Final_Project_Writeup_Movie_Review

December 14, 2020

1 Sentiment Analysis on Movie Reviews

Project mentor: Zach Wood-Doughty

Shihong Wei swei15@jh.edu, Jingyi Ren jren20@jh.edu, Xuan Wu xwu78@jh.edu, Zihui Wang zwang220@jh.edu

Link to Github Repository: GitHub: JHU_CS675_Project_SentimentAnalysisonMovieReviews

2 Outline and Deliverables

2.0.1 Outline

There are thousands of movies coming out each year. Movie reviews reflect the quality of movies, and influence many people in their choice of movie-watching. In this project, we want to perform a thorough sentiment analysis to identify the polarity of textual reviews. This research work can hopefully help viewers decide whether to watch a newly released movie or not. It may also be of interest to the movie industry to tell what kind of movies the market will like and help recommend movies to users based on previous reviews. Each input is a paragraph of movie review consisting of several English sentences. We will train and develop different SVM, neural network and LSTM models to predict whether the movie review is positive or negative. Then, with the trained models, we will investigate what are key features in a movie review that reveals most of its attitude (e.g., frequency of particular key words). Furthermore, we will conduct a comparison analysis in following two aspects: First, we will compare different texture feature engineering techniques such as word2vec, N-Gram, TFIDF. Second, we will compare those best performing models in each classes based not only on their test accuracy, time complexity, but also on their fairness and interpretability (details explained in the deliverables section).

2.0.2 Uncompleted Deliverables

1. High test accuracy of neural network models (CNN and LSTM): We have not achieved a very satisfactory test accuracy for neural network models, becuase training it is very time consuming. Also, after we do the vectorization, the CNN layer may not be very meaningful. for example, after we perform the TFIDF/Ngram vectorization of the top 1500 words, the idea "neighborhood" is not interpretable. The 1st number in a document vector has just as much relation to the 2nd and to the 1500th. For discussion of Word2vec,please see Section **??**
2. As is discussed in the presentation, we did not use the our homework codes for TFIDF and NGram vectorization for the following to reasons: First, we want to accelerate model

training implementation and focus more on analysis rather than devoting too much time on repetiation of programming hws. Second, the python function provide more user-specified hyperparameters for ease of grid search and fair comparison

2.0.3 Completed Deliverables

1. Appropriate data preprocessing: Correct tokenization of the raw textual data should be performed. Details are explained in the . In particular, for the N-Gram and TFIDF part, we used "sklearn", and for the word2vec part we will learn and use the python library named "word2vec". Plase see Section **??**
2. Valid modeling procedure and criteria: The standard training-validation-testing split should be followed to ensure fair comparison and to avoid information leakage. We will select different model structures within SVM, NN, or LSTM based on both accuracy and information criteria. Besides modifying the relevant homework codes, the libraries are stated in the methods section. Please see Section **??** and Section **??**
3. Exploration of model interpretation in textual analysis: For example, we want to explore what are the key words that appears most frequently in the movie review contexts, and how they will affects our trained model in classify an example. For example, please see Section **??** and Section **??**
4. Optimal model parameters and hyperparameters (for example, the number of training epochs, the size of training batches and the penalty C in SVM) were chosen based on some grid search methods. Please see the the python notebook in the github repo Python Notebook Link in Githhub
5. High test accuracy of logistic regression and SVM models: With a relatively large-sized and potentially representative sample, we expect our trained model to perform well on the testing set under at least one textual feature engineering techniques among word2vec, TFIDF and N-Gram. Please see Section **??**
6. Model comparison across different classes: For the best performing SVM, NN, LSTM models, we want compare their relative test accuracy and rough time complexity, and we want to explain at least partially the reason behind it. Please see Section **??**
7. Detecting and handling potential overfitting problem when training the models. We have tried out techniques such as l1 and l2 regularization in the grid search of logistic regression, and dropout in neural network models. Please see the the python notebook in the github repo Python Notebook Link in Githhub
8. Further evaluation of the model performance and fairness based on the domain of movie review. For example, we have examined whether our trained model is fair in dealing with the review texts of different types of movies. Please see Section **??** and the the python notebook in the github repo Python Notebook Link in Githhub
9. Moreover, we will learned and tried out some web crawler algorithms to construct an extra testing data set on IMDB that includes information such as movie type and country. Please see the python notebook in the Github repoWeb Clawer Algo

3 Preliminaries

Real-World Implication

1. To get a model with high accuracy, it's better to train the sentiment analysis model based on data collected from professional movie review websites like IMDB. In this way, we were trying to capture the features that people used to express their attitude about a movie. Then for a new-releasesd movie, it is difficult to form an objective rate because few people see them. But there would be some people who share their experience of watching the movie on Twitter, Facebook and other places. The audience can make decisions with the assistence of our model prediction results without reading many introductions, reviews from different websites.

2. On the other hand, we can train the model based on dataset of different movie genres and runtime to see the whether genres or runtime would have an influence on people's preference, which furtheremore, helps the production company to analyze the market situation.

Similarity to Lecture/ Breakout/ Homework It's a binary classification problem, similar to what we learned in HW3 Lab. And we were trying to use contents of Lecture 22 to deal with texture analysis. Besides, we have already learned how to transform texture data to numerical features by Kernel SVM in HW4 Programming part, which would help us to preprocess data and do the feature selection.

What makes this problem unique? As we talked in Real-World Implication, it provides reference for the selection of newly-released movies. Although the dataset we collected just come from the IMDB website, the test data can be movie reviews from Twitter, Facebook, Instagram and other movie websites. Also, we can check the performance of our model based on various movie genres, runtimes because "it's terrifying" sounds positive for a horror, but might not good for an animation.

Ethical Implications

1. The dataset is representative of English speakers, not opinions from viewers of other cultures, which means we are doing the sentiment analysis of part popularion in the world. The reviews can only reflect the values and cultural preferences of some people, not a general case.
2. Numerical features we learned from the huge dataset correspond to various word, which means for a machine without feelings, it can produce positive or negative reviews. And public relations or paid posters may write many fake reviews in this way to lead the public opinion.
3. Movie reviews are related with the movie itself, the actors and directors. So fake reviews may have a possible negative social impact on movie castings. On the other hand, some propaganda team might produce many fake reviews to attract the attention of the public so that people remeber the actors' name and want to see the movie. And people may recommend the good movie to their friends. Eventually, it will have a positive social impact on this actor.

3.1 Dataset(s)

We used the Large Movie Review Dataset, which is a collection of English textual movie reviews extracted before year 2011 from IMDB website, each with a label being either positive or negative. There are around 25,000 examples in the original training set, and around 25,000 examples in the original testing set, so we used a training (70%)-validation (15%)- testing (15%) split.

Furthermore, because the IMDB dataset does not include example information such as movie type, runtime, when evaluating the fairness of the model, we tried out web crawler algorithms on IMDB website to gather an extra testing set for movie reviews after year 2013, 500 reviews for each of 8 genres (action, adventure, animation, biography, comedy, horror, romance, and sci-fi) and runtime (less than 100 minutes; larger than 100 minutes).

Next, let us load our data and print out some examples.

[ ]:
```
# Load your data and print 2-3
!unzip -q '/content/archive.zip' -d '/content'
import os
print(os.listdir("/content"))
import warnings
warnings.filterwarnings('ignore')

#importing the training data
import pandas as pd
imdb_data=pd.read_csv('/content/IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(3)
```

```
['.config', 'archive.zip', 'IMDB Dataset.csv', 'sample_data']
(50000, 2)
```

[ ]:

	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

[ ]:
```
df = pd.DataFrame(imdb_data)
df.columns=['review','label']
```

[ ]:
```
df.iloc[1][0] #positive
```

[ ]:
```
'A wonderful little production. <br /><br />The filming technique is very
unassuming- very old-time-BBC fashion and gives a comforting, and sometimes
discomforting, sense of realism to the entire piece. <br /><br />The actors are
extremely well chosen- Michael Sheen not only "has got all the polari" but he
has all the voices down pat too! You can truly see the seamless editing guided
by the references to Williams\' diary entries, not only is it well worth the
watching but it is a terrificly written and performed piece. A masterful
production about one of the great master\'s of comedy and his life. <br /><br
```

```
/>The realism really comes home with the little things: the fantasy of the guard
which, rather than use the traditional \'dream\' techniques remains solid then
disappears. It plays on our knowledge and our senses, particularly with the
scenes concerning Orton and Halliwell and the sets (particularly of their flat
with Halliwell\'s murals decorating every surface) are terribly well done.'
```

[ ]:
```
df.iloc[3][0] #negative
```

[ ]:
```
"Basically there's a family where a little boy (Jake) thinks there's a zombie in
his closet & his parents are fighting all the time.<br /><br />This movie is
slower than a soap opera... and suddenly, Jake decides to become Rambo and kill
the zombie.<br /><br />OK, first of all when you're going to make a film you
must Decide if its a thriller or a drama! As a drama the movie is watchable.
Parents are divorcing & arguing like in real life. And then we have Jake with
his closet which totally ruins all the film! I expected to see a BOOGEYMAN
similar movie, and instead i watched a drama with some meaningless thriller
spots.<br /><br />3 out of 10 just for the well playing parents & descent
dialogs. As for the shots with Jake: just ignore them."
```

3.2 Pre-processing

Our datasets has binary labels, where 1 represents positive sentiment and 0 represents negative sentiment, and they are class-balanced. There is no missing data or outliers in our datasets.

To prepocess the review text, we removed html strips, square brackets, noisy text, special characters, and stopwords.

To extract features from the raw textual data, we use four different vectorization methods to transform the text into vectors: bags of words, ngram of characters, TFIDF, and Word2Vec, and we select top words (or n-characters) that appear most frequently as our features.

[ ]:
```
# For those same examples above, what do they look like after being␣
 ↪pre-processed?
import numpy as np
import pandas as pd
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.tokenize.toktok import ToktokTokenizer
from bs4 import BeautifulSoup
import spacy
import re,string,unicodedata
df = pd.DataFrame(imdb_data)
df.columns=['review','label']
df.head()

import nltk
nltk.download('stopwords')

#Tokenization of text
```

```
tokenizer=ToktokTokenizer()
#Setting English stopwords
from nltk.corpus import stopwords
stopword_list=stopwords.words('english')

#Removing the html strips
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
df['review']=df['review'].apply(denoise_text)

#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
df['review']=df['review'].apply(remove_special_characters)

#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
```

```
df['review']=df['review'].apply(remove_stopwords)

df2 = df.copy()
df2.label[df2.label=="positive"] = 1
df2.label[df2.label=='negative'] = 0
df2.head(3)
```

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

[ ]:

	review	label
0	One reviewers mentioned watching 1 Oz episode ...	1
1	wonderful little production filming technique ...	1
2	thought wonderful way spend time hot summer we...	1

Data Visualization using Wordcould

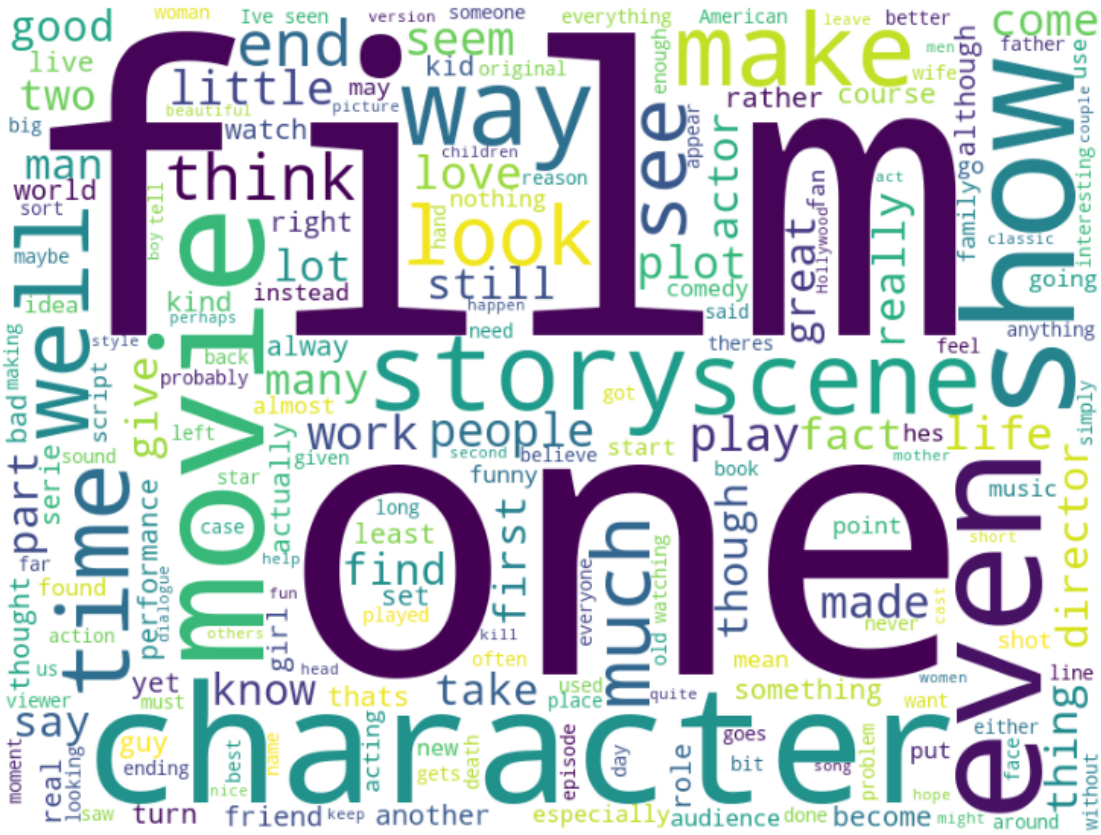
[ ]:
```
# Visualize the distribution of your data before and after pre-processing.
#   You may borrow from how we visualized data in the Lab homeworks.
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

%matplotlib inline
def cloud(data,backgroundcolor = 'white', width = 800, height = 600):
```

```
    wordcloud = WordCloud(stopwords = STOPWORDS, background_color =␣
 ↪backgroundcolor,
                          width = width, height = height).generate(data)
    plt.figure(figsize = (15, 10))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()

# before preprocessing
raw = pd.DataFrame(imdb_data)
raw.columns=['review','label']
cloud(' '.join(raw.review))
```


```
[ ]: # after preprocessing
cloud(' '.join(df2.review))
```



4 Models and Evaluation

4.1 Experimental Setup

To evaluate the performance of our model, we compute TN, FP, TP, FN, Accuracy, Precision, Recall, Specificity, FPR, F1, and Balanced_Accuracy. The mothods are reasonable because we have binary labels, and these evaluations show how much portion of the data we predict correctly.

The loss function we use for logistic regression and SVM is hinge_squared. We did grid search on multiple hyperparameters including loss function, and it gives us the best result.

For our main dataset, we use 70% as the train set to fit models, 15% as the validation set to tune hyperparameters, and 15% as the test set to do comparison for different models. We have additional test set where we get from IMDB using web crawler to test the fairness of our models, whether they perform differently for different movie genres and duration.

[ ]:
```
# Code for loss functions, evaluation metrics or link to Git repo
def evaluate_pred(ytrue, ypred):
  TN, FP, FN, TP = confusion_matrix(ytrue, ypred).ravel()
  Accuracy = round((TP + TN)/(TP + TN + FP + FN),3)
  Precision = round(TP/(TP + FP),3)
```

```
  Recall = round(TP/(TP + FN),3)
  Specificity = round(TN/(TN + FP),3)
  FPR = round(FP/(FP + TN),3)
  F1 = round(2*(Precision * Recall)/(Precision + Recall),3)
  Balanced_Accuracy = round((Precision + Specificity)/2,3)
  print("TP = "+str(TP))
  print("FP = "+str(FP))
  print("FN = "+str(FN))
  print("TN = "+str(TN))
  print("Accuracy = "+str(Accuracy))
  print("Precision = "+str(Precision))
  print("Recall = "+str(Recall))
  print("Specificity = "+str(Specificity))
  print("False_Positive_Rate = "+str(FPR))
  print("F1_Score = "+str(F1))
  print("Balanced_Accuracy = "+str(Balanced_Accuracy))
  res=pd.DataFrame([TN, FP, FN, TP, Accuracy, Precision, Recall, Specificity,␣
 ↪FPR, F1, Balanced_Accuracy])
  return(res)

res=pd.concat([res_test, res_runtime_1_100, res_runtime_101_600, res_action,␣
 ↪res_adventure, res_animation, res_biography, res_comedy, res_horror,␣
 ↪res_romance, res_scifi], axis=1, ignore_index=True)
res.columns=['Test_Split', 'Runtime_1_100', 'Runtime_101_600','Action',␣
 ↪'Adventure', 'Animation','Biography', 'Comedy', 'Horror','Romance', 'Sci_fi']
res.index=['TN', 'FP', 'FN', 'TP',' Accuracy', 'Precision', 'Recall',␣
 ↪'Specificity', 'FPR', 'F1', 'Balanced_Accuracy']
```

4.2 Baselines

We chose Logistic Regression (LR) with Bag-of-Words as our basline model. It is because our problem is binary text classification and LR is a widely used model for binary classification, and Bag-of-Words is commonly used in text classification where the frequency of each word is used as a feature for training a classifier.

In the previous studies, CNN with term frequency vectorization has been established, which accuracy is 84.67%. (https://github.com/ovguyo/moviereview)

Also Random Forest with {1,2}-Bag of Words vectorization has been established and its accuracy reaches around 81%. (https://www.kaggle.com/ramanchandra/sentiment-analysis-on-imdb-movie-reviews/data)

4.3 Methods

We chose Linear Regression, SVM, CNN and LSTM with different vectorization techniques (word2vec, character N-Gram and TFIDF). Since our input data is textual, we made our data numerical by those different vectorization techniques. Our label is "Positive" and "Negative", so we chose methods which usally handle binary classification problem well.

We used model libraries like sklearn.linear_model (for LR), sklearn.svm (for SVM) and keras (for CNN and LSTM) to train these models using our training set, and used hyperparameter grid search (sklearn.model_selection) to find optimal hyperparameters using validation data. Taking hinge, squared hinge with regularization (l1, l2) as loss funtions, we measured our model performance by TN, FP, FN, TP, Accuracy, Precision, Recall, Specificity, FPR, F1 Score and Balanced Accuracy. Moreover, we consider time complexity, ease of implementation, fairness and interpretability for evaluating our models.

In terms of difficulties, LR and SVM are relatively easier to train. But the situation is different for CNN and LSTM, since it is hard to find the best architecture of CNN and training LSTM is really time consuming.

For logistic regression, we searched the best hyperparameters for penalty, dual, class_weight, solver and inverse of regularization. For SVM, we evaluated loss function, class_weight, inverse of regularization and penalty. Aftering using optimal hyperparameters we searched by GridSearchCV, our models performed a little bit better than before.

The Code and results for training different models with features extracted by different vectorization can be found in the following Github repo folder Python Notebooks.

Each notebook in the foler coresponds to the models with one vectorization.

4.4 Results

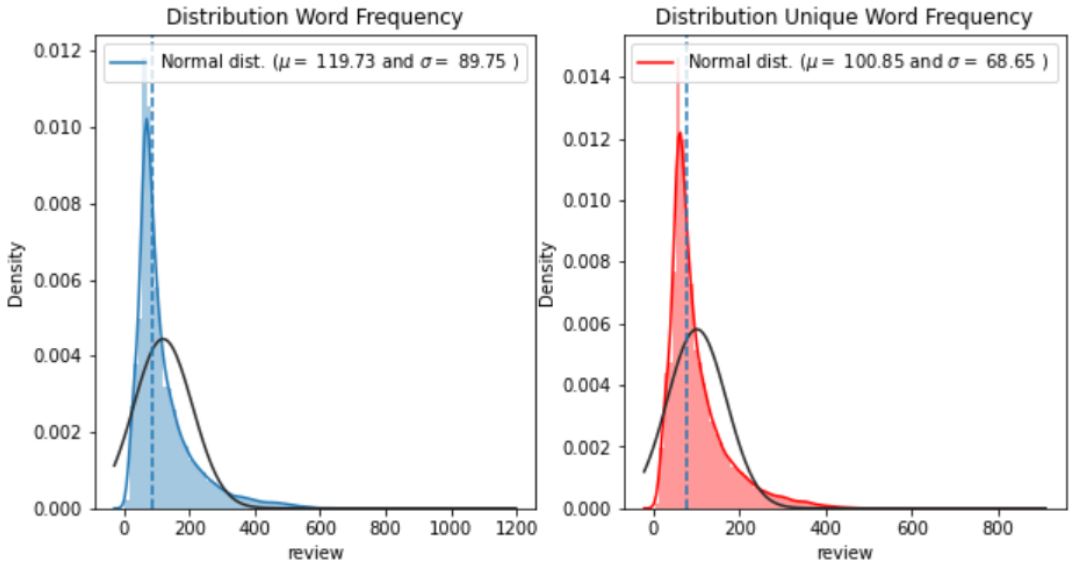
4.4.1 Results with Baseline Vectorization (Bag-of-Word)

The codes and results details in this part can be found in python notebook Bag Of Words at our Github project repository

We try 2-word and 3-word vectorization, and select top 1000 features which appear at least in 5 reviews. After transforming the train set reviews into vectors, we fit a logistic regression model. For the 2-word model, the test accuracy is around 0.754, and for the 3-word model, the test accuracy is arount 0.617.

4.4.2 Results with TFIDF Vectorization

The codes and results details in this part can be found in python notebook TFIDF Vectorization at our Github project repository



Thus, we select the top 1500 words during the initial TFIDF vectorization, and then applying PCA to reduce the input dimension to around 400. After ranking the TFIDF scores of each unique words in training set, we have that the top 30 words in the measure of TFIDF score summing over the training sets are as follows:

	tfidfscore
movie	1843.980008
film	1622.581414
one	1315.324304
like	1155.835807
good	1032.563317
see	896.793586
even	890.822372
would	890.627754
really	887.135275
time	882.141362
story	875.501801
great	812.139414
bad	787.202219
much	774.062155
well	765.752353

As we can see, this is in line with our initial understanding of what kind of words may appear in the movie reviews.

First, with TFIDF vectorization and logistic regression model, the best model we have selected from the grid search has the following hyperparameters: {'C': 10, 'penalty': 'l2', 'solver': 'newton-cg'}.

With the best trained logistic regression model, we have the following performance on the nine

testing sets

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	3226.000	164.000	148.000	173.000	165.000	61.000	110.000	158.000	205.000	136.000	186.000
FP	570.000	86.000	102.000	77.000	85.000	55.000	121.000	92.000	45.000	114.000	64.000
FN	487.000	66.000	41.000	65.000	47.000	10.000	27.000	39.000	72.000	40.000	81.000
TP	3217.000	184.000	209.000	185.000	203.000	106.000	204.000	211.000	178.000	210.000	169.000
Accuracy	0.859	0.696	0.714	0.716	0.736	0.720	0.680	0.738	0.766	0.692	0.710
Precision	0.849	0.681	0.672	0.706	0.705	0.658	0.628	0.696	0.798	0.648	0.725
Recall	0.869	0.736	0.836	0.740	0.812	0.914	0.883	0.844	0.712	0.840	0.676
Specificity	0.850	0.656	0.592	0.692	0.660	0.526	0.476	0.632	0.820	0.544	0.744
FPR	0.150	0.344	0.408	0.308	0.340	0.474	0.524	0.368	0.180	0.456	0.256
F1	0.859	0.707	0.745	0.723	0.755	0.765	0.734	0.763	0.753	0.732	0.700
Balanced_Accuracy	0.849	0.669	0.632	0.699	0.682	0.592	0.552	0.664	0.809	0.596	0.734

Second, with TFIDF vectorization and support vector machine (SVM) model, we have the following optimal hyperparameters during the grid search {'C': 0.3, 'loss': 'squared_hinge', 'penalty': 'l2'}.

With the best trained SVM model, we have the following performance on the nine testing sets

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	3222.000	164.000	148.000	174.000	164.000	61.000	103.000	156.000	205.000	135.000	187.000
FP	574.000	86.000	102.000	76.000	86.000	55.000	128.000	94.000	45.000	115.000	63.000
FN	479.000	67.000	41.000	65.000	46.000	8.000	25.000	38.000	71.000	40.000	81.000
TP	3225.000	183.000	209.000	185.000	204.000	108.000	206.000	212.000	179.000	210.000	169.000
Accuracy	0.860	0.694	0.714	0.718	0.736	0.728	0.669	0.736	0.768	0.690	0.712
Precision	0.849	0.680	0.672	0.709	0.703	0.663	0.617	0.693	0.799	0.646	0.728
Recall	0.871	0.732	0.836	0.740	0.816	0.931	0.892	0.848	0.716	0.840	0.676
Specificity	0.849	0.656	0.592	0.696	0.656	0.526	0.446	0.624	0.820	0.540	0.748
FPR	0.151	0.344	0.408	0.304	0.344	0.474	0.554	0.376	0.180	0.460	0.252
F1	0.860	0.705	0.745	0.724	0.755	0.774	0.729	0.763	0.755	0.730	0.701
Balanced_Accuracy	0.849	0.668	0.632	0.702	0.680	0.594	0.532	0.658	0.810	0.593	0.738

As we can see from the above two tabels, the accuracy on the original test split is comparable with the best accuracy on the validation set druing the grid search of hyperparameters.

Overall, for both logistic and svm, we have found that the accuracy on the extra testing sets are relatively lower compared with the accuracy on the original testing set. This is possibly becuase that we extra testing sets are extracted from the those movies of year after 2013. The original LMRD dataset includes instance of movies of year before 2011. Thus, maybe the way people express their ideas about the movie has changed over the years. Moreover, we notice that the performance of logistic regression and SVM are very similar

Besides, we do notice some group difference, for example, in both logistic regression and SVM, for the movies of runtime less than 100 minutes, the **recall** is 0.1 smaller than the movies of runtimes larger than 100 minutes. Also, in the Biography movies, the **accuracy** is relatively lower than all the other movie genres. This may due to the fact that the biography movie genres has fewer movies in the training set in general, so the models are not well trained to predict reviews of this movie genre. (Thesame problem seems to happen to animation genre as well.

4.4.3 Results with NGRAM Vectorization

The codes and results detail in this part can be found in python notebook NGram at our Github project repository.

We use five-gram of characters and select top 1000 features which appear at least in five reviews. Some most frequent grams are as following.

	Word	Count	Frequency
0	acti	14047	0.002143
1	acto	9388	0.001433
2	actu	7060	0.001077
3	almo	4356	0.000665
4	alon	4133	0.000631

After transforming the reviews using ngram vectorization, we fit a logistic regression model, the best model we have selected from the grid search has the following hyperparameters: {'C': 0.1, 'dual': False, 'penalty': 'l2'}.

With the best trained logistic regression model, we have the following performance on our testing sets.

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	3149.000	166.000	152.000	176.000	160.000	66.000	113.000	156.000	199.000	137.000	189.000
FP	647.000	84.000	98.000	74.000	90.000	50.000	118.000	94.000	51.000	113.000	61.000
FN	536.000	70.000	52.000	67.000	57.000	15.000	38.000	51.000	75.000	42.000	79.000
TP	3168.000	180.000	198.000	183.000	193.000	101.000	193.000	199.000	175.000	208.000	171.000
Accuracy	0.842	0.692	0.700	0.718	0.706	0.720	0.662	0.710	0.748	0.690	0.720
Precision	0.830	0.682	0.669	0.712	0.682	0.669	0.621	0.679	0.774	0.648	0.737
Recall	0.855	0.720	0.792	0.732	0.772	0.871	0.835	0.796	0.700	0.832	0.684
Specificity	0.830	0.664	0.608	0.704	0.640	0.569	0.489	0.624	0.796	0.548	0.756
FPR	0.170	0.336	0.392	0.296	0.360	0.431	0.511	0.376	0.204	0.452	0.244
F1	0.842	0.700	0.725	0.722	0.724	0.757	0.712	0.733	0.735	0.729	0.710
Balanced_Accuracy	0.830	0.673	0.639	0.708	0.661	0.619	0.555	0.652	0.785	0.598	0.746

Then, we fit a support vector machine (SVM) model, the best model we have selected from the grid search has the following hyperparameters: {'C': 0.2, 'class_weight': {1: 4}, 'loss': 'squared_hinge'}.

With the best trained SVM model, we have the following performance on our testing sets.

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	2357.000	126.000	111.000	125.000	122.000	38.000	64.000	113.000	154.000	89.000	145.000
FP	1439.000	124.000	139.000	125.000	128.000	78.000	167.000	137.000	96.000	161.000	105.000
FN	164.000	41.000	25.000	35.000	28.000	5.000	14.000	21.000	37.000	19.000	51.000
TP	3540.000	209.000	225.000	215.000	222.000	111.000	217.000	229.000	213.000	231.000	199.000
Accuracy	0.786	0.670	0.672	0.680	0.688	0.642	0.608	0.684	0.734	0.640	0.688
Precision	0.711	0.628	0.618	0.632	0.634	0.587	0.565	0.626	0.689	0.589	0.655
Recall	0.956	0.836	0.900	0.860	0.888	0.957	0.939	0.916	0.852	0.924	0.796
Specificity	0.621	0.504	0.444	0.500	0.488	0.328	0.277	0.452	0.616	0.356	0.580
FPR	0.379	0.496	0.556	0.500	0.512	0.672	0.723	0.548	0.384	0.644	0.420
F1	0.815	0.717	0.733	0.729	0.740	0.728	0.705	0.744	0.762	0.719	0.719
Balanced_Accuracy	0.666	0.566	0.531	0.566	0.561	0.458	0.421	0.539	0.652	0.472	0.617

From above tables we can see that the test split accuracy of both models outperform the baseline. The test split accuracy of logistic regression is higher than SVM, but the accuracies on our extra test datasets with different genres and durations are lower.

The models performance on biography movies are relatively worse compared to other genres, and that on horror movies are better.

4.4.4 Results with Word2Vec Vectorization

The codes and results detail in this part can be found in python notebook word2vec1 Vectorization and word2vec2 Vectorization at our Github project repository.

First, With Word2Vec vectorization and logistic regression model, the best model we selected from grid search has the hyperparameters: {'C': 40, 'class_weight': 'balanced', 'dual': False, 'penalty': 'l1', 'solver': 'saga'}.

With the best trained logistic regression model, we have the following performance on the nine testing sets:

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	3222.000	162.000	157.000	175.000	165.000	55.000	100.000	155.000	204.000	134.000	190.000
FP	574.000	88.000	93.000	75.000	85.000	61.000	131.000	95.000	46.000	116.000	60.000
FN	534.000	71.000	38.000	64.000	51.000	12.000	36.000	48.000	73.000	40.000	84.000
TP	3170.000	179.000	212.000	186.000	199.000	104.000	195.000	202.000	177.000	210.000	166.000
Accuracy	0.852	0.682	0.738	0.722	0.728	0.685	0.639	0.714	0.762	0.688	0.712
Precision	0.847	0.670	0.695	0.713	0.701	0.630	0.598	0.680	0.794	0.644	0.735
Recall	0.856	0.716	0.848	0.744	0.796	0.897	0.844	0.808	0.708	0.840	0.664
Specificity	0.849	0.648	0.628	0.700	0.660	0.474	0.433	0.620	0.816	0.536	0.760
FPR	0.151	0.352	0.372	0.300	0.340	0.526	0.567	0.380	0.184	0.464	0.240
F1	0.851	0.692	0.764	0.728	0.745	0.740	0.700	0.738	0.749	0.729	0.698
Balanced_Accuracy	0.852	0.682	0.738	0.722	0.728	0.686	0.638	0.714	0.762	0.688	0.712

As we can see from the above table, Test_Split has highest accuracy among all other test sets. This might be because Test_Split and traing set are from the same dataset which extrated before year 2011 from IMDB website, but all other test sets are crawled from movie reviews after 2013 in IMDB website. People's language style and usage may change with time, so if the training set and test sets are not in the same time period, the accuracy would be lower. In addition to Test_Split, the accuracy of Horror is highest and Biograpy is lowest. This might be because the keywords representing positive or negative reviews between these two genres of movies are kind of different, even opposite. In addition, the animation test set only contains 232 examples, which

might lead to a relatively low accuracy. Moreover, the FPR for animation and biography test sets are relatively high. Overall, the results varied across movie genres, possibly because words identified as positive by one movie might be identified as negative by another one.

Second, with Word2Vec vectorization and support vector machine (SVM) model, we have the following optimal hyperparameters during the grid search {'C': 20, 'class_weight': 'balanced', 'loss': 'squared_hinge', 'penalty': 'l2'}.

With the best trained SVM model, we have the following performance on the nine testing sets:

	Test_Split	Runtime_1_100	Runtime_101_600	Action	Adventure	Animation	Biography	Comedy	Horror	Romance	Sci_fi
TN	3298.000	166.000	162.000	180.000	173.000	61.000	109.000	158.000	207.000	140.000	197.000
FP	498.000	84.000	88.000	70.000	77.000	55.000	122.000	92.000	43.000	110.000	53.000
FN	583.000	80.000	47.000	73.000	58.000	13.000	40.000	46.000	86.000	43.000	93.000
TP	3121.000	170.000	203.000	177.000	192.000	103.000	191.000	204.000	164.000	207.000	157.000
Accuracy	0.856	0.672	0.730	0.714	0.730	0.707	0.649	0.724	0.742	0.694	0.708
Precision	0.862	0.669	0.698	0.717	0.714	0.652	0.610	0.689	0.792	0.653	0.748
Recall	0.843	0.680	0.812	0.708	0.768	0.888	0.827	0.816	0.656	0.828	0.628
Specificity	0.869	0.664	0.648	0.720	0.692	0.526	0.472	0.632	0.828	0.560	0.788
FPR	0.131	0.336	0.352	0.280	0.308	0.474	0.528	0.368	0.172	0.440	0.212
F1	0.852	0.674	0.751	0.712	0.740	0.752	0.702	0.747	0.718	0.730	0.683
Balanced_Accuracy	0.856	0.672	0.730	0.714	0.730	0.707	0.650	0.724	0.742	0.694	0.708

As we can see from the above tabel, the accuracy of test dataset is good, especially compared with the performance of extra test dataset. And we can found the accuracy for Horror films is better than movies with other genres. This might be because reviews including words like "frightening", "scared" are easily regarded as good reviews of horror films, but other films contain those words are often negative reviews. And this is why the precision of horror films is high, but the recall rate is low.

On the other hand, we found that the FPR of biography is pretty high, this might be because it hard to find some words to express obvious positive or negative feelings for a biography film. In addition, the number of animation reviews is small, which may affect the performance of prediction.

In a word, there is obvious difference of prediction performance among movie reviews with different genres and runtimes. On the one hand, sentiment analysis seems not efficient to some type of movie like biography. On the other hand, some words are positive for this genre but negative in another type, which is we need to pay attention while doing the sentiment analysis.

4.4.5 Results with Neural Networks

We found prediction with neural networks were not good (about 0.5) for all four vectorization (please see the python notebooks Python Notebooks for Each Vectorzation, which means there was something wrong, so we ran the code from https://github.com/ovguyo/moviereview as follows and tried to find why this problem happens.

[ ]:
```
import numpy as np
import pandas as pd
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras.layers import Conv1D, Flatten, Dropout, MaxPooling1D
from keras.layers import Embedding
from keras.preprocessing import sequence
```

[ ]:
```
np.random.seed(7)
top_words = 5000
(X_train1, y_train1), (X_test1, y_test1) = imdb.load_data(num_words=top_words)
max_length = 500
X_train1 = sequence.pad_sequences(X_train1, maxlen = max_length)
X_test1 = sequence.pad_sequences(X_test1, maxlen = max_length)
embedding_vector_length=32
model1 = Sequential()
model1.add(Embedding(top_words, embedding_vector_length,␣
 ↪input_length=max_length))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
```

[ ]:
```
model1.add(Conv1D(32, kernel_size= 3, padding= 'same', input_shape=(max_length,␣
 ↪embedding_vector_length)))
model1.add(Conv1D(32, kernel_size= 3, padding= 'same'))
model1.add(MaxPooling1D(pool_size=2))
model1.add(Conv1D(16, kernel_size= 3, padding= 'same'))
model1.add(MaxPooling1D(pool_size=2))
model1.add(Conv1D(16, kernel_size= 3, padding= 'same'))
model1.add(MaxPooling1D(pool_size=2))
model1.add(Flatten())
model1.add(Dropout(0.2))
model1.add(Dense(64, activation='sigmoid'))
model1.add(Dropout(0.5))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
model1.fit(X_train1, y_train1, epochs=3, batch_size=32)
```

```
Epoch 1/3
782/782 [==============================] - 34s 43ms/step - loss: 0.4719 -
accuracy: 0.7382
Epoch 2/3
782/782 [==============================] - 34s 43ms/step - loss: 0.2651 -
accuracy: 0.8965
Epoch 3/3
782/782 [==============================] - 34s 44ms/step - loss: 0.2246 -
accuracy: 0.9125
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8cdae8f7b8>
```

For Word2Vec vectorization, I found that the difference was the Embedding layer, the above code gave corresponding index for each word in the review sentence, and then used "sequence.pad_sequence" to make sure each transformed sentence (from texture to numerical) have the same length. Then the result of Embedding layer is a len(sample)× max_length×embedding_vector_length matrix, where the embedding_vector_length means the Embedding layer mapped each word into a embedding_vector_length-dimension space and got its feature vetor.

However, what the Word2Vec method did was: first of all, this method counted the word frequency of all words in the trainning set, and assigned indexes for each word according to the word frequency from high to low. Secondly, for each review in the testing data set, feature vectors (gotten from the training word2vec model) are first assigned to each word that appeared in the previous trainning set, then the average value of the word vectors of this review was calculated, and finally the len(sample)×num_features matrix was obtained, which means we lack the dimension corresponding to "max_length" in the Embedding layer example.

In order to make the output dimension from Word2Vec model match that from the Embedding layer, I tried to change the makeFeatureVec function in Word2Vec part, however, since the num_features is 400 and the training dataset is large, the colab always crashed once I was trying to use Word2Vec model to transform texture data. Therefore, I put the result of Embedding layer method to show the code works, and due to the time limitation of project and memory restrication in Colab, I cannot find a good method to solve this problem.

Also, after we do the vectorization, the CNN layer may not be very meaningful. for example, after we perform the TFIDF/Ngram vectorization of the top 1500 words, the idea "neighborhood" is not interpretable. The 1st number in a document vector has just as much relation to the 2nd and to the 1500th.

4.4.6 Time Complexity Comparison

Time (in terms of seconds)	Grid Search Time	Neural Network Training Time	Predict Nine Testing Sets
Baseline (Bag of Word + LR)	47	/	/
TFIDF+LR	92	/	0.023
TFIDF+SVM	15	/	0.027
TFIDF+CNN	/	727	
TFIDF+LSTM	/	153	
NGram+LR	27	/	0.025
NGram+SVM	145	/	0.028
NGram+CNN	/	1323	/
NGram+LSTM	/	3650	/
Word2Vec+LR	251	/	0.024
Word2Vec+SVM	618	/	0.032
Word2Vec+CNN	/	/	/
Word2Vec+LSTM	/	828	/

For four different vectorizations, we found that the time-complexity is ranked from high to low as Word2vec> TFIDF> NGram ~ Bag-of Word.

For the different models, we found that the time-complexity is ranked from high to low as LSTM > CNN > Logistic Regression ~SVM

5 Discussion

5.1 What you've learned-Shihong Wei

Note: you don't have to answer all of these, and you can answer other questions if you'd like. We just want you to demonstrate what you've learned from the project.

What concepts from lecture/breakout were most relevant to your project? How so?

What aspects of your project did you find most surprising?

What lessons did you take from this project that you want to remember for the next ML project you work on? Do you think those lessons would transfer to other datasets and/or models? Why or why not?

What was the most helpful feedback you received during your presentation? Why?

If you had two more weeks to work on this project, what would you do next? Why?

Answer:

There are many concepts from lecture/breakout most relevant to the project: First of all, we have a supervised learning (binary classification) problem. Secondly, we follow the valid modeling procedure, for example, the classic Train-Valid-Test Split. Third, in the feature engineering part, we used some vectorizations of texture data (Word of Bags, Ngram, TFIDF) and applied dimension reduction techiniques such as PCA. Fourth, in terms of fitting methods, we used Logistic Regression, SVM, CNN, LSTM. Fifth, we explored model interpretation and model fairness evaluation with our trained models.

The aspects we have found to be most surprisingly is that:

First, we can use Machine larning algo to predict sentiments of movie reviews with quite high accuracy even people may have the diverse ways in expressing their opinions.

Second, even relatively simple model such as Logistic regression can do well than complex methods such as NN under some circumstances.

With lessons did you take from this project, we hope we will know how to model the binary classicification of texture data from raw texts to model evaluation in the future. Also, we find that sometimes, we do not need to use very complicated model to gain good prediciton performance. Meanwhile, simpler model is more time-efficient to train, use and interpret.

During the presentation, the most helpful feedback is from Zach asking us to explore the reason why cirtain model does not perform well. That inspires us to investigate the story behind the training examples

If we have two more weeks for the project, we will try using web crawlers to extract a more recent training set from different websites, because the LMRD dataset is constructed at 2011, they way people express their opinions about movies and those popular words on the internet may have changed silightly over the years

6 reference

6.1 Papers

[1] Maas, A.L. & Daly, R.E. & Pham, P.T. & Huang,D. & Ng, A.Y & Potts,C (2011). Learning Word Vectors for Sentiment Analysis. *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).* Available at https://ai.stanford.edu/ amaas/data/sentiment/

[2] Mikolov, T. & Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[3] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

6.2 Software Packages (Main Python Libraries)

numpy, pandas, sklearn, gensim, keras, nltk, bs4, seaborn, matplotlib, textblob, wordcloud

For python webclawer algorithm on IMDB, please click on link Web Clawer