MTAT.03.227 Machine Learning (Spring 2014)
Exercise session XV: Kernel Methods

Konstantin Tretyakov

May 27, 2014

The aim of this exercise session is to get acquainted with the idea of kernel methods. As usual, for all exercises you need to write a brief explanation and, for most of them, also a short piece of code demonstrating the result. You can submit your whole solution as a single *decently commented* R or Rmd file, provided it is sequentially readable and/or executable.

We shall use the `kernlab` R package. Install it using `install.packages`. *kernlab*

```
install.packages("kernlab")
library(kernlab)
```

We shall be working with the collection of Reuters articles from 1987 – one of the famous benchmark datasets in the text mining field. Load it[1], using the `load.data` function in the provided base code `kernels_base.R`. *Reuters dataset*

```
reuters = load.data()
```

The resulting data frame contains 500 news items on two topics – *crude oil* and *grain*[2]. The news item text is in the column `Content` and its category is in the column `Topic`. In addition, column `y` is equal to $+1$ whenever `Topic == "grain"` and -1 otherwise. Let us split the dataset into training and test folds of size 300 and 200:

```
reuters_train = reuters[1:300,]
reuters_test = reuters[301:500,]
```

Our first goal will be to train and evaluate a classifier for the two types of articles.

String Kernels

To deal with textual data we need to use a *string kernel*. Several such kernels are implemented in the `stringdot` method of the `kernlab` package. We shall *String kernel*

[1] The invocation of `load.data` downloads the data, so you need to be online. The data file is 7.4M, so it might take a while.

[2] I actually provide you with a larger dataset to play with, if you wish. The data is getting trimmed to just two topics and 500 items in the `load.data` function

use the simplest one – the *p-spectrum* kernel. The feature mapping for this kernel represents the string as a multiset of its substrings of length p. E.g. for $p = 2$ we have

$$\phi(\text{“ababc”}) \to \{\text{“ab”} \to 2, \text{“ba”} \to 1, \text{“bc”} \to 1, \text{other} \to 0\}.$$

Using the `kernlab` package, a p-spectrum kernel is created as follows:

```
k = stringdot("spectrum", length=2, normalized=F)
```

It can then be applied to pairs of strings:

```
k("first string", "second string")
```

We can also compute the complete *kernel matrix* for a list of strings:

```
K = kernelMatrix(k, list_of_strings)
```

p-spectrum

Exercise 1* (1pt). Study the implementation of the p-spectrum kernel in `kernlab`. Choose a small p (e.g. 2) and try applying k to strings like ‘aa’, ‘aaa’, etc. Does the result correspond to your expectations? If not, guess the reasons for the observed differences and explain.

For most text classification tasks a *normalized* kernel performs better. A normalized version k_{norm} of any kernel k can be obtained as

$$k_{\text{norm}}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}},$$

but using `kernlab` it is sufficient to specify[3] `normalized=T` in `stringdot`.

Kernel matrices

Exercise 2 (0.5pt). In the following, we shall need two matrices:

- The 300×300 *training* kernel matrix $\mathbf{K}$, such that $\mathbf{K}_{(i,j)} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are i-th and j-th elements of `reuters_train`.
- The 300×200 *train-vs-test* kernel matrix $\mathbf{K}_{\text{test}}$ such that $\mathbf{K}_{\text{test}(i,j)} = k(\mathbf{x}_i, \mathbf{x}'_j)$, where $\mathbf{x}'_j$ is the j-th element of `reuters_test`.

Use `stringdot` and `kernelMatrix` to compute those two matrices. Use a normalized 5-spectrum string kernel. Name the corresponding variables `K` and `K_test`.

Kernel Classification

A kernel classifier is a function of the form:

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$$

where α_i is the *dual representation* of the classifier normal $\mathbf{w}$ in the (possibly high-dimensional) feature space.

[3] In fact, it is the default option.

Exercise 3 (0.5pt). Suppose you have trained an SVM classifier and obtained the SVM parameters α'_i from the SVM dual optimization. Are those α'_i also the dual representation of $\mathbf{w}$ as defined in the paragraph above? *Dual confusion*

Exercise 4 (2pt). Suppose we trained a kernel classifier on our `reuters_train` data. Suppose $b = 0$, $\alpha_1 = 1, \alpha_2 = -1$ and all the remaining α_i values are 0. *Kernel classifier*

1. Evaluate the prediction of such classifier on two strings: "eat more corn" and "petroleum".
2. Evaluate the prediction of this classifier on all training examples and compute its prediction accuracy on the training set. Hint: make use of the matrix $\mathbf{K}$ you computed recently, don't recompute the kernel values!
3. Evaluate the prediction of this classifier on all test examples and compute its prediction accuracy on the test set. Hint: make use of the matrix $\mathbf{K}_{\text{test}}$ you computed recently.
4. In general, suppose you have a "training" kernel $\mathbf{K}$ and a kernel-based classifier with parameters $(\boldsymbol{\alpha}, b)$, that was trained on this data. Write a matrix expression which computes the predictions of the classifier for *all* training examples as a single vector.
5. Similarly, write a single matrix expression, which computes predictions of the classifier $(\boldsymbol{\alpha}, b)$ for all test examples as a single vector.

Exercise 5 (1pt). The base code `kernels_base.R` provides you with a bare implementation of the kernelized perceptron algorithm. Some statements are missing, however. Fix the implementation to have a working algorithm. Then train the algorithm on the training set and evaluate its performance on the test set. *Kernel perceptron*

Exercise 6 (1pt). The `kernlab` package has its own implementation of most popular kernel methods, and SVM in particular. *`ksvm`*

1. Use the `ksvm` function with the kernel matrix $\mathbf{K}$ to train an SVM classifier using the chosen kernel.
2. Evaluate the performance of the resulting algorithm on the test set. Hint: chances are high you will have troubles if you try to use the `predict` method to apply the model on the test set. Instead just take the raw alpha values (`model@alpha`, `model@alphaindex`) and compute the model predictions manually.

Exercise 7 (1pt). An important issue in kernel methods is overfitting. Indeed, when the feature space is infinite-dimensional, it is easy to achieve perfect performance on *any* training set which often leads to severe overfitting. This is best seen on a kernelized formulation of ordinary least squares regression. *Regularization*

Derive the kernelized version of ordinary least squares and show that if the kernel is positive definite, it is indeed possible to classify any training set perfectly. An important consequence of this is that you should generally avoid using kernel methods without regularization[4]

Hint. The coefficient vector $\mathbf{w}$ for the linear regression problem is the least-squares solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$. Rewrite this condition in dual form by replacing $\mathbf{X}$ with the higher-dimensional data matrix $\boldsymbol{\Xi}$ and substituting the dual representation $\mathbf{w} = \boldsymbol{\Xi}^T\boldsymbol{\alpha}$ for the weight vector. The positive definiteness of the kernel matrix is equivalent to its invertibility.

Kernel PCA

Principal Components Analysis (PCA) is a method for finding linear "components" in the data. It typically proceeds as follows:

1. First, center[5] the data matrix $\mathbf{X}$.
2. Solve $\mathbf{X}^T\mathbf{X}\mathbf{v} = \lambda\mathbf{v}$ for $\mathbf{v}$ (by finding the eigenvalues of $\mathbf{X}^T\mathbf{X}$).
3. Project the data onto the largest eigenvalue(s): $\mathbf{p} = \mathbf{X}\mathbf{v}$.

As all linear methods, PCA may be performed in the higher-dimensional feature space using only the dual representation. This allows to extract non-obvious non-linear patterns from the data as well provide nice visualizations.

To kernelize PCA[6] we denote the hypothetical high-dimensional data matrix $\phi(\mathbf{X})$ by $\boldsymbol{\Xi}$. We shall be looking for linear component $\mathbf{v}$ in that space by solving

$$\boldsymbol{\Xi}^T\boldsymbol{\Xi}\mathbf{v} = \lambda\mathbf{v}.$$

However, instead of working directly with $\mathbf{v}$ we substitute its dual representation $\mathbf{v} = \boldsymbol{\Xi}^T\boldsymbol{\alpha}$. Then,

$$\begin{aligned}
\boldsymbol{\Xi}^T\boldsymbol{\Xi}\mathbf{v} &= \lambda\mathbf{v},\\
\boldsymbol{\Xi}^T\boldsymbol{\Xi}\boldsymbol{\Xi}^T\boldsymbol{\alpha} &= \lambda\boldsymbol{\Xi}^T\boldsymbol{\alpha},\\
\boldsymbol{\Xi}\boldsymbol{\Xi}^T\boldsymbol{\Xi}\boldsymbol{\Xi}^T\boldsymbol{\alpha} &= \lambda\boldsymbol{\Xi}\boldsymbol{\Xi}^T\boldsymbol{\alpha},\\
\mathbf{K}\mathbf{K}\boldsymbol{\alpha} &= \lambda\mathbf{K}\boldsymbol{\alpha},\\
\mathbf{K}\boldsymbol{\alpha} &= \lambda\boldsymbol{\alpha}.
\end{aligned}$$

[4] However, as we have just seen in the perceptron example, they can sometimes work well none the less despite the lack of regularization. It is also customary to close eyes on the regularization issue when performing unsupervised learning, as we do in the following section.

[5] By the way, PCA often works fine even without centering.

[6] This is not the most formally rigorous derivation, but it's correct none the less.

The last operation is obviously legal if $\mathbf{K}$ is positive definite (and thus invertible). Using some careful math it is possible to show that it works out for positive semidefinite $\mathbf{K}$ as well.

Consequently, dual representations of pricipal components are simply the eigenvectors of the kernel matrix.

The projection of the training data to the corresponding eigenvalue is:

$$\mathbf{p} = \Xi\mathbf{v} = \Xi\Xi^T\boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}.$$

Thus, the Kernel PCA algorithm is:

1. Center the kernel $\mathbf{K}$.
2. Solve $\mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$ (by finding the eigenvalues of $\mathbf{K}$).
3. Project the training data onto the largest eigenvalue(s): $\mathbf{p} = \lambda\boldsymbol{\alpha}$.

Exercise 8 (2pt). Implement Kernel PCA on the Reuters data and visualize the two largest principal components. *Kernel PCA*

Hints.

1. Kernel centering:
$$\mathbf{K}_{\text{cent}} = \mathbf{K} - \frac{1}{n}\mathbf{1}\mathbf{1}^T\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1}\mathbf{1}^T + \frac{1}{n^2}\mathbf{1}\mathbf{1}^T\mathbf{K}\mathbf{1}\mathbf{1}^T.$$
2. Eigenvalue decomposition:
```
e = eigen(K) # The largest eigenvector is in e$vectors[,1]
```
3. For the purposes of visualization you can ignore the multiplication by λ.
4. For a nicer picture, don't forget to color the points according to their class:
```
plot(<pc1>, <pc2>, col=reuters_train$Topic)
```

Exercise 9 (1pt). Naturally, Kernel PCA is implemented in `kernlab`. Use the `kpca` function and visualize the two largest principal components. Do you get the same picture? *`kpca`*

Exercise 10 (1pt). We have just seen the use of Kernel PCA to visualize non-numeric data, but this is not the only exciting feature of KPCA. More importantly, KPCA is capable of discerning *nonlinear* components. To see that, load the dataset produced by the `circle.data` function in the base code: *Nonlinear PCA*

```
data = circle.data()
plot(data$x, data$y, col=data$c)
```

Then apply KPCA to this dataset (use the default `rbfdot` kernel) and visualize the transformed points.

Exercise 11* (4pt). Derive and implement the kernelized version of the *K-means* algorithm. Run it on the Reuters dataset. Can the clustering separate the two topics? Compare the results to the implementation in `kernlab`.

Kernel K-means