Multi-mood Classification Project

Jin Qu
11/13/2017

OVERVIEW

In this project, I developed three models for classifying mood from messages. The first model was developed using fastText open source toolkit (https://fasttext.cc). However, the official fastText is not able to process emojis and misspellings directly. The second model is a slight variation of fastText, which I call fastText2. The fastText2 was implemented to include emojis into the model. The last one is a character level convolutional neural net (charCNN) model. This model is able to overcome misspellings, which is beyond the scope of the previous two models.

Precision, recall and F1 score are used to access model performance. In order to meet the requirement – "*It is most important that your classifier outputs an empathy of the same polarity as the target empathy (positive/negative/neutral), but it is optimal if it outputs exactly the same empathy/empathies*". I developed two metrics. One is a relaxed metrics, which means that if the predicted class is not exactly same as the ground truth, but has the same polarity score, then the prediction can still be counted as true positive. The other metrics is non-relaxed -- the prediction has to perfectly match the ground truth in order to be valid for true positive. To have a detailed overview of the model performance on each class, I also plotted bar-chart for the relaxed and non-relaxed metrics on every class.

Some of the fastText2 code is modified from the Keras github repo for examples:
https://github.com/fchollet/keras/blob/master/examples/imdb_fasttext.py

The character level model is based on a public github repo:
https://github.com/offbit/char-models/blob/master/doc-cnn4.py . The original model was designed to classify documents. So the model contains both sentence-level and document-level representational learning. I modified the original model by removing the document-level encoder and changed a few setups such as the dimension of character embedding.

METHOD

Data Processing

The whole dataset -- "labeled-messages.csv" was shuffled and split into training and validation dataset. The training dataset has 2,500 messages, which is about 70% of the total dataset. The remaining 30% was used for validation.

Note: in the original datasets, the "labeled_messages.csv" has three labels that do not have an assigned polarity score in "empathies.csv". So I assigned the three labels -- "hopeful", "good" and "frustrated" with polarity score 0.5, 0.5 and -0.8 respectively.

Training

- fastText model was trained using official fastText toolkit. Multiple models were trained with different setups with regard to n-gram and epochs. The models share similar performance. The 3-gram model trained with 30 epochs was picked for further analysis.
- The "fastText2_final" file in the notebooks folder contain detailed code for developing and training fastText2 model. The basic idea is to replicate fastText model architecture in Python and include emoji during tokenization process. The NLTK (http://www.nltk.org) tweet tokenizer was used for identifying emoji.
- The "charModel_final" file in the notebooks folder includes details about the charCNN model.

RESULTS

fastText model has the highest relaxed precision, recall and F1 score on the validation dataset. fastText2 is the second while charCNN is the third. The following table shows the relaxed metrics on the three models.

Table 1. Relaxed Metrics Results

Relaxed metrics	fastText	fastText2	charCNN
Precision	0.91	0.82	0.67
Recall	0.90	0.77	0.61
F1 score	0.91	0.79	0.64

To have a more detailed analysis on how the model performs on each class, three bar-chart was generated to reflect the relaxed metrics on the three models. It is also can be easily seen from these figures that fastText enjoys the best performance on almost every class.

Figure 1. fastText, Relaxed Counts of True Positives, False Positives and False Negatives on Every class (https://plot.ly/create/?fid=JinQu:7)

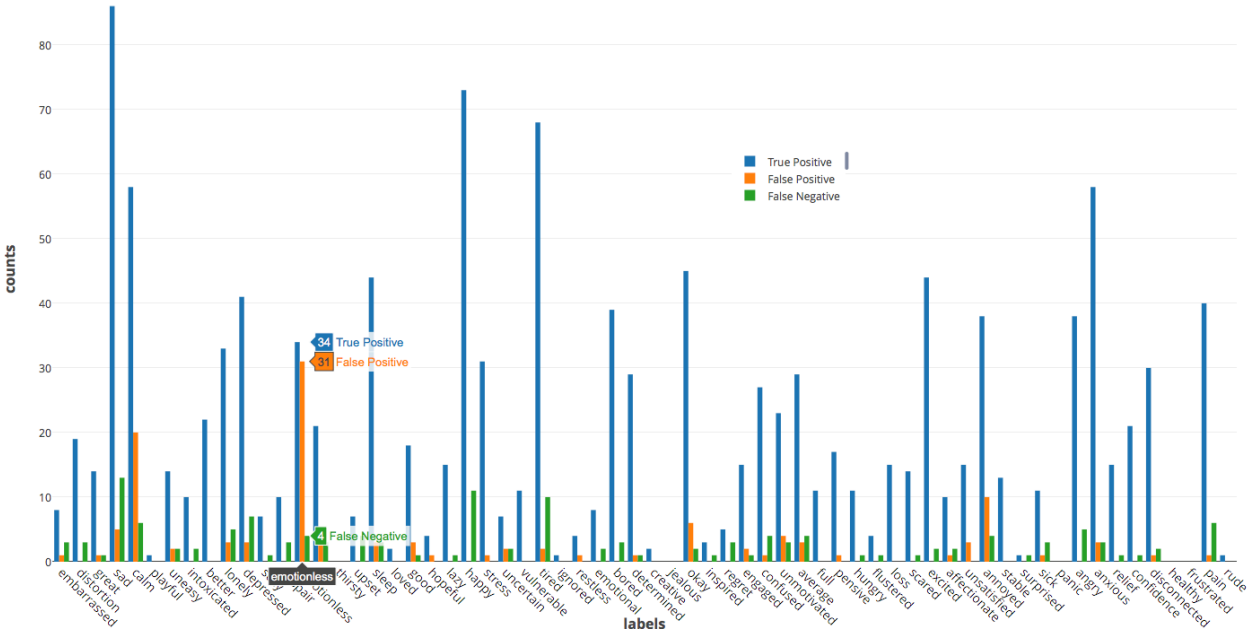


Figure 2. fastText2, Relaxed Counts of True Positives, False Positives and False Negatives on Every class (https://plot.ly/create/?fid=JinQu:9)

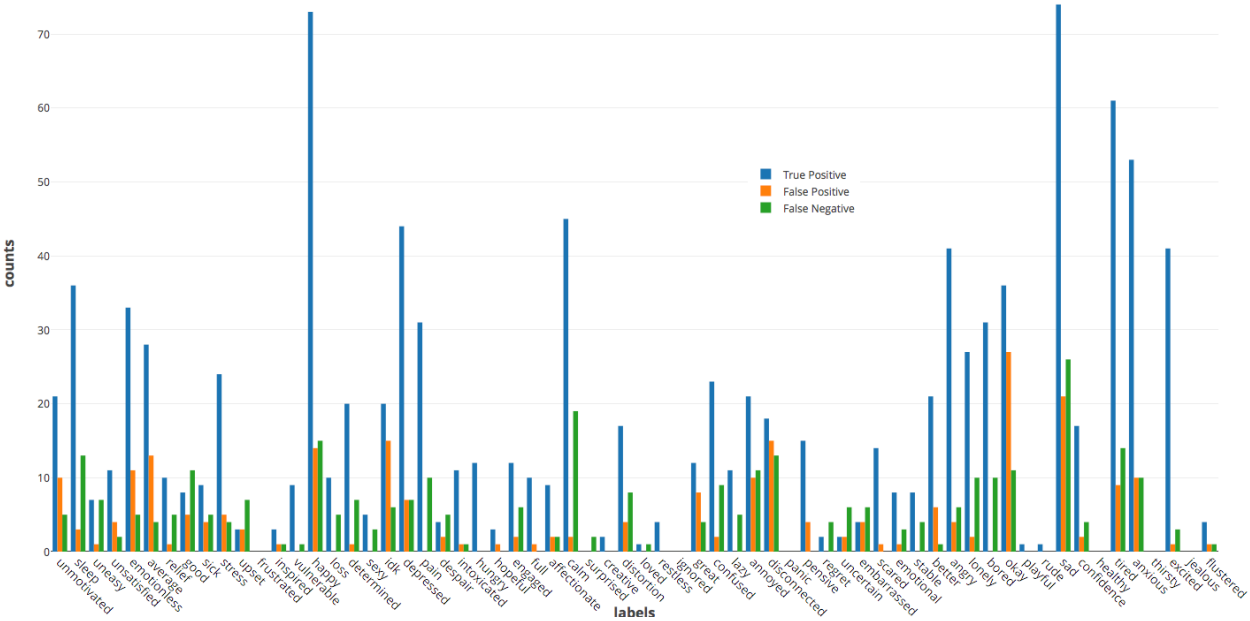
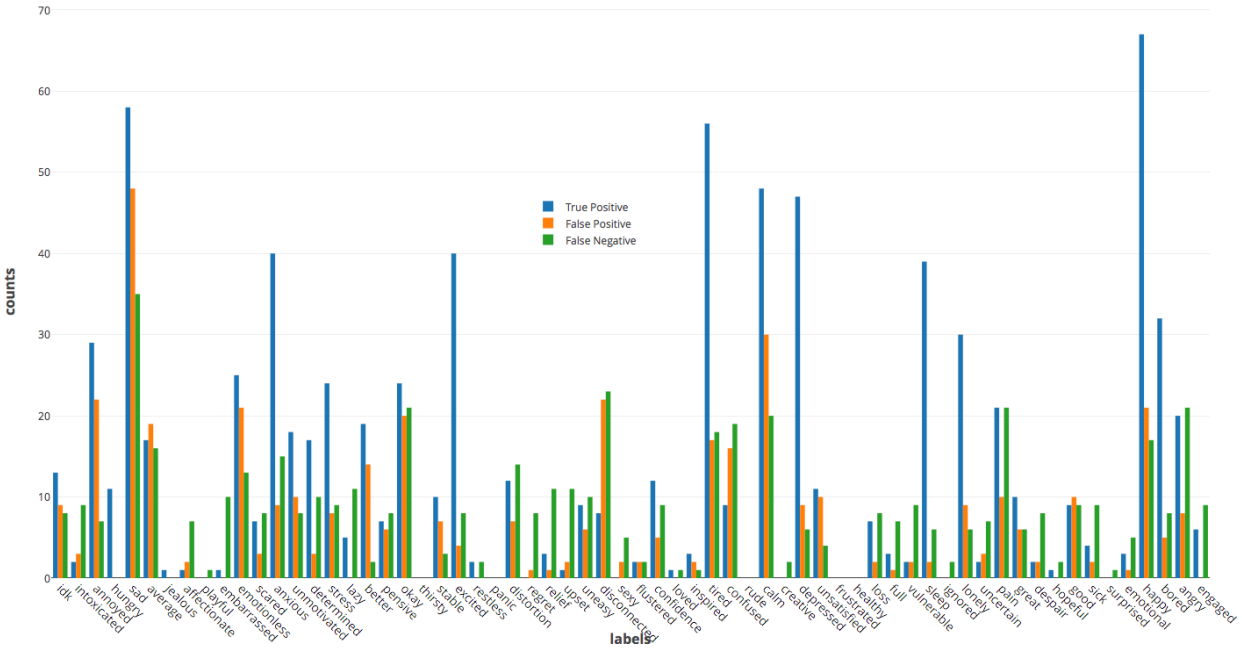


Figure 3. charCNN, Relaxed Counts of True Positives, False Positives and False Negatives on Every class (https://plot.ly/create/?fid=JinQu:11)



However, the fastText is not able to process misspelling, and emoji. For example, if we send a message “hungry” to the model. The model is able to classify this message as the hungry mood. But if we intentionally misspell “hungry” as “huungry”, the model would fail to recognize this string. Also, if we use a hungry emoji, the model would also fail. See screenshot below for more details.

test model performance on misspelling

```
model.predict('hungry', 5)
```

```
(('__label__hungry',
  '__label__sad',
  '__label__surprised',
  '__label__frustrated',
  '__label__playful'),
 array([  1.00000000e+00,   1.95312652e-08,   1.95312652e-08,
          1.95312652e-08,   1.95312652e-08]))
```

```
model.predict('huungry', 5)
```

```
(('__label__calm',
  '__label__emotionless',
  '__label__unmotivated',
  '__label__unsatisfied',
  '__label__annoyed'),
 array([ 0.24414064,  0.18554689,  0.07812502,  0.06445314,  0.05859377]))
```

test model performance on emoji

```
model.predict('😊', 5)
```

```
(('__label__calm',
  '__label__emotionless',
  '__label__unmotivated',
  '__label__unsatisfied',
  '__label__annoyed'),
 array([ 0.24414064,  0.18750002,  0.07617191,  0.06640628,  0.05664065]))
```

Since the fastText2 model included emoji during training. The model is able to process emoji, but not misspellings. See below:

test model performance on emoji

```
random_input = text_processor('😋', token_indices)
random_res = model.predict(random_input)
# print out top five predictions
list(empathies['empathy'][np.argsort(random_res[0])[63:58:-1]])
```

```
['hungry', 'okay', 'sad', 'tired', 'happy']
```

Although charCNN has the lowest score, it is able to process misspelling:

```
random_input = text_processor('huuungry', char_indices)
```

```
random_pred = model.predict(random_input)
```

```
[list(empathies['empathy'][np.argsort(random_pred)[0][63:58:-1]]), np.sort(random_pred)[0][63:58:-1]]
```

```
[['hungry', 'angry', 'sad', 'better', 'upset'],
 array([ 0.76542383,  0.17750278,  0.01207582,  0.0068391 ,  0.00452387], dtype=float32)]
```

FINAL WORDS

I presented three models for this multi-label classification project and used relaxed and non-relaxed metrics to assess model performance. The naïve fastText model gave the best performance according to the metrics. It was also fast to train. However, the disadvantage is also quite obvious – fast Text is unable to directly process misspelling and emoji.

The charCNN model did not have a high metrics score compared with the other two models. However, it is more robust when misspelling and emoji appear in the text, which makes it well suit the social media setting. In this project, given the small size of the dataset, it is highly possible that the charCNN model overfit the training data. This model can be further improved if there more training data are available.