HW0: Classification

Noah Golowich and Jesse Zhang

February 1, 2018

1 Introduction

In this assignment we experiment with different techniques for sentence sentiment classification. We use the Stanford Sentiment Treebank dataset Socher et al. (2013), which consists of sentences ,each of which is labeled either "positive" or "negative" sentiment. We use a Naive Bayes unigram classifier, a logistic regression model, a continuous bag of words model (using the FastText embeddings Bojanowski et al. (2016)), and a convolutional neural network.

In addition to the four basic models issued in the assignment, we also applied our own extensions, in the form of variations on the CNN, which we will detail below.

2 Problem Description

Let $\mathcal{V}$ denote the vocabulary and $\mathcal{T} = \{\text{pos}, \text{neg}\}$ denote the possible labels that can be given to a sentence. We model each label $y^j \in \{0,1\}$. At times it will be more convenient for us to write $y^j \in \{-1,1\}$; we will specify when we do so. We are given a collection of sentences $\mathcal{S} = \{S^1, \dots, S^N\}$, where each S^j consists of an ordered tuple $(\boldsymbol{x}_1^j, \dots, \boldsymbol{x}_{\ell(j)}^j)$ of words. Here $\boldsymbol{x}_i^j \in \{0,1\}^{|\mathcal{V}|}$ is a one-hot encoding of the i-th word in the j-th sentence. Corresponding to each S^j is a label $y^j \in \mathcal{T}$. $\ell(j)$ denotes the length of sentence j, $1 \le j \le N$; sentences can have varying lengths. Our goal is to predict y^j given S^j.

3 Model and Algorithms

3.1 Naive Bayes

In this section we describe our implementation of the Multinomial Naive Bayes algorithm from Wang and Manning (2012). We will denote the labels $y^j \in \{-1,1\}$ to follow the notation of Wang and Manning (2012). We represent a sentence $(\boldsymbol{x}_1^j, \dots, \boldsymbol{x}_\ell^j)$ as a *bag-of-words* $\boldsymbol{x}^j = \sum_{i=1}^{\ell} \boldsymbol{x}_i^j \in \{0,1\}^{|\mathcal{V}|}$. For $1 \le v \le |\mathcal{V}|$, the v-th feature $(x^j)_v$ represents how many times word v appears in sentence j.

To define the prediction rule, let α be the vector of all 1's, and define the feature count vectors $\boldsymbol{p} = \alpha + \sum_{j:y^j=1} \boldsymbol{x}^j$, and $\boldsymbol{q} = \alpha + \sum_{j:y^j=-1} \boldsymbol{x}^j$. Also let $b = \log(|\{j : y^j = 1\} / \{j : y^j = -1\}|)$ be the logarithm of the number of positive examples divided by the number of negative examples.

Then we form the prediction

$$f(\mathbf{x}^j) = \text{sign}(\langle \boldsymbol{w}, \boldsymbol{x}^j \rangle + b),$$

where $\boldsymbol{w} = \log\left(\frac{p/\|p\|_1}{q/\|q\|_1}\right)$, where the division and logarithm are both taken element-wise.

3.1.1 Set-of-words

Wang and Manning (2012) report that instead modeling $(\boldsymbol{x}_1^j, \ldots, \boldsymbol{x}_\ell^j)$ as a *set-of-words* $\boldsymbol{x}^j = \mathbb{1}[\sum_{i=1}^{\ell} \boldsymbol{x}_i^j > 0]$ yielded better results. When using this method instead, we achieved nearly the same accuracy, as shown in the row of Table 1 labeled Multinomial NB-Set.

3.2 Logistic regression

For our next model, we again represent a sentence $(\boldsymbol{x}_1^j, \ldots, \boldsymbol{x}_{\ell(j)}^j)$ as the bag-of-words $\boldsymbol{x}^j = \sum_{i=1}^{\ell(j)} \boldsymbol{x}_i^j$, but now use a logistic regression. In particular, we model the probability of y^j given $\boldsymbol{x}^j$ as $p(y^j|\boldsymbol{x}^j) = \text{Softmax}(\boldsymbol{W}\boldsymbol{x}^j + \boldsymbol{b})[y^j]$, where $y^j \in \{0, 1\}$ and $\text{Softmax}(\boldsymbol{z}) = \frac{\exp(\boldsymbol{z})}{\sum_i \exp(z_i)}$ for a vector $\boldsymbol{z}$. Here $\boldsymbol{W} \in \mathbb{R}^{2\times|\mathcal{V}|}, \boldsymbol{b} \in \mathbb{R}^2$. We perform stochastic gradient descent on the negative log likelihood

$$-\sum_j \sum_{k=0}^{1} \mathbb{1}[y^j = k] \log(\text{Softmax}(\boldsymbol{W}\boldsymbol{x}^j + \boldsymbol{b})[k]).$$

3.2.1 Hyperparameters

We ran stochastic gradient descent to minimize the above objective, with a batch size of 10 and a learning rate of 0.1. We trained for 5000 iterations (batches).

3.3 CBoW with embeddings

Next we implement a continuous-bag-of-words model with embeddings, which is roughly similar to that in Mikolov et al. (2013). In previous sections we were implicitly using the feature map $\phi_{Id} : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|}$, given by the identity $\phi_{Id}(\boldsymbol{x}) = \boldsymbol{x}$. We now ϕ_{Id} with a feature map $\phi_{CBow} : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|d'|}$, where $d' \ll |\mathcal{V}|$ is the dimension of a smaller feature space. In particular, we take $\phi_{CBoW}(\boldsymbol{x}) = \boldsymbol{W}^{(E)}\boldsymbol{x}$, where $\boldsymbol{W}^{(E)} \in \mathbb{R}^{d'\times|\mathcal{V}|}$. Making this transformation has two advantages: first, it reduces the dimension of our feature space, which (at least in the case of a fixed $\boldsymbol{W}^{(E)}$) increases the rate of uniform convergence of our class of estimators. Second, it allows us to take advantage of known relationships between words; in particular, we take $\boldsymbol{W}^{(E)}$ as the FastText embedding matrix Bojanowski et al. (2016). We experiment with two variants of the CBoW model:

1. CBOW-fixed: We fix $\boldsymbol{W}^{(E)}$ throughout the training process, so that our model is simply a logistic regression.

2. CBOW-dynamic: We initialize $\boldsymbol{W}^{(E)}$ to the embedding matrix mentioned above but back-propagate gradients to the elements of $\boldsymbol{W}^{(E)}$ during stochastic gradient descent. In particular, this model is equivalent to a 2-layer neural network with linear activations in the hidden layer:
$$f(\boldsymbol{x}^j) = \text{Softmax}(\boldsymbol{W}\boldsymbol{W}^{(E)}\boldsymbol{x}^j + \boldsymbol{b}),$$
where $\boldsymbol{W} \in \mathbb{R}^{2\times d'}, \boldsymbol{b} \in \mathbb{R}^2$.

3.3.1 Hyperparameters

We ran stochastic gradient descent to minimize the above objective, with a batch size of 10 and a learning rate of 0.1. We trained for 5000 iterations (batches).

3.4 Convolutional neural networks

Finally we implement the Convolutional Neural Network described in Kim (2014). The general architecture consists of applying 100 kernels each of sizes 3, 4, and 5 over our word embeddings. Max over time pooling is then applied, and together, the results are then fed into a final fully-connected layer to obtain the logits for our classification. Prior to the final layer, we implemented dropout layer with parameter 0.5. The diagram for the entire model, borrowed from Yoon's paper is shown in Figure 1. We will call this model CNN-1. The training loss over time is shown in Figure 2.

3.4.1 Hyperparameters

We emulated the parameters in Kim (2014) as best we could. We used the Adadelta optimizer (with a learning rate of 0.1) and batches of size 50. We used the hyperbolic tangent as the activation function for the hidden layer, and trained for 10000 iterations (batches).

Though the convolutional layer in Kim (2014) consists of kernels of size 3, 4, and 5 (each with 100 output channels), we also experimented with kernels of size 2, 3, and 4 (each with 100 channels), early stopping, as well as a ReLU activation function; neither of these extensions significantly affected the model's performance.

3.5 Softmax and sigmoid

In the descriptions above, we use the log softmax as the activation function to obtain our log probabilities, paired with a loss function of negative log likelihood. Since we are performing binary classification, we can, given a feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, instead of computing $\text{Softmax}(\boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b})$ in the above models, simply compute $\sigma(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) + b)$, where $\boldsymbol{w}$ is a vector. This model is no less expressive than the ones above, and has fewer parameters, but it turns out that this choice does not make much of a difference. For instance, when we changed to σ for our CNN-1 model (obtaining a model which we will call CNN-2), the test set accuracy only did not increase significantly (see Table 1).

3.6 Ensemble

Having completed a relative diverse collection of different models, with varying degrees of success, the next logical step is to investigate how well an ensemble of them performs. A simple linear layer, with a log softmax activation, is trained on the binary classification results of four separate models: MNB, Logistic Regression, CBOW, and CNN-1. Suppose each produces a $N \times 1$ prediction vector y_i with $1 \leq i \leq 4$, then we take their concatenation, a $N \times 4$ matrix X. Our final prediction is determined by

$$y = \sigma(WX + b),$$

where the resulting dimension of the linear transformation $WX + b$ is $N \times C$, with C equal to the size of the label vocabulary (i.e. 3 with the given SST dataset). This model behaved somewhat as expected, capping out at the same accuracy as MNB, the best performer.

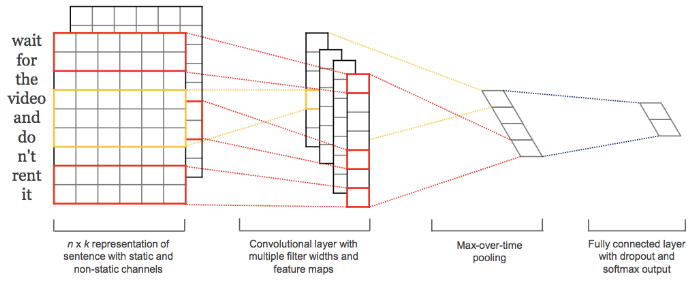
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: The CNN

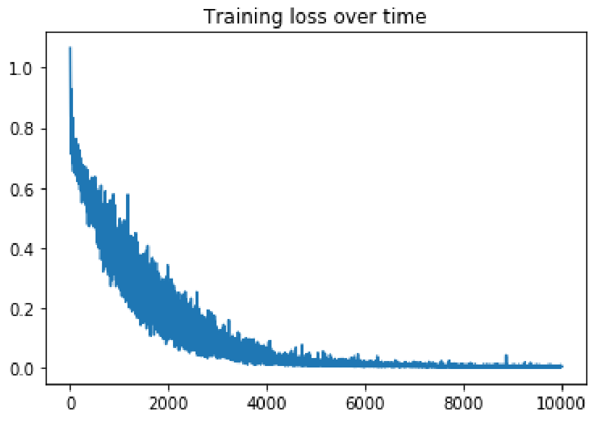

Figure 2: Training of CNN-1

4 Experiments

In this section we report our results on the SST dataset, which consists of 1821 examples. Table 1 presents, for each of our models described in the previous selection, the classification accuracy of the model on a held-out test set. The training set consists of around 6900 samples.

As can be seen, the Naive Bayes models perform the best. CBOW with fixed embeddings performs significantly better than CBOW with dynamic embeddings, which still outperforms logistic regression. Both CNN models outperform all other models apart from the Naive Bayes ones.

5 Conclusion

Ultimately, we find that the multinomial naive bayes (MNB) model performs the best on the test set. We were unable to achieve higher accuracies, described in Kim (2014), using CNNs. While

Model	Acc.
MULTINOMIAL NB	82.43
MULTINOMIAL NB-SET	82.15
LOGISTIC REGRESSION	71.83
CBOW-FIXED	77.27
CBOW-DYNAMIC	72.38
CNN-1	80.23
CNN-2	80.17
ENSEMBLE	82.43

Table 1: Table with the main results.

theoretically logistic regression should always outperform the MNB model, due to its greater flexibility, this did not hold up empirically due to the finite amount of data available. MNB was able to achieve a high accuracy in a short time. Upon ensembling several of the models together, the ensemble learns to trust the MNB entirely, achieving the same final test accuracy. Our implementation of the CNNs were able to achieve reasonable results, outperforming both logistic regression and CBOW, but require further refining of the parameters.

References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv:1607.04606 [cs]*. arXiv: 1607.04606.

Kim, Y. (2014). Convolutiona neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. arXiv: 1301.3781.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank*.

Wang, S. and Manning, C. (2012). Baselines and bigrams: simple, good sentiment and topic classification. In *Proceedings for the 50th Meeting of the Association of Computational Linguistics*, pages 119–122, Jeju, Republic of Korea.