DAT410 Module 8 Assignment 8 – Group 26

Yahui Wu (MPMOB) (15 hrs)
yahuiw@chalmers.se
Personal number: 000617-3918
Tianshuo Xiao (MPMOB) (15 hrs)
tianshuo@chalmers.se
Personal number: 000922-7950

March 17, 2023

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions

Background

In the passing of the epidemic, the movie industry resumed its usual prosperity and more and more people went to the movies. Today there are many studios that make a large number of films, but the quality varies and people have mixed reviews of these films.
Depending on the problem space and available data, there are several approaches to building ML models for various text-based applications. In the AI tool, we learn a lot of machine learning models. Classical ML methods for spam filtering, such as "plain Bayesian" or "support vector machines", have been widely used. Deep learning techniques work better for natural language processing problems such as sentiment analysis and language translation. Sometimes, deep learning models are very slow to train, and it can be seen that classical ML methods can give similar results for simple text classification problems with much faster training times. In the module "Natural language processing", we implemented a translation system using machine translation methods based on IBM model 1 which could translate a word based on the alignment and find the most likely structure of the sentence. However, IBM model 1 just put the translated words into sentence grammatically based on Markov chains. In a machine translation problem, the context will affect the accuracy of our translation directly. We need to consider the surroundings of a word to give the correct translation. In module 7, the chatbot also needs to consider the previous dialogue to give an answer more accurately and naturally. In both modules, a system with memory to deal with the context problem could be useful. As a more advanced neural network with better memory than traditional RNNs, LSTM is a great option in natural language processing since it has long-term memory to store previous outcome. As a typical natural language processing problem, the inputs in sentiment analysis also have strong contextual relevance which can be solved by LSTM. In our project, we compare the performance between Classical ML methods and Neural Network Model.
We collect movie reviews from different audiences on Rotten Tomatoes Movie Reviews from Kaggle, analyze critics' multi-categorized sentiments, and determine from the text corpus whether the sentiment is positive or negative for any topic or product, etc. We will use different classifiers and compare their accuracy and performance.[1]

Assumptions

Before we process the data, we made the following assumptions
1.Sentiment expressed in movie reviews is context independent. Machine learning algorithms for sentiment analysis assume that the sentiment expressed in a review is independent of the context of the review. This assumption means that the algorithm does not consider the theme or topic of the movie being reviewed, but only focuses on the sentiment expressed in the text.
2.The sentiment expressed in a movie review is independent of the personal characteristics of the reviewer. The machine learning algorithm assumes that the sentiment expressed in the review is independent of the personal characteristics of the reviewer, such as their age, gender, cultural background and personal preferences. This assumption means that the algorithm does not consider the identity of the reviewer when analyzing the sentiment of the review.
3.The sentiment expressed in a movie review is represented by a specific word or phrase. Machine learning algorithms used for sentiment analysis assume that emotions are expressed by specific words or phrases. This assumption means that the algorithm identifies certain words or phrases as positive or negative sentiments, rather than considering the overall tone or context of the text.

Implementation

Classical machine learning models

Data pre-processing

In the dataset, we labeled phrases according to the mood of the movie reviews on a scale of five values: negative, somewhat negative, neutral, somewhat positive, and positive, as shown in the Table 1 below.

Labels	Sentiment
0	negative
1	somewhat negative
2	neutral
3	somewhat positive
4	positive

Table 1: The sentiment labels

In our train data, there are 156,060 data sets in total.In the data, people negatively evaluated a total of 7072 groups, accounting for 4.5%, somewhat negative a total of 27273 groups, accounting for 17.5%, neutral a total of 79,582 groups, accounting for 51%, somewhat positive a total of 32,927 groups, accounting for 21.1%, positive a total of 9,206 groups and percentage is 5.9%.
In order to balance the data, we decided to use two types of data, negative and positive. After processing, there are 7072 sets of negative data, which represent 43.5%, and 9,206 sets of positive data, which represent 56.55%. Therefore, this data set is relatively balanced through our data processing.

	Phrase	Sentiment
0	A series of escapades demonstrating the adage ...	1
1	A series of escapades demonstrating the adage ...	2
2	A series	2
3	A	2
4	series	2

Figure 1: Data processing

Then we removed some unimportant data, such as PhraseId, SentenceId, and then draw box plot to obtain the overall distribution of comment sentence lengths for each sentiment category.From the box plot, we saw that some of the comments were over 100 characters long.

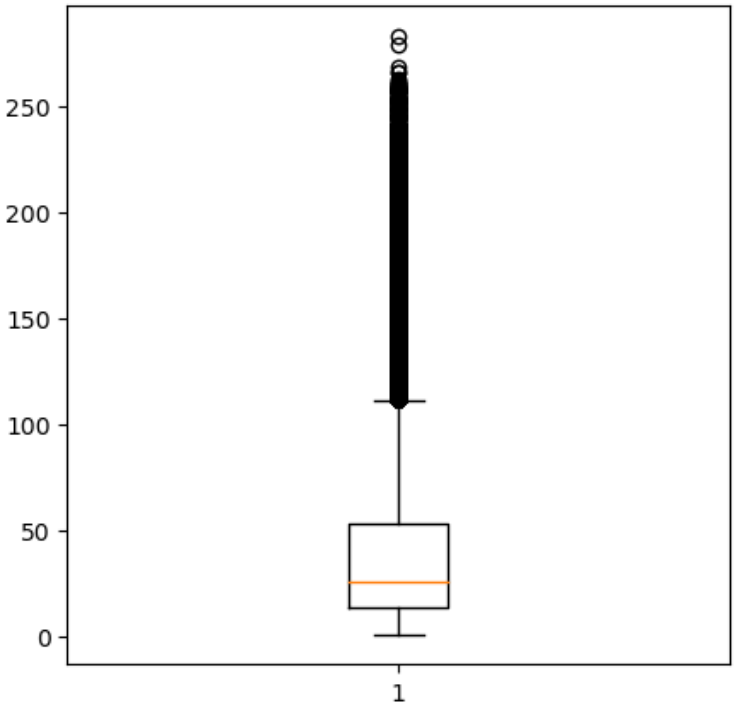


Figure 2: Box plot of sentence lengths

Visualization of vocabulary

We filtered out positive and negative reviews.We found that even if the text contains the word "good", it is still a negative sentiment because it indicates that the movie is not a good movie.Next, we prepared to visualize these reviews. In this case we used the method of creating word clouds of negative and positive movie reviews to visualize the reviews.

Figure 3: Word cloud of negative movie reviews

Figure 4: Word cloud of positive movie reviews

With the cloud chart we see how often most words appear in the dataset. We need to see which words are used in the movie reviews and how many times they are used. In this case, we extracted the comments of different emotions and used CountVectorizer to calculate the word frequency.

Terms	negative	somewhat negative	neutral	somewhat positive	positive	total
the	3462	10885	20619	12459	4208	51633
and	2549	6204	10241	9180	4003	32177
of	2277	6660	12287	8405	3073	32702
to	1916	5571	8295	5411	1568	22761
is	1372	3362	3703	3489	1550	13476
...	...	...	...	...	...	...
obsession beneath	0	1	6	0	0	7
beneath hearst	0	1	7	0	0	8
hearst forced	0	1	8	0	0	9
forced avuncular	0	2	8	0	0	10
avuncular chortles	0	2	8	1	0	11

Figure 5: Word frequency

Through the above figure, we can clearly see that words like "the", "in" and "it" are much more frequent, but they do not have any significance on the mood of the movie review. On the other hand, words such as "forced avuncular" appear less frequently in the document, but seem to have a strong relationship with the mood of the movie, which means that if we increase the weight of these low-frequency words in the classification, we will get a higher accuracy rate of prediction.

Then, we counted the 50 most commonly used words in negative comments and positive comments, as shown in the table below.

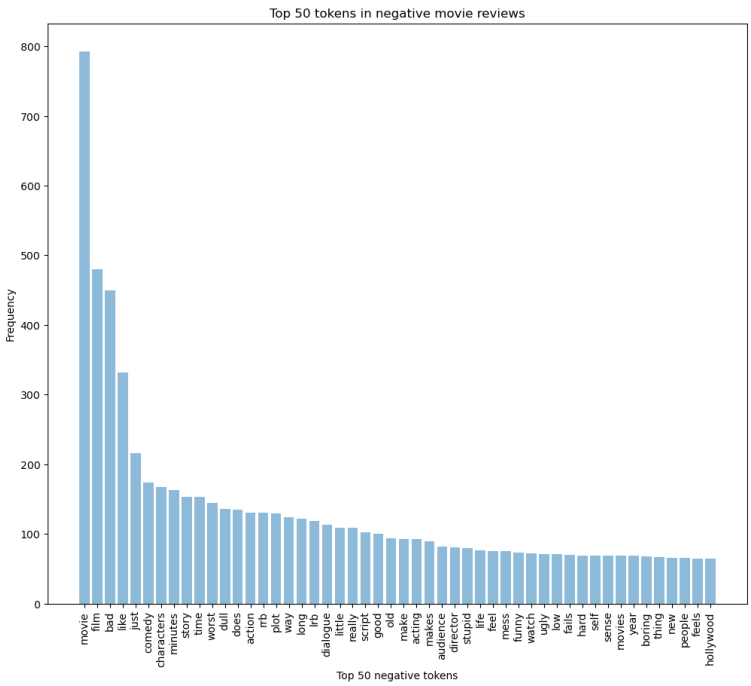


Figure 6: Negative tokens

Among the negative tokens, we found that the words appear mainly such as 'bad', 'dull', 'ugly', etc., but there is no lack of positive words, such as good, comedy, etc. It is possible that among the negative reviews, some of them have satirical meaning, which may affect the accuracy of the model.However, in positive tokens, most of the words are positive, such as 'funny', 'like', etc.

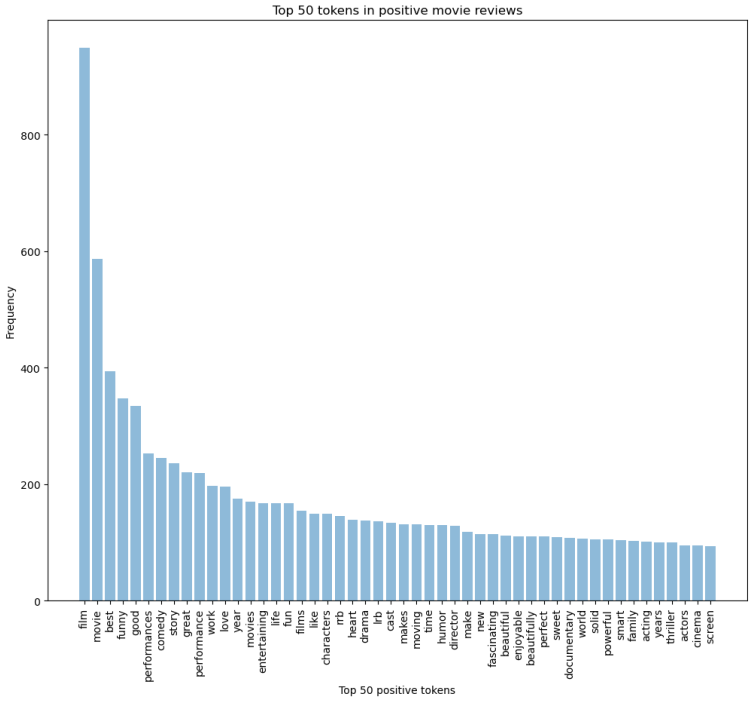


Figure 7: Positive tokens

Data features

The original training set we handled with is comments about movie comments consisting of a list of documents, where each document (comment of each instance in this case) is represented as a single string in English. To make the training set interpretable for machine learning classifiers, we would like to process it and obtain numerical features for each instance in the data.
We used TfidfVectorizer/CountVectorizer to convert the comments to rows of vectors. CountVectorizer is a common class of feature value calculation, a text feature extraction method. For each training text, it only considers how often each word appears in that training text. countVectorizer converts the words in the text into a word frequency matrix, and it calculates the number of occurrences of each word by the fit_transform function.
The number of rows equals to the number of instances from training set and each element in the row vector is a numerical value that represents the probability of a certain word appearing in the document. By processing the comments in training set using TfidfVectorizer, we could obtain the feature of each instance, where the number of features reflected how many words in the bag-of-words and the feature value was the word frequency. In the following we will compare the performance of these two feature calculations.

Comparison of different models

The task is to predict a given comment whether it is pro-vaccine or anti-vaccine. We have to do the prediction and label a test comment as 1 (positive) or 0 (negative). Therefore, we considered some machine learning algorithms that were widely used to solve such a binary classification problem.
The SGD algorithm works by iteratively updating the model's weights to minimize a loss function on the training data. As the algorithm progresses, the SGD classifier gradually learns to adjust its decision boundary to better separate the two classes of data. LogisticRegression is another useful way

to handle binary classification problem. It calculates the probability that a certain comment belongs to the positive class using sigmoid function. We verify the performance of TF-IDF with CountVectorizer and compare the performance of logistic regression model, SGD model and random forest model for text prediction.

We use Accuracy and F1 Score to evaluate our models. Accuracy represents the proportion of correctly classified samples to the total number of samples. There are two components of correctly classified samples: those with positive prediction and positive true, TP, and those with negative prediction and negative true, TN.The F1 score combines Precision, which reflects the model's ability to distinguish negative samples, and Recall, which reflects the model's ability to identify positive samples. So, the higher the F1 score, the more robust the model is.

We plot the ROC curve by our own defined function.If the number of classes is 2, the function computes the predicted probabilities or confidence scores for the positive class and computes the FPR, TPR, and ROC AUC for the binary classification case. It then plots the ROC curve with the AUC score.If the number of classes is greater than 2, the function computes the predicted probabilities or confidence scores for each class, and computes the FPR, TPR, and ROC AUC for each class. It then computes the micro-average and macro-average ROC curves and their AUC scores.

CountVectorizer

LogisticRegression

Model Performance metrics:
Accuracy: 0.6369
Precision: 0.6176
Recall: 0.6369
F1 Score: 0.613

	precision	recall	f1-score	support
0	0.55	0.28	0.37	1426
1	0.53	0.36	0.43	5428
2	0.68	0.88	0.77	15995
3	0.57	0.44	0.50	6603
4	0.55	0.35	0.42	1760
accuracy			0.64	31212
macro avg	0.58	0.46	0.50	31212
weighted avg	0.62	0.64	0.61	31212

Table 2: Model Classification report

Predicted and Actual	0	1	2	3	4
0	405	610	353	51	7
1	261	1945	2954	247	21
2	61	848	14007	1010	69
3	15	232	3039	2912	405
4	1	23	253	874	609

Table 3: Prediction Confusion Matrix

ROC curve

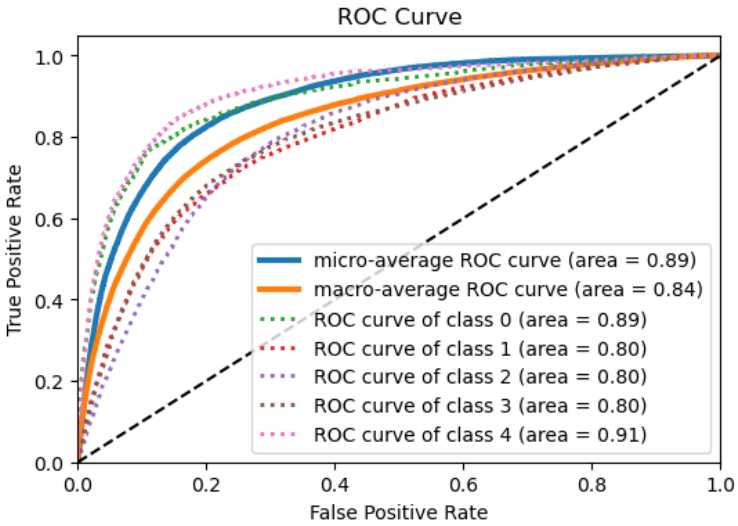


Figure 8: CountVectorizer LogisticRegression ROC curve

SGDClassifier
Model Performance metrics:
Accuracy: 0.5985
Precision: 0.577
Recall: 0.5985
F1 Score: 0.5448

	precision	recall	f1-score	support
0	0.49	0.24	0.32	1426
1	0.53	0.19	0.28	5428
2	0.62	0.93	0.74	15995
3	0.55	0.29	0.38	6603
4	0.53	0.28	0.37	1760
accuracy			0.60	31212
macro avg	0.54	0.39	0.42	31212
weighted avg	0.58	0.60	0.54	31212

Table 4: Model Classification report

Predicted and Actual	0	1	2	3	4
0	337	385	652	49	3
1	252	1019	3911	223	23
2	66	373	14889	609	58
3	22	116	4173	1938	354
4	5	16	507	734	498

Table 5: Prediction Confusion Matrix

ROC curve

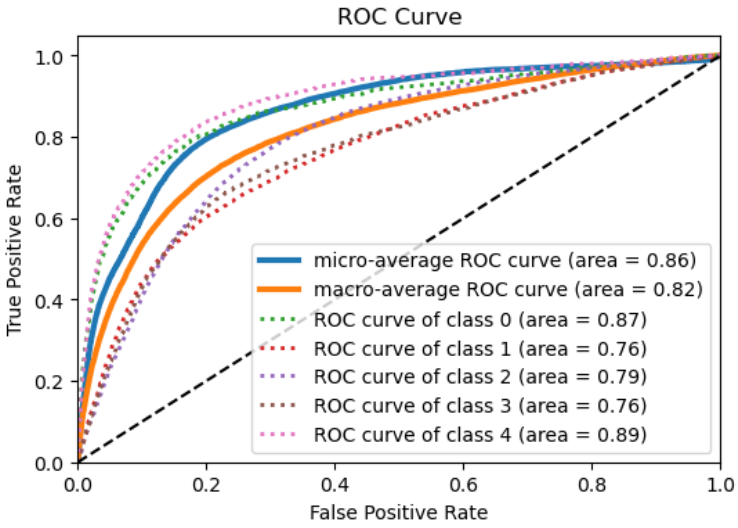


Figure 9: CountVectorizer SGDClassifier ROC curve

From the above data we can get that by CountVectorizer, LogisticRegression has higher Accuracy score(0.6369) and F1 score(0.613) than SGDClassifier.Besides, LogisticRegression has higher AUC values than SGD, which means LogisticRegression performs better.

Classifier	Accuracy score	F1 score
LogisticRegression	0.6369	0.613
SGDClassifier	0.5985	0.5448

Table 6: Model Evaluation

TfidfVectorizer
We used TfidfVectorizer to convert the comments to rows of vectors. The number of rows equals to the number of instances from training set and each element in the row vector is a numerical value that represents the probability of a certain word appearing in the document. By processing the comments in training set using TfidfVectorizer, we could obtain the feature of each instance, where the number of features reflected how many words in the bag-of-words and the feature value was the word frequency.
LogisticRegression
Model Performance metrics:
Accuracy: 0.6455
Precision: 0.6314
Recall: 0.6455
F1 Score: 0.6189

	precision	recall	f1-score	support
0	0.60	0.22	0.32	1426
1	0.56	0.38	0.45	5428
2	0.67	0.89	0.77	15995
3	0.60	0.47	0.53	6603
4	0.60	0.29	0.39	1760
accuracy			0.65	31212
macro avg	0.61	0.46	0.49	31212
weighted avg	0.63	0.64	0.62	31212

Table 7: Model Classification report

Predicted and Actual	0	1	2	3	4
0	312	681	408	22	3
1	177	2051	3066	125	9
2	29	793	14193	944	36
3	2	109	3115	3088	289
4	0	9	281	966	504

Table 8: Prediction Confusion Matrix

ROC curve

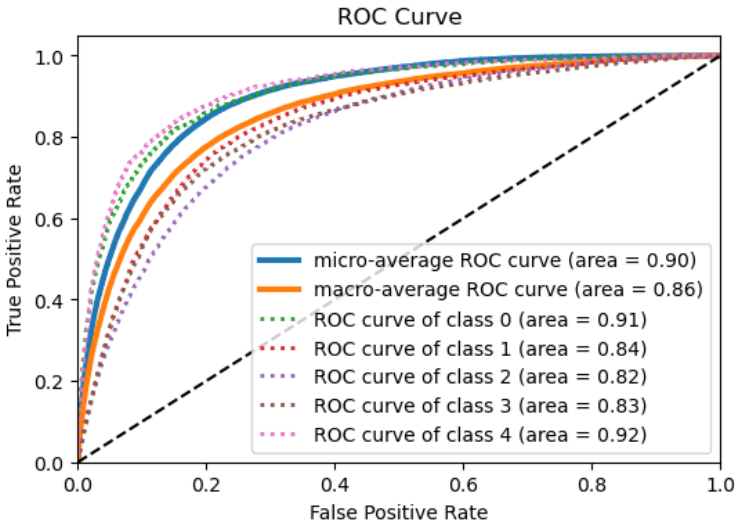


Figure 10: TfidfVectorizer LogisticRegression ROC curve

SGDClassifier
Model Performance metrics:
Accuracy: 0.5593
Precision: 0.5544
Recall: 0.5593
F1 Score: 0.4669

	precision	recall	f1-score	support
0	0.62	0.11	0.19	1426
1	0.52	0.10	0.16	5428
2	0.56	0.96	0.71	15995
3	0.55	0.16	0.25	6603
4	0.58	0.15	0.24	1760
accuracy			0.56	31212
macro avg	0.56	0.30	0.31	31212
weighted avg	0.55	0.56	0.47	31212

Table 9: Model Classification report

Predicted and Actual	0	1	2	3	4
0	156	238	1018	13	1
1	80	516	4758	65	9
2	16	197	15435	320	27
3	1	38	5322	1087	155
4	0	2	991	504	264

Table 10: Prediction Confusion Matrix

ROC curve

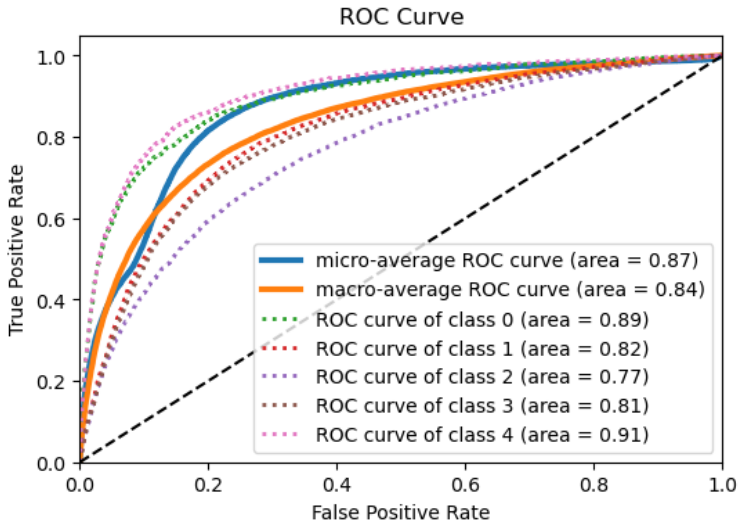


Figure 11: TfidfVectorizer SGDClassifier ROC curve

RandomForestClassifier
Model Performance metrics:
Accuracy: 0.6428
Precision: 0.6272
Recall: 0.6428
F1 Score: 0.6278

	precision	recall	f1-score	support
0	0.48	0.36	0.41	1426
1	0.56	0.42	0.48	5428
2	0.70	0.84	0.76	15995
3	0.58	0.46	0.51	6603
4	0.51	0.40	0.45	1760
accuracy			0.64	31212
macro avg	0.56	0.50	0.52	31212
weighted avg	0.63	0.64	0.63	31212

Table 11: Model Classification report

Predicted and Actual	0	1	2	3	4
0	520	606	281	17	2
1	460	2296	2532	133	7
2	103	1105	13498	1232	57
3	8	117	2810	3046	622
4	1	10	215	830	704

Table 12: Prediction Confusion Matrix

ROC curve

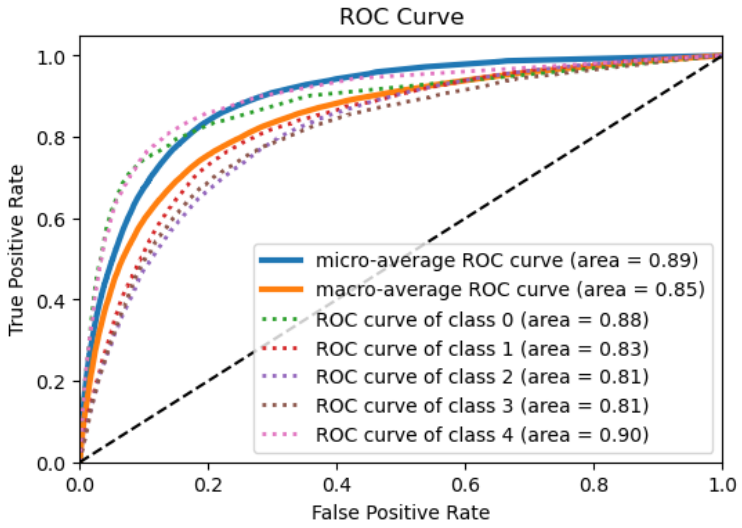


Figure 12: TfidfVectorizer RandomForestClassifier ROC curve

By comparing the accuracy score and F1 score of three models, LogisticRegression, SGDClassifier, and RandomForestClassifier, we find that SGDClassifier has the worst performance, LogisticRegression is higher than RandomForestClassifier, but F1 score is lower than it.

Classifier	Accuracy score	F1 score
LogisticRegression	0.6455	0.6189
SGDClassifier	0.5593	0.4669
RandomForestClassifier	0.6428	0.6278

Table 13: Model Evaluation

By comparing the ROC curves, we find that the AUC value of LogisticRegression(0.9) is higher than RandomForestClassifier(0.89), so LogisticRegression performs better. By comparing several classical machine learning models under two different ways of vectorizing text, we found that under TfidfVectorizer, LogisticRegression model performs better.

Neural Network Model: LSTM

Data pre-processing

By observing our data, we could find that each sentence of a comment given by a certain user of Rotten Tomatoes were split into phrases and each phrase was stored as a instance and given a certain label thus the contextual relevance is stronger.
At the very beginning, we did the standard data preprocessing such as converting upper case letters into lower case letters, removing stop words from each phrase, splitting the phrases into words using a word tokenizer and generating a dictionary containing all the words that occurred.
In natural language processing, a word embedding is used to represent a word so that the document can be converted to the numerical features a neural network can deal with. In our LSTM system, every word of each phrase in the dataset was first converted to the numerical value indicated the position of this word (the index of this word) in the dictionary. Then, we split the data into training set and test set and used the word embedding method to convert each phrase in the document to a 30×400 vector where 30 is the number of words in each phrase (we padded all phrases so that they could have the same number of words which was 30) and 400 is the dimension of each word embedding vector.

	Feature Shape
Train set	(109242, 30)
Test set	(46818, 30)

Build the model

We built the LSTM model based on the pytorch framework. In our model, the batch size is 54 and the time step is 30 which is also the number of cells. There are two hidden layers containing the weights of the input and recurrent connections. The size of weights in each hidden layer is shown as below:

```
torch.Size([14993, 400])
torch.Size([1024, 400])
torch.Size([1024, 256])
torch.Size([1024])
torch.Size([1024])
torch.Size([1024, 256])
torch.Size([1024, 256])
torch.Size([1024])
torch.Size([1024])
torch.Size([5, 256])
torch.Size([5])
```

Figure 13: Layers

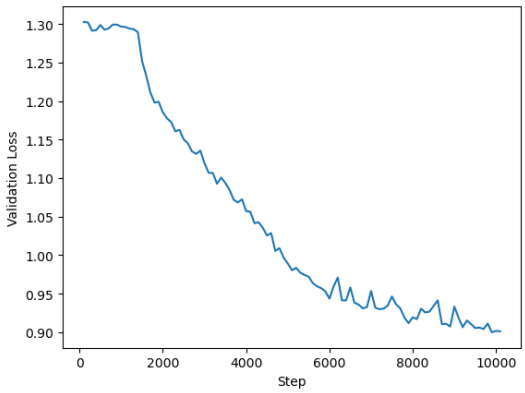
The dimension of hidden layer was set to be 256 and there are 4 groups of weights. Therefore, the size of weights vector in the first hidden layer is 1024×400 where 400 is the dimension of the word embedding vector. The weight of the hidden state from last cell has a size of 1024×256. The output from the first hidden layer has a size of 256×256 so the input weight matrix of the second layer has a size of 1024×256.

Train the model

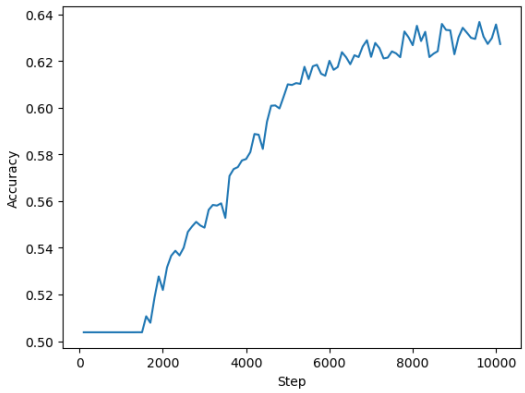
To train the model on the training data, we set the learning rate to be 0.003 and epoch to be 5. It's a muti-classification problem so we used cross entropy loss function. During one epoch, each batch of training data is trained by the model and validation is performed after every 100 batches have been trained. The backpropagation computes the gradient of the cross-entropy loss function based on the Adam optimization algorithm to update the weights.

Evaluation

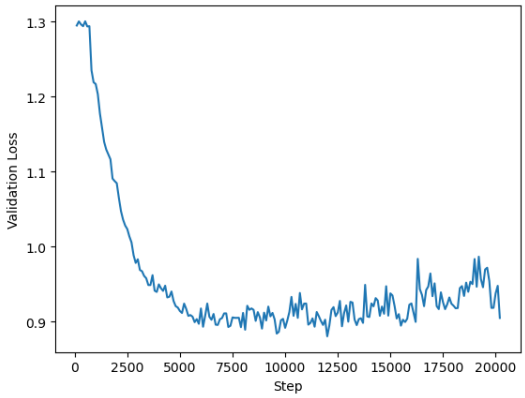
After training for 5 epochs, the loss on validation set dropped to around 0.9 and the accuracy could reach to 0.63. When we set the epoch to 10, the loss curve oscillated a lot and the loss value increased from step 10000 and the accuracy dropped. The reason for the poor performance is that the model will suffer from overfitting when epoch is 10. The loss of training could drop to 0.4 while the during the last epoch while the accuracy no longer increased largely since the training step reached to 7000. However, the performance of this model is still not excellent compared with other LSTM models whose validation loss could drop to a lower value after training for many times and have better accuracy. We believe that the moderate performance comes from the repetitiveness of the data. Many phrases in this data set come from the same sentence that they share the same sentence id. Although the split-up of a whole sentence can let the LSTM model extract features by considering contexts, it still reduce the quality of data.



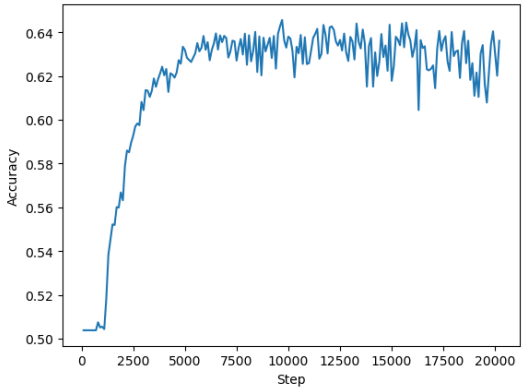
(a) validation loss when epoch=5



(b) accuracy when epoch=5



(c) validation loss when epoch=10



(d) accuracy when epoch=10

Conclusion and discussion

In conclusion, by using TfidfVectorizer, LogisticRegression performs best in classical machine learning model and neural network model.We think the possible reasons are as follows. TfidfVectorizer is a powerful technique for extracting meaningful features from text data. The vectorizer calculates a term frequency-inverse document frequency (TF-IDF) score for each term in the text, which represents its importance in distinguishing between positive and negative sentiment. Logistic Regression can use these features to learn a linear decision boundary that separates the two classes.
However,LSTM performs well in tasks that require modeling long-term dependencies, and in our movie sentiment classification task that only requires modeling short-term dependencies, then LSTM may not have a significant advantage over classical machine learning models. Moreover, through the previous data processing we can see that many positive reviews are included in the negative reviews, which makes the dataset unrepresentative and leads to a low accuracy rate of LSTM.We can also optimize our model by making hyperparameter tuning, which may help us get more accurate results.

References

[1] Movie review sentiment analysis (kernels only). Kaggle. (n.d.). Retrieved March 17, 2023, from https://www.kaggle.com/competitions/movie-review-sentiment-analysis-kernels-only/data

[2] Shekhar, S. (2021, June 30). LSTM for text classification: Beginners Guide to Text Classification. Analytics Vidhya. Retrieved March 17, 2023, from https://www.analyticsvidhya.com/blog/2021/06/lstm-for-text-classification/

Reflection on the previous module

Yahui Wu

In the last module, we implemented a simple dialogue system which could help users find restaurants, the next bus and today's weather. This AI assistant was built based on keywords search and no other more advanced methods were used to ensure the naturalness of dialogue. In our simple digital assistant system, the answer from the bot is given by searching the aim keyword in the sentence which means only a certain keyword could activate the bot and let the dialogue go on. The user must follow the steps to get an answer instead of asking directly. To achieve more intelligent function, a system which can process a whole sentence and infer the meaning of the sentence is needed. RNNs could be a good choice in NLP since it considers the prior word while predicting the next word to take the contextual factor into account. Other similar methods with a encoding-decoding mode could also be helpful. The dialogue or sentence processing based on RNNs can be improved by introducing attention mechanism which considers the surroundings of the dialogue of contexts more thoroughly. With the attention mechanism, the encoder distributes different weights to every word in the context and the decoder will take account of those weights. Many algorithms which introduced the attention mechanism has shown great performance in NLP. Transformer contains seff-attention mechanism and forward networks in each encoder. The decoder in transform has similar structure but a encoder attention mechanism is added to take the information from encoding process into account. Many variants from transformer are used in NLP recently such as BERT and GPT.

Tianshuo Xiao

We reflected on our last dialogue system. The biggest drawback in our model is the lack of natural language processing. Our model relies only on keyword search to carry out dialogues, and the dialog system may fail to recognize when people ask for words or sentences with similar meanings to the keywords, so we need to improve this.
We came up with the following enhancements. Increase training data. Our dialogue system is based on machine learning models, and the quality and quantity of training data can have a significant impact on performance. Therefore, we can collect a large corpus to train our model and thus improve the accuracy of the system. Fine-tune pre-trained models. A pre-trained language model, such as BERT, can be fine-tuned for your specific task or domain to improve its performance. Fine-tuning involves training the model on additional task-specific data, which can help it better understand the nuances of your language. Use word embeddings. Word embeddings are vector representations of words that can help capture their semantics. By using word embeddings in conversational systems, you can improve the accuracy of natural language processing tasks, such as sentiment analysis or entity recognition. Use context. Understanding the context of a conversation can help dialogue systems better interpret and respond to user input. We can use techniques such as named entity recognition or coreference to identify entities and their relationships in a conversation. Dealing with ambiguity. Natural language is inherently ambiguous, and users may express the same idea in different ways. Our dialogue systems should be able to handle ambiguity and understand the user's intent, regardless of the wording they use. Support multiple rounds of conversation. To create a more natural and engaging conversation, the dialogue system should be able to remember and go back to previous rounds in the conversation. This can be achieved through technologies such as conversation state tracking or memory networks. Ability to handle user input errors. Users may make mistakes when typing or speaking. d Dialogue systems should be able to handle these errors gracefully. This may involve technologies such as spell checking, error correction, or providing suggestions for alternative input.

Reflection on the course as a whole

Through Design of AI systems, we opened the door to the world of AI. We learned about different AI algorithms and solution solutions, which gave us how to build a reasonable AI model. First, we tried to characterize each problem, then chose different algorithms to build the model. Then we learned to build a recommendation system to recommend movies using collaborative filtering. Besides, we learn to use AI tools to solve problems, to predict different cities air quality. In this part, we acquire how to build a K-Means Classifier and evaluate the performance of the system. Then, we learned about NPL. We knew about bigram language model, and the main idea of this algorithm is word alignment, which translates other languages into the target language. We discussed the use of diagnostic systems in AI, focusing on understanding the interpretability of AI models and the problems that black boxes cause to people. Then, we studied Game playing systems, which mainly used the Monte Carlo tree search algorithm. We studied the principles of this algorithm and became very interested in how the game is driven. Finally, we learned about Dialogue systems and question answering, which are used in all parts of people's lives, and nowadays chatbot is all the rage to make our lives easier.
Although we are not computer science majors' students, this cause has made us interested in AI systems. We learned a lot about AI system development especially the translator for natural language processing and the human-computer confrontation of Go based on Monte Carlo algorithm. In conclusion, this course gave us a perfect experience.