UNIMELB at SemEval-2016 Tasks 4A and 4B: An Ensemble of Neural Networks and a Word2Vec Based Model for Sentiment Classification

XingYi Xu   HuiZhi Liang   Timothy Baldwin
Department of Computing and Information Systems
The University of Melbourne VIC 3010, Australia
stevenxxiu@gmail.com oklianghuizi@gmail.com tb@ldwin.net

Abstract

This paper describes our sentiment classification system for microblog-sized documents, and documents where a topic is present. The system consists of a soft-voting ensemble of a `word2vec` language model adapted to classification, a convolutional neural network (CNN), and a long-short term memory network (LSTM). Our main contribution consists of a way to introduce topic information into this model, by concatenating a topic embedding, consisting of the averaged word embedding for that topic, to each word embedding vector in our neural networks. When we apply our models to SemEval 2016 Task 4 subtasks A and B, we demonstrate that the ensemble performed better than any single classifier, and our method of including topic information achieves a substantial performance gain. According to results on the official test sets, our model ranked 3rd for F^{PN}in the message-only subtask A (among 34 teams) and 1st for accuracy on the topic-dependent subtask B (among 19 teams).

1 Introduction

The rapid growth of user-generated content, much of which is sentiment-laden, has fueled an interest in sentiment analysis (Pang and Lee, 2008; Liu, 2010). One popular form of sentiment analysis involves classifying a document into discrete classes, depending on whether it expresses positive or negative sentiment (or neither). The classification can also be dependent upon a particular topic. In this work, we describe the method we used for the sentiment classification of tweets, with or without a topic.

Our approach to the document classification task consists of an ensemble of 3 classifiers via soft-voting, 2 of which are neural network models. One is the convolutional neural network (CNN) architecture of Kim (2014), and another is a Long Short Term Memory (LSTM)-based network (Hochreiter and Schmidhuber, 1997). Both were first tuned on a distant-labelled data set. The third classifier adapted `word2vec` to output classification probabilities using Bayes' formula, a slightly modified version of Taddy (2015). Despite the `word2vec` classifier being intended as a baseline, and having a small weight in the ensemble, it proved crucial for the ensemble to work well. To adapt our models to the case where a topic is present, in the neural network models, we concatenated the embedding vectors for each word with a topic embedding, which consisted of the element-wise average of all word vectors in a particular topic.

We applied our approach to SemEval 2016 Task 4, including the message-only subtask (Task A) and the topic-dependent subtask (Task B) (Nakov et al., to appear). Our model ranked third for F^{PN}in the message-only subtask A (among 34 teams) and first for accuracy[1] on the topic-dependent subtask B (among 19 teams).

[1]There were some issues surrounding the evaluation metrics. We only got 7th for $\rho^{\mathrm{PN}}$and 2nd for F^{PN}officially, but when we retrained our model using ρ^{PN}as the subtask intended, we place first across all metrics.

The source code for our approach is available at `https://github.com/stevenxxiu/senti`.

2 Models

We now describe the classifiers we used in detail, our ensemble method, and our motivations behind choosing these classifiers.

2.1 Convolutional neural network

We used the dynamic architecture of Kim (2014) for our convolutional neural network. This consists of a single 1-d convolution layer with a non-linearity, a max-pooling layer, a dropout layer, and a softmax classification layer.

This model was chosen since it was a good performer empirically. However, due to max-pooling, this model is essentially a bag-of-phrases model, which ignores important ordering information if the tweet contains a long argument, or 2 sentences. We now give a review of the layers used.

2.1.1 Word embedding layer

The input to the model is a document, treated as a sequence of words. Each word can possibly be represented by a vector of occurrences, a vector of counts, or a vector of features. A vector of learnt, instead of hand-crafted features, is also called a word embedding. This tends to have dimensionality $d \ll |V|$, the vocabulary size. Hence the vectors are dense, which allows us to learn more functions of features with limited data.

Given an embedding of dimension d, $W_{\text{emb}} \in \mathbf{R}^{d\times|V|}$, we mapped map each document s to a matrix $W_{\text{emb},s} \in \mathbf{R}^{d\times|s|}$, with each word corresponding to a row vector in the order they appear in. W_{emb} can be trained.

2.1.2 1-d convolution layer

A 1-d convolution layer aims to extract patterns useful for classification, by sliding a fixed-length filter along the input. The convolution operation for an input matrix $S \in \mathbf{R}^{d\times|s|}$ and a single filter $F \in \mathbf{R}^{d\times m}$ of width m creates a feature $y_{\text{conv}} \in \mathbf{R}^{|s|+m-1}$ by:

$$y_{\text{conv},i} = \sum_{k,j} (S_{[:,i:i+m-1]} \odot F)_{k,j} + b_{\text{conv}}$$

where $\odot$ is element-wise multiplication, and b_{conv} is a bias. There are typically $n > 1$ filters, which by stacking the feature vectors, results in $Y_{\text{conv}} \in \mathbf{R}^{n\times(|s|+m-1)}$. Each filter has its own separate bias.

We used a common modification to the filter sliding, by padding the document embedding matrix with $m-1$ zeroes on its top and bottom. This is done so that every word in the document is covered by m filters.

2.1.3 Max pooling layer

There may be very few phrases targeted by a feature map in the document. For this reason, we only need to know if the desired feature is present in the document, which can be obtained by taking the maximum. Formally, we obtain a vector $d \in \mathbf{R}^n$, such that:

$$y_{\text{pool},i} = \max_j Y_{\text{conv},i,j}$$

2.1.4 Softmax layer

To convert our features into classification probabilities, we first use a dense layer, defined by:

$$y_{\text{dense}} = W_{\text{dense}} \cdot y_{\text{pool}} + b_{\text{dense}}$$

with a softmax activation function:

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

such that the output dimension is the same as the number of classes. Note that output values are non-negative and sum to 1, which form a discrete probability distribution.

2.1.5 Regularization

To regularize our CNN model, dropout (Srivastava et al., 2014) is used after the max pooling layer. Intuitively, dropout assumes that we can still obtain a reasonable classification even when some of the features are dropped. To do this, each dimension is randomly set to 0 using a Bernoulli distribution $B(p)$. In order to have the training and testing to be of the same order, the test outputs can be scaled by p.

The softmax layer for the CNN model also uses a form of empirical Bayes regularization, where each row of W_{soft} is restricted using an ℓ_2

norm, by re-normalizing the vector if the norm threshold is exceeded.

2.2 Long short term memory network

We used an LSTM for our recurrent architecture, which consisted of an embedding layer, LSTM layer, and a softmax classification layer.

A recurrent neural network is a neural network designed for sequential problems. Even simple RNNs are Turing complete, and they can theoretically obtain information from the entire sequence instead of only an unordered bag of phrases. But finding good architectures which can capture this and training them can be difficult. Indeed, there were many instances where our LSTM failed to capture important ordering information. We now give a brief review of the LSTM.

Given an input sequence $x = [x_1, \ldots, x_T]$, a recurrent network defines an internal state function f_c and an output function f_y to iterate over x, so that at time step t:

$$c_t = f_c(x_t, y_{t-1}, c_{t-1})$$
$$y_t = f_y(x_t, y_{t-1}, c_t)$$

where c_0 and y_0 are initial bias states. The simplest RNN, where:

$$f_y(x_t, y_{t-1}, c_t) \quad = \quad \tanh(W \cdot \begin{bmatrix} x_t \\ y_{t-1} \end{bmatrix})$$

suffers from the gradient vanishing and exploding problem (Hochreiter and Schmidhuber, 1997). In particular, products of saturated tanh activations can vanish the gradient, and products of W can vanish or explode the gradient.

The LSTM is a way to remedy this (Hochreiter and Schmidhuber, 1997). It sets:

$$\begin{aligned} i_t &= \sigma_i(W_{xi}x_t + W_{hi}h_{t-1} + w_{ci} \odot c_{t-1} + b_i) \\ f_t &= \sigma_f(W_{xf}x_t + W_{hf}h_{t-1} + w_{cf} \odot c_{t-1} + b_f) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \sigma_c(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ y_t &= \sigma_o(W_{xo}x_t + W_{ho}h_{t-1} + w_{co} \odot c_t + b_o) \odot \sigma_h(c_t) \end{aligned}$$

Here, $W_\bullet$ is made up of weights; i_t, f_t, c_t, y_t are the input gates, forget gates, cell states and output states. Cell states have an identity activation function. The gradient will not vanish if input x_i needs to be carried to x_j, since this can only happen when the forget gates are near 1. However, gradient explosion can still be present. We use the common approach of cutting off gradients above a threshold for the gradients inside $\sigma_i, \sigma_f, \sigma_c, \sigma_o$.

To use the LSTM, we first used a document embedding matrix in the same manner as our convolutional neural network architecture. This was fed into to an LSTM layer. The output for the final timestep of the LSTM layer was then fed into a final softmax layer with the appropriate output size for classification.

We also experimented with a similar and commonly used network, the Gated Recurrent Network (GRU), but it was not used due to lower results compared to the LSTM.

2.3 `word2vec` Bayes

Our `word2vec` Bayes model is our baseline model, and described in Taddy (2015), with the inclusion of a class prior. Taddy (2015) uses Bayes formula to compute the probabilities of a document belonging to a sentiment class. Given a document d, its words $\{w\}_i$, label y, Bayes formula is:

$$p(y|d) = \frac{p(d|y)p(y)}{p(d)}$$

For classification problems, we can ignore $p(d)$ since d is fixed. $p(d|y)$ is estimated by first training `word2vec` on a subset of the corpus with label y, then using the skipgram objective composite likelihood as an approximation:

$$\log p(d|y) \approx \sum_{s\in d} \sum_{j=1}^{|s|} \sum_{k=1}^{|s|} 1_{1\le|k-j|\le b} \log p(w_k|w_j, y)$$

We estimated $p(y)$ via class frequencies, i.e. the MLE for the categorical distribution, compared to Taddy (2015), who used the discrete uniform prior.

This model was chosen since it provides a reasonable baseline, and also appears to be independent enough from our neural networks to provide a performance gain in the ensemble. The `word2vec` based model benefits from being a semi-supervised method, but it also loses ordering information outside a word's context window, and the prediction of neighboring words also ignores distance to that word. The limited amount of data for each class is also an issue.

2.4 Ensemble

If the errors made by each classifier are independent enough, then combining them in an ensemble can reduce the overall error rate. We used soft voting as a method to combine the outputs of the above classifiers. We define soft voting as:

$$y_{\text{vote}} = \sum_i w_i y_i, \text{ s.t. } \sum_i w_i = 1, \forall i : w_i \geq 0$$

where y_i is the output of classifier i.

3 Topic dependent models

To adapt our neural networks to topic-dependent sentiment classification, in our neural network models, we augmented each embedding vector by concatenating it with a topic embedding. The motivation behind this approach is to allow the model to interpret each word to be within the context of some topic.

Our topic embeddings were obtained by the element-wise average of word embeddings for each word in that topic. We found that empirically, this is a simple yet effective way of achieving a document embedding. When used directly as a feature vector in logistic regression for sentiment analysis, we have found this to outperform methods described in Le and Mikolov (2014).

Word embeddings of dimension $d = 300$ pretrained over Google News were used directly, without any further tuning. Words in the tweet which were not present in this pretrained embedding were ignored.

4 Experiments and evaluation

We evaluated our models on SemEval 2016 Task 4 subtask A, the message-only subtask, and subtask B, the topic-dependent subtask.

4.1 Data

Task A consisted of 3 sentiment classes — POSITIVE, NEUTRAL and NEGATIVE — whilst Task B consisted of 2 sentiment classes — POSITIVE and NEGATIVE. We only managed to download 90% of the entire set of tweets for the 2016 SemEval data, due to tweets becoming "not available". In addition to the Twitter data for 2016, for Task A we also used training data from SemEval 2016 Task 10. The data is summarized in Table 1.

To pretrain our network using distant learning (described below), we took a random sample of 10M English tweets from a 5.3TB Twitter dataset crawled from 18 June to 4 Dec, 2014 using the Twitter Trending API. We then processed tweets with a regular expression: tweets which contained emoticons like *:)* were considered POSITIVE, while those which contained emoticons like *:(* were considered NEGATIVE; tweets which contained both positive and negative emoticons, or others emoticons such as *:—*, were ignored. We extracted 1M tweets each for the POSITIVE and NEGATIVE classes.

4.2 Evaluation

Evaluation consisted of accuracy, macro-averaged F_1 across the POSITIVE and NEGATIVE classes, which we denote F^{PN}, and macro-averaged recall across the POSITIVE and NEGATIVE classes, which we denote ρ^{PN}.

4.3 Preprocessing

All methods use the same preprocessing. We normalized the tweets by first replacing URLs with *_url* and author methods such as *@Ladiibubblezz* with *_author*. Casing was preserved, as the pretrained `word2vec` vectors included casing. The tweets were tokenized using `twokenize`, with *it's* being split into *it* and *'s*.

4.4 Training and hyperparameters

4.4.1 Neural networks

For both our models, we initialized W_{emb} to word embeddings pretrained using `word2vec`'s skip-gram model (Mikolov et al., 2013) on the Google News corpus, where $d = 300$. Unknown words were drawn from $U[-0.25, 0.25]$ to match the variance of the pretrained vectors. For the CNN model, we also stripped words so all documents had a length ≤ 56.

Datset	A total	A used	B total	B used
Twitter 2016-train	6000	5465 (+11340)	4346	3941
Twitter 2016-dev	2000	1829	1325	1210
Twitter 2016-test	2000	1807	1417	1270

Table 1: Semeval-2016 data. The NEGATIVE : NEUTRAL : POSITIVE split was 16 : 42 : 42 for all of 2016 Task A used. The NEGATIVE : POSITIVE split was 19 : 81 for all of 2016 Task B used.

For our CNN, we used used 3 1-d convolutions, with filter sizes of $3, 4, 5$, each with 100 filters. Our dropout rate was 0.5, and our ℓ_2 norm restriction was 3. For our LSTM, we used a cell dimension size of 300, and our activations were chosen empirically to be $\sigma_i = \sigma_f = \sigma_o =$ sigmoid, $\sigma_c =$ tanh, our gradient cutoff was 100.

To train our neural networks, we used cross entropy loss and minibatch gradient descent with a batch size of 64. For our CNN, we used the adadelta (Zeiler, 2012) gradient descent method with the default settings. For our LSTM, we used the rmsprop gradient descent method with a learning rate of 0.01.

Due to limited training data, we can use distant learning (Severyn and Moschitti, 2015), by initializing the weights of our neural networks by first training them on a silver standard data set (generated using Twitter emoticons which we describe below), then tuning them further on the gold standard data set (Severyn and Moschitti, 2015). However, we did not use this for our topic-dependent models, as there was no performance gain.

We split the distant data into 10^4 tweets per epoch, and took the best epoch on the validation set as the initial weights, using F^{PN}as our scoring metric. We repeatedly iterated over the SemEval data with 10^3 tweets per epoch for 10^2 epochs, and again took the best epoch on the validation set as the final weights.

4.4.2 `word2vec` Bayes

Gensim (Řehůřek and Sojka, 2010) was used to train the word embeddings and obtain $p(d|y)$. We used the skipgram objective, with a embedding dimension of 100, window size of 10, hierarchical softmax with 5 samples, 20 training iterations, no frequent word cutoff, and a sample coefficient of 0.

Dataset	F^{PN}
Twitter 2013	0.687_7
Twitter 2014	0.706_7
Twitter 2015	0.650_4
Twitter 2016	0.617_3
Twitter Sarcasm 2014	0.449_{11}
SMS 2013	0.593_{10}
LiveJournal 2014	0.683_9

Table 2: Official test scores and ranks for Task A.

Dataset	Val metric	ρ^{PN}	F^{PN}	Acc
Twitter 2016	F^{PN}	0.758_7	0.788_2	0.870_1
Twitter 2016	ρ^{PN}	0.807_1	0.806_1	0.867_1

Table 3: Test scores and ranks for Task B. The official run incorrectly used F^{PN}as the validation metric.

4.4.3 Soft voting

To find w_i, we first relaxed the sum condition of w_i by setting $w_1 = 1$ and noting that $\max_k y_{vote,i,k}$ is invariant under scaling. We then used the `L-BFGS-B` algorithm, with initial weights of 1, combined with basin-hopping for 1000 iterations. We optimized for accuracy, since this most-consistently improved results.

5 Results

The official evaluation results are shown in Table 2 and Table 3. The results for Task A suggest that our models are overfitting. Our best position was achieved on the Twitter 2016 dataset, and indeed, this is what our parameters were chosen on.

Our own evaluation of our different classifiers, using Twitter 2016-test, is shown in Table 4 and

Model	Accuracy	F^{PN}
soft voting all	0.5772	0.6000
lstm	0.5379	0.5869
soft voting cnn + lstm	0.5606	0.5848
cnn	0.5612	0.5841
`word2vec` Bayes	0.5130	0.4983

Table 4: Results for Task A, sorted by F^{PN}.

Model	Accuracy	ρ^{PN}
soft voting all	0.8008	0.7849
soft voting cnn + lstm	0.7976	0.7846
cnn topic	0.8047	0.7762
lstm topic	0.6756	0.7494
cnn	0.8354	0.7253
`word2vec` Bayes	0.7654	0.7138
lstm	0.8118	0.6916

Table 5: Results for Task B, sorted by ρ^{PN}. The dashed line separates topic models from message-only models. The lstm topic model has poorer accuracy due to being optimized on ρ^{PN}.

Table 5. Taking into account all evaluation metrics, we can see that in both tasks, our CNN outperforms our LSTM, in Task A slightly and in Task B substantially. The `word2vec` Bayes model is worse than both, moreso in Task B.

Soft voting outperforms all classifiers, showing that there is some independence amongst the errors made. In Task A, there appears to be more correlation between the CNN and LSTM classifiers, as excluding the `word2vec` Bayes model reduces the performance. In Task B, the `word2vec` Bayes model appears to perform too poorly to provide a marked benefit.

From Table 5 we can see that the inclusion of topic information provides a substantial boost to both of our neural networks. This shows that our method of incorporating topic information is a useful way of modifying neural networks, and provides a strong baseline for alternative ways of doing this.

6 Conclusions

We described our ensemble approach to sentiment analysis both the task of topic-dependent document classification and document classification by itself. We gave a detailed description of how to modify our classifiers to be topic-dependent. The results show that ensembles can work for neural nets, and that our way of including topics achieves performance gains, and forms a good basis for future research in this area.

References

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Kim2014] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar.

[Le and Mikolov2014] Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *arXiv preprint arXiv:1405.4053*.

[Liu2010] Bing Liu. 2010. Sentiment analysis and subjectivity. In Nitin Indurkhya and Fred J. Damerau, editors, *Handbook of Natural Language Processing*. Chapman & Hall/CRC, Boca Raton, USA, 2nd edition.

[Mikolov et al.2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Nakov et al.to appear] Preslav Nakov, Alan Ritter, Sara Rosenthal, Veselin Stoyanov, and Fabrizio Sebastiani. to appear. SemEval-2016 Task 4: Sentiment analysis in Twitter. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, San Diego, USA.

[Pang and Lee2008] Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135.

[Řehůřek and Sojka2010] Radim Řehůřek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

[Severyn and Moschitti2015] Aliaksei Severyn and Alessandro Moschitti. 2015. UNITN: Training deep convolutional neural network for Twitter sentiment classification. In *Proceedings of the 9th*

International Workshop on Semantic Evaluation (SemEval), pages 464–469, Denver, USA.

[Srivastava et al.2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Taddy2015] Matt Taddy. 2015. Document classification by inversion of distributed language representations. In *arXiv:1504.07295 [cs, stat]*.

[Zeiler2012] Matthew D. Zeiler. 2012. ADADELTA: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.