Sentiment analysis on Twitter using semeval 2016 data

Zepeng Huo
Department of Computer
Science and Engineering
Texas A&M University
Email: Guangzhou92@tamu.edu

Pradnyil Sranjay
Department of Industrial
and Systems Engineering
Texas A&M University
Email: pradnyil@tamu.edu

***Abstract*—Sentiment analysis is getting popular on social media. In this paper, we propose a method to classify the emotion valence of a given tweet. The whole framework can be separated as two parts: first, we cluster the similar tweets based on the fact that all data of semeval-2016 are collected from specific topics. And the topic can be distinguished as three different parts. Based on the result of clustering, we feed the tweet into the Tensorflow network to classify.**

***Keywords*—Social Media, Sentiment Analysis, Clustering, Neural Network, Tensorflow.**

I. Introduction

The problem of sentiment classification is not a new one and now nor is that of twitter sentiment classification. The first appearance in the Semeval tasks was in 2013 and this particular task has had a rerun ever since. But over the years the approach and methodology used in for this task has undergone a lot of changes. As noted by HLTCOE, JHU. Et all (2013). in the review of the 2013 team participants all the teams used a supervised learning approach to the task by employing human generated features to feed to a classification algorithm. In the most recent review of the same task by Nakov, Preslav, 2016.et all, the most popular systems used are the deep neural networks like convolution neural networks and recurrent neural networks. The trend of shifting away from human trained features to a more semi supervised technique was the motivation for the use of our approach to the problem.

II. Task and Problem

In Semeval 2016 competition task 4, Twitter sentiment analysis, we need to find out a way to classify sentiment of tweets. The data they provided is of two parts, training set and testing set. Each dataset consists of a couple thousand tweets and each tweets are labeled as either 'positive' or 'negative'. There are 5 subtasks and we only focus on the first one, binary classification. The data are collected by topics. Each topic has a couple hundred tweets. In other subtasks, the data comes with another label called 'topic'. And these subtasks aim to classify sentiment of tweets based on topics. However, in the subtask, we don't have this label. But it gives us idea that these tweets are distinguishable by topics. Thus, in order to make use of topic relatedness, we cluster the tweets in three broad categories. In each cluster, we have a particular classification model to classify that category. As for classification model, we use the Tensorflow neural network. The reason we want to use this model is because it's robust to noisy input but also sensitive to distinguishable text which can carry emotional expression.

III. High-level description of approach

We will have two tasks. The first task was training a model using all the data in the dataset irrespective of the cluster it belongs to. This model does a binary classification of the input tweets into positive and negative tweets. For this problem we used the SVM as a baseline and then a convolution neural network was used to compare the two models.

In the second task, we first cluster the data into the 3 relevant clusters and then we feed the clustered tweets as input to the CNN model. We built a simple single layer CNN to handle the classification task. The tweets In the model tuning process, the data input type to the model and the model parameters were tuned. We will take a closer look at the neural network in the next section.

Before the classification, we need to tokenize all the tweets in Twitter context. This step is necessary because some expression may be unique on social media like Twitter and thus carry specific emotion. For example, the emoticon ':)' means happy or smiley, and it is highly likely related to positive sentiment. But if we use normal tokenization method, it may separate it as ':' and ')', so it loses its original meaning. The tokenization method we use is twokenize. It's specific to Twitter and make every token intact, without adding or losing meanings. After tokenization, we can feed each tweets into a binary classification. Since we use two steps to classify each tweet, we'll introduce the problems we tackled on each step.

As for clustering, we want to make use of the hidden

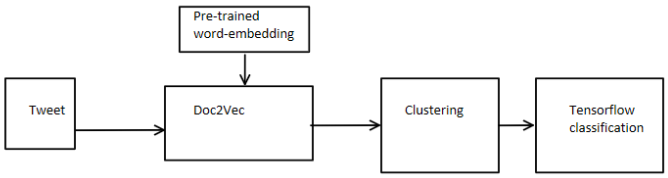

Fig. 1. Framework of 2-step classification

information under the data provided by Semeval-2106. Even though this subtask doesn't require to classify topics, we still want to have related tweets to be classified when they share similar topics. Normally, other sentiment analysis always feed

all the texts into one classification model. But sometimes, some texts are harder than others to classify, for example, two tweets talk about the same topic and thus likely use similar wording. Distinguishing the nuance between them is harder than other tweets. So the intuition is that we need to use a powerful classification model to distinguish the tweets that are similar, which are thus harder to classify. So we use a pre-trained word embedding to make the underlying topic in each tweet more salient. The way pre-trained word embedding was created is to cluster similar tweets sharing similar topics. Therefore, each word vector representing one original English word is similar to another word vector carrying a word with meaning close to the first one. At the original data in other subtasks, the dataset has 60 topics. But most of them are highly related, for example, 'apple' and 'apple watch', 'Donald Trump' and 'Hillary'. Usually, when people are talking about this similar topics, they tend to use similar words carrying similar emotions. But when they talk about other topics, this words can express opposite sentiment, thus making the classification give unsatisfactory results. We want to classify similar tweets in one topic-specific classification. We chose the size of clustering categories as 3. The number of categories is not too big, such as originally 60, making the model underfitted. Nor it's not too small to make the less related tweets have fallen into same cluster.

IV. Specific System Implementations

In the part of clustering, we have our novelty in the system. First, as for converting the tweet into a mathematical representation, most common way is to use bag of words and word embedding. Then the whole tweet is concatenation or averaged vector of word embedding. These methods have problems of either making tweets vectors too long or losing lots of latent information by averaging word vectors. In our system, we use Doc2Vec. It can learn the vector of one sentence of one document by going through the corpus we provide. It then simultaneously learns the word vector in each tweet and the document vector of that tweet. The words appearing in similar context will likely have similar vector representation. The tweets consisting of similar word vectors will have similar document vectors. Thus, the Doc2Vec can preserve the original information in the tweet at the greatest extend and also avoid increasing the tweet vector dimension unnecessarily. Both the time complexity and space complexity can come to a balance.

Since the dataset Semeval provides is relatively small (up to 6000 tweets for training), we may have the problem of over-fitting the data thus making the noisy input contaminate the model. For example, if some of the tweets are using sarcasm and making positive adjectives carrying negative emotions. If we only train on these tweets, it's highly likely that similar positive words will be classified negative in other tweets, even though these tweets didn't use sarcasm. In the light of this, we propose a method to make the vector initialization closer to a well-established understanding of English words in Twitter. By initializing the word embedding using pre-trained word vectors, the document vector is more likely to come to a optimum. For the pre-trained word embedding, we used the library called Global Vectors for Word Representation (GloVe). The pre-trained word vectors are trained on 2 billion tweets, ending having 27 billion tokens and 1.2 million vocabulary. After feeding pre-trained word embedding into the Doc2Vec model, we make our model carry more well-known usage of English words in Twitter. In this way, some noisy input from the training set we provide the classification model may be alleviated by giving each word the meaning all users on Twitter can easily agree on.

After converting each tweet into a vector, we start clustering the tweets into three categories we discussed before. The cluster method we used is k-means. It can cluster most similar vectors together and make different vectors far away from each other. Based on the dimensionality of the vector space, 300, k-means performs well in time complexity.

When finishing clustering, each tweet will be assigned a index indicating which cluster it belongs to. Then, we feed these tweet into the classification model specific to that topic category.

As for classification part we trained a simple single layer CNN which is on top of the vectorized word embeddings. This model is a takes in the tweets as input, and cleans the tokenizes the tweets using the twokenize tweet tokenizer. The next step is the padding of the all the tweets in the data set. Due to the way a CNN is structured all the input data to the model needs to be of the same dimensional form. This is achieved by padding all the tweets to the length of the longest tweet in the dataset which was 38 tokens long. The character limit on twitter of 144 characters helps to keep in check the issues of loosing data information due to excessive padding. These are then suitably converted to matrices first in by just using a bag of words encoding. Let us look into the details of the model In many of the cases the choice of the hyper-parameters used and the implementation details there is an over whelming amount to choose from. The the very useful review of conducted by Nakov, Preslav, et al. . 2016.et all has served as a very helpful guide in the implementation of our model.

1) The first layer is the embedding layer which will be a matrix of dimensions n * k, where n is the length of the tweet and k is the length word vector representation of each word. Over this layer the next layer performs the convolution using different filter sizes. For the words not in the word2vec database a k dimensional vector is initialized with a randomized values. This is a very similar approach to the one used by Kim, Yoon. (2014).
2) We choose to use filter sizes of 3 ,4 and 5 words at a time which is used as a sliding window over the first layer. Note that the filter width, k is constrained to the full length of the word embedding. In case of each of the filters it generates a local image of the tweet matrix in the form of a matrix of size w * k, where w is the filter size. In our case we experimented with a few feature sizes and settled in on 128 features. A stride length of 1 was used for the model.
3) This is then passed on to an activation function to generate the feature maps. In our case we finally used the ReLU activation function after trying the ReLU and the tanh. The difference in the performance was not significant.
4) The next layer is the 1 max pooling layer. This uses the variable length feature maps and creates a univariate feature vector. These vectors are concatenated and they form the penultimate layer which maps to the output layer via a softmax function which gives

the probabilities for the output labels. This describes the acyclic map built in Tensor Flow for this specific CNN.

5) The next issue is of regularization in the neural network. Of the two common ways, L2norm constraints and neuron drop out, we can regularize a CNN, in our implementation we choose to use the neuron dropout method. We choose to use 0.5 as the drop out probabilities during training the network. This was motivated by the observations of Nakov, Preslav, et al. 2016.about L2 not leading to significant improvements in the model performance.

6) The next issue is of regularization in the neural network. Of the two common ways, L2norm constraints and neuron drop out, we can regularize a CNN, in our implementation we choose to use the neuron dropout method. We choose to use 0.5 as the drop out probabilities during training the network. This was motivated by the observations of Nakov, Preslav, et al. 2016.about L2 not leading to significant improvements in the model performance.

V. Evaluation, Metric and Experimental results

In order to see the performance of the model we design, we compare it with some other baseline models. First, we compare it with a couple of SVM classification models with different kernels. The kernels we use here are RBF kernel and linear kernel. Both of the SVM models used the tf-idf vectorization to form the vector for each tweet. Secondly, in order to testify the effect of clustering before the classification, we compare our model with the one without pre-clustering processing. This model classifies all the tweets in one step. Third, in order to see the effect of the size of dataset, we incorporate the dataset from previous years into the model, and use non-clustering classification method to see how does bigger size of data will affect the performance of Tensorflow network. This is because Tensorflow requires a relatively huge training dataset, but we can't provide that due to the given dataset from the Semeval. Therefore, we add more data, which may not have the same topics of this year's data and thus we don't know what does the topics look like in previous dataset. In this case, we can't use the cluster we train on this year's dataset to cluster the old dataset. But we can still compare the effect of data size.

In the result we can see the last three column in fig 2 is the model we built and it's overall performance is better than other models. The most improvement is accuracy. The non-clustering Tensorflow model has similar performance of SVM with linear kernel. This is because if we feed all the tweets into a classification without pre-processing, the model can't focus on building a boundary between similar topics which therefore is hard to classify and needs to be treated individually. But the model with linear svm has highest F1-score. It means our model still can't beat some of the baselines in terms of some evaluation metrics.

				Tensorflow-with-clustering		
	svm-rbf	svm-linear	Tensorflow-non-clustering	cluster 0	cluster 1	cluster 2
Precision	0.47	0.69	0.68	0.64	0.61	**0.94**
Accuracy	0.68	0.71	0.71	0.805	0.58	**0.88**
F-1 measu	0.47	**0.66**	0.58	0.53	0.58	0.53

Fig. 2. Result of different classifications

As we can see the result of each cluster in fig 2, the second cluster didn't perform as good as the other two. We don't know what are the tweets in this cluster, but if can improve the performance of it, the average of three clusters can get higher precision or accuracy.

	Tensorflow-with-clustering	Tensor-with-previous-data
Precision	0.73	0.68
Accuracy	0.755	0.73
F-1 measure	0.55	0.68

Fig. 3. Result of two Tensorflow models

In fig 3, we compare our model with the one training on both this year's and previous year's data. The Tensorflow with clustering means it only trains on this year's data. Tensorflow with previous data has both this year's and previous year's data. As we can see when the data grows, the F-1 score can easily beat the SVM.

	Tensorflow-with-clustering	Tensor-with-previous-data	SVM-rbf	svm-linear
Precision	**0.73**	0.68	0.47	0.69
Accuracy	**0.755**	0.73	0.68	0.71
F-1 measure	0.55	**0.68**	0.55	0.66

Fig. 4. Result of Tensorflow and SVM models with added data

To make it more convincing, we add previous year's data to svm model on both rbf and linear kernel.As we can see in fig 4, with more data, Tensorflow has a boost on F1-score, but the SVM models don't have much improvement. We conclude that, since our model has better performance than non-clustering one. And when data size grows, non-clustering Tensorflow model can beat normal classification models. Therefore, if we have more data which shares same topic scope of this year's data, our model will beat most of the classification models.

In a similar data set using the Semeval Twitter 2016 and the Twitter 2013 data for used by Vikrant et all 2016 , a direct comparison can be made between our model and the multilayer RNN used for binary classification.

Scoring Metric	Multilayer RNN	CNN
AvgF1	0.762	0.68
Accuracy	0.823	0.73

Fig. 5. A quick comparison of Multilayer RNN and CNN

VI. Conclusion

In this paper, we proposed a model for sentiment analysis with pre-clustering processing by using pre-trained word embedding for initialization of Doc2Vec to make Twitter-specific topics easier to cluster. The tweets belong to one cluster carry the similar document vectors which derived from the GloVe word embedding specifically focusing on Twitter data. Then when finishing clustering, we feed all the tweets into Tensorflow neural network, which is good at find the boundary between tweets that are difficult to distinguish. By giving each Tensorflow model similar tweets, the fine line can be found individually on each cluster. When combined together, the overall performance is better than the model

without pre-clustering. But the svm with linear kernel also has reasonable performance, especially the F1-score. Even though the first cluster and the last one have good performance in terms of precision and accuracy, the overall performance (shown in fig 3, second column) is not a great improvement, especially F1-score. But if we can give our model more tweets with the same topic scope collected for this year's data, we can claim that our model will have better performance than the baseline method we have. For the future work, we aim to distinguish which cluster has undesired results and then have more processing on that cluster. As we discussed, the second cluster has worst performance, and thus contaminate the overall result. By improving this one, we may be able to have larger improvement against our baseline methods.

APPENDICES

1) The CNN model training in takes a lot of time especially if it is run on the CPU only mode. A model with about 10000 tweets in total for a two class classification took about 15 hours to train.
2) One of the desired results of the experiments was getting the comparative performance of the word embeddings of bag of words, word2vec and the GloVE vectors but as the models took so much time to build and train we were not able to achieve the goal.
3) In the current state we have not put in place any mechanism to correct the spellings in the tweets or to find the stems of the misspelled words in the tweets. In the same vein the emoticons in the current model are treated as word characters not in the vocabulary.

REFERENCES

[1] HLTCOE, JHU. "SemEval-2013 Task 2: Sentiment Analysis in Twitter." Atlanta, Georgia, USA 312 (2013).

[2] Nakov, Preslav, et al. "SemEval-2016 task 4: Sentiment analysis in Twitter." Proceedings of the 10th international workshop on semantic evaluation (SemEval 2016), San Diego, US (forthcoming). 2016.

[3] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[4] Zhang, Ye, and Byron Wallace. "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification." arXiv preprint arXiv:1510.03820 (2015).

[5] Yadav, Vikrant. "thecerealkiller at SemEval-2016 Task 4: Deep learning based system for classifying sentiment of tweets on two point scale." Proceedings of SemEval (2016): 100-102.