# Collobert and Weston (almost) from Scratch

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

## 1 Introduction

We approach the challenge of part of speech tagging using traditional classification methods and a neural network model described in the Collobert and Weston (2011) paper on learning NLP from scratch. Using only word context windows, we attempt to replicate classification accuracy baselines on the Penn Treebank data.

## 2 Dataset

We use the Penn Treebank data for the part of speech tagging task. The Penn Treebank is a large annotated corpus of English text contained nearly 2,500 stories from the Wall Street Journal [1]. Each English word has been annotated for its respective part of speech. There are 45 different parts of speech tags. The data consists of sentences such as the famous, ”Pierre Vinken [2], 61 years old, will join the board as a nonexecutive director Nov. 29.” We summarize the dataset in the table below:

| **Data** | C | Train | Valid | Test |
|---|---|---|---|---|
| PennTB | 45 | 912666 | 131808 | 129696 |

We split the data into a training, validation, and testing set. We assess our models on accuracy on the validation set. The testing set consisted of unlabeled words whose parts of speech were to be predicted for an in-class Kaggle competition [3].

In addition, we also used a set of pre-trained word embeddings learned in the work of Pennington et al. (2014) [3]. These features were used in our neural network models and for feature engineering.

## 3 Learning Methods

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

[1]https://catalog.ldc.upenn.edu/LDC99T42
[2]http://languagelog.ldc.upenn.edu/nll/?p=3594
[3]https://inclass.kaggle.com/c/cs287-hw2

### 3.1 Naive Bayes

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 $\theta_N$ 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.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 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. During the course of creating features we use windowed setups to capture the informative context of a word. As such, we have

$$p(x,y) = \prod_{i=-[\text{dwin}/2]}^{[\text{dwin}/2]} p(\text{word at i in } x|y=c,\theta)$$

Any addition feature parameters (capitalization, sentiment, etc.) we include it as $p(\text{feat. at i in } x|y=c,\theta)$ after our probability of a word at $i$ in $x$.

### 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. We can imagine our $x$ inputs as feature vectors of size dwin $\times V$ where $|V|$ is number of the vocabulary. However, this is a very inefficient as memory intensive representation; instead, we make use of Look up Tables to speed things up. We extend the use of Look up Tables to our Neural Network model as well.

### 3.3 Neural Network

We follow the NN model as presented by Collobert et. al (2011) [4]. We use pretrained word embeddings on a 5-length window where each word is represented as a vector in $R^{50}$. We concatenate this 250-length vector with binary capitalization features for each position in the window. This gives us a 255-length vector. We feed this vector into a linear layer of output dimension 500,

$$y_1 = W_1 x_{input} + b_1$$

although this number may be experimented with (as demonstrated by Collobert et. al). After applying a non-linearity,

$$y_2 = tanh(y_1)$$

we feed this into a final linear layer of output dimension equal to the number of classes

$$y_f = W_2 y_2 + b_2$$

We train via batches using stochastic gradient descent.

## 4 Feature Engineering

We read in words in sentences from the Penn Treebank dataset for the train, validation, and test sets. We only consider unigram representation of words. Going beyond bag-of-words, we capture the contextual representation of a unigram in its respective sentence. Our `preprocess.py` code allows for the option window size parameter. the context window surrounding. We run models for window of sizes 3, 5, and 7. The only additional feature we include is capitalization.

## 5 Results

### 5.1 Naive Bayes

Our implementation of Multinomial Naive Bayes was very quick to run on the entire dataset. The total time for training and validation took close to 50 seconds at most and achieved well-performing accuracy scores for a simple generative classifier.

Our model considered windowed features of size 5 with and without capitalization as well as varying the $\alpha$ Laplacian additive smoothing parameter. Tuning the MNB model during training and validation initially revealed that class priors had a negative impact on the classification accuracy. When selecting the most probable class in

$$\hat{c} = \arg\max_{i \in C} \left(x_i W^T + b\right)$$

the bias $b$ is the logarithm of the class probabilities defined as

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

For an $\alpha = 1$ additive one smoothing, the validation accuracy of MNB with class priors without capitalization was 0.9002 while without class priors gave 0.9119. This trend of no priors performing better than including priors was observed for additional experiments on just windowed features without capitalization.

However, when capitalization was included, the impact of priors on validation accuracy was positive. An initial hypothesis behind this difference was a drastic difference between the train and validation class priors. However, both class probabilities for each set were similar. The most likely explanation for this phenomenon is the independence assumption of MNB since natural language contains many dependencies. In addition, document size may play a role in affecting the accuracy of priors. Since MNB can be thought of as a linear separator, the feature space is very sensitive to the bias [1].

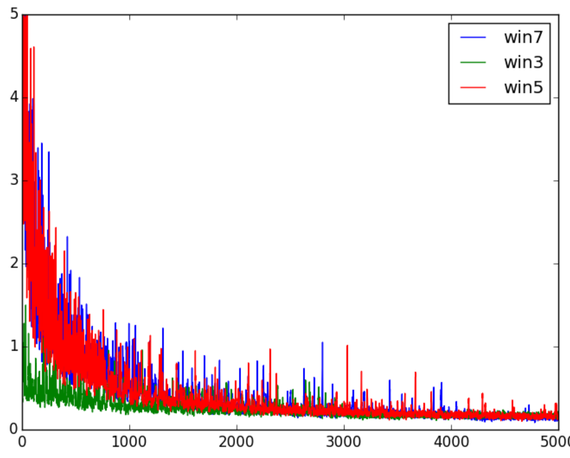
| **Method** | Win | $\alpha$ | Caps | Validation | Time |
|---|---|---|---|---|---|
| MNB | 5 | 0.25 | No | 0.9153 | 26.51 |
| MNB | 5 | 0.5 | No | 0.9169 | 26.55 |
| MNB | 5 | 1 | No | 0.9119 | 26.55 |
| MNB | 5 | 2 | No | 0.8988 | 26.69 |
| MNB | 5 | 0.2 | Yes | 0.9271 | 49.27 |
| MNB | 5 | 0.25 | Yes | 0.9266 | 50.13 |
| MNB | 5 | 0.5 | Yes | 0.9209 | 50.29 |
| MNB | 5 | 1 | Yes | 0.9057 | 50.79 |
| MNB | 3 | 0.05 | Yes | **0.9462** | 29.62 |
| MNB | 3 | 1 | Yes | 0.9168 | 29.92 |

The results indicate that including capitalization of words within the context window improves classification performance. Furthermore, lower values of $\alpha$ for the additive smoothing parameter increases classification accuracy when capitalization is included. This impact is further observed for decreasing window sizes. Window sizes of 3 appear to work best for NB while sizes larger than 5 demonstrate poor performance.

## 5.2 Logistic Regression

Our Logistic Regression model achieved scores that were higher than the MNB model without capitalization. This is expected since we did not include capitalization features in LR. Furthermore, the LR model does not rely on the independence assumption of NB can consider dependencies between word features. Natural language is fraught with dependent word features, and we expected a higher score than NB when feature engineering was not varied.

Additionally, we add an option in our LR model that allows fine tuning of the learning rate $\eta$ during training. We found that in order to get the most out of our LR model, we would have to start $\eta$ at a high value and decrement during training to achieve the best loss. Setting it a a constant $\eta$ would still allow the model to obtain consistent classification accuracies. This non-automatic method is similar to the adaptive gradient algorithm Adagrad [2].



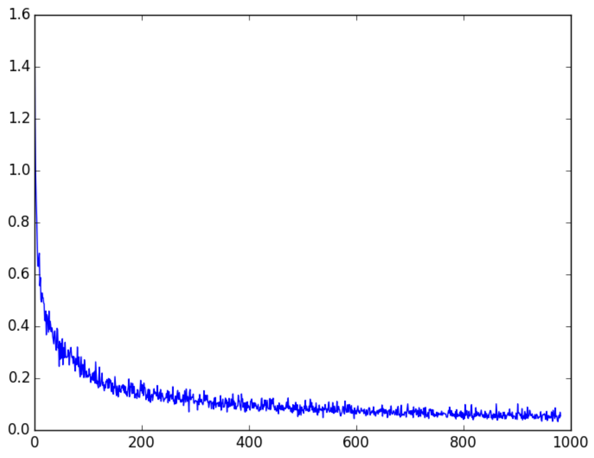
Our loss over time is plotted per iteration for a batch size of 1000. Our model converges for different window sizes.

| **Method** | Win | Caps | Validation |
|---|---|---|---|
| LR | 3 | No | 0.9296 |
| LR | 5 | No | **0.9400** |
| LR | 7 | No | 0.9341 |

For all LR models, we used a batch size of 1000. Unlike the NB model, we did not see significant difference in validation accuracy for window size variations. Furthermore, we used the top 100,000 words from `glove.txt` and set all other words to a label of 'rare'. To obtain an accuracy of at least 0.91, each model would only need to be trained on 3 epochs. To obtain the accuracy scores in the table, some manual fine tuning of $\eta$ might be necessary. We did not implement capitalization for LR since it was meant as a preliminary exercise for working with `nn` on Torch. We do however use capitalization for the NN model.

## 5.3 Neural Network

Our Neural Network model as constructed from Collobert et al. (2011) achieved the best accuracy scores. We found that including capitalization resulted in significantly higher scores. Without capitalization would never reach 0.96 accuracy. We found this trend of improvement to be true among all models. Unfortunately, due to the long training times of the NN model, we did not test on varying window sizes. However, we note that Collobert et al. (2011) achieve the best scores with window sizes of 5. Scores of in this paper with capitalization feature sfor windows of size 5 range around 0.96 [4].



An example of the loss over time is shown below for 1000 batched examples with a 700 hidden nodes. For

| **Method** | Win | Caps | Hidden | Validation |
|---|---|---|---|---|
| NN | 5 | No | 500 | 0.9513 |
| NN | 5 | Yes | 500 | 0.9627 |
| NN | 5 | Yes | 700 | **0.9636** |

each of our models, we keep the number of training examples for each batch at 1000. We modify the number of hidden nodes for our NN model and see that. This is consistent with the observation from Collobert et. al (2011) that 700 hidden nodes performs better than 500 hidden nodes. We keep $\eta = 0.1$ for all models as we found this value to work the best.

### 5.4 Kaggle

We submit 6 models to the in-class Kaggle competition. Even though the public leaderboard only shows 50% of the test set, we noticed that our Kaggle score was always a hundredth of a point higher in accuracy than our validation scores. As of the time of submission of this assignment, we are currently ranked 9 on the public leaderboard under the teamname *Jeb!*.

| **Model** | Win | Caps | Params | Kaggle Acc. |
|---|---|---|---|---|
| NB | 5 | No | 1 | 0.9156 |
| NB | 5 | No | 1 | 0.9186 |
| NB | 3 | Yes | 0.535 | 0.9511 |
| NN | 5 | No | 500 | 0.9536 |
| NN | 5 | Yes | 500 | **0.9639** |
| NN | 5 | Yes | 700 | Currently running... |

## 6 Conclusion

The NN model as presented by Collobert et al. (2011) illustrates an inventive way of learning natural language from just the words themselves. While this model does not achieve the state-of-the-art accuracy scores set by CRF and clustering methods, the exercise of replicating the NN model allowed us to experiment with the architecture of `nn` in Torch.

Were we to spend more time on this problem, we would be interested in exploring additional features such as sentiment. Using the context window approach, capturing the sentiment in a sentence may prove fruitful in determining part of speech. We feel that for the reverse task – determining sentiment – that using part of speech information is useful. A possible future exploration would be to determine the effect of sentiment on part of speech. However, this task may be difficult because the sentence in the Penn Tree Bank are not very emotive.

## References

[1] Karl-Michael Schneider. *Techniques for Improving the Performance of Naive Bayes for Text Classification* University of Passau, Department of General Linguistics. 2005.

[2] John Duchi, Elad Hazan, Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization* Journal of Machine Learning Research. 12, 2121-2159. 2011.

[3] Jeffrey Pennington, Richard Socher, Christopher D. Manning. *GloVe: Global Vectors for Word Representation* Stanford University. Computer Science Department. 2014.

[4] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa. *Natural Language Processing (almost) from Scratch* The Journal of Machine Learning Research. 12, 2493-2537. 2011.