Text Analytics 2018
Homework 2

Naïve Bayes Classifier

A Naïve Bayes classifier is a generative classifier. You will have to build such classifier for classifying movie reviews.

Download the Movie Review dataset from:

```
http://www.cs.cornell.edu/People/pabo/movie-reviewdata/review_polarity.tar.gz
```

It contains 1000 positive and 1000 negative reviews, preprocessed and tokenized. The directory neg contains the negative reviews, the directory pos contains the positive reviews.

The collection is split into 10 folds: fold 1 consists of documents in files cv000* to cv099*, fold 2 those in files cv100* to cv199*, etc.

Create a Naïve Bayes classifier. It should consist of two parts:

a) Training: the program takes two lists of files: one containing the positive review files, the other containing the negative ones, and outputs a model file.
b) Testing and evaluation: the program takes a model file and two lists of files: positive and negative review lists, and outputs the classification accuracy for the test set, in terms of accuracy, precision, recall and F1 measure.

Test the classifier on the Movie Review dataset, using fold 10 for testing and the rest for training.

Optional. Explore the technique of cross validation, i.e. perform 10 runs of the classifier, using a different fold for testing at each run and compute the average accuracy, precision, recall and F1 macro-average of the 10 runs.

POS Tagger with a Maximum Entropy Classifier

A Maximum Entropy Classifier is a discriminative classifier. You will have to train a POS tagger using the nltk module `classify.maxent`.

The training and test set of the Evalita 2016 POSTWITA task can be found in the folder:

```
Text Analytics/data/POSTWITA/
```

on the Jupyter server at:

```
attardi=4.di.unipi.it:8000
```

The input consists of sentences, one token per line, separated by one empty line.

Each line contains two, tab separated, fields:

```
FORM    POSTAG
```

For example:

```
Ma          CONJ
Il          DET
Governo     PROPN
Monti       PROPN
Ha          AUX
Fatto       VERB
Quello      PRON
Che         PRON
chiedeva    VERB
l'          DET
```

```
Europa      PROPN
?           PUNCT
```

To decide which POS to assign to a given token, the classifier is given a set of *features* to represent the token. For example, such features may include orthographic features, such as whether the word starts with a capital letter, or it contains a number, or any hyphenation, or position of the token the sentence. You can use regular expressions to check patterns in the words. You can also use feature conjunctions, such as whether the word is capitalized AND the value of the previous label.

The features describing a token are encoded using a *featureset*, which is a dictionary that maps from *feature names* to booleans. Feature names are unique strings that indicate what aspect of the token is present/absent in the token. Examples include: "Cap" with value True, which means that the word is uppercased, "containsDigit" with value True means that the token contains a digit, while value False means it does not, "prevtag=DET" means that the tag of the previous token is "DET ", or "ends-with=ando" will be True for a token that ends with "ando".

In order to train the classifier you will have to create a list of training examples. Each example consists in a pair:

```
(featureset, label)
```

To create a classifier on those examples, call:

```
classifier = MaxentClassifier.train(train_examples)
```

Use the same feature extractor for implementing the tagger, and perform tagging with:

```
Tags = classifier.classify(test_examples)
```

Check its accuracy by computing the macro-average F1 on all POS tags.