HW1: Text Classification

Alex Saich
asaich@college.harvard.edu

Colton Gyulay
cgyulay@college.harvard.edu

February 9, 2016

1 Introduction

For the first assignment of CS287 we are tasked with text classification, specifically within the realm of movie reviews. Our primary dataset consisted of short strings of words each labeled with a rating in the range $1 - 5$. These ratings correspond to each reviewers' sentiment in regard to the film, where lower ratings correspond generally to negative language.

We utilized three different models to learn the statistical associations between specific vocabulary and sentiment. These include a naive Bayes model (Murphy, 2012), a logistic regression model, and a linear support vector machine (Wang and Manning, 2012). The naive Bayes model learned the underlying class distribution and token distribution within these classes, which was then used to predict ratings on examples. The logistic regression and linear SVM models were trained using mini-batch stochastic gradient descent. We experimented with k-fold cross-validation as an improvement over standard methods.

2 Problem Description and Model

2.1 Dataset

Our data initially came in the format of a string of words (for example, "undeniably fascinating and playful fellow") associated with a class y in the range $1 - 5$. The total set of vocabulary $\mathcal{V}$ seen across training examples included around 17000 words. We converted each example to a vector $\boldsymbol{x}_{1\times\mathcal{V}}$ where each index represented the number of occurrences of a specific token within that example.

2.2 Naive Bayes

For the naive Bayes model, we first compiled a prior probability vector $\boldsymbol{y}_{1\times y}$ to model $P(y)$. We then built a class-word probability matrix $\boldsymbol{M}_{y\times\mathcal{V}}$ that indicates the probability that a word is associated with a given class; $\boldsymbol{M}$ models $P(x|y)$. To avoid zeroing out probabilities of examples with words not seen in a specific class, we add a smoothing constant α to each occurrence count. By an assumption of independence among words in an individual example and by Bayes' theorem, we can treat the probability that an example belongs to a class as proportional to the product of the individual word probabilities and the prior probability of that class. More formally, $p(y_i|x) \propto p(y_i) \times p(x|y_i)$, where $p(x|y_i) = \prod_{j=1}^{\mathcal{V}} p(x_j|y_j)$ from $\boldsymbol{M}$.

2.3 Logistic Regression and Linear SVM

Our logistic regression and Linear SVM models use similar topologies and training methods, though differ in cost and gradient calculation. Both models were trained using mini-batch stochastic gradient descent with mini-batch size m. A weight matrix $\boldsymbol{W}_{\nu \times y}$ and bias vector $\boldsymbol{b}_{1\times y}$ were combined in the form $\boldsymbol{xW} + \boldsymbol{b} = \boldsymbol{z}$ and then passed to a Softmax function to convert scores to a normalized probability distribution $\hat{y}_i$ for each class y_i. A model's prediction for any example is given by $argmax_i(p(y_i|x))$.

The logistic regression model sought to minimize a cross-entropy loss objective for clarity in this formulation, c represents the correct class: $L_{cross-entropy}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_c \hat{\boldsymbol{y}}_c log(\hat{\boldsymbol{y}}_c)$. Gradients for computing $\frac{\partial L(y,\hat{y}_i)}{\partial z_i}$ took the following form, which calculates how the loss varies in regard to the Softmax output $\boldsymbol{z}$:

$$\hat{\boldsymbol{g}} = \begin{cases} -(1-\hat{\boldsymbol{y}}_i) & i = y \\ \hat{\boldsymbol{y}}_i & ow. \end{cases}$$

The gradient for $\boldsymbol{b}$ is simply $\hat{\boldsymbol{g}}$, while the gradient for $\boldsymbol{W}$ is given by $\boldsymbol{x}\hat{\boldsymbol{g}}$. The linear SVM model sought to minimize a hinge loss objective (for clarity in this formulation, c represents the correct class and c' the highest scoring *incorrect* class): $L_{hinge}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = max\{0, 1-(\hat{\boldsymbol{y}}_c - \hat{\boldsymbol{y}}_{c'})\}$ Gradients for computing $\frac{\partial L(y,\hat{y})}{\partial \hat{y}_j}$ took the following form:

$$\hat{\boldsymbol{g}} = \begin{cases} 0 & \hat{\boldsymbol{y}}_c - \hat{\boldsymbol{y}}_{c'} > 1 \\ 1 & j = c' \\ -1 & j = c \\ 0 & ow. \end{cases}$$

The gradients for $\boldsymbol{b}$ and $\boldsymbol{W}$ follow the same pattern as described above. For both regressions, we used $L2$ regularization to prevent overfitting with a regularization coefficient represented by λ. In our gradient descent, we use a learning rate of α that determines how descent step size. The propagation of these gradients is explained further through pseudo-code in the following **Algorithms** section.

3 Algorithms

The learning using the naive Bayes classifier and the two regressions was done slightly differently. In the case of the naive Bayes, a prior probability of each class' totals was created out of the training data, and then a count matrix for words present in each class. $\boldsymbol{b}$ and $\boldsymbol{W}$ were then normalized to represent probabilities, and new x vectors would be transformed by the weights to find a class distribution. A more in-depth explanation is outlined in pseudo-code below.

The regressions were used mini-batch gradient descent. Because our training set was so large, and because regression inherently takes much longer to train than naive Bayes (gradient weights must be calculated at each step rather than just calculating occurrence sums), we would take randomized batches of the training data for learning. While this meant that the chances of training again on the same datapoint were not 0, we found it was the most efficient way of sampling a subset of the dataset that did not sacrifice our performance. For each mini-batch, an aggregated weight gradient matrix and bias gradient term were created and then applied to our $\boldsymbol{W}$ and $\boldsymbol{b}$.

Performing descent in batches rather than stochastically was more efficient because it meant that derivatives only needed to be found every n inputs rather than every time.

We also tried running the regression using k-fold cross-validation (not mentioned in pseudo-code below) in order to see if this would yield a more accurate result. We split the group into k groups, iterating through the groups and holding each of them apart as a validation set while the algorithm trained on the other $k-1$ sets. The weights were then averaged across the k iterations to give a model that better represents all the data. Listed below is pseudo-code for our the algorithms that did the heavy lifting: naive Bayes and SGD. We also include code for the cross-entropy and hinge loss objectives.

```
1: procedure NAIVE BAYES($x_1 \ldots x_N, y_1 \ldots y_N$)
2:     $W \leftarrow 0$
3:     $b \leftarrow 0$
4:     for $i = 0, \ldots, C$ do
5:         $b_c \leftarrow \sum_{i=1}^{n} \frac{\mathbf{1}(\mathbf{y_i}=c)}{n}$
6:         $F_f \leftarrow \sum_{i=1}^{n} \mathbf{1}(\mathbf{y_i} = c)\mathbf{1}(x_{i,f} = 1)$ for all $f \in F$
7:         $W_c \leftarrow \frac{F_f}{\sum_{f' \in F} F_{f',c}}$ for all $f \in F$
8:     return $W, b$
```

```
1: procedure MINI-BATCH GRADIENT DESCENT($x_1 \ldots x_N, y_1 \ldots y_N, S, \alpha, \lambda, f$)
2:     $W \leftarrow 0$
3:     $b \leftarrow 0$
4:     for $s = 0, \ldots, S$ do
5:         $\vec{X} = x_s, \ldots, x_{s+\frac{N}{S}}$
6:         $\vec{Y} = y_s, \ldots, y_{s+\frac{N}{S}}$
7:         $L_2 = \frac{\lambda}{2}\|W\|_2^2$
8:         $\hat{y} = WX + b$
9:         if $f =$ cross entropy then
10:            $z = \frac{exp(\hat{y})}{\sum_c exp(\hat{y_c})}$
11:            $\frac{\partial L}{\partial z}$ = crossentropy($z$)
12:        else if $f$ = hinge then
13:            $\frac{\partial L}{\partial y}$ = hinge($\hat{y}$)
14:        $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y}$
15:        $\frac{\partial L}{\partial W} = x\frac{\partial L}{\partial y}$
16:        $b \leftarrow b - \alpha(\frac{\partial L}{\partial b} + \lambda b)$
17:        $W \leftarrow W - \alpha(\frac{\partial L}{\partial W} + \lambda W)$
18:    return $W, b$
```

```
1: procedure CROSS ENTROPY($\hat{Y}_{1,\ldots,N}, Y_{1,\ldots,N}$)
2:     $\frac{\partial L}{\partial y} \leftarrow 0$
3:     for $i = 0, \ldots, N$ do
4:         $\frac{\partial L}{\partial z}_{i,c} \leftarrow -(1 - \hat{y}_i)$
5:         $\frac{\partial L}{\partial z}_{i,c'} \leftarrow \hat{y}_i$
6:     return $\frac{\partial L}{\partial y}$
```

```
1: procedure HINGE LOSS($\hat{Y}_{1,\ldots,N}$, $Y_{1,\ldots,N}$)
2:     $\frac{\partial L}{\partial y} \leftarrow 0$
3:     for $i = 0, \ldots, N$ do
4:         if $\hat{Y}_{i,c} - \hat{Y}_{i,c'} > 1$ then
5:             ${\frac{\partial L}{\partial y}}_i \leftarrow 0$
6:         else
7:             ${\frac{\partial L}{\partial y}}_{i,c} \leftarrow -1$
8:             ${\frac{\partial L}{\partial y}}_{i,c'} \leftarrow 1$
9:     return $\frac{\partial L}{\partial y}$
```

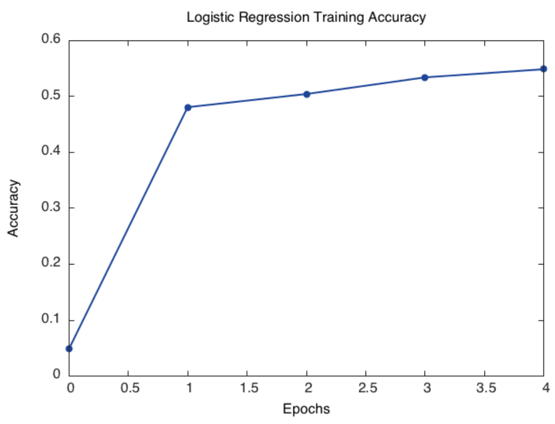
4 Experiments

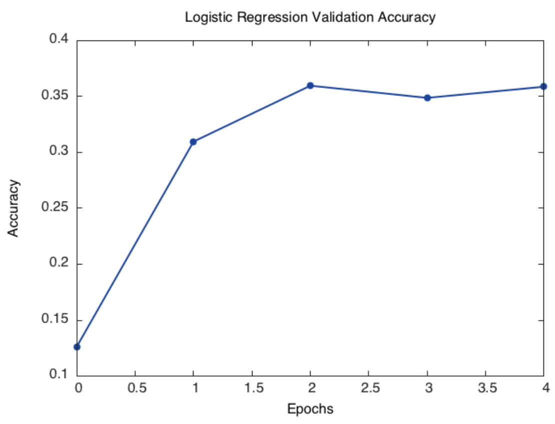
We found that our results performed considerably better than a baseline model with no learned weights, confirming that our model was in fact learning on the dataset. Between the three distinct models we did not find a wild divergence, as all three performed reasonably well, though the vanilla logistic regression model might have performed better with further hyperparameter tuning. For its simplicity, the naive Bayes model performed impressively. Regression with at least a few passes over the entire dataset performed comparably. All scores reported in Table 1 are based on validation accuracy.

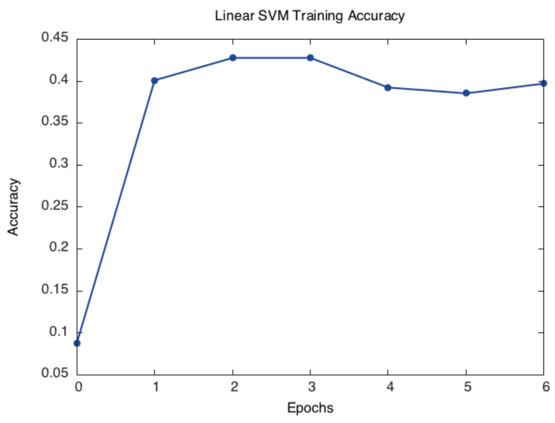
Model	SST1	SST2
NAIVE BAYES	39.9	80.1
LOGISTIC REGRESSION	38.44	81.2
LINEAR SVM	40.9	77.4
LOGISTIC WITH 10-FOLD CV	35.8	78.8

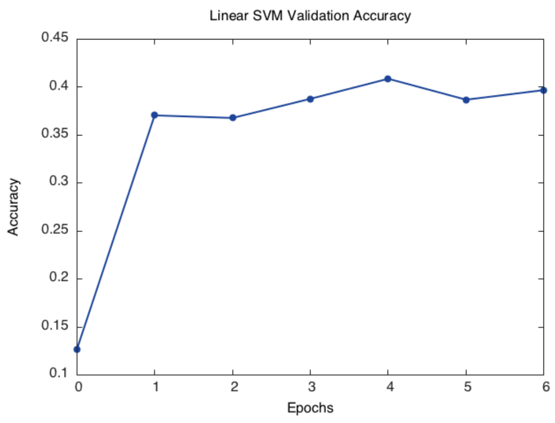
Table 1: Results on datasets SST1 & SST2

We've also included plots showing training and validation accuracy over epochs for the logistic regression and linear SVM models. These scores are recorded on the SST1 dataset. Epoch 0 represents prediction accuracy prior to any training.









5 Conclusion

Our final resulting accuracy was on par with others in the class in regard to Kaggle submission. Our models vastly outperformed our baseline performance, and we found that we had the most success with cross-entropy logistic regression (although naive Bayes was the fastest). Furthermore, we found pronounced success using k-fold cross-validation to complement the learning in our mini-batch gradient descent.

We encountered problems with our loss functions as well as trying to get our weights to converge. Our SVM loss at times diverged into the hundreds unless a large $L2$ value was set, despite the fact that our accuracy was continually improving. This could point towards overfitting on our training model given how weight accumulation. Given more time we would have liked to resolve this issue in order to keep the weights and loss lower.

Some additional steps that could have been taken include the implementation of bigrams for inputs, which has been shown to produce better performance and more accurate results. The downside of this, however, would have been the drastic increase in vocabulary size ($\|V\|^2$, all possible combinations of words present). Our limited processing power was already struggling with the size of the massive sparse matrices. We would have also liked to have optimized our sparsify function to run without for-loops, as we found that the run-time of the function was limiting the scope of the dataset that could be used at one time. Since we were using mini-batch

SGD, however, this problem was actually not too pronounced. In total, we were able to produce fairly effective models without extensive manual feature engineering. The full source code can be found on GitHub: `https://github.com/cgyulay/cs287`.

References

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

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.