Text Classification: Replicating Research Baseline Results

Vincent Nguyen and **Austin Shin**
CS 287: Statistical Natural Language Processing
Harvard University
Cambridge, MA 02138
{vnguyen01, ashin01}@college.harvard.edu

Abstract

The text classification performances of different different classifiers such as Naive Bayes (NB), Logistic Regression (LR), and Linear Support Vector Machine (SVM) have been widely published and extensively researched. These statistical pattern recognizing approaches contain variations in performance based on hyperparameter tuning, corpus characteristics, the feature space, etc. In this assignment, we attempt to: (i) explain similarities and differences of classifier performance, (ii) achieve similar performances on known, public datasets by building machine learning classifiers from scratch using Lua and Torch, and (iii) explore additional methods of feature engineering.

1 Introduction

The task of automatic text classification is one that is explored in natural language processing (NLP), machine learning (ML), and information retrieval (IR). Despite the focus on deep learning approaches in recent years, popular classifiers such as Naive Bayes (NB), Logistic Regression (LR), and Linear Support Vector Machine (SVM) often achieve comparable performances to more recent state-of-the-art neural network techniques [3]. To explore the strengths and weaknesses of these generative and discriminative classifiers, we built them from scratch using Lua and Torch without the use of any additional machine learning packages. This process allows us to thoroughly understand in detail the architecture of each classifier which we then use to present our performance accuracy on public datasets.

2 Datasets

The datasets which are publicly available for download include the following files:[1]

- `SST1`: Stanford Sentiment Treebank - an extension of MR but with train/dev/test splits provided and fine-grained labels (very positive, positive, neutral, negative, very negative), re-labeled by Socher et al.[2]
- `SST2`: Same as SST-1 but with neutral reviews removed and binary labels.

Each file contains strings such as sentences, phrases, or words and their assigned categorical class (ratings, sentiments, etc.). All of these files contain training and validation sets while only some include a test set. For the purposes of this assignment, there was a Kaggle competition for the best model accuracy on the `SST1` test set.

Data	C	Sen	Vocab	Train	Valid	Test
SST1	5	18	17836	156817	1101	2210
SST2	2	19	16185	76961	872	1821

3 Learning Methods

We can formally define the task of classification as given an input document x and discrete set of N possible classes $C = \{1, \ldots, d_N\}$, return a predicted class $c \in C$. We focus on three general algorithms for supervised classification.

3.1 Naive Bayes Classifier

NB takes the *generative* approach to classification by modeling the joint distribution of feature inputs and classes. For each class, there is assumed to be a parameter vector θ_N that generates documents indepedently. To model the probability of choosing a class given some input, the classifier makes use of Bayes' Theorem:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

where the posterior is result of the likelihood multiplied by the prior divided by the evidence. We can disregard

[1] The dataset descriptions are obtained from the CS 287 assignment `README.md` specifications. The files are available for download at: `www.github.com/CS287/HW1/tree/master/data`

[2] http://nlp.stanford.edu/sentiment/

$p(x)$ and write $p(y|x) \propto p(x|y)(y)$ since the evidence is the same for all classes and we only want to maximize the posterior. The class conditional density of the likelihood can be written as

$$p(x|y=c,\theta) = \prod_{j=1}^{K} p(x_j|y=c,\theta_{jc})$$

The form and distribution associated with the class-conditional density depends on the type of features in the input vector. For binary features, the Bernoulli distribution is used, and for categorical features, we can use the multinomial distribution. The general form of the negative-loglikelihood is

$$L(\theta) = \sum_{i=1}^{N} \log p(y) + \sum_{i=1}^{N} \sum_{j=1}^{K} \log p(x|y)$$

where we use maximum likelihood estimation (MLE) to estimate the parameters of the distribution.

3.2 Logistic Regression

LR is a discriminative classifier belong to the family of linear models, and rather than modeling the joint distribution, for the multinomial case, LR is concerned with directly modeling conditional $p(y|x) = \text{softmax}(\theta^T x)$. The probability of picking a class is using the softmax function is

$$p(y=c|x,\theta) = \frac{\exp(\theta_c^T x)}{\sum_{c'=1}^{C} \exp(\theta_{c'}^T x)}$$

The negative log likelihood (NLL) is

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \left(\sum_{c=1}^{C} y_i \theta_c^T x_i - \log \left(\sum_{c'=1}^{C} \exp\left(\theta_{c'}^T x_i\right) \right) \right)$$

To obtain our coefficients, we minimize the NLL using stochastic optimization methods.

3.3 Linear Support Vector Machine

Linear SVM is a discriminative classifier that also models a linear relationship, $y = w \cdot x + b$. However, the loss function in this case is the Hinge loss,

$$L(\theta) = \sum_{i=1}^{N} ReLU(1 - (\hat{y_c} - \hat{y_{c'}}))$$

We add L2 regularization to prevent overfitting. We also implemented the Hinge squared loss for the SVM, where the Loss function may be written as:

$$L(\theta) = \sum_{i=1}^{N} (ReLU(1 - (\hat{y_c} - \hat{y_{c'}})))^2$$

Again, we add L2 regularization to prevent overfitting.

3.4 Regularization

To prevent overfitting, a penalty term is added to the NLL. L2 regularization is the sum of squares of the coefficients

$$\lambda \|\theta\|_2^2 = \lambda \sum_{i=1}^{N} |\theta_i^2|$$

With L2, the objective function remains differentiable while L1 regularization which is

$$\lambda \|\theta\|_1 = \lambda \sum_{i=1}^{N} |\theta_i|$$

creates nondifferentiability. While L1 regularization works well with sparse data by causing many of the coefficients to go to 0, the optimization problem is more difficult. As such, for second-order optimizers, L1-regularized models sometimes use different ones such as OWL-QNS, a variant of L-BFGS [2].

3.5 Optimization

Stochastic Gradient Descent A reliable method of obtaining θ is SGD. By computing the first-order gradient $\hat{g}$ w.r.t θ, we update for each iteration as

$$\theta_t \leftarrow \theta_{t-1} - \eta \hat{g}_t \qquad \text{Update weights}$$

where η is the learning rate. Rather than sampling an individual example, *mini-batch* sampling takes in m samples. This is a much more efficient method than using all examples or only one example.

Adam Another first-order stochastic optimization method is Adam which achieves better performance than SGD. Adam computes first and second moment estimates used as update parameters [1]:

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\hat{g}_t \qquad \text{Update 1st moment}$$
$$v_t \leftarrow B_2 v_{t-1} + (1-\beta_2)\hat{g}_t^2 \qquad \text{Update 2nd moment}$$
$$\hat{m}_t \leftarrow m_t/(1-\beta_1^t) \qquad \text{Correct 1st moment}$$
$$\hat{v}_t \leftarrow \beta_t/(1-\beta_2^t) \qquad \text{Corrected 2nd moment}$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \hat{m}_t/(\sqrt{\hat{v}} + \varepsilon) \qquad \text{Update weights}$$

Both optimization techniques above use very little memory and only require the first-order gradient. Hyper parameter tuning of learning rate and the number of iterations is often necessary for optimal results.

4 Experiments and Results

4.1 Text Preprocessing

The text was preprocessed with word tokenization and some punctuation removal. Stopwords were not removed due to the already sparse dataset.

4.2 Feature Weighting

We use bag-of-words unigrams with boolean weights and count weights. We note that the datasets consist of short phrases usually taken out of context. While more complex approaches can learn continuous distributed fixed-length representations, we use a bag-of-words model for its speed and the sparsity of the dataset.

For the binary case, we adopt a feature weight w_{ij} for document d_j that corresponds to the presence of absence of a feature φ_i:

$$w_{ij}(\varphi_i, d_j) = \begin{cases} 1, \text{ if } \exists\ \varphi_i \in d_j \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

This weight can also be compatible with more complex Boolean expressions such as assigning a feature the value '1' when a set of words occur together in a document and '0' otherwise.

Another weight used is feature counts where the correponding weight function is of the form

$$w_{ij}(\varphi_i, d_j) = n_{ij} \quad (2)$$

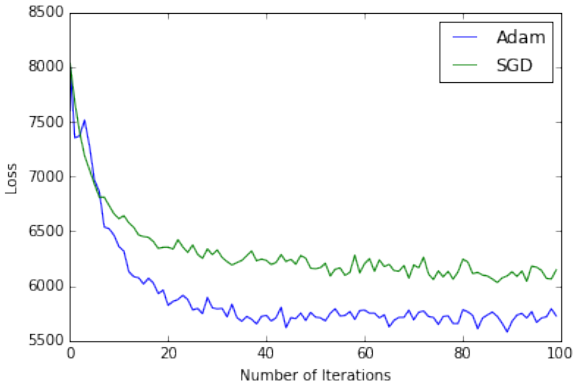
Both of these above weighting schemes gave good results but are not without downsides. Binary weights can often be too simplistic and count weights have a bias towards longer documents with high counts regardless of the actual importance of the feature. We did not use term frequency or tf-idf (term frequency inverse document frequency) due to the extremely short sentences and unique phrase data.

4.3 Batch Training

Since the entire corpus of reviews is large (roughly 150,000), we use batch training to reduce optimization time. It is important to choose the right batch size. Too small of a batch size will mean the model gravely overfits to the batch sample. Too large a batch size means the weight updates will take too long. We found that batch sizes of around 500 sufficed to get decent accuracies in a short period of time, while batch sizes of 10000 were required to get optimal results.

4.4 SGD vs. Adam

As part of our exploration, Adam in addition to SGD was used to optimize both the LR and SVM. We found that the number of iterations until convergence was significantly less for Adam than SGD. For the following experiments, we rely on Adam as our first-order gradient optimizer. The loss per number of iterations for the `SST1` dataset is shown for both optimizers:



On the `SST1` dataset with a mini-batch size of 10,000 (7% of the training examples). Furthermore, SGD is extremely sensitive for η, requiring $\eta < 0.001$ to converge and the loss is less smooth. We found that for convergence, η and mini-batch size are inverseley correlated (extremely so) for SGD. Adam is much better suited for varying batch sizes for a given unscaled η.

4.5 Initialization

All weights are initialized randomly around 0 with standard deviation 10^{-3}. For LR, the moment estimates and parameters of Adam are initialized as $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1e8^{-}$. We use a learning rate $\eta = 0.01$ for LR and SVM. The max number of iterations varies for each dataset, but generally we find convergence occurs by $t = 100$ in the worst case.

4.6 Baseline Results

The following tables contain the validation scores. MNB was very fast and achieved scores comparable to

Model	SST1	SST2
MNB, add 1	0.3917	0.8130
MNB, add 2	0.411	0.807
MNB, add 5	0.3906	0.813
LR-L1	0.4021	0.8211
LR-L2	0.4232	0.8291
SVM-L2	0.3912	0.798
SVM-Square	0.3704	0.816

LR and SVM. Experimenting with different α smoothing parameters did not remarkably increase the accuracy. In addition, we could run the entire training dataset with MNB while the same could not be done by the other two models due to time.

In terms of accuracy, this model outperformed the other two. However, the time that it took to achieve the top results was around an hour or two. Adjusting the λ parameter for LR did not seem to have a noticeable effect on accuracy. We were surprised by the little to no change in performance for L1 versus L2 regularization.

Given the sparsity of the dataset, we had expected L1 regularization to give a slight boost over L2.

For this Linear SVM, we preprocessed the data so each datapoint was a vector of ones and zeros corresponding to the presence of at least one instance of a given word. This feature change was seen to improve both the loss and validation accuracy.

With weights initialized randomly around 0 with standard deviation 10^{-3}, we train our model by minimizing the Hinge loss.

With small weights, the loss is roughly equal to the batch size, 10000. As we trained, the regularization term grew, but the Hinge loss decreased to roughly 9000 with a corresponding training accuracy of .65 and validation accuracy of .386.

4.7 Kaggle

Predictions on the `SST1` file was a class competition on Kaggle. Since the `SST1` dataset included a train, validation, and test set, the goal was to obtain the best classification accuracy on the test set given the constraint of only two submissions per day. In addition, 50% of the test data was without, so the accuracy score only reflected a portion of the test examples. Under the team name of *Unemployed (currently)*, we placed 5th with a score of 0.39095. All of our submissions can be seen in the following summary Our best scores for validation

#	Model	Valid	Score
1	MNB, additive 1, class priors	0.3873	**0.38643**
2	MNB, additive 1, no priors	0.3829	0.36199
3	MNB, additive 1, train+valid	0.3917	0.34118
4	Accidental submission	0.3912	0.34118
5	LR, 0.01 rate, 100 iter, L2, 1000	0.3972	0.36018
6	LR, 0.001, 500 iter, L2, 10000	0.4070	**0.39095**
7	LR, 0.001, 500 iter, L1, 10000	**0.4120**	0.38009
8	LR, 0.01, 500 iter, L2, 75000	**0.4232**	0.37557

and Kaggle are bolded. In particular, we expected to break 0.4 due to our validation scores. We feel the lower Kaggle score may reflect overfitting. Furthermore, since the Kaggle score only reflects 50% of the test data, our models could be performing better once the full test data is released.

5 Conclusions

Building these three classifiers from scratch using Lua and Torch was a helpful exercise in understanding the architecture of these models. While we only used a bag of words approach with, interesting things to try could involve fixed-length distribution vector representations such as word2vec and paragraph vectors. In addition, it will be helpful to see the full test data on Kaggle to tell whether our models overfit.

In particular, an exciting aspect that we added on was to look at different first-order gradient optimizers, such as Adam. Future work in this assignment could involve L-BFGS or other second order gradient approximations. Moving forwards, we will continue to explore optimization techniques and different methods of feature representation.

References

[1] Diederik Kingma and Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization* ICLR. 2015.

[2] Galen Andrew and Jianfeng Gao. *Scalable Training of L1-Regularized Log-Linear Models* Microsoft Research. ICML 2007.

[3] Sida Wang and Christopher Manning. *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification* Department of Computer Science, Stanford University, 2012.