COMP24111 – Ex 3:
Naïve Bayes Classifier for Spam Filtering

Deadline: See website.
Extension policy: **NO EXTENSIONS** (EVEN ON REQUEST)

In general, a pattern recognition system consists of three modules; i.e., pre-processing, feature extraction and classification. As a typical pattern recognition task, a spam filterer needs to pre-process text messages for cleaning up data, extract salient features for distinguishing between normal and spam mails that form a feature vector for an email message, and classify email messages based on feature vectors for making a decision. While pre-processing and feature extraction require knowledge in the field of natural language processing and computational linguistics, classification needs to find a proper classifier to provide the underpinning technique for decision-making.

Bayesian methods have become very widely used for spam filtering, and you may have such a filter on your computer. Mozilla has a Bayesian filter, Spam Assassin uses Bayesian methods, and even Google uses a Bayesian filter to prevent "search-engine spammers" from dominating search results. In this lab, you are going to implement naïve Bayes classifiers for spam filtering emails based on a UCI machine learning benchmark database named `spambase` (http://archive.ics.uci.edu/ml/datasets/Spambase) where 57 features have already been extracted for each email message and all instances were labelled as "**0**" (normal) or "**1**" (spam). The purpose of this lab is to use Matlab to carry out the naïve Bayes classification algorithm learned from the module to provide underpinning techniques for developing a spam filter working in different situations in order to enhance your understanding of the naïve Bayes classifier and its application to spam filtering. **Note that you are not allowed to use any Matlab NaiveBayes build-in functions in your code.**

In order to apply the discrete-valued naïve Bayes classifier for spam filtering, the `spambase` database has been modified by the further discretisation of continuous attribute values. Also some instances have been re-labelled as "**2**" to indicate "uncertain", a newly added class.

You need to do three things:

1) Set up your path (and ensure it stays set up for every Matlab session)
2) Copy datafiles

then (and ONLY then) …

3) Do the lab exercises.

SET UP YOUR PATH
The following directory needs to be added to your Matlab path:

```
/opt/info/courses/COMP24111/ex3
```

You have to do this *within* Matlab - either with the "addpath" function (type: help addpath) which will modify the path only for the current session, or you can change it permanently with the "pathtool" (type: help pathtool).

COPY DATAFILES
Go to the directory**: `/opt/info/courses/COMP24111/ex3`**
you will find three data files with the suffix `.mat` as follows:

```
av2_c2 - a dataset for binary classification where each
         attribute has two discrete values.
av3_c2 - a dataset for binary classification where each
         attribute has three discrete values.
av7_c3 - a dataset for three-class classification where each
         attribute has seven discrete values.
```

Each of the aforementioned datasets contains the same data format as follows:

```
AttributeSet    –  a M*57 matrix where each row represents the
                   feature vector of an training example, M is
                   the number of training examples.
LabelSet        –  a M*1 column vector where elements
                   represent the labels of corresponding to
                   training examples in AttributeSet.
testAttibuteSet –  a N*57 matrix where each row represent the
                   feature vector an test instance, N is the
                   number of test instances.
validLabel      -  a N*1 column vector where elements
                   represent the labels corresponding to
                   test instances in testAttributeSet (used
                   to achieve the accuracy of a classifier).
```

In addition, a function (`main.m`) is also provided to load a dataset for training and testing your implementation. As a result, you need to embed your code into this `main` function for evaluation. After embedding your code to this `main` function, you need to run this function only.
A typical scenario for machine learning is to create a learning system by training it on a given training data set. Later on the system will be applied to different test data sets. As a result, *you must write two generic functions; i.e., one for training a naïve Bayes classifier and the other for test based on the trained naïve Bayes classifier so that they can be applied to* ***any datasets*** *with the* `main` *function.* A typical format is shown below where "Parameter List" stands for a set of data structures (e.g., vector and matrix) used to store conditional probabilities and prior probabilities to be learned. You need to design such data structures yourself for this purpose.

```
% for NB training
[ "Parameter List" ] = NBTrain(AttributeSet, LabelSet)
% for NB test
[predictLabel, accuracy] = NBTest("Parameter List",
                                  testAttributeSet, validLabel)
```

Note that the `accuracy` of a naïve Bayes classifier is defined as follows:

$$accuracy = \frac{\text{number of correctly classified test instances}}{\text{number of test instances}}$$

For example, if you are given a test dataset of 100 instances and your code correctly predicts labels (i.e., predicted labels are as same as their true labels) for 85 instances, the accuracy is 85/100 = 0.85. Apart from the accuracy measure, you also need to report test results with a confusion matrix as described in lab 2 by appending the corresponding code to the end of the `main` function. If you wrote your code for calculating a confusion matrix for lab 2, you can reuse it here.

Now… the lab exercises…..

PART 1 – Implementation for discrete input attribute values

The main task of this exercise is to implement a naïve Bayes classifier for discrete input attribute values (see lecture notes of Naïve Bayes Classifier for details). Your code must be able to handle all possible situations; i.e., different attributes could have a various number of values (e.g., the first attribute has two values, the second one has four values and so on) and a problem could be either a binary or a multi-class classification task. If you do not have a clear idea of implementing a generic naïve Bayes classifier at the beginning, you might want to start coding a simplified naïve Bayes classifier for a simple situation, e.g., all attributes have only two values and the task is binary

classification. After gaining the experience on simple situations, you should be able to work on a generic naïve Bayes classifier. With the `main` function, you should test your code based on three datasets of discrete input attribute values. In terms of functionalities, three goals are set for Part 1:

1. your code works for binary attribute values and binary classification tasks,
2. your code works for multiple attribute values and binary classification tasks
3. your code works for different attribute values and multi-class classification tasks.

PART 2 – Bonus marks

Additionally, bonus marks are available for truly exceptional students: those able to show observations and knowledge that was not explicitly supplied in lectures. To restate - to obtain marks in this category you should **work independently and show evidence of learning outside all the supplied lecture notes in this course unit and gaining deeper understanding of machine learning. It is worth stating that marking this part is based on not only the improved performance achieved but also your motivation and understanding on algorithms you applied.** Examples of things you could do and good practice in the past years include feature extraction from real email messages, or apply other forms of non-trivial evaluation beyond those learned from this course unit to the given datasets (with a satisfactory justification), or other feature extraction/selection routines or an extension of the naïve Bayes classifier or other suitable classifiers that can improve the classification performance on this database etc. The original database `spambase.data` is given under the same directory as other datasets are located in so that you can use it if needed. If you work on feature extraction, you can use any non-trivial open source spam mail datasets (e.g., some in http://www.esi.uem.es/~jmgomez/spam/).

DELIVERABLES

By the deadline, you should submit two files, using the **`submit`** command as you did last year:

```
ex3code.zip      - all your .m files, zipped up
ex3report.pdf    - your report, as detailed below
```

A report should be submitted, on **one side of A4** as a PDF file.

Anything beyond 1 side will be ignored. It is essential to give a description of "Parameter List" you designed to store conditional probabilities and prior probabilities in Part 1, how those probabilities are estimated and test results on the given data sets. Take note, we are not interested in the details of your code, e.g., what Matlab functions are called, what they return etc. This module is about algorithms and is indifferent to how you program them.

There is no specific format – marks will be allocated roughly on the basis of:

- functionality of your program and rigours experimentation,
- knowledge displayed when talking to the demonstrator,
- imagination and deeper understanding of machine learning in Part 2,
- how informative in your report about your experiments,
- grammar, ease of reading.

The lab is marked out of 20:

Part 1 – Implementation for discrete input attribute values	(**14 marks)**
Part 2 – Bonus	(**6 marks)**

Bonus marks will not be given unless you engage in materials OUTSIDE the scope of all the lectures, have a clear motivation (instead of trial-and-test) and exhibit deeper understanding of machine learning via your demonstration and question answering.