HW1: Classification

Shangyan Li
shangyanli@college.harvard.edu

Kevin Liu
kevinliu01@college.harvard.edu

Artidoro Pagnoni
apagnoni@college.harvard.edu

February 1, 2018

1 Introduction

Text classification is a classic topic in natural language processing, and consists of assigning predefined labels to a series of variable-length texts. In this project we compare the performance of several varieties of text classifiers. We explore both classical machine learning approaches using a Naive Bayes unigram classifier (Wang and Manning (2012)) and a logistic regression model over word types, as well as neural network approaches using continuous bag-of-word (Mikolov et al. (2013)) with embeddings, convolutional neural networks (Kim (2014)) as well as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber (1997)). We also show that meta learning techniques such as ensembling can improve final prediction results.

2 Problem Description

The problem of interest in the present work is predicting the sentiment of reviews in the Stanford Sentiment Treebank 2 (SST2) dataset. This is a binary classification task with labels 1 and 2 corresponding to positive and negative respectively. In the SST2 dataset we are given input sentences $\boldsymbol{X}$ and labels y. The input sentences have varying lengths and are composed of words from a vocabulary $\mathcal{V}$ containing 16284 words in total. The label set of the language can only take two values $\mathcal{T} = \{1, 2\}$. Our task is to use the training data $\{(\boldsymbol{X}, y)\}^n$ to predict the labels of unseen sentences.

We propose seven models for this task that we optimize through likelihood maximization MLE. The first two models Naive Bayes and Logistic Regression use a bag-of-words representation. In particular, Naive Bayes uses a binarized bag-of-words representations of the sentence $\boldsymbol{X}$, where $\boldsymbol{x} \in \{0,1\}^{\|\mathcal{V}\|}$. Each entry in the vector x is 1 if the corresponding word appears at least once in the sentence, otherwise it is 0. Logistic Regression instead uses the regular bag-of-words representation $\boldsymbol{x} \in \mathbb{N}^{\|\mathcal{V}\|}$ with word counts at each entry. To represent the weights associated with each word $\boldsymbol{x}_i$, we use a one-column matrix $\boldsymbol{W} \in \mathbb{R}^{\|\mathcal{V}\| \times 1}$.

On the other hand, the deep learning models rely on pretrained word embeddings. We have experimented both with fastText and Glove embeddings, finally opting for Glove which gave better validation results. We use an embedding matrix $W \in \mathbb{R}^{\|\mathcal{V} \times d\|}$, where d is the embedding vector length for each word. We use the pre-trained word embeddings for all models, with embedding dimension $d = 300$.

3 Models and Algorithms

We explored statistical and deep learning based models and in total we report seven models for this task.

3.1 Naive Bayes

As a non deep learning based model and baseline for this task we use Naive Bayes as described in Wang and Manning (2012). Here we model the likelihood of the positive class as:

$$y_i = \text{sigmoid}(\boldsymbol{w}^T \boldsymbol{X} + b)$$

As explained earlier the representation $\boldsymbol{X}$ used for sentences is the binarized bag-of-words model. A closed form expression can be derived by MLE for parameters $\boldsymbol{w}$ and $\boldsymbol{b}$. We define the count vectors $\boldsymbol{p} = \alpha + \sum_{i:y^{(i)}=1} \boldsymbol{x}^{(i)}$ and $\boldsymbol{q} = \alpha + \sum_{i:y^{(i)}=2} \boldsymbol{x}^{(i)}$ where α is a smoothing parameter and $\boldsymbol{x}^{(i)}, y^{(i)}$ are the sentence representation and label for data point i in the dataset. Then the parameter $\boldsymbol{w}$ is equal to the log-count ratio:

$$\boldsymbol{w} = \log\left(\frac{\boldsymbol{p}/\|\boldsymbol{p}\|_1}{\boldsymbol{p}/\|\boldsymbol{p}\|_1}\right)$$

And the parameter $b = \log(N_+/N_-)$ where N_+ and N_- are respectively the number of positive and negative training cases.

3.2 Logistic Regression

In logistic regression, we try to predict the probability that a given example belongs to the "1" class as opposed to the "0" class. Given a sentence $\boldsymbol{X} \in \mathbb{R}^{n \times \|\mathcal{V}\|}$ consisting of words $\boldsymbol{x}_{1:n}$ and label y, we try to learn a function of the form $P(y = 1|x) = \sigma(\sum_i \boldsymbol{W}\boldsymbol{x}_i + b)$, where the sigmoid function squashes the value of the linear output $\sum_i \boldsymbol{W}\boldsymbol{x}_i + b$ into the range [0, 1]. In order to learn the model, we take advantage of the convexity of the binary cross entropy loss function (with $\hat{y}$ being the model prediction)

$$-\sum_i (y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

and use gradient descent to optimize the model weights. In practice, our PyTorch module outputs only the linear result, and uses `BCEWithLogitsLoss`, which combines a *Sigmoid* layer and the *BCELoss* in one single class (This approach proved to be more numerically stable than using the two in separate layers by taking advantage of the log-sum-exp trick).

3.3 Continuous Bag-of-Words

We use a variant of the CBOW technique proposed in Mikolov et al. (2013), based on the intuition that words can have multiple degrees of similarity that can be captured in continuously distributed, d-dimensional vectors. Instead of using CBOW to predict a current word, as described in the original paper, we simply leverage the pre-trained embeddings, summing over the sentence's word vectors and feeding the result through a final fully connected layer.

3.4 Convolutional Neural Network (1, 2 layers)

CNN 1, our single layer CNN, is an attempt to replicate the results presented in Kim (2014). As in the CNN-multichannel presented in the paper, we maintain two sets of 300 dimensional word vectors, only one of which gets gradients backpropagated through during training. The convolutional layer consists of 100 channels each of filters of kernel sizes 3, 4, 5. We use max pooling over time and ReLU activations. This is followed by a single fully-connected layer and a sigmoid output. For regularization, we also employ dropout on the final fully-connected layer, a cap of 3 on the weights of that layer, and early stopping. Differences include initializing word vectors with GloVe as opposed to word2vec, training with mini-batch size of 10, and using Adamax as opposed to Adadelta.

CNN 2, our two layer CNN, is an extension of CNN 1. The motivation behind this extension was that having deeper convolutional layers would widen the receptive field for learnable features in the convolutional layers. Intuitively, the hope is that the first convolutional layer followed by pooling would aggregate information among groups of contiguous words, and the second convolutional layer may learn higher level information about the sentence for the sentiment analysis task. This model maintains the multichannel property (two sets word vectors, only one of which is trained). The first convolutional layer contains 32 channels each of filters of kernel sizes 3, 4, 5, followed by a max pooling layer of kernel size 3 and ReLU activation. The second convolutional layer contains 64 channels each of filters of kernel size 3, and is followed by ReLU and max pooling over time. Finally, we again have a fully connected layer and sigmoid output.

3.5 Long Short-Term Memory Network (1, 2 layers)

Following the motivation behind the use of CNN in Kim (2014), we experimented with LSTM models. Word embeddings are able to capture word specific characteristics and provide great representations for the words, however they lack the ability to harness the context in which they appear in the sentence. Our objective using CNNs and LSTMs was to produce word representations that were able to take into account the greater context of the sentence, and have a large receptive field. LSTMs have been used in many language applications such as speech recognition, and language modeling, and have demonstrated that they can produce representations that are dependent on their context. Others have successfully experimented with the use of LSTM in the context of sentiment classification as in Qian et al. (2017) which further supports our intuition in using this model.

We propose two LSTM models LSTM 1 and LSTM 2, which respectively are 1 and 2 layers LSTMs. To obtain dependence from both past and future words in the sentence we employ bidirectional LSTMs outputting 200 dimensional hidden representations of the words $\boldsymbol{h}_i$. The variable number of word representations $\boldsymbol{h}_i$ is then combined into a single 200 dimensional vector $\hat{\boldsymbol{h}}_i$ with a max pooling overtime layer and ReLU activation. This is again followed by a single fully-connected layer and a sigmoid output. For regularization, we follow Kim (2014) and use dropout on the final fully-connected layer, a cap of 3 on the weights of that layer, and early stopping. Training is done with Adamax.

3.6 LSTM-CNN

The LSTM-CNN model is a combination of our basic single layer LSTM model and our CNN model. Intuitively, convolutions captures local contiguous group structure while LSTMs seem to maintain a more global memory. We combine these in hopes of capturing the advantages of both. LSTMs and CNNs have different approaches to increasing the receptive fields of the word representations, and we explored whether combining these approaches also brought improved representations. The model is composed of a single convolution layer consisting of 64 channels and three different kernel sizes of 3,4,5 on top of a bidirectional LSTM layer with hidden state size 64. This is followed by max pooling over time, a fully connected layer, and sigmoid output. We use similar training and regularization methods as above.

3.7 Ensemble

The final Kaggle submission was an ensemble of the Naive Bayes, CNN 1, and LSTM 1 models, taking the majority vote of the three predictions provided by those models. Ensembling helps to reduce variance and combines the benefits of the various models involved. This achieved test accuracy of 0.85557. Since, the dataset was relatively small, the models for this final submission were trained on both the train and validation sets.

4 Experiments

Table 1 shows the training, validation, and test accuracies of the aforementioned models.

Model	Epochs	Train Acc.	Validation Acc.	Test Acc.
MNB	NA	0.94812	0.79243	0.82482
LR	500	0.99768	0.74197	0.77539
CBoW	50	1.0	0.77752	0.79846
CNN 1	10	0.99985	0.82339	0.83415
CNN 2	5	0.99739	0.82339	0.83745
LSTM 1	5	0.98627	0.81307	0.83470
LSTM 2	5	0.94320	0.81995	0.83800
LSTM-CNN	5	0.97745	0.82224	0.83964
ENSEMBLE	NA	NA	NA	0.85557

Table 1: Table with the main results.

5 Conclusion

This work involved implementing and experimenting with statistical and deep learning models (Naive Bayes, Logistic Regression, CBoW, CNN, LSTM) for text classification applied to the SST2 dataset for sentiment analysis. We find that the simplest model of Naive Bayes performs quite well, considerably better than logsitic regression and CBoW with pretrained word embeddings.

This is likely because the dataset was relatively small, and naive bayes tends to be very data efficient. However, as logistic regression and CBoW are more flexible models, we would expect them to perform better with additional data. Also, we achieved better results with more complex neural network architectures, namely CNNs and LSTMs. We believe this was due to these models being able to consider groups of words and larger receptive fields. Our final Kaggle submission was an ensemble model and achieved test accuracy of 0.85557.

A notable challenge was managing overfitting in the larger neural network models. Since the dataset was relatively small (6920 train, 880 val.), it was easy to overfit these models. Following Kim (2014), we employed a combination of dropout, early stopping, and capping the norm of some parameter weights. Additionally, in our extensions, we reduced hidden layer sizes as we made our models deeper and observed clear overfitting with a few training epochs.

Potential extensions include incorporating attention into our RNN models, and running these models on larger datasets to compare their performance with more data.

References

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Qian, Q., Huang, M., Lei, J., and Zhu, X. (2017). Linguistically regularized LSTM for sentiment classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1679–1689.

Wang, S. I. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 2: Short Papers*, pages 90–94. The Association for Computer Linguistics.