MTAT.03.183 Data Mining

Machine Learning, Part III. Home Assignment.

Konstantin Tretyakov

Due: November 26, 2009

1. Support vector machines. Consider the training set $\{(0,0),1),((1,0),1),((2,3),-1),((3,2),-1)\}$, consisting of two-dimensional points, each one having a class label 1 or -1. Depict the points on a plane and indicate a separating hyperplane with the maximal margin. The points with the smallest margin are called *support vectors*. Which points are the support vectors?

2. Notion of a kernel. Suppose that we are classifying two-dimensional points using a linear classifier. That is, for each point (x_1, x_2) our classification rule looks as follows

$$f(\mathbf{x}) = \operatorname{sign}(\mathbf{w}\cdot\mathbf{x}+b) = \operatorname{sign}(w_1x_1+w_2x_2+b)\,,$$

where (w_1, w_2, b) are the parameters, which we somehow learn from the data.

Let us now transform our points $\mathbf{x}$ to a higher-dimensional space using a *feature map* ϕ of the following kind:

$$\phi(\mathbf{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2).$$

Note that if we apply the linear classifier to the transformed points, we won't be dealing with a linear function of $\mathbf{x}$ any more, but with a polynomial:

$$f(\phi(\mathbf{x})) = \operatorname{sign}(\mathbf{w}\cdot\phi(\mathbf{x})+b) = \operatorname{sign}(\sqrt{2}w_1x_1+\sqrt{2}w_2x_2+\sqrt{2}w_3x_1x_2+w_4x_1^2+w_5x_2^2+b)\,.$$

Finally, note that our learning algorithm can be implemented so that it would only need the inner products $\phi(\mathbf{x})\cdot\phi(\mathbf{y})$. It turns out that it is possible to compute these inner products in a "shortcut", without explicitly computing $\phi(\mathbf{x})$.

Your task here is to understand the meaning of the text above and also show that in our example

$$\phi(\mathbf{x})\cdot\phi(\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^2-1$$

3. Kernelized perceptron. The perceptron algorithm is a linear classifier, which, for a given dataset $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \mathbb{R}^m$, $y_i \in \{-1, 1\}$ finds a separating hyperplane $(\mathbf{w}, b)$ in the following way:

- Start with $(\mathbf{w}, b) = (\mathbf{0}, 0)$
- Find an item $(\mathbf{x}_j, y_j)$ for which $\text{sign}(\mathbf{w} \cdot \mathbf{x}_j + b) \neq y_j$ and update the parameters:
  $$\mathbf{w} := \mathbf{w} + y_j \mathbf{x}_j \qquad\qquad b := b + y_j$$
  Repeat.
- If no such item exists, return $(\mathbf{w}, b)$.

We shall now transform this algorithm to a *kernelized form*, that is, make sure that the algorithm only uses the training set via the inner product (i.e. kernel) function. For that we only need to represent $\mathbf{w}$ as:

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$$

and work with the values α_i instead of $\mathbf{w}$. The algorithm will then look as follows:

- Start with $(\boldsymbol{\alpha}, b) = (\mathbf{0}, 0)$
- Find an item $(\mathbf{x}_j, y_j)$ for which $\text{sign}(\sum_i \alpha_i \mathbf{x}_i \cdot \mathbf{x}_j + b) \neq y_j$ and update the parameters:
  $$\boldsymbol{\alpha} := ??? \qquad b := b + y_j$$
  Repeat.
- If no such item exists, return $(\boldsymbol{\alpha}, b)$.

Questions:

- What is the update step for $\boldsymbol{\alpha}$? Can you interpret the final value of α_i?
- Can you now plug in arbitrary kernels into this algorithm? How?

4. Learning with kernels. Open the file `languages.arff` in Weka. This is a small dataset of Estonian and English phrases, each phrase labeled with the corresponding language. Try building an SVM classifier for the languages in two different ways.

- Firstly, attempt converting strings to word vectors using the *StringToWordVector* filter and apply the SVM classifier (*weka.classifiers.functions.SMO*) with the *linear kernel.*
- Secondly, instead of converting strings to word vectors use a *string kernel* within the SVM. Train a language classifier using string kernel and compare its performance to the previous case. Can you explain the observed difference in performance? Hint: the string kernel computes similarity of pairs of strings by considering their *subsequences.*

5. Case of the Sauron (cont-d). In the last home assignment you had to analyze the situation with Sauron and his two students. The true answer was unintuitive for many. In this exercise you will simulate the Sauron's case and see how it goes for yourself.

Let us suppose that the second student got *really lucky* and guessed a proper representation for the spells. That is, indeed, each of Sauron's 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 the *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 the situation 1000 times or more, and observe:

- How often would the first student guess 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?
- Could you somehow improve the second student's performance by, say, doing training in a smarter way? Tuning method parameters? Changing the number of attributes used in training?
- Finally, see what happens when the dataset becomes larger (i.e. increase n to 30, 40, ..., 100, 200, ...). How many training points are required so that the second student's expected generalization error would go down to 10%? To 5%? What happens with the first student's expected generalization error?

NB: A sample script is available for your convenience at the assignment webpage. Note that in order to train SVM classifiers you will need to install the `e1071` *R package. The command that installs it, is given in the first line of that script (yet commented out).*

6*. Machine learning for music. I've downloaded 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` for you, and invite you to 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 (which should be a good application for the string kernel, by the way). 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.