Lab 2
Naive Bayes SPAM detector

Handout: 14 September 2012
Deadline: 27 September 2012

September 14, 2012

The goal of this exercise is to build a SPAM detector using the Naive Bayes classifier. Real-world SPAM filters, both filters running on your local machine and system-wide filters running on the servers of service providers, contain a Naive Bayes classifier as part of the system.[1] In this exercise, we will implement and evaluate such a system on real email.

The purpose of the exercise is to gain experience with standard problems of machine learning, including feature selection, building the model and evaluating it. You have two weeks to do this lab. You should get a good, working system in the first week, and use the second week to delve deeper in the feature selection issues.

1 Introduction

The Naive Bayes classifier is a classifier that uses Bayes' rule to compute the posterior probability of the class given the data and makes strong independence assumptions among the features. That is, the posterior probability of a class $p(\mathcal{C}_k|\mathbf{x})$ is given by:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \tag{1}$$

where $\mathbf{x}$ is a d-dimensional vector of features, and the likelihood factorises as

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^{d} p(\mathbf{x}_i|\mathcal{C}_k) \tag{2}$$

For this lab, we will use the occurrence of certain words (or at least parts thereof) as features. The probability of an observation (*i.e.*, an email) given the class (*i.e.*, ham or spam), $p(\mathbf{x}|\mathcal{C}_k)$ is then modelled as the probability of seeing specific keywords in the email.

1.1 The Data

The data consists of email messages which can be categorised in two classes: spam (messages you do not want to receive) and ham (messages you cannot live without). The messages have

[1] However they also use other features, such as whether the source email address is valid, or (in the case of service providers,) the number of people that receive the same message in a given time span: if 2M users receive the same email around the same time, it's not very likely to be legitimate.

been processed in order to remove the headers (for privacy reasons) and in order to remove non-ASCII characters (because MatLab cannot handle those efficiently for our purposes.) The spam has been collected from various accounts, and contain both scams (emails trying to lure and deceive you) and regular spam (advertisement emails.) The "ham", the legitimate emails, are genuine emails from the managers of a company that went bankrupt and were investigated on fraud-related charges.[2] In the process of the investigation, the emails were subpoenaed and released in the public domain. However notice that most of these people were not indicted for fraud — so please do not abuse your right to read and use these emails.

The data has been organised in four directories, containing train and test sets of legitimate email and train and test sets of spam.

1.2 The code

In order to help you in your search for the perfect spam filter, you are provided with some code. While working in MatLab, you can always get some information about this code, using the help command. The code consists of the following functions:

presentre evaluates whether given strings are present in a file or not. The first parameter indicates which file to look at, while the second parameter contains a cell array of the words to check for. Those words are in the form of regular expressions, in order to give you more flexibility to deal with the typical obfuscation present in spam (*e.g.*, 'V1agra', 'C1@lis', ...) The return value of the function is a vector of the same length as the cell array, containing ones for the expressions that were matched, and zeroes for those that were not.

countre is very similar to `presentre`, but rather than checking a single file for regular expressions, it checks all files in a directory, and reports how many files matched each regular expression.

countwords As the name indicates, this function lists all the words that occur in the files of a given directory, and counts the number of files that contain each word. This is useful to find out which words are informative of spam, and which are not. Notice, however, that "informative of spam" does not mean that the word must occur in spam often: a word that occurs a lot in emails from both classes would not be very informative, while a word that often occurs in regular mail and rarely in spam would be useful.

wordtable The previous function uses a MatLab struct to keep track of the words. This is MatLab's slightly convoluted way of providing a hash table. However in order to make computations easier, it is interesting to have a matrix, where the rows correspond to the words and the columns contain the occurrence counts given the classes.

Wordtable returns the list of words as a cell matrix. Cell matrix are more like "real" arrays, in that you cannot do matrix operations on them and each cell can have arbitrary contents (such as strings of variable length.) To access a cell of a cell matrix, use curly brackets rather than parentheses. For example, the element in the third row, second column of cell array **A** is accessed as `A{3,2}`.

Other commands that may be useful are `pwd`, which prints the current working directory, and `ls`, which gives a listing of the files in the directory.

[2] The name of the company was removed from the emails, as it would be an unfair feature to use for discrimination, being specific to the dataset rather than to the problem.

2 Exercise 1: Feature selection **(10 marks)**

The very first problem to solve, when tackling a machine learning problem, is to decide how we perceive the world (or to find it out, if we don't have the luxury of deciding what the measurements are.) In this case, we have access to the complete body of email messages (including the Subject line, for most of them). We will use the words in the messages to decide on the class. The problem is now to decide which words are informative, and which ones are not.

The provided function `presentre` takes the file name of an email and a list of keywords as input, and returns a feature vector. Each element x_i of the vector is either zero or one, depending on whether the corresponding word was present in the email. The probability of an email is then given by

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^{d} \mu_{i,k}^{x_i}(1-\mu_{i,k})^{1-x_i} \tag{3}$$

In this first exercise, we need to find words that are as informative as possible of the class label. For example, we can find such words using the mutual information between the presence of the candidate words and the class label, or by computing the class-conditional probabilities for all the words and finding those words for which the difference between the probabilities is the largest. Do think a bit about this.

Select a set of regular expressions that is informative of the classes, list them your report and describe the selection process. However I would recommend to do this quickly in the beginning (just create a list that you think would be good), and move on to the next part first. It will be more interesting (and more fun) to come back to this part when you have a working classifier, and you can compare how the features impact on the classification performance.

3 Building the classifier **(10 marks)**

Build a naive Bayes classifier using the features you have chosen in the previous part of the exercise and train it. That is, first write the code to compute $p(\mathbf{x}|\mathcal{C}_k)$ based on the output of `presentre` and the model parameters, and use this to compute $p(\mathcal{C}_k|\mathbf{x})$. The training then consists of finding the parameters of the class-conditional distributions $\mu_{i,k} = p(x_i|\mathcal{C}_k)$ and the priors. You can learn the prior probabilities from the data, however it is worth mentioning that the training data may not accurately reflect the real priors. For example, we may train on similar amounts of ham and spam email, while in reality there is much more spam than legitimate email on the internet. Choose a prior that reflects your real prior knowledge, and explain your choice in your report. Then optimise the class-conditional distributions by regularised maximum likelihood, and describe what regularisation you used.

4 Evaluation **(10 marks)**

The final part of the exercise is the evaluation of the classifier. Use the data in the `test` directories[3] to evaluate the misclassification rate with the features you have chosen and the parameters you have learnt from the training data.

[3] Have a look at the provided code to see how to walk through the files.

Spam detection is one of these problems where not all misclassifications carry the same cost. If you let a spam message go through it's a nuisance, while if you discard a real email it can have far-reaching consequences. We can find a trade-off by varying the decision threshold. Plot the ROC curve of your classifier by varying the decision threshold between zero and one, and discuss which threshold you would deem acceptable in a real-world scenario. Compute the AUC of your classifier, and try to optimise that.

Can you also evaluate the expected true error of your classifier? How do you do that?