Tim auf der Landwehr
t.auf.der.landwehr@student.rug.nl
S2548682
Last revision: November 24, 2013

Learning from Data
Assignment 2

Summary

This is the assignment 2 in the course Learning from Data at the University of Groningen.

Exercise 1: Preprocessing

Feature generation

The script `feature_generator.py` creates a file called `features.txt` that contains all the features. The script can be modified to produce different feature sets. By setting the parameter `--out filename` the output file name can be set manually.

.arff-file generation

The script `arff_generator.py` is able to generate an `.arff`-file that can be handled by weka. The script can be used with or without the following parameters:

- `--dutch [file]`
  This file should contain the dutch tweets, default: `NL.txt`
- `--other [file]`
  This file should contain the other tweets, default: `OTHER.txt`
- `--features [file]`
  This file should have been generated by `feature_generator.py` and contain the features, default: `features.txt`
- `--out [file]`
  This is the output file, default: `NL_OTHER_features.arff`

The output file can be opened with *Weka*. The only thing that has to be considered is, that the class of a tweet is specified in the first element of a line in the sparse notation. Therefore any classifier of *Weka* has to be run with the parameter `-c 1`.

Exercise 2: Feature selection

My dataset contains the following 76 features:

1 !	9 -	17 5	25 ?	33 G	41 O	49 Z	57 g	65 o	73 w
2 "	10 .	18 6	26 @	34 H	42 P	50 _	58 h	66 p	74 x
3 #	11 /	19 7	27 A	35 I	43 R	51 a	59 i	67 q	75 y
4 ’	12 0	20 8	28 B	36 J	44 S	52 b	60 j	68 r	76 z
5 (	13 1	21 9	29 C	37 K	45 T	53 c	61 k	69 s	
6)	14 2	22 :	30 D	38 L	46 U	54 d	62 l	70 t	
7 +	15 3	23 ;	31 E	39 M	47 V	55 e	63 m	71 u	
8 ,	16 4	24 <	32 F	40 N	48 W	56 f	64 n	72 v	

A set of features consisting of words could probably improve the results of the classifiers used in the follwing tasks, but unfortunately I was not able to generate the `.arff` file for such a set of features on my computer in a reasonable amount of time.

The simplest baseline for this classification problem is marking all tweets as ’dutch’ or ’other’. For the given dataset a classifier, which marks all tweets as ’dutch’ would get an accuracy of 0.66 whereas a classifier that always classifies as ’other’ would get an accuracy of 0.34. This is simply caused by the distribution of the classes in the given dataset.

Exercise 3: Naive Bayes

Now the *Weka* implementation of Naive Bayes is run on the generated `.arff` file from the command line:

```
java -cp weka-3-6-10/weka.jar weka.classifiers.bayes.NaiveBayesMultinomial -D -c 1 -t
    NL_OTHER_features.arff
```

The output is shown in listing 1. Naive Bayes has an accuracy of 74.8309% on this dataset. This is about 2 % less than my own implementation in assignment 1. The explanation for this difference is simply the difference in the feature set. My own implementation used all letters that it found in the dataset as features. The set of features here is limited to the 76 features shown in exercise 2. From the confusion Matrix in the output you can see, that the accuracy for dutch tweets is much better that for other tweets: 35201 of 41940 ($\approx$ 83.93%) Dutch tweets were classified as Dutch, but only of 21340 ($\approx$ 56.94%) other tweets were classified as other. So other tweets get misclassified more often than Dutch tweets.

Listing 1: Output of the NaiveBayesMultinomial classifier

```
=== Stratified cross-validation ===

Correctly Classified Instances   47353             74.8309 %
Incorrectly Classified Instances 15927             25.1691 %
Kappa statistic                      0.4206
Mean absolute error                  0.2632
Root mean squared error              0.4434
Relative absolute error             58.8839 %
Root relative squared error         93.7785 %
Total Number of Instances        63280

=== Confusion Matrix ===

     a     b   <-- classified as
 35201  6739 |     a = NL
  9188 12152 |     b = OTHER
```

Exercise 4: Perceptron

The *Weka* implementaion of Perceptron is run in the same way as Naive Bayes before:

```
java -cp weka-3-6-10/weka.jar weka.classifiers.functions.VotedPerceptron -c 1 -t
    NL_OTHER_features.arff
```

The output is shown in listing 2. The accuracy of Perceptron is with 82.1508% approximately 7% higher than the one of Naive Bayes. But the bias is is even worse: 37160 of 41940 ($\approx$ 88.60%) Dutch tweets and 12133 of 21340 ($\approx$ 56.85%)other tweets were classified. From these values you can see, that the Perceptron only improved the classification of Dutch tweets.

Listing 2: Output of the VotedPerceptron classifier

```
=== Stratified cross-validation ===
```

```
Correctly Classified Instances   51985             82.1508 %
Incorrectly Classified Instances 11295             17.8492 %
Kappa statistic                      0.5926
Mean absolute error                  0.1785
Root mean squared error              0.4225
Relative absolute error             39.9352 %
Root relative squared error         89.3674 %
Total Number of Instances        63280

=== Confusion Matrix ===

     a     b   <-- classified as
 37160  4780 |     a = NL
  6515 14825 |     b = OTHER
```

Python3 Implementation of Perceptron

My own implementation of Perceptron is provided in `perceptron.py`. A 10 fold cross validation on the given dataset can be run using the parameter `--run-cross-validation`. This needs the files `features.txt`, `NL.txt` and `OTHER.txt` to lie in the same directory. The features file can be created using the script `feature_generator.py` introduced in exercise 1.

By default this implementation of perceptron uses only 1 iteration over the training data. This is aparently not good enough to reach the results that are achieved by the *Weka* implementation of perceptron. Additionally I figured, that the accuracy actually varies when run several times on the same data. This can be explaned by the fact, that the input data is shuffled each time before one training iteration, and the success of the training depends on the order of the training data. These issues explain, why my implementation does not produce the same results as the *Weka* implementation.