MTAT.03.183 Data Mining

Machine Learning, Part II. Home Assignment.

Konstantin Tretyakov

Due: November 19, 2009

1. Learning from data. After the lecture on supervised machine learning at the University of Mordor, the teacher, Sauron, gave the students a dataset of the following form:

```
 1) ABACDAFXYZ    -> good
 2) CEGXEAELWMN   -> good
 3) NUWAB         -> bad
         ...
20) QRELAZMNPCXA  -> good
```

The inputs were seemingly arbitrary strings of latin characters: these were the terrible spells known only to Sauron and chosen by Sauron at random from his Great Book of Terrible Spells. The output was a binary variable, classifying each spell as good or bad. There were 20 observations in total in the dataset.

The students were given a task: on the basis of this data, come up with a generic spell classifier. Sauron promised to test their result on the next lecture as follows: he will pick another random terrible spell from the book and require each student to make a prediction. The student whose prediction is wrong will have to jump down from the tower as a punishment.

The first student, Aghargh, who happened to be solving his previous week's homework instead of listening to the lecture, couldn't come up with any smart ways to solve the task, so he ended up just counting the proportion of "good" and "bad" spells in the dataset. Having observed that 16 of the 20 spells were "good", he decided to predict "good" when asked.

The second student, Bughrorc, chose a smarter way. Firstly, he converted each string into a vector of binary attributes – one attribute for each letter, having value "1" if that letter was present in the corresponding spell and "0" otherwise. He then split the data randomly into a training set (15 instances) and a test set (5 instances), and attempted training various classifiers using the MordorMiner software. After some experimenting he finally found five classifiers that could predict all of the training examples correctly. One of these also predicted all of the testing examples correctly. He decided to use this classifier on the forthcoming test.

On the day of testing, Sauron asked the students to classify the spell `YOZAZA`. Aghargh, as he decided, provided the answer "good". Bughrorc's classifier analyzed the string and claimed that the answer should be "bad".

Which of the students, do you think, might have a better chance of not jumping from the tower? Why? Can you quantify your answer?

2. Naïve Bayes classifier. Consider the following dataset:

Sunny	**Windy**	**PlayTennis**
Yes	No	Yes
Yes	No	Yes
Yes	No	Yes
No	Yes	Yes
No	Yes	Yes
Yes	Yes	No
Yes	Yes	No
No	No	No
No	No	No
No	No	No

It is *windy* and *not sunny* today. What prediction regarding playing tennis would you obtain from a Bayes classifier? From a Naïve Bayes classifier? Which prediction is correct? Why? What properties of the Naïve Bayes classifier are involved?

3. Linear regression. In the lecture we studied the task of fitting a function $f_w(x) = wx$ to a set of data points $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ by looking for the *least-squares solution*

$$\hat{w} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (f_w(x_i) - y_i)^2 \,.$$

Consider now a linear function with two parameters

$$f_{w,b}(x) = wx + b\,,$$

and derive a least-squares fit for both w and b:

$$(\hat{w}, \hat{b}) = \underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} (f_{w,b}(x_i) - y_i)^2 \,.$$

Report just the obtained expressions for $\hat{w}$ and $\hat{b}$. No need to report the derivation (but be prepared to reproduce it).

4. K nearest neighbors. Suppose you are given a task of classifying texts (e.g. sorting e-mail as spam or not). Is it a good idea to apply the K-nearest neighbors algorithm? How could you apply it? What if your training set is *very* large, how would you solve the algorithm performance problems?

Be brief. No more than two-three short paragraphs total.

5. Instance-based methods. Consider a two-class classification problem. Let $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}, y_i \in \{-1, 1\}$ be the training set.

1. Assuming that you use kernel smoothing to estimate probability densities, how would the Bayes classifier look like, when trained on this data. Show that the resulting function can be expressed as

$$\text{BayesClassifier}(x) = \text{sign}\left(\sum_{i=1}^{n} y_i a_i K(x, x_i)\right) .$$

   What is a_i equal to? (Just report this single answer, do not bother writing up the whole derivation, yet be prepared to explain).

   Note that this classifier is sometimes referred to as the *Parzen classifier.*

2. It turns out that the K-nearest neighbor classifier can also in principle be represented as

$$\text{KNN}(x) = \text{sign}\left(\sum_{i=1}^{n} y_i a_i K_D(x, x_i)\right) ,$$

   where $a_i = 1$ for all i. What is $K_D(x, x_i)$ here? Hint:

$$K_D(x, x_i) = \begin{cases} 1, & \text{if } x_i \text{ is } \dots, \\ 0, & \text{otherwise} \end{cases} .$$

3. Suppose you are willing to apply the linear regression function $f(x) = wx$ to the classification task. Of course, it does not make any sense unless x is a vector, but for simplicity let us ignore this issue. In this case, your classifier will look as

$$\text{LinearClassifier}(x) = \text{sign}(wx) .$$

   There is a popular (and very simple) algorithm for learning linear classifiers (i.e. finding the proper w) – the *perceptron.* Due to the specifics of the algorithm, the resulting w is always of the form $w = \sum_{i=1}^{n} y_i a_i x_i$ for some $a_i \geq 0$. Show that in this case

$$\text{LinearClassifier}(x) = \text{sign}\left(\sum_{i=1}^{n} y_i a_i K(x, x_i)\right) .$$

   What is $K(x, x_i)$ here?

4. Finally, just for your reference. A *support vector machine* classifier is also a function of the form:

$$\text{SVM}(x) = \text{sign}\left(\sum_{i=1}^{n} y_i a_i K(x, x_i)\right) .$$

Also the *radial basis function* is a regression function of the form:

$$\mathrm{RBF}(x) = \sum_{i=1}^{n} y_i a_i K(x, x_i)\,.$$

See the pattern?

6*. Spam classification. Try building a classifier for discriminating spam in *your* mailbox. How good is your classifier? What words contribute mostly to its decisions?

You are free to approach this task in any way you deem appropriate, but here are some hints to help some of you get started.

1. If you use the @ut.ee e-mail, you will easily find all your emails in your home directory. They will most probably either be in a single `mbox` file or in a number of files under the `Mail` subdirectory (the `Trash` folder will then be a good place to look for data as it might contain a fair amount of both spam and non-spam emails).

   If you use some other mail service, find out a way of exporting all the messages or just cooperate with a friend who has easier access to data.

2. To get started you might first try to classify messages just based on the `Subject` header. Exporting all the `Subject` headers from all the messages in your `mbox` file can be done for example as follows:

```
> cat ~/mbox | grep '^Subject:' > my_emails.txt
```

3. The university server contains a spam filter installation that automatically appends the word `"**Spam**"` to the subjects of suspicious emails. This can be easily extracted from the subject and used as a label:

```
> cat ~/mbox | grep '^Subject:' | grep '**Spam**' > spam_emails.txt
> cat ~/mbox | grep '^Subject:' | grep -v '**Spam**' > nonspam_emails.txt
```

   If you use a different spam filter, you might have to look for a certain field in the message header (typically *X-Spam-Status*).

4. If you are not interested in implementing a classifier yourself, you can use Weka. For that you will need to convert your data to the *arff* format. A python script like that might be helpful:

```
m = open("my_emails.txt").readlines()
v = map(lambda x: '**Spam**:' in x, m)
m = map(lambda x: x.replace('**Spam**:','').replace("'",'').strip(), m)
f = open("my_mails.arff", "w")
print >>f, "@relation 'mailbox'"
print >>f, "@attribute text string"
print >>f, "@attribute class {0,1}"
print >>f, "@data"
for (s,c) in zip(m,v):
    print >>f, "'%s',%d" % (s,c)
f.close()
```

5. After loading the file into Weka you will need to convert the string to a set of binary variables, one for each word. This can be done using the *StringToWordVector* filter.