

# HW2: Tagging from Scratch

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

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

February 22, 2016

## 1 Introduction

This assignment 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 led 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 multi-class classification of tags on words, aka part-of-speech tagging based.

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 ?. The words-index were extracted from the dictionary of words of the embeddings. For the capitalisations features, we used the following values:

- 1: lower case words
- 2: capital letter feature
- 3: first letter in caps
- 4: else

The output is a set of 45 classes of tags. We evaluated our model on the validation set to tune the hyperparameters.

### 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  $p(\mathbf{x}|\mathbf{y} = c)$  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:

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

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

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

Then,

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

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

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

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

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

#### 3.2 Multinomial Logistic Regression

As a second baseline model, we implement a multiclass logistic regression for this problem. We recall the theory from homework 1 below.

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

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

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:

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

For this particular problem of speech tagging, we had the two same set of input features as for the Naive Bayes approach, i.e. the word windows as well as the capital windows. The main difference however is that for fitting this particular model, we needed to create some new features based on the position of the word and the type of capitalisation in the window. To do so we indexed words in first position from 1 to 100 002 (the size of the vocabulary), words in second position from 100 003 to 200 004, etc. For instance if the first training point was originally represented by:

$$(1 \quad 1 \quad 5032 \quad 2 \quad 4),$$

it was now inputed as:

$$(1 \quad 100\,003 \quad 205\,036 \quad 300\,008 \quad 400\,014)$$

In total, we needed to fit  $100\,002 \times 5 = 500\,010$  parameters. After writing a functional explicit stochastic gradient descent algorithm, we decided to use the more optimised function **nn.StochasticGradient()** in order to gain some valuable training time. This function, for which documentation can be found on [here](#) (hyperlink), takes for input a dataset with a special table structure, the model itself and the criterion. We can also tune the learning rate and number of epochs to run. It is not clear though what the size of the minibatch is but the function was fast enough not to worry about it.

As the primary goal of this homework was building the neural network, we did not intensively work on the Logistic Regression and kept it rather simple, only implementing a version that only took into consideration the word windows. We will present the results in the Experiments section.

### 3.3 Neural Network Model

Now we try to directly find  $\mathbf{W}$  and  $\mathbf{b}$  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:

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

We use the *Hinge* loss:

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

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. In order to simplify the code and be able to use the above function that only takes a single data-structure argument, we chose to reformat the input data. Instead of having to deal with two different input tensors, i.e. the word and caps tensors, we concatenated them into one single tensor of size:  $\text{dim}(\text{train}) \times 10$ . To be able to then use a single Look-up table, we had to index the caps

## 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	$4 \times 10^{12}$
Linear SVM	0.631	0.411	0.350	86-90s	$1.21 \times 10^5$

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

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.

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

## 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.