# Sentiment Analysis on Imdb Database

Daniele Gilio

May 5, 2020

## 0.1 Introduction

For this assignment we are going to perform sentiment analysis on film reviews from the popular Imdb site. In order to do that we were asked to build a Naïve Bayesian Classifier able to assign a binary class (good or bad) given the word content of the review itself. The dataset is made of 25000 training samples, 12500 validation samples and 12500 test samples.

## 0.2 Data Preprocessing

The first thing needed to make the Naïve Bayesian Classifier work is some data processing. A crucial passage is the creation of a dictionary which we will use to build a Bag of Words representation of the reviews. To do that we scan all the training documents, clean the inputs by removing unwanted characters and record the frequency of appearance of each word, then we select the $n$-most frequent ones. The choice of $n$ is obviously going to affect the classifier performance. After the dictionary is built, we proceed to create the BoW representation of the documents which is basically a matrix in which columns represent the word in the dictionary and rows documents. The entries in the matrix are then the frequencies of occurrence of each word in each document.

## 0.3 Classification

We used the pvml library to build our Naïve Bayesian Classifier, which by means of Laplacian smoothing, ensures that we never encounter a zero probability. The Naïve Bayesian Classifier has a very low computation time expense since it does not require SGD optimization. On the other hand this kind of classifier might not be the most versatile and makes a strong assumption on the dataset (the data items MUST be conditionally independent). Since in reality this is not always the case the Naïve Bayesian Classifier might suffer performance-wise if compared with other classifiers. Its simplicity though makes it a very good starting point in tasks like sentiment analysis. Given these premises we also built an SVM classifier and a Logistic Regression one. For our baseline training we chose $n = 1000$, the results we obtained can be seen in Table 1. As we can see our concerns

| | Training Accuracy [%] | Validation Accuracy [%] | Test Accuracy [%] |
|---|---|---|---|
| Bayes | 82.004 | 82.064 | 81.632 |
| SVM | 87.432 | 86.528 | 86.192 |
| LogReg | 86.492 | 85.995 | 85.712 |

Table 1: Baseline Training Results

about the performance of the Naïve Bayesian Classifier revealed to be true, even if the performance difference is of a couple of percentage points.

## 0.4 Classification Variants

In the assignment we were hinted to use more preprocessing techniques that may lead to performance improvements, which were Common Word Exclusion and Stemming[1]. The first one, as the name implies, consists in discarding common word which do not carry much useful information. The second one can be regarded as a normalization technique, the words are basically cut to their roots, e.g. "running" becomes "run". The results of the application of these techniques are shown in Tables 2, 3 and 4.

| | Training Accuracy [%] | Validation Accuracy [%] | Test Accuracy [%] |
|---|---|---|---|
| Bayes | 82.92 | 82.992 | 82.576 |
| SVM | 86.98 | 85.351 | 85.536 |
| LogReg | 85.988 | 84.8639 | 84.894 |

Table 2: Ignore Common Words Results

All the results above were produced while keeping every parameter constant. We can observe that the application of the described techniques improves the performance of all the models most of the times. From now on we will keep using both since it seems that that is the most balanced choice.

[1]For this technique we used the Porter Stemming Algorithm provided with the dataset

|  | Training Accuracy [%] | Validation Accuracy [%] | Test Accuracy [%] |
|---|---|---|---|
| Bayes | 82.32 | 81.94 | 81.624 |
| SVM | 87.44 | 86.236 | 86.048 |
| LogReg | 86.632 | 85.688 | 85.784 |

Table 3: Stemming Results

|  | Training Accuracy [%] | Validation Accuracy [%] | Test Accuracy [%] |
|---|---|---|---|
| Bayes | 82.54 | 82.296 | 81.888 |
| SVM | 86.848 | 85.576 | 85.56 |
| LogReg | 86.144 | 85.264 | 85.168 |

Table 4: Stemming + Ignore Common Words Results

## 0.5 Hyperparameters

### 0.5.1 Dictionary Size

As we previously stated the dictionary size is a very important parameter because it carries the most information content given to the model. In order to verify that and to find the best size for our purposes we trained all the classifiers varying only the dictionary size. Figure 1 shows the plot of Test Accuracy against dictionary size. We can see a very clear peak at $n = 2500$ for the Naïve Bayesian Classifier, whereas the SVM and the LogReg

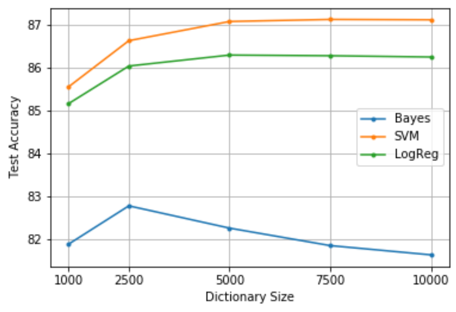

Figure 1: Test Accuracies vs Dictionary Size

increase until $n = 5000$ and then remain constant. We can conclude that the optimal values are those ones both in terms of accuracy and computation time.

### 0.5.2 Learning Rate, Steps and Regularization (SVM & LogReg)

After a couple of quick tests we determined that the optimal number of steps is 2000 and the regularization values are 0.0001 for the SVM and 0 for the LogReg. We did not perform a proper grid search due to the long computation time required. The Learning Rate was a little bit more challenging: we started trying with a fixed one but the result were not satisfactory, they were on par or under the Naïve Bayesian Classifier. After a little bit of online research we settled on an inverse scaling learning rate, basically the learning rate is initialized as $\gamma_0 = 1$ and updated according to:

$$\gamma(t) = \frac{\gamma_0}{\sqrt{t}} \tag{1}$$

where $t$ is the number of steps.

## 0.6 Results Analysis

## 0.7 Conclusions