HW1: Text Classification

Sebastian Gehrmann
gehrmann@seas.harvard.edu

February 9, 2016

1 Introduction

Classifying documents is a common problem in computer science. Whether it is detecting spam mail or numbers in handwriting, classification is used everywhere. There are many different algorithms that are used in this area. In this write-up, three of them are described, implemented and compared with each other.

The data that was mainly used for development was the SST1 set. This includes parts of textual movie reviews and the corresponding rating on a scale from 1 to 5. The goal was to predict the score given the text.

2 Problem Description

The given problem is to predict the probability distribution over $|\mathcal{C}|$ ratings $\boldsymbol{y}$ given a feature set $\boldsymbol{x}$. The feature set is a vector of length $|\mathcal{F}|$, where $\mathcal{F}$ is the full vocabulary of the set. It is defined as $x = \sum_i \delta(f_i)$. In this case, δ indicates a one-hot vector with the 1 at position f_i. That means that the input vector consists of 0's except for the words that the input actually contains. There, the input has 1's. The output has the form $\boldsymbol{y} = \delta(c)$, with c being the correct class. Single predictions are named $\hat{\mathbf{y}}$ and are not one-hot vectors.

Thus the ultimate goal of this problem is to find $p(\boldsymbol{Y}|\boldsymbol{X})$.

The accuracy's in this write-up will all be measured as the ratio of correctly predicted ratings.

3 Model and Algorithms

This write-up will compare three different linear models which will be explained separately.

In general, a linear model has two parameter, $\boldsymbol{W}$ and $\boldsymbol{b}$. $\boldsymbol{b}$ is a bias term that has a value for each possible outcome class, and $\boldsymbol{W}$ has parameters for every possible feature f and class c. Scores are predicted by computing

$$\hat{\mathbf{y}} = f(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$$

. The inner part of the function is called z.

The goal of this is to minimize the loss of the parameters θ, which is defined as the negative sum as the log-likelihoods of $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$.

3.1 Naive Bayes

Naive Bayes is the model

$$\prod_{i=1}^{k} p(x_{f_i}|\boldsymbol{y})p(y)$$

This model interpreted means nothing else that the probability of a sentence being in class c is the product of the probability of that class multiplied with the probability of every single word in that sentence.

A simple example is the sentence ''A terrible" being classified in one of two categories. In this example, in our randomly chosen training data we have 50% samples of class 0. For this class, the probability for ''A" is 20% and for ''terrible", it is 10%. The non-normalized probability for class 0 is now 1

Class 1, however, has the probabilities 20% and 1%. Thus, the non-normalized probability for it is 0.1%. Normalizing the probabilities would give a $\frac{10}{11}$ probability to class 0, making it the more probable one of the two.

The model used here has an added word count of α added to every single word in order to account for rare words.

3.2 Logistic Regression

The intuition behind logistic regression is to force $\hat{\mathbf{y}}$ to be a distribution by using the softmax. The softmax of $\boldsymbol{z}$ is defined as

$$\texttt{softmax}(\boldsymbol{z}) = \frac{\exp(\boldsymbol{z})}{\sum_c \exp(z_c)}$$

Using this in the loss-function yields the log of the softmax of z which does not have a closed form. Thus, we need to minimize the loss using different methods. In this case, we used stochastic gradient descent. This method uses the partial derivatives of the loss-function with respect to the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ to compute the minimum of the loss.

To not have the parameters grow and potentially overfit the training data, L_2-Regularization is used as well.

3.3 Linear SVM

The last model is the linear SVM using Hinge-loss. The goal in this approach is to directly find the parameters without using any function on z. That means that $\hat{y}$ is calculated by $\boldsymbol{xW} + \boldsymbol{b}$ Hinge-loss is defined as

$$L(\hat{y}, y) = \texttt{max}\{0, 1 - (\hat{y_c} + \hat{y_{c'}})\}$$

4 Experiments

In the experiments, all three algorithms have been trained on the five data sets SST1, SST2, TREC, SUBJ and MPQA. Logistic regression and linear SVM both were trained for 3 full epochs using minibatch-SGD with a batchsize of 50 and a learning rate of .02. λ in the regularization term has been set to 2. For Naive Bayes, the α hyperparameter was 10. Below in table 1 we report on the

Table 1: The results of the experiments

Model	Data	Training Acc.	Validation Acc.
NB	SST1	0.49	0.27
LR	SST1	0.57	0.35
SVM	SST1	0.52	0.30
NB	SST2	0.55	0.51
LR	SST2	0.57	0.64
SVM	SST2	0.51	0.55
NB	TREC	0.02	
LR	TREC	0.69	
SVM	TREC	0.71	
NB	SUBJ	0.93	
LR	SUBJ	0.50	
SVM	SUBJ	0.50	
NB	MPQA	0.58	
LR	MPQA	0.38	
SVM	MPQA	0.33	

accuracy's for both training and validation data if it exists. An additional experiment with SST1 and LR trained for epochs yielded 0.60 and 0.39 as accuracy's.

5 Conclusion

The results show that even though naive bayes is a simple model (and way faster to compute), it can compete or even outperform (see SUBJ) the more sophisticated algorithms. In general, all three models perform better than the baseline, although they might not be 100% working (see NB in TREC or LR and SVM in SUBJ). Due to a lack of time, not many hyperparameter settings have been tried. Tuning those will probably yield a few extra percentage points, as the experiment with SST1 and LR with 10 epochs shows. Unfortunately, the training for 10 epochs takes too long to be considered in this report. Additionally, using a bigram feature would increase the algorithms' success greatly since a greater context contains more information. An example for this is ''not bad" which has a positive sentiment while the individual words might have a negative one.

References

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71.

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