Name: 葉偉良
Student ID: 108065511
Kaggle Name: Kenneth Yap
Competition: DM19 Lab2 Kaggle Competition (11/30-12/22)
Position: 22nd with accuracy 0.45771 (Public) & 20th with accuracy 0.46020(Private)

INTRODUCTION

This is a report that introduce my work on the DM19 Lab2 Kaggle Competition.. For convenience, I would not show all the code here, and will state all of the main step I have done to achieve the greatest accuracy I got in the competition.

Let's go straight to the result, the final model I have chosen is a deep learning model called Convolutional Neural Network, the main feature engineering technique is word2vec embedding.

METHOD

1.The very first step is to separate raw dataset into training and testing datasets. Those steps are:

a. Read the raw data.

b. Split the column '_source' into several columns.

c. Separate the training and testing datasets.

d. Check if there is any null or duplicated data in both training and testing datasets.

Before 1.b step:

	_score	_index	_source	_crawldate	_type
0	391	hashtag_tweets	{'tweet': {'hashtags': ['Snapchat'], 'tweet_id...	2015-05-23 11:42:47	tweets
1	433	hashtag_tweets	{'tweet': {'hashtags': ['freepress', 'TrumpLeg...	2016-01-28 04:52:09	tweets
2	232	hashtag_tweets	{'tweet': {'hashtags': ['bibleverse'], 'tweet_...	2017-12-25 04:39:20	tweets

After 1.b step:

	_score	_index	_source	_crawldate	_type	hashtags	tweetId	text
0	391	hashtag_tweets	{'tweet': {'hashtags': ['Snapchat'], 'tweet_id...	2015-05-23 11:42:47	tweets	snapchat	0x376b20	People who post "add me on #Snapchat" must be ...
1	433	hashtag_tweets	{'tweet': {'hashtags': ['freepress', 'TrumpLeg...	2016-01-28 04:52:09	tweets	freepress,trumplegacy,cnn	0x2d5350	@brianklaas As we see, Trump is dangerous to #...
2	232	hashtag_tweets	{'tweet': {'hashtags': ['bibleverse'], 'tweet_...	2017-12-25 04:39:20	tweets	bibleverse	0x28b412	Confident of your obedience, I write to you, k...

Separate training datasets and testing datasets

1.Training datasets:

	_score	_index	_source	_crawldate	_type	hashtags	tweetId	text	identification	emotion
0	391.0	hashtag_tweets	{'tweet': {'hashtags': ['Snapchat'], 'tweet_id...	2015-05-23 11:42:47	tweets	snapchat	0x376b20	People who post "add me on #Snapchat" must be ...	train	anticipation
1	433.0	hashtag_tweets	{'tweet': {'hashtags': ['freepress', 'TrumpLeg...	2016-01-28 04:52:09	tweets	freepress,trumplegacy,cnn	0x2d5350	@brianklaas As we see, Trump is dangerous to #...	train	sadness
2	376.0	hashtag_tweets	{'tweet': {'hashtags': [], 'tweet_id': '0x1cd5...	2016-01-24 23:53:05	tweets		0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂 <LH>	train	fear
3	120.0	hashtag_tweets	{'tweet': {'hashtags': ['authentic', 'LaughOut...	2015-06-11 04:44:05	tweets	authentic,laughoutloud	0x1d755c	@RISKshow @TheKevinAllison Thx for the BEST TI...	train	joy
4	1021.0	hashtag_tweets	{'tweet': {'hashtags': [], 'tweet_id': '0x2c91...	2015-08-18 02:30:07	tweets		0x2c91a8	Still waiting on those supplies Liscus. <LH>	train	anticipation

2. Testing datasets:

	_score	_index	_source	_crawldate	_type	hashtags	tweetId	text	Identification
0	232.0	hashtag_tweets	{'tweet': {'hashtags': ['bibleverse'], 'tweet_...	2017-12-25 04:39:20	tweets	bibleverse	0x28b412	Confident of your obedience, I write to you, k...	test
1	989.0	hashtag_tweets	{'tweet': {'hashtags': [], 'tweet_id': '0x2de2...	2016-01-08 17:18:59	tweets		0x2de201	"Trust is not the same as faith. A friend is s...	test
2	66.0	hashtag_tweets	{'tweet': {'hashtags': ['materialism', 'money'...	2015-09-09 09:22:55	tweets	materialism,money,possessions	0x218443	When do you have enough ? When are you satisfi...	test

2. On the data pre-processing part,

a. Examine how many types of the '_index' provided in the datasets

```
index_train = []
X_index = df_Train['_index'].values.copy()

for idx in X_index:
    if not idx in index_train:
        index_train.append(idx)

print("In trainig dataset, there are %d number of type of value in column '_index'" %len(index_train))
```

```
In trainig dataset, there are 1 number of type of value in column '_index'
```

```
index_test = []
X_index = df_Test['_index'].values.copy()

for idx in X_index:
    if not idx in index_test:
        index_test.append(idx)

print("In testing dataset, there are %d number of type of value in column '_index'" %len(index_test))
```

```
In testing dataset, there are 1 number of type of value in column '_index'
```

b. Examine how many types of the '_types' provided in the datasets

```
type_train = []
X_type = df_Train['_type'].values.copy()

for tp in X_type:
    if not tp in type_train:
        type_train.append(tp)

print("In trainig dataset, there are %d number of type of value in column '_type'" %len(type_train))
```

```
In trainig dataset, there are 1 number of type of value in column '_type'
```

```
type_test = []
X_type = df_Test['_type'].values.copy()

for tp in X_type:
    if not tp in type_test:
        type_test.append(tp)

print("In testing dataset, there are %d number of type of value in column '_type'" %len(type_test))
```

```
In testing dataset, there are 1 number of type of value in column '_type'
```

From the 2.a and 2.b step, I think that those '_index' and '_type' columns will not be the important feature.

c. Here is the main work on the data pre-processing: remove punctuation(not include emotion symbol), and stemming on sentences of both training and testing data.

```
count_vect = CountVectorizer(tokenizer=nltk.word_tokenize)
analyze = count_vect.build_analyzer()
```

```
X_text = df_Train['text'].values.copy()

stopWords = ENGLISH_STOP_WORDS

Lemma = WordNetLemmatizer()
#stemmer = PorterStemmer()
puncLen = len(string.punctuation)

for i in range(len(X_text)):
    s = X_text[i]
    s = s.lower().translate(str.maketrans(string.punctuation, puncLen*':')) #remove punctuation with
    s.strip() #remove white space
    s = s.replace(':','') #replace back ':' into string empty

    temp = []
    for words in analyze(s):
        if words in stopWords: #remove stopwords
            continue
        words = Lemma.lemmatize(words.strip())
        temp.append(words)

    X_text[i] = " ".join(temp).strip(" ")

df_Train['StemmedText'] = pd.DataFrame(data = X_text)
df_Train[['text', 'StemmedText']].head()
```

	text	StemmedText
0	Confident of your obedience, I write to you, k...	confident obedience write knowing ask philemon...
1	"Trust is not the same as faith. A friend is s...	trust faith friend trust putting faith mistake...
2	When do you have enough ? When are you satisfi...	satisfied goal really money materialism money ...
3	God woke you up, now chase the day #GodsPlan #...	god woke chase day godsplan godswork lh
4	In these tough times, who do YOU turn to as yo...	tough time turn symbol hope lh

d. Do Label Binarize to the column ‘emotion’

```
# encode label first
lb_le =  LabelBinarizer()
X_label = df_Train['emotion'].values
X_label_matr = lb_le.fit_transform(X_label)
X_label
```

```
array(['anticipation', 'sadness', 'fear', ..., 'anticipation', 'joy',
       'trust'], dtype=object)
```

```
lb_le.inverse_transform(X_label_matr[:10]).tolist()
```

```
['anticipation',
 'sadness',
 'fear',
 'joy',
 'anticipation',
 'joy',
 'sadness',
 'anticipation',
 'joy',
 'anger']
```

e.Do One hot encoder on the column ‘hashtags’

```
dict_hashtags = dict()
X_hashtags = df_Mix['hashtags'].values.copy()

for hashtags in X_hashtags:
    for ht in hashtags.split(','):
        ht = ht.lower().strip().replace(' ','')
        if not ht in dict_hashtags.keys():
            dict_hashtags[ht] = 1
        else:
            dict_hashtags[ht] += 1

print('There are %d of types of hashtags' %len(dict_hashtags.keys()))
```

```
There are 370762 of types of hashtags
```

There are some hardship I faced when I doing the one hot encoder on the column 'hashtags'. As shown on the figure above, there are over 370000 types of hashtags, which if I do the one hot encoder on the hashtags, it will totally cause to **MEMORY ERROR** due to the datasets are huge.

To overcome this, I have make statistics on the frequency of each hashtags.

```
dict_tmp = sorted(dict_hashtags.items(), key=lambda d: d[1], reverse=True)
dict_tmp[:101]
```

```
[('', 964715),
 ('life', 27660),
 ('love', 14268),
 ('god', 12544),
 ('dream', 6983),
 ('blessed', 6980),
 ('sad', 6025),
 ('trump', 5363),
 ('faith', 5210),
 ('motivation', 4939),
 ('special', 4817),
```

By removing the empty string hashtags, I have chosen the top 50 hashtags as the feature, and wish to do one hot encoder on them. However, one hot encoder from the module ***sklearn*** will still face the **MEMORY ERROR**. From the information I have searched, there is another module called ***dummyPy***, which also provided one hot encoder technique, but had faster computation and lower memory consuming compared to the module ***sklearn***. The one hot encoder from the module ***dummyPy*** still result in **MEMORY ERROR**. Finally, I make a simple one hot encoder by create dummies by myself, which works without **MEMORY ERROR**.

```
X_dummyHash = []

for l in range(len(Top_Keys)):
    X_dummyHash.append( np.zeros(df_Mix.shape[0], dtype=int).tolist() )

#Stat the existence of hashtags of each data
#eg. the data with no hashtag 'power' will have the value 0 in column 'power'
#eg. the data with hashtag 'love' will have the value 1 in columns 'love'
for i in range(len(X_hashtags)):
    hashtags = X_hashtags[i]

    for ht in hashtags.split(','):
        ht = ht.lower().strip()

        if ht in Top_Keys:
            dIdx = Top_Keys.index(ht)
            X_dummyHash[dIdx][i] = 1
```

```
df_Hash = pd.DataFrame(data = X_dummyHash, columns = Top_Keys)
df_Hash.head(3)
```

	life	love	god	dream	blessed	sad	trump	faith	motivation	special	...	thursdaythoughts	america	tuesdaythoughts	lol	business	shame	entrepre
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

3. Features Engineering

a.Word2Vec Embedding (FINAL MODEL)

From the information I have searched, I could only do word2vec embedding to fit the input on my model, which is Convolutional Neural Network. Of course there exist multiple ways to make the feature fit to the model (Including fit multiples features to it), but I have tried several times, it still failed. So, to at least make the model works, I have just simple done the word2vec embedding to the feature 'text' and fit them to the model.

```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 9000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 9000
# This is fixed.
EMBEDDING_DIM = 150

tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df_Train['text'].values)
tokenizer.fit_on_texts(df_Test['text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 770621 unique tokens.
```

The figure above shows the word2vec embedding

b. TFIDF

This is the most common way in my mind when I start dueling on feature extraction from the sentences of each data. However, I think that the TFIDF might find out the important words, but It could not find out the relationship between those words, so I elimated this approach. (And it does not result a good accuracy as well.)

Do TF-IDF Vectorization on the column 'text' of both training and testing datasets

```
vectorizer = TfidfVectorizer(max_features=300, tokenizer=nltk.word_tokenize)

# apply analyzer to training data
vectorizer.fit(df_Train['StemmedText'])
vectorizer.fit(df_Test['StemmedText'])

train_tfidf_features = vectorizer.transform(df_Train['StemmedText'])
test_tfidf_features = vectorizer.transform(df_Test['StemmedText'])

## check dimension
train_tfidf_features.shape
```

```
(1455563, 300)
```

c. Pre-trained Word2Vec

This approach is introduced on the last lab, I have followed the step taught on the class, and it results a well looking accuracy when I apply it to the machine learning model. The reason I was still eliminate it is, the pre-trained word2vec model I used could only produce 300 dimension, which is not enough to describe the relationship of the sentence and the emotion of the huge datasets. Of course they would exist other pre-trained model which will produce more 300 dimension, but what really stop me from continue to work with word2vec is, I believe that doc2vec might work better in this case.

```
#load pre-trained model
wv = gensim.models.KeyedVectors.load_word2vec_format("Pre-trained Model/GoogleNews-vectors-negative300.bin.gz", binary=True)
wv.init_sims(replace=True)
```

```
series_W2V = df_Mix.apply(lambda r: w2v_tokenize_text(r['StemmedText']), axis=1)
series_W2V.head(3)
```

```
0    [people, post, add, snapchat, dehydrated, cuz,...
1    [brianklaas, trump, dangerous, freepress, worl...
2                    [issa, stalking, tasha, 😂😂😂, lh]
dtype: object
```

```
X_W2V = series_W2V.values
X_word_ave = word_averaging_list(wv,X_W2V)
type(X_word_ave)
```

```
display(X_word_ave.shape)
df_W2V = pd.DataFrame(data = X_word_ave)
df_W2V.head(3)
```

```
(1867535, 300)
```

	0	1	2	3	4	5	6	7	8	9	...	290	291	292	293	
0	0.049211	0.006211	-0.005735	0.111876	-0.064706	0.028648	-0.015090	-0.075939	0.031530	0.015289	...	0.070716	0.047472	-0.098041	0.039606	-0.04
1	-0.025338	0.030486	0.073989	0.145414	-0.087391	-0.000234	0.031842	-0.046021	0.045276	0.062107	...	0.117231	-0.071234	-0.113496	0.043389	-0.0(
2	-0.023264	0.058045	0.009170	0.030794	-0.028630	0.008735	-0.055967	-0.033752	-0.005382	0.038508	...	-0.015264	0.005514	-0.084022	0.016609	0.0(

3 rows × 300 columns

d.Doc2Vec

If I able to fit the deep learning model with what feature I want, I would say the doc2vec feature will be my best choice. Doc2vec is the extension of word2vec, it find out the relationship between the sentences and documents instead of learning feature representation for words. Futhermore, in those machine learning model I have tried, fitting features produced from Doc2vec result the greatest accuracy.

```
vs = 500
```

```
model_dbow = Doc2Vec(dm=0, vector_size=vs, negative=0, min_count=1, alpha=0.065, min_alpha=0.065)
model_dbow.build_vocab([x for x in tqdm(List_Mix)])

for epoch in range(40):
    model_dbow.train(utils.shuffle([x for x in tqdm(List_Mix)]), total_examples=len(List_Mix), epochs=1)
    model_dbow.alpha -= 0.002
    model_dbow.min_alpha = model_dbow.alpha
```

```
100%|██████████| 1867535/1867535 [00:00<00:00, 2900574.27it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2883547.43it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2904035.88it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2905049.37it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2937539.03it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2978245.69it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2924559.86it/s]
100%|██████████| 1867535/1867535 [00:00<00:00, 2938310.38it/s]
```

```
df_D2V_Train.head(3)
```

	0	1	2	3	4	5	6	7	8	9	...	490
0	0.000152	-0.000349	0.000059	-0.000330	-0.000868	-0.000141	0.000810	-0.000011	0.000854	0.000685	...	-0.000551
1	-0.000835	-0.000918	0.000146	0.000929	-0.000399	-0.000587	-0.000126	0.000110	-0.000684	-0.000929	...	-0.000838
2	-0.000043	-0.000381	0.000322	-0.000898	0.000036	-0.000589	0.000742	0.000074	-0.000024	-0.000823	...	-0.000896

3 rows × 500 columns

4. Training Model.

Before I have chosen the Convolutional Neural Network as my final model, I actually have tried multiple models to test the validation accuracies, included Decision Tree, Random Forest Classifier, K-Nearest Neighbors, Support Vector Machine, XGBoosting and so on. However, each of them result around 43% of validation accuracies and around 41% of public accuracies.(which is at least over the 40% benchmark).

To find a better approach on pushing the accuracy, I chose to develop a deep learning model. The first model I used is just a simple neural network with multiple Dense layers, which result was not much different from the machine learning model I have used before.

Then, I found that LSTM model might work better than what I have done so far, but it's heavy time consuming on training the model even I just setting a small value of epochs. Due to the limitation of time, I decided to give up on this model.

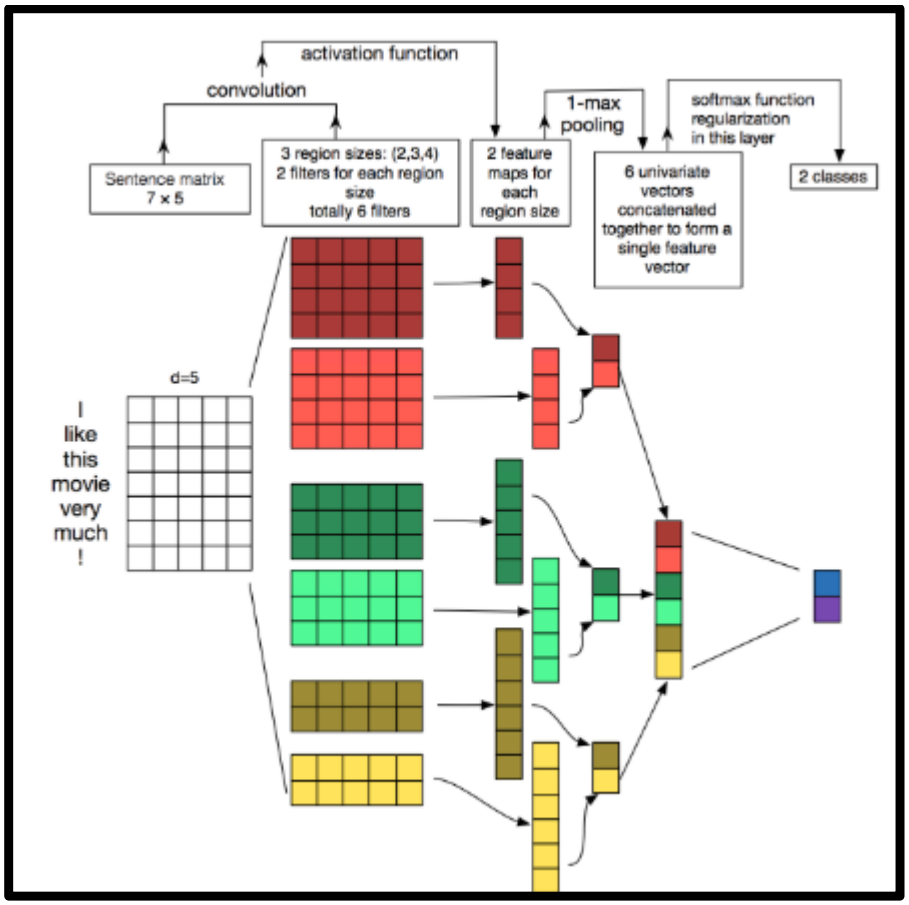
Finally, I chose Convolutional Neural Network as my final model. If we search on the internet about the Convolutional Neural Network, basically there will almost related to the image prediction stuff. However, CNN actually is not only deal with the images.

```
[0.001 estimators logistic Regression with Pipeline]
Accuracy: 0.4590
              precision    recall  f1-score   support

anticipation       0.52      0.07      0.12      7882
     sadness       0.52      0.43      0.47     49577
        fear       0.40      0.27      0.32     27788
         joy       0.56      0.15      0.23     12857
       anger       0.46      0.82      0.59    103161
       trust       0.37      0.29      0.32     38902
     disgust       0.63      0.05      0.09      9859
    surprise       0.45      0.14      0.22     40907

    accuracy                           0.46    290933
   macro avg       0.49      0.28      0.30    290933
weighted avg       0.46      0.46      0.41    290933
```

The machine learning model achieved similar accuracies.



The figure above shows the basic concept of CNN on text classification.

```
ccuracy: 0.5206
Epoch 18/20
1163732/1163732 [==============================] - 157s 135us/step - loss: 1.23
ccuracy: 0.5202
Epoch 19/20
1163732/1163732 [==============================] - 171s 147us/step - loss: 1.23
ccuracy: 0.5202
Epoch 20/20
1163732/1163732 [==============================] - 156s 134us/step - loss: 1.23
ccuracy: 0.5189
[Convolutional Neural Network]
Accuracy: 0.5189
```

CNN model achieved better validation accuracy in my case.

There are some interesting part when I train the CNN model. I found that even with the same settings (EPOCHS, BATCH_SIZE, etc), the validation accuracy will be different when model is restart and finish training. However, the validation accuracy will not different so much. I think this happened because the value of epoch is too small to make the model prediction converge. (I have only set the epoch with 20) Since, the CNN model predicted the best accuracy I have achieved so far, so I make it as the final model.

HARDSHIP

As mentioned on the METHOD section, I think the biggest hardship is to deal with the **MEMORY ERROR** problem. This might spend so much time if we wish to extract some good features from the datasets. Due to the huge datasets, a few dimension of features might not enough to describe the

datasets, but it will face **MEMORY ERROR** if we put over 1000 dimension of features (depends on the feature engineering used).

The second hardship is to deal with the input format for the deep learning model. As stated before, there are so many ways to let us fit as much as features we want to the deep learning model, I have tried them so hard, and they are still failed on fitting the model. Even they can fit to the model, it still have a problem of gradient exploding which result of training loss with *nan*. Due to the limitation of time, to at least make the CNN model works, I have to eliminate all the features engineering I have done, and just simple do word2vec embedding to the sentences.

CONCLUSION

My best result on the competition

With this competition, I have no regret on the result I have achieved (Well, at least I have tried it so hard). However, I have learned that, sometimes, it does not much matter on what model we use on classification, It might be more important on how we dealing with the features stuff (And there is why Data Science is one of the hottest job nowadays). Futuremore, even we have learned so many techniques of data mining, each of them had drawbacks, and sometimes does not work on the reality cases.