Lab 2: A naive Bayes spam detector

Maarten Inja Patrick de Kok

September 28, 2012

For this assignment we have implemented and trained a naive Bayes classifier for detecting spam messages. According to the assignment, we return the most likely class, `spam` or `ham`, of a message. The classification of a message is done by finding the most likely class of a characterizing feature vector of the message. This abstraction is based on the presence or absence of specially selected words. The words which are highly informative for the class are selected as the best features for our purpose.

1 Program design

For the implementation we have chosen for Python. Because the computers of us both do not have a working Matlab installation, at first we wanted to run the provided code in Octave. Our Linux distributions (Linux Mint, Ubuntu) come with Octave 3.2 and the `io` library (version 1.0.14-2) packaged in its repositories. However, there are some known bugs with the `textread()` function in these versions. Updating Octave to 3.6 did not solve this problem on our system. After looking for Ninghang Hu in his office, we decided to talk it over with Gwenn Englebienne. He allowed us to work in Python.

We decided to reimplement the provided Matlab functions in Python, so we can stick to the letter of the assignment. These functions, and other general tools can be found in `src/toolkit.py`.

Python uses binary floating point numbers by default. When working with really small values, the floating point arithmetic errors might give imprecise computations. However, working with arbitrary precision decimal numbers is much slower. The software is design in such a way, that we use the value of `toolkit.NUM` as the type of all non-symbolic instances of numeric values. By default, the system will look for the presence of Python language bindings of the GNU Multiple Precision Arithmetic Library, the fastest known implementation of numbers for arbitrary precision arithmetic. If it is present, the software uses the GMP's numeric type. If it is not present, it choses Python's provided `decimal.Decimal` type. The value can easily be set to `float` by hand.

File `src/features.py` contains the functions associated with feature selection, required in section 2 of the assignment. File `src/bayes.py` contains the code for computing the probabilities needed in section 3. File `src/validate.py` ties these together by functions that train and test the classifier, as is needed for section 4.

2 Feature selection

To classify a message as spam or ham, we compute a vector of features $\mathbf{W}$. Each feature W_i indicates whether a certain word is present in the message. If it is present, it is set to 1. Otherwise, $W_i = 0$. Non-alphabetical characters are considered word boundaries, and words are considered invariant under capitalization. The empty string, which might result after applying these operations in specific orders, is not considered as a word and has been removed.

Which features are best to use is computed through the mutual information measure of equation 5.44 of the lecture notes, between the feature we might be interested in and all possible classifications $C = \{\texttt{spam}, \texttt{ham}\}$:

$$\mathbf{I}\left[W_i, C\right] = \int p\left(W_i, C\right) \ln\left(\frac{p\left(W_i\right) p\left(C\right)}{p\left(W_i, C\right)}\right) \mathrm{d}W_i \mathrm{d}C$$

Because our variables are discrete, integration is equal to summation over

all possible values of the variables, and we thus get:

$$\begin{aligned}
\mathbf{I}[W_i, C] &= \sum_{c \in C} \sum_{w_i \in W_i} p(w_i, c) \ln \left(\frac{p(w_i)\, p(c)}{p(w_i, c)} \right) \\
&= \sum_{c \in C} \sum_{w_i \in W_i} p(w_i|c)\, p(c) \ln \left(\frac{p(w_i)\, p(c)}{p(w_i|c)\, p(c)} \right) \\
&= \sum_{c \in C} \sum_{w_i \in W_i} p(w_i|c)\, p(c) \ln \left(\frac{p(w_i)}{p(w_i|c)} \right) \\
&= p(w_i|\texttt{spam})\, p(\texttt{spam}) \ln \left(\frac{p(w_i)}{p(w_i|\texttt{spam})} \right) \\
&\quad + p(\neg w_i|\texttt{spam})\, p(\texttt{spam}) \ln \left(\frac{p(\neg w_i)}{p(\neg w_i|\texttt{spam})} \right) \\
&\quad + p(w_i|\texttt{ham})\, p(\texttt{ham}) \ln \left(\frac{p(w_i)}{p(w_i|\texttt{ham})} \right) \\
&\quad + p(\neg w_i|\texttt{ham})\, p(\texttt{ham}) \ln \left(\frac{p(\neg w_i)}{p(\neg w_i|\texttt{ham})} \right)
\end{aligned}$$

The probability that a word occurs w_i given the classification c of that message is computed as the number of messages of type c containing that word, divided by the total number of messages of type c. The (unconditional) probability of a word occurring is then computed as $p(w_i) = p(w_i|\texttt{spam})p(\texttt{spam}) + p(w|\texttt{ham})p(\texttt{ham})$.

The training set contained 182 training instances of class `spam` and 226 training instances of class `ham`. According to the Messaging Anti-Abuse Work Group's, these numbers are not representative of the class distribution besides our data. In the third quarter of 2011, $p(\texttt{spam}) = 88.8\%$ of the email messages were spam[1]. We have assumed the messages to come from this time period.

When we have done the above for every occurring word in the training files, we sort them based on this information measure. We have chosen to select the 300 highest ranking features. Although we could have tried to train a classifier with different sizes of feature vectors, we were not able to because of time constraints.

[1]The report can be found at `http://www.maawg.org/sites/maawg/files/news/MAAWG_2011_Q1Q2Q3_Metrics_Report_15.pdf`.

Figure 1: The ROC curve for our data. Each data point corresponds to $\left\langle \frac{\text{True } \texttt{spam}}{\text{Classified as } \texttt{spam}}, \frac{\text{False } \texttt{spam}}{\text{Total } \texttt{ham}} \right\rangle$ for a certain threshold difference in $|p(\mathbf{W}|\texttt{spam}) - p(\mathbf{W}|\texttt{ham})|$.

3 Classification

As given in the assignment, the classifier should maximize the likelihood of the data over the possible classifications. Put together with equation 3 of the assignment, we get, with d the number of features:

$$\begin{aligned} c_{\texttt{ML}}(\mathbf{W}) &= \arg\max_{c \in C} p(\mathbf{W}|c) \\ &= \arg\max_{c \in C} \prod_{i=1}^{d} p(W_i|c)^{W_i}(1 - p(W_i|c))^{1-W_i} \\ &= \arg\max_{c \in C} \prod_{i=1}^{d} \text{Bern}(W_i; p(W_i|c)) \end{aligned}$$

There is one caveat. If even one of the inner terms of the big product is just zero ($p(W_i|c)^{W_i} = 0$ or $(1 - p(W_i|c))^{1-W_i} = 0$ for a certain feature W_i), the whole product is 0. To solve this, we apply Laplace smoothing:

$$\begin{aligned} c_{\texttt{ML}}(\mathbf{W}) &\approx \arg\max_{c \in C} \prod_{i=1}^{d} \frac{p(W_i|c)^{W_i}(1 - p(W_i|c))^{1-W_i} + \alpha}{1 + \alpha d} \\ &= \arg\max_{c \in C} \frac{\prod_{i=1}^{d} p(W_i|c)^{W_i}(1 - p(W_i|c))^{1-W_i} + \alpha}{1 + \alpha d} \end{aligned}$$

We apply the smoothing with $\alpha = 1$.

4 Evaluation

After training the classifier on the described feature vector, we have tested it on the test data sets. In Figure 1 one can see the receiver operating characteristic (ROC curve) for different values for the threshold.

We use the loss function depicted in **??**. Based on this function, we minimized the expected error value $\mathbf{E}(\theta) = CM_\theta^T L$ for the confusion matrix CM_θ corresponding to the (mis-)classifications under threshold θ.

We have not evaluated the expected true error of the classifier. This can be done when applying cross validation.

Figure 2: The loss function $L(\mathbf{W}, C)$. Cell L_{ij} represents the weight of classifying an instance of class c_i as a member of class c_j.

	Classified as	
	`spam`	`ham`
True `spam`	0	1
True `ham`	1000	0