HW2: Language Modeling

Alex Lin
alexanderlin01@college.harvard.edu

Melissa Yu
melissayu@college.harvard.edu

February 14, 2018

1 Introduction

Language modeling is a very interesting problem that has experienced much development over the past few years. The central idea is to build a statistical model that can accurately estimate the distribution of natural language. For this problem set, we examined the issue of trying to predict the 20 most likely words for continuing a pre-specified sentence. We examined three main models for this problem: (1) a basic trigram model, (2) a neural network language model (NNLM), and (3) a long short term memory recurrent network model (LSTM). We tuned this models to try to optimize *perplexity*, our metric of choice that measures how well a probability distribution predicts a sample.

2 Problem Description

Given a sequence of words encoded as vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the objective is to accurately predict the distribution of $\boldsymbol{x}_{N+1}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Let p be the true distribution and q be an approximation to this distribution.

3 Model and Algorithms

We trained the four different models specified in the instructions. These included (1) a Multivariate Naive Bayes unigram classifier, (2) a logistic regression model over word types, (3) a continuous bag-of-words neural network with embeddings, and (4) a convolutional neural network. We also tried a combination model in which we combined (1) and (4) in an attempt to achieve better accuracy.

3.1 Multivariate Naive Bayes

For the sake of convenience, we map the labels $y \in \{1, 2\}$ to $y \in \{-1, 1\}$ in this model. The Naive Bayes model is a linear classifier of the form

$$y = \text{sign}(\boldsymbol{w}^\mathsf{T}\boldsymbol{x} + b),$$

where $\boldsymbol{w} = \log\left(\frac{p/\|p\|_1}{q/\|q\|_1}\right)$ and $b = \log(N_+/N_-)$. Here, the feature vectors $\boldsymbol{x} \in \mathbb{R}^{|\mathcal{V}|}$ are simply binarized bag-of-words vectors representing each sentence in the text. Note that $\boldsymbol{p} = \alpha + \sum_{i:y_i=1} \boldsymbol{x}_i$ and $\boldsymbol{q} = \alpha + \sum_{i:y_i=-1} \boldsymbol{x}_i$ are count vectors for the training data with smoothing parameter $\alpha = 0.5$. We do not use a smoothing parameter (prior) for the class distribution.

Additionally, we also experiment with non-binarized feature vectors, where each feature can be understood to follow a catergorical distribution over counts having a Dirichlet prior.

3.2 Logistic Regression

The logistic regression model involved takes in a vector representation of a sentence $\boldsymbol{x}$, applies a linear function with weights $\boldsymbol{w}$ and bias b, and uses a sigmoid transformation to return the probability of that sentence having positive sentiment p. That is,

$$p = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

The form of $\boldsymbol{x}$ we choose here is simply a binary vector with size $|\mathcal{V}|$ that has a value of 1 if the associated word appears in the sentence and has a value of 0 otherwise.

We train the weights and bias using the Adam optimizer, which converges to lower losses more quickly than stochastic gradient descent. Our chosen learning rate is 0.01. Using training batches of size 100, we optimize the loss function as the negative log-likelihood of the parameters $L(\boldsymbol{w}, b)$. For this model, we primarily experiment with varying a regularization penalty λ applied to the weights $\boldsymbol{w}$, which changes the overall loss function to

$$L(\boldsymbol{w}, b) + \lambda \|\boldsymbol{w}\|_2$$

3.3 Continuous Bag-of-Words

For this model, we utilize pre-trained word embeddings of length 300 generated using the skip-gram word2vec model, which have been shown to yield better results than the CBOW word2vec embeddings. After embedding all words in each sentence, we pool the embeddings per sentence by taking the sum, and then apply a simple logistic regression model for classification.

As in the previous section, we train the weights and bias of the model using the Adam optimizer and adopt the learning rate 0.01. We optimize the same loss function described previously.

3.4 Convolutional Neural Network

For the convolutional neural network, we first utilize the classic `word2vec` transformation to convert between words in sentences and their continuous vector representations. For each word in a sentence, we have a 300-by-1 vector. Then, we apply three sets of convolutions with kernel sizes of $h = 3, 4, 5$ and 100 out-channels each. The resultant values for each kernel size and out-channel are first put through a rectified linear activation function and then pooled over time. Thus, each original sentence becomes mapped to a 300-feature vector. This vector is then passed through an affine transformation with weights $\boldsymbol{w}$ and bias b, giving us a single value that we pass through a sigmoid activation to generate p, the probability of the original sentence having positive sentiment.

In the spirit of Kim's paper, we also utilize regularization techniques. First, we use dropout with probability 0.5 in the final layer to mitigate co-adaptation of convolutional weight vectors.

Next, we also restrict all weights to have a 2-norm of at most 3. This prevents large weights and overfitting. In our experiments, we mainly vary across different optimizers (Adam, SGD, Adadelta), different regularization parameters, and adding in different convolutions.

3.5 Multivariate Naive Bayes + Convolutional Neural Network

This model is simply an aggregation of Multivariate Naive Bayes and the Convolutional Neural Network. Noting that both models give us discriminative probability distributions $p(y|\boldsymbol{x})$ over the sentiment of a sentence, we thought of the following natural way to combine the two predictions: For a given sentence, if the two models agree on a classification, then we simply output that consensus. If they differ, then we choose the model that is "more sure" of its answer; in other words, we output the proposed class of the model who has a higher probability estimate $p(y|\boldsymbol{x})$. We hoped that this aggregation of the two models could improve the overall accuracy.

4 Experiments

We performed several experiments to tune the hyper-parameters of our models. Perhaps surprisingly, Multivariate Naive Bayes seemed to perform the best on this dataset. In-depth explanations of our tuning procedure and associated results can be found in this section. Note that all accuracies are test set accuracies, which were different from the validation set accuracies used to adjust the hyper-parameters of the model.

Model	Acc.
BASELINE (SINGLE CLASS)	0.538
MULTIVARIATE NAIVE BAYES (BINARY)	0.822
MULTIVARIATE NAIVE BAYES	0.751
LOGISTIC REGRESSION	0.797
CONTINUOUS BAG-OF-WORDS	
CONVOLUTIONAL NEURAL NETWORK	0.802
MULTIVARIATE NAIVE BAYES + CONVOLUTIONAL NEURAL NETWORK	0.805

Table 1: Classification accuracies of our models.

4.1 Multivariate Naive Bayes

Firstly, we experimented with non-binarized features for MNB. In this setting, the feature vectors were bag-of-words counts over each sentence, with symmetric Dirichlet priors over the possible feature count space $[0, 10]$. After tuning the smoothing parameter to $\alpha = 0.5$ using grid search on the validation set, we achieved a best test set accuracy of 0.751.

We conducted a similar tuning process over the validation set for α in the case of binarized features, and achieved a best test set accuracy of 0.822 when $\alpha = 0.5$. The superior performance of binarized MNB compared to standard MNB is unsurprising given the literature.

α	1	0.5	0.1	0.01
Valid Set Acc.	0.629	0.791	0.768	0.764

4.2 Logistic Regression

As specified earlier, we primarily tuned the regularization parameters λ. We tried values of $\lambda = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. Here are the corresponding accuracies on the validation set.

λ	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Valid Set Acc.	0.704	0.771	0.797	0.798	0.792

The differences are not too significant, but we settle on a regularization parameter of $\lambda = 10^{-5}$ from this experiment.

4.3 Continuous Bag-of-Words

Using the summed CBOW word embeddings as the inputs to the logistic regression model, we perform a similar hyper-parameter tuning procedure as before; we pool the train and validation sets generated by Torchtext's data loader and use 5-fold cross-validation to select a learning rate. After finding that CBOW + logistic regression did not yield very good results, we additionally explored the use of deeper models with 2 hidden layers, but did not obtain good results.

4.4 Convolutional Neural Network

For the convolutional neural network, we varies our optimizer, our regularization parameters, and the convolution structure. For the optimizer, we had choices between Adam, Stochastic Gradient Descent, and Adadelta. We saw that Adadelta and Adam generally converged faster than stochastic gradient descent. For the basic model outlined in [4] and described in Section 3.4, we found that Adam's training after 20 iterations led to a higher accuracy than Adadelta. Here are the accuracies on the validation set after 20 iterations for the three different optimization methods.

Optimization Method	Adam	Adadelta	SGD
Valid Set Acc.	0.803	0.792	0.775

Next, we also looked at varying the regularization parameter s, which restricted the maximum 2-norm of the weight vectors. The results were not too interesting. Making s larger than 3 resulted in higher training accuracies, yet lower validation accuracies. Making s smaller than 3 led to decreases in both.

Finally, we also looked into varying the convolutional structure. We added in 100 convolutions for sliding window of $h = 2$ and (separately) 100 convolutions for sliding window of $h = 6$. Neither of these raised the overall validation accuracy.

Convolution Structure	$h = 3, 4, 5$	$h = 2, 3, 4, 5$	$h = 3, 4, 5, 6$
Valid Set Acc.	0.803	0.778	0.801

In addition, we considered the dynamic convolutional neural net described in [4], in which the word embeddings are updated during training as well. However, none of our experiments with the dynamic convolution increases the validation set accuracy past 0.803, which was achieved using the static convolutional neural net.

4.5 Multivariate Naive Bayes + Convolutional Neural Network

In the combined Multivariate Naive Bayes and Convolutional Neural Network model, we actually saw a decrease in performance when compared to the original Naive Bayes model. Taking the best Multivariate Naive Bayes model from Section 4.1 and the best Convolutional Neural Network model from Section 4.4 did not actually lead to a better overall model. We suspect that this is because the Convolutional Neural Net typically predicts high probabilities and it overrode the Naive Bayes's decisions incorrectly in certain cases. The overall validation set accuracy for this combined model was 0.809.

5 Conclusion

Using the Stanford Sentiment Treebank dataset, we have evaluated the performance of four standard sentiment classification models and one additional "hybrid" model, and have found that the simplest model, Multivariate Naive Bayes, performs the best. These results strongly illustrate the superior performance of MNB on short snippets. With the findings of [5] in mind, we envision that the application of bigrams to the MNB model has the potential to further improve classification accuracy.

Perhaps the most important takeaway of this exercise is that overly complex models may perform significantly more poorly in practice than the simplest idea.