# Practical 7: Natural Language Processing (version 1)

## 1 Overview

The aim of this lab exercise is to give you some hands-on experience with **Natural Language Processing (NLP)** tools. In this practical exercise, you will create a simple SPAM detection programme.
For this exercise, you will be using the **nltk** toolkit. This should be installed as part of your Anaconda Python installation, but if not, you can install it by running (at the $ prompt):

```
$ conda install nltk
```

Information about NLTK is available online here: `http://www.nltk.org/`

## 2 Obtain a corpus

The first step in the exercise is to obtain a **corpus** (a set of text) to work with. I have put on KEATS a data set containing over 5000 SMS messages. This data set is known as the *SMS Spam Collection v.1* and is publicly available via the UCI Repository.

1. Start by downloading the corpus (in a zip archive) from KEATS.
2. Unzip the archive and take a look at the data file.
3. You will see that each line begins with a single word that is a **tag** or **label** or **class** — either *ham* or *spam*.

   **Note** that the label is separated from the rest of the line—the actual SMS message—with a tab (`\t`) character.
4. Write a Python programme that will open the file and count the number of lines that are labelled *ham* and the number of lines that are labelled *spam*.

   **Note:** this data set is not as clean as others we have provided you with. You may find that some of the lines contain characters that you cannot process. For this exercise, you can just skip those lines.

## 3 Split the data set

The next step in this exercise is to divide your data into a **training set** and a **test set**.
Remember that a good training set will be **balanced** and contain similar numbers of labelled items you are trying to learn. Suppose your data has two classes: A and B. Suppose you have $1000$ entries labelled A and only $200$ entries labelled B. You can either *boost* the number of B entries (from $200$ to closer to $1000$) by increasing the number of B entries; or you can *filter* the number of A entries (from $1000$ to closer to $200$). If you decide to boost the B entries, then you can do

that in (at least) two ways. The simplest way is just to create additional copies of entries that are already there. You can do that by randomly selecting $1000 - 200$ B entries and copying them in your training set. Another way is to create synthetic data that is similar to the B entries that are already there. You can do that by building a model of the existing B entries and then generating new entries by randomly selecting attribute values that fall within the distribution of those belonging to B. Obviously this second way is more complicated and so I suggest you start with the easy way and come back to the harder way later—it will be easier after you've done more of this lab exercise. If you decide to filter A entries, then make sure that you randomly select the A entries to keep in your training set.

1. Based on the statistics you computed in Part 2, decide how many records of each class to put in your training set and in your test set.
2. Create a balanced data set for training.
3. Create a test set that contains different records from the training set.

## 4 Characterise the training data set

We need to figure out how to characterise records that are spam versus not spam ("ham"). Let's start by computing a set of attribute values for both types of entries in your training data set. Here's a set of attributes to start with:

- message length in words
- message length in characters
- number of digits in message
- number of punctuation characters in message
- number of uppercase characters in message

If you think of other attributes that you want to compute, then you can add them to this list.
Note that the

1. Create a matrix for your training data set, using the list of attributes (above) as columns in the matrix. Each row in the matrix is a message in the training set. Compute the feature values for each message to fill in the matrix.

   You will likely find it handy to use functions from **nltk** in order to compute some of these attribute values, such as:

   - **nltk.word_tokenize()** which takes a string argument and returns a list of all the words in the string

2. Compute some statistics based on the data in your matrix, within each class (spam and not spam). Determine which statistics give you the best ways of distinguishing between spam and ham in your data.

## 5 Train your classifier

Now that you've created a matrix with which you characterise your training data set, try training a classifier using that matrix.
You can use any of the classifiers we've discussed in class. As you know, there are classifiers in **scikit-learn**. There are also classifiers in **nltk**.

1. Start by using the *multinomial Naive Bayes classifier* in scikit-learn: `http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`
2. Compute an attribute vector for each record in the test data set and user your classifier to predict the test data labels.
3. Store the number of true positives, true negatives, false positives and false negatives that you predict.
4. Print the **confusion matrix** to evaluate the success of your classifier.
5. Compute and print the values of **precision**, **recall** and **F1**.

Once you have gotten this working, you can go back to Part 4 and try to improve your classification results by changing the attributes in your matrix.