

HW1: Text Classification

Virgile Audi
Kaggle ID: Virgile Audi
vaudi@g.harvard.edu

Nicolas Drizard
Kaggle ID: nicodri
nicolasdrizard@g.harvard.edu

February 9, 2016

1 Introduction

This assignment aims to build three different linear models for text classification and to tune them to fit the Stanford Sentiment dataset (Socher et al., 2013).

We first build a Naive Bayes which is pretty fast to train, a multinomial logistic regression and a linear support vector machine. We evaluate each of these models with their accuracy on the validation set. Our main work was then on tuning the hyperparameters. First, we applied the models with several set of hyperparameters and select the most accurate to the validation set and then we decided to build a k-fold cross-validation pipeline.

The Stanford Sentiment dataset contains about 150 000 text reviews of movies with their rating (from 1 to 5), containing about 17 000 unique words. The reviews are already pre-processed and come as sparse bag-of-words features. The goal is to predict the rating of each review.

We used the Torch Lua framework to build the models and implemented them in the file **HW1.lua**. This file contains a *main()* function which can be called with different parameters from the command line. We also submit results for the team under our Kaggle ID, both ID were used for the team with no regards to the username used. The files can be found in the following github repository:

<http://github.com/virgodi/cs287>

This repository also contains iTorch notebooks where we drafted code.

2 Problem Description

The problem to solve is a multi-class classification on text reviews. We build three linear models which share the same workflow.

The signature of the classifier is the following:

- **Input:** Movie review input with sparse features corresponding to one-hot vector for each word, with \mathcal{F} the vocabulary of the language.

$$x = \sum_{f \in \mathcal{F}} \delta(f)$$

- **Output:** Movie rating, with $c \in \mathcal{C}$, $\mathcal{C} = 1, \dots, 5$ the set of possible output classes

$$\hat{y} = \delta(\hat{c})$$

The pipeline to build and train the classifier is the following:

- **Linear model formulation:** $\mathbf{W} \in \mathcal{R}^{|\mathcal{F}| \times |\mathcal{C}|}$, $\mathbf{b} \in \mathcal{R}^{1 \times |\mathcal{C}|}$ the model parameters

$$\hat{\mathbf{y}} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

- **Loss function:** we use the Negative Log-Likelihood (NLL) for the probabilistic models (NB and Log-Reg) and a loss function for the linear SVM, with $\theta = (\mathbf{W}, \mathbf{b})$

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \log p(\mathbf{y}_i | \mathbf{S}_i; \theta)$$

- **Optimization:** here we want to find the parameters which minimize the loss function

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\theta)$$

NB: this leads to a closed formula with the Naive Bayes, we used a gradient descent for the two other models.

- **Prediction**

$$\hat{c} = \arg \max p(y|x)$$

- **Evaluation:** we use the accuracy (on the training and validation set)

$$\sum_{i=1}^n \mathbf{1}\left(\frac{\delta(\hat{c}_i) = \delta(c_i)}{n}\right)$$

3 Model and Algorithms

We present here in more details each model with its specificity and the different algorithms we used.

3.1 Multinomial Naive Bayes

The multinomial Naive Bayes (Murphy, 2012) is a generative model, it means that we specify the class conditional distribution $p(\mathbf{x}|y = c)$ as a multinoulli distribution. The main assumption here, which justifies the 'naive' name, is that the feature are conditionnaly independent given the class label.

The goal is then to select the parameters that maximizes the likelihood of the training data:

$$p(\mathbf{y} = \delta(c)) = \sum_{i=1}^n \frac{\mathbf{1}(y_i = c)}{n}$$

We also define the count matrix F to compute the closed form of the probability of x given y ,

$$F_{f,c} = \sum_{i=1}^n \mathbf{1}(y_i = c) \mathbf{1}(x_{i,f} = 1) \text{ for all } c \in \mathcal{C}, f \in \mathcal{F}$$

Then,

$$p(x_f = 1 | y = \delta(c)) = \frac{F_{f,c}}{\sum_{f' \in \mathcal{F}} F_{f',c}}$$

Knowing these parameters we can compute the probability of the class given the features:

$$p(y = c | x) \propto p(y = c) \prod_{f \in \mathcal{F}} p(x_f = 1 | y = \delta(c))$$

We can add a hyperparameter to handle the long tail of words by distributing the means. We add a Laplacian smoothing parameter α as follows:

$$\hat{F} = \alpha + F$$

3.2 Multinomial Logistic Regression

The Multinomial Logistic Regression is a discriminative model. The model formulation is the following:

$$\hat{y} = p(y = c | x; \theta) = \text{softmax}(xW + b)$$

On the contrary to the Naive Bayes, there is no closed form for this optimization problem. We use in practice a gradient descent to find a global optimum as the NLL is still convex. We use the cross entropy loss:

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \log p(y_i | x_i; \theta) = - \sum_{i=1}^n \left((x_i W + b)_c + \log \sum_{c'} \exp(x_i W + b)_{c'} \right)$$

To prevent overfitting on the training set, we add a l2 regularization:

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \left((x_i W + b)_c + \log \sum_{c'} \exp(x_i W + b)_{c'} \right) + \frac{\lambda}{2} ||\theta||_2^2$$

We have two kinds of hyperparameters: λ from the penalization term and the gradient descent parameters.

3.3 Linear Support Vector Machine

Now we try to directly find W and b without any probabilistic interpretation.

$$\hat{y} = xW + b$$

We use the linear support vector machine model. The loss function is related to the number of wrong classifications:

$$\mathcal{L}(\theta) = \sum_{i=1}^n L_{0/1}(y, \hat{y}) = \mathbf{1}(\arg \max_{c'} \hat{y}_{c'} \neq c)$$

We use the *Hinge* loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^n L_{\text{hinge}}(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^n \max\{0, 1 - (\hat{y}_{ci} - \hat{y}_{ci'})\}$$

with l2 regularization

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \left((\mathbf{x}_i \mathbf{W} + \mathbf{b})_{ci} + \log \sum_{c'} \exp(\mathbf{x}_i \mathbf{W} + \mathbf{b})_{c'} \right) + \frac{\lambda}{2} \|\theta\|_2^2$$

We have the same hyperparameters as with the multinomial logistic regression.

3.4 Stochastic Gradient Descent

We used a stochastic gradient descent (Bottou, 2012) with minibatch. η is a crucial parameter to tune to make it converge fast. We can also tune the size of the mini-batch. The number of total iteration, also called epoch, could be tuned but we noticed with our experiment that given the size of the data set (around 150 000 rows) one epoch is enough to converge.

Pseudo-code for the SGD with mini-batch and regularisation:

```

1: for  $iteration = 1, \dots, epoch_{max}$  do
2:   Sample a minibatch of  $m$  examples
3:    $\hat{\mathbf{g}} \leftarrow 0$ 
4:   for  $i = 1, \dots, m$  do
5:     Compute the loss  $L(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$ 
6:     Compute gradients  $\mathbf{g}'$  of  $L(\hat{\mathbf{y}}_i, \mathbf{y}_i; \theta)$  with respect to  $\theta$ 
7:      $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \frac{1}{m} \mathbf{g}'$ 
8:    $\theta \leftarrow (1 - \frac{\eta \lambda}{n}) \theta - \eta_k \hat{\mathbf{g}}$ 
9: return  $\theta$ 

```

In order to speed up the code, we took advantage of the hint made by Sasha about the rather sparse structure of our data. We therefore only updated the gradients with the terms derived by the differentiation of the regularisation term only once every 10 minibatch. This allowed us to get significant speed improvements (about 10 seconds faster).

4 Experiments

We applied our three models on the Stanford Sentimental dataset and report in a table below our results. We show the running time to emphasize how faster is the Naive Bayes and the accuracy both on the training and test set. We also show the loss, its the exact value for the Naive Bayes and an approximation on the last mini-batch of the epoch (extrapolated then to the whole dataset) for the two other models.

We ran a validation pipeline to come up with the best set of parameters. We also coded a k-fold cross validation but due to a lack of time, did not experiment enough to show interesting results. We therefore retained the set of parameters using validation which optimizes the accuracy. We

kept the same seed and the exact other same parameters for the different models training. We obtained the following parameters for each model:

- **Naive Bayes** $\alpha = 1$
- **Logistic Regression** Batch size = 50, $\eta = 1$, $\lambda = 0.1$
- **Linear SVM** Batch size = 50, $\eta = 1$, $\lambda = 0.1$

If we look at the results below, we can note that Naive Bayes has the highest Training accuracy but smallest Test accuracy, which seems to indicate that Naive Bayes might be a slightly more overfitting algorithm than the other two. We report the accuracy on the three dataset: train, validation and test (from our Kaggle submission).

Table 1: Results Summary

	Training	Validation	Test	Run Time	Loss
Naive Bayes	0.666	0.399	0.344	5-6s	XX
Logistic Regression	0.601	0.403	0.354	85-87s	4×10^{12}
Linear SVM	0.631	0.411	0.350	86-90s	1.21×10^5

Variances of the outputs of these algorithms are also key insights for analysis.

Table 2: Range of prediction accuracy

	Min Accuracy	Max Accuracy
	Validation	Validation
Naive Bayes	0.257	0.399
Logistic Regression	0.367	0.403
Linear SVM	0.333	0.411

What we can see is that parametrisation is much more crucial for the Naive Bayes algorithm, with performance almost increased by 50% by adding the smoothing parameter $\alpha = 1$. Logistic Regression and linear SVM have very similar performance, both on accuracy and runtime. On the other hand, the Naive Bayes algorithm runs way faster and this aspect must be taken into account when having to choose an algorithm to run.

5 Conclusion

This assignment made us build three different linear models for a text classification task. Whereas the naive bayes is fast to train, the linear SVM and the logistic regression require an optimization algorithm (here stochastic gradient descent). However, the accuracy reached are pretty similar for the different models. We also realized the importance of the tuning part on the hyperparameters to improve our classification accuracy.

There is still room for improvement if more time. First, we could build more features on our reviews. In the current models, we just considered each word separately and built a one-hot feature from it. We could also consider the sequence of consecutive words of a given size n , called also n -grams, to incorporate the relationships between the words to our features. We could also think of additional linear models.

References

- Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer.
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- Socher, R., Perelygin, A., Wu, J. Y., 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 conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer.