# Mining of Massive Datasets:
# Lab on Naive Bayes and Decision Trees

Mauro Sozio
Oana Balalau (TA), Maximilien Danisch(TA)
J.B. Griesner(TA), Raphael Charbey(TA)
firstname.lastname@telecom-paristech.fr

This lab session consists of two main exercises: Naive-Bayes classifier and decision-tree classifier. We are also going to evaluate the models we build with a $k$-fold cross evaluation and by computing a confidence interval, respectively. You should send the code as well as a report (1.5 pages max.) to the TA in charge of your classroom. The deadline for this lab session is **02/11 at 1145**. Late submissions will not be considered.

## 1 Naive Bayes Classifier (8/20 points)

We are going to train a naive bayes classifier for the same problem we considered in the previous lab session, i.e. predict whether a given document deals mainly with apple the IT company or the fruit. You can use the same collection of documents you used last time or another collection of documents. Similarly to last time, we turn the documents into vectors in the euclidean space as follows:

```
count_vect = CountVectorizer(stop_words='english')
vectors = count_vect.fit_transform(dataFiles)
X=vectors.toarray()
```

We can compute a random permutation of the instances as follows:

```
X=np.random.permutation(X)
```

We then train the classifier Naive Bayes classifier as follows

```
clf = MultinomialNB()
clf.fit(X_train, X_train_Cl)
```

where we use the multinomial Naive Bayes classifier `http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB`. We then predict the class of instances in the test set by using: $clf.predict(X\_test)$.

### 1.1 $k$-fold cross validation

In this part of the question, we are going to implement a $k$-fold cross validation. Even if there are modules taking care of this, in order to get full points for this question you are supposed to provide your own implementation. In order to evaluate a classifier with such an evaluation method, we first split the instances into $k$ subsets with the same size (you can assume that the number of instances is a multiple of $k$). Each subset is used as test set exactly once while the rest of the subsets are used as training set. Therefore, this requires to run a given classifier $k$ times. Each time, the accuracy is computed as the fraction of instances (documents) in the test set that have been classified correctly. The overall accuracy of the classifier is then measured as the average accuracy of the classifier over the $k$ evaluations. In order to implement the $k$-fold cross validation the following could be helpful:

```
import numpy as np
a=[1,2,3,4,5,6,7,8,9,10]
a=np.array(a)
b=a[0:4]
c=a[6:]
np.append(b,c,axis=0)
#prints array([ 1,  2,  3,  4,  7,  8,  9, 10])
```

### 1.2 Report

Include in your report the average accuracy of the Naive-Bayes classifier on your collection of documents.

## 2 Decision Tree Classifier

We are going to focus on the Iris flower dataset, which was introduced by Ronald Fisher in a 1936 research paper about classification. It is a well-known dataset in data mining and machine learning which is relatively easy to study. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). For each sample, four features are considered: the length and the width of the sepals and petals, in centimetres. We are going to train a decision tree classifier which is able to predict the species (class) of a given flower given the values of those four features. Informations about the Iris dataset can be found here: `http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html`. In our case, we have split the dataset into training and test sets in a "bad way" leading to overfitting. After building the decision tree, you are asked to overcome the overfitting problem.

**Overfitting**. In classification tasks, the main goal is to be able to build a model (e.g. a decision tree) which determines the (unknown) class of a given instance. The effectiveness of a given model is measured by how well the model is able to determine the class in *unseen* data. To be able to determine the effectiveness of a model, the dataset is split into training set and test set. The model is then built starting from the training set and evaluated on the test set, which has not been seen by the classifier. We have overfitting when the model classifies very well the instances in the training set but performs poorly on the test set (and likely on other unseen data). This happens when the model "memorizes" the training data rather than "learning" from the data and generalizing to unseen data. Therefore, "complicated" models are often the results of overfitting, with one extreme case being when the model is as large as the training data itself. In our class, we have seen a few strategies to overcome overfitting.

The exercise consists of the following steps:

1. Building the decision tree and print the resulting decision tree on a pdf file.
2. Computing the confidence interval on the accuracy of the decision tree.
3. Overcoming the overfitting issues.

### 2.1 Building the Decision tree (4/20 points)

We start by loading the data by means of *numpy.load*() as follows:

```
import numpy as np
training_data = np.load('training_data.npy')
```

You are provided with the files

`training_data, training_class, test_data, test_class`

containing the training data, the class for each instance in the training data, the test data, and the class for each instance in the test data, respectively. The type of all those files is *numpy.ndarray* (see `https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html` for the documentation), therefore the type of *training_data* and the other variables is *numpy.ndarray*.

We will build the decision tree by means of *tree.DecisionTreeClassifier* (`http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`), which is done as follows:

```
clf = tree.DecisionTreeClassifier(criterion='gini', random_state=RandomState(130))
clf = clf.fit(data,class)
```

where 'gini' specifies the criterion according to which the decision tree is built (as we have seen during our class), while $random\_state = RandomState(130)$ makes the algorithm to build the tree deterministic. Do not change those parameters in this part of the exercise. Prediction can be done by $clf.predict()$.

In order to visualize the decision tree and print it on a pdf file, you can use *graphviz* `http://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html`. This yields a '.dot' file which should be converted into a pdf file by means of the *dot* command in Linux or by using the Python modules *pydotplus* or *pydot* (see for example `http://scikit-learn.org/stable/modules/tree.html#tree` and `http://pydotplus.readthedocs.io/reference.html`).

## 2.2 Computing Confidence Interval (2/20 points)

The confidence interval for the accuracy of the classifier can be computed as follows:

```
stats.norm.interval(0.95,loc=numpy.mean(a),scale=stats.sem(a))
```

where 0.95 specifies the confidence, while $a$ is an array where $a[i] = 1$ if and only if the $i$th instance has been classified correctly.

## 2.3 Overcoming Overfitting (6/20 points)

The decision tree built on *training_data* is very good at classifying the instances in the training data but performs poorly on the test set. This is a classical example of overfitting. In order to overcome this problem several strategies are possible, such as *early stopping rule*, *postprocessing*, etc that we mentioned during our class (see the slides on the decision trees). You are free to tune the parameters of the $tree.DecisionTreeClassifier$ so as to overcome the problem of overfitting. You should try to do that by looking only at the structure of the tree, in particular without taking into account the accuracy of the decision tree on the test set.

## 2.4 Report

You should write a report (one page max.) including the visualization of the decision tree with overfitting as well as a visualization of the decision tree after your method to fix the issue has been applied. You should include also the confidence intervals for both cases. You should explain how did you come up with such a method and why you think that your method improves the accuracy of the classifier on the test set (if this is the case).