Movie reviews
Machine Learning

Alice Daldossi

Abstract

In this laboratory activity the goal is to implement a multinomial Naive Bayes (mNB) classifier in order to analyze the movie reviews written by the users of the IMDB website (`www.imdb.com`). In particular, I will try to predict the polarity of the reviews distinguishing the positive comments (class 1) from the negative ones (class 0). In order to build the vocabulary, I tried different methods (*classic*, *stop-words* or *stemming*) and values of the hyperparameter n (that represents the number of words considered in the vocabulary).

1 Build the vocabulary

Script: `A_BuildVocabulary.py`

The data are given in different txt files, so we need to build a vocabulary that collects all of the words and then build a list of the n most frequent ones; this list makes up the file `vocabulary_train.txt`. The number n is the hyperparameter that we are going to choose using the validation set. Different values of n will be used in the two variants of the model that I will discuss in section 5.
During the development of the model it is more useful to work with a small set of training data in order to get fast responses, so at first I used `smalltraining`, but the results reported here are obtained with the entire training set.

2 Feature extraction

Script: `B_FeatureExtraction.py`

A good representation of the data is needed and in order to do so I will use *Bag of Words* (BoW). The function `load_vocabulary` creates a vocabulary in which each word is associated to its position in the file `vocabulary_train.txt`. After removing any punctuation and upper letters, the vector of the BoW representation is built creating a new vocabulary from `vocabulary_train.txt` that associates to each number (that is the position corresponding to each word) the number of times the corresponding word occurs in the text; moreover, I build a vector to keep track of the labels associated to the documents. Finally, the BoW vector and the labels are saved in a matrix and then converted in a compressed text file (`train.txt.gz`, `test.txt.gz`, `validation.txt.gz`) for later use.

3 Train a classifier

Script: `C_TrainClassifier.py`

Let's call $\pi_{y,j}$ the probability that a random word taken from a document of class y is the j-th word in the vocabulary and $P(y)$ the prior probability for class y. Once obtained the BoW representation, I can build a mNB model that computes a binary classification: positive comments ($\hat{y} = 1$) and negative ones ($\hat{y} = 0$). Since this model has only $k = 2$ classes, it can be simplified using:

$$\hat{y} = 1 \iff \mathbf{w}^T\mathbf{x} + b > 0, \quad w_j = \log \pi_{1,j} - \log \pi_{0,j}, \quad b = \log P(1) - \log P(0).$$

The values of $\pi_{y,j}$ are estimated using the Laplacian smoothing and $P(y)$ computing the frequency of the classes in the training set.
If I use a vocabulary of $n = 1000$ words the training accuracy is 82.005% and the test accuracy is 81.632% (validation accuracy of 82.04%).

4 Analysis

Script: D_Analysis.py

I wanted to know which words were the most common in negative reviews and which in positive ones. So I sorted the vector of the weights $\mathbf{w}$ in a crescent order and picked the first 20 and then the last 20 corresponding words.
It is possible to identify the texts from the test set that are wrongly classified. In particular, I was interested in finding the top 5 worst errors on the test set and so I looked at the highest absolute value of the *logit*.

5 Variants

Script: A_BuildVocabulary.py

I propose two different variants of the method above used: the first one ignores very common words that are listed in the `stopwords.txt` file; while the second one uses a technique called *stemming*, that consists in making no distinction among words with the same root (such as “joke”, “joking”, “joker”). In order to add the possibility of choosing between the two variants and the classic approach, I introduced two parameters in the function `read_document`, they are: `stopw` and `stemming`; if they are set to `True`, they implement the corresponding method before filtering the number of words we want to consider using the hyperparameter n. In particular, for each word in the text the first method checks if the word is not contained in the `stopwords.txt` file before considering its insertion in the list; while the second approach adds in the list just the stem of every word in the text. I applied these methods to both training and validation set but in order to choose the best method I have to look at the highest validation accuracy: it appears to be the stop-words'.

	Classic	Stop words	Stemming	Both
Validation accuracy	82.040	**83.008**	78.736	78.632

Table 1: Comparison of the accuracy of the different methods.

6 Hyperparameter: dictionary size

Script: `F_DictionarySize.py`

In the model above implemented I have an hyperparameter n that represents the number of words I am considering in the vocabulary. I repeated the main steps of the classic method for different values of n ($n = 1000, 2000, \ldots, 10000$) and each time I estimated the test and validation accuracy. I ended up having the following approximate result: as the vocabulary size increases the validation accuracy increases too till 4000 (where it is equal to 83.528%), then it drops. Moreover, a big vocabulary size corresponds to a longer computation time. So, I concluded that among the values of the hyperparameter n considered, the optimal one on the validation set is 4000.

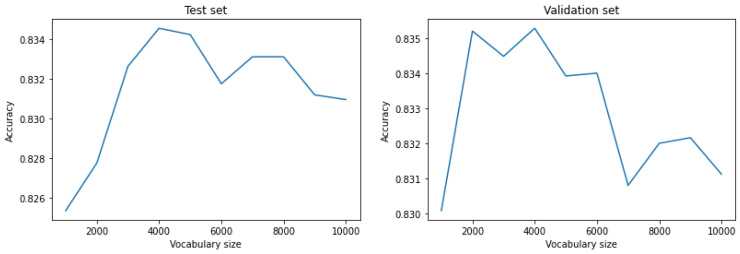

Figure 1: Comparison of the accuracy of the mNB using the stop-words method and changing the number of words considered in the vocabulary.

7 Comparison

Script: `G_Comparison.py`

Machine learning problems of classification can be solved with other classifiers, such as logistic regression (LR) or Support Vector Machine (SVM). Using a vocabulary of $n = 4000$ (the optimal value according to the validation set) created through the stop-words method (the one with the highest validation accuracy), I wanted to compare the accuracy on the validation set of the LR and the Naive Bayes above presented. I conclude that in terms of computing time the mNB is the best choice, but the validation accuracy is higher for the LR.

	Naive Bayes	LR
Training	82.005	86.164
Test	81.632	84.984
Validation	82.040	84.992

Table 2: Comparison of the percentage of the accuracy of the LR and the mNB.

I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.