DS595 NLP Assignment 2

Name: Xinyi Fang

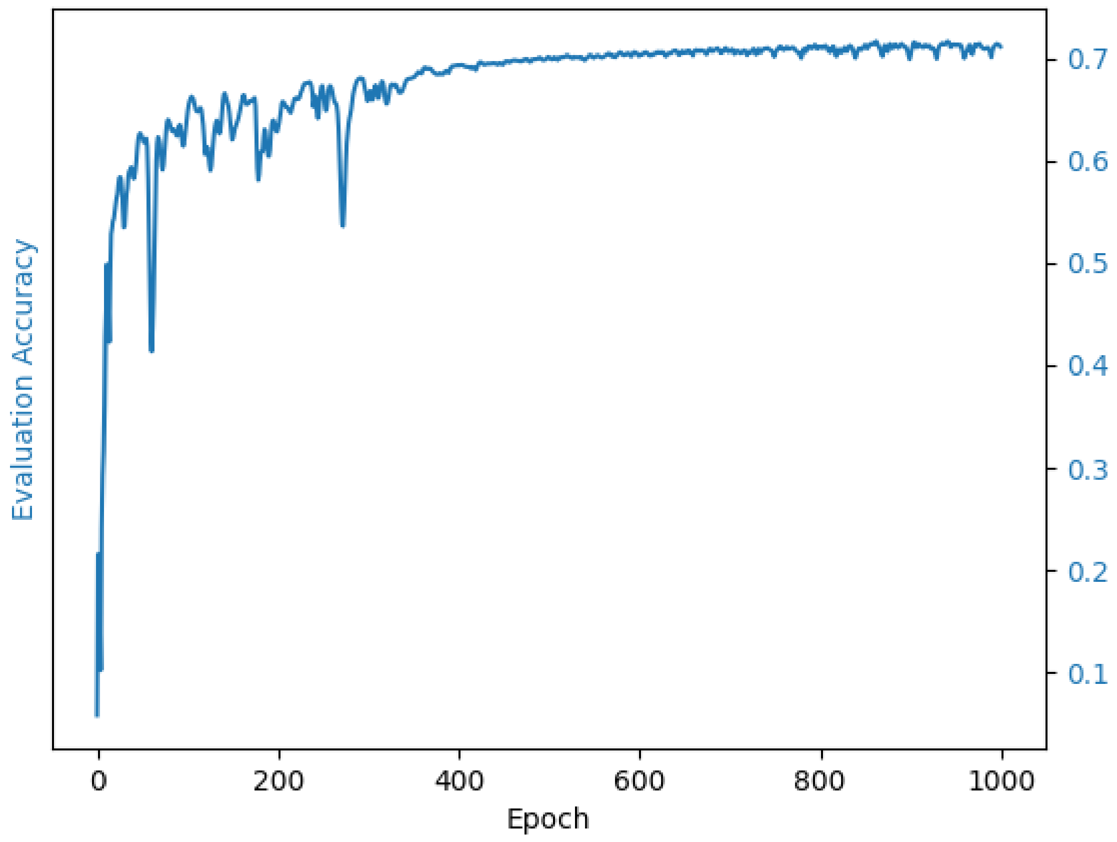
Email: xfang1@wpi.edu

> To make it easier to submit the assignment, I put all the code into a submission.py file. But in the actual work process, each task is run separately. When handing in the assignment, I also put all 8 separate files into CodeAssignment2 - Google Drive for easy viewing.

Task 0. Data Preprocessing and Exploratory Analysis

> for detailed code of this task please see *initialization.py* and *preprocess.py*

Since we only use ratings and text reviews in the dataset, let's see the score distribution first.



We define score labels:

score>3: positive, score<3: negative.

As we can see most of the scores of reviews are positive. That leads to an imbalanced dataset. The label distribution is:

	label distribution
1	0.843981
0	0.156019

In order to resample a new balanced dataset, we ensure the number of positive reviews and negative reviews are equal. The new sample size is 20% of the total size, half of them are positive and the other half are negative. And they account for approximately 18.5% of the original data.

We save this new dataset as *balanced_reviews_sample.csv*. And then we preprocess this dataset further. These four steps are normally helpful:

1. remove HTML tags
2. convert words to lowercase
3. remove stop words
4. remove punctuation and special characters

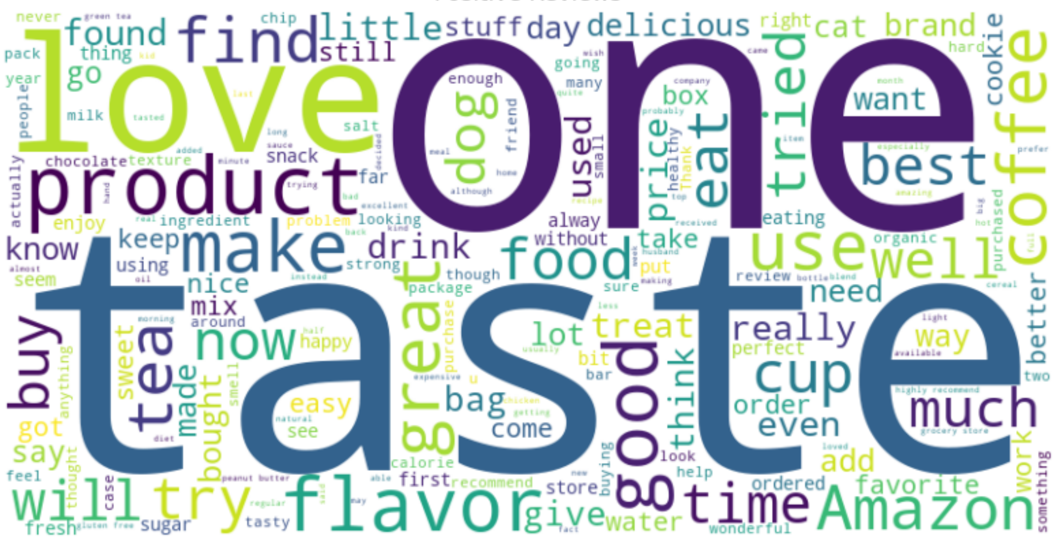
But after the test, we only use the first step. Take TF-IDF-based logistic regression as an example, we test the dataset with different preprocessing steps:

Steps Usage	Accuracy	Precision	Recall	F1
none	0.902	0.909	0.896	0.902
1(2)	0.902	0.909	0.897	0.903
123	0.897	0.902	0.893	0.898
124	0.9019	0.910	0.894	0.9019

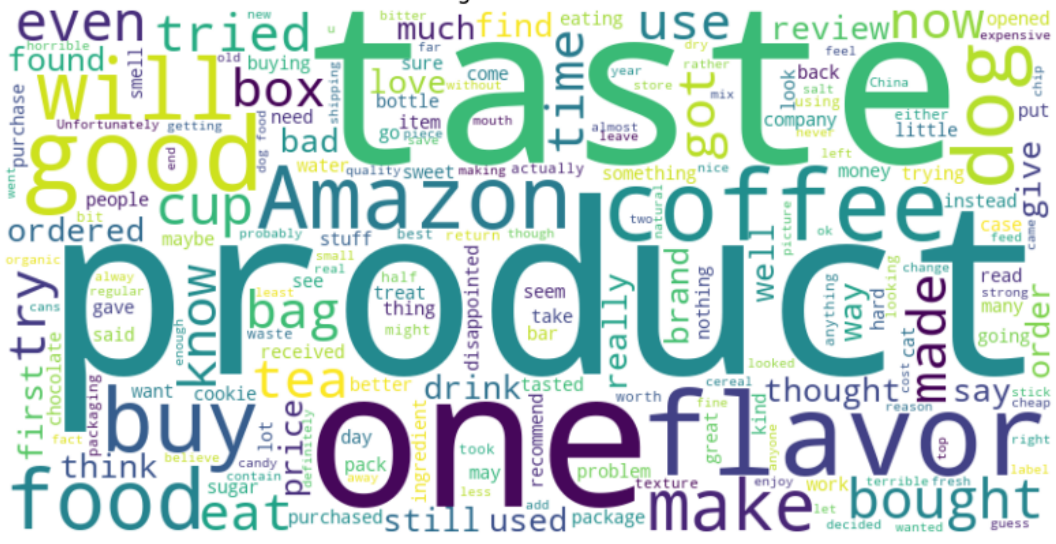
Applying the last two steps would make model performance worse.

Save this dataset as *preprocess_simple.csv* and let's see how positive reviews and negative reviews like through word clouds. We will use these two datasets to proceed with our following tasks.

Positive Reviews



Negative Reviews



We can see that for positive comments there are more positive words like "love". As for negative reviews, there are also many positive words like "good", but there aren't a lot of negative words.

Task 1. TF-IDF approach

> for detailed code of this task please see *tfidf_nn.py* and *tfidf_lr.py*

We use one of the efficient methods we used in Assignment 1 and add another method, a simple neural network model for comparison. The TF-IDF based results:

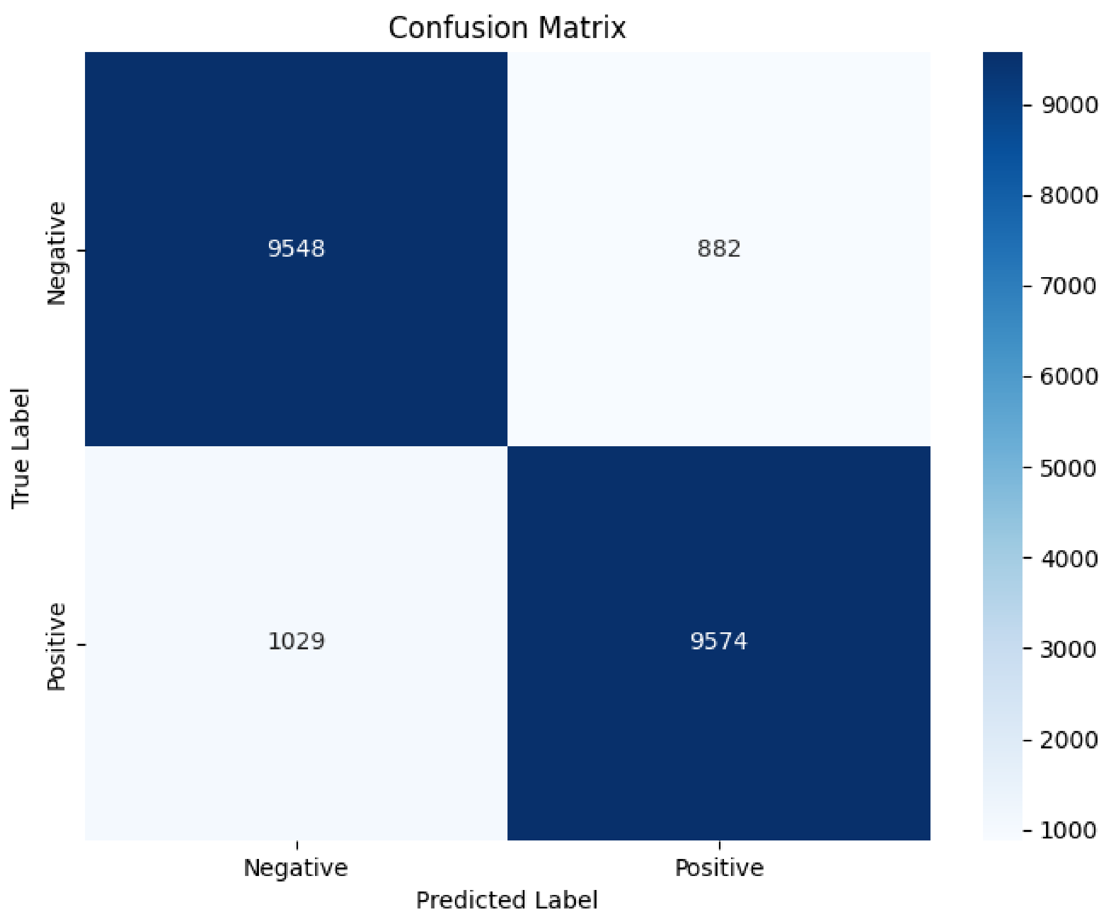
	Accuracy	Precision	Recall	F1
logistic regression	0.902	0.909	0.896	0.902
neural network	0.911	0.911	0.911	0.911

The preprocessed version:

	Accuracy	Precision	Recall	F1
logistic regression	0.902	0.909	0.897	0.903
neural network	0.909	0.916	0.903	0.909

Apparently, the neural network model is better, we will use this for the upcoming tasks.

The confusion matrix of the neural network model:



Task 2. Review classification by using word2vec

> for detailed code of this task please see *word2vec_nn.py*

We will use the neural network model in this task, but not the same simple model as we used in task 1. We need a more complex neural network to better handle the high-dimensional features of Word2Vec.

Here we make three improvements and see the results.

1. Increase the number of layers, more neurons are used, and Batch Normalization and Dropout are added after each fully connected layer. The activation function was replaced by LeakyReLU, and the output layer used LogSoftmax.

2. Build a deeper network and implement residual connections, where inputs are added to the outputs.

3. Introduce a learning rate scheduler to adjust the learning rate according to the performance of the verification set during the training process.

   These changes are intended to improve the model's learning and generalization capabilities. And after we had done this, the accuracy improved to 0.875 from 0.834. The accuracy of the final version with preprocessed data is 0.883. Details are as follows:

	Accuracy	**Precision**	**Recall**	**F1**
origin nn	0.834	0.828	0.846	0.831
1st improve	0.858	0.864	0.851	0.861
2nd improve	0.870	0.874	0.868	0.872
3rd improve	0.872	0.872	0.875	0.872
preprocess version	0.883	0.890	0.875	0.886

Task 3. BERT (without fine-tune) for review classification

> for detailed code of this task please see *BERT_without.py*

Pre-trained language models such as BERT can capture rich language features, including grammar, context, relationships between words, etc., by learning on large-scale text data. This universal language representation provides a powerful starting point for a variety of downstream tasks.

We have the results:

Accuracy: 0.8572868526654115 Precision: 0.8810726383902311 Recall: 0.826077860824252 F1 Score: 0.852689438555163

Tasks 4. BERT (with fine-tune) for review classification

> for detailed code of this task please see *BERT_with.py*

Fine-tuning allows BERT models to be pretrained on a large corpus to learn broad properties of language and then adapted to a specific task by fine-tuning on a smaller dataset for that specific task.

We have the results:

Accuracy: 0.9609204145668917 Precision: 0.963855421686747 Recall: 0.9566055930568949 F1 Score: 0.9602168231536153

Task 5. BERT (with LoRA) for review classification

> for detailed code of this task please see *BERT_withLora.py*

LoRA (Low-Rank Adaptation) adjusts model parameters by introducing a low-rank matrix on the weight matrix of the model, aiming to reduce the number of parameters that need to be trained. Thereby we could reduce computational costs and avoid overfitting while maintaining or improving model performance.

This part is amazing, evaluation indicators have been fluctuating, but the best case reached 1.

Accuracy: 1.0 Precision: 1.0 Recall: 1.0 F1 Score: 1.0

Task 6. Results Analysis

Method	Precision	Recall	Accuracy	F1
TFIDF-LR	0.902	0.909	0.896	0.902
TFIDF-NN	0.911	0.911	0.911	0.911
TFIDF-LR_pre	0.902	0.909	0.897	0.903
TFIDF-NN_pre	0.909	0.916	0.903	0.909
word2vec-NN	0.834	0.828	0.846	0.831
word2vec-NN-1st	0.858	0.864	0.851	0.861
word2vec-NN-2nd	0.870	0.874	0.868	0.872
word2vec-NN-3rd	0.872	0.872	0.875	0.872
word2vec-NN-pre	0.883	0.890	0.875	0.886
BERT w/o fine tune	0.857	0.881	0.826	0.853
BERT fine tune	0.961	0.964	0.957	0.960
Bert Lora	1.0	1.0	1.0	1.0

As mentioned in the previous tasks, we have 2 versions of datasets. One is preprocessed while the other one is merely approximately 18.5% of the original data after balance.

So we have methods with a "_pre" which denotes the preprocessed dataset used. For TFIDF-based Logistic Regression and Neural Network methods, two different datasets have similar results. However, the non-preprocessed TFIDF-based Neural Network has a better result.

Analysis: TFIDF relies on the frequency of words in the document to calculate weights. The removal of HTML tags changes the word distribution of the document, which may lead to a reduction in the weight of key information, thereby affecting the performance of the model.

For the word2vec part, as we mentioned, we have 3 steps of improvement, and every step of improvement leads to a better result according to accuracy. After implementing the preprocessed dataset, it has the best results in the word2vec-based method.

Analysis: Regarding the 3-step improvements, we have discussed in task 2. So we only analyze how preprocessed data improved performance according to the F1-score which is different from the TFIDF-based method.

Word2Vec focuses on the semantic relationship between words, and the generated vector representation can capture context and word meaning similarity. Therefore, even if HTML tags are removed, the semantic relationships between words are still preserved and sometimes even become more obvious because non-content structural interference is removed. For the Word2Vec model, clear and consistent text data helps to better learn the relationship between words.

The presence of HTML tags may interfere with the learning of these relationships. Once these labels are removed, it may be easier for the model to recognize and learn valid semantic patterns.

Then we can move on to the BERT part.

The overall performance: Bert with Lora > fine-tuned BERT > without fine-tuned BERT

Analysis:

Through fine-tuning, the model further learns task-specific features and details based on pre-training. Even a small amount of task-specific data is enough for the model to adjust its parameters to better solve a specific classification task. This kind of adaptability is lacking in untuned models, which can only rely on their general language understanding capabilities and are not optimized for specific tasks.

By adjusting fewer parameters, LoRA can achieve rapid fine-tuning with limited resources. And because the number of parameters to adjust is reduced, the model is less likely to overfit small datasets. And subsequent more detailed adjustments to the hyperparameters may promote the stability of the learning process.