Movie Review Sentiment Analysis

Undergraduate AI Capstone: Project 1

Introduction

In this project, I collect the movie reviews on IMDb website, and predict the sentiment of the reviews.

Sentiment analysis is a popular field in natural language processing. I want to explore some famous machine learning strategies for this text classification task.

Full codes and datasets are on my github repository: FuliniF/movie-review-sentiment-analysis

Dataset

Web Scrapping Method

I scrapped the website: Most Popular Movies on IMDb, getting all the reviews and scores for each movie on the website. All the data is collected based on the data before 6th March.

The reason why I choose this website and “Most Popular Movies” specifically is because it is one of the largest movie review websites in the worlds. I want to collect data which may not be clean, but has enough information that contains **as many formats of sentiment expressions as we human can have**. In “Most Popular Movies”, there aren’t just movies that people love, but also movies people complain about. I believe it is a better choice to collect more data generally than collect it from “Top Ranked Movies” or other websites.

I use Beautiful Soup to scrap all the reviews. After opening the url with Chrome driver, It will automatically scroll to the bottom of the review area, and then scrap the data from html.

Dataset Introduction

There are 5 columns in this dataset: **movie**, **score**, **title**, **review**, **sentiment**.

The score is taken from 1 to 10. I want to convert it into 2 sentiments: positive or negative.

The rule I made for this convertion is:

> If the score ≥ 6, the review is marked as 1 (positive).
>
> If the score < 6, the review is marked as 0 (negative).

Here is some examples for this raw dataset:

```
movie: 蜘蛛夫人
score: 1.0
title: Dreadful
review: As bad as everyone is saying, what is more shocking is the 1200 (at t
sentiment: 0
```

```
movie: 教父
score: 9.0
title: Very awesome
review: Spectacular and nice movie. It is one of the best movies in all time.
sentiment: 1
```

Cleaning Dataset

There are many puncuations and meaningless words that may let the model hard to train. I use NLTK library to do the cleaning process.

I remove all the puncuations, stop words and numbers in the review and title column, and make every alphabet lower one. After cleaning, the dataset should look like this:

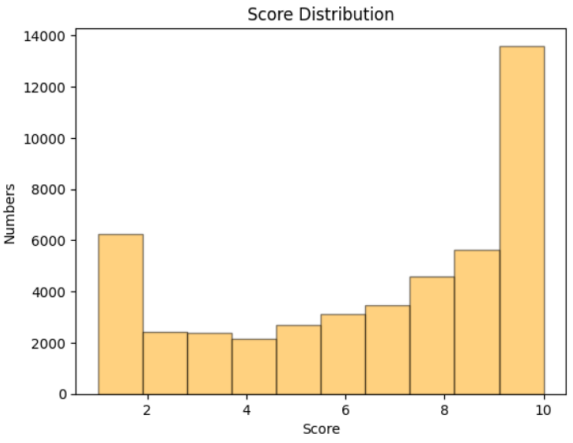
```
movie: 蜘蛛夫人
score: 1.0
title: dreadful
review: bad everyone saying shocking time rewiew star ratings // omitted
sentiment: 0

movie: 教父
score: 9.0
title: awesome
review: spectacular nice movie one best movies time watch
sentiment: 1
```

Dataset Size

After all the preprocessing process, there are 46,299 data in total. I divide them into 41,669 data for training and validating dataset, and 4,630 data for testing dataset. The number of sentiments for all dataset are as shown:

- Positive reviews: 27295
- Negative reviews: 14374
- Ratio (pos/neg) : **1.8989147071100598**



Methods, Algorithms and Result Analysis

All the implementations are in the Appendix.

Before introducing the methods, here are some important things for my experiment settings:

1. **Cross-validation (supervised):** I choose **5-fold cross-validation** for both of the supervised algorithms. The implement methods are a little bit different due to the different libraries I use.

After training for 5 models for each folding, the total performance on training dataset is evaluated by all the numbers in its confusion matrix, and calculate their metrics in average. To evaluate the performance on testing dataset, I choose the selected best model from previous training process, and use all the training samples to optimize its parameter, according to the method written on lecture slide page 21. Finally, I use the final classifier to predict on testing dataset.

2. **Metrics:** Since this is a binary classification task, I calculate all the following measures for all the algorithms:
   a. **Accuracy** = How many predictions are correct?

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

   b. **Precision** = Among all positive predictions, how precise are they to be predicted? (how sensitive for positive instance)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

   c. **Recall** = Among all positive ground truths, how many of them are predicted as positive correctly? (coverage of TP in all positive instance)

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

   d. **F1** = Consider both precision and recall in equal weights.

$$\text{F}_1 = \frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}$$

   e. **ROC-AUC** = ROC defines the False Positive Rate (FPR) as x-axis, and the True Positive Rate (TPR) as y-axis. If AUC equals to 50%, this means that it is same as randomly guessing, with only 50% of probability to be correctly predicted. The higher this score is, the better the model can perfectly classify the inputs.

      Since this is a binary classification task, we can actually calculate the value simply by the following formula, or use the function in **scikit-learn** library:

$$\frac{\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{TN}+\text{FP}}}{2}$$

I calculate these metrics twice for every model: one for the **training and validating** dataset, one for **testing** dataset.

1. Naive Bayes Classifier

Description

For the first supervised learning approach, I choose one of the most common and effective way to do this. To implement this method, I use **NLTK** library for data preprocessing and the base model.

First, I get a frequency distribution list from all the words in the training dataset, and leave only the 10000 most frequently used words in the dataset. This is because there might be some words that are not commonly used by many people, which means these words may not be good features for classifying.

After preprocessing the dataset, I define a Naive Bayes classifier in NLTK library, and train it with self-defined 5-fold cross validation process.

Results

For all training and validation dataset:

	True: Pos	True: Neg
Pred: Pos	22706	1913
Pred: Neg	4586	12460

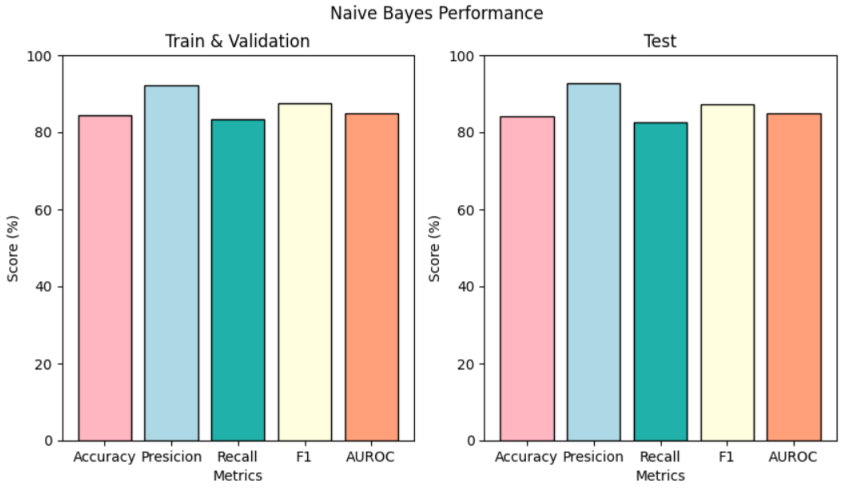
```
Accuracy:  0.8440177607104284
Precision:  0.922295787806166
Recall:  0.8319654111094826
F1:  0.8748049546338926
AUC:  0.8494343162136155
```

For testing dataset:

	True: Pos	True: Neg
Pred: Pos	2528	199
Pred: Neg	539	1364

```
Accuracy:  0.8406047516198704
Precision:  0.927026035936927
Recall:  0.8242582328007825
F1:  0.8726268553676217
AUROC:  0.8484694874816452
```

Analysis



In the results, we can see that this model has a high precision score with over 92% for both training and testing process. This means that it has a higher ability to make less mistakes on classifying positive inputs. There isn't a big gap for the metric scores between training and testing, means that this model doesn't suffer from overfitting.

2. Support Vector Machine

Description

Support vector machine (SVM) is a method to map the data from low dimension to higher dimension using different kernel functions. After mapping the data points, we want to find a

hyperplane such that all the data points can be divided into 2 different areas.

I use **scikit-learn** library to implement this method, and choose linear kernel to train this SVM model.

Before training, I transform the input text into vectors with TF-IDF vectorizer in **scikit-learn** library. TF means “term frequency”, which calculates the frequencies of all the words. IDF means “inverse document frequency”. To calculate the IDF of a word, we need to calculate the frequency for this word in all document. After that, we use the formula for IDF: (D=number of all documents, dt=number of documents that word t appears in)

$$idf_t = \log \frac{D}{d_t}$$

After calculating TF and IDF, we simply multiply them to get the final value of a TF-IDF vector.

Results

For all training and validation dataset:

	True: Pos	True: Neg
Pred: Pos	25009	3198
Pred: Neg	2286	11176

```
Accuracy:  0.8683913700832754
Precision:  0.8866238876874535
Recall:  0.9162483971423337
F1:  0.9011927498108177
AUC:  0.8468816773522995
```

For testing dataset:

	True: Pos	True: Neg
Pred: Pos	2799	314
Pred: Neg	268	1249

```
Accuracy:  0.8742980561555076
Precision:  0.8991326694506907
Recall:  0.9126181936746006
F1:  0.9058252427184468
AUC:  0.8558612401514398
```

Analysis

In the figure shown above, we can see that SVM has an average performance on every metrics we evaluate. the Although the dataset is not perfectly balanced, we don’t see a huge effect on this model. This model has a higher score in Recall than Naive Bayes. This means that when the ground truth is positive, it is more possible to be predicted as positive than Naive Bayes classifier.

3. Lbl2Vec

Description

Lbl2Vec is an algorithm for unsupervised document classification tasks. In this method, some keywords related to certain fields should be defined at first, since this model will calculate the distance between the words in a document and the keywords. This can be used for category classifications such news topic classification. I want to try this method on sentiment classification. I want to check whether it can work on binary classification task as well or not.

First, I define a keyword list for positive and negative. I add these keywords:

- Positive: nice, masterpiece, beautiful, excellent, awesome, amazing, great, fantastic
- Negative: horrible, bad, disgusting, terrible, awful, boring, overrated, waste

We also need to tokenize the words in documents. I use the functions in **gensim** library to do the preprocessing. I make the words as tags so that the model can calculate the distance among tags and keywords.

After training, the model will return similarity scores on different labels, which are “positive” and “negative” in this case. I take the label of highest similarity score as its final prediction. I trained the model for 10 epochs.

Results

For all training and validation dataset:

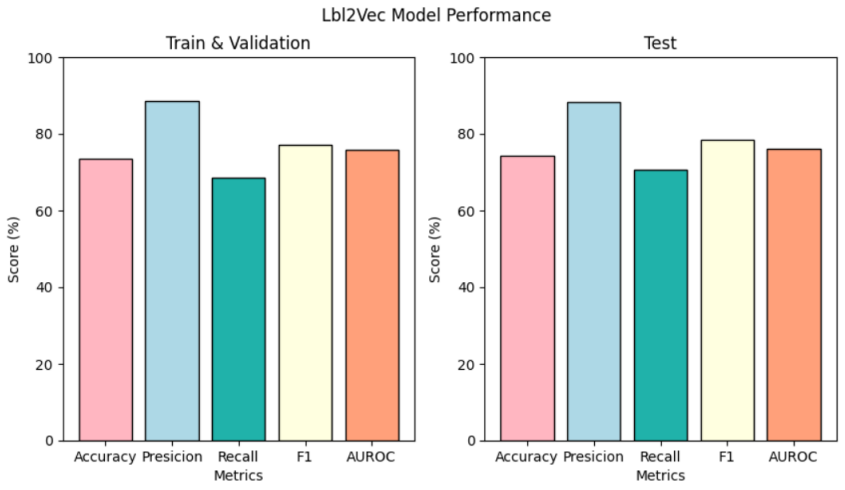
	True: Pos	True: Neg
Pred: Pos	18670	2436
Pred: Neg	8625	11938

```
Accuracy:  0.7345508651515515
Precision:  0.8845825831517105
Recall:  0.6840080600842645
F1:  0.7714716638085989
AUC:  0.7572677005583421
```

For testing dataset:

	True: Pos	True: Neg
Pred: Pos	2164	290
Pred: Neg	903	1273

```
Accuracy:  0.7423326133909287
Precision:  0.8818255908720456
Recall:  0.7055754809259863
F1:  0.7839159572541206
AUC:  0.7600174269633131
```



Analysis

The results for this model is not as good as the above models. From the confusion matrix, We can see that the number of False Negative is relatively high, lead to a low score on Recall. Overall, this model doesn't achieve a good performance, with the lowest scores for all the mretrics I calculate on all the models.

In my opinion, one of the main reason may be the key idea of this algorithm doesn't fully match my task. This is a model that can be widely used in category classification tasks, such as classifying news related to "basketball" or "soccer". Maybe the keyword list I defined myself doesn't well represent the sentiments which is relatively abstract.

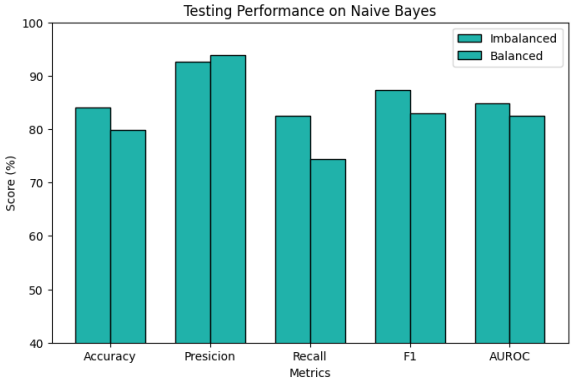
Experiments

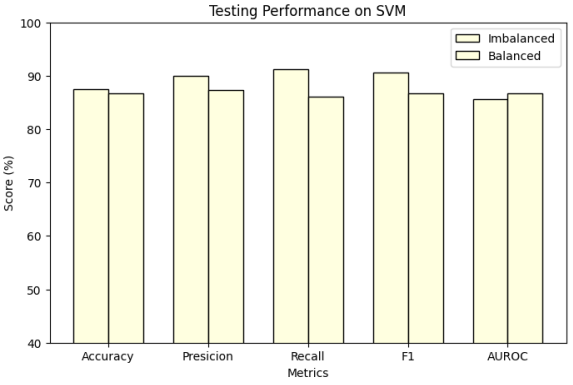
I have done two additional experiments on this task. One is to make the number of positive and negative reviews more balanced, and the other is to use different columns in the dataset.

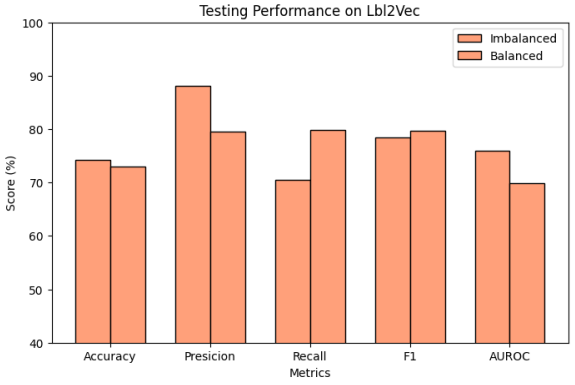
1. Data Balancing

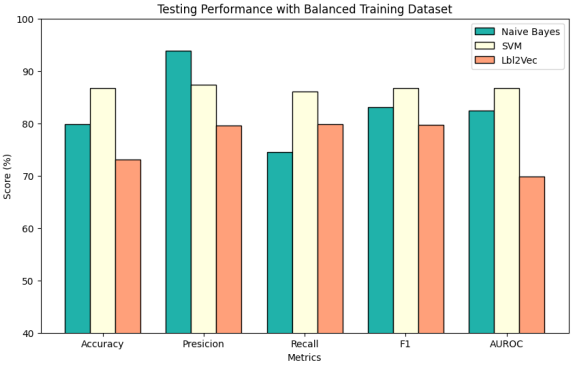
According to the documentation for dataset, we can find that the number of positive reviews is higher than that of negative ones, especially for those with score 10 out of 10. The imbalanced data may lead to worse performance since the models don't have as much negative reviews as they do on positive ones.

In this experiment, I take the same amount of positive reviews as negative ones. There are 27,295 positive reviews and 14,374 negative reviews in my dataset. I only take the first 14,374 positive reviews with all negative reviews to do this experiment. The results are shown as below.









It surprised me that the results for Naive Bayes and SVM did not improve on most of the metrics. I think this is because balancing the dataset makes the total training amount less than original training settings. However, it becomes more balanced between Precision and Recall scores for Lbl2Vec model. Although this experiment doesn't have a great affect on the accuracy, it makes the unsupervised model more capable to cover all the true positives among all positive predictions.

In the figure with all models' testing performance on balanced dataset, we can see that SVM has a relatively balanced performance among all metrics. Whereas this experiment doesn't lead these models to have a better performance.

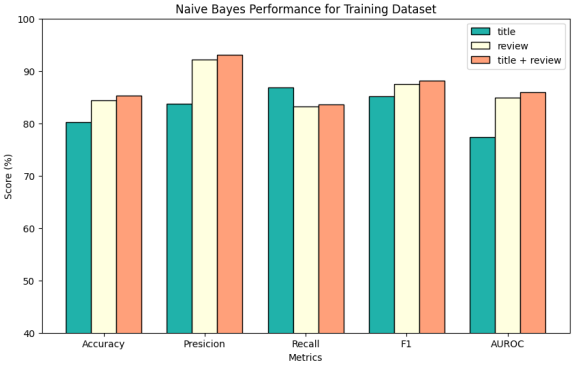
2. Training on Different Texts

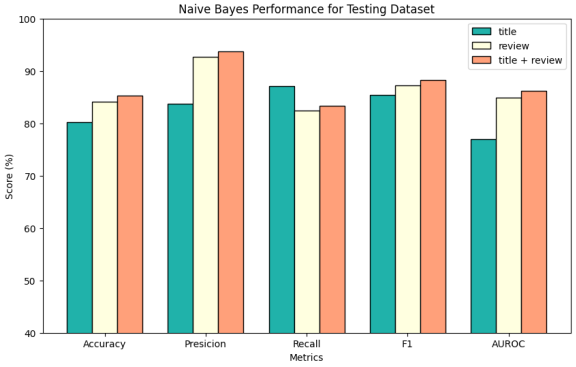
When scrapping the dataset, I found that the "title" for each data seems to represent the scoring better than those longer review documents. Take the examples on previous dataset introduction, we can quickly understand that this reviewer doesn't like this movie by the title "dreadful" instead of the review itself.

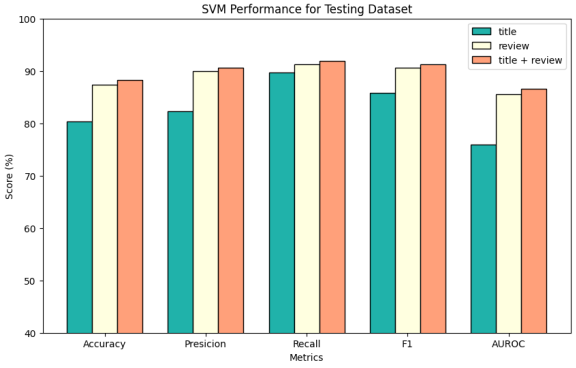
```
movie: 蜘蛛夫人
score: 1.0
title: dreadful
review: bad everyone saying shocking time rewiew star ratings // omitted
sentiment: 0
```

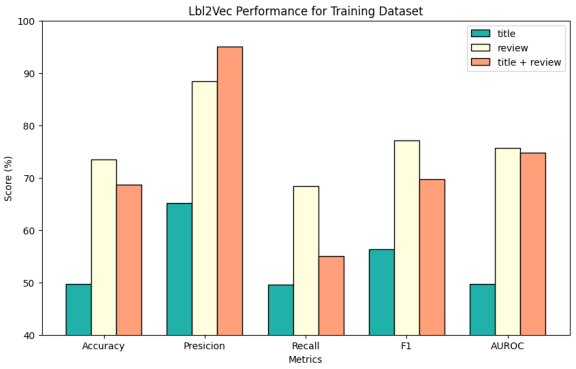
However, the models may not think the same as us human do. Therefore, I want to try to train the classifier on each title of the dataset, and see whether the performance become better or not.

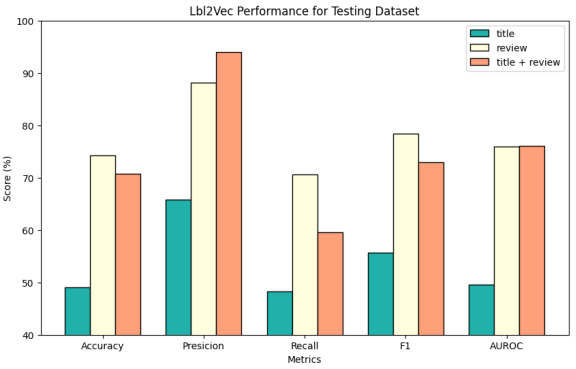
In this experiment, I am going to train these models for three training texts: only title, only review, and on both title and review columns at once. The results are shown as below:











Overall, the models trained only on “title” column perform worse than any other model, Especially for the Lbl2Vec model, which has only roughly 50% of accuracy. The ROC-AUC score of Lbl2Vec model shows that the model didn’t trained well, as it only achieve the same accuracy as a random classifier. We can see that for naive Bayes and SVM models, using both title and review text has slightly better performance than using only review column as training text. It is interesting to note that for Lbl2Vec model, although precision score for training on both columns is higher than only review column, other scores are lower than only review column.

There are some possible reasons leading to this results. First, maybe the epochs are not enough to optimize the performance for the models. Additionally, title column has much less words, therefore contains fewer information than reviews have. It is harder for the models to learn enough features to classify the sentiments correctly.

Discussion

1. **Based on your experiments, are the results and observed behaviors what you expect?**

   I didn’t expect that the unsupervised model would get much worse performance than the other two supervised models. I think this may because of the different idea of supervised and unsupervised learning.

   Besides, SVM got higher scores than I expected. I found that however there are many powerful models nowadays, this is actually still a very useful method for classification tasks.

2. **Discuss factors that affect the performance, including dataset characteristics.**

   The data collected has too many reviews with 10/10 scoring. This may lead to biases when training models. However, during the additional experiment I did for this problem, I found that the models didn’t get a better performance. I guess it’s because I removed almost 1/3 training dataset to make the two sentiments have balanced amount.

   Aside from the number of dataset, the dataset preprocessing is done by myself, with capital transformation, puncuation and number removal. There should definitely be other better ways to preprocess these data.

   The training setting is different for supervised model and unsupervised model as well. I didn’t do cross validation on Lbl2Vec (unsupervised) model, and I trained it for 10 epochs, which is different from my supervised models.

3. **Describe experiments that you would do if there were more time available.**

There are still many models that I want to try out very much. For example, I did try using LSTM model to train it. However, the model isn't trained well that it has a bad performance. Because of that, I didn't put this model on my project. I will try to make this model at least get an average performance if I have more time.

Beside the LSTM model, I also want to train this dataset on transformer models. Transformer models use attention masks and other encoder-decoder strategies to make it more capable on NLP tasks. I want to see the performance it can achieve on such models.

4. **Indicate what you have learned from the experiments and remaining questions.**

   I have learned that it is really important to plan all the experiments you want to do before start doing it, and record all the results. Git and github are very powerful for version control although there is only me to do this project. I've also encountered some git problems when doing this project. This is a good chance for me to learn how to solve problems when using git.

   One of the remaining questions is why my unsupervised model didn't get a good performance. I want to try out some other unsupervised models as well.

References

Most Popular Movies (imdb.com)

Naive Bayes Classifier with NLTK

[2210.06023] Lbl2Vec: An Embedding-Based Approach for Unsupervised Document Retrieval on Predefined Topics (arxiv.org)

Lbl2Vec — Lbl2Vec 1.0.2 documentation

3.1. Cross-validation: evaluating estimator performance — scikit-learn 1.4.1 documentation