Mining of Massive Datasets: Lab on k-Nearest Neighbor Classifier

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

In this lab session, we are going to train and test a k-nearest neighbor classifier on a collection of documents. In particular, we are going to train a binary classifier which determines whether a given text document deals either with "apple" the fruit or "apple" the software company. This lab session should be done individually and the results (code + report) should be submitted before the end of the lab, i.e. **11.45am**. Students should send their code in Python and the report to the teaching assistant in charge of their lab room. Each email should have [**miningLab**] in the subject of the email. Failure to do so might result in failure of the lab session.

The lab consists of the following three steps:

1. Collecting a set of documents (in English), each of them dealing with either "apple" the fruit or "apple" the company. They can be downloaded from the Web, we recommend at least 10 text documents.

2. Splitting such a dataset in *training set* and *test set*. To this purpose, we use a hold-out evaluation method where $\frac{2}{3}$ of the documents are used as training set and the remaining $\frac{1}{3}$ as test set. We recommend to do the splitting uniformly at random.

3. Learning the k-nearest neighbor classifier on the training set and evaluation of the classifier on the test set . You should report the *accuracy* of the classifier which is defined as the number of documents in the test set that are classified correctly divided by the number of documents in the test set. Choose the value of k which maximizes the accuracy.

4. Write a report explaining what you did (max 1 page, 11pts font). In particular the report should include: 1) how many documents you have for each class. Moreover, provide some info about the source of the documents (e.g. Wikipedia, which website etc.) 2) show the accuracy for at least 3 different values of k. Does the accuracy improve as a function of k? Do you expect it to increase or decrease as a function of k? Motivate your answers.

5. Once the previous steps are complete, try to improve your classifier by removing stop-words, using a tf-idf representation for the collection of documents, or anything else that you think it might improve the results. (see documentation on the sci-kit learn package). Write in the report what you used to improve the accuracy and argue why you think it

should improve the results. Report the results and explain why you got those results. In this case the report can be at most two pages long. This step contributes 4 points (out of 20) to the final grade of the lab (the other points contribute 16 points in total).

You are going to be evaluated according to the code you wrote, the report, and the accuracy of your classifier on our own test set. In the next section, you will find more info about how to learn a k-nearest neighbor classifier in Python.

1 k-nearest neighbor classifier in Python

We are going to use the sci-kit learn library (`http://scikit-learn.org/`) in Python, which is a project sponsored by INRIA, Telecom ParisTech (where the main contributor graduated in 2004) and Google. The first step is to turn a collection of documents into vectors in the Euclidean space. To this end, we are going to use $CountVectorizer$ (see documentation at `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html` and the tutorial at `http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`). The first thing to do is to import the relevant libraries.

```
import sklearn as sk
from sklearn.feature_extraction.text import CountVectorizer
```

Next we use CountVectorizer to turn our training set into a set of points in the euclidean space. In particular the method

`count_vect.fit_transform()`

receives in argument a list of strings, each of them representing the text of the corresponding document. It builds a so called term-document matrix, where each row represents a document while each column represents a term (or word in the document). Each entry of such a matrix represents the number of occurrences of the corresponding term in the corresponding document. This is done as follows.

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(trainingFiles)
```

One could specify to ignore stopwords (such as articles and common adjectives) which are not informative and might lead to bad results. This is done by specifying

`stop_words='english'`

as an argument of

`CountVectorizer()`

Next, we need to convert the results as an array, which is done as follows:

```
training=X_train_counts.toarray()
```

Next, we are going to train our classifier on our training set. We use KNeighborsClassifier (see documentation at `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`) as follows:

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=k)
neigh.fit(training,trainingCl)
```

where k specifies the value of the parameter k in the k-nearest neighbor classifier, while trainingCl is a list specifying for each of the document its training class (i.e. in our case whether the document is about the fruit or the computer).

For evaluating the classifier on the test set we can use the method

```
transform(testFiles)
```

of $CountVectorizer()$ to turn the documents into vector as well as the method $score(test, testCl)$ of KNeighborsClassifier, which computes the accuracy of the classifier on *test* (an array of text documents) whose class is specified by a list *testCl*.