Sentiment Analysis

For IMDB movie reviews

Team members:

Chong Xian Jun,
Wang Tian,
Li Yexin,
Ahmed hashim taha salim

Executive Summary

In this study, we conducted a sentiment analysis of movie reviews from IMDB to predict the overall sentiment of the audience towards a given movie. We used a combination of natural language processing techniques and machine learning algorithms to analyze the text of the reviews and classify them as either positive or negative.

We found that our model was able to accurately predict the sentiment of the reviews with a high degree of accuracy. Additionally, we also identified some common themes and words that were more prevalent in positive or negative reviews that becomes indicator of the audiences' sentiment.

Overall, our study provides valuable insights into the sentiment of movie audiences and can be used to inform the marketing and promotion of films.

Content List

Introduction and Dataset

- Background and Research Questions
- Introducing the iMDB Movie Review Dataset

Methodology

- Overview
- Preprocessing
- Model Training
- Model Evaluation

Results and Findings

- Models to Predict Sentiment
- Important Features

Conclusion

Introduction

Background and Research Questions

Conventionally, marketing and promotional campaigns for movies are based on historical data and tend to focus on movies that are similar to previously successful ones. This approach can exclude new or growing markets that might have different preferences.

To better capture the evolving preferences of movie audiences, we made use of natural-language processing methods and machine learning model to perform sentiment analysis on IMDB movie review data to build the predictive model. This can be helpful by automating the extraction of audiences' sentiment on a movie efficiently, thereby informing the promotion and distribution of film.

The study aims to answer the following questions:

1. How do we predict the overall sentiment of the audiences on a particular movie with reviews?
2. What are the important themes that can indicate the audience sentiment about a movie?

IMDB Movie Reviews Dataset

Source: [IMDB Dataset of 50K Movie Reviews | Kaggle]

- Consist of only two columns, “review” and “sentiment”.
- The sentiments are labelled as positive or negative, with each sentiment consist of 25000 samples.
- Contains many syntaxes, symbols and punctuations to be pruned off before analysing with NLP methods

	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

Dataset samples

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB

{'positive': 25000, 'negative': 25000}
```

Dataset info

Methodology

Methodology Overview

Preprocessing

Normalization

Converting text into a standardized representation.

1. Cleaning
2. Tokenization
3. Lemmatization

Vectorization

Converting text into a numerical representation

1. TF-IDF

Model Training and Evaluation

- Logistic Regression
- Neural Network
- Multinomial Naive Bayes

Evaluated by **accuracy** and **speed.**

Preprocessing

Normalization (Cleaning)

The steps involved in cleaning vary according to data. For the movie reviews, we

1. remove html syntax
2. remove url
3. standardise letter casing
4. fix contractions
5. remove stopwords
6. remove all that are not words

Before actually cleaning, we write regex function to check if they are properly cleaned.
Call function:

`check_if_contain(data, x, pattern)`

```
# define function to check whether they consist of unwanted stuffs with regex
def search_review(target_str, pattern):
  cpat = re.compile(pattern)
  if cpat.search(target_str):
    return True
  else:
    return False

def check_if_contain(data, x, pattern, contain=True):
  # avoiding 'contains_{x}' columns accumulating, set up temp_df
  temp_df = data.copy()
  temp_df[f'contains_{x}'] = temp_df.apply(lambda x: search_review(x['review'], pattern), axis=1)
  return temp_df[temp_df[f'contains_{x}'] == contain]
```

1. remove html syntax
    - Use html-parser from ‘BeautifulSoup’ to get rid of the html syntaxes.

```python
# clean html
cleaned_df.review = cleaned_df.review.apply(lambda string: BeautifulSoup(string, 'html.parser').get_text())

html_regex = r'<.*?>'
check_if_contain(cleaned_df, 'symbol', html_regex, True)
```

	review	sentiment	contains_symbol
1513	This all-but-ignored masterpiece is about the ...	positive	True
13736	>>> Great News there is a BBC DVD release sche...	positive	True
22164	In Truffaut book-length interview with Hitchco...	positive	True
27107	One minute into THE UNTOLD and it`s already ri...	negative	True
33891	and quite frankly that just sums it up.It is a...	positive	True
42217	Well, I guess I'm emotionally attached to this...	positive	True
48617	<-----Minor Spoilers!----->A woman gets pregna...	positive	True

2. remove url

- remove url with regex pattern starting with https:// or www.

```
# clean url and check if there's still url
cleaned_df.review = cleaned_df.review.str.replace(r"https://\S+|www\.\S+", '', regex = True)
```

3. Standardise letter casing

- apply lower() function

```
[9]
# standardise casing to lower case and check
cleaned_df.review = cleaned_df.review.apply(lambda x: x.lower())

upper_case_regex = r'[A-Z]'
check_if_contain(cleaned_df, 'upper', upper_case_regex, True)
```

review	sentiment	contains_upper

4. Fix contractions

- contractions such as I’m, they’re… are expanded with regex

```
replacement_patterns = [
  (r'won\'t', 'will not'),
  (r'can\'t', 'cannot'),
  (r'i\'m', 'i am'),
  (r'ain\'t', 'is not'),
  (r'(\w+)\'ll', '\g<1> will'),
  (r'(\w+)n\'t', '\g<1> not'),
  (r'(\w+)\'ve', '\g<1> have'),
  (r'(\w+)\'s', '\g<1> is'),
  (r'(\w+)\'re', '\g<1> are'),
  (r'(\w+)\'d', '\g<1> would')
]
```

```
class RegexpReplacer(object):
  def __init__(self, patterns=replacement_patterns):
    self.patterns = [(re.compile(regex), repl) for (regex, repl) in patterns]
    # regex applies to regex pattern to be replaced
    # repl refers to the replacements

  def replace(self, text):
    s = text
    for (pattern, repl) in self.patterns:
      s = re.sub(pattern, repl, s)
    return s
```

```
replacer = RegexpReplacer()
cleaned_df.review = cleaned_df.review.apply(lambda x: replacer.replace(x))
```

4. Fix contractions (cont)

- check if there is still contractions

```
contraction_regex = r"(?!\")\[A-Za-z]+('[A-Za-z]*)\b"
# \b matches word boundary, ensuring regex matches complete word not only part of it
# [A-Za-z]+ matches first part of the word
# ('[A-Za-z]*) matches the contraction part
# (?!\") checks that the it does not detect a quotation mark (") at the beginning and take as contraction

check_if_contain(cleaned_df, "contraction", contraction_regex, True)
```

review	sentiment	contains_contraction

5. Remove stopwords

- make use of nltk package to remove stop words such as “I’m, myself, the…”

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
nltk.download('stopwords')
nltk.download('punkt')
```

```
stopwords = nltk.corpus.stopwords.words('english')

# define a function to remove stopwords from a string
# not compulsory, it can be done by vectorization
def remove_stopwords(s):
    # split the string into a list of words
    words = word_tokenize(s)

    # remove stopwords from the list of words
    filtered_words = [w for w in words if w not in stopwords]

    # join the filtered words back into a string
    return " ".join(filtered_words)
```

```
cleaned_df.review = cleaned_df.review.apply(remove_stopwords)
```

5. Remove stopwords

- check the first review

```
one reviewers mentioned watching 1 oz episode hooked . right , exactly happened me.the first thing
struck oz brutality unflinching scenes violence , set right word go . trust , show faint hearted timid .
show pulls punches regards drugs , sex violence . hardcore , classic use word.it called oz nickname
given oswald maximum security state penitentary . focuses mainly emerald city , experimental section
prison cells glass fronts face inwards , privacy high agenda . em city home many .. aryans , muslims ,
gangstas , latinos , christians , italians , irish .... scuffles , death stares , dodgy dealings shady
agreements never far away.i would say main appeal show due fact goes shows would dare . forget pretty
pictures painted mainstream audiences , forget charm , forget romance ... oz mess around . first episode
ever saw struck nasty surreal , could say ready , watched , developed taste oz , got accustomed high
levels graphic violence . violence , injustice ( crooked guards sold nickel , inmates kill order get
away , well mannered , middle class inmates turned prison bitches due lack street skills prison
experience ) watching oz , may become comfortable uncomfortable viewing .... thats get touch darker side
```

6. Remove all that are not words

- use regex to choose only words and white space

```
2 symbol_regex = r'[^a-z|\s]'
3 cleaned_df.review = cleaned_df.review.str.replace(symbol_regex, '', regex = True)
```

The first review:

```
one reviewers mentioned watching  oz episode hooked  right  exactly happened methe first thing struck oz brutality unflinching scenes violence  set right word go  trust  show faint hearted timid  show pulls punches regards drugs  sex violence  hardcore  classic use wordit called oz nickname given oswald maximum security state penitentary  focuses mainly emerald city  experimental section prison cells glass fronts face inwards  privacy high agenda  em city home many  aryans  muslims  gangstas  latinos christians  italians  irish  scuffles  death stares  dodgy dealings shady agreements never far awayi would say main appeal show due fact goes shows would dare  forget pretty pictures painted mainstream audiences  forget charm  forget romance  oz mess around  first episode ever saw struck nasty surreal could say ready  watched  developed taste oz  got accustomed high levels graphic violence  violence injustice  crooked guards sold nickel  inmates kill order get away  well mannered  middle class inmates turned prison bitches due lack street skills prison experience  watching oz  may become comfortable uncomfortable viewing  thats get touch darker side
```

Preprocessing

Normalization (Tokenization + Lemmatization)

Tokenization is the process of extracting words as separate tokens.
Lemmatization is the process of returning words into its base form, e.g. “plays” -> “play”

Tokenization is necessary before lemmatization as it allows the lemmatization algorithm to operate on individual words rather than on the entire text. This makes the lemmatization process more efficient and accurate.

Several packages from nltk library are used:

```
# packages to lemmatize the dataset
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('wordnet')
nltk.download('omw-1.4')
```

Normalization (Tokenization + Lemmatization)

```
def lemmatize_string(s):
  lemmatizer = WordNetLemmatizer()
  # split the string into a list of words (tokenization)
  words = word_tokenize(s)

  # lemmatize each word in the list
  lemmas = [lemmatizer.lemmatize(w) for w in words]

  # join the lemmas back into a string
  return " ".join(lemmas)
```

```
cleaned_df['lemma_review']=cleaned_df['review'].apply(lambda z: lemmatize_string(z))
```

Lemmatized review sample compared with that before lemmatization:

```
1 cleaned_df.lemma_review[0]
```

```
'one reviewer mentioned watching oz episode hooked right exactly happened me the first thing struck oz brutality unflinching scene violence set right word go trust show faint hearted timid show pull punch regard drug sex violence hardcore classic use word it called oz nickname given oswald maximum security state penitentary focus mainly emerald city experimental section prison cell glass front face inwards privacy high agenda em city home many aryan muslim gangsta latino christian italian irish scuffle death stare dodgy dealing shady agreement never far away i would say main appeal show due fact go show would dare forget pretty picture painted mainstream audience forget charm forget romance oz mess around first episode ever saw struck nasty surreal could say ready watched developed taste oz got accustomed high level graphic violence violence injustice crooked guard sold nickel inmate kill order get away well mannered middle class inmate turned prison bitch due lack street skill prison experience watching oz may become comfortable uncomfortable viewing thats get touch darker side<'
```

```
1 cleaned_df.review[0]
```

```
'one reviewers mentioned watching   oz episode hooked   right   exactly happened me the first thing struck oz brutality unflinching scenes violence   set right word go   trust   show faint hearted timid   show pulls punches regards drugs   sex violence   hardcore   classic use word it called oz nickname given oswald maximum security state penitentary   focuses mainly emerald city   experimental section prison cells glass fronts face inwards   privacy high agenda   em city home many    aryans   muslims   gangstas   latinos   christians   italians   irish      scuffles   death stares   dodgy dealings shady agreements never far away i would say main appeal show due fact goes shows would dare   forget pretty pictures painted mainstream audiences   forget charm   forget romance    oz mess around   first episode ever saw struck nasty surreal   could say ready   watched   developed taste oz   got accustomed high levels graphic violence   violence   injustice   crooked guards sold nickel   inmates kill order get away   well mannered   middle class inmates turned prison bitches due lack street skills prison experience   watching oz   may become comfortable uncomfortable viewing      thats get touch darker side   <'
```

Preprocessing

Vectorization

Encode the texts into numerical form as before inputting the data into machine learning models.
For this study, TF-IDF algorithm is used due to its ability to take into account the frequency and context of words.

Before vectorization, the normalized dataset is split into X_train, X_test, y_train, y_test.
The following is the code for TF-IDF vectorization:

```
from sklearn.feature_extraction.text import TfidfVectorizer
t_vect = TfidfVectorizer(max_df = 0.8, min_df = 3, tokenizer=word_tokenize,
                         stop_words=frozenset(stop_words_tokens), max_features=5000)

X_tfidfvect_train = pd.DataFrame(t_vect.fit_transform(X_train).toarray(),
                                 columns=t_vect.get_feature_names_out())
X_tfidfvect_test = pd.DataFrame(t_vect.transform(X_test).toarray(),
                                columns=t_vect.get_feature_names_out())
```

Vectorized output, `X_tfidfvect_train`

```
12 X_tfidfvect_train.head()
```

	a	aaron	abandoned	abc	ability	able	absence	absent	absolute	absolutely	...
0	0.053855	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

`X_tfidfvect_train.shape= (50000, 5000)`

Vectorize y_train and y_test to numerical values as well. {‘positive’ : 1, ‘negative’: 0}

```python
def encode_y(y):
  y[y == 'positive'] = 1
  y[y == 'negative'] = 0
  y.astype(int)
  return y

y_train = encode_y(y_train).astype(int)
y_test = encode_y(y_test).astype(int)
```

Model Training and Evaluation

Models trained are:

- Logistic Regression
- Neural Network
- Multinomial Naive Bayes

Using sklearn API, we are able to train and test different models easily. In our study, the models are evaluated by their accuracy as the data has equal class distribution. The speed of models are evaluated by using time() function as well to determine the best model.

- Logistic Regression

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()

t0 = time()    # start time
lr.fit(X_tfidfvect_train, y_train)
print ("training done in %fs" %(time() - t0))   # end of training

t1 = time()
lr_accuracy = lr.score(X_tfidfvect_test, y_test)
print ("testing done in %fs" %(time() - t1))
print('logistic regression accuracy:', lr_accuracy)
```

```
training done in 6.706810s
testing done in 0.146578s
logistic regression accuracy: 0.8864
```

- Neural Network

```
# neural network
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1, max_iter=200)

t0 = time()    # start time
mlp.fit(X_tfidfvect_train, y_train)
print ("training done in %fs" %(time() - t0))   # end of training

t1 = time()
mlp_accuracy = mlp.score(X_tfidfvect_test, y_test)
print ("testing done in %fs" %(time() - t1))

print('mlp accuracy:', mlp_accuracy)
```

```
training done in 516.074917s
testing done in 0.197302s
mlp accuracy: 0.8795333333333333
```

- Multinomial Naive Bayes

```
#Multinomial Naive Bayes
from sklearn.naive_bayes import MultinomialNB
mnb=MultinomialNB()

t0 = time()    # start time
mnb.fit(X_tfidfvect_train,y_train)
print ("training done in %fs" %(time() - t0))   # end of training

t1 = time()
mnb_accuracy = mnb.score(X_tfidfvect_test, y_test)
print ("testing done in %fs" %(time() - t1))

print('mnb accuracy:', mnb_accuracy)
```

```
training done in 0.396690s
testing done in 0.141874s
mnb accuracy: 0.8536666666666667
```

Results and Findings

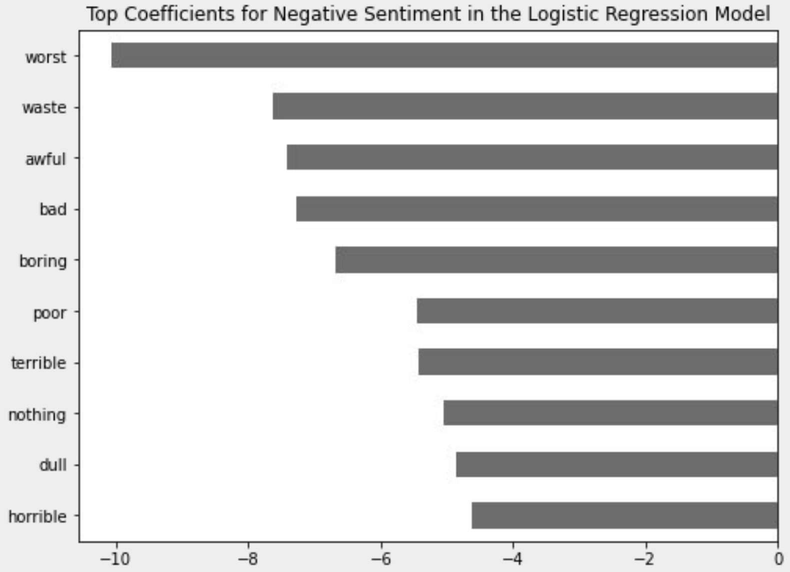
Results and Findings

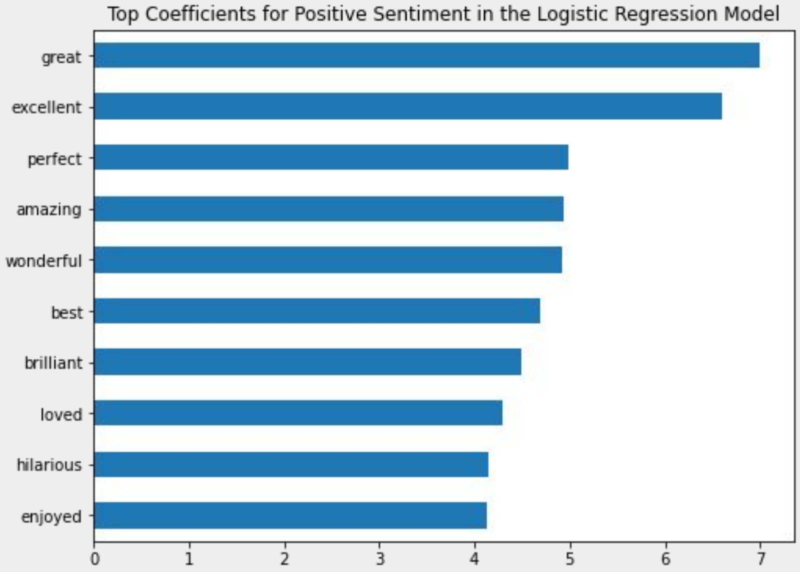
All in all, we can make use of predictive models to predict audience sentiment based on their review comments. For this purpose, **logistic regression** is the best performing model as it has the highest accuracy with fairly low training and testing time.

	Models		
	Logistic Regression	Multinomial Naive Bayes	Neural Network
Accuracy	0.8864	0.8537	0.8795
Training Time (s)	6.706810	0.396690	509.5316
Testing Time (s)	0.146578	0.141874	0.1730

Results and Findings

From the logistic regression, we extract the most important features (words) that people associate strongly with positive or negative sentiments.





Conclusion

Conclusion

In this study, we conducted a sentiment analysis of movie reviews from IMDB to predict the sentiment of the audience towards a given movie based on unstructured review. It is important to note that thorough preprocessing involving text normalization and vectorization is needed before model training. Our findings showed that logistic regression was able to accurately predict the sentiment of the reviews with accuracy of 0.8864 and fairly efficient training time. he common themes and words that were indicative of positive or negative sentiment are extracted from the model to allow for interpretation.

These findings have important implications for the film industry as they can be used to inform the marketing and promotion of movies, helping the stakeholders to capture the new market with ever-evolving preferences on movies.