Homework 5, Machine Learning

Naïve Bayes classification

1 Task

The task in this homework is to build a naïve Bayes spam filter that classifies emails as spam or non-spam. Naïve Bayes classifier computes the posterior probabilities of an email being either spam or non-spam according to the Bayes rule:

$$P(S|D) = \frac{P(D|S)P(S)}{P(D)} = \frac{P(D|S)P(S)}{P(D|S)P(S) + P(D|\bar{S})P(\bar{S})} \tag{1}$$

where S denotes the email being labeled as spam and $\bar{S}$ as non-spam. D denotes the email that is represented as a collection of words (bag-of-words). Naïve Bayes model assumes conditional independence of words given the class labels. This means that all the words in an email can be treated as conditionally independent of each other given that we know whether the email was spam or not:

$$\begin{aligned} P(D|S) &= \prod_{i=1}^{n} P(w_i|S) \\ P(D|\bar{S}) &= \prod_{i=1}^{n} P(w_i|\bar{S}) \end{aligned} \tag{2}$$

For building the model the following parameters must be learned:

$$\begin{aligned} P(S) &= \frac{N_S}{N} \\ P(\hat{S}) &= \frac{N_{\hat{S}}}{N} \\ P(w_i|S) &= \frac{N_{w_i,S}}{N_S} \\ P(w_i, \hat{S}) &= \frac{N_{w_i,\hat{S}}}{N_{\hat{S}}}, \end{aligned}$$

where N_S is the number of training items labeled as spam and $N_{w_i,S}$ is the number of spam emails containing the word w_i. Similar quantities with the non-spam emails use the label $\hat{S}$.

Use smoothing to avoid zero-probabilities by adding 1 in the numerator and the number of possible outcomes (2 in the current case, because the email either contains the word or not) into denominator like this:

$$P(w_i|S) = \frac{N_{w_i,S} + 1}{N_S + 2} \tag{3}$$

2 Data

We are using a publicly available dataset that can be downloaded from the address: `http://archive.ics.uci.edu/ml/datasets/Spambase`. However, as this dataset contains also other features than words then the part of the data necessary for this homework has been already extracted and is available from the course website. Each row represents an email. The vocabulary is quite small, containing only 48 different words. The emails may have contained also other words but they have been discarded. Each feature is either 0 or a positive number. The positive number indicates that this word was seen in the respective email at least once. Thus, when building the model, we treat the positive number in the data as 1 (the word was in the email) and 0 means that this word was not present in the email. The correct labels (1 for spam and 0 for non-spam) are in the first column.

3 Prediction

For predicting the label of a test item compute both the probability of the email being spam and non-spam and normalize. If the probability of being spam is greater than 0.5 then classify it as spam, otherwise it should be classified as non-spam. When computing the probabilities keep in mind that the probability of the feature being turned on (word is present in the email) is given by the model, but we also must multiply in the 0-valued features that are just complementary to the events of the feature being turned on.

3.1 Implementation

It is better to keep the model probabilities in log-probabilities because this allows during predicition to sum the probability terms instead of taking the product and in this way we can avoid the possible numeric underflows as when we multiply together many numbers smaller than 1 the result will always get smaller and smaller. When normalizing the log-probabilities, use the log-sum-exp trick, which allows to exponentiate, sum and then log again a list of small numbers given in logs without risking the numeric underflow. In Python, the log-sum-exp is built into the scipy.misc library. It is also relatively easy to implement it, for details look for example `https://hips.seas.harvard.edu/blog/2013/01/09/computing-log-sum-exp/`.

4 Evaluation

For evaluation use the k-fold cross validation. Compute the accuracy (the number of emails with correctly predicted labels) for each fold and print out the average accuracy of all folds.

5 Toolkits

You can also use some toolkit or library to write the wrapper for executing experiments. For python the recommended toolkit is `scikit-learn`: `http://scikit-learn.org`. Look for examples in `http://scikit-learn.org/stable/modules/naive_bayes.html` and the reference in `http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html#sklearn.naive_bayes.BernoulliNB`.

6 Experiments

When writing your own program it is enough to just build a smoothed model and run k-fold cross-validation on it. If you want to, you can also implement a more general smoothing scheme where the smoothed probabilities are computed as:

$$P(w_i|S) = \frac{N_{w_i,S} + \alpha}{N_S + \alpha K}, \tag{4}$$

where K is the number of possible outcomes (2 in our case) and α is a hyperparameter. You can experiment with different α values and find which works the best. A common way to choose a hyperparameter is to try different values on a log-scale (for example 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, etc up to some value, for example 10).

In case you are using library tools you should experiment thoroughly with the options provided by the library and report the configuration that performs the best.

7 Write-up

The report should include a short description of the task, data and the implementation (including how to run it and what options can be set). It should contain the detailed description of the experiments you did, what parameters did you vary, how the results changed with different parameter values.

When you vary some parameter value and record the squared loss with each value, it would be good to represent such results with a figure that plots the accuracy as a function of the parameter.

The report should also state clearly, which setting according to your experiments produces the best results.