Data Mining
Homework Assignment #10

Dmytro Fishman, Anna Leontjeva and Jaak Vilo

April 25, 2014

From the lecture we know that in order to classify two-dimensional points using a linear classifier following classification rule must be applied for each point (x_1, x_2):

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b) = sign(w_1 x_1 + w_2 x_2 + b),$$

where (w_1, w_2, b) are the parameters that learned from the data (it should remind you of $(ax + by + c)$.

Task 1

Consider the training set {((0,0), 1), ((1,0),1), ((2,3),-1), ((3,2),-1)}, consisting of two-dimensional points, each one having class label 1 or −1. Depict the points on a plane (you can use function plot.data() from the attached code file or pen and paper) and sketch the location of the maximal margin separating hyperplane. Small recap from the lecture: margin is the distance to closest point in each class. The points with the smallest margin are called support vectors.

- Which points are the support vectors?
- How many support vectors you would expect to have on average (e.g. for randomly generated data points)? Explain.

Task 2

Consider now another training set {((1,1), -1), ((1,3),-1), ((2,5),-1), ((4,2),-1), ((5,5),1), ((5,9),1), ((6,1),-1), ((7,5),1), ((9,1),1), ((11,3),1), ((11,7),1)}. You happened to know that 5th, 7th and 9th points are support vectors. Plot these data. Find maximal margin separating hyperplane from these support vectors and calculate corresponding margin. Hint:

- you will probably need to find the equation of the line that goes through points that belong to the same class.

- resulting separating hyperplane should be parallel to this line, explain why.
- it should be equally distanced from all support vectors.

Task 3

On the lecture we discussed the perceptron algorithm, which is a linear classifier that for a given dataset $\{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \mathbb{R}^2, y_i \in \{-1, 1\}$ finds a separating hyperplane in the following way:

- Start with $\mathbf{w} = (0, 0)$ (let us assume that b also equals to 0 for simplicity).
- Find an item $\{(\mathbf{x}_j, y_j)\}$ for which, $sign(\mathbf{w} \cdot \mathbf{x}_j + b) \neq y_j$ and update the parameter $\mathbf{w}$ in the following way:

$$\mathbf{w} = \mathbf{w} + \mu \sum_j \mathbf{x}_j y_j,$$

  where μ is a learning rate of our algorithm.
- Repeat this procedure until there is no such item, return $\mathbf{w}$.

First, try to understand the text above. One of the possible implementation of this algorithm is given to you in the code.r. Your task will be to correct one line of this code to make it work as expected (all the instances are classified correctly). Include resulting plot into the report.

- How many steps did it take to converge to the final solution?
- Try changing μ. Does anything change besides the scale?

Task 4

Sauron case study again. In the last (not-official) homework assignment (`http://fouryears.eu/2009/11/21/machine-learning-in-mordor/`) you had to analyze and predict which student Aghargh or Bughrorc has better chances of not jumping from the tower. In this exercise will try to simulate Sauron's case ourselves. Let us supposed that the second student got really lucky and guessed a proper representation for the spells. That is, indeed each spell can be represented as a set of letters. When a spell is chosen from the Great Book at random, each letter will appear independently with probability 0.3:

```
make_one_random_spell = function() {
rbinom(26,1,0.3)
}
```

Let us also suppose that *true* classification of the spells can be achieved using a linear classifier:

```
true_spell_class = function(spell) {
   w = c(1,-2,3,-4,5,-6,7,-8,9,-10,11,-12,13,-14,15,
        -16,17,-18,19,-20,21,-22,23,-24,25,-26)

   sign(spell %*% w - 27.5)
}
```

Finally, let us assume that the second student got even more lucky because for training he attempted to fit exactly the *linear* model to the data, using the SVM algorithm. For simplicity let us suppose that instead of doing the irrelevant training/testing set split, the student just used all 20 instances to train the model. In this case, the situation that happened with Sauron and the students can be simulated as follows:

```
   # Sauron generates a dataset:
   spells = make_n_random_spells(20)
   c = true_spell_class(spells)

   # First student finds the majority class
   majority_class = sign(sum(c) + 0.5)

   # Second student trains an SVM:
   svm_model = svm(spells, c, type='C', kernel='linear')

   # Sauron generates a new example:
   test_example = make_one_random_spell()

   # .. and tests each student's predictions:
   student1_correct = (majority_class ==
       true_spell_class(test_example))
   student2_correct = (predict(svm_model, t(test_example)) ==
       true_spell_class(test_example))
```

Simulate this situation 1000 times or more, and observe:

- How often the first student guesses correctly (i.e. what is his method's expected generalization error)?
- How often would the second student guess correctly?
- In those cases when the predictions differ, how often is the first student right?

Task 5

Could you somehow improve the second student's performance by, say, doing training an a smarter way, tunning method parameters or changing the number

of attributes used in training? What happens when the dataset becomes larger (i.e. increase n to 30, 40, ... , 100, 200, ...)?

Task 6

In this exercise we shall use the chord sequences of some popular songs by *The Beatles*, *Paul McCartney*, *Eric Clapton*, *Red Hot Chili Peppers*, *Metallica* and *Nirvana* from the website `http://www.e-chords.com`. Analyze this dataset using whatever machine learning or data mining methods you find applicable. As a minimum, you could check whether it is possible to discriminate the artists on the basis of the chord sequences. However, you need not limit yourself to classification only. Other options, such as clustering, automated chord sequence generation, frequent pattern detection, association rule mining or visualization might be rather exciting. Report your findings.

This homework includes a lot of small and large chunks of text stolen from homeworks, tutorials and blogs published by Konstantin Tretyakov, yappie! ☺