HW1: Classification

Emily Tseng
et397@cornell.edu

January 31, 2020

1 Introduction

Given a sentence from a movie review, e.g. "*Never inspires more than an interested detachment.*", how might we classify its sentiment, in this case *negative*?

In this assignment, we use the implementation of a handful of basic statistical models for this task to familiarize ourselves with core concepts and technologies in modern NLP. Specifically, we explored the use of the following models:

- **Multinomial Naive Bayes (MNB)**, as outlined in Wang and Manning (2012)
- **Logistic Regression (LR)**
- **Continuous Bag of Words (CBOW)**, as outlined in Mikolov et al. (2013)
- **Convolutional Neural Network (CNN)**, as outlined in Kim (2014)
- **CNN***, a variant of the CNN using pretrained word embeddings

As a dataset, we used the Stanford Sentiment Treebank (SST-2) (Socher et al., 2013), which provides a corpus of sentences from movie reviews labeled as *positive*, *negative*, or *neutral*. For this assignment, we removed neutral samples and formulated the task as a binary classification problem between positive and negative sentiment.

2 Problem Description

Given a sentence represented as a feature vector $\boldsymbol{x}$, we can find its predicted class $y \in \{-1, 1\}$, where -1 indicates negative sentiment and 1 indicates positive sentiment, via:

$$y = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1}$$

Where σ is an activation function, $\boldsymbol{W}$ is a matrix of weights, and $\boldsymbol{b}$ is a vector of biases. $\boldsymbol{x}$ itself is generated using a feature function ϕ, which can take a number of forms. We elaborate on the specific structures used per model in the subsequent section.

3 Model and Algorithms

3.1 Multinomial Naive Bayes (MNB) over unigrams

As formulated in Wang and Manning (2012), we consider the linear model where a prediction for test case k is expressed as:

$$y^{(k)} = sign(\boldsymbol{W}^T \boldsymbol{x}^{(k)} + b) \tag{2}$$

We consider as features the unigrams within vocabulary $\mathcal{V}$, and define $\boldsymbol{f}^{(i)} \in \mathbb{R}^{|\mathcal{V}|}$ as the feature count vector for training case i with label $y^{(i)} \in \{-1, 1\}$. $\boldsymbol{f}_j^{(i)}$ is thus the number of occurrences of feature j in training sample i. We additionally consider the log-count ratio r of the two count vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, given smoothing parameter α:

$$\boldsymbol{p} = \alpha + \sum_{i:y^{(i)}=1} \boldsymbol{f}^{(i)} \tag{3}$$

$$\boldsymbol{q} = \alpha + \sum_{i:y^{(i)}=-1} \boldsymbol{f}^{(i)} \tag{4}$$

$$\boldsymbol{r} = log(\frac{\boldsymbol{p}/||\boldsymbol{p}||_1}{\boldsymbol{q}/||\boldsymbol{q}||_1}) \tag{5}$$

In this implementation, we consider *binarized count vectors*, in which $\hat{\boldsymbol{f}}_j$ is 1 if feature j occurs at least once in the sentence, and 0 otherwise. We additionally consider our weights $\boldsymbol{W}$ as a vector of log-count ratios, and use as a bias the log-ratio of the number of positive to negative samples. Formally, we define as parameters the following, where $N+$ is the number of positive samples and $N-$ is the number of negative samples:

$$\boldsymbol{x}^{(k)} = \hat{\boldsymbol{f}}^{(k)}, \hat{\boldsymbol{f}}^{(k)} = \mathbf{1}\{f^{(k)} > 0\} \tag{6}$$

$$\boldsymbol{W}^T = \hat{\boldsymbol{r}} = log(\frac{\hat{\boldsymbol{p}}/||\hat{\boldsymbol{p}}||_1}{\hat{\boldsymbol{q}}/||\hat{\boldsymbol{q}}||_1}) \tag{7}$$

$$\hat{\boldsymbol{p}} = \alpha + \sum_{i:y^{(i)}=1} \hat{\boldsymbol{f}}^{(i)} \tag{8}$$

$$\hat{\boldsymbol{q}} = \alpha + \sum_{i:y^{(i)}=-1} \hat{\boldsymbol{f}}^{(i)} \tag{9}$$

$$b = log(\frac{N+}{N-}) \tag{10}$$

3.2 Logistic Regression

Recalling (1), we express this model as a single-layer, fully-connected neural network using a sigmoid activation σ to learn weight matrix $\boldsymbol{W}$ and bias vector $\boldsymbol{b}$. Here, $\boldsymbol{x}^{(i)}$ represents an input sentence as binarized counts of words in vocabulary $\mathcal{V}$, as in (6). The outputs $\boldsymbol{y}$ are thus 1x2 vectors representing the distribution over the two classes, and we take the argmax of $\boldsymbol{y}$ as the prediction. Our implementation randomly initializes all model parameters, uses Pytorch's built-in CrossEntropy loss function and standard stochastic gradient descent (SGD) optimizer, and backpropagates loss at each batch.

3.3 CBOW

The Continuous Bag of Words (CBOW) model as described in (Mikolov et al., 2013) represents an input sentence $\boldsymbol{x}^{(i)}$ of length n as a k-dimensional embedding vector $\boldsymbol{e}$ produced by averaging the individual k-dimensional embeddings for each word in the sentence. Formally:

$$\boldsymbol{e} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{e}_i \tag{11}$$

This embedding is the input to a single-layer, fully-connected neural network using a softmax activation to learn a weight matrix and bias vector, as before. We take the argmax of the output of this linear layer as the prediction. As before, our implementation randomly initializes all model parameters, including the embeddings $\mathcal{E}$.

3.4 CNN

The Convolutional Neural Network (CNN) approach to sentence classification as described in Kim (2014) represents input sentences as the concatenation of fixed-length embeddings. In other words, for an input sentence of length n:

$$\boldsymbol{x}_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_{n-1} \oplus x_n \tag{12}$$

This input is put through a convolutional layer consisting of a set of *filters*. A filter generates a new feature thus:

$$c_i = f(\boldsymbol{w}\boldsymbol{x}_{i:i+h-1} + b) \tag{13}$$

where f is a non-linear activation function such as the rectified linear unit (ReLU), $\boldsymbol{w}$ is the filter, h is the window size for that convolutional layer, and b is a bias term. Filters are applied over every possible *window* (or *kernel*) of the input to generate a feature map:

$$\boldsymbol{c} = [c_1, c_2, ...c_{n-h+1}] \tag{14}$$

A max-pooling over time operation is then applied over the feature map, which outputs the c_i with the highest value as the value for that feature.

In our implementation, we follow Kim (2014) and use the ReLU nonlinearity, filters with window sizes 3, 4 and 5, a dropout rate of $p = 0.5$ to regularize against overfitting during training, and a softmax activation function over the final output. The argmax of the output of this final layer is our prediction. Of note, in our implementation we padded inputs to a minimum length of 5 to ensure the provided window sizes would work.

3.5 CNN*

The CNN model above is initialized with random values as the initial word embeddings. We also experimented with a variant that initialized with pretrained word embeddings, but otherwise used the same structure as the CNN described above.

4 Experiments

As depicted in Table 1, our results show that when the learned models (LR, CBOW, CNN and CNN*) are trained for 20 epochs with a learning rate of $5e^{-2}$, MNB performs best, with a test-set accuracy of 0.82.

Model	Test Acc.
MNB	0.82
LR	0.71
CBOW	0.77
CNN	0.70
CNN*	0.78

Table 1: Results from each model. LR, CBOW, CNN and CNN models run with $lr = 5e^{-2}$, $epochs = 20$.*

Of note, the CBOW experiments used Pytorch's built-in NLLLoss loss function and the Adam optimizer, while the CNN and CNN* experiments used NLLLoss and the Adadelta optimizer per Kim (2014). The LR experiments used Pytorch's built-in CrossEntropy and the Adam optimizer.

Examination of the losses per epoch shows the learned models are in fact learning, e.g. the losses per epoch decrease over training time (Figure 1). From this we infer the low accuracies for the learned models are due to the training parameters used, and do not accurately speak to their performance relative to MNB.

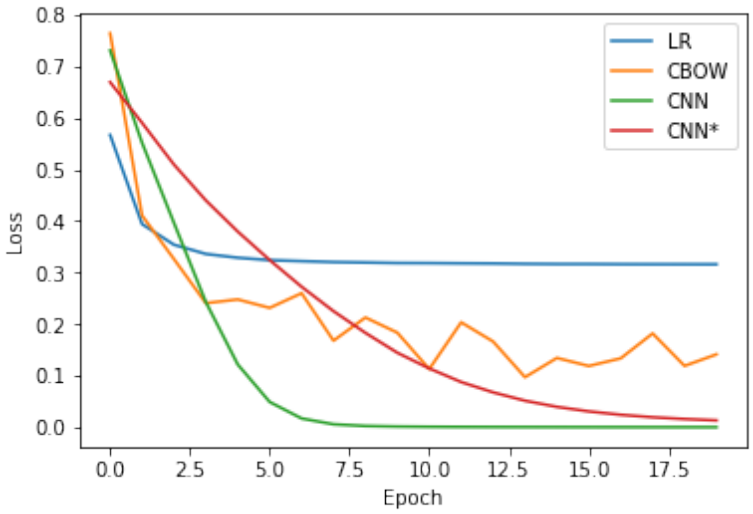

Figure 1: Losses over time for the four learned models. All models trained with $lr = 5e^{-2}$, $epochs = 20$.

Still, we can draw conclusions from the relative differences in performance between the learned models: for instance, CNN* significantly outperforms CNN, as expected.

5 Conclusion

This assignment explored the use of 5 simple models for text classification, as a way to gain familiarity with the tools to be used in this course.

We implemented one count-based model, the Multinomial Naive Bayes (MNB), and four learned models, Logistic Regression (LR), Continuous Bag-of-Words (CBOW), Convolutional Neural Network (CNN) and a variant of the CNN initialized with pretrained word embeddings (CNN*). As shown in Table 1, MNB performed best in our set of experiments, with a test-set accuracy of 0.82. Trained for 20 epochs with a learning rate of $5e^{-2}$, the four learned models (LR, CBOW, CNN, CNN*) did not perform as well as the MNB standard, but the relative accuracies achieved by these models highlight the differences between them. Notably, CNN* significantly outperformed CNN, as expected.

Further work might investigate additional hyperparameter tuning schemes to refine the learned models, for example by using learning rate annealing techniques. Further work should also pursue the use of novel representations of words and sentences as inputs to these models, for instance larger pretrained word embeddings derived from different sources (e.g. BERT), the use of parse trees and other semantic information, or even simply additional hand-tuned features, such as bigram and trigram frequencies.

References

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

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

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2*, pages 90–94. Association for Computational Linguistics.