Wee Kim Wee School of Communication and Information

IMDB Sentiment Classification

H6751 TEXT & WEB MINING

Professor: Assoc Prof Na Jin Cheong

Tian Silei (G2002586G)
Yang Yue (G2002446B)
Tang Zhong (G2002462K)

Problem Statement

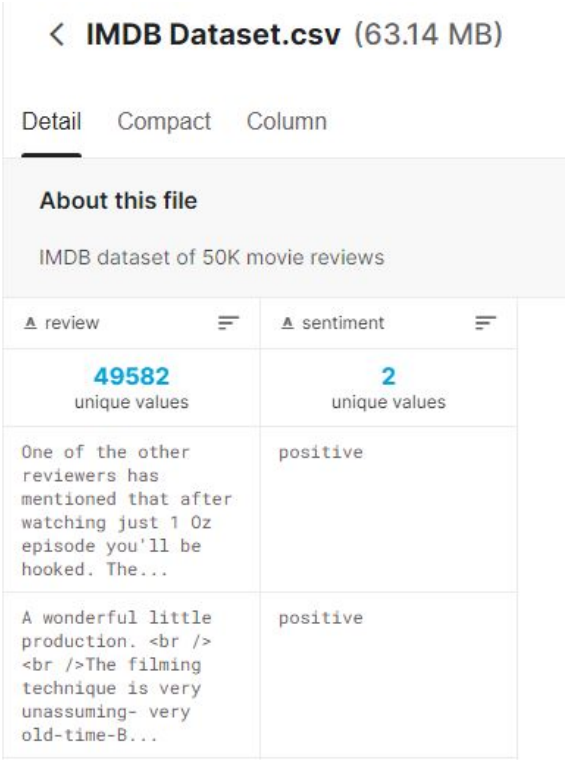
There are multiple approaches and algorithms in the field of neural networks as a method of deep learning. Each of them are capable of the classification problem with their own strength and weakness.

The purpose of this project is to compare the performance with MNN, RNN and CNN on sentiment analysis with 2 output features (positive/negative).

Data Source & Description

The dataset is the IMDB Dataset of 50k Movie Reviews. The dataset consists of 50k movie reviews with sentiment labels as positive or negative.

We used the labeled dataset so that we don't need to manually label the 50k reviews by ourselves. The link below is the data source from Kaggle. The link is provided in the Bibliography 1.



Data Pre-processing

```
# Preprocess the reviews
def preprocess_text(text):
    text = re.sub(r"([.,!?])", r" \1 ", text)     # E.g., convert "end." to "end . "
    text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)  # replace special characters with empty string

    # additional pre-processing
    text = text.lower() # Casefolding

    # Remove stop words
    text = remove_stopwords(text)
    #return text

    # lemmatization and stemming
#     """
    t = ""
    for word, tag in pos_tag(word_tokenize(text)):
        wntag = tag[0].lower()
        wntag = wntag if wntag in ['a', 'r', 'n', 'v'] else None
        if not wntag:
            #lemma = word
            stem = word
        else:
            #lemma = lemmatizer.lemmatize(word, wntag)
            stem = ls.stem(word)
        #t += lemma + " "
        t += stem + " "
    return t
#     """

final_reviews.review = final_reviews.review.apply(preprocess_text)
```

The default pre-processing is removing the special characters.
The additional pre-processing include:

- Case Folding
- Remove stop words
- Lemmatization
- Stemming

```
#final_reviews.to_csv(args.output_munged_csv_default, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_casefolding, index=False)
#final_reviews.to_csv(args.output_munged_csv_without_stopword, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_casefolding_without_stopword, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_lemmatization, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_casefolding_lemmatization, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_casefolding_without_stopword_lemmatization, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_stemming, index=False)
#final_reviews.to_csv(args.output_munged_csv_with_casefolding_stemming, index=False)
final_reviews.to_csv(args.output_munged_csv_with_casefolding_without_stopword_stemming, index=False)
```

The combination of these methods are implemented and stored respectively.

The additional processes like bigram and trigrams will be implemented along with the training process, which will be discussed in the section below.

MNN Model

- **Model Source**

  The model is based on the lab 5 "3_5_Classifying_Yelp_Review_Sentiment".

- **Testing**

Parameters	Test default	1	2	3	4	5	6
frequency_cutoff	25	0	25	25	25	25	25
batch_size	128	128	64	256	128	128	128
early_stopping	5	5	5	5	10	5	5
learning_rate	0.001	0.001	0.001	0.001	0.001	0.0001	0.01
Accuracy	89.86	90.4	90.09	89.95	89.9	90.05	89.28
Loss	0.264	0.257	0.263	0.264	0.264	0.266	0.28

- Adjusting parameters on default dataset

Batch_size=64 and frequency_cutoff=0 provides the biggest improvement. So, we do further tests based on these.

Parameters	Test 0	1	2	3	4	5	6
frequency_cutoff	0	0	0	0	0	0	0
batch_size	64	64	64	64	64	64	64
early_stopping	5	5	5	5	5	5	5
learning_rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001
hidden_dim	/	20	40	10	20	40	10
hidden_layer	0	1	1	1	2	2	2
Accuracy	90.5	89.94	89.18	90.33	89.34	88.74	89.81
Loss	0.256	0.281	0.31	0.259	0.318	0.312	0.299

- Test on additional layers with default dataset with best parameter from above

Unfortunately, adding additional layers does not improve the performance. However, we have the assumption that with the default dataset, additional layers will be too much. Keep in mind that we may test on these later.

- Test on different pre-processings with best parameter from above

Parameters	Test 0	1	2	3	4	5	6	7	8	9
frequency_cutoff	0	0	0	0	0	0	0	0	0	0
batch_size	64	64	64	64	64	64	64	64	64	64
early_stopping	5	5	5	5	5	5	5	5	5	5
learning_rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Casefolding										
Remove_stopwords										
Lemmatization										
Stemming										
Accuracy	90.5	90.21	89.92	89.97	90.12	89.77	89.21	89.42	89.54	88.72
Loss	0.256	0.258	0.261	0.266	0.26	0.263	0.272	0.259	0.269	0.278

Unfortunately, using these pre-processed dataset does not improve the result.

- Bigrams and Trigrams implementation:
  Ngrams are implemented by modifying the *vectorize* and *from_dataframe*.

```
# ngrams
reviews = nltk.word_tokenize(review)
grams = list(' '.join(i) for i in bigrams(reviews)) #bigram
#grams = list(' '.join(i) for i in ngrams(reviews,3)) #trigram
for token in grams:
    if token in grams:
        if token not in string.punctuation:
            one_hot[self.review_vocab.lookup_token(token)] = 1
#"""

return one_hot
```

```
# ngrams
for review in review_df.review:
    reviews = nltk.word_tokenize(review)
    grams = list(' '.join(i) for i in bigrams(reviews)) #bigram
    #grams = list(' '.join(i) for i in ngrams(reviews,3)) #trigram
    for word in grams:
        if word not in string.punctuation:
            word_counts[word] += 1
#"""

for word, count in word_counts.items():
    if count > cutoff:
        review_vocab.add_token(word)

return cls(review_vocab, rating_vocab)
```

- Test on bigram/trigrams with default dataset and default settings

Parameters	Test 0	1	2
frequency_cutoff	25	25	25
batch_size	128	128	128
early_stopping	5	5	5
learning_rate	0.001	0.001	0.001
Bigram			
Trigram			
Accuracy	89.86	90.15	85.59
Loss	0.264	0.253	0.347

Bigram works but trigram doesn't. So, we will do further tests based on bigram.

- Further test based on bigram and previous optimal parameters

Parameters	Test 0	1	2	3	4
frequency_cutoff	0	0	0	0	0
batch_size	64	64	64	64	64
early_stopping	5	5	5	5	5
learning_rate	0.001	0.001	0.001	0.001	0.001
hidden_dim	/	10	20	10	5
hidden_layer	0	1	1	2	1
Bigram					
Accuracy	91.83	92	91.41	91.39	92.17
Loss	0.223	0.222	0.243	0.246	0.219

We bring back the hidden layer into the test and find that one additional layer actually improves the result with a lower hidden_dim.

- **Optimal result**

Parameters	
frequency_cutoff	0
batch_size	64
early_stopping	5
learning_rate	0.001
hidden_dim	5
hidden_layer	1
Bigram	Yes
Accuracy	92.17
Loss	0.219

RNN Model

- **Model Source & Description**

  The model is based on LSTM.

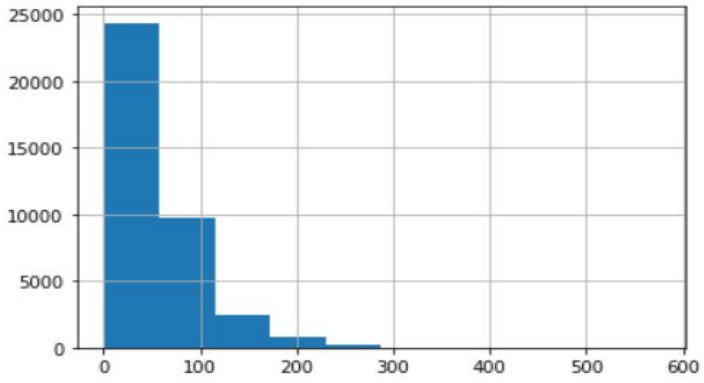
  The link of the model source is provided in the Bibliography 2.

  Analysing review length :

  we have very less number of reviews with length > 500

  So we will consider only those below it

```
rev_len = [len(i) for i in x_train]
pd.Series(rev_len).hist()
plt.show()
pd.Series(rev_len).describe()
```




```
x_train_pad = padding_(x_train,500)
x_test_pad = padding_(x_test,500)
```

- **Testing**

  Firstly, use the same dataset “reviews_with_splits_default”, find the best parameters with the consequence “learning rate” to “drop out rate”.

  Learning rate=0.002 is better can be got by comparing parameter 1,2,3
  The number of iterations=5 is better can be got by comparing parameter 2,4
  Hidden layers=2 is better can be got by comparing parameter 2,5
  Units in each layers=256 is better can be got by comparing parameter 2,6,7
  The number of embedding_dim=32 is better can be got by comparing parameter 2,8,9
  Drop out rate=0.2 is better can be got by comparing parameter 9,10,11

Parameters	default1	2	3	4	5	6	7	8	9	10	11
Learning rate	0.001	0.002	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
The number of iterations (epoch)	5	5	5	6	5	5	5	5	5	5	5
The number of hidden layers	2	2	2	2	3	2	2	2	2	2	2
The number of hidden units in each layer	256	256	256	256	256	210	290	256	256	256	256
The number of embedding_dim	64	64	64	64	64	64	64	48	32	32	32
Regularizations : drop out rate	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.2	0.1
Accuracy	84.528	85.768	85.152	85.616	50.0	85.672	86.032	86.488	86.216	86.568	85.76
Loss	0.812	0.346	0.365	0.865	0.693	0.355	0.377	0.337	0.315	0.310	0.328

Secondly, test datasets with different preprocessing methods with parameters No.10 above. The result is the dataset with case-folding and lemmatization can get the best accuracy and loss.

Datasets	Accuracy	Loss function
reviews_with_splits_default	86.568	0.310
reviews_with_casefolding_without_stopword	84.6	0.345
reviews_with_splits_with_casefolding	86.528	0.328
reviews_with_splits_with_casefolding_lemmatization	87.144	0.303
reviews_with_splits_with_casefolding_stemming	85.176	0.344
reviews_with_splits_with_casefolding_without_stopword_lemmatization	85.896	0.334
reviews_with_splits_with_casefolding_without_stopword_stemming	84.88	0.362
reviews_with_splits_with_lemmatization	86.592	0.314
reviews_with_splits_with_stemming	85.912	0.327
reviews_with_splits_without_stopword	85.464	0.371

- **Optimal result**

Dataset Preprocessing Method	Case-folding and Lemmatization
Parameters	
Learning rate	0.002
The number of iterations (epoch)	5
The number of hidden layers	2
The number of hidden units in each layer	256
The number of embedding_dim	32
Regularizations: drop out rate	0.2
Accuracy	87.144
Loss	0.303

CNN Model

- **Model Source & Description**

  The link of the model source is provided in the Bibliography 3.

- **Testing**

  We use the IMDB dataset to find the best parameters combination among the epoch, number of filters, number of embedding dimensions and drop out ratio.

Paraments	test1	2	3	4	5	6	7	8	9	10	11
The number of iterations (epoch)	5	5	5	5	5	5	5	6	6	5	5
The number of filters	100	100	200	50	100	100	100	100	50	200	100
The number of embedding_dim	100	100	100	100	100	100	50	100	50	200	100
Regularizations: drop out ratio	0.5	0.3	0.3	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.7
Accuracy	88.42	88.63	88.42	88.47	88.69	88.97	49.99	88.79	86.29	88.13	87.96
Loss	0.311	0.314	0.342	0.300	0.322	0.309	0.912	0.343	0.319	0.311	0.301

We build up the CNN model as follow:

```
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, output_dim, dropout):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.conv_0 = nn.Conv2d(in_channels=1, out_channels=n_filters, kernel_size=(filter_sizes[0],embedding_dim))
        self.conv_1 = nn.Conv2d(in_channels=1, out_channels=n_filters, kernel_size=(filter_sizes[1],embedding_dim))
        self.conv_2 = nn.Conv2d(in_channels=1, out_channels=n_filters, kernel_size=(filter_sizes[2],embedding_dim))
        self.fc = nn.Linear(len(filter_sizes)*n_filters, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        x = x.permute(1, 0)
        embedded = self.embedding(x)
        embedded = embedded.unsqueeze(1)
        conved_0 = F.relu(self.conv_0(embedded).squeeze(3))
        conved_1 = F.relu(self.conv_1(embedded).squeeze(3))
        conved_2 = F.relu(self.conv_2(embedded).squeeze(3))
        pooled_0 = F.max_pool1d(conved_0, conved_0.shape[2]).squeeze(2)
        pooled_1 = F.max_pool1d(conved_1, conved_1.shape[2]).squeeze(2)
        pooled_2 = F.max_pool1d(conved_2, conved_2.shape[2]).squeeze(2)
        cat = self.dropout(torch.cat((pooled_0, pooled_1, pooled_2), dim=1))
        return self.fc(cat)
```

- **Optimal result**

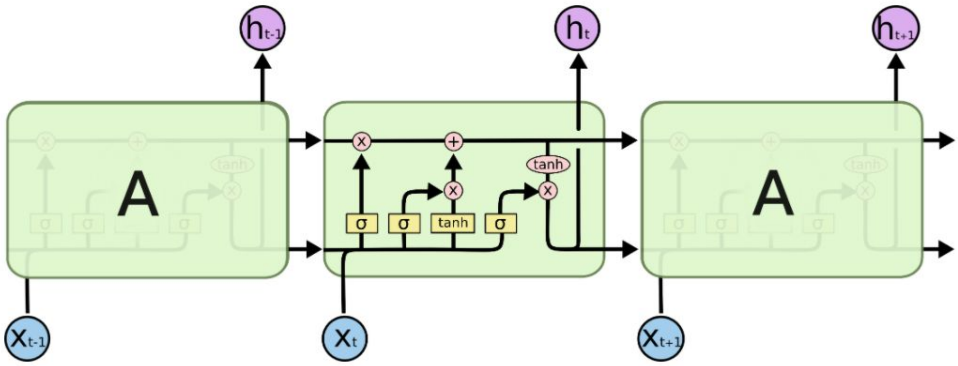
Paraments	
The number of iterations (epoch)	**5**
The number of filters	**100**
The number of embedding_dim	**100**
Regularizations: drop out rate	**0.1**
Accuracy	**88.97**
Loss	**0.309**

Compare three models

A single neural can be imagined as a logistic regression, MNN(Multiple layer Neural Networks) is a group of multiple neurals at each layer, including input layer, hidden layer, output layer, and there may be several hidden layers. MNN can be used to solve problems related to: Tabular data, Image data, Text data.

RNN(Recurrent neural networks) are networks that are designed to interpret temporal or sequential information. RNNs use other data points in a sequence to make better predictions. They do this by taking in input and reusing the activations of previous nodes or later nodes in the sequence to influence the output.

LSTM(Long Short Term Memory networks) are a special kind of RNN, capable of learning long-term dependencies.



The repeating module in an LSTM contains four interacting layers – Image via colah.github.io

CNN(Convolutional neural networks) are one of the most common types of neural networks used in computer vision to recognize objects and patterns in images. One of their defining traits is the use of filters within convolutional layers.

Conclusion

As we can see from above, multi-layer neural networks is the best model among MNN, CNN and RNN models. The reason why is because MNN is capable of learning any nonlinear function and has the ability to work with incomplete knowledge. If we have a very large dataset, MNN will have the bigger fault tolerance while CNN lacks the ability to be spatially invariant to the input data and RNN can not process very long sequences under some conditions.

Bibliography

1. Data Source: IMDB Dataset of 50k movie reviews https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
2. LSTM Model Source https://www.kaggle.com/arunmohan003/sentiment-analysis-using-lstm-pytorch
3. CNN Model Source https://github.com/viritaromero/Sentiment-Analysis-using-a-CNN/blob/master/Sentiment_Analysis_CNN.ipynb
4. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
5. https://lionbridge.ai/articles/difference-between-cnn-and-rnn/
6. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/