Kernel Methods for Pattern Analysis: Project

John Shawe-Taylor Tijl De Bie

October 21, 2004

Abstract

The project should take about 30 hours. It should be handed in by email to Eerika Savia (`Eerika.Savia@hut.fi`) at the latest on Sunday November 28th.

By now, you probably have a good idea of the power of kernel methods. On the one hand, a large choice of fast pattern analysis algorithms can be used for a variety of tasks going from classification over novelty detection to regression. On the other hand a wide range of data types can be dealt with using kernel functions that are specialized for the data and the problem at hand. Furthermore, subspace methods such as PCA and CCA allow to select interesting features from the data. This modular approach to pattern analysis is one of the most crucial aspects of kernel methods.

In this project, you will work out a case study during which you will be able to fully appreciate this modularity. Basically all the algorithms and one of the two kernels to be used in this project have already been implemented in the practical sessions. Reuse these where possible...

1 The data

The data you will work with are the articles of the Finnish constitution, available both in Finnish and in Swedish on the website `http://www.om.fi/21910.htm`. Extract the different articles from the constitution (you can do this in matlab or in perl or by hand in a text editor,...whatever you like), and load them into matlab (or into R if you prefer). In matlab, it's probably most convenient to store the articles in a cell array.

2 The tasks to solve

There are two main tasks in the project:

- **Cross-language retrieval of articles.** Using a subset of articles in both languages that are translations of each other (the training set), train a query system that after training is capable to select (from a corpus of Swedish texts) a Swedish text that is relevant to (or semantically related with) a given Finnish query text, that is not necessarily an exact translation.

- **Classification of articles according to the chapter they belong to.** Since there are 13 different chapters in the constitution, this will be a so-called multi-class classification problem (in fact we will only use 9 of these chapters here). There are several ways to do this as will be explained further down in this text.

3 The kernels

There exist various kernels designed for strings and text, a few of which were discussed and used during the classes and practical sessions. In this project you will use two of them: the bag of words kernel and the 3-mer kernel.

Keep in mind that in order to use a bag of words kernel, actually a few preprocessing steps have to be performed first: usually the words are *stemmed* (which means that e.g. English words like ‘go’, ‘gone’ and ‘went’ all would be considered to be the same), and stop words are removed. You are free to do this, however it is not required for the project, you might even wonder if it’s appropriate to do this in a language like Finnish. For both the bag of words kernel and the 3-mer kernel the case of the letters should be ignored (this is a requirement of the project). Also make sure you remove numbers and punctuation before applying the kernels.

Implement these kernels, and run them on both the Finnish and Swedish version of the constitution.

To report 1 *Explain which additional preprocessing operations you performed on the kernel matrix, if any, and why. Plot of the resulting kernels evaluated between all Finnish and Swedish documents (in matlab using the* `imagesc` *function).*
Additionally, hand in the code for the 3-mer kernel.

4 The algorithms

4.1 Cross-language article retrieval

For the cross-language retrieval of articles, randomly select a training set of 80% of the Finnish articles, and the corresponding Swedish articles. Since we want to build a system that responds with the corresponding Swedish article given a certain Finnish article, we would like to get a representation of each of these articles that does not depend on the language or vocabulary, but that rather represents the meaning of the articles.

In order to extract such a representation, you can perform kCCA between both languages. Then each kCCA component corresponds to a feature capturing part of the semantics of the texts. Therefore, for Finnish and Swedish texts that are translations of each other, the features corresponding to dominant canonical components will be likely to be similar (since their meaning should be close to identical).

We will thus proceed as follows.

1. The training phase: perform kCCA on the training set; select the k dominant canonical components. Normalize the dual vectors α_x and α_y such that, with $\mathbf{K}_x$ and $\mathbf{K}_y$ the matrices corresponding to the training sample, $\alpha_x'\mathbf{K}_x^2\alpha_x = 1$ and $\alpha_y'\mathbf{K}_y^2\alpha_y = 1$. The projections of a text on these components are the (hopefully) language independent features we will use.

2. The query phase:

   a For each article in the test set, compute its projection on these canonical component directions. Do this for both languages on their respective canonical components. This is the semantic feature representation of the test set articles.

   b Thus, when one of these Finnish articles (from the test set) is queried, you can look up its semantic feature representation. Then, among all Swedish test set articles, look for that article whose semantic feature representation is closest to that of the Finnish article (you can use the Euclidian distance for this). This will then be outputted as the translated version.

In order to assess the performance, compute how often the translated version was found as the best match. Additionally, compute how often the translated text was within the top 5 of best matches.

Note there are two parameters to be tuned in the training phase: the number of canonical components (i.e. the number of features in the semantic feature representation), and the regularization parameter of the kCCA algorithm. Both can be trained using cross-validation.

To report 2 *Make a (small, say, 5 by 5) table of cross-validation errors for a range of values for the regularization parameter and the number of kCCA components. Mention which parameter setting leads to the smallest cross-validation error, and what the corresponding test set error is.*
Additionally, hand in the code for the CCA algorithm.

4.2 Classification of the articles

Here we will only use the *Finnish version* of the constitution, and only the *bag of words kernel*. Do *not* center the kernel, but *do* normalize it. (This is important because the kernel adatron algorithm is only guaranteed to converge when the diagonal elements of the kernel matrix are not too different from inverse of the coordinate ascent step size η. You can take $\eta = 1$ here.)

As you can see, the articles are organized into chapters. The goal of this exercise is to make a classifier that classifies the articles into the right chapter. Because each of the classes has to be reasonably large for a machine learning algorithm to work, you can omit the 4 smallest chapters from the data set for this part of the project. Thus, 9 chapters remain: chapters 2, 3, 4, 5, 6, 7, 9, 10, 11 and 12. Now, randomly select a *stratified* training set containing roughly 80% of the articles. 'Stratified' means that *for each class*, the test sample contains (up to a rounding error) 20% of the samples of that class.

In the lectures, you only learned how to do binary classification, using an SVM, kFDA, the kernel adatron,... Luckily, a 9-class (or in general multi-class) classification problem can be reduced to a set of binary classification problems in the following ways (there exist other approaches too):

- Perform 9 binary classification problems, each of which learns to distinguish one of the 9 classes (denoted as the positive class) against the other 8 (together named the negative class).
  To estimate the class of a test article, compute the prediction of each of the 9 binary classifiers. The one that returns the largest value tells you in which class to classify the test sample at hand. This type of multi-class classification is called *one-versus-all (OvA).*

- Perform $\frac{9 \cdot 8}{2} = 36$ binary classification problems, each of them classifying one of the 9 classes against one of the remaining 8.
  To estimate the class for a test sample, evaluate each of the 36 classifiers. In this way, for each of the 9 classes you will get 8 scores. The class getting the largest sum of scores for the test sample is the class it will be assigned to. This type of multi-class classification is referred to as *all-versus-all (AvA).*

Implement both the OvA and the AvA schemes using the kernel adatron. To tune the regularization parameter there are two options. The first approach would be to tune it optimally within each binary classification problem. However, some of the chapters are so small that cross-validation is difficult to do: in some of the 'folds', the small class may well be empty in the validation set. Therefore, you can choose for the other option in this case: use the *same* regularization parameter for all binary classification problems, and tune it in such a way that the cross-validation error on the final multi-class classification problem is minimal. In this exercise, you could use the so-called leave-one-out cross-validation, which is essentially n-fold cross-validation, where n is the size of the training set. (If your computer is too slow, use 10-fold cross-validation.)

(Note: also in the multi-class case, the error is usually defined as the fraction of misclassified points. Note that since we are having 9 classes, everything better than 8/9 classification error is in some sense better than random...)

To report 3 *Show a small table containing the cross-validation error for a range of values for the regularization parameter, and this both for the OvA and the AvA schemes. Indicate which value is optimal, and compute the corresponding test set error. Also compute the test set error corresponding to other*

regularization parameter settings, and put it in a table. This allows you to check in how far the cross-validation succeeded in finding the optimal value.
Additionally, hand in your code for the multi-class classification, and the code for the kernel adatron.