HW4 - Sentiment Analysis 2

0853412 資管碩一 吳宛儒

資料前處理

a. 讀取資料

```
def load_data(dataset):
    if dataset == 'train':
        data = pd.read_csv('training_label.txt',sep='\t',header=None)
    elif dataset == 'test':
        data = pd.read_csv('testing_label.txt',sep='\t',header=None)
    data.columns = ['text']
    data['label'] = data['text'].str[0]
    data['data'] = data['text'].str[10:]
    del data['text']
    return data

#%%
# load data
train = load_data('train')
train = train[:][:10000]
test = load_data('test')

def split_data(dataset):
    X, y = [], []
    X = dataset['data'].values
    y = dataset['label'].values
    return X, y

trainX, trainy, testX, testy = [], [], [], []
trainX, trainy = split_data(train)
testX, testy = split_data(test)
```

如同 HW3 所做的，將資料 load 進來並切成 train 和 test dataset。其中 train 取前 10000 筆。

b. 去除停頓詞

原本使用 nltk 或自定義 stop words，但發現 tf-idf 便能在裡面設定，因此這個步驟連同下個步驟一起做。

c. 文字轉向量

此處使用 tf-idf，stop_words 也宜並在裡面使用 english 的設定。

```
# tf-idf
max_features = 5000
vectorizer = TfidfVectorizer(max_features=max_features,stop_words='english')
ldf_train = vectorizer.fit_transform(trainX)
ldf_test = vectorizer.transform(testX)
tfidf_feature = vectorizer.get_feature_names()
```

原本會有 13242 個 features，但因為跑起來會有 memory 或 OS error，因此將 features 數量最大設置為 5000 做為上限。

建模

a. RNN & LSTM

RNN：

網路架構：

```
# RNN
modelRNN = Sequential()
modelRNN.add(Embedding(output_dim=128,
                       input_dim=5000,
                       input_length=5000))
modelRNN.add(Dropout(0.7))
modelRNN.add(SimpleRNN(units=16))
modelRNN.add(Dense(units=128,activation='relu'))
modelRNN.add(Dropout(0.35))
modelRNN.add(Dense(units=1,activation='sigmoid'))
modelRNN.summary()
modelRNN.compile(loss='binary_crossentropy',
      optimizer='adam',
      metrics=['accuracy'])
```

此部分參考老師上課的簡報，並將 input_dim 和 input_length 設為剛才 tf-idf 出來的維度。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 5000, 128)         640000
_________________________________________________________________
dropout_1 (Dropout)          (None, 5000, 128)         0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 16)                2320
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2176
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 644,625
Trainable params: 644,625
Non-trainable params: 0
```

設定驗證集比例：0.2

```
Train on 8000 samples, validate on 2000 samples
```

Fit model：

```
# fit
train_history = modelRNN.fit(ldf_train,trainy,
          epochs=7,
          batch_size=32,
          verbose=1,
          validation_split=0.2)
```

訓練過程：

```
Epoch 1/7
8000/8000 [==============================] - 303s 38ms/step - loss: 0.6948 - acc: 0.4955 -
val_loss: 0.6932 - val_acc: 0.4980
Epoch 2/7
8000/8000 [==============================] - 312s 39ms/step - loss: 0.6948 - acc: 0.4977 -
val_loss: 0.6934 - val_acc: 0.4980
Epoch 3/7
8000/8000 [==============================] - 323s 40ms/step - loss: 0.6940 - acc: 0.4994 -
val_loss: 0.6931 - val_acc: 0.5020
Epoch 4/7
8000/8000 [==============================] - 329s 41ms/step - loss: 0.6937 - acc: 0.5092 -
val_loss: 0.6931 - val_acc: 0.5020
Epoch 5/7
8000/8000 [==============================] - 338s 42ms/step - loss: 0.6938 - acc: 0.5020 -
val_loss: 0.6932 - val_acc: 0.5020
Epoch 6/7
8000/8000 [==============================] - 364s 46ms/step - loss: 0.6937 - acc: 0.5030 -
val_loss: 0.6933 - val_acc: 0.5020
Epoch 7/7
8000/8000 [==============================] - 365s 46ms/step - loss: 0.6939 - acc: 0.5026 -
val_loss: 0.6932 - val_acc: 0.4980
```

*由於跑十次 epoch 的時候每次跑到第七次就會跑出 OS error(不太確定是否就是 memory error，而且用實驗室 server 跑也是一樣)因此，只讓他跑了七次。

*同上述的原因，也將 tf-idf 取出來的特徵取 5000 個而非原本的全取(全取=13242 個，tf-idf 做完會是 10000*13242 的 sparse matrix)。

LSTM：

網路架構：

```
# LSTM

modelLSTM = Sequential()
modelLSTM.add(Embedding(output_dim=64,
                        input_dim=5000,
                        input_length=5000))
modelLSTM.add(Dropout(0.7))
modelLSTM.add(LSTM(32))
modelLSTM.add(Dense(units=32,activation='relu'))
modelLSTM.add(Dropout(0.5))
modelLSTM.add(Dense(units=1,activation='sigmoid'))
modelLSTM.summary()
modelLSTM.compile(loss='binary_crossentropy',
      optimizer='adam',
      metrics=['accuracy'])
```

此部分參考老師上課的簡報，並將 input_dim 和 input_length 設為剛才 tf-idf 出來的維度。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 5000, 64)          320000
_________________________________________________________________
dropout_3 (Dropout)          (None, 5000, 64)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                12416
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 33
=================================================================
Total params: 333,505
Trainable params: 333,505
Non-trainable params: 0
```

設定驗證集比例：0.2

```
Train on 6400 samples, validate on 1600 samples
```

LSTM 的部分因為發現訓練速度很慢的關係，只取了 8000 筆 train 資料來做訓練。

Fit model：

```
# fit
train_history = modelLSTM.fit(ldf_train,trainy,
          epochs=7,
          batch_size=128,
          verbose=1,
          validation_split=0.2)
```

訓練過程：

```
Epoch 1/3
6400/6400 [==============================] - 447s 70ms/step - loss: 0.6935 - acc:
0.5034 - val_loss: 0.6933 - val_acc: 0.5012
Epoch 2/3
6400/6400 [==============================] - 354s 55ms/step - loss: 0.6933 - acc:
0.5022 - val_loss: 0.6932 - val_acc: 0.5012
Epoch 3/3
6400/6400 [==============================] - 376s 59ms/step - loss: 0.6933 - acc:
0.5038 - val_loss: 0.6932 - val_acc: 0.5012
```

*由於跑十次 epoch 的時候每次跑到第七次就會跑出 OS error(不太確定是否就是 memory

error，而且用實驗室 server 跑也是一樣)因此，在 LSTM 讓他跑了三次，並且發現與跑七次效果差不多。

*同上述原因，也將 tf-idf 取出來的特徵取 5000 個而非原本的全取(全取=13242 個，tf-idf 做完會是 10000*13242 的 sparse matrix)。

*同上述原因，原本取了 10000 筆 train 資料這次也只先取了 8000 筆。

b. Dropout

如同 a.所述，RNN 和 LSTM 都使用了兩層 Dropout Layer，並設置 0.7 以及 0.5 或 0.35 來跑。

下一個 section 會針對有無 Dropout 來做比較。

c. Plot accuracy and loss

history 裡面存了哪些資料：

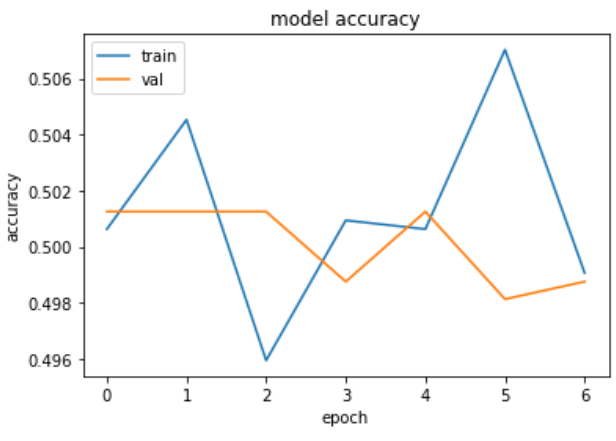
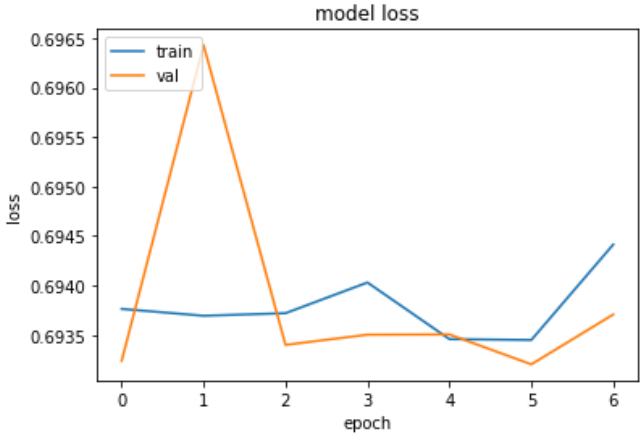
```
In [7]: print(train_history.history.keys())
   ...:
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

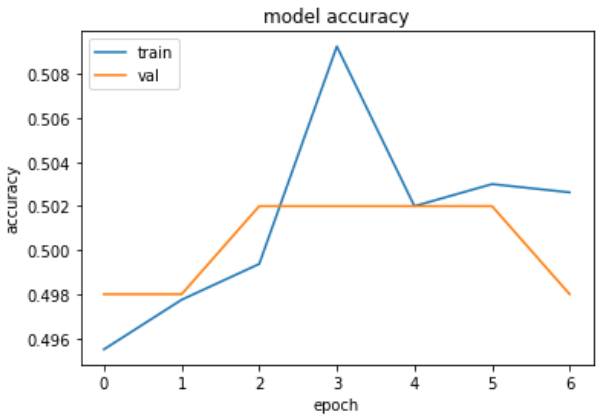
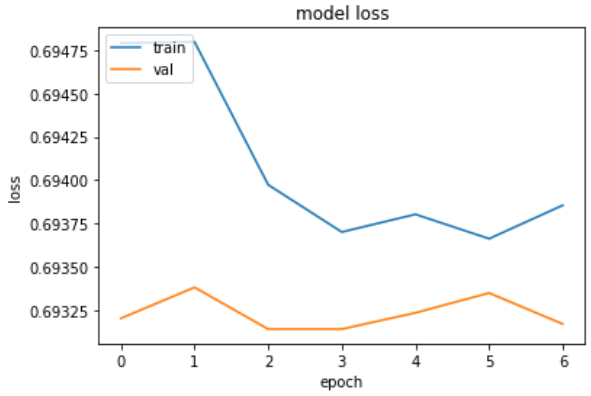
```
def Plot(history):
    # summarize history for accuracy
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()

Plot(train_history)
```

RNN：

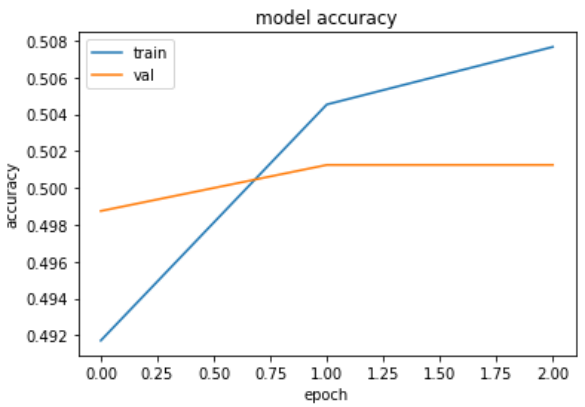
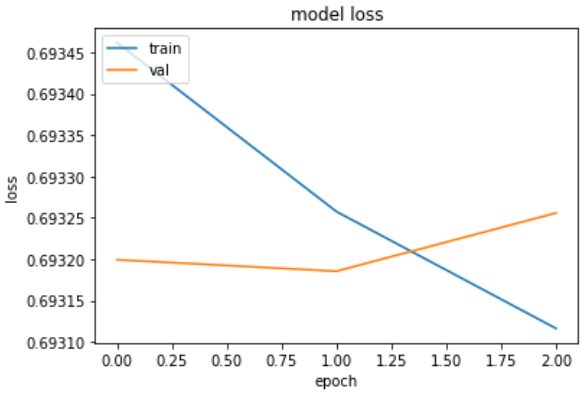
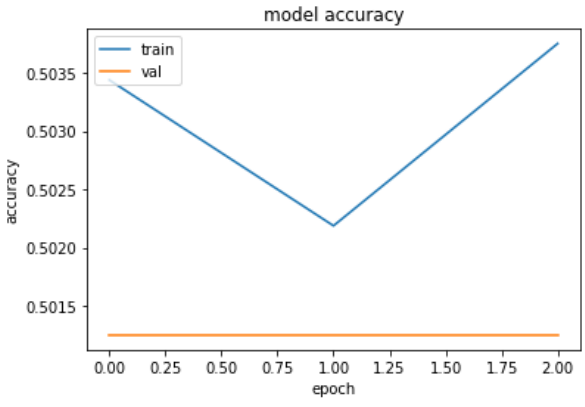
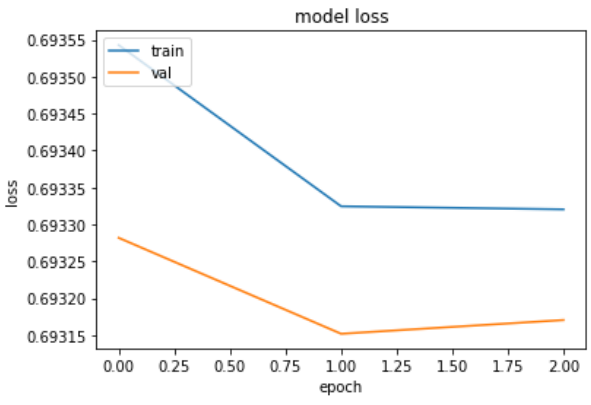
Epoch=7

Without Dropout	
Accuracy	Loss
	

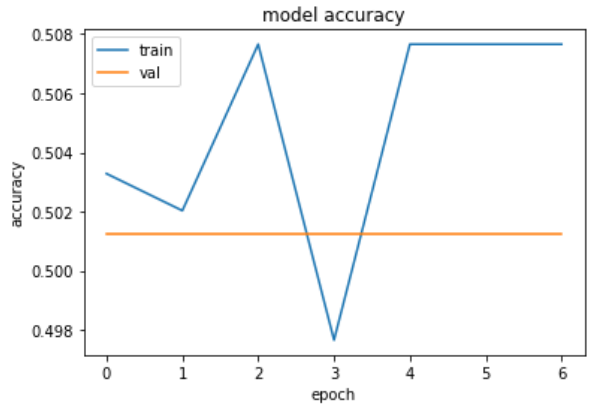
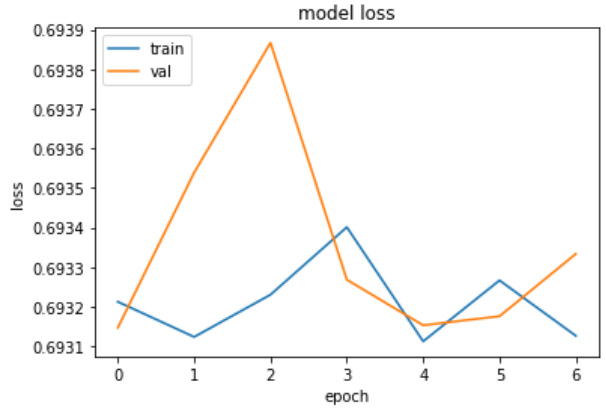
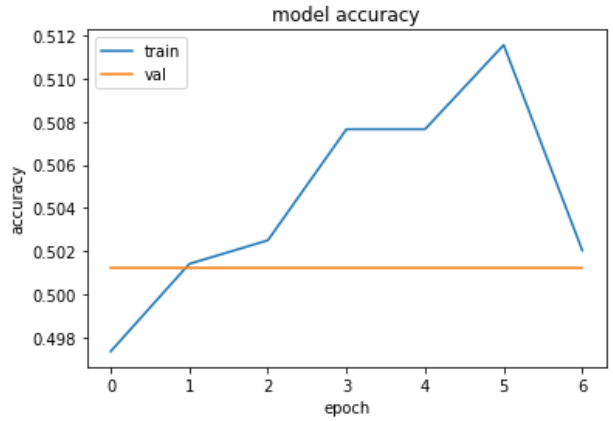
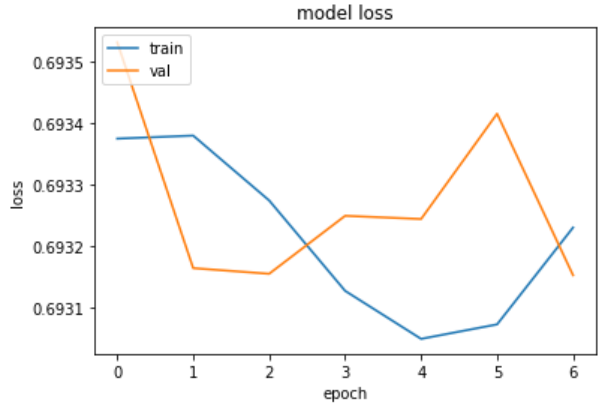
With Dropout	
Accuracy	Loss
	

LSTM：

Epoch=3

Without Dropout	
Accuracy	Loss
	
With Dropout	
Accuracy	Loss
	

Epoch=7

Without Dropout	
Accuracy	Loss
	
With Dropout	
Accuracy	**Loss**
	

這邊不太確定是不是因為 epoch 數量跑得不夠，在 accuracy 和 loss 上都呈現浮動很大的現象。且有 Dropout 也不能確定會比沒有 Dropout 時表現還要好，這可能就是沒有跑足夠多次的結果。loss 的部分有下降的趨勢，但在幾次 validation 的部分有微微上升，因為並沒有參數過多的問題，因此推斷可能是 model 本身參數以及 embedding 不足以代表文字的情緒分析。

評估模型

a. 測試並計算 accuracy

RNN：

```
77 # evaluate
78 scores = modelRNN.evaluate(ldf_test, testy,verbose=1)
79 scores[1]
```

Without Dropout	In [11]: scores = modelRNN.evaluate(ldf_test, testy,verbose=1) ...: scores[1] 90/90 [==============================] - 1s 9ms/step Out[11]: 0.5888888968361748 此為 test 上的 accuracy Predict 結果皆為[0.51543665]，為 label=1
With Dropout	90/90 [==============================] - 1s 14ms/step Out[6]: 0.41111111177338494 此為 test 上的 accuracy Predict 結果都分為 label=0

LSTM：

```
133 # evaluate
134 scores = modelLSTM.evaluate(ldf_test, testy,verbose=1)
135 scores[1]
```

Without Dropout	In [13]: scores = modelLSTM.evaluate(ldf_test, testy,verbose=1) ...: scores[1] 90/90 [==============================] - 2s 20ms/step Out[13]: 0.41111111177338494 此為 test 上的 accuracy [0.49127826] 此為全部 predict 的結果，皆為 label=0
With Dropout	In [9]: scores = modelLSTM.evaluate(ldf_test, testy,verbose=1) ...: scores[1] 90/90 [==============================] - 2s 20ms/step Out[9]: 0.41111111177338494 此為 test 上的 accuracy [0.49510723] 此為全部 predict 的結果，皆為 label=0

在 predict 的部分也很奇怪，除了 RNN without Dropout，其他的 evaluate 的結果都一樣，且 90 筆資料預測出來的值也是一樣，也都被分類為 label=0。這樣的結果在 epoch 為 7 以內都會發生，epoch7 次以上則會出現 memory or OS error，因此沒有實驗到。