COMP4332 Project 1 Report

Group 28
HUI Man Wah, WONG Ho Leong, WONG Yu Ning, YIP Nga Yin

Introduction:

This project aims to predict the sentiment score over a set of reviews. The reviews are mostly about the content of movies or the dvd/blu-rays they purchased. This report aims to explain the attempts we made to build our model for this task.

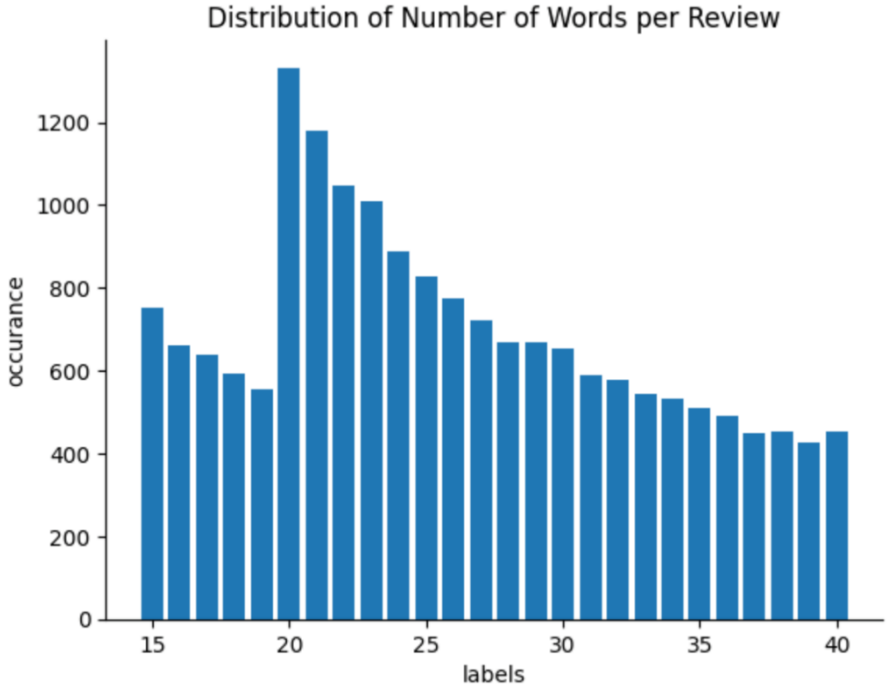
Dataset Description:

There are only 2 features in the dataset, id and text. Since id is unique for all reviews, it is not used for prediction. The sentiment labels of the reviews range from 1 to 5. Training dataset contains 18000 data instances, validation set contains 2000 and testing dataset contains 4000 data instances.

Exploratory Analysis:

It is important to gain insights on the dataset before deciding the data preprocessing techniques and model used for this task.

The average review length of the review in training dataset is 26, with no missing values. The following are the distribution of number of words per review in the training dataset.

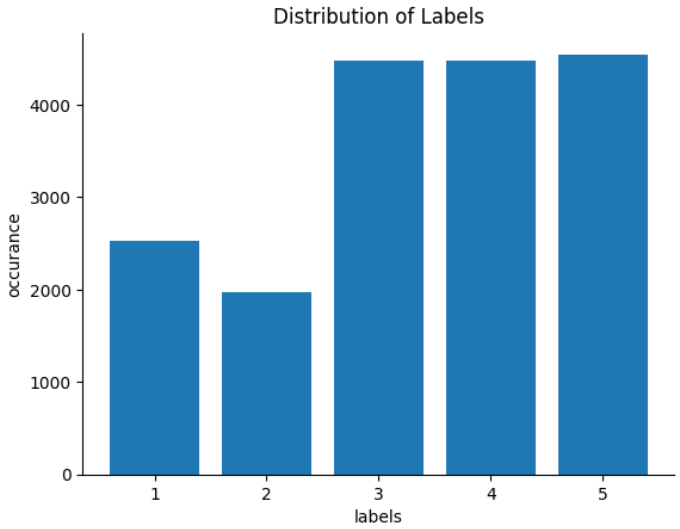


The minimum and maximum number of words per review are 15 and 40 respectively.

In addition, we attempt to look for the most common words exists in each category. In this analysis, we exclude the stopwords using nltk package.

Table 1 Common words for each sentiment labels				
1	2	3	4	5
movie	movie	movie	movie	movie
one	like	good	good	great
dvd	good	like	great	one
like	one	one	one	love
bad	film	great	like	good
film	story	story	story	watch
watch	much	watch	film	movies
time	didnt	film	watch	classic
good	better	movies	movies	time
would	watch	would	time	story

We can see that most common words are the same for all labels. Therefore, it is unlikely that bag-of-words models can obtain a satisfying result. Due to this finding, we believe that n-grams modelling may be necessary for this task.
Finally, we visualize the distribution of labels in the training dataset.



We can see that the class imbalance issue is serious in the dataset. Specifically, the number of data instance with sentiment score 1 or 2 are significantly less than that of other labels. Undersampling or Oversampling techniques may need to be used to address this issue.

Furthermore, when we take a closer look at the dataset, we can see that there are garbled characters, misspelled words, hyperlinks or wrong words in the text. Some examples that we think worth our attention are listed below.

Table 2 Special D	
Garbled characters	very good very wonderful ve fo scvjkldnhvclksnr sdlk/fnhrskdvcnh jdgsklvnhlsdfkn ,/sdgfnsvhk lksdngfwklrsvn /lskdngfrlkvfnhlkn lksdfnglkvnhlkdzsfne/Kn lk/gtrnjwlrkjnlgkn/s,dfnewL:Knj .,sdngtwlrknvtgnlsrkn sdfnglwkre lknjsdFK sdnmfkl; .,dfgnmrkl;nj ,snmkdfkl
Misspelled words	This movie is a definate classic witch has been remastered into its original black & white release which is clearest sharpest that I have ever seen.
Wrong words	this is one of the best movies ,i am one of the # 1 fans I have the very fist case of it it truly unlocks you're child inside. MUST SEE.
Hyperlinks	<a data-hook="product-link-linked" class="a-link-normal" href="/Resident-Evil-Apocalypse-Blu-ray/dp/B000EZ7ZZE/ref=cm_cr_arp_d_rvw_txt?ie=UTF8">Resident Evil: Apocalypse [Blu-ray]MILLA JOVOVICH, MILLA JOVOVICH. NEED I SAY MORE. I THINK NOT.
Reviews with all Capital letters	DOUBLE LAYER & DOUBLE SIDED...THAT SAYS IT ALL ABOUT THE VERY POOR QUALITY OF THE MEDIA. THE ONES I BOUGHT HAD TO BE RETURNED AND I'M STILL SEARCHING FOR SEASON ONE ON SINGLE LAYER, SINGLE SIDED.
Short form of words	This is a movie that u need in ur collection that involves passion of love though diffcult times and trying to make the best out of everything that is given to you
Repeated words	Ok now I am afraid to try any other Our Gang DVDs! No one mentions how poor the sound is, so that it it almost impossible to understand what is being said! This made this DVD worthless in my opinion.
Separate letters that form a word	it's just what i need v e r y n i c e p r o d u c t
Excesss punctuations	good old western,,not a stupid Italian one either,,it's more the way it should be,,eat your heart out ,,[who's that guy that talks to chairs]
Masked words	How dare you send me a copy of a movie in a badly damaged casing you should close up shop and find another profession . I WILL NEVER BUY FROM YOU B********DS AGAIN tml
Contains other languages	Excellent purchase..... I liked and I buying other in the future.... Excellent....! Me gusta muchas veces, muchas veces, muchas veces.....
Text with negations	Five stars for the movie, but minus 5 stars for colorizing it. Oh yuck. It was perfect as it was.

Names:

Apart from the examples listed above, we also found a lot of Actor's and movie's name in the review. They may not be distinctive features because some may think the movie is bad while some may think that it is good. However, since names are often uncommon, they may be considered as distinctive features during model training and gives false predictions.

Emoji:

Emoji are also common in the review. For instance, :), :D can be seen in the comment. This indicate that removing all punctuations may not be a good choice since punctuations like !, :,) may contain emotions.

Key Data Preprocessing Techniques

We used to tfidf + logistic regression baseline model to test the effectiveness of each data preprocessing techniques.

Spelling Checking

As mentioned in Exploratory Analysis Section, we found a lot of spelling mistakes in the datasets. However spelling corrections libraries (ie. SpellChecker, TextBlob) either did not give satisfying results or has long running time. Therefore, we cleaned the data using the autocorrecting functions in Microsoft Excel and Word. Apart from spelling mistakes, the autocorrecting function also helps to locate the repeated words and masked words in the text. After running spelling correction, the baseline model's accuracy increases.

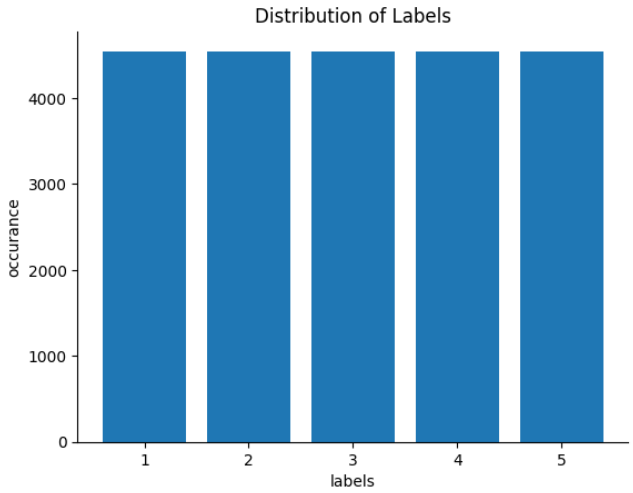
The results after spelling corrections:

Table 3 Accuracy table Spell Correction	
Data preprocessing techniques used	Baseline model's accuracy on validation set
Without any preprocessing (baseline model)	0.527
Spelling Correction	0.5315

OverSampling and UnderSampling

Then, we noticed class imbalance problem in the training dataset. To solve this issue, we tried the RandomOversampling() in scikit-learn and Class Weight Calculation methods to deal with this issue.

RandomOversampling randomly create samples for classes with less data. The distribution of class label after resampling is shown below:



Another method to deal with class imbalance problem is to include class weights during model training, class with less data will weigh more during loss calculations. The formula we used to calculate the class weight is:

*Weight for class c = Total number of data/(number of class*number of occurrence of class c)*

Table 4 Accuracy table of model using Oversampling and Undersampling method	
Data preprocessing techniques used	Baseline model's accuracy on validation set
Spelling Correction	0.5315
Spelling Correction + OverSampling	0.5320
Spelling Correction + Undersampling	0.5280

We can see that model accuracy slightly increases after applying oversampling.

Partial Punctuation Removal

In Exploratory Analysis Section, we mentioned some people likes adding emojis into their review, and these emoji can help to distinguish emotions. Therefore, we tried to remove punctuations that have the highest frequency among training dataset, which is comma and full stops.

```
def partialpunctuationRemoval(s:str):
  s = s.translate(str.maketrans('', '', ",."))
  return s
```

Then, we compared the accuracy of removing all punctuation and removing commas and full stops only. We can see that removing partial punctuations yields higher accuracy than removing all punctuations.

Table 5 Accuracy table of model using Punctuation Removal	
/	Baseline model's accuracy on validation set
Punctuation removal	0.5270
Partial Punctuation removal (only remove , and .)	0.5340

HyperLinks, HTML tags Removal

We observed the hyperlinks and html tags in the text. These links are often garbled characters and does not give valuable information about the sentiment of the text.
To perform links removal, we implement the following function:

```
def hyperlinksRemoval(s:str):
  s_processed = re.sub(r'^https?:\/\/.*[\r\n]*', '', s,
flags=re.MULTILINE)
  p = re.compile(r'<.*?>')
  return p.sub("",s_processed)
```

The first line of code aims to remove http links from the text and the second line of code removes anything inside <>. After performing hyperlinks and html tags removal, The accuracy does not show significant increase.

Special Character Removal

We also found '\n', '\t' in the text. The special characters may be treated as characters and punctuations during tokenization and impact the accuracy of the model. We implemented the following function to remove these special characters. The baseline model accuracy increased by 0.0015.

```
def specialcharacterRemoval(s:str):
  return s.translate(str.maketrans('', '', "\t\n"))
```

Stop words Removal

Stop words removal are common preprocessing technique. However, the accuracy of the baseline model dropped from 0.5315 to 0.4995 after applying stop word removal using nltk package. We think the possible reason for this observation is stop words such as "not", "no" contain important information about the sentiment of the text. If we remove "not" from the text, the meaning can become opposite, which affects the accuracy. Therefore, we think stop words removal may not be useful in this task.

Stemming and Lemmatizing

Both techniques result in worse accuracy compared to the baseline model (accuracy = 0.5315).

Table 6 Accuracy table of model using Stemming and Lemmatizing

Data preprocessing techniques used	Baseline model's accuracy on validation set
Stemming	0.4810
Lemmatizing	0.520

Ngrams

We tried all combinations from 1-gram to 4-gram. Among them, the combination of 1-gram and 2-gram gives the best accuracy (0.5315).

To summarize, we found that using spelling corrections, removing hyperlinks, html tags, special characters and partial punctuation removal and using both unigrams and bigrams of the text gives the best result in the baseline model (0.5385).

Experiments

This section aims to list and explain the key models we have tried.

1. Logistic Regression model + TFIDF + Regexp Tokenizer

After trying several tokenizers, we found that using the Regular Expression Tokenizer (RegexpTokenizer) with the pattern r'\w+|[^\w\s]+' gives most satisfying result. It splits the text into tokens based on the specified regular expression pattern. In this case, the pattern r'\w+|[^\w\s]+' has two parts:

1. \w+: This part matches one or more word characters (\w). Word characters include letters, digits, and underscores. This pattern represents tokens consisting only of word characters.

2. [^\w\s]+: This part matches one or more non-word and non-whitespace characters. The ^ inside the square brackets negates the character class, so it matches any character that is not a word character (\w) or whitespace character (\s). This pattern represents tokens consisting of non-word characters like punctuation marks or symbols.

Advantages of using regex tokenizer:

1. Specific tokenization: The regex tokenizer allows more precise tokenization by considering word boundaries and separating non-word characters. In our data, there are punctuation, symbols, and specialized text patterns that may affect the meaning or sentiment of the text.
2. Handling non-standard text: The Regex tokenizer can handle non-standard text scenarios, such as hashtags, mentions, URLs, or emoticons, which may be important in our analysis. By properly tokenizing these special patterns, the model could capture more meaningful information.
3. Better feature representation: By capturing specific patterns through tokenization, the Regex tokenizer can contribute to better feature representation in the vectorized form. This can provide richer information to

the logistic regression model and improve its accuracy.

Implementation

```
import regex
from nltk.tokenize import RegexpTokenizer
tokenizer1 = RegexpTokenizer(r'\w+|[^\w\s]+')
def fitTFIDFandLR(x_train, y_train, x_test,y_test,classWeight =
None,n_gram = (1,1),resample = False):
  tfidf =
TfidfVectorizer(tokenizer=tokenizer1.tokenize,ngram_range =
n_gram)
  lr = LogisticRegression(class_weight=classWeight, max_iter=1000
  tfidf.fit(x_train)
  x_train_vectorized = tfidf.transform(x_train)
  x_valid_vectorized = tfidf.transform(x_valid)
  y_train_mod = y_train
  if (resample):
    x_train_vectorized, y_train_mod =
ros.fit_resample(x_train_vectorized, y_train_mod)
  lr.fit(x_train_vectorized, y_train_mod)
  #predictions
  y_pred = lr.predict(x_valid_vectorized)
  print(classification_report(y_valid, y_pred))
  print("\n\n")
  print(confusion_matrix(y_valid, y_pred))
  print('accuracy', np.mean(y_valid == y_pred))
  return tfidf, lr
```

Comparison with the other tokenizer:

Table 7 Accuracy table of model using different tokenizers

	Bag of words	Word_tokenizer	Gensim tokenizer	Regexp	Baseline tfidf
Without any preprocessing	0.5025	0.532	0.532	0.536	0.5285
lemmatize	0.491	0.526	0.526	0.5285	0.527
Remove stopwords	0.504	0.518	0.518	0.5215	0.518
lower	0.5025	0.532	0.532	0.536	0.5285
Punctuation removel	0.499	0.529	0.529	0.5295	0.5295

Lemmatize + punctuation removel	0.5005	0.533	0.533	0.532	0.531
Hyperlink	0.507	0.5295	0.5295	0.5345	0.5275
One letter word removal	0.5005	0.5325	0.5325	0.534	0.5315
Name word	0.4815	0.523	0.523	0.521	0.5275
spacy	0.2935	0.2955	0.2955	0.293	0.2935
Lower+ hyperlink+ lemmatize+ punctuation removal	0.491	0.526	0.526	0.5285	0.527
Max iter 1000, no processing	0.503	0.5275	0.528	0.5365	0.5295

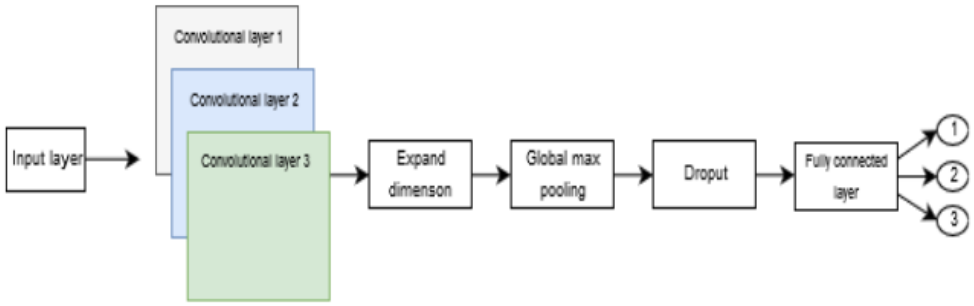
In most preprocessing methods, using tokenizer Regexp generates the best accuracy among all tokenizers tested.

By combining our findings in data preprocessing techniques, the tf-idf + logistic regression model + Regexp tokenizer achieves highest accuracy (0.5445) when oversampling is applied, and the following preprocessing techniques are used:

(1) Hyperlinks Removal
(2) Special character Removal
(3) Partial punctuation Removal

2. Embedding + CNN model

We found that the baseline CNN model will be overfitted to the training data easily. To avoid overfitting, we introduce another Embedding + CNN model architecture [1].



In our model, we use a Word2Vec Embedding layer and 2 CNN layers.

```
ourMLP(
(embedding): Embedding(1193514, 25)
(cnn1): Sequential(
(0): Conv1d(25, 25, kernel_size=(3,), stride=(1,), padding=same)
```

```
(1): ReLU()
)
(cnn2): Sequential(
(0): Conv1d(25, 25, kernel_size=(4,), stride=(1,), padding=same)
(1): ReLU()
)
(dropout): Sequential(
(0): Dropout(p=0.5, inplace=False)
)
(linear): Sequential(
(0): Linear(in_features=50, out_features=5, bias=True)
(1): Softmax(dim=-1)
)
)
```

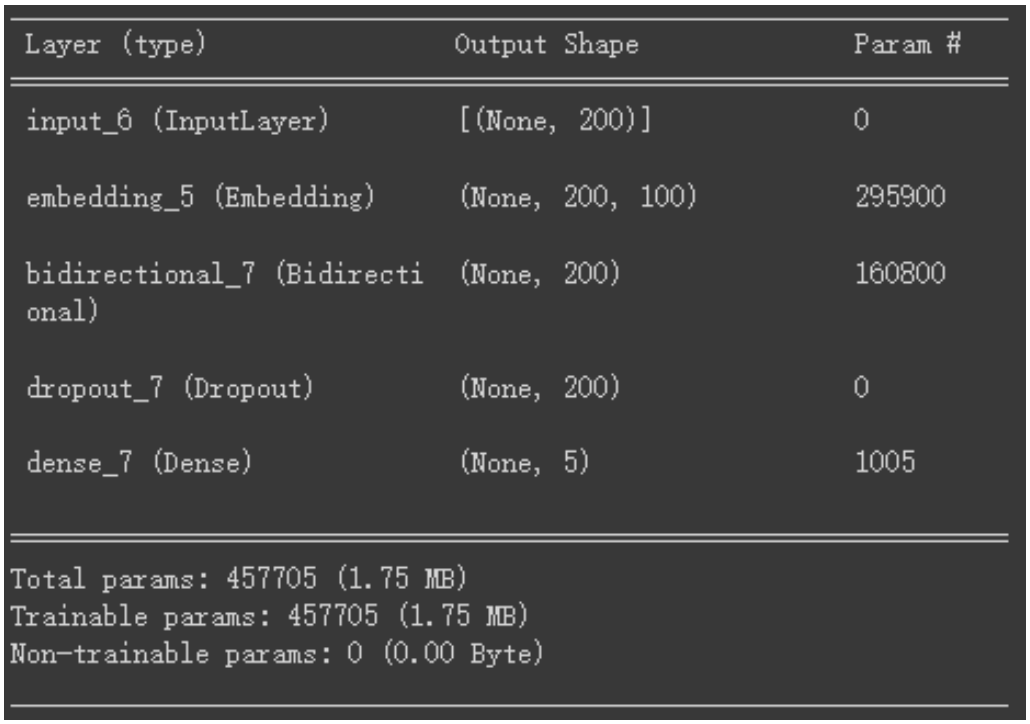
During Training, we used the same preprocessing techniques as the logistic regression. The accuracy reaches 48% at 32th epoch:

```
epoch 32
100%|■■■■■■■■■■| 282/282 [07:08<00:00,  1.52s/it, loss=0.0187, acc=0.707]
100%|■■■■■■■■■■| 32/32 [00:00<00:00, 133.13it/s]
              precision    recall  f1-score   support

           0       0.50      0.52      0.51       295
           1       0.29      0.41      0.34       198
           2       0.47      0.41      0.44       508
           3       0.45      0.41      0.43       523
           4       0.59      0.62      0.61       476

    accuracy                           0.48      2000
   macro avg       0.46      0.47      0.46      2000
weighted avg       0.48      0.48      0.48      2000
```

3. Bi-LSTM model

Bi-LSTM networks could capture contextual information from both past and future inputs, making them adept at understanding the sequential nature of language. This approach excels in accurately predicting sentiment scores. In comparison to simpler models like logistic regression, Bi-LSTM offers superior performance by leveraging the inherent complexity of natural language patterns.

The following shows the Bi-LSTM model architecture.



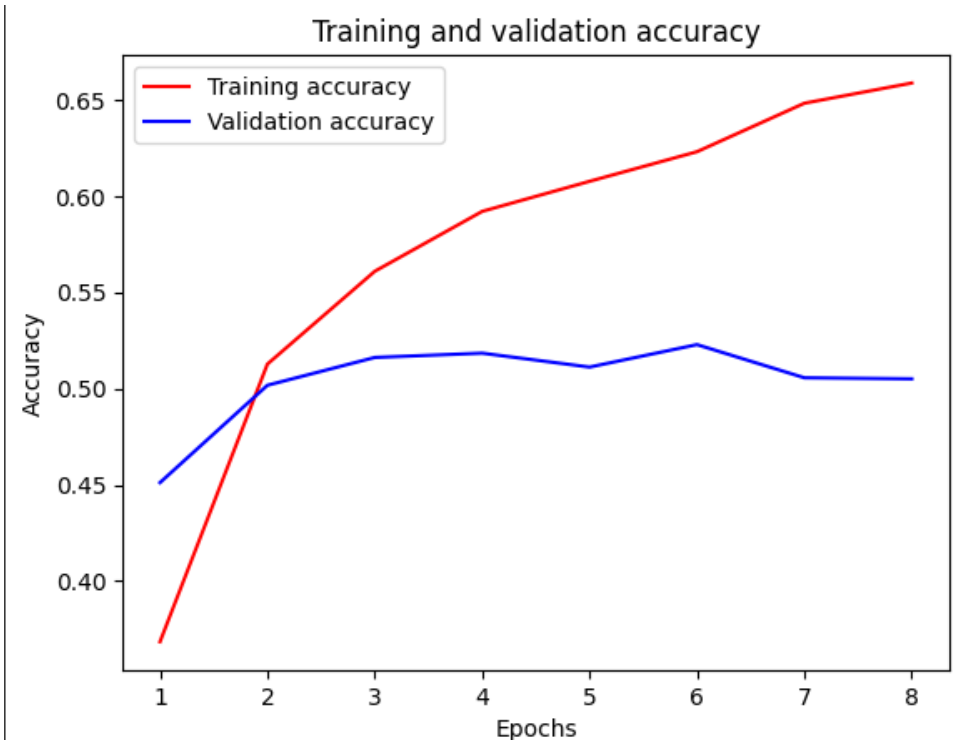
Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 200)]	0
embedding_5 (Embedding)	(None, 200, 100)	295900
bidirectional_7 (Bidirectional)	(None, 200)	160800
dropout_7 (Dropout)	(None, 200)	0
dense_7 (Dense)	(None, 5)	1005

Total params: 457705 (1.75 MB)
Trainable params: 457705 (1.75 MB)
Non-trainable params: 0 (0.00 Byte)

Table 8 Hyperparameters for Bi-LSTM model	
embedding_size	100
hidden_size	100
num_rnn_layers	1
num_mlp_layers	1
activation	tanh
dropout_rate	0.6
batch_norm	True
l2_reg	0.01

After experiments, the Bi-LSTM model gives the best results when the following data preprocessing methods are applied.

Table 9 Accuracy of Bi-LSTM model with/without negations	
Preprocessing techniques used	Accuracy
Lower+ hyperlink+ lemmatize+ punctuation removal + Remove stopwords + tokenization + negation	0.5009



Conclusion

To conclude, we conduct experiments on data preprocessing and model training, and revealed that preprocessing plays an important role in improving model performance. Techniques such as partial punctuation removal, spelling correction, and hyperlinks and HTML tags removal strategies had improved the accuracy of sentiment prediction models. Moreover, after experimenting on various models, we find that logistic regression with TF-IDF gives the best accuracy in validation dataset (54.5%).

Future Work

From our experiments, we can see that TF-IDF + logistic regression still gives the best accuracy. It is because almost all of our deep learning models are overfitted to the training dataset. Therefore, we will continue to explore more on regularization techniques and simpler model architectures.

Additionally, we found that validation accuracy in class 2 is particularly low in both logistic regression model and deep learning models, indicating the models are unable find distinctive features on class 2 data. We will continue to explore the possible reasons behind this situation and look for more suitable models.

Finally, we have only addressed some special case we found in the Exploratory Analysis Section. Problems such as containing other languages in the review are yet to be solved.

Submission

Since **tf-idf + logistic regression model + Regexp tokenizer** achieves highest accuracy (0.5445), we use this model to generate the final pred.csv.

The submission folder includes two Colab notebooks, one contains a number of models, including the tf-idf model that gives the best accuracy, and another one includes the implementation on the Bi-LSTM model. In addition, the cleaned datasets are also included in the folder. To get the logistic model that gives the best accuracy, please run the following code in Models.ipynb.

```
best_tfidf, best_lr =
fitTFIDFandLR(train_text,y_train,valid_text,y_valid,classWeight =
None,n_gram = (1,2),resample = True)
```

Reference

[1] K. Svensson, Sentiment analysis with Convolutional Neural Networks, http://www.diva-portal.org/smash/get/diva2:1105494/FULLTEXT01.pdf.