HW2: Tagging from Scratch

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

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

February 21, 2016

1 Introduction

This assignement aims to tackle the task of part-of-speech tagging based on the paper **?**.

We applied several models, both in terms of the features used and of the models. First, we applied a multi-class Naive Bayes and then a Multinomial Logistic Regression. We used slicing window features with information from the words and capitalization. We tried both with and without the position of the words inside the window.

Then, we used a neural network architecture with two layers. We ran several experiments on this model which was the most accurate. For instance, we trained it on a pre-trained embeddings of words from **?**. This lead to our best prediction on the test set.

http://github.com/virgodi/cs287

This repository also contains iTorch notebooks where we drafted code.

2 Problem Description

The problem to solve is mult-class classification of tags on words, aka part-of-speech tagging based.

Data We have a training set of around 600 000 words in sentences, and a validation and test set of both around 100 000 words. We pre-processed them to extract in slicing windows of 5 words a vector of word-index and a vector of capitalisation feature for each word. We followed the conventions used in (**?**)

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 $p(\boldsymbol{x}|y = c)$ as a multinoulli distribution. The main assumption here, which justifies the 'naive' name, is that the feature are condionnaly independent given the class label.

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

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

We also define the count matrix $\boldsymbol{F}$ to compute the closed form of the probability of $\boldsymbol{x}$ given $\boldsymbol{y}$,

$$F_{f,c} = \sum_{i=1}^{n} \mathbf{1}(\boldsymbol{y}_i = c)\mathbf{1}(x_{i,f} = 1) \text{ forall } c \in \mathcal{C}, f \in \mathcal{F}$$

Then,

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

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

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

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:

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

3.2 Multinomial Logistic Regression

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

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

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

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

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

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

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 directy find **W** and **b** without any probabilistic interpretation.

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

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

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

We use the *Hinge* loss:

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

with l2 regularization

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

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:

for $iteration = 1, \ldots, epoch_{max}$ **do**
    Sample a minibatch of m examples
    $\hat{\mathbf{g}} \leftarrow 0$
    for $i = 1, \ldots,$m **do**
        Compute the loss $L(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i}; \theta)$
        Compute gradients $\mathbf{g'}$ of $L(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y}_\mathbf{i}; \theta$ with respect to $\theta)$
        $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} + \frac{1}{m}\boldsymbol{g}'$
    $\theta \leftarrow (1 - \frac{\eta\lambda}{n})\theta - \eta_k \hat{\boldsymbol{g}}$
return θ

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 $= 50, \quad \eta = 1, \quad \lambda = 0.1$
- **Linear SVM** Batch size $= 50, \quad \eta = 1, \quad \lambda = 0.1$

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	$4x10^{12}$
Linear SVM	0.631	0.411	0.350	86-90s	$1.21x10^{5}$

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 paramater $\alpha = 1$. 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 assignement 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.