiment, actually were positive sentiment. Precision gives us the ratio of true positives to all positives.

Informally, precision is

```
Precision = [True Positives / (True Positives + False Positives)]
```

The `sklearn.metrics` module can calculate the `precision_score` for a prediction.

Recall

Conversely, a model might also label a positive review as negative sentiment. The recall metric (also known as sensitivity) can tell us the proportion of reviews that were negative, where actually classified as negative reviews by the model.

Informally, recall is

$$\text{Recall} = [\text{True Positives} / (\text{True Positives} + \text{False Negatives})]$$

The `sklearn.metrics` module can calculate the `recall_score` for a prediction.

The scores for both precision and recall ranges from 0 to 1, and as with accuracy, the closer you get to 1, the better your model is performing.

Evaluation cutoffs

Cutoffs for accuracy scores will depend on the nature of the project, but Parashar (2011) suggests the following:

score > 0.9 excellent

score >= 0.7 acceptable

score < 0.7 poor

We will use these cutoffs in our evaluation of our models.

II Implementation

In this section, we will preprocess that dataset, build and test the classifier, and obtain results.

The implementation phase of this project is contained in a jupyter notebook **midterm.ipyn**, which can be accessed on request. This document contains all of the code, and screenshots of the outputs generated in the notebook. I had difficulties getting my machine to export the notebook to PDF straight from the document itself.

Other people's code

This project is an evolution of the Kaggle notebook created by user Lakshmipathi N. Code adapted closely from her notebook will be indicated with a comment '#Lakshmipathi N 2020'. Where I have taken inspiration or guidance from other notebooks or code, similar comments with the author's name will be given. A comprehensive bibliography is provided at the end of the project.

My own code or code that has been significantly adapted will be marked with a '#New Code' comment.

Baseline

For the initial preprocessing and classification, this project followed a subset of work done by Lakshmipathi N (2020) in the Kaggle notebook. She uses both a Term Frequency-Inverse Document Frequency model and a Bag of Words approach to feature extraction. For this project, the focus is on using only the Bag of Words approach with the Multinomial Naive Bayes model and seeing if we can improve the baseline score from the original notebook.

We'll use the accuracy score she computed for the model as a baseline, and compare our own model performance based on various preprocessing iterations.

The baseline performance for the Lakshmipathi N notebook is `mnb_bow_score` : 0.751

The classification report was as follows:

	precision	recall	f1-score	support
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.75	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

	precision	recall	f1-score	support
Positive	0.75	0.76	0.75	4993
Negative	0.75	0.74	0.75	5007
accuracy			0.75	10000
macro avg	0.75	0.75	0.75	10000
weighted avg	0.75	0.75	0.75	10000

```
# Import necessary libraries
import pandas as pd

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
```



```
from nltk.tokenize.toktok import ToktokTokenizer

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt
plt.style.use('ggplot')
from wordcloud import WordCloud
from bs4 import BeautifulSoup
```

```
# Import training data
imdb_data=pd.read_csv('IMDB_dataset.csv')

#Adapted from Lakshmipathi N 2020
```

Exploratory data analysis

We will now explore the data and see what we are working with. This is useful to help us see what type of cleaning we need to do.

Summary of the dataset

Once we have created the dataframe by reading the csv file, we can use `describe()`, `df.shape`, and `df.head` to form an overview of the data. We can also generate a word cloud to get an overview of the most occurring words.

```

In 7 1 # Description of the dataframe
      2 imdb_data.describe()

```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety ...	positive
freq	5	25000

4 rows × 2 columns [Open in new tab](#)

```

In 8 1 # Shape of the dataframe (rows, columns)
      2 print(imdb_data.shape)

```

```
(50000, 2)
```

```

In 9 1 # View the top 10 rows of the dataset
      2 imdb_data.head(10)

```

	review	sentiment
0	One of the other reviewers has mentione...	positive
1	A wonderful little production. <b...	positive
2	I thought this was a wonderful way to s...	positive
3	Basically there's a family where a litt...	negative
4	Petter Mattei's "Love in the Time of Mo...	positive
5	Probably my all-time favorite movie, a ...	positive
6	I sure would like to see a resurrection...	positive
7	This show was an amazing, fresh & innov...	negative
8	Encouraged by the positive comments abo...	negative
9	If you like original gut wrenching laug...	positive

```

# Create a word cloud
# Concatenate the text from the review column into a single string.
wordcloud_text = ' '.join(imdb_data['review'])

# Generate the wordcloud
wordcloud = WordCloud(width=800, height=400).generate(wordcloud_text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()

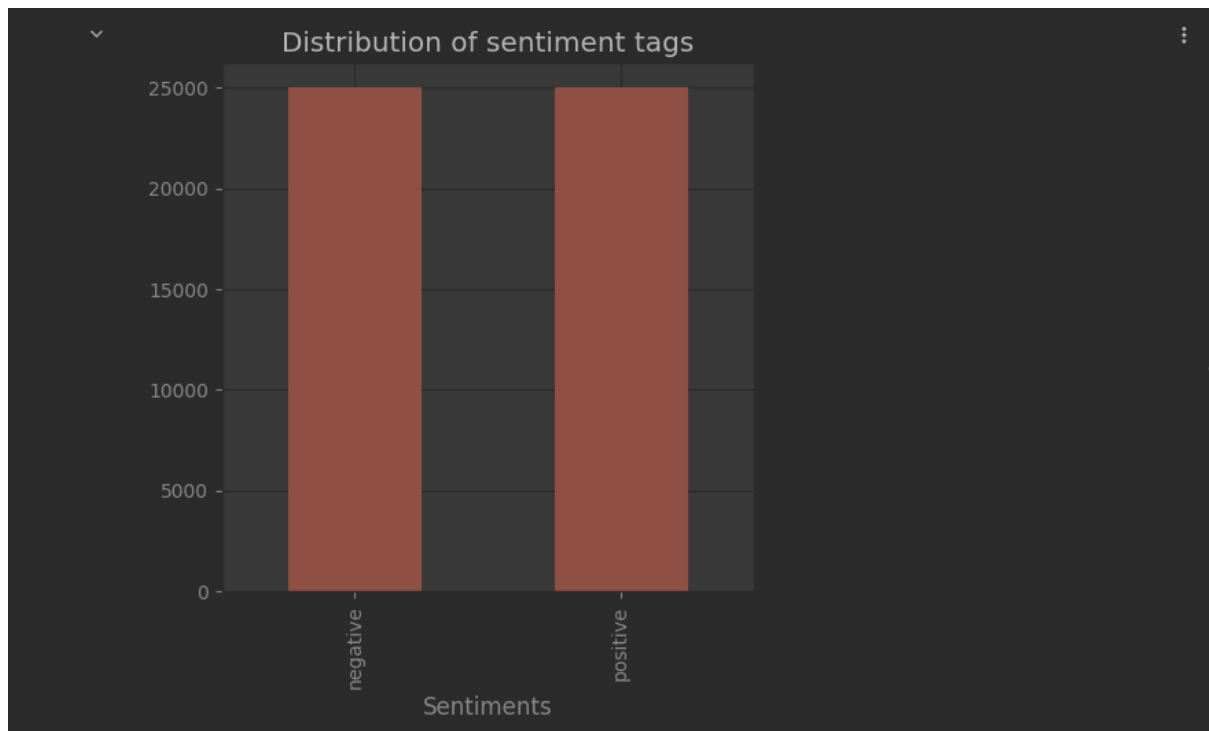
# Adapted from Valkov 2019

```


Sentiment distribution

We want to ensure that we have a balanced dataset to avoid introducing bias into our model. We can draw a bar chart to show us the sentiment labels are distributed in the data.

```
# Draw a bar chart to show the distribution of the data in the sentiment
column.
axis = imdb_data['sentiment'].value_counts().sort_index()\
    .plot(kind="bar",
          title="Distribution of sentiment tags",
          figsize=(5,5))
axis.set_xlabel('Sentiments')
plt.show()
# New Code
```



We are lucky that the data is balanced. We won't need to make any adjustments to try and prevent a bias forming in our model.

Following Lakshmipathi N's approach

A first step is to attempt to reproduce Lakshmipathi's process and compare the results we achieve with the model this way.

Pre-processing

In this section we'll clean up the data. We'll be following Lakshmipathi N's approach to preprocessing for now and then introduce some additional steps later.

According to Valkov (2019) we should be aware that real world text data can be messy. When working with this dataset, they recommend removing HTML tags, punctuation and other special characters, and any excessive spaces.

We'll create some functions to do this cleaning for us and then apply them to the IMDB data.

The approach followed in the Lakshmipathi notebook was:

1. Remove HTML and junk text
2. Remove special characters
3. Remove spaces
4. Text stemming
5. Remove stopwords

Remove HTML and junk text

Removing HTML and other junk text like the "
" we noticed in the word cloud.

```
#Function to strip HTML tags
def strip_html(text):
    # Check if the input looks like a file name
    if os.path.isfile(text):
        with open(text, 'r') as file:
            soup = BeautifulSoup(file, "html.parser")
    else:
        soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

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

# Function that combines the above two functions to clean up junk text
def remove_junk_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text

# Apply the remove_junk_text function to the IMDB dataset
imdb_data['review'] = imdb_data['review'].apply(remove_junk_text)

#Adapted code from Lakshmipathi N 2020 with some changes
```

Remove special characters

Removing punctuation and other characters that are not letters or digits.

```
# Function to remove 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 the remove_special_characters function to the dataset review
column
imdb_data['review'] =
imdb_data['review'].apply(remove_special_characters)

#Lakshmipathi N 2020
```

Remove spaces

Removing extra whitespace other than spaces between words.

```
# The function to remove white spaces
def remove_excessive_spaces(text):
    cleaned_text = re.sub('\s+', ' ', text)
    return cleaned_text.strip()

# Apply the function to the review column of the IMDB data
imdb_data['review'] = imdb_data['review'].apply(remove_excessive_spaces)
```

Text stemming

Here we will be stemming the text using the Porter stemming algorithm and applying it to the 'review' column of the imdb_data dataframe.

By applying stemming, the code reduces words to their base or root form, which can help in text analysis tasks such as text classification, information retrieval, or topic modeling.

Stemming aims to normalize words and treat different variations of the same word as the same base word.

```
#Stemming the text with the Porter stemming algorithm from NLTK
def simple_stemmer(text):
    ps=nlk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)

# Check the result of the stemming process
imdb_data.head(5)
```

```
#Lakshmipathi N 2020
```

Out 16 ▾	review	sentiment	:
0	one of the other review ha mention that...	positive	
1	a wonder littl product the film techniq...	positive	
2	i thought thi wa a wonder way to spend ...	positive	
3	basic there a famili where a littl boy ...	negative	
4	petter mattei love in the time of money...	positive	

Remove stopwords

By removing stopwords we try to eliminate common words that often do not contribute significant meaning to the text analysis tasks. This can help improve the accuracy and efficiency of NLP applications such as sentiment analysis.

First, we set the stopwords list to English, and then let's just check the list of words that will be excluded in the next step.

```
# Setting English stopwords
stop=set(stopwords.words('english'))

#Initialise a tokenizer
tokenizer=ToktokTokenizer()

# Function to remove 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
stop]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not
in stop]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)

#Lakshmipathi N 2020

# Test an example
review = "This is an example review with stopwords."
filtered_review = remove_stopwords(review)
print(filtered_review)
#New Code
```

```
example review stopwords .

In 18 1 imdb_data.head(10)

Out 18  ✓
```

	review	sentiment
0	one review ha mention watch 1 oz episod...	positive
1	wonder littl product film techniqu veri...	positive
2	thought thi wa wonder way spend time ho...	positive
3	basic famili littl boy jake think zombi...	negative
4	petter mattei love time money visual st...	positive
5	probabl alltim favorit movi stori selfl...	positive
6	sure would like see resurrect date seah...	positive
7	thi show wa amaz fresh innov idea 70 fi...	negative
8	encourag posit comment thi film wa look...	negative
9	like origin gut wrench laughter like th...	positive

Classification approach

Feature extraction

In this section we'll do some work to represent the text data numerically by extracting relevant features. We'll use the Bag-of-Words model, where each review is represented as a vector of word frequencies or presence/absence indicators. This will pair well with the Naive Bayes model we'll be using later.

Let's use the CountVectorizer from the scikit-learn library to perform feature extraction with a bag-of-words approach.

```
# Use the CountVectorizer for bag of words
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))

# Transform reviews
cv_reviews=cv.fit_transform(imdb_data.review)

# Check results
print('Bag of Words - cv_reviews:',cv_reviews.shape)

#Adapted code from Lakshmipathi N 2020 with some changes
```

```
Bag of Words - cv_reviews: (50000, 7528779)
```

We also need to convert the sentiment data into binary form. This is because when optimising for sentiment analysis, occurrence of a word matters more than the frequency of the word.

Let's use LabelBinarizer from scikit-learn to give each sentiment value (which is "positive" and "negative" at the moment) a binary value.


```
# Label the sentiment data
lb=LabelBinarizer()

# Transform the sentiment data
sentiment_data=lb.fit_transform(imdb_data.sentiment)

# Check the results
print(sentiment_data.shape)

#Adapted code from Lakshmipathi N 2020 with some changes
```

```
(50000, 1)
```

Testing and training data split

It's common to split a dataset into two separate sets to use for training and testing. As we can infer from the names, one set, usually larger, is used to train a model, while the other set is used to test the model on to assess the performance.

While it seems to work, the approach to splitting testing and training data used by Lakshmipathi N 2020 is somewhat cumbersome.

Let's use the `train_test_split` function from SKlearn to do this for us instead.

```
# Use sklearn to split the dataset into training and testing datasets
x = cv_reviews #input features
y = sentiment_data #labels
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=1)

#New Code
```

Model training

In this section we'll fit the data to a Naive Bayes model. We'll compare the results of this model with the baseline we got from the Lakshmipathi N notebook. We discussed this in the previous section.

Multinomial Naive Bayes classification

- Features are generated from a multinomial distribution by observing counts across categories.
- Data must be converted from strings to numbers. We can use TfidfVectorizer, or a bag-of-words approach with the CountVectorizer like we did earlier.

```
# Train the multinomial Naive Bayes model
from sklearn.naive_bayes import MultinomialNB
import numpy as np

mnb_model = MultinomialNB()

# Reshape train_sentiments into a 1-dimensional array
y_train_1d = np.ravel(y_train)
y_test_1d = np.ravel(y_test)

# Fit the bag of words data using the training reviews and 1-dimensional
train_sentiments
mnb_model_bagofwords = mnb_model.fit(x_train, y_train_1d)
print(mnb_model_bagofwords)

#Adapted code from Lakshmipathi N 2020
```

```
# Predict the model
mnb_model_bagofwords_predict = mnb_model_bagofwords.predict(x_test)
```

Model evaluation

```
#Accuracy score for bag of words

mnb_model_bagofwords_score =
accuracy_score(y_test_1d,mnb_model_bagofwords_predict)
print("mnb_model_bagofwords_score :",mnb_model_bagofwords_score)
```

```
mnb_model_bagofwords_score : 0.4954
```

```
# Compile a classification report
from sklearn.metrics import classification_report
mnb_model_bagofwords_report = classification_report(y_test_1d,
mnb_model_bagofwords_predict, target_names = ['Positive', 'Negative'])
print(mnb_model_bagofwords_report)
```

	precision	recall	f1-score	support
Positive	0.50	0.97	0.66	5044
Negative	0.23	0.01	0.01	4956
accuracy			0.50	10000
macro avg	0.36	0.49	0.34	10000
weighted avg	0.36	0.50	0.34	10000

The implementation we created following the Lakshmipathi notebook approach did not perform well. In the first place, it did not match or exceed the benchmark accuracy score of 0.751

Precision and recall for both positive and negative review prediction was also very low.

Refinements and iteration

In this section, we will apply the lessons learned during the first approach and attempt to improve model performance.

```
# Import necessary libraries
import pandas as pd

import nltk

from sklearn.feature_extraction.text import CountVectorizer

import re
import os
```

```
#Import training data
movie_data = pd.read_csv('IMDB_dataset.csv')
```

```
print(movie_data.head(10))
```

	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
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Preprocessing

Our first step will be to alter the preprocessing steps.

Firstly, we'll create one function that will handle the preprocessing. This makes the code more readable, and less redundant. It's also simple to comment out one or more of the steps if needed and the process of applying the function to the dataframe is simplified.

The steps in the preprocess() function are as follows:

1. Remove HTML tags and junk text
2. Remove extra spaces
3. Convert all words to lowercase (so 'good' and 'Good' aren't counted as two different words)
4. Keep special characters like punctuation to make handling negation easier
5. Preserve negation by prepending logical negations with not_ as suggested by Jurafsky and Martin (2023).

```
#Preprocessing function

def preprocess(text):
    # Check if the input looks like a file name
    if os.path.isfile(text):
        with open(text, 'r') as file:
            soup = BeautifulSoup(file, "html.parser")
    else:
        soup = BeautifulSoup(text, "html.parser")

    cleaned_text = soup.get_text()

    # Replace [] with nothing
    cleaned_text = re.sub(r'\[[^\]]*\]', '', cleaned_text)

    # Remove excessive white spaces
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()

    # Convert to lowercase
```

```

cleaned_text = cleaned_text.lower()

# Preserve negation
cleaned_text = re.sub(r'\b(n\'t|not|no|never)\s+(\w+)', r'not_\1 \2',
cleaned_text)

return cleaned_text

# New Code

```

```

# Apply preprocessing to the dataframe
movie_data['review'] = movie_data['review'].apply(preprocess)

#Adapted code from Lakshmipathi N 2020

```

Furthermore, for this iteration we will avoid stemming the words to avoid impacting word frequencies in an unexpected way. We'll also avoid removing stopwords in this iteration; according to Jurafsky and Martin (2023) removing stopwords doesn't improve the performance of text classifications applications.

```

print(movie_data.head(10))

```

	review	sentiment
0	one of the other reviewers has mentioned that ...	positive
1	a wonderful little production. the filming tec...	positive
2	i thought this was a wonderful way to spend ti...	positive
3	basically there's a family where a little boy ...	negative
4	petter mattei's "love in the time of money" is...	positive
5	probably my all-time favorite movie, a story o...	positive
6	i sure would like to see a resurrection of a u...	positive
7	this show was an amazing, fresh & innovative i...	negative
8	encouraged by the positive comments about this...	negative
9	if you like original gut wrenching laughter yo...	positive

A final addition to the preprocessing step is to convert labels in the sentiment column to binary form. The 'positive' label is converted to a 1 and the 'negative' label to a 0. This is done in a much simpler way using .map() to make code more readable.

```

# Convert labels to binary
movie_data['sentiment'] = movie_data.sentiment.map({'positive':1,
'negative':0})

print(movie_data.head(10))

# New Code

```

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production. the filming tec...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically there's a family where a little boy ...	0
4	petter mattei's "love in the time of money" is...	1
5	probably my all-time favorite movie, a story o...	1
6	i sure would like to see a resurrection of a u...	1
7	this show was an amazing, fresh & innovative i...	0
8	encouraged by the positive comments about this...	0
9	if you like original gut wrenching laughter yo...	1

Classification approach

Test and training data split

Here we deviate from the Lakshmipathi notebook by splitting data into testing and training subsets before we vectorize the text data. This was done to simplify the variables that would ultimately be used to train and test models on and make it easier to pass the training and testing data on to other models if needed.

```
# Split data into training and testing data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(movie_data['review'], movie_data['sentiment'],
random_state=1)

print('Number of rows in the total set: {}'.format(movie_data.shape[0]))
print('Number of rows in the training set: {}'.format(X_train.shape[0]))
print('Number of rows in the test set: {}'.format(X_test.shape[0]))

# Code from Roman, 2019 and sklearn documentation
```

```
Number of rows in the total set: 50000
Number of rows in the training set: 37500
Number of rows in the test set: 12500
```

Feature extraction

```
# Instantiate the CountVectorizer method
count_vector = CountVectorizer()

# Fit the training data and then return the matrix
```

```

training_data = count_vector.fit_transform(X_train)

# Transform testing data and return the matrix.
testing_data = count_vector.transform(X_test)

#Adapted code from Roman 2019

```

Model training and predictions

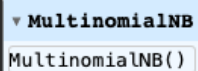
```

# Naive Bayes implementation
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()

naive_bayes.fit(training_data, y_train)

```



▼ MultinomialNB
MultinomialNB()

```

# Make predictions
predictions = naive_bayes.predict(testing_data)

```

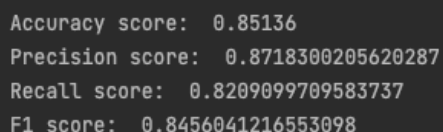
Model evaluation

Accuracy score

```

# Model metrics
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
print('Accuracy score: ', format(accuracy_score(y_test, predictions)))
print('Precision score: ', format(precision_score(y_test, predictions)))
print('Recall score: ', format(recall_score(y_test, predictions)))
print('F1 score: ', format(f1_score(y_test, predictions)))

```



```

Accuracy score:  0.85136
Precision score:  0.8718300205620287
Recall score:    0.8209099709583737
F1 score:        0.8456041216553098

```

Classification report

```

# Compile a classification report
from sklearn.metrics import classification_report

```

```
naive_bayes_report = classification_report(y_test, predictions,  
target_names = ['Positive', 'Negative'])  
print(naive_bayes_report)
```

	precision	recall	f1-score	support
Positive	0.83	0.88	0.86	6302
Negative	0.87	0.82	0.85	6198
accuracy			0.85	12500
macro avg	0.85	0.85	0.85	12500
weighted avg	0.85	0.85	0.85	12500

III Outcome

Performance

In our earlier discussion on how we would evaluate this project, we settled on the cutoffs below. There are three tiers of performance:

Score > 0.9	Excellent
Score >= 0.7	Acceptable
Score < 0.7	Poor

The table below provides a summary of the three metrics that were tracked. The attempt at simply replicating the original approach was performing poorly, with a low accuracy score and very low precision and recall scores.

The refined approach took into consideration some suggestions on the ideal process for preparing data for a Naive Bayes classifier from the literature, such as preserving negation and leaving out the step where text was stemmed and stopwords removed. We saw a significant increase to an acceptable performance.

Approach	Model	Accuracy	Precision	Recall
Baseline: Lakshmipathi N's results <ul style="list-style-type: none">Text stemmingStopword removalSpecial character removalFeature extraction before data split	Multinomial Naive Bayes	0.751 Baseline	0.75 Baseline	0.76 Baseline
Duplicating approach form Lakshmipathi N's Kaggle notebook <ul style="list-style-type: none">Text stemmingStopword removalSpecial character removalFeature extraction before data split	Multinomial Naive Bayes	0.4954 Poor	0.2256 Poor	0.0075 Poor
Refined approach: <ul style="list-style-type: none">Preserve punctuationPreserving negationNo stemmingNo stopwords removalData split before feature extraction with CountVectorizer and bag of words approach	Multinomial Naive Bayes	0.85136 Acceptable	0.87183 Acceptable	0.82099 Acceptable

Summary

The abundance of user generated reviews available from various sources presents an opportunity for the film industry to gain insights into audience perception, and may be used in decision-making and marketing approaches, and future developments.

This project focused on analysing sentiments of a dataset of movie reviews from IMDB, a popular movie review aggregator and film information website using NLP techniques and machine learning algorithms. Sentiment analysis, is a form of text categorization that aims to computationally determine the sentiment or opinion expressed by a piece of text.

The project's objective was to create a text classifier that can analyse and predict the sentiment of a large dataset. An existing Kaggle notebook analysing the dataset we chose to work with was used as the baseline for performance evaluation. The implementation of the project took two main steps; first attempting to simply recreate the Kaggle notebook approach and reproducing the results from the baseline. Secondly, to make improvements and changes to the approach informed by the literature to attempt to improve the model performance.

The process involved preprocessing the dataset, extracting features using the Bag of Words approach, and training a Multinomial Naive Bayes model from the Scikit Learn library

Results from the replication step were not very promising, and yielded an overall poor performance. Kaggle notebooks, like the movie reviews, are user generated material and not always bound to any requirements for performance or other benchmarks. Replicating another person's process was however a good learning opportunity and we could apply the lessons learned from the initial process into an improved second version.

The refined preprocessing model performed much better, scoring an acceptable level. The improvements were mainly because of a simplified feature extraction process and more care taken during text processing. That being said, in the future, the model performance could perhaps be improved even more, to the excellent tier, by perhaps expanding the feature extraction process.

We could, for example, count capitalised words in each review since people often use capitals to express strong sentiment in text. In a similar vein, a count of punctuation might also give us better insight into the sentiment.

Sentiment lexicons like SentiWordNet could also be used to match words in the review with related sentiment labels or scores.

Resources and bibliography

Ashar, Talha. (n.d.) What is the accuracy_score function in Sklearn? Available at: <https://www.educative.io/answers/what-is-the-accuracyscore-function-in-sklearn> (Accessed: 2 July 2023)

Bernico, M. (n.d.). Using Naive Bayes for Sentiment Analysis [Video file]. Retrieved from https://www.youtube.com/watch?v=oXZThwEF4r0&ab_channel=MikeBernico

Jurafsky, Daniel & Martin, James. (2023). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (pp. 142-150). Portland, Oregon, USA: Association for Computational Linguistics. Retrieved from <http://www.aclweb.org/anthology/P11-1015>

Metacritic. (n.d.). About Metacritic. Retrieved from <https://www.metacritic.com/about-metacritic>

Mulla, Rob. (2023, June 16). Python Sentiment Analysis Project with NLTK and 😊 Transformers. Classify Amazon Reviews!! [Video file]. Retrieved from https://www.youtube.com/watch?v=QpzMWQvxXWk&ab_channel=RobMulla

N, Lakshmipathi. (2020). "Sentiment Analysis of IMDB Movie Reviews." Kaggle notebook. Available at: <https://www.kaggle.com/code/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews/notebook> (Accessed: 18 June 2023).

Parashar, Nilesh. (2011). What is an Accuracy Score and How to Check it? Available at: <https://medium.com/@niitwork0921/what-is-an-accuracy-score-and-how-to-check-it-13b23eed6a3> (Accessed: 2 July 2023)

Roman, Victor. (2019) Naive Bayes Algorithm: Intuition and Implementation in a Spam Detector. Available at: <https://towardsdatascience.com/naive-bayes-intuition-and-implementation-ac328f9c9718> (Accessed: 2 July 2023)

Rotten Tomatoes. (n.d.). About Rotten Tomatoes. Retrieved from <https://www.rottentomatoes.com/about>

Valkov, V. (2019). Movie review sentiment analysis with Naive Bayes | Machine Learning from Scratch (Part V). Level Up Coding. Retrieved from

<https://levelup.gitconnected.com/movie-review-sentiment-analysis-with-naive-bayes-machine-learning-from-scratch-part-v-7bb869391bab> (Accessed: 2 July 2023)