Kaggle Report 108064532

- ## Result/Method

1. I use **(word embedding + w2v + LSTM)** and **(Bag of Word)**
2. And the result of **(word embedding + w2v + LSTM)** is better than **(Bag of Word)**

- ## Insights and Observation

1. I found the choice of optimizer is quite important, the best suitable optimizer have strong impact on my training process (my optimizer is adam).
2. I also found if I lower the batch size, the result would be better.
3. The emotions are imbalanced, but it's not suitable to use sampling. Because it's not going to gain better result.
4. Loss function is correlated with the last layer.
   - I found if using binary_crossentropy, the last layer should use sigmoid
   - If using categorical_crossentropy, the last layer should use softmax

- ## Problems I Encountered

1. I encountered underfitting at first, so my solutions are to make more epoch, make my model more complex or modify the input.
2. It also means my model is to simple, so I adjust layer of my model and the units in hidden layer.
3. And it also means my input is not helping the model to distinguish different emotions.
4. Another problem is overfitting, if didn't add dropout layer, it would may go into overfitting. So I tried the solutions below:
   - To add drop layer, prevent model to train to well
   - Change learning rate, so I decrease the learning rate

- Preprocessing

1. Read the raw twitter dataset

   - Merge the data set to a pandas data frame and also split the data set into training/testing data set

2. Clean the Data

   - Remove duplicate'#
   - Count the occurrence of words, and trys to see if any useful or not helpful
   - Remove tag <LH>, because it's noise
   - Remove stop words
   - Remove conjunctions, such as 'the', 'of'……
   - Remove meaningless punctuation
   - Exclude the words not both exist in training and testing datasets

- Feature Engineering

1. Use Bag of Word

   - Represent sentences with word counts

2. Use Word2vec

   - Represent sentences with sum of the word vectors

3. Use Word2vec and WordEmbedding

   - Each word in the sentence will be transformed into integer, and the integer will be used to find corresponding word vector

- Model

1. Below is the structure of my model:

- Embedding layer to receive input data
- A LSTM layer with shape 256
- And then one dropout layer with shape 512
- Two Dense layer with shape 8 and output the result

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 20, 50)	6950000
lstm_1 (LSTM)	(None, 256)	314368
dense_1 (Dense)	(None, 512)	131584
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 8)	4104
dense_3 (Dense)	(None, 8)	72

```
Total params: 7,400,128
Trainable params: 450,128
Non-trainable params: 6,950,000
```

- Code on jupyter (I will explain code in below snapshot)

In [1]:

```
import pandas as pd
from gensim.models import Word2Vec

# form embedding matrix (w2v)
import numpy as np
from tqdm import tqdm

# padding for word embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import keras
from sklearn.preprocessing import LabelEncoder

word2vec_model = Word2Vec.load("word2vec_twitter_50.model")

train_df = pd.read_pickle("./train_df_clean.pkl")
test_df = pd.read_pickle("./test_df_clean.pkl")



MAX_SEQUENCE_LENGTH = 20

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_df['clean_text'])
sequences = tokenizer.texts_to_sequences(train_df['clean_text'])

word_index = tokenizer.word_index

wordEmbedding_w2v_train  = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)


vector_dim = 50
embedding_matrix = np.zeros((len(word_index) + 1, vector_dim))
missingWord = []
for word, i in tqdm(word_index.items()):
    try:
        embedding_vector = word2vec_model.wv[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    except:
        missingWord.append(word)

#modeling
# for a classificaiton problem, you need to provide both training & testing dat
div = int(train_df.shape[0]*0.8)

wordEmbedding_w2v_X_train = wordEmbedding_w2v_train[:div]
wordEmbedding_w2v_y_train = train_df['emotion'][:div]

wordEmbedding_w2v_X_test = wordEmbedding_w2v_train[div:]
wordEmbedding_w2v_y_test = train_df['emotion'][div:]

def label_encode(le, labels):
    enc = le.transform(labels)
    return keras.utils.to_categorical(enc)

def label_decode(le, one_hot_label):
    dec = np.argmax(one_hot_label, axis=1)
```

```
    return le.inverse_transform(dec)

def encode(y_train, y_test):
    label_encoder = LabelEncoder()
    label_encoder.fit(y_train)
    print('check label: ', label_encoder.classes_)
    print('\n## Before convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    re_y_train = label_encode(label_encoder, y_train)
    re_y_test = label_encode(label_encoder, y_test)

    print('\n\n## After convert')
    print('y_train[0:4]:\n', y_train[0:4])
    print('\ny_train.shape: ', y_train.shape)
    print('y_test.shape: ', y_test.shape)

    return re_y_train, re_y_test, label_encoder

le_wordEmbedding_w2v_y_train, le_wordEmbedding_w2v_y_test, label_encoder = enco

# I/O check
input_shape = wordEmbedding_w2v_X_train.shape[1]
print('input_shape: ', input_shape)

output_shape = len(label_encoder.classes_)
print('output_shape: ', output_shape)

from keras.models import Sequential
from keras.layers import Dense, LSTM, GRU, Dropout, Activation, ActivityRegular
from keras.layers.embeddings import Embedding
from keras.regularizers import l2
from keras.initializers import Constant
from keras import optimizers

model = Sequential()
model.add(Embedding(len(word_index) + 1,
                            vector_dim,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False))
# model.add(SpatialDropout1D(0.7))
model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))#0.2
model.add(Dense(8, activation='softmax'))
model.add(Dense(8, activation='softmax'))

# adam = optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999)
model.compile(
    optimizer="nadam",
    loss='categorical_crossentropy',
    metrics=['acc'])

model.summary()

epochs = 5
batch_size = 32
history = model.fit(wordEmbedding_w2v_X_train, le_wordEmbedding_w2v_y_train,
```

```
                    epochs=epochs, batch_size=batch_size, verbose=1,
                    validation_data = (wordEmbedding_w2v_X_test, le_wordEmbeddi

## precision, recall, f1-score,
from sklearn.metrics import classification_report

y_pred_result = model.predict(x = wordEmbedding_w2v_X_test, batch_size=128)
y_pred_result = label_decode(label_encoder, y_pred_result)
print(classification_report(y_true=wordEmbedding_w2v_y_test, y_pred=y_pred_resu
```

```
Using TensorFlow backend.
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:523: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:524: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:526: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:527: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/hsnl-iot/DataMining_2019/VENV/DataMining/lib/python3.5/site-pack
ages/tensorflow/python/framework/dtypes.py:532: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future v
ersion of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
100%|██████████| 138999/138999 [00:00<00:00, 241942.83it/s]

check label:  ['anger' 'anticipation' 'disgust' 'fear' 'joy' 'sadness'
'surprise'
 'trust']

## Before convert
y_train[0:4]:
 0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)


## After convert
y_train[0:4]:
```

```
  0    anticipation
1         sadness
3            fear
5             joy
Name: emotion, dtype: object

y_train.shape:  (1164450,)
y_test.shape:  (291113,)
input_shape:  20
output_shape:  8
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 50)            6950000
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               314368
_________________________________________________________________
dense_1 (Dense)              (None, 512)               131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 4104
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 72
=================================================================
Total params: 7,400,128
Trainable params: 450,128
Non-trainable params: 6,950,000
_________________________________________________________________
Train on 1164450 samples, validate on 291113 samples
Epoch 1/5
1164450/1164450 [==============================] - 842s 723us/step - l
oss: 1.3942 - acc: 0.4976 - val_loss: 1.3239 - val_acc: 0.5259
Epoch 2/5
1164450/1164450 [==============================] - 846s 726us/step - l
oss: 1.3554 - acc: 0.5123 - val_loss: 1.3132 - val_acc: 0.5307
Epoch 3/5
1164450/1164450 [==============================] - 847s 727us/step - l
oss: 1.3524 - acc: 0.5137 - val_loss: 1.3099 - val_acc: 0.5297
Epoch 4/5
1164450/1164450 [==============================] - 844s 724us/step - l
oss: 1.3486 - acc: 0.5150 - val_loss: 1.3099 - val_acc: 0.5322
Epoch 5/5
1164450/1164450 [==============================] - 846s 727us/step - l
oss: 1.3483 - acc: 0.5153 - val_loss: 1.3010 - val_acc: 0.5335
              precision    recall  f1-score   support

       anger       0.68      0.16      0.26      8003
anticipation       0.66      0.49      0.56     49901
     disgust       0.44      0.38      0.41     27449
        fear       0.70      0.33      0.45     12844
         joy       0.52      0.83      0.64    103127
     sadness       0.45      0.47      0.46     38638
    surprise       0.72      0.14      0.24      9816
       trust       0.61      0.23      0.34     41335

    accuracy                           0.53    291113
   macro avg       0.60      0.38      0.42    291113
weighted avg       0.56      0.53      0.51    291113
```

In [2]:

```
## predict
# tests = pd.read_pickle("test_df_clean.pkl")

test_sequences = tokenizer.texts_to_sequences(test_df["clean_text"])
test_sequences  = pad_sequences(test_sequences, maxlen=MAX_SEQUENCE_LENGTH)

pred_result = model.predict(test_sequences, batch_size=128)
pred_result = np.array(label_decode(label_encoder, pred_result))
print(pred_result.shape)
test_df['emotion'] = pred_result
test_df = test_df.drop('hashtags', axis=1)
test_df = test_df.drop('text', axis=1)
test_df = test_df.drop('identification', axis=1)
test_df = test_df.drop('clean_text', axis=1)
test_df = test_df.drop('tokenized', axis=1)
test_df.rename(columns={'tweet_id':'id'}, inplace=True)

test_df.to_csv('prediction.csv', index=False)
```

```
(411972,)
```