General NLP Classification Using Neural Network

Zhaoguo Zhu
Boston University, Computer Science Department
Boston, MA 02215
zhuzg@bu.edu/idcili393939@gmail.com

Abstract

This paper discusses general approaches needed for a text classification problem, including preliminary analysis, text cleaning, tokenization, word embedding, model selection and optimization. The practice sample is Kaggle Disaster Tweets. The overall accuracy reaches 97% for training and 75% for validation and testing. The package used for this project is tensorflow.

1. Introduction

1.1 Uniqueness of NLP Problems

The two main types of supervised learning problems are Classification and Regression problems. Classification problems are usually stricter, it is more of a yes or no question, while regression problems' output requires only a prediction that lies in a continuous range. Computer Vision and Natural Language Processing meet classification problems very frequently, and among these two subjects the problem regarding NLP is much harder. The reason is that the way how computer scientists do feature extraction for these two problems are not the same. For computer vision feature extraction, it is much easier to filter out irrelevant pixels and focus on very specific object. For instance, human vision captures views in a very large range, but the brain ignores the image part that are not being focused on. For NLP feature extraction on the other hand is not the same. The information in sentences needs to be very carefully taken care of because the whole sentence together holds the context. If scientists filter out a lot of elements, they might lose the original meaning. The topic of this paper is general NLP classification with neural networks. Techniques within the papers can be utilized in many common text classifications problems such as determine spam emails, fake news, sarcasm etc.

1.2 General Procedure of NLP

Procedure to solve NLP problems usually goes over these steps: preliminary dataset observation and analysis, data cleaning, post cleaning dataset observation and analysis, tokenization and embedding, training model and optimization. Each step will determine how the customization of next step will be. The layer of this paper will be in the exact order with brief introduction of the concept and detail of the step.

2. Preliminary Dataset Observation and Analysis

2.1 Brief Introduction

The purpose of this step is to determine what elements within the text needed to be filter out and what is the data distribution of different categories.

2.2 Dataset

The dataset used for this project is Kaggle Natural Language Processing with Disaster Tweets[1]. The dataset contains 5 features: id, keyword, location, text, and target. Only text and target are needed for this project. The target feature is the label, indicating whether a text is regarding a real disaster or not.

2.3 Observation and analysis

The size of the dataset is 7613 and the distributions of 0's and 1's are not the same, with 4342 0's and 3271 1's. Data explorer on the Kaggle indicate that there are 7503 unique values of text feature, meaning that there are some texts are of same contents. Within the texts, there are numeric values, hashtag, website links, punctuations, emoji, non-English character, and proper noun.

3. Data Cleaning

3.1 Brief Introduction

The purpose of this step is to resolve the problems observed in preliminary data observation and analysis ---- to filter the elements not required and to solve unbalance distributions issue if exists.

3.2 Solve Problem of Unbalance Distribution

To solve problem of unbalance distributions of different categories usually relies on 3 approaches: removing oversized data, adding more data or apply data augmentation. The unbalance issue of Kaggle Disaster Tweets is not overwhelmed therefore none of the methods mentioned above were applied during developing.

3.3 Text Preprocessing

The following filters are applied during developing:

a. Removing stop words in English (waste of space for tokenization because appearance frequency)
b. Removing all numeric values (unrecognizable by word embedding)
c. Removing all non-English character (unrecognizable by word embedding)
d. Removing all Hashtag (involving proper noun)
e. Removing all links (meaningless in word embedding)
f. Lower case (uniform word pattern that will make tokenization easier)
g. Removing all punctuation and extra space (meaningless in word embedding)

4. Post Cleaning Dataset Observation and Analysis

4.1 Brief Introduction

The purpose of this step is to gather information of what is left of the text after being cleaned. Explore the average length, most frequent word for each category etc. to determine what you want to do in tokenization and word embedding.

4.2 Observation and Analysis

The description of text feature after being cleaned are shown in Figure 1:

```
count    7613.000000
mean        8.386182
std         3.548712
min         0.000000
25%         6.000000
50%         8.000000
75%        11.000000
max        23.000000
Name: length, dtype: float64
```

Figure 1: Sample description post-cleaning

5. Tokenization and Embedding

5.1 Brief Introduction

The purpose of this step is to transform all the text after being cleaned into the same length that is represented by a meaning numeric value.

5.2 Tokenization

Knowledge acquired from post cleaning dataset observation and analysis indicates that the average length of the text after being cleaned is 8, so 8 will be the uniform length for tokenization, which means no matter what length these sentences were before, they will be 8 from the point of tokenization forward. The maximum vocabulary size is set as 20000, which is common. The vocabulary size determines how many unique words are in the set. After the tokenization, all the text will be transformed into a vector of integers of length 8, where each integer is a unique word to tensorflow package[2]. See Figure 2 for effect:

```
>>> X_train[0]
'trauma happen anywhere school home etc time learn abcs trauma parent'
>>> text_vectorizer(X_train[0])
<tf.Tensor: shape=(8,), dtype=int64, numpy=
array([  197,   969, 12388,    90,    73,  1145,    19,   670],
      dtype=int64)>
```

Figure 2: text after cleaning and text after tokenization

5.3 Embedding

In short terms, word embedding requires a sentence and return a similarity that is represented in digits based on a word relationship model that is pre-trained. Imaging a sentence formula like “crown + man == king,” “crown + woman == queen.” A proper word relationship model can understand this relationship and therefore if the word “crown” is given to the model, it will return a similarity vector of digit. The size of this vector is determined by developer and the size means the amount of relationship. In the development, the output dimension was selected to be 256 for the reason of easier computational calculations. After embedding, the text will be represented by 8 words an d each word will be represented by 256 relationship, which yields a shape of (8, 256) in Figure 3:

```
>>> embedding(text_vectorizer(X_train[0]))
<tf.Tensor: shape=(8, 256), dtype=float32, numpy=
array([[ 0.02498122,  0.12195171, -0.0218547 , ...,  0.0506405 ,
         0.02873882,  0.0319295 ],
       [-0.0191502 , -0.02578357, -0.04083975, ..., -0.04937926,
         0.05607995,  0.04411818],
       [-0.01392165, -0.05262939, -0.05212688, ..., -0.02913958,
        -0.00891752, -0.0152895 ],
       ...,
       [-0.00295796, -0.0672919 , -0.01566605, ..., -0.06156883,
         0.02863346, -0.04355982],
       [ 0.00957798, -0.00275438, -0.0072049 , ..., -0.07192276,
         0.00145524, -0.00708579],
       [-0.01964861,  0.06967038,  0.04494005, ...,  0.00640108,
         0.04422046, -0.02686536]], dtype=float32)>
```

Figure 3: text after embedding

6. Training Model

6.1 Brief Introduction

The purpose of this step is to determine the layout of a neural network model.

6.2 Model Layout

In this project, 3 different types of neural networks are selected:

a. Simple Neural Network:

```
# Training with simple neural network
model_1 = tf.keras.Sequential()
model_1.add(text_vectorizer)
model_1.add(embedding)
model_1.add(tf.keras.layers.Flatten())
model_1.add(tf.keras.layers.Dense(128,activation='relu',activity_regularizer=tf.keras.regularizers.L2(1)))
model_1.add(tf.keras.layers.Dropout(0.2))
model_1.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

b. Conv1D:

```
# Training with conv neural network
model_2 = tf.keras.Sequential()
model_2.add(text_vectorizer)
model_2.add(embedding)
model_2.add(tf.keras.layers.Conv1D(32, 2, activation='relu'))
model_2.add(tf.keras.layers.Dropout(0.7)),
model_2.add(tf.keras.layers.MaxPooling1D())
model_2.add(tf.keras.layers.Conv1D(32, 2, activation='relu'))
model_2.add(tf.keras.layers.Dropout(0.7)),
model_2.add(tf.keras.layers.GlobalMaxPooling1D())
model_2.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

c. BERT[3]:

```
# Training with BERT neural network
model_3 = tf.keras.Sequential()
model_3.add(text_vectorizer)
model_3.add(embedding)
model_3.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, activation='tanh',return_sequences=True)))
model_3.add(tf.keras.layers.Dense(64,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.Dense(128,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.Dense(128,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.Dense(64,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.LSTM(64,return_sequences=True))
model_3.add(tf.keras.layers.Dense(64,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.LSTM(64))
model_3.add(tf.keras.layers.Dense(32,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.Dense(16,activation='relu',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
model_3.add(tf.keras.layers.Dropout(0.2))
model_3.add(tf.keras.layers.Dense(1,activation='sigmoid',activity_regularizer=tf.keras.regularizers.L2(0.0004)))
```

Each of the model is using “adam” as optimizer with 0.001 default learning rate during 5 epochs, calculating loss using binary cross entropy.

6.3 Output

All 3 models give very close result of 97% accuracy in training dataset and 75% accuracy in validation and testing dataset, and the loss of validation had been increasing since the first epoch, indicating there is an overfitting problem. See Figure 4:

```
Epoch 1/5
172/172 - 7s - loss: 0.5567 - accuracy: 0.7234 - val_loss: 0.4562 - val_accuracy
: 0.8001 - 7s/epoch - 38ms/step
Epoch 2/5
172/172 - 6s - loss: 0.2133 - accuracy: 0.9232 - val_loss: 0.5585 - val_accuracy
: 0.7549 - 6s/epoch - 36ms/step
Epoch 3/5
172/172 - 6s - loss: 0.0834 - accuracy: 0.9692 - val_loss: 0.6247 - val_accuracy
: 0.7578 - 6s/epoch - 36ms/step
Epoch 4/5
172/172 - 6s - loss: 0.0515 - accuracy: 0.9765 - val_loss: 0.6854 - val_accuracy
: 0.7571 - 6s/epoch - 36ms/step
Epoch 5/5
172/172 - 6s - loss: 0.0440 - accuracy: 0.9797 - val_loss: 0.7315 - val_accuracy
: 0.7688 - 6s/epoch - 36ms/step
 1/24 [>.............................] - ETA: 0s - loss: 0.8122 - accuracy: 0.7
500
24/24 [==============================] - 0s 2ms/step - loss: 0.8157 - accu
racy: 0.7441
```

Figure 4: Simple Neural Network training log

7. Optimization

7.1 Brief Introduction

The purpose of this step is to resolve issues occur during training the model such as underfitting or overfitting. A sign of underfitting is that the accuracy of training dataset is very poor, meaning that model selected cannot analyze the pattern based on the features given; A sign of overfitting is that the accuracy of training dataset is high, but the accuracy of validation dataset is not that good, and the loss of validation is constantly increasing.

7.2 Overfitting

Overfitting usually occurs when you have more features than number of samples. The hypothesis function in this case will pass through basically each data point in the corresponding data dimension space. Instead of making the model perform perfect in training data, developer wants the model to perform a "rough" prediction so that the hypothesis function, or decision boundary can fit unseen data as well. The common ways to solve overfitting increasing the size of datasets, removing features, or applying regularization.

7.3 Dropout and Regularization

Neural Network takes in original features and turn them into other features in hidden layers. Dropout layers[4] in the tensorflow will randomly selected a proportion of nodes to be ignored during calculating forward propagation, which reduces the number of features in the hidden layers to reduce overfitting. In the project, a dropout layer is placed after each dense layer that ignores randomly 20% of the nodes. Regularization can help penalize the parameter (weight in neural networks) during calculate gradient descent. In this project, L2 normalization is applied. The choice of penalize parameter varies based on what model is. For Simple Neural Network model, the parameter is 1 but in BERT it is 0.0004. See Figure 5 the loss begins to decrease, indicating that the overfitting problems are becoming better:

```
Epoch 1/15
172/172 - 7s - loss: 0.6984 - accuracy: 0.5947 - val_loss: 0.6872 - val_accuracy: 0.5806 - 7s/epoch - 42ms/step
Epoch 2/15
172/172 - 6s - loss: 0.6817 - accuracy: 0.6000 - val_loss: 0.6786 - val_accuracy: 0.6594 - 6s/epoch - 36ms/step
Epoch 3/15
172/172 - 6s - loss: 0.6441 - accuracy: 0.7971 - val_loss: 0.6489 - val_accuracy: 0.7374 - 6s/epoch - 37ms/step
Epoch 4/15
172/172 - 6s - loss: 0.5452 - accuracy: 0.9301 - val_loss: 0.6205 - val_accuracy: 0.7513 - 6s/epoch - 37ms/step
Epoch 5/15
172/172 - 7s - loss: 0.4592 - accuracy: 0.9646 - val_loss: 0.6174 - val_accuracy: 0.7381 - 7s/epoch - 38ms/step
Epoch 6/15
172/172 - 6s - loss: 0.4038 - accuracy: 0.9712 - val_loss: 0.5915 - val_accuracy: 0.7651 - 6s/epoch - 37ms/step
Epoch 7/15
172/172 - 7s - loss: 0.3650 - accuracy: 0.9759 - val_loss: 0.5935 - val_accuracy: 0.7520 - 7s/epoch - 38ms/step
Epoch 8/15
172/172 - 6s - loss: 0.3360 - accuracy: 0.9763 - val_loss: 0.5724 - val_accuracy: 0.7666 - 6s/epoch - 37ms/step
Epoch 9/15
172/172 - 6s - loss: 0.3112 - accuracy: 0.9772 - val_loss: 0.5729 - val_accuracy: 0.7586 - 6s/epoch - 37ms/step
```

Figure 5: Simple Neural Network with dropout and regularization

8. Conclusion and Future Improvement

8.1 Conclusion

The procedure described in this paper can be applied to any common text classification problems with some small modifications in word embedding and final output layer of the neural network based on different type of problem needed to be solve. If the problem is multiclass problem the final layers need to have the same number of nodes and activation function needs to be changed to softmax. Loss calculation need to be categorical cross entropy. The topic of the problem will influence how to choose the proper embedding library. For instance, problem regarding food classification need to rely on a embedding library regarding the topic food. Although general word embedding will always do the job, it is better to be high focus on the subject.

8.2 Improvement

Possible improvement can be done in feature extraction by selecting only the parameter that can be separated in one dimension. Suppose that parameter A can be separated in one dimension and parameter B can be separated in one dimension, then (A,B) can be separated in two dimension. This logic is transitive to higher dimension too.

Another improvement is sentiment analysis, which analyze the emotion level that each sentence has, which will give us a new helpful feature. Similar approaches are also welcome because they will make the data more understandable and separatable to the model.

9. Reference

[1] Kaggle: Natural Language Processing with Disaster Tweets ---- https://www.kaggle.com/competitions/nlp-getting-started/data

[2] Tensorflow ---- https://www.tensorflow.org/

[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

[4] Dropout ---- https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

[5] The evolution of sentiment analysis ---- A review of research topics, venues and top cited papers, Mike V. Mantyla, Daniel Graziotin, Miikka Kuutila