Computational Intelligence Lab: Sentiment Analysis Project

Teymur Babayev babayevt@student.ethz.ch
Misels Kaporins miselsk@student.ethz.ch
Pius von Däniken piusv@student.ethz.ch
Team: greentshirt
Department of Computer Science, ETH Zürich, Switzerland

Spring 2016

Abstract

GloVe and similar word embeddings are useful to measure semantic similarities on the word level. We try to create sentence level features based on a clustering of word vectors and test whether they can help improve the accuracy on a text classification task. We find that this is not the case.

1 Introduction

1.1 Sentiment Analysis

In sentiment analysis we want to extract subjective information from text. It is usually formulated as a text classification task where we want to distinguish text sections by the attitude of the speaker or author. A sentence like *Just got my midterm and I'm impressed!* might be categorized as 'positive' whereas *Exams are coming soon.* might be considered 'negative'.

For our application we try to classify tweets into 'positive' and 'negative' categories based on whether the contain a happy (':)') or sad (':(') smiley.

Real time sentiment analysis of social media posts in general can be particularly useful for public relations and marketing purposes as it can provide insight into customer satisfaction and related metrics.

1.2 Baselines

Most modern state of the art text classification systems are based on convolutional neural networks. We will use a simple convolutional neural network similar to the one presented in [4] as a baseline for comparison.

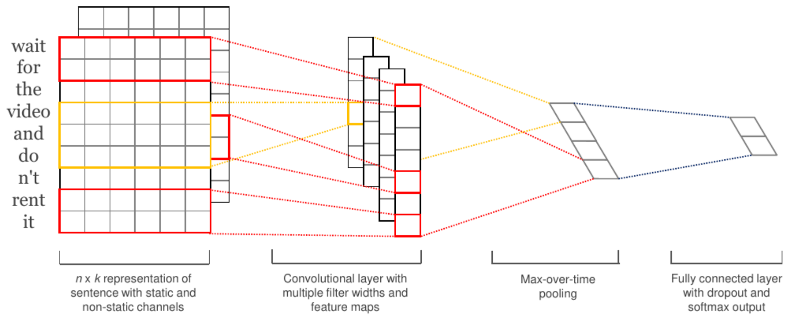
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: CNN Architecture For Text Classification [4]

As a second baseline we use a traditional feature based classifier. We extract tfidf (term frequency-inverse document frequency) features from the tweets and train a tree ensemble classifier using gradient boosting [3].

1.3 Contribution

For our main contribution we extract additional features based on a clustering of the vector space embedding of the words in a tweet. We first compute global vectors [6] for the words in our dataset and use simple k-means to cluster them.

2 Models and Implementation

2.1 Convolutional Neural Network

The basic structure of the CNN we use can be seen in fig. 1 which is taken directly from [4]. Every tweet will be split in a sequence of n words $(w_1, w_2, \ldots, w_n)$. For shorter tweets we pad the sequence with a special pad word w_{pad}. Every word w_i is associated with an embedding vector $\mathbf{x}_{w_i} \in \mathbb{R}^k$.

These embedding vectors can either be static and come from a pretrained embedding, such as *word2vec* [5] or *GloVe* [6], or be dynamically adapted during training.

Each tweet has therefore a representation $\mathbf{X} \in \mathbb{R}^{n \times k}$. Let $\mathbf{W} \in \mathbb{R}^{h \times k} h \leq n$ and $v_t = \sum_{i=t}^{t+h} \sum_{j=0}^{k} \mathbf{W}_{i,j} \mathbf{X}_{i,j}$ $\forall 1 \leq t \leq n-h+1$. The resulting vector $\mathbf{v}$ is the convolution of $\mathbf{X}$ and $\mathbf{W}$. Furthermore let $c_t = f(v_t + b)$ where f is some non-linear function and $b \in \mathbb{R}$ a bias term. The resulting vector $\mathbf{c} \in \mathbb{R}^{n-h+1}$ is called a feature map.

The feature map $\mathbf{c}$ is then maxpooled which means we select $\hat{c} = \max_t \{c_t\}$.

This process shows how we extract one feature $\hat{c}$ using one filter $\mathbf{W}$. We use several filters with possibly different heights h in parallel to get a feature vector $\hat{\mathbf{c}}$, which is then fed into a fully connected neural network to predict the correct label [4] [1].

We based our implementation of the described network on [1] and [2]. We chose filter lengths 3, 4, 5 and 6. For each filter length we used 128 different filters, resulting in a total of 512 features. We initialize the filter parameters and embedding vectors with gaussian random vectors. We use a dropout rate of 0.5 for regularization.

2.2 TF-IDF

This model is based on the bag of words assumption where each document is represented by the multiset of words it contains. *TF* stands for *term freqruency* and measures how often a certain word w appears in a specific document d. Given some document d of length n, $d = \{w_i\}_{i=1}^{n}$ we have $tf\,(w, d) = \frac{|\{w_i = w | 1 \leq i \leq n\}|}{n}$

IDF stands for the *inverse document frequency* where the *document frequency* measures in how many documents w appears at least once given some corpus of documents $\mathcal{C}$. $idf\,(w, \mathcal{C}) = \frac{|\mathcal{C}|}{|\{d \in \mathcal{C} | w \in d\}|}$.

In its simplest form the *TF-IDF* score of a word w given a document d and corpus $\mathcal{C}$ is $tfidf\,(w, d, \mathcal{C}) = tf\,(w, d)\,idf\,(w, \mathcal{C})$.

Using this we can build a sparse feature vector $\mathbf{x}_d \in \mathbb{R}^{|\mathcal{V}|}$ for some document d where $\mathcal{V}$ is the set of all words in the whole corpus $\mathcal{C}$.

We use these features to train a tree ensemble using gradient boosting [3].

2.3 GloVe Cluster Features

Methods such as *GloVe* [6] and *word2vec* [5] embed words in an euclidean vector space, suggesting that similar words are closer together in that space.

Assume we have an embedded vector $x_w \in \mathbb{R}^m$ for each word w in our vocabulary. We cluster these vectors using *k-means* and get a clustering $c : \mathbb{R}^m \to \{1, 2, 3, \ldots, k\}$. Given a document d of length n, $d = \{w_i\}_{i=1}^{n}$, we can construct the following features: $f_j = \frac{|\{w \in d | c(x_w) = j\}|}{n}$ for $1 \leq j \leq k$.

The features f_j represent the normalized histogram of words in the document belonging to a certain cluster j of words.

When training *GloVe* vectors we used an embedding dimension of 128 and trained the model for 10 epochs.

3 Experiments

3.1 Dataset

The dataset consists of a total of 2.5 million tweets of which half belong to the positive and half to the negative class. The validation set consists of 10000 unlabeled tweets.

We used the whole training data set to build the vocabulary, compute *GloVe* [6] vectors and train the convolutional neural network but only used 100000 samples of each class to train the ensemble classifiers.

3.2 Preprocessing

To prepare the data we first compute a mapping from words to integer ids. This allows us to operate on sequences of integers instead of sequences of strings. In order to reduce the size of the vocabulary we remove stopwords using a list of stopwords. Furthermore we stem all words and filter words that only appear rarely.

3.3 Results

Method	Classification Accuracy in %
CNN	76.76%
tfidf	81.5%
GloVe cluster histograms	61.74%
tfidf & clusters	81.86%
tfidf & clusters[1]	82.22%

Table 1: Different Models Evaluated On The Kaggle Public Leaderboard

Table 1 shows the classification accuracy of the different models on the validation set as reported by the public leaderboard of the Kaggle competition.

We can clearly see that the GloVe cluster histograms bring no significant improvement over the simple tfidf features. Moreover they perform significantly worse when used on their own.

Surprisingly the convolutional neural network performs worse than the tfidf model as well.

A possible reason for the cluster histogram features to perform this badly is that a lot of important antonyms fall into the same cluster. For example the words ‘happy’, ‘sad’, ‘good’, ‘bad’, ‘excited’ tend to consistently fall into the same cluster. Increasing the number of clusters does not help since even when we increase the number of clusters to 1000 usually less than 100 contain more than a single word.

References

[1] Denny Britz. Implementing a cnn for text classification in tensorflow. `http://www.wildml.com/2015/12/`

[1] using the same features but 3000 instead of 1000 trees in the ensemble

`implementing-a-cnn-for-text-classification-in-tensorflow/`, 2015. last accessed 2016-06-30.

[2] Denny Britz. cnn-text-classification-tf. `https://github.com/dennybritz/cnn-text-classification-tf`, 2016. last accessed 2016-06-30.

[3] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[4] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[5] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.