Homework 4

Yixing Chen

1. General Information

1.1 Purpose

The purpose of this assignment for me is to explore text classification problem. The detailed target is to find if a review of a movie is positive or negative, e.g. If the review is “This movie is fantastic! I really like it because it is so good!”, this review should be judged as a positive review. This report explains how this type of text classification is realized.

1.2 Dataset

The dataset I use for this assignment is IMDB movie reviews. This dataset is always used for sentiment analysis beginners to start deep learning in the past. Sentiment analysis is also a part of text classification. Hereby I use this dataset to learn this method, too. IMDB is a movie information website. Users could present their thoughts and reviews for the movie they watched. The dataset is downloaded from http://ai.stanford.edu/~amaas/data/sentiment/. All the reviews of this dataset are binary sentiment classification. Pre-processing is implemented to the dataset. The text reviews are labelled as 1 or 0 for positive or negative judgment, respectively. After pre-processed, the reviews are saved in the file *movie_data.csv*. Each review has a corresponding binary label.

Semi supervised is implemented in this assignment, thus, an unlabeled text corpus is used, whose size is more than 5x the size of labeled text corpus. Totally, the labeled movie review number is 8000 while the unlabeled movie review number is 42000.

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import json
import pickle
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn import metrics, model_selection, ensemble, preprocessing

data_labeled = pd.DataFrame()
data_labeled = pd.read_excel('movie_data.xlsx', encoding='utf-8')
```

```
data_unlabeled = pd.read_excel('movie_data_unlabeled.xlsx')
new_X = []
X = data_labeled.drop('sentiment', axis = 1)
for value in X['review']:
    new_X.append(value)
y = data_labeled['sentiment']
X_unlabeled = data_unlabeled
for value in X_unlabeled['review']:
    new_X.append(value)
#yy = data_unlabeled['class']
huge_X = pd.DataFrame(new_X, columns = ['review'])
```

This code imports labeled movie review dataset and unlabeled movie review dataset from *movie_data.xlsx* and *movie_data_unlabeled.xlsx*, respectively. For labeled dataset, it has two columns. One is 'sentiment', and the other is 'review'. For unlabeled dataset, it has only one column, which is 'review'. The 'review' columns of both labeled and unlabeled dataset are concatenated together as well.

2. Main Architecture

In this assignment, text classification problem is explored in the following steps:

2.1 Word Embedding

Word Embedding used in this assignment is generated. All reviews need to be encoded as a unique representation. The text corpus of the reviews is vectorized after turning each text into a sequence of integers. The Python package used here is Keras. The following code is used to realize this function. The code provides each English word a corresponding integer, e.g. 1 represents 'the'. 3 represents 'and'. 806 represents 'check', etc. Keras includes 125,601 English words and their integers. Thus, all IMDB reviews are transformed into tokens after punctuation and space are removed. After all the words are transformed into integers, every review is converted to a vector of integers. All the vectors are defined to have the same length. The length is *(max_length)*, which is the length of the text which has most words. In this assignment, this value equals 2678. For the vector whose length is less than 2678, '0's are supplemented before it.

```
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
```

```
tokenizer_object = Tokenizer()
total_reviews = X_train + X_test
tokenizer_object.fit_on_texts(total_reviews)

# pad sequences
max_length = max([len(s.split()) for s in total_reviews])
#max_length=500

# define vocabulary size
vocab_size = len(tokenizer_object.word_index) + 1

X_train_tokens =    tokenizer_object.texts_to_sequences(X_train)
X_test_tokens = tokenizer_object.texts_to_sequences(X_test)


X_train_pad = pad_sequences(X_train_tokens, maxlen=max_length, padding='pre')
X_test_pad = pad_sequences(X_test_tokens, maxlen=max_length, padding='pre')
```

The vocabulary size *(vocab_size)* of network input is the number of all vocabulary in Keras plus 1. The dimension of embedding layer *(embedding_dim)* is set to 100.

2.2 Building Model

The neural network model used in this assignment is sequential. This means this model passes a list of layer instances to the constructor. The first layer of this model is to receive information about its input shape. Hereby the first hidden layer of this neural network is embedding layer. The input length is *(max_length).* The embedding dimension is set to 10. The shape of output of the model is (None, 10, 64), where None is the embedding dimension. The second layer is LSTM layer, which is different from the model of homewor1. To make this report concise, the main principle of LSTM is not given here. The second hidden layer contain 32 units of nodes. The third layer is a dense layer, which contains one unit here. The initial weights of the network are randomly set. The weights will be updated with the network training process. The number of parameters of the model trained is 12,573,001. Among these parameters, 12,560,200 are for embedding layer, which is the result of *vocab_size* multiplied by *embedding_dim*. 12,768 are for LSTM network's second later. 33 are for the merged one dense node. The activation function used in this network is sigmoid function.

```
Summary of the built model...
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 2678, 10)          1256020
_________________________________________________________________
lstm_7 (LSTM)                (None, 32)                5504
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,261,557
Trainable params: 1,261,557
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(LSTM(units=32,  dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

2.3 Training Model

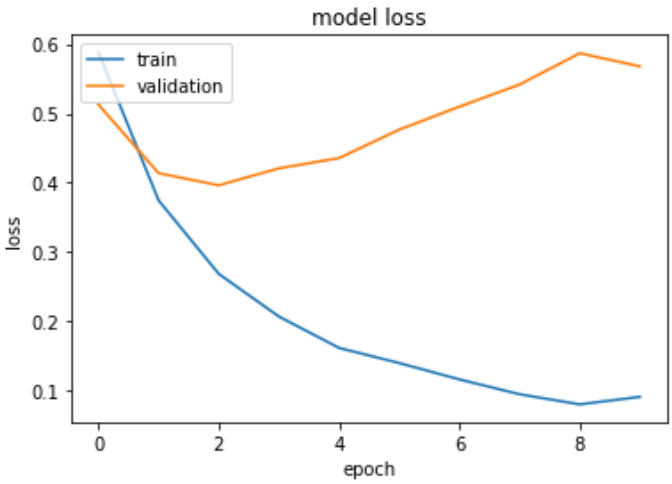
The neural network model is trained with the 8000 labeled movie reviews mentioned beforehand. The training method of this model is *Adam*, which is a stochastic optimization algorithm. The details of this algorithm could be found in the paper https://arxiv.org/abs/1412.6980v8. The batch size is set to 128. The training epoch is set to 10. The learning rate is the default value, 0.001. The loss function is binary cross entropy, which means the error between real value and estimated value is 0 or 1.

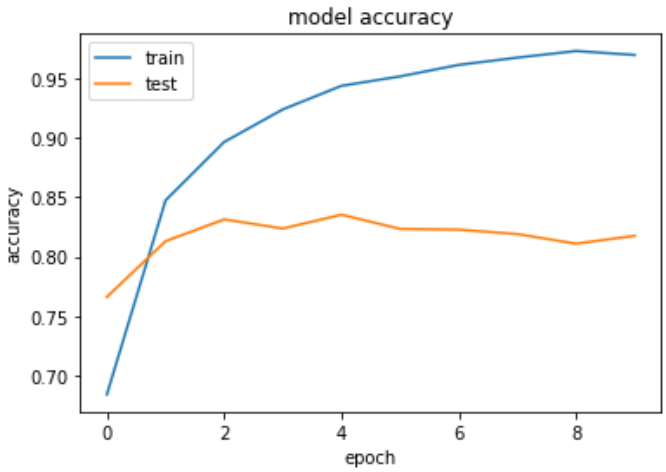
```
# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())

print('Train...')

history = model.fit(X_train_pad, y_train, batch_size=128, epochs=10,
validation_data=(X_test_pad, y_test), verbose=2)
```

```
print(history.history.keys())
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

The training process is shown as the following:

```
Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 606s - loss: 0.5881 - acc: 0.6838 - val_loss: 0.5127 - val_acc: 0.7660
Epoch 2/10
 - 595s - loss: 0.3744 - acc: 0.8474 - val_loss: 0.4136 - val_acc: 0.8130
Epoch 3/10
 - 597s - loss: 0.2683 - acc: 0.8966 - val_loss: 0.3963 - val_acc: 0.8314
Epoch 4/10
 - 599s - loss: 0.2067 - acc: 0.9241 - val_loss: 0.4207 - val_acc: 0.8237
Epoch 5/10
 - 602s - loss: 0.1614 - acc: 0.9440 - val_loss: 0.4355 - val_acc: 0.8353
Epoch 6/10
 - 604s - loss: 0.1396 - acc: 0.9519 - val_loss: 0.4766 - val_acc: 0.8234
Epoch 7/10
 - 597s - loss: 0.1161 - acc: 0.9616 - val_loss: 0.5098 - val_acc: 0.8227
Epoch 8/10
 - 564s - loss: 0.0946 - acc: 0.9678 - val_loss: 0.5416 - val_acc: 0.8190
Epoch 9/10
 - 553s - loss: 0.0800 - acc: 0.9733 - val_loss: 0.5869 - val_acc: 0.8109
Epoch 10/10
 - 553s - loss: 0.0908 - acc: 0.9700 - val_loss: 0.5679 - val_acc: 0.8175
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```





To validate the model, 25,000 test reviews are utilized. The batch size is 128. The metrics to be evaluated by the model during training and testing is accuracy. The accuracy is 81.75% after 10 epochs. Compared to the accuracy of GRU network in homework1, which was 79.74%, the accuracy of LSTM is slightly higher.

```
print('Testing...')
score, accuracy = model.evaluate(X_test_pad, y_test, batch_size=128)

print('Test score:', score)
print('Test accuracy:', accuracy)

print("Accuracy: {0:.2%}".format(accuracy))
```

The result is shown as the following:

```
Testing...
25000/25000 [==============================] - 64s 3ms/step
Test score: 0.5679277250671386
Test accuracy: 0.8175199999809265
Accuracy: 81.75%
```

2.4 Semi supervised learning

Semi-supervised learning algorithms use not only the labeled data but also unlabeled data to construct a classifier. The goal of semi-supervised learning is to use unlabeled instances and combine the information in the unlabeled data with the explicit classification information of labeled data for improving the classification performance. In this assignment, a basic self-training method is used to combine RNN model.

Generally, self-training trains a model mm on a labeled training set L and an unlabeled data set U. At each iteration, the model provides predictions $m(x)$ in the form of a probability distribution over the C classes for all unlabeled examples x in U. If the probability assigned to the most likely class is higher than a predetermined threshold τ, x is added to the labeled examples with p(x)=argmax m(x) as pseudo-label. This process is generally repeated for a fixed number of iterations or until no more predictions on unlabeled examples are confident. The pseudocode of self-training method is:

1:repeat
2: $m \leftarrow train\_model(L)$
3: for $x \in U$ do
4: if max $m(x) > \tau$ then
5: $L \leftarrow L \cup \{(x,p(x))\}$
6:until no more predictions are confident

The corresponding code of this method is as follows, and here the threshold is set to 0.5. This could be explained by [1]. Here the model of semi supervised method also uses LSTM with the same parameters mentioned in 2.2 and 2.3.

```
tao=0.5
b=1
predictions = []
while b!=0|(len(y_train))<=41996:
    model = pickle.load(open('Model_save.pkl', 'rb'))
    b = 0
    yy_predict = model.predict(X_unlabeled_pad)
    yy_predictt = yy_predict.tolist()
    predictions.append(yy_predictt)
    probability=model.predict_proba(X_unlabeled_pad)
    for i in range(len(probability)):
        if probability[i][0]>tao:
            b=b+1
            X_train_pad =
np.concatenate((X_train_pad,X_unlabeled_pad[i].copy()[np.newaxis,:]),axis=0)
            if predictions[0][i][0]>0.5:
```

```
                y_train= np.append(y_train,1)
            else:
                y_train=np.append(y_train,0)

    history = model.fit(X_train_pad, y_train, batch_size=128, epochs=2, validation_data=(X_test_pad, y_test), verbose=2)
    with open(filename, 'wb') as file:
        pickle.dump(model, file)

score, acc = model.evaluate(X_test_pad, y_test, batch_size=128)

print('Test score:', score)
print('Test accuracy:', acc)

print("Accuracy of self training: {0:.2%}".format(acc))
```

The result is shown as the following:

```
1598/1598 [==============================] - 2s 1ms/step
Test score: 0.7134618336178633
Test accuracy: 0.7959949938913758
Accuracy of self training: 79.60%
```

2.4 Comparison

In the previous assignment, the accuracy of prediction with LSTM is 81.75%. This result is also shown in this report.

With unlabeled dataset and semi supervised method, e.g. self-training method. The accuracy is 79.60%. The accuracy is slightly lower than the previous assignment. The reason of this could possibly be that the unlabeled movie reviews modifies the parameters of models, and makes the accuracy fluctuates. If the unlabeled movie review number increases, the accuracy might be slightly improved. This result cannot definitely determine if self-training method has a better or worse performance than methods used in previous assignments.

Reference

[1] Caruana, Rich, and Alexandru Niculescu-Mizil. "An empirical comparison of supervised learning algorithms." *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006.