Computationally Intelligent Hybrid Systems: The Fusion of Soft Computing and Hard Computing, Wiley, New York, 2005, pp. 317-365

Chapter 10: Introduction to Scientific Data Mining: Direct Kernel Methods & Applications

Mark J. Embrechts
Rensselaer Polytechnic Institute, Troy, New York, USA

Boleslaw Szymanski
Rensselaer Polytechnic Institute, Troy, New York, USA

Karsten Sternickel
Cardiomag Imaging Inc., Schenectady, New York, USA

10.1 INTRODUCTION

The purpose of this chapter is to give a brief overview of data mining and to introduce direct kernel methods as a general-purpose and powerful data mining tool for predictive modeling, feature selection and visualization. Direct kernel methods are a generalized methodology to convert linear modeling tools into nonlinear regression models by applying the kernel transformation as a data pre-processing step. We will illustrate direct kernel methods for ridge regression and the self-organizing map and apply these methods to some challenging scientific data mining problems. Direct kernel methods are introduced in this chapter because they transpire the powerful nonlinear modeling power of support vector machines in a straightforward manner to more traditional regression and classification algorithms. An additional advantage of direct kernel methods is that only linear algebra is required.

Direct kernel methods will be introduced as a true fusion of soft and hard computing. We will present such direct kernel methods as simple multi-layered neural networks, where the weights can actually be determined based on linear algebra, rather than the typical heuristic neural network approach. Direct kernel methods are inherently nonlinear methods, and can be represented as a multi-layered neural network that now combines elements of soft and hard computing. The hard computing takes place in the scientific domain where data are generated, in a way that often involves elaborate (hard) computing algorithms. Hard computing is also used here to make up the kernel and to calculate the weights for the underlying neural networks in direct kernel methods. Soft computing occurs because of the underlying neural network framework and in estimating the hyper-parameters for direct kernel models. These hyper-parameters usually deal with the proper choice for the nonlinear kernel, and the selection of a close to optimal regularization penalty term.

Support Vector Machines or SVMs have proven to be formidable machine learning tools because of their efficiency, model flexibility, predictive power, and theoretical transparency [1-3]. While the nonlinear properties of SVMs can be exclusively attributed to the kernel transformation, other methods such as self-organizing maps or SOMs [4] are inherently nonlinear because they incorporate various neighborhood-based manipulations. This way of accounting for nonlinearity effects is similar to the way how K-nearest neighbor algorithms incorporate nonlinearity. Unlike SVMs, the prime use for SOMs is often as a visualization tool [5] for revealing the underlying similarity/cluster structure of high-dimensional data on a two-dimensional map, rather than for regression or classification predictions. SOMs have the additional advantage that they incorporate class ordinality in a rather natural way, i.e., via their self-organization properties that preserve the topology of a high-dimensional space in the two-dimensional SOM. SOMs are therefore quite powerful for multi-class classification, especially when the classes are not ordinal: a problem that is far from trivial. They are also very effective for outlier and novelty detection.

Before explaining direct kernel methods, we will present a brief overview of scientific data mining. The standard data mining problem will be introduced as the underlying framework for different data mining tasks. We will then build a simple linear regression model, explain the data mining and machine learning dilemmas, and provide a simple solution to overcome this type of uncertainty principle. These linear methods will then be translated into an equivalent, but still linear, neural network model, for which the weights can be obtained with hard computing. The linear regression model or predictive data mining model can be transformed into powerful nonlinear prediction method by applying the kernel transformation as a data transformation rather than an inherent ingredient in the mathematical derivation of the modeling algorithm. Many traditional linear regression models can be elegantly transformed into nonlinear direct kernel methods that share many desirable characteristics with support vector machines: they can incorporate regularization and they do not involve the controversial heuristics, common in the neural network approach. We will finally apply this methodology to a challenging scientific data mining problem and illustrate predictive modeling, feature selection and data visualization based on direct kernel methods for predicting ischemia from magneto-cardiogram data.

10.2 WHAT IS DATA MINING?

10.2.1 Introduction to Data Mining

Data mining is often defined as the automated extraction of novel and interesting information from large data sets. Data mining, as we currently know it, has its roots in statistics, probability theory, neural networks, and the experts systems angle of artificial intelligence (AI). The term data mining used to have a negative co-notation, meaning the existence of spurious correlations, indicating that if one looks far enough in a variety of data sets one might find a coincidental rise in the stock market when there is a peak of two-headed sheep born in New Zealand. This out-of-date interpretation of data mining can be summarized as “the torturing the data until they confess approach.” The current popularity of the term data mining can be attributed largely to the rise of the Knowledge Discovery and Data Mining (or KDD) Conference. The KDD conference started in the early nineties as a small workshop, spearheaded by Usuama Fayyad, Gregory Pietatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy. The KDD conference is now an annual event and has a good attendance. In the book that resulted from the 1995 KDD conference in Montreal [6], data mining was defined as: “Data mining is the process of automatically extracting valid, novel, potentially useful, and ultimately comprehensible information from large databases.” We will adhere to this definition to introduce data mining in this chapter. Recommended books on data mining are summarized in [7-10]. One of the underlying principles of knowledge discovery in data is to promote the process of building data-driven expert systems as an extension of the more traditional AI expert systems approach. The idea is now that experts learn from new findings in the data as well.

Data mining is not a narrowly focused discipline, but requires a combination of multiple disciplines and techniques. Data mining distinguishes itself from traditional statistics in the sense that we now deal with potentially very large datasets that can range from gigabytes, terabytes, to pentabytes. For a while, a problem was considered a data mining problem only if the data could not be stored in the working memory of a computer all-at-once. Other definitions of data mining insisted for a while that the data has to come from a variety of different databases. Of course, interesting and extremely challenging problems such as gene discovery and protein folding in bio-informatics, would not qualify as legitimate data mining problems under these restrictive definitions. Data mining is different from the more traditional methods in the sense that for large amounts of data, many classical algorithms, such as the K-means algorithm for clustering, do not scale well with ever-larger datasets. In general, one can summarize that for a typical data mining case: (i) the data set can be quite large; (ii) the problem is generally challenging and is often not well defined; (iii) there are missing and faulty data; and, (iv) there are redundancies in the data fields, but the redundant fields do not all have the same quality.

Data mining distinguishes itself also from statistics and artificial intelligence in the sense that the expert now exercises a different role. While the goal of AI expert systems was to query the experts in order to come up with a rule base that captures their expertise, that approach often led to failure because the experts, even though knowledgeable and mostly right, are not necessarily in the best position to formulate an explicit set of rules. In data mining, rather than letting the expert formulate the rules up front, the idea is now to let the rules appear in a more or less automated and data-driven way. The expert comes in at the end stage of this data-driven rule discovery/formulation process and applies his domain knowledge to validate the rules.

The first very successful data mining applications were often driven by database marketing and business applications. Typical applications of database marketing are the use of a database to decide on a mail-order campaign, or linking a sales campaign in a supermarket with product positioning and discounting. A classical case here is has been observed that beer sales go up when the beer is positioned close to the diapers in a supermarket store, because dad is more likely to puck up a 6-pack of beer when he is sent to the story to by diapers in case of an emergency. The tongue-in-cheek corollary is here that the reverse is not true. Other early successful applications of data mining relate to credit card fraud, establishing lending and refinancing policies, and telephone fraud.

Data mining is an interdisciplinary science ranging from the domain area and statistics to information processing, database systems, machine learning, artificial intelligence and soft computing. The emphasis in data mining is not just building predictive models or good classifiers for out-of-sample real world data, but obtaining a novel or deeper understanding. In real world problems, data distributions are usually not Gaussian. There also tend to be outliers and missing data. Often there are faulty and imprecise data to deal with as well. Data mining emphasizes the use of innovative and effective data visualization techniques, such as self-organizing maps [4], that can go way beyond the common bar and pie charts. The exact purpose and outcome of a data mining study should probably not be clearly defined up front. The idea of data mining is to look at data in a different way, and in a sense, to let the data speak for themselves.

10.2.2 Scientific Data Mining

Scientific data mining is defined as data mining applied to scientific problems, rather than database marketing, finance, or business-driven applications. Scientific data mining distinguishes itself in the sense that the nature of the datasets is often very different from traditional market-driven data mining applications. The datasets now might involve vast amounts of precise and continuous data, and accounting for underlying system nonlinearities can be extremely challenging from a machine learning point of view.

Applications of data mining to astronomy-based data is a clear example of the case where datasets are vast, and dealing with such vast amounts of data now poses a challenge on it's own. On the other hand, for bio-informatics related applications such as gene finding and protein folding, the datasets are more modest, but the modeling part can be extremely challenging. Scientific data mining might involve just building an on-the-fly predictive model that mimics a large computer program that is too slow to be used in real time. Other interesting examples of scientific data mining can be found in bioengineering, and might present themselves as challenging pattern recognition problems based on images (e.g., brain scans) or (multi-variate) time series signals (e.g., electrocardiograms and magneto-cardiograms).

An interesting application relates to in-silico drug design [11]. The idea is to identify and select small molecules (ligands) with superior drug qualities from a huge library of potential often not yet synthesized molecules. The challenge is that the library of molecules with known pharmaceutical properties is often relatively small (~50 - 2000), but that there is a large number of descriptive features or attributes (e.g., 500 - 20000). We define such problems where the number of descriptive features exceeds the number of data by far, as data strip mining problems [12]. We call them data strip mining problems, because if the data are placed in an Excel sheet all the data seem to be now on the surface rather than going on and on for thousands and thousands of rows of cells. There is one additional interesting aspect here: many computer programs such as editors and the Excel spreadsheet were not design to handle this type of data. A key challenge for in-silico drug design problems is now to identify a relatively small subset of relevant features that explain the pharmaceutical properties of the molecule. One ultimate aim of in-silico drug design is real-time invention and synthesis of novel drugs to mitigate natural or man-made society threatening diseases. A second type of strip mining problems occurs in the use of gene expression micro-arrays for the identification of relevant genes that are indicative for the presence of a disease. A typical case is a dataset of 72 micro-array data with 6000 descriptive features related to the identification of leukemia.

In a data mining context, common techniques such as clustering might now be used in a very different way. The clustering does not necessarily have to provide a good overall clustering, but just finding one relatively small and fairly homogeneous cluster might offer a significant pay-off in database marketing. Kohonen's self-organizing map has been extensively applied as an efficient visualization tool for high-dimensional data on a two-dimensional map while preserving important aspects of the underlying topology.

10.2.3 The Data Mining Process

Many data mining applications can be represented in a cartoon model that we will call the standard data mining process. This process involves the gathering of data, data cleansing, data pre-processing and transforming a subset of data to a flat file, building one or more models that can be predictive models, clusters or data visualizations that lead to the formulation of rules, and finally piecing together the larger picture. This process is outlined in Fig. 10.1.

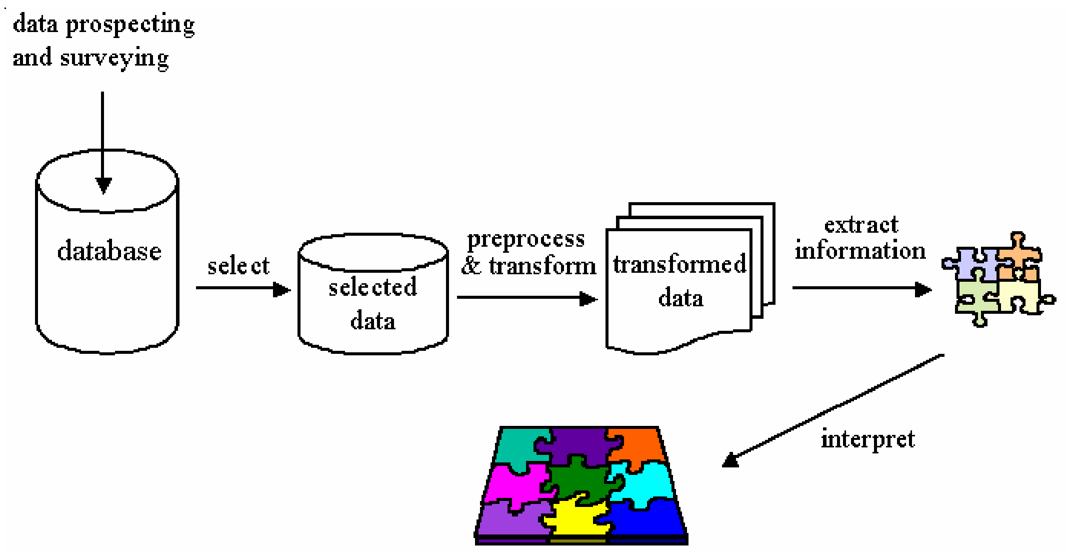


Figure 10.1 Cartoon Illustration of the data mining process.

It is interesting to note here that often a large amount of effort is required before the data can be presented in a flat file. Data cleansing and data pre-processing often takes up a large part of the resources committed to a typical data mining project and might involve 80 percent of the effort. It is often necessary to experiment with different data transformations (e.g., Fourier and wavelet transforms) in the data pre-processing stage.

Another representation of the data mining process is the data mining wisdom pyramid in Fig. 10.2, where we progress from raw data to information, knowledge and understanding, and ultimately wisdom. The art of data mining is to charm the data into a confession. An informal way to define data mining is to say that we are looking for a needle in a haystack without knowing what a needle looks like and where the haystack is located.

10.2.4 Data Mining Methods and Techniques

A wide variety of techniques and methods are commonly used in data mining applications. Data mining often involves clustering, the building of predictive regression or classification models, attribute and/or feature selection, the formation of rules and outlier or novelty detection. These techniques can be based on statistics, probability theory, Bayesian networks, decision trees, association rules, neural networks, evolutionary computation, and fuzzy logic.

While the reader may already be familiar with some of these techniques, they often have an additional flavor to them, when it comes to data mining. It is not the purpose of this introduction to discuss data mining in wide breadth, but rather to emphasize the proper use of soft computing techniques for scientific data mining in the context of the fusion of soft computing and hard computing methodologies.

Rather than exposing a breadth of data mining techniques we will introduce here only direct kernel methods for predictive data mining, feature detection and visualization. Direct-kernel based techniques will then be applied to a challenging problem related to the prediction of ischemia from magneto-cardiogram data.

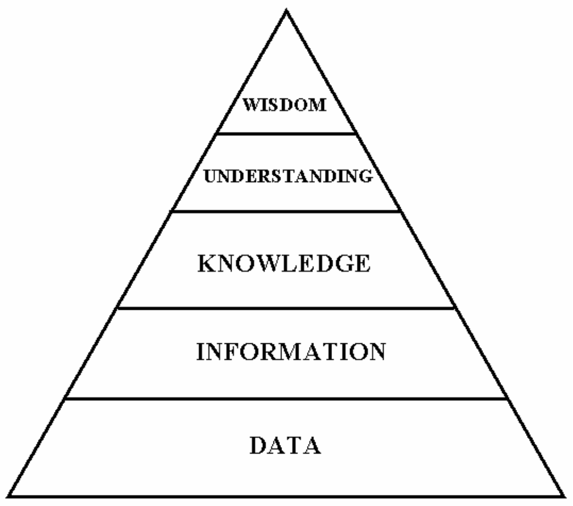


Figure 10.2. Data mining wisdom pyramid.

10.3 BASIC DEFINITIONS FOR DATA MINING

10.3.1 The MetaNeural Data Format

In this section, the standard data mining problem will be introduced and it will be shown how the standard data mining problem actually relates to many interesting types of real-world applications. It is assumed here that the data are already prepared and available in the form of a single, i.e., flat, data file, rather than a relational database. Note that extracting such flat files from different databases is often a challenging task on its own. Consider the flat file data from Table 10.1, provided by Svante Wold [13]. They actually represent a cartoon example for a QSAR or QSPR (quantitative structural property and quantitative structural activity relationship) problem [11], where it is often the purpose to predict chemical properties in the case of QSPR and bio-activities in the case of QSAR from other basic properties (or molecular descriptors) from a molecular dataset. In this case, the activity of interest (or in data mining lingo, the response) for which we would like to make a model is in the second column, represented by DDGTS.

Table 10.1 is a spreadsheet-like table, with 20 horizontal row entries and 9 vertical fields. The first row contains names MOL, DDGTS, PIE, PIF, DGR, SAC, MR, Lam and Vol that describe entries in each vertical field. The first column actually contains the abbreviations for 19 amino acid (AA) names (i.e., all the coded amino acids, except arginine). The second column contains the free energy for unfolding a protein. This free energy, nicknamed DDGTS, is called the response. In this case, we want to build a predictive model for the response based on the remaining seven fields, which contain the chemical properties for the 19 amino-acids listed here. Data entries that are used in predictive models are called descriptors, attributes, or descriptive attributes. Sometimes they are also called features. In a machine learning context, a feature, strictly speaking, is not the same as an attribute, but rather a combination of descriptors, such as principal components in principal component analysis [19] and latent variables in the case of partial least-squares or PLS methods [13]. In this case, PIE and PIF are the lipophilicity constants of the AA side, DGR is the free energy of transfer of an AA side chain from protein interior to water, SAC is the water-accessible surface area, MR the molecular refractivity, Lam is a polarity parameter and Vol represents the molecular volume.

Table 10.1 Example of a flat data file.

MOL	DDGTS	PIE	PIF	DGR	SAC	MR	Lam	Vol
Ala	8.5	0.23	0.31	-0.55	254.2	2.126	-0.02	82.2
Asn	8.2	-0.48	-0.6	0.51	303.6	2.994	-1.24	112.3
Asp	8.5	-0.61	-0.77	1.2	287.9	2.994	-1.08	103.7
Cys	11	0.45	1.54	-1.4	282.9	2.933	-0.11	99.1
Gln	6.3	-0.11	-0.22	0.29	335	3.458	-1.19	127.5
Glu	8.8	-0.51	-0.64	0.76	311.6	3.243	-1.43	120.5
Gly	7.1	0	0	0	224.9	1.662	0.03	65
His	10.1	0.15	0.13	-0.25	337.2	3.856	-1.06	140.6
Ile	16.8	1.2	1.8	-2.1	322.6	3.35	0.04	131.7
Leu	15	1.28	1.7	-2	324	3.518	0.12	131.5
Lys	7.9	-0.77	-0.99	0.78	336.6	2.933	-2.26	144.3
Met	13.3	0.9	1.23	-1.6	336.3	3.86	-0.33	132.3
Phe	11.2	1.56	1.79	-2.6	366.1	4.638	-0.05	155.8
Pro	8.2	0.38	0.49	-1.5	288.5	2.876	-0.32	106.7
Ser	7.4	0	-0.04	0.09	266.7	2.279	-0.4	88.5
Thr	8.8	0.17	0.26	-0.58	283.9	2.743	-0.53	105.3
Trp	9.9	1.85	2.25	-2.7	401.8	5.755	-0.31	185.9
Tyr	8.8	0.89	0.96	-1.7	377.8	4.791	-0.84	162.7
Val	12	0.71	1.22	-1.6	295.1	3.054	-0.13	115.6

While this table is definitely informative, we will introduce some conventions and standard formats here to make it easier for a computer program to automate the data analysis procedure. It has been our experience that, when looking at the data related to many different industrial applications, there is no standard way of presenting data. Each applications has its own different way of presenting data and often a lot of time is spent just trying to read and organize these data before actually doing an analysis or starting the data mining cycle. We will therefore first rearrange and format data into a standard shape, so that we can feed them into a computer program or data mining software. We will be very unforgiving when it comes to adhering to this standard format, in order to reduce the amount of potential computer problems. There is no uniform flat file standard in the data mining community, and each data mining program assumes that the data are organized and presented differently. We will introduce here just one way to organize data: the MetaNeural format. The MetaNeural format will be assumed as the standard format for data representation in this chapter.

An intermediate step towards the MetaNeural format is presented in Table 10.2, which contains almost the same information as Table 10.1, but with a few changes. (i) The column containing the names for each data entry is now placed last and the names are translated into numerical ID numbers 1 – 19. (ii) The response of interest, DDGTS, is now made the next to last column field, and the descriptive alphanumerical entry is now called Response. (iii) The first row, or header row, now contains different names, indicating the columns with descriptive features or attributes (Feature_1 … Feature_7), followed by one or more names for the response, followed by the ID.

Table II. Different representation for the data of Table 10.2.

Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Response	ID
0.23	0.31	-0.55	254.2	2.126	-0.02	82.2	8.5	1
-0.48	-0.6	0.51	303.6	2.994	-1.24	112.3	8.2	2
-0.61	-0.77	1.2	287.9	2.994	-1.08	103.7	8.5	3
0.45	1.54	-1.4	282.9	2.933	-0.11	99.1	11	4
-0.11	-0.22	0.29	335	3.458	-1.19	127.5	6.3	5
-0.51	-0.64	0.76	311.6	3.243	-1.43	120.5	8.8	6
0	0	0	224.9	1.662	0.03	65	7.1	7
0.15	0.13	-0.25	337.2	3.856	-1.06	140.6	10.1	8
1.2	1.8	-2.1	322.6	3.35	0.04	131.7	16.8	9
1.28	1.7	-2	324	3.518	0.12	131.5	15	10
-0.77	-0.99	0.78	336.6	2.933	-2.26	144.3	7.9	11
0.9	1.23	-1.6	336.3	3.86	-0.33	132.3	13.3	12
1.56	1.79	-2.6	366.1	4.638	-0.05	155.8	11.2	13
0.38	0.49	-1.5	288.5	2.876	-0.32	106.7	8.2	14
0	-0.04	0.09	266.7	2.279	-0.4	88.5	7.4	15
0.17	0.26	-0.58	283.9	2.743	-0.53	105.3	8.8	16
1.85	2.25	-2.7	401.8	5.755	-0.31	185.9	9.9	17
0.89	0.96	-1.7	377.8	4.791	-0.84	162.7	8.8	18
0.71	1.22	-1.6	295.1	3.054	-0.13	115.6	12	19

In order to convert this flat data file to the MetaNeural format, all the alphanumerical information in the file will be discarted. This is done by just eliminating the first header row in the file, as shown in Table III. Basically, the MetaNeural format contains only numerical information, where the data are ordered as follows: first are the descriptors or attributes, next is the response (or responses, in the rarer case of multiple responses), and finally some record ID. If the original data contained symbolic or descriptive attribute entries, they have to be somehow converted to numbers.

Data sets often contain missing data. It is a standard practice to code missing data as "-999". Before actually processing the data, it is often common to drop columns and/or rows containing many data entries with –999, or replace the –999 data with the average value for the corresponding data descriptor.

Table 10.3. The MetaNeural format as a standard format for presenting flat file data.
`

0.23	0.31	-0.55	254.2	2.126	-0.02	82.2	8.5	1
-0.48	-0.6	0.51	303.6	2.994	-1.24	112.3	8.2	2
-0.61	-0.77	1.2	287.9	2.994	-1.08	103.7	8.5	3
0.45	1.54	-1.4	282.9	2.933	-0.11	99.1	11	4
-0.11	-0.22	0.29	335	3.458	-1.19	127.5	6.3	5
-0.51	-0.64	0.76	311.6	3.243	-1.43	120.5	8.8	6
0	0	0	224.9	1.662	0.03	65	7.1	7
0.15	0.13	-0.25	337.2	3.856	-1.06	140.6	10.1	8
1.2	1.8	-2.1	322.6	3.35	0.04	131.7	16.8	9
1.28	1.7	-2	324	3.518	0.12	131.5	15	10
-0.77	-0.99	0.78	336.6	2.933	-2.26	144.3	7.9	11
0.9	1.23	-1.6	336.3	3.86	-0.33	132.3	13.3	12
1.56	1.79	-2.6	366.1	4.638	-0.05	155.8	11.2	13
0.38	0.49	-1.5	288.5	2.876	-0.32	106.7	8.2	14
0	-0.04	0.09	266.7	2.279	-0.4	88.5	7.4	15
0.17	0.26	-0.58	283.9	2.743	-0.53	105.3	8.8	16
1.85	2.25	-2.7	401.8	5.755	-0.31	185.9	9.9	17
0.89	0.96	-1.7	377.8	4.791	-0.84	162.7	8.8	18
0.71	1.22	-1.6	295.1	3.054	-0.13	115.6	12	19

10.3.2 The "Standard Data Mining Problem"

We will define the standard (predictive) data mining problem as a regression problem of predicting the response from the descriptive features. In order to do so, we will first build a predictive model based on training data, evaluate the performance of this predictive model based on validation data, and finally use this predictive model to make actual predictions on a test data set for which we generally do not know, or pretend not to know, the response value. There are many different ways to build such predictive regression models. Just to mention a few possibilities here, such a regression model could be a linear statistical model, a Neural Network based model (NN), or a Support Vector Machine (SVM) based model. Examples for linear statistical models are Principal Component Regression models (PCR) and Partial-Least Squares models (PLS). Popular examples of neural network-based models include feedforward neural networks (trained with one of the many popular learning methods), Sef-Organizing Maps (SOMs), and Radial Basis Function Networks (RBFN). Examples of Support Vector Machine algorithms include the perceptron-like SVM for classification, and Least-Squares Support Vector Machines (LS-SVM), also known as kernel ridge regression.

It is customary to denote the data matrix as X_{nm} and the response vector as $\vec{y}_n$. In this case, there are *n* data points and *m* descriptive features in the dataset. We would like to infer $\vec{y}_n$ from X_{nm} by induction, denoted as $X_{nm} \Rightarrow \vec{y}_n$, in such a way that our inference model works not only for the training data, but also does a good job on the out-of-sample data (i.e., validation data and test data). In other words, we aim at building a linear predictive model of the type:

$$\hat{\vec{y}}_n = X_{nm}\vec{w}_m \qquad (10.1)$$

The hat symbol indicates that we are making predictions that are not perfect (especially for the validation and test data). Equation (10.1) answers to the question "wouldn't it be nice if we could apply wisdom to the data, and pop comes out the answer?" The vector $\vec{w}_n$ is that wisdom vector and is usually called the weight vector in machine learning. By introducing the standard data mining problem we are doing a lot of over-simplifying. In a typical data mining study, the questions related to what we are trying to find out are not a priori defined. In this context, precisely formulating the right questions might actually be a more difficult task than answering them. A typical data mining case is therefore more complex than the standard data mining problem. There actually might be many regression models involved and there will be a whole set of additional constraints as well. For example, a more realistic problem for data mining that is still close to the standard data mining problem might be picking an a priori unspecified small subset of the chemically most transparent descriptive features or descriptors from Table 10.3, in order to make a good model for the protein folding energy. Note that now we are using terms that are not precisely defined (but "fuzzy") such as "small subset," "chemically most transparent", and "pretty good model." Also, keep in mind that the predictive model that was proposed so far is strictly linear. It therefore can't be expected that the model is going to be very accurate, but that will change very soon.

It should be pointed out that many problems that are typical for data mining can be posed as a variation of the standard data mining problem. In the above case, we considered a regression problem. Note, that a classification problem can be treated as a special case of a regression problem, e.g., the data entries for the response could be just –1 and 1 in the case of a two-class classification problem. For a multi-class classification problem, the different classes could be presented as 1, 2, 3, and so on. However, there is one difficulty for multi-class classification. If the multi-class classification problem is posed as a regression problem, the classes should be ordinal, i.e., class 2 should be indicative for a response that is in between class 1 and class 3. In practice, that is often the case, e.g., consider the case where we have five alert levels from 1 to 5, where a higher alert number means a more severe type of alert. On the other hand, when we have a classification problem where the classes represent five cities, they are usually not fully ordinal. In that case, it is common to represent the response not as a single response, but encode the response on orthogonal set of 5 response vectors of the type {0, 0, 0, 0, 1}, {0, 0, 0, 1, 0}, {0, 0, 1, 0, 0}, and so on. Non-orthogonal classification problems are often not-trivial to solve, and we refer the reader to the literature [3] for a further discussion.

A different and difficult classification problem is the case with just two classes that have very unbalanced representation, i.e., cases in which there are much more samples from one class than from the other. Consider the cartoon problem of a trinket manufacturing case were a product line produces 1000000 good trinkets that pass the inspection line and 1000 defect trinkets that should not pass the inspection line before the product is shipped out. It is often difficult to build good models for such problems without being aware that we are dealing here with an outlier problem. Naively applying machine learning models could result in the case where the model now predicts that all trinkets belong to the majority class. Concluding that now only 1000 out 1001000 cases are missed, and that the classification is therefore 99.9% correct, does not make sense of course, because in reality 100% of the cases that we are interested in catching are now missed altogether. It therefore should be a common practice for classification problems to represent the results in the form of a confusion matrix. In case of a binary classification problem where the classifications results can be presented as four numbers: the number of true negatives, the number of false negatives, the number of true positives and the number of false positives. The false negatives and false positives are called Type I and Type II errors. For medical applications, it is customary to convert these numbers further into sensitivity and specificity measures.

Note that outlier problems do not only occur in classification problems, but also in regression problems. Predicting the swing for the stock market is a regression problem, and predicting a stock market crash is an outlier problem in that context. Predicting rare events such as stock market crashes, earthquakes, or tornados can also be considered as extreme outlier problems or rare event problems. Such cases are usually very difficult if not impossible to deal with using data mining techniques.

A special case of an outlier problem is novelty detection, where we often have a dataset consisting of normal cases and maybe a few instances of abnormal cases. The different categories for abnormal cases are often not known a priori. The problem is now to devise a detection model that tags data samples that are very different from what has been seen before.

A difficult type of data mining problem is a data strip mining problem. Data strip mining is a special case of predictive data mining, where the data have much more descriptors than there are data. Such problems are common for in-silico drug design and the analysis of gene-expression arrays. The task is now to identify a combination of a subset of the most relevant features and make a predictive model that works well on external test data.

A whole different class of data mining relates to signal and time series analysis. Time series analysis problems are of common interest to the finance industry, process control, and medical diagnostics. Challenging time series analysis problems deal with multi-variate time series, problems where the time series exhibits complex dynamics (such as nonlinear chaotic time sequences), or cases with the non-stationary time series.

Note that for a legitimate data mining case, we are not just interested in building predictive models. What is of real interest is the understanding and explaining of these models (e.g., in the form of a fuzzy rules system). It is often revealing for such a rule set to identify the most important descriptors and/or features, and then explain what these descriptors or features do for the model. Feature detection is a sub-discipline on its own and an important part of the data mining process. There are many different ways for feature detection, and the most appropriate method depends on the modeling method, the data characteristics and the application domain. It is important to point out here that the most interesting features are often not the most correlated features.

10.3.3 Predictive Data Mining

In this section, a simple statistical regression solution to the standard data mining problem will be introduced. In this context, the standard data mining problem can be interpreted as a predictive data mining problem.

Looking back at Eq. (10.1), let us try now whether we can actually make a “pop-comes-out the answer” model for Svante Wold’s cartoon QSAR data from Table 10.3. In this case, one is actually trying to predict the free energy for protein folding, or the entry next to last column, based on the descriptive features in the prior columns. A model will be constructed here by finding an approximate solution for the weights in (10.1). Note that the data matrix is generally not symmetric. If that were the case, it would be straightforward to come up with an answer by using the inverse of the data matrix. We will therefore apply the pseudo-inverse transformation, which will generally not lead to precise predictions for y, but will predict y in a way that is optimal in a least-squares sense. The pseudo-inverse solution for the weight vector is illustrated in equation below:

$$\begin{aligned} X_{mn}^T X_{nm} \vec{w}_m &= X_{mn}^T \vec{y}_n \\ \left(X_{mn}^T X_{nm}\right)^{-1} \left(X_{mn}^T X_{nm}\right) \vec{w}_m &= \left(X_{mn}^T X_{nm}\right)^{-1} X_{mn}^T \vec{y}_n \quad (10.2) \\ \vec{w}_m &= \left(X_{mn}^T X_{nm}\right)^{-1} X_{mn}^T \vec{y}_n \end{aligned}$$

Predictions for the training set can now be made for y by substituting (10.2) in (10.1):

$$\hat{\vec{y}}_n = X_{nm} \left(X_{mn}^T X_{nm}\right)^{-1} X_{mn}^T \vec{y}_n \quad (10.3)$$

Before applying this formula for the prediction of the free binding energy of amino acids, we have to introduce one more important stage of the data mining cycle: data pre-processing. The seven descriptors for the amino acids have very different underlying metrics and scales, i.e., some columns have all very small entries and other columns have relatively large entries. It is a common procedure in data mining to center all the descriptors and to bring them to a unity variance. The same process is then applied to the response. This procedure of centering and variance normalization is known as Mahalanobis scaling [ref]. While Mahalanobis scaling is not the only way to pre-process the data, it is probably the most general and the most robust way to do pre-processing that applies well across the board. If we represent a feature vector as $\vec{z}$, Mahalanobis scaling will result in a rescaled feature vector $\vec{z}'$ and can be summarized as:

$$\vec{z}' = \frac{\vec{z} - \bar{z}}{std(\vec{z})} \quad (10.4)$$

where $\bar{z}$ represents the average value and $std(\vec{z})$ represents the standard deviation for attribute $\vec{z}$. After Mahalanobis scaling, the data matrix from Table 10.3 now changes to Table 10.4.

Table 10.4. Amino acid protein folding data after Mahalanobis scaling during preprocessing.

-0.205	-0.240	0.195	-1.340	-1.233	0.881	-1.344	-0.502	1
-1.149	-1.155	1.067	-0.204	-0.336	-1.022	-0.318	-0.611	2
-1.322	-1.326	1.635	-0.565	-0.336	-0.772	-0.611	-0.502	3
0.088	0.997	-0.504	-0.680	-0.399	0.740	-0.768	0.405	4
-0.657	-0.773	0.886	0.518	0.144	-0.944	0.200	-1.300	5
-1.189	-1.195	1.273	-0.020	-0.079	-1.318	-0.039	-0.393	6
-0.511	-0.552	0.647	-2.013	-1.713	0.959	-1.930	-1.010	7
-0.311	-0.421	0.442	0.569	0.555	-0.741	0.646	0.078	8
1.086	1.259	-1.080	0.233	0.032	0.974	0.343	2.508	9
1.193	1.158	-0.998	0.265	0.206	1.099	0.336	1.855	10
-1.535	-1.547	1.289	0.555	-0.399	-2.612	0.772	-0.719	11
0.687	0.686	-0.669	0.548	0.559	0.397	0.363	1.239	12
1.566	1.249	-1.492	1.234	1.363	0.834	1.164	0.477	13
-0.005	-0.059	-0.587	-0.551	-0.458	0.413	-0.509	-0.611	14
-0.511	-0.592	0.721	-1.052	-1.075	0.288	-1.129	-0.901	15
-0.284	-0.290	0.170	-0.657	-0.596	0.085	-0.557	-0.393	16
1.951	1.712	-1.574	2.055	2.518	0.428	2.189	0.006	17
0.674	0.414	-0.751	1.503	1.522	-0.398	1.399	-0.393	18
0.434	0.676	-0.669	-0.399	-0.274	0.709	-0.206	0.767	19

The modeling results obtained from Eq. (10.3) after de-scaling the “predictions” back to the original distribution are shown in Fig. 10.3. After a first inspection, these “predictions” do not look bad. However, there is one caveat: these are “predictions” for training data. It is imperative for predictive data mining to verify how good the model really is on a validation set, i.e., a set of data that was not used for training. Of course, there is now a problem here: there are only 19 data points. If we would build a model using 13 data points and test on the remaining 6 data points, the model is probably not going to be as accurate as it potentially could be, because all the available data were not used for model building. There are several ways out of that dilemma. (i) Because data mining in principle deals with large data sets, there are normally enough data available to split the dataset up in a training set and a test set. A good practice would be to use a random sample of 2/3 of the data for the training, and 1/3 for testing. (ii) If one is truly concerned about compromising the quality of the model, one can follow the leave-one-out (LOO) method. In this case, one would build 19 different models, each time using 18 data for training and test on the one remaining data-point. The 19 individual tests would then be combined and displayed in a simple plot similar to the plot of Fig. 10.4. (iii) An obvious extension to the LOO practice is to leave several samples out. This procedure is called bootstrapping and in analogy with the LOO method will be indicated by the acronym BOO. Using it, it is easy to make actually 100 or more models from the 19 data samples, leaving three samples out each time for testing, and then combining the predictions.

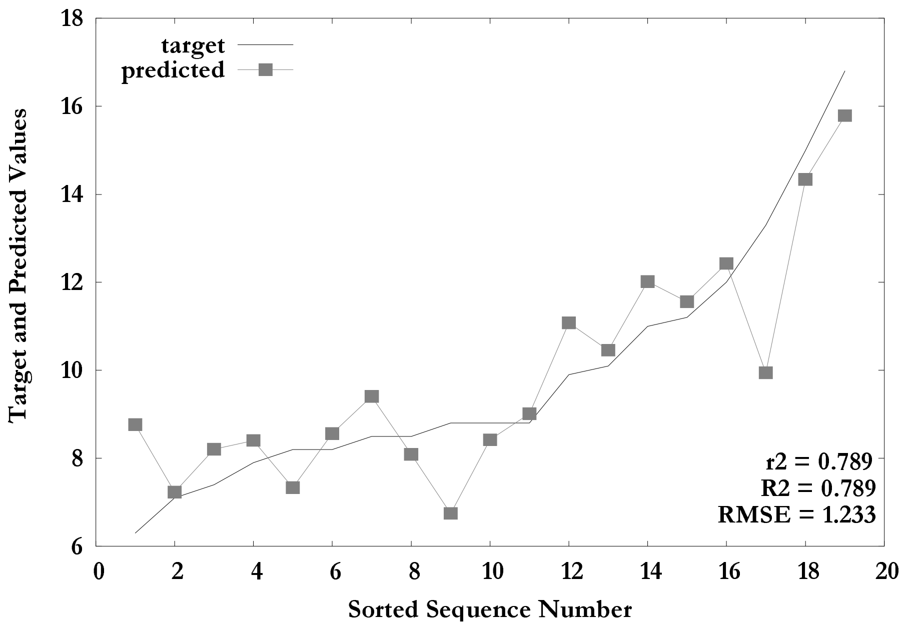


Fig. 10.3 Predictions for training model for the 19 amino-acid data form Table IV.

Making a test model proceed in a very similar way as for training: the "wisdom vector" or the weight vector will be applied to the test data to make predictions according to:

$$\hat{\vec{y}}_k^{test} = X_{km}^{test} \vec{w}_m \quad (10.5)$$

In the above expression it was assumed that there are *k* test data, and the subscript 'test" is use to explicitly indicate that the wisdom vector will be applied to a set of *k* test data with *m* attributes or descriptors. If one considers testing for one sample data point at a time, Eq. (10.5) can be represented as a simple neural network with an input layer and just a single neuron, as shown in Fig. 10.4. The neuron produces the weighted sum of the average input features. Note that the transfer function, commonly found in neural networks, is not present here. Note also that that the number of weights for this one-layer neural networks equals the number of input descriptors or attributes.

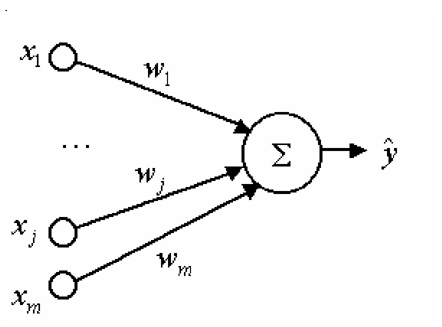


Fig 10.4. Neural network representation for the simple regression model.

Let us just proceed with training the simple learning model on the first 13 data points, and make predictions on the last six data points. The results are shown in Fig. 10.5. It is clear that this model looks less convincing for being able to make good predictions on the test data.

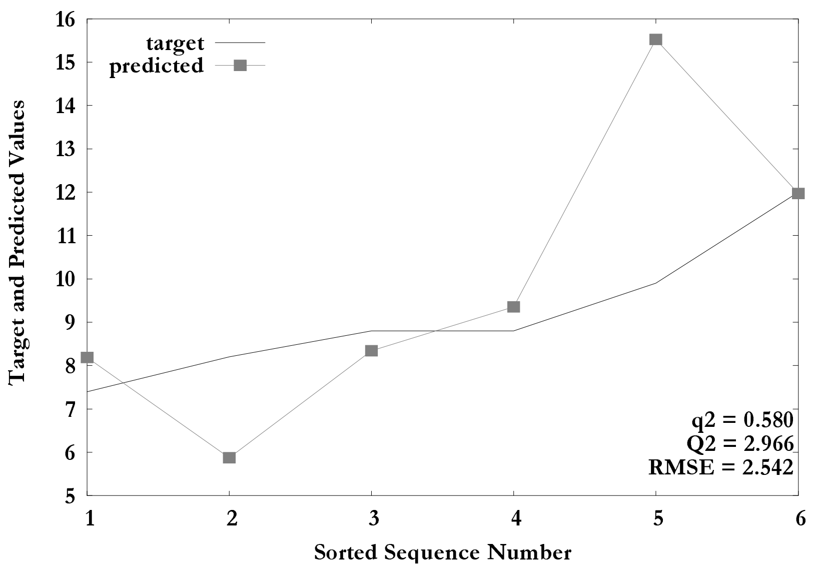


Fig. 10.5 Test data predictions for the simple regression model trained on the first 13 data samples in Table10.4 and tested on the six last data samples.

10.3.4 Metrics for Assessing the Model Quality

An obvious question that now comes to mind is how to assess or describe the quality of a model for training data and test data, such as is the case for the data shown in Figs. 10.4 and 10.6. In the case of a classification problem that would be relatively easy, and one would ultimately present the number of hits and misses in the form of a confusion matrix as described earlier. For a regression problem, a common way to capture the error is by the Root Mean Square Error index or RMSE, which is defined as the average value of the squared error (either for the training set or the test set) according to:

$$RMSE = \sqrt{\frac{1}{n}\sum_i \left(\hat{y}_i - y_i\right)^2} \quad (10.6)$$

While the root mean square error is an efficient way to compare the performance of different prediction methods on the same data, it is not an absolute metric in the sense that the RMSE will depend on how the response for the data was scaled. In order to overcome this handicap, additional error measures will be introduced that are less dependent on the scaling and magnitude of the response value. A first metric that will be used for assessing the quality of a trained model is r^2, where r^2 is defined as the correlation coefficient squared between target values and predictions for the response according to:

$$r^2 = \frac{\sum_{i=1}^{n_{\mathrm{train}}} (\hat{y}_i - \bar{y})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n_{\mathrm{train}}} (\hat{y}_i - \bar{y})^2}\sqrt{\sum_{i=1}^{n_{\mathrm{train}}} (y_i - \bar{y})^2}} \quad (10.7)$$

where n_{train} represents the number of data points in the training set. r^2 takes values between zero and unity, and the higher the r^2 value, the better the model. An obvious drawback of r^2 for assessing the model quality is that r^2 only expresses a linear correlation, indicating how well the predictions follow a line if $\hat{y}$ is plotted as function of y. While one would expect a nearly perfect model when r^2 is unity, this is not always the case. A second and more powerful measure to assess the quality of a trained model is the so-called “Press r squared”, or R^2, often used in chemometric modeling [14], where R^2 is defined as [15]:

$$R^2 = 1 - \frac{\sum_{i=1}^{n_{train}} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n_{train}} (y_i - \bar{y})^2} \quad (10.8)$$

We consider R^2 as a better measure than r^2, because it accounts for the residual error as well. The higher the value for R^2, the better the model. Note that in certain cases the R^2 metric can actually be negative. The R^2 metric is commonly used in chemometrics and is generally smaller than r^2. For large datasets, R^2 tends to converge to r^2, and the comparison between r^2 and R^2 for such data often reveals hidden biases.

For assessing the quality of the validation set or a test set, we will introduce similar metrics, q^2 and Q^2, where q^2 and Q^2 are defined as $1-r^2$ and $1-R^2$ for the data in the test set. For a model that perfectly predicts on the test data, we now would expect q^2 and Q^2 to be zero. The reason for introducing metrics that are symmetric between the training set and the test set is actually to avoid confusion. Q^2 and q^2 values will always apply to a validation set or a test set, and that we would expect these values to be quite low in order to have a good predictive model. R^2 and r^2 values will always apply to training data, and should be close to unity for a good training model. For the example above, trained on 13 training data, we obtained RMSE = 0.1306, r^2 = 0.826, and

$R^2 = 0.815$ for the training data. Similarly, with a model in which the six data points are put aside for the validation set, we obtained 2.524, 0.580, 2.966, for the RMSE, q^2, and Q^2 respectively.

Note that for the above example Q^2 is significantly larger than unity (2.966). While $0 < q^2 < 1$, inspecting Eq. (10.8) reveals that this upper limit does not hold anymore for Q^2. The Q^2 measure for the six test data actually indicates that the predictions on the test data are poor. The large difference between q^2 and Q^2 indicates that the model also has a lot of uncertainty. Looking at Fig.10.6a, this conclusion is not entirely obvious: the predictions in this figure seem to follow the right trend, and there are two data points that are clearly missed in the predictions. A better type of plot, that clearly supports the conclusions from the q^2 and Q^2 analysis, is the scatterplot. A scatterplot is a plot where the true values are indicated on the horizontal axis and the predicted values correspond to the *y*-axis. For a perfect predictive model, all the data should fall on the main diagonal. Fig.10.6b shows the scatterplot for the six test data for the amino acid example. From looking at the scatterplot, it now becomes immediately clear that the predictive model is not good at all in predicting on test data. The scatterplot on the left hand side is for the six test data (the six last samples from Table 10.3), while the scatterplot on the right hand side is obtained from running 200 different bootstraps, and testing on a random selection of six test samples. The variance on the bootstrap predictions for this case is indicated with error bars on the figure.

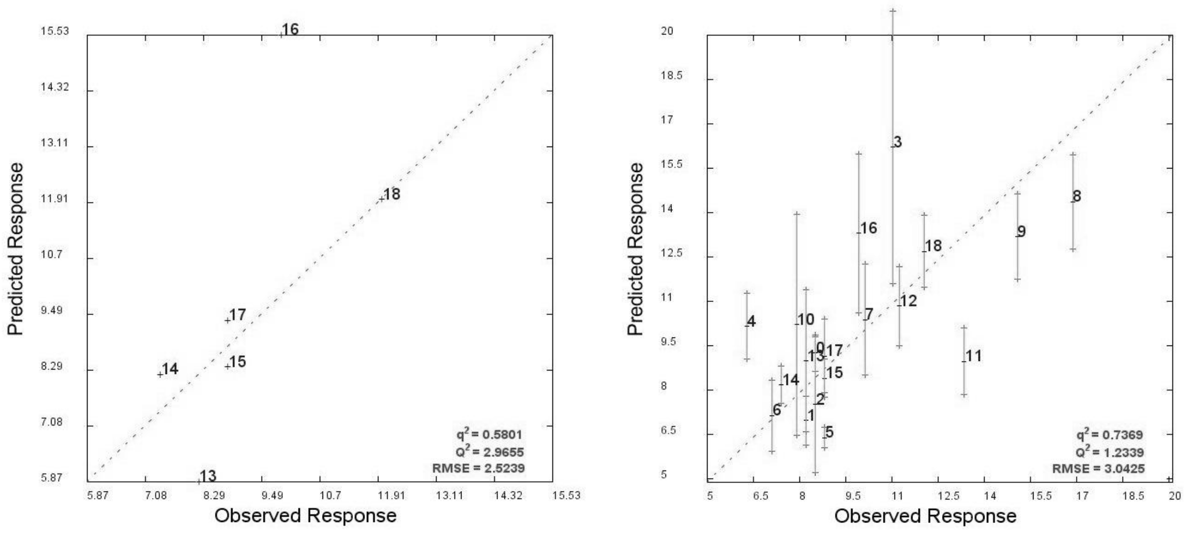


Figure 10.6 Scatterplot for predictions on (a) six test data for the amino acid example, and (b) 200 bootstraps with six sample test data each.

10.4 INTRODUCTION TO DIRECT KERNEL METHODS

10.4.1 The data mining dilemma and the machine learning dilemma for real-world data

The example above is a simple "toy" problem and rather naïve. Real-world data mining problems differ in many ways. Real-world datasets can be vast. They are often so large that they cannot be elegantly presented and looked at in a spreadsheet anymore. Furthermore, real-world data sets have missing data, errors, outliers, and minority classes.

There is also the problem of diminishing “information density” in large datasets. As datasets become larger and larger, we would expect, on the one hand, that there is more information out there to build good and robust models. On the other hand, there might also be so much spurious and superfluous data in the dataset that the information density is actually lower. Even if it is possible to obtain better models from larger datasets because there is more useful and relevant information out there, it is actually a harder to extract that information. We call this the phenomenon “the data mining dilemma.”

Another observation will be even more fundamental for predictive data mining. Looking back at equations (2) and (3) it can be noticed that they contain the inverse of the feature kernel, K_F, defined as:

$$K_F = X_{mn}^T X_{nm} \quad (10.9)$$

The feature kernel is a $m \times m$ symmetric matrix where each entry represents the similarity between features. Obviously, if there were two features that would be completely redundant the feature matrix would contain two columns and two rows that are (exactly) identical, and the inverse does not exist. For the case of data strip mining problems, where there are more descriptors than data, this matrix would be rank deficient and the inverse would not exist. Consequently, the simple regression model we proposed above would not work anymore. One can argue that all is still well, and that in order to make the simple regression method work one would just make sure that the same descriptor or attribute is not included twice. By the same argument, highly correlated descriptors (i.e., “cousin features” in data mining lingo) should be eliminated as well. While this argument sounds plausible, the truth of the matter is more subtle. Let us repeat Eq. (10.2) again and go just one step further as shown below.

$$\begin{aligned} X_{mn}^T X_{nm} \vec{w}_m &= X_{mn}^T \vec{y}_n \\ \left(X_{mn}^T X_{nm}\right)^{-1}\left(X_{mn}^T X_{nm}\right)\vec{w}_m &= \left(X_{mn}^T X_{nm}\right)^{-1} X_{mn}^T \vec{y}_n \\ \vec{w}_m &= \left(X_{mn}^T X_{nm}\right)^{-1} X_{mn}^T \vec{y}_n \\ \vec{w}_m &= X_{mn}^T \left(X_{nm} X_{mn}^T\right)^{-1} \vec{y}_n \end{aligned} \quad (10.10)$$

Eq. (10.10) is the derivation of an equivalent linear formulation to (2.2), based on the so-called right-hand pseudo-inverse or Penrose inverse, rather than using the more common left-hand pseudo-inverse. It was not shown here how that last line followed from the previous equation, but the proof is straightforward and left as an exercise to the reader. Note that now the inverse is needed for a different entity matrix, which now has an $n \times n$ dimensionality, and is called the data kernel, K_D, as defined by:

$$K_D = X_{nm} X_{mn}^T \quad (10.11)$$

The right-hand pseudo-inverse formulation is less frequently cited in the literature, because it can only be non-rank deficient when there are more descriptive attributes than data points, which is not the usual case for data mining problems (except for data strip mining cases). The data kernel matrix is a symmetrical matrix that contains entries representing similarities between data points. The solution to this problem seems to be straightforward. We will first try to explain here what seems to be an obvious solution, and then actually show why this won't work. Looking at Eqs. (10.10) and (10.11) it can be concluded that, except for rare cases where there are as many data records as there are features, either the feature kernel is rank deficient (in case that $m > n$, i.e., there are more attributes than data), or the data kernel is rank deficient (in case that $n > m$, i.e., there are more data than attributes). It can be now argued that for the $m < n$ case one can proceed with the usual left-hand pseudo-inverse method of Eq. (10.2), and that for the $m > n$ case one should proceed with the right-hand pseudo inverse, or Penrose inverse following Eq. (10.10).

While the approach just proposed here seems to be reasonable, it will not work. Learning occurs by discovering patterns in data through redundancies present in the data. Data redundancies imply that there are data present that seem to be very similar to each other (and that have similar values for the response as well). An extreme example for data redundancy would be a dataset that contains the same data point twice. Obviously, in that case, the data matrix is ill-conditioned and the inverse does not exist. This type of redundancy, where data repeat themselves, will be called here a "hard redundancy." However, for any dataset that one can possibly learn from, there have to be many "soft redundancies" as well. While these soft redundancies will not necessarily make the data matrix ill-conditioned, in the sense that the inverse does not exist because the determinant of the data kernel is zero, in practice this determinant will be very small. In other words, regardless whether one proceeds with a left-hand or a right-hand inverse, if data contain information that can be learnt from, there have to be soft or hard redundancies in the data. Unfortunately, Eqs. (10.2) and (10.10) can't be solved for the weight vector in that case, because the kernel will either be rank deficient (i.e., ill-conditioned), or poor-conditioned, i.e., calculating the inverse will be numerically unstable. We call this phenomenon "the data mining dilemma:" (i) machine learning from data can only occur when data contain redundancies; (ii) but, in that case the kernel inverse in Eq. (10.2) or Eq. (10.10) is either not defined or numerically unstable because of poor conditioning. Taking the inverse of a poor-conditioned matrix is possible, but the inverse is not "sharply defined" and most numerical methods, with the exception of methods based on single value decomposition (SVD), will run into numerical instabilities. The data mining dilemma seems to have some similarity with the uncertainty principle in physics, but we will not try to draw that parallel too far.

Statisticians have been aware of the data mining dilemma for a long time, and have devised various methods around this paradox. In the next sections, we will propose several methods to deal with the data mining dilemma, and obtain efficient and robust prediction models in the process.

10.4.2 Regression Models Based on the Data Kernel Model

In this section, we will consider the data kernel formulation of Eq. (10.10) for predictive modeling. Not because we have to, but because this formulation is just in the right form to apply the kernel transformation on test data. There are several well-known methods for dealing with the data mining dilemma by using techniques that ensure that the kernel matrix will not be rank deficient anymore. Two well-known methods are principal component regression [16] and ridge regression [17-18]. In order to keep the mathematical diversions to its bare minimum, only ridge regression will be discussed.

Ridge regression is a very straightforward way to ensure that the kernel matrix is positive definite (or well-conditioned), before inverting the data kernel. In ridge regression, a small positive value, λ, is added to each element on the main diagonal of the data matrix. Usually the same value for λ is used for each entry. Obviously, we are not solving the same problem anymore. In order to not deviate too much from the original problem, the value for λ will be kept as small as we reasonably can tolerate. A good choice for λ is a small value that will make the newly defined data kernel matrix barely positive definite, so that the inverse exists and is mathematically stable. In data kernel space, the solution for the weight vector that will be used in the ridge regression prediction model now becomes:

$$\vec{w}_n = X_{mn}^T \left(X_{nm} X_{mn}^T + \lambda I \right)^{-1} \vec{y}_n \quad (10.12)$$

and predictions for y can now be made according to:

$$\begin{aligned} \hat{\vec{y}} &= X_{nm} X_{mn}^T \left(X_{nm} X_{mn}^T + \lambda I \right)^{-1} \vec{y}_n \\ &= K_D \left(K_D + \lambda I \right)^{-1} \vec{y}_n \qquad (10.13) \\ &= K_D \vec{w}_n \end{aligned}$$

where a very different weight vector was introduced: $\vec{w}_n$. This weight vector is applied directly to the data kernel matrix (rather than the training data matrix) and has the same dimensionality as the number of training data. To make a prediction on the test set, one proceeds in a similar way, but applies the weight vector on the data kernel for the test data, which is generally a rectangular matrix, and projects the test data on the training data according to:

$$K_D^{test} = X_{km}^{test} \left(X_{mn}^{train} \right)^T \quad (10.14)$$

where it is assumed that there are k data points in the test set.

10.4.3 The Kernel Transformation

The kernel transformation is an elegant way to make a regression model nonlinear. The kernel transformation goes back at least to the early 1900's, when Hilbert addressed kernels in the mathematical literature. A kernel is a matrix containing similarity measures for a dataset: either between the data of the dataset itself, or with other data (e.g., support vectors [1]). A classical use of a kernel is the correlation matrix used for determining the principal components in principal component analysis, where the feature kernel contains linear similarity measures between (centered) attributes. In support vector machines, the kernel entries are similarity measures between data rather than features and these similarity measures are usually nonlinear, unlike the dot product similarity measure that we used before to define a kernel. There are many possible nonlinear similarity measures, but in order to be mathematically tractable the kernel has to satisfy certain conditions, the so-called Mercer conditions [1-3].

$$\overleftrightarrow{K}_{nn} = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1n} \\ k_{21} & k_{22} & \dots & k_{2n} \\ & \dots & & \\ k_{n1} & k_{n2} & \dots & k_{nn} \end{bmatrix} \quad (10.15)$$

The expression above, introduces the general structure for the data kernel matrix, $\overleftrightarrow{K}_{nm}$, for n data. The kernel matrix is a symmetrical matrix where each entry contains a (linear or nonlinear) similarity between two data vectors. There are many different possibilities for defining similarity metrics such as the dot product, which is a linear similarity measure and the Radial Basis Function kernel or RBF kernel, which is a nonlinear similarity measure. The RBF kernel is the most widely used nonlinear kernel and the kernel entries are defined by

$$k_{ij} \equiv e^{-\frac{\left\|\vec{x}_j - \vec{x}_l\right\|_2^2}{2\sigma^2}} \quad (10.16)$$

Note that in the kernel definition above, the kernel entry contains the square of the Euclidean distance (or two-norm) between data points, which is a dissimilarity measure (rather than a similarity), in a negative exponential. The negative exponential also contains a free parameter, σ, which is the Parzen window width for the RBF kernel. The proper choice for selecting the Parzen window is usually determined by an additional tuning, also called hyper-tuning, on an external validation set. The precise choice for σ is not crucial, there usually is a relatively broad range for the choice for σ for which the model quality should be stable.

Different learning methods distinguish themselves in the way by which the weights are determined. Obviously, the model in Eqs. (10.12) - (10.14) to produce estimates or predictions for y is linear. Such a linear model has a handicap in the sense that it cannot capture inherent nonlinearities in the data. This handicap can easily be overcome by applying the kernel transformation directly as a data transformation. We will therefore not operate directly on the data, but on a nonlinear transform of the data, in this case the nonlinear data kernel. This is very similar to what is done in principal component analysis, where the data are substituted by their principal components before building a model. A similar procedure will be applied here, but rather than substituting data by their principal components, the data will be substituted by their kernel transform (either linear or nonlinear) before building a predictive model.

The kernel transformation is applied here as a data transformation in a separate pre-processing stage. We actually replace the data by a nonlinear data kernel and apply a traditional linear predictive model. Methods where a traditional linear algorithm is used on a nonlinear kernel transform of the data are introduced in this chapter as “direct kernel methods.” The elegance and advantage of such a direct kernel method is that the nonlinear aspects of the problem are captured entirely in the kernel and are transparent to the applied algorithm. If a linear algorithm was used before introducing the kernel transformation, the required mathematical operations remain linear. It is now clear how linear methods such as principal component regression, ridge regression, and partial least squares can be turned into nonlinear direct kernel methods, by using exactly the same algorithm and code: only the data are different, and we operate on the kernel transformation of the data rather than the data themselves. This same approach for converting algorithms to direct kernel methods can also be applied to nonlinear learning algorithms such as the self-organizing map [4].

In order to make out-of-sample predictions on true test data, a similar kernel transformation needs to be applied to the test data, as shown in Eq. (10.14). The idea of direct kernel methods is illustrated in Fig. 10.7, by showing how any regression model can be applied to kernel-transformed data. One could also represent the kernel transformation in a neural network type of flow diagram and the first hidden layer would now yield the kernel-transformed data, and the weights in the first layer would be just the descriptors of the training data. The second layer contains the weights that can be calculated with a hard computing method, such as kernel ridge regression. When a radial basis function kernel is used, this type of neural network would look very similar to a radial basis function neural network [19-20], except that the weights in the second layer are calculated differently.

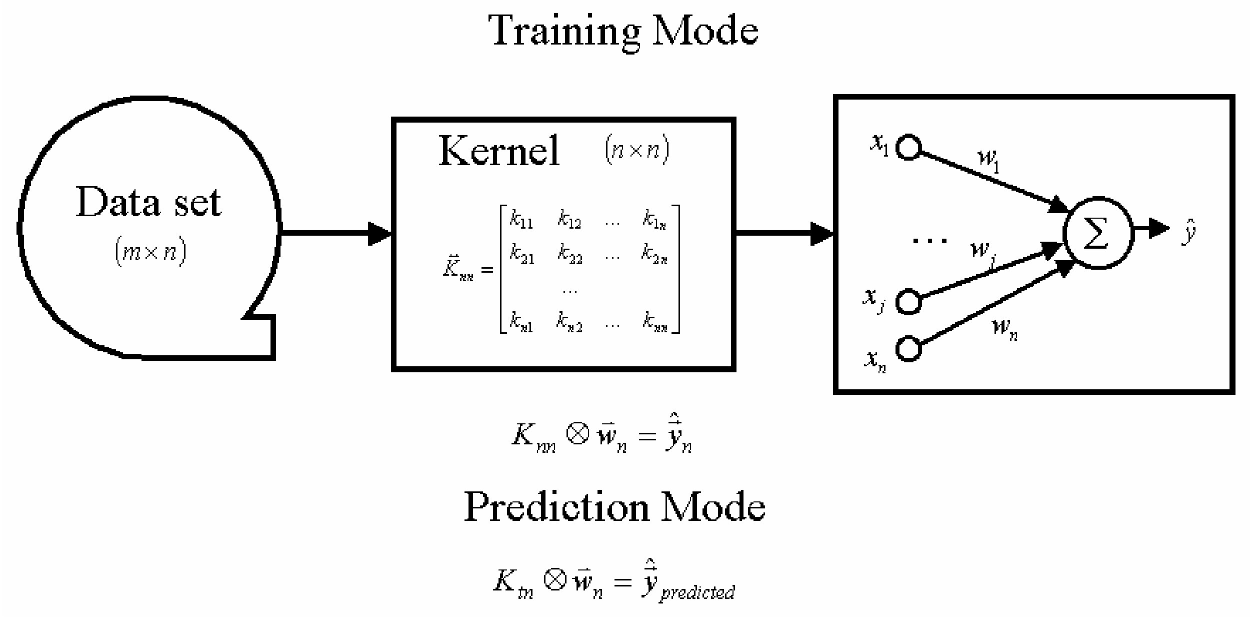

Training Mode
Data set
$(m \times n)$
Kernel $(n \times n)$
$\vec{K}_{nn} = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1n} \\ k_{21} & k_{22} & \dots & k_{2n} \\ & \dots & & \\ k_{n1} & k_{n2} & \dots & k_{nn} \end{bmatrix}$
x_1
x_j
x_n
w_1
w_j
w_n
Σ
$\hat{y}$
$K_{nn} \otimes \vec{w}_n = \hat{\vec{y}}_n$
Prediction Mode
$K_{tn} \otimes \vec{w}_n = \hat{\vec{y}}_{predicted}$

Figure 10.7 Operation schematic for direct kernel methods as a data pre-processing step.

10.4.4 Dealing with the Bias: Centering the Kernel

There is still one important detail that was overlooked so far, and that is necessary to make direct kernel methods work. Looking at the prediction equations in which the weight vector is applied to data as in Eq. (10.1), there is no constant offset term or bias. It turns out that for data that are centered this offset term is always zero and does not have to be included explicitly. In machine learning lingo the proper name for this offset term is the bias, and rather than applying Eq. (10.1), a more general predictive model that includes this bias can be written as:

$$\hat{\hat{y}}_n = X_{nm}\vec{w}_m + b \quad (10.17)$$

where b is the bias term. Because we made it a practice in data mining to center the data first by Mahalanobis scaling, this bias term is zero and can be ignored.

When dealing with kernels, the situation is more complex, as they need some type of bias as well. We will give only a recipe here, that works well in practice, and refer the reader to the literature for a more detailed explanation [3, 21-23]. Even when the data were Mahalanobis-scaled, before applying a kernel transform, the kernel still needs some type of centering to be able to omit the bias term in the prediction model. A straightforward way for kernel centering is to subtract the average from each column of the training data kernel, and store this average for later recall, when centering the test kernel. A second step for centering the kernel is going through the newly obtained vertically centered kernel again, this time row by row, and subtracting the row average form each horizontal row.

The kernel of the test data needs to be centered in a consistent way, following a similar procedure. In this case, the stored column centers from the kernel of the training data will be used for the vertical centering of the kernel of the test data. This vertically centered test kernel is then centered horizontally, i.e., for each row, the average of the vertically centered test kernel is calculated, and each horizontal entry of the vertically centered test kernel is substituted by that entry minus the row average.

Mathematical formulations for centering square kernels are explained in the literature [21-23]. The advantage of the kernel-centering algorithm introduced (and described above in words) in this section is that it also applies to rectangular data kernels. The flow chart for pre-processing the data, applying a kernel transform on this data, and centering the kernel for the training data, validation data, and test data is shown in Fig. 10.8.

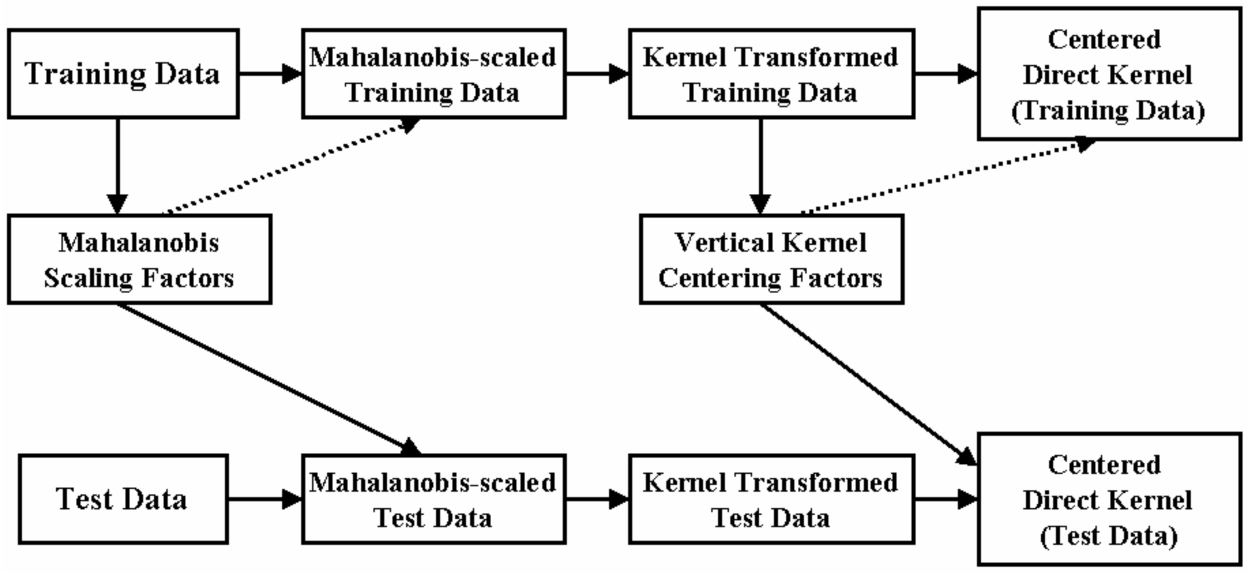

Training Data
Mahalanobis-scaled Training Data
Kernel Transformed Training Data
Centered Direct Kernel (Training Data)
Mahalanobis Scaling Factors
Vertical Kernel Centering Factors
Test Data
Mahalanobis-scaled Test Data
Kernel Transformed Test Data
Centered Direct Kernel (Test Data)

Figure 10.8 Data pre-processing with kernel centering for direct kernel methods.

10.5. DIRECT KERNEL RIDGE REGRESSION

10.5.1 Overview

So far, the argument was made that by applying the kernel transformation in Eqs. (10.13) and (10.14), many traditional linear regression models can be transformed into a nonlinear direct kernel method. The kernel transformation and kernel centering proceed as data pre-processing steps (Fig. 10.8). In order to make the predictive model inherently nonlinear, the radial basis function kernel will be applied, rather than the (linear) dot product kernel, used in Eqs. (10.2) and (10.10). There are actually several alternate choices for the kernel [1-3, 19], but the RBF kernel is the most widely applied kernel. In order to overcome the machine learning dilemma, a ridge can be applied to the main diagonal of the data kernel matrix. Because the kernel transformation is applied directly on the data, before applying ridge regression, this method is called direct-kernel ridge regression.

Kernel ridge regression and (direct) kernel ridge regression are not new. The roots for ridge regression can be traced back to the statistics literature [18]. Methods equivalent to kernel ridge regression were recently introduced under different names in the machine literature (e.g., proximal SVMs were introduced by Mangasarian et al. [24], kernel ridge regression was introduced by Poggio et al. [25-27], and Least-Squares Support Vector Machines were introduced by Suykens et al. [28-29]). In these works, Kerned Ridge Regression is usually introduced as a regularization method that solves a convex optimization problem in a Langrangian formulation for the dual problem that is very similar to traditional SVMs. The equivalency with ridge regression techniques then appears after a series of mathematical manipulations. By contrast, in this chapter direct kernel ridge regression was introduced with few mathematical diversions in the context of the machine learning dilemma. For all practical purposes, kernel ridge regression is similar to support vector machines, works in the same feature space as support vector machines, and was therefore named least-squares support vector machines by Suykens et al. [28-29].

Note that kernel ridge regression still requires the computation of an inverse for a $n \times n$ matrix, that can be quite large. This task is computationally demanding for large datasets, as is the case in a typical data mining problem. Because the kernel matrix now scales with the number of data squared, this method can also become prohibitive from a practical computer implementation point of view, because both memory and processing requirements can be very demanding. Krylov space-based methods and conjugate gradient methods are relatively efficient ways to speed up the matrix inverse transformation of large matrices, where the computation time now scales as n^2, rather than n^3. Krylov-space methods are discussed in [30]. Conjugate gradient-based methods for inverting large matrices are discussed in [1] and [29]. The Analyze/Stripminer [31] code used for the analysis presented here applies MØller's scaled conjugate gradient method to calculate the matrix inverse [32].

The issue of dealing with large datasets is even more profound. There are several potential solutions that will not be discussed in detail. One approach would be to use a rectangular kernel, were not all the data are used as bases to calculate the kernel, but a good subset of "support vectors" is estimated by chunking [1] or other techniques such as sensitivity analysis (explained further on in this chapter). More efficient ways for inverting large matrices are based on piece-wise inversion. Alternatively, the matrix inversion may be avoided altogether by adhering to the support vector machine formulation of kernel ridge regression and solving the dual Lagrangian optimization problem and applying the sequential minimum optimization or SMO algorithm as explained in [33].

10.5.2 Choosing the Ridge Parameter, λ

It has been shown in the literature [29] that kernel ridge regression can be expressed as an optimization method, where rather than minimizing the residual error on the training set, according to:

$$\sum_{i=1}^{n_{\text{train}}} \left\| \hat{\vec{y}}_i - \vec{y}_i \right\|_2 \quad (10.18)$$

we now minimize:

$$\sum_{i=1}^{n_{\text{train}}} \left\| \hat{\vec{y}}_i - \vec{y}_i \right\|_2 + \frac{\lambda}{2} \left\| \vec{w} \right\|_2 \quad (10.19)$$

The above equation is a form of Tikhonov regularization [34-35] that has been explained in detail by Cherkassky and Mulier [17] in the context of empirical versus structural risk minimization. Minimizing the norm of the weight vector is in a sense similar to an error penalization for prediction models with a large number of free parameters. An obvious question in this context relates to the proper choice for the regularization parameter or ridge parameter λ.

In the machine learning, it is common to tune the hyper-parameter λ by making use of a tuning/validation set. This tuning procedure can be quite time consuming for large datasets, especially in consideration that a simultaneous tuning for the RBF kernel width must proceed in a similar manner. We therefore propose a heuristic formula for the proper choice for the ridge parameter, that has proven to be close to optimal in numerous practical cases [36]. If the data were originally Mahalanobis scaled, it was found by scaling experiments that a near optimal choice for λ is

$$\lambda = \min\left\{1; \quad 0.05\left(\frac{n}{200}\right)^{\frac{3}{2}}\right\} \quad (10.20)$$

where *n* is the number of data for the training set.

Note that in order to apply the above heuristic the data have to be Mahalanobis scaled first. Eq. (10.20) was validated on a variety of standard benchmark datasets from the UCI data repository [36], and provided results that are nearly identical to an optimally tuned λ on a tuning/validation set. In any case, the heuristic formula for λ should be an excellent starting choice for the tuning process for λ. The above formula proved to be also useful for the initial choice for the regularization parameter C of SVMs, where C is now taken as 1/λ.

10.6. CASE STUDIES

10.6.1 Case Study #1: Predicting the Binding Energy for Amino Acids

In this section, predicting the free energy for unfolding amino acids will be revisited by applying direct kernel ridge regression with a Gaussian or RBF kernel. The ridge parameter λ was chosen as 0.00083, following Eq. (10.20), and the Parzen window parameter, σ, was chosen to be unity (obtained by tuning with the leave-one-out method on the training set of 13 data). The predictions for the six amino acids are shown in Fig. 10.9. The lower values for q^2, Q^2, and RMSE show a clear improvement over the predictions in Fig. 10.5. Figure 10.10 illustrates the scatterplot for 200 bootstrap predictions on six randomly selected samples. The values of q^2 and Q^2 are now 0.366 and 0.374, compared to 0.737 and 1.233 for the corresponding values in the linear bootstrap model shown in Fig. 10.6b. The execution time for the 200 bootstraps was 13 seconds for kernel ridge regression, compared to 0.5 seconds with the simple regression model using the Analyze/Stripminer code on a 128MHz Pentium III computer. Note also that the bootstrapped values for q^2 and $Q^{2,}$ i.e., 0.366 and 0.374, are now almost equal. The similar values for $q^2$and Q^2 indicate that there is no bias in the models, and that the choices for the hyper-parameters σ and λ, are at least close to optimal.

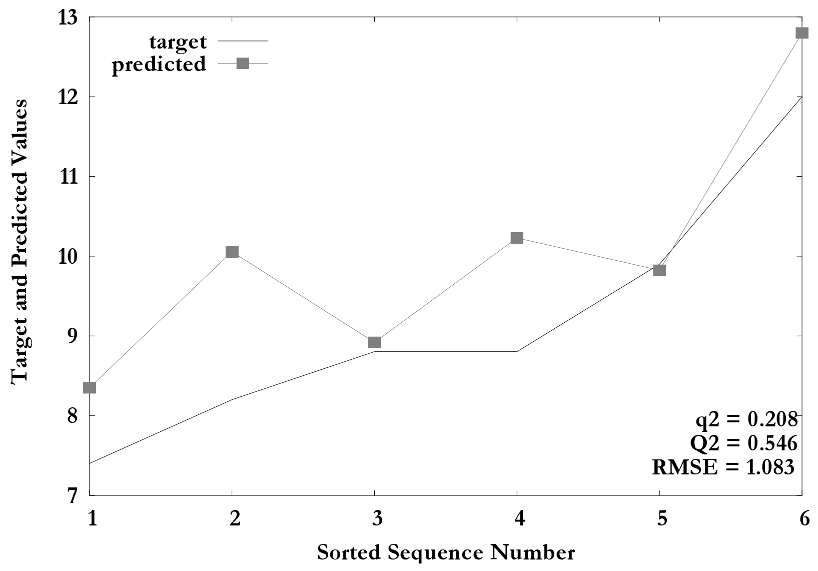


Figure 10.9 Predictions for DDGTS for the last six amino acids from Table IV with direct kernel ridge regression.

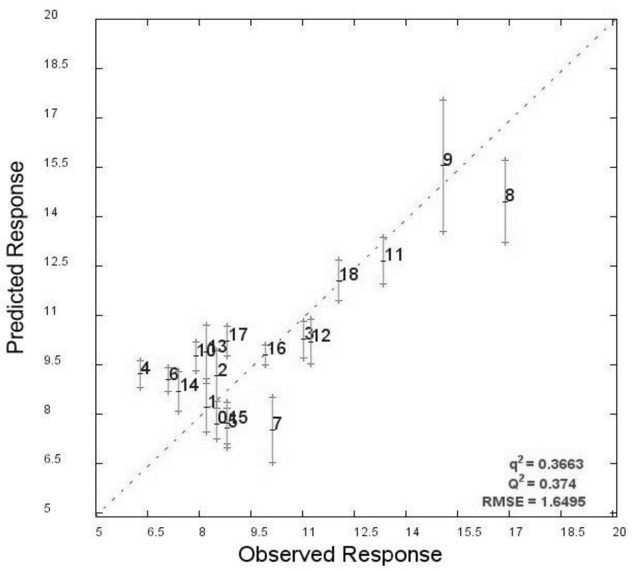
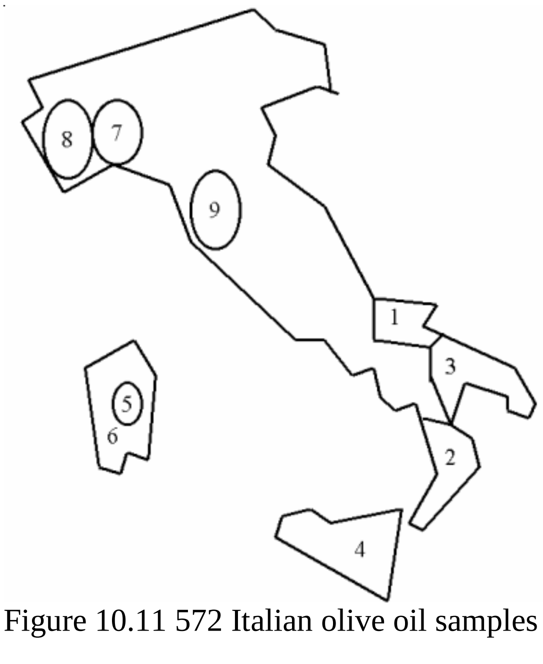
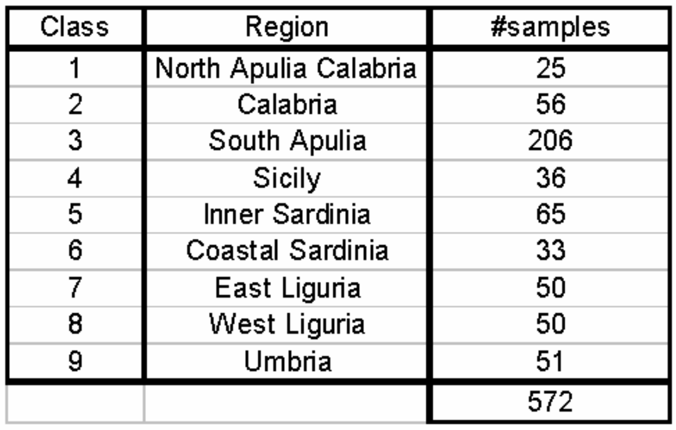


Figure 10.10 Scatterplot for predictions on six test data for DDGTS with 200 bootstraps with six sample test data each.

10.6.2 Case study #2: Predicting the Region of Origin for 572 Italian Olive Oils

The second case study deals with classifying 572 Italian olive oils by their region of origin, based on eight fatty acid contents. We chose this problem, because it is a multi-class problem with nine classes that are not ordinal. With the term non-ordinal we mean that the class numbers are not hierarchical and do not reflect a natural ordering.





Class	Region	#samples
1	North Apulia Calabria	25
2	Calabria	56
3	South Apulia	206
4	Sicily	36
5	Inner Sardinia	65
6	Coastal Sardinia	33
7	East Liguria	50
8	West Liguria	50
9	Umbria	51
		572

Figure 10.11 572 Italian olive oil samples by nine regions of origin [37-38].

The olive oil data were introduced by Forina [37] and extensively analyzed by Zupan and Gasteiger [38]. They can be downloaded from the web site referenced in [37]. Following [38], the data were split in 250 training data and 322 test data, but the split is different from the one used in [38].

The data were preprocessed as shown in Fig. 10.8. The response for the nine classes is now coded as {1 ... 9}, and Mahalanobis scaled before applying kernel ridge regression. σ is tuned on the training set, and assumed a value of 2. The ridge parameter, λ, is 0.07, based on Eq. (10.20). The errors on the test set, after de-scaling, are shown in Fig. 10.12. Figure 10.13 shows the scatterplot for the same data.

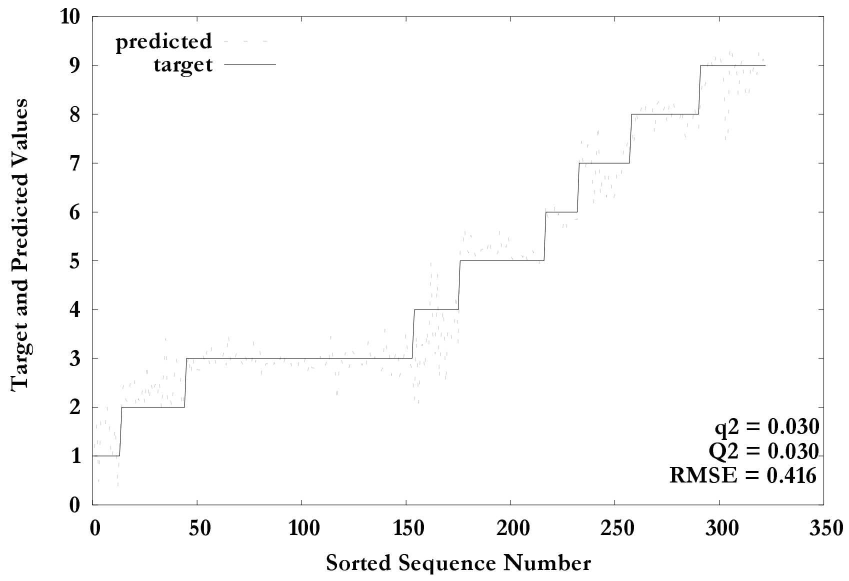


Figure 10.12 Test results for nine olive oil classes on 322 test data (kernel ridge regression).

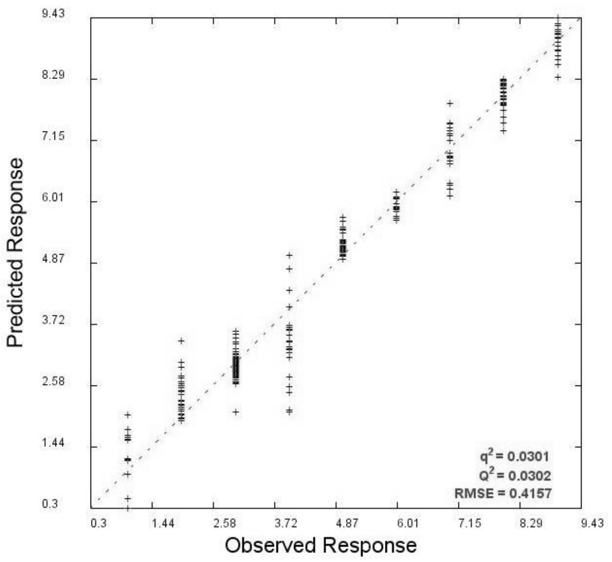


Figure 10.13 Scatterplot for nine olive oil classes on 322 test data (kernel ridge regression).

In this case, 84% of the classes were correctly predicted. This is significantly better than the 40% prediction rate with a neural net with one output neuron for the classes, as reported in [38], but also clearly below the 90% prediction rate with a neural network with nine output neurons and an orthogonal encoding reported in the same reference. In order to improve the prediction results one could either train nine different kernel ridge models, and predict for one-class versus the other eight classes at a time, or train for all possible 36 combinations on two class problems, and let a voting scheme decide on the final class predictions. While this latter scheme is definitely not as straightforward as training a single neural network with nine orthogonally encoded output neurons, the results should be more comparable to the best neural network results.

Note that there is a near perfect separation for distinguishing the olive oils from regions in southern Italy from the olive oils from the northern regions. Most of the misses in the prediction in Figs. 10.12 and 10.13, are off by just one class, locating the olive oils that were misclassified close to the actual region of origin. The single-field encoding for the output class, still works reasonably well on this non-ordinal classification problem, because the classes were labeled in an almost ordinal fashion. By this we mean that class numbers that are close to each other, e.g., 1, 2, and 3, are also geographically located nearby on the map in Fig. 10.11b.

Direct Kernel Partial Least-Squares (DK-PLS) is a direct-kernel implementation form of the PLS algorithm, popular in chemometrics, and similar to kernel-PLS first introduced in [39]. DK-PLS yields an 83% correct classification rate, using the same pre-processing procedure and 12 latent variables. In this case, latent variables are the equivalent of principal components in PCA. The traditional (linear) PLS algorithm yields a 28% correct classification rate. Direct-kernel principal component analysis with 12 principal components yields a 54% correct classification rate, while principal component analysis with six principal components results in a 30% correct classification rate. The classification results reported in this section indicate a clear improvement of direct kernel methods over their linear counterpart. The excellent comparison between kernel ridge regression and direct kernel PLS is also a good confidence indicator for heuristic formula, Eq. (10.20), for selecting the ridge parameter.

Rather than reporting improved results from combining binary classification models, we will illustrate Kohonen's self-organizing map or SOM [4]. Because self-organizing maps already inherently account for nonlinear effects, it is not necessary to apply a Direct-Kernel Self Organizing Map (DK-SOM). Fig. 10.14 shows a 13×20 Kohonen map based on 250 training data. 93% of the 322 test samples are now correctly classified as shown by the confusion matrix in Table 10.5. The classification performance of the DK-SOM is similar to the SOM, but the DK-SOM in this case requires 3 minutes training, rather than 6 seconds training for the SOM. This is not entirely surprising because the DK-SOM operates on data with 250 features or kernel entries, while the SOM operates on the eight original features.

An efficient SOM module was incorporated in the Analyze/StripMiner software [31] with a goal of keeping user decisions minimal by incorporating robust default parameter settings. To do so, the SOM is trained in its usual two-stage procedure: an ordering phase and a fine-tuning phase. The weights are trained by competitive learning where the winning neuron and its neighboring neurons in the map are iteratively updated, according to

$$\vec{w}_m^{new} = (1-\alpha)\vec{w}_m^{old} + \alpha\,\vec{x} \qquad (10.21)$$

where $\vec{x}$ is a pattern vector with m features, α is the learning parameter and $\vec{w}$ represents the weight vector.

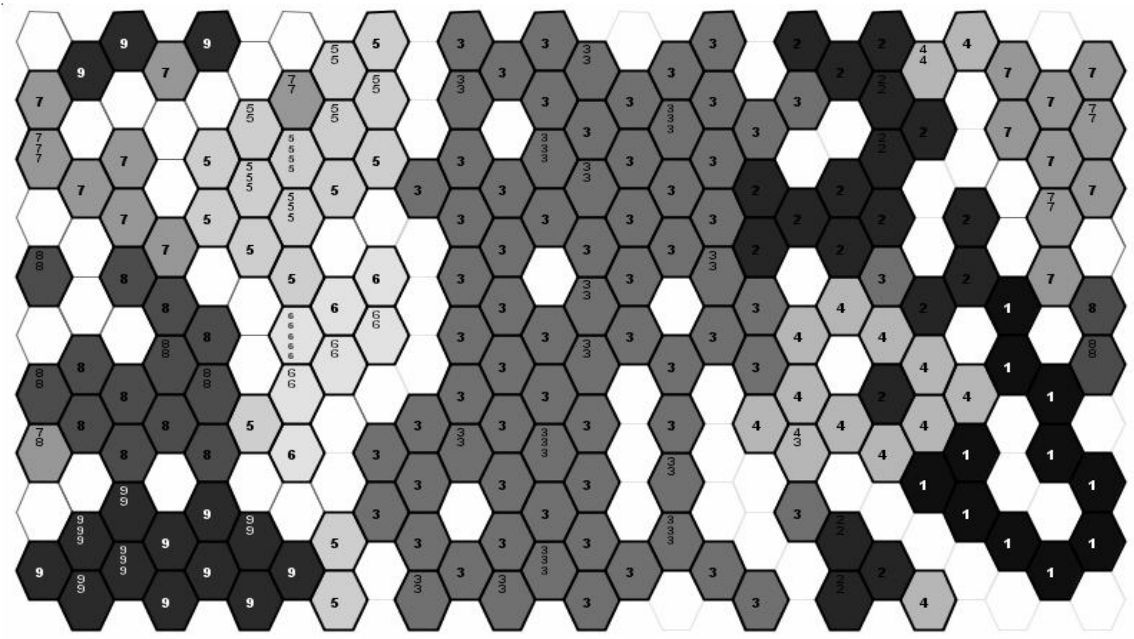


Figure 10.14 SOM for 250 training data for olive oil region of origin.

Table 10.5 Confusion matrix for 332 test data for nine non-ordinal classes.

Region										# test data
North Apulia	10	1	1	1	0	0	0	0	0	13
Calabria	0	25	6	0	0	0	0	0	0	31
South Apulia	0	1	108	0	0	0	0	0	0	109
Sicily	1	5	3	13	0	0	0	0	0	22
Inner Sardinia	0	0	0	0	40	1	0	0	0	41
Coast. Sardinia	0	0	0	0	0	16	0	0	0	16
East Liguria	0	0	0	0	1	0	23	1	0	25
West Liguria	0	0	0	0	0	0	0	33	0	33
Umbria	0	0	0	0	0	0	0	0	32	32
										322

Data patterns are presented at random and the learning parameter α is linearly reduced from 0.9 to 0.1 during the ordering phase. During this phase, the neighborhood size of the SOM is reduced from six to one on a hexagonal grid in a linear fashion. The number of iterations with Eq. (10.24) can be user-specified: 100 times the number of samples is a robust default for the ordering phase. During the fine iteration stage, α is reduced from 0.1 to 0.01 in a linear fashion. The code defaults to 20000 iterations for the ordering phase and 50000 iterations in the fine-tuning phase.

The map size defaults to a 9×18 hexagonal grid. Following Kohonen [4], the initial weights are random data samples and a supervised Learning Vector Quantization (LVQ) algorithm was implemented in the fine-tuning stage following [4]. The cells in the Kohonen map are colored by applying semi-supervised learning. By this, we mean that the weight vector is augmented by an additional entry for the class, which is called the color. The color code entry is not used in the distance metrics used in the SOM, but is otherwise updated in the same manner as the other weights in the SOM. A second optional cell-coloring scheme implemented in the code is based on a cellular automaton rule. Note also that in our study the data were preprocessed by Mahalanobis scaling each column entry first. The only difference between DK-SOM and SOM is that for DK-SOM, there is an additional pre-processing stage where the data are kernel transformed using a Gaussian kernel. The Parzen width, σ, for the default Gaussian kernel in the SOM is kept the same as for the kernel-ridge regression model.

10.6.3 Case Study #3: Predicting Ischemia from Magnetocardiography

10.6.3.1 Introduction

We describe in this section the use of direct-kernel methods and support vector machines for pattern recognition in magnetocardiography (MCG) that measures magnetic fields emitted by the electrophysiological activity of the heart. A SQUID (or Superconducting Interference Device) measures MCGs in a regular, magnetically unshielded hospital room. The operation of the system is computer-controlled and largely automated. Procedures are applied for electric/magnetic activity localization, heart current reconstruction, and derivation of diagnostic scores**.** However, the interpretation of MCG recordings remains a challenge since there are no databases available from which precise rules could be educed. Hence, there is a need to automate interpretation of MCG measurements to minimize human input for the analysis. In this particular case we are interested in detecting ischemia, which is a loss of conductivity because of damaged cell tissue in the heart and the main cause of heart attacks, the leading cause of death in the USA.

10.6.3.2 Data acquisition and pre-processing

MCG data are acquired at 36 locations above the torso for 90 seconds using a sampling rate of 1000 Hz leading to 36 individual time series. To eliminate noise components, the complete time series is low-pass filtered at 20 Hz and averaged using the maximum of the R peak of the cardiac cycle as trigger point. For automatic classification, we used data from a time window within the ST-segment [43] of the cardiac cycle in which values for 32 evenly spaced points were interpolated from the measured data. The training data consist of 73 cases that were easy to classify visually by trained experts. The testing was done on a set of 36 cases that included patients whose magnetocardiograms misled or confused trained experts doing visual classification.

We experimented with different pre-processing strategies. Data are pre-processed in this case by first subtracting the bias from each signal, each signal is then wavelet transformed by applying the Daubechies 4 wavelet transform [41]. Finally, there is a (horizontal) Mahalanobis scaling for each patient record over all the 36 signals combined. The data are then vertically Mahalanobis scaled on each attribute (except for the SOM based methods, where no further vertical scaling was applied).

10.6.3.3 Results from predictive modeling for binary classification of magnetocardiograms

The aim of this application is the automatic pattern recognition and classification for MCG data in order to separate abnormal from normal heart patterns. For unsupervised learning, we used DK-SOMs, because SOMs are often applied for novelty detection and automated clustering. The DK-SOM has a 9×18 hexagonal grid with unwrapped edges. Three kernel-based regression algorithms were used for supervised learning: support vector machines, direct kernel partial least squares (DK-PLS), and kernel ridge regression (also known as least-squares support vector machines). The Analyze/StripMiner software package, developed in-house, was used for this analysis. LibSVM was applied for the SVM model [40]. The parameter values for DK-SOM, SVM, DK-PLS and LS-SVM were tuned on the training set, before testing. The results are similar to the quality of classification achieved by the trained experts and similar for all three methods. A typical dataset for 36 signals that are interpolated to 32 equally spaced points in the analysis window [43] and after Mahalanobis scaling on each of the individual signals is shown in Fig. 10.15. The results for different methods are shown in table 10.6. Table 10.6 also indicates the number of correctly classified patterns and the number of misses on the negative and the positive cases.

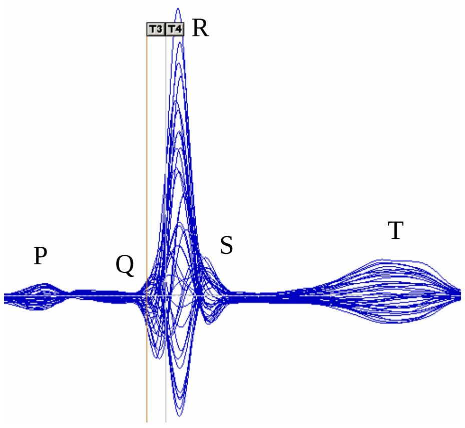


Figure 10.15 Superposition of all traces of the 36-lead MCG. The typical wave forms as seen in ECG are the P wave (P, atrial activation), the Q wave (Q,, septal activation), the R peak (R, left ventricular depolarization), the S wave (S, late right ventricular depolarization), and the T wave (T, ventricular repolarization).

Better results were generally obtained with wavelet-transformed data rather than pure time series data. For wavelet-transformed data, the Daubechies-4 or D4 wavelet transform [41] was chosen, because of the relatively small set of data (32) in each of the interpolated time signals. The agreement between K-PLS as proposed by Rosipal [40], direct kernel PLS or DK-PLS, SVMLib, and LS-SVM is generally excellent, and there are no noticeable differences between these methods on these data. In this case, DK-PLS gave a superior performance, but the differences between kernel-based methods are usually insignificant. After tuning, σ was chosen as 10, λ was determined from Eq. (10.20), and the regularization parameter, C, in SVMLib was set as 1/ λ as suggested in [39].

Table 10.6 RMSE, q2 and Q2, # of correct patterns and # of misses and execution time (on negative and positive cases on 36 test data) for magnetocardiogram data.

Method	Domain	q2	Q2	RMSE	%correct	#misses	time (s)	comment
SVMLib	time	0.767	0.842	0.852	74	4+5	10	lambda = 0.011, sigma = 10
K-PLS	time	0.779	0.849	0.856	74	4+5	6	5 latent variables, sgma = 10
DK-PCA	D4-wavelet	0.783	0.812	0.87	71	7+3	5	5 principal components
PLS	D4-wavelet	0.841	0.142	1.146	63	2+11	3	5 latent variables
K-PLS	D4-wavelet	0.591	0.694	0.773	80	2+5	6	5 latent variables, sigma = 10
DK-PLS	**D4-wavelet**	**0.554**	**0.662**	**0.75**	**83**	**1+5**	5	5 latent variables, sigma = 10
SVMLib	D4-wavelet	0.591	0.697	0.775	80	2+5	10	lambda = 0.011, sigma = 10
LS-SVM	D4-wavelet	0.59	0.692	0.772	80	2+5	**0.5**	lambda = 0.011, sigma = 10
SOM	D4-wavelet	0.866	1.304	1.06	63	3+10	960	9x18 hexagonal grid
DK-SOM	D4-wavelet	0.855	1.0113	0.934	71	5+5	28	9x18 hex grid, sigma = 10
DK-SOM	D4-wavelet	0.755	0.859	0.861	77	3+5	28	18x18 hexagonal, sigma = 8

The excellent agreement between the direct kernel methods (DK-PLS and LS-SVM) and the traditional kernel methods (K-PLS and SVMLib) shows the robustness of the direct kernel methods and also indicates that Eq. (10.20) results in a near-optimal choice for the ridge parameter.

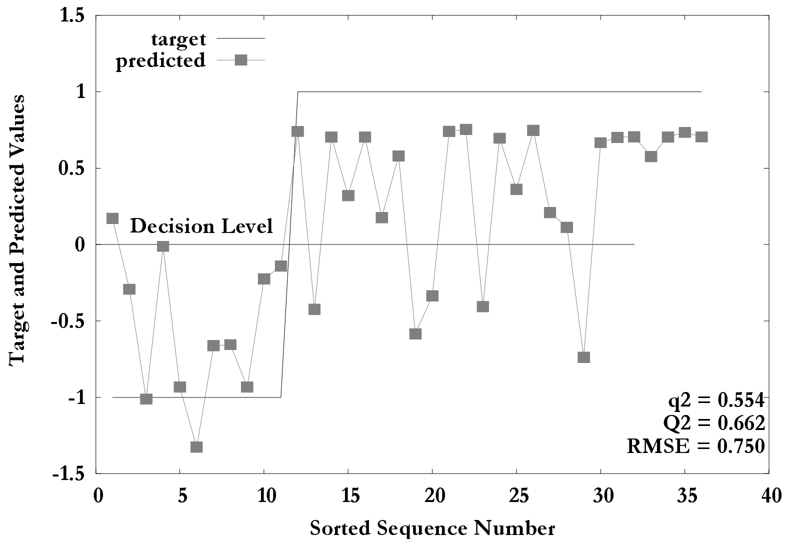


Figure 10.16 Error plot for 35 test cases, based on K-PLS for wavelet-transformed magnetocardiograms.

Not only does Eq. (10.20) apply to the selection of the ridge parameter, but also to selecting the regularization parameter, C, in support vector machines, when C is taken as 1/ λ. Linear methods such as partial-least squares result in an inferior predictive model as compared to the kernel methods. For K-PLS and DK-PLS we chose 5 latent variables, but the results were not critically dependent on the exact choice of the number of latent variables. We also tried Direct Kernel Principal Component Analysis (DK-PCA), the direct kernel version of K-PCA [3, 21-23], but the results were more sensitive to the choice for the number of principal components and not as good as for the other direct kernel methods.

Typical prediction results for the magnetocardiogram data based on wavelet transformed data and DK-PLS are shown in Fig. 10.16. We can see from this figure, that the predictions miss 6/36 test cases (1 healthy or negative case, and 5 ischemia cases). The missed cases were also difficult for the trained expert to identify, based on a 2-D visual display of the time-varying magnetic field, obtained by proprietary methods.

For medical data, it is often important to be able to make a trade-off between false negative and false-positive cases, or between sensitivity and specificity (which are different metrics related to false positives and false negatives). In machine-learning methods such a trade-off can easily be accomplished by changing the threshold for interpreting the classification, i.e., in Fig. 10.16, rather than using the zero as the discrimination level, one could shift the discrimination threshold towards a more desirable level, hereby influencing the false positive/false negative ratio. A summary of all possible outcomes can be displayed in an ROC curve as shown in Fig. 10.17 for the above case. The concept of ROC curves (or Receiver Operator Characteristics) originated from the early development of the radar in the 1940's for identifying airplanes and is summarized in [42].

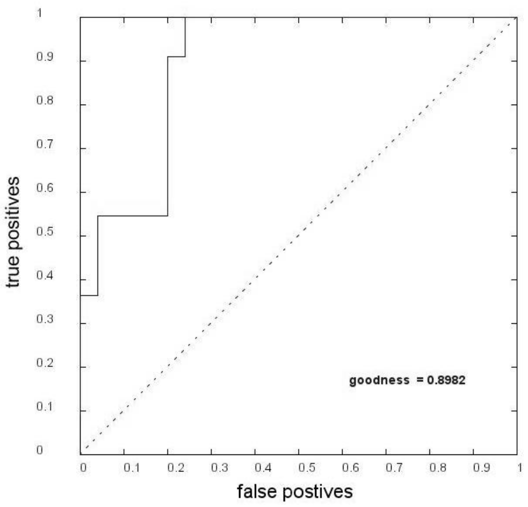


Fig. 10.17 ROC curve showing possible trade-offs between false positive and false negatives.

Figure 10.18 displays a projection of 73 training data, based on (a) Direct Kernel Principal Component Analysis (DK-PCA), and (b) Direct Kernel PLS (DK-PLS). Diseased cases are shown as filled circles. Figure 10.18b shows a clearer separation and wider margin between different classes, based on the first two components for DK-PLS as compared to DK-PCA in figure 10.18a.

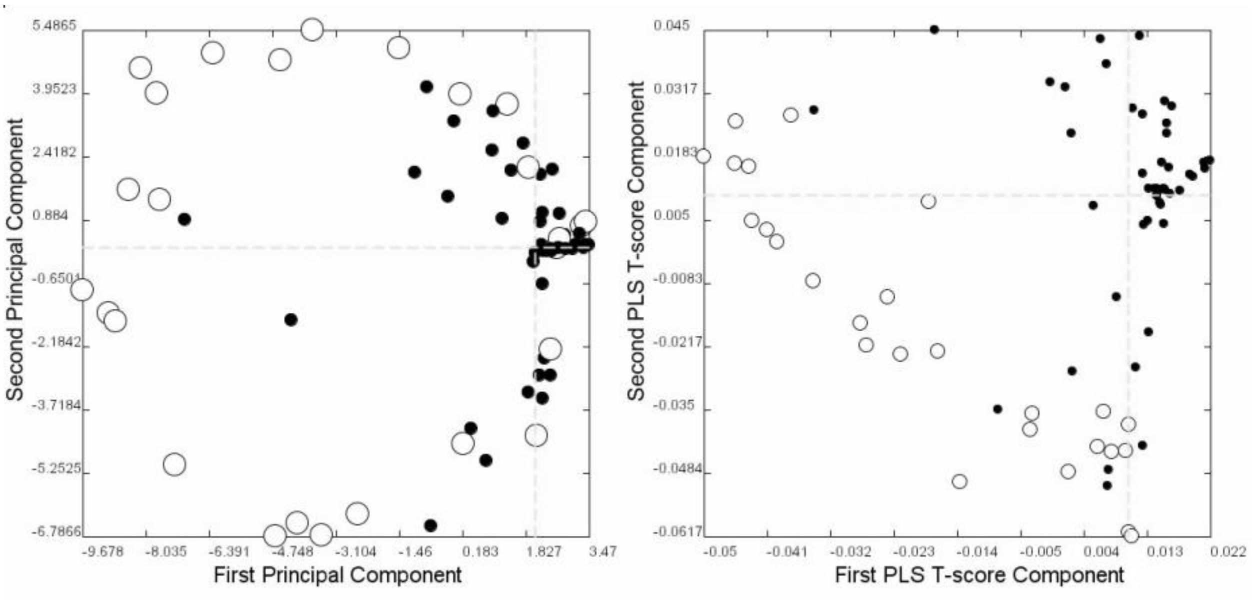


Figure 10.18 Projection of 73 training data, based on (a) Direct Kernel Principal Component Analysis (DK-PCA), and (b) Direct Kernel PLS (DK-PLS). Diseased cases are shown as filled circles. The test data are not shown on these plots.

A typical 9×18 self-organizing map on a hexagonal grid in wrap-around mode, based on the direct kernel SOM, is shown in Figure 10.19. The wrap-around mode means that the left and right boundaries (and also the top and bottom boundaries) flow into each other, and that the map is an unfolding of a toroidal projection. The dark hexagons indicate diseased cases, while the light hexagons indicate healthy cases. Fully colored hexagons indicate the positions for the training data, while the white and dark-shaded numbers are the pattern identifiers for healthy and diseased test cases. Most misclassifications actually occur on boundary regions in the map. The cells in the map are colored by semi-supervised learning, i.e., each data vector, containing 36x32 or 1152 features are augmented by an additional field that indicates the color. The color entry in the data vectors are updated in a similar way as for the weight vectors, as indicated by Eq. (10.21), but are not used to calculate the distance metrics for determining the winning cell. The resulting map for a regular SOM implementation is very similar to the corresponding map based on direct kernel DK-SOM. The execution time for generating DK-SOM on a 128 MHz Pentium III computer was 28 seconds, rather than 960 seconds required for generating the regular SOM. The kernel transformation caused this significant speedup, because the data dimensionality dropped from the original 1152 descriptive features to 73 after the kernel transformation. The fine-tuning stage for the SOM and DK-SOM was done in a supervised mode with learning vector quantization [4], following Kohonen’s suggestion for obtaining better classification results. While the results based on SOM and DK-SOM are still excellent, they are not as good as those obtained with the other kernel-based methods (SVMLib, LS-SVM, and K-PLS).

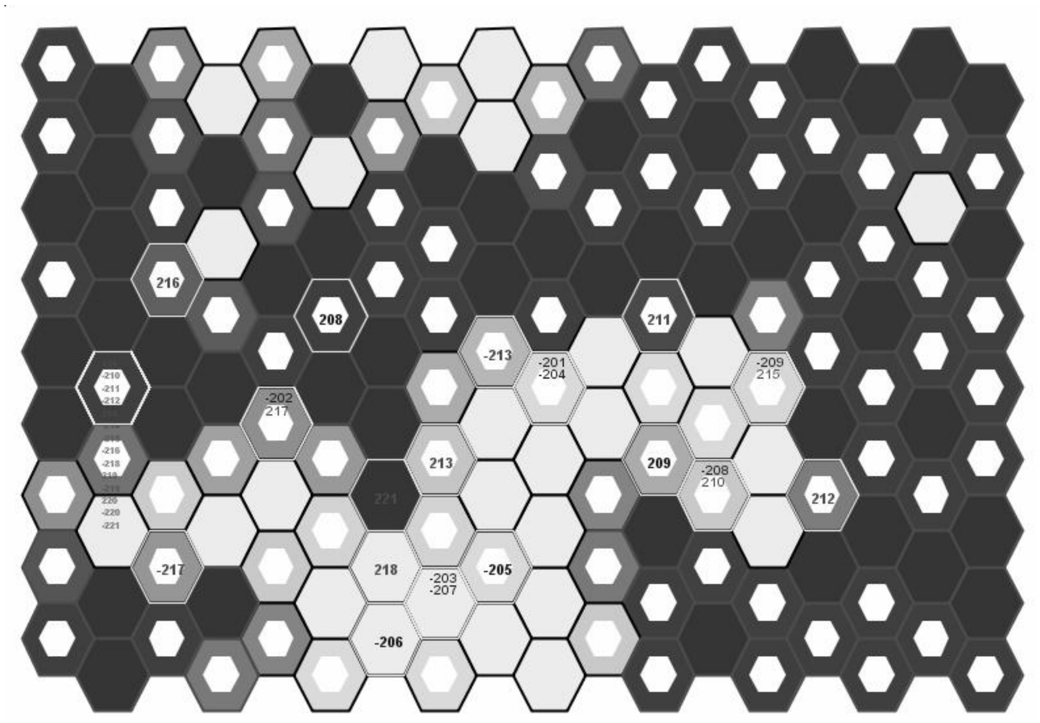


Figure 19. Test data displayed on the self-organizing map based on a 9×18 DK-SOM in wrap-around mode. Light colored cells indicate healthy cases, and dark colored cells diseased cases. Patient IDs for the test cases are displayed as well.

10.6.3.4 Feature selection

The results in the previous section were obtained using all 1152 (36×32) descriptors. It would be most informative to the domain expert if we were able to identify, where exactly in the time or wavelet signals, and for which of the 36 magnetocardiogram signals that were measured at different positions for each patient, the most important information necessary for good binary classification is located. Such information can be derived from feature selection.

Feature selection, i.e., the identification of the most important input parameters for the data vector, can proceed in two different ways: the filtering mode and the wrap-around mode. In the filtering mode, features are eliminated based on a prescribed, and generally unsupervised procedure. An example of such a procedure could be the elimination of descriptor columns that contain 4-σ outliers, as is often the case in PLS applications for chemometrics. Depending on the modeling method, it is often common practice to drop the cousin descriptors (descriptors that show more than 95% correlation between each other) and only retain the descriptors that (i) either show the highest correlation with the response variable, or (ii) have the clearest domain transparency to the domain expert for explaining the model.

The second mode of feature selection is based on the wrap-around mode. It is the aim to retain the most relevant features necessary to have a good predictive model. Often the modeling quality improves with a good feature subset selection. Determining the right subset of features can proceed based on different concepts. The particular choice for the features subset often depends on the modeling method. Feature selection in a wrap-around mode generally proceeds by using a training set and a validation set. In this case, the validation set is used to confirm that the model is not over-trained by selecting a spurious set of descriptors.

Two generally applicable methods for feature selections are based on the use of genetic algorithms and sensitivity analysis. The idea with the genetic algorithm approach is to be able to obtain an optimal subset of features from the training set, showing a good performance on the validation set as well. The concept of sensitivity analysis [12] exploits the saliency of features, i.e., once a predictive model has been built, the model is used for the average value of each descriptor, and the descriptors are tweaked, one-at-a time between a minimum and maximum value. The sensitivity for a descriptor is the change in predicted response. The premise is that when the sensitivity for a descriptor is low, it is probably not an essential descriptor for making a good model. A few of the least sensitive features can be dropped during one iteration step, and the procedure of sensitivity analysis is repeated many times until a near optimal set of features is retained. Both the genetic algorithm approach and the sensitivity analysis approach are true soft computing methods and require quite a few heuristics and experience. The advantage of both approaches here is that the genetic algorithm and sensitivity approach are general methods that do not depend on the specific modeling method.

10.7 FUSION OF SOFT AND HARD COMPUTING

In this chapter the fusion of soft and hard computing occurred on several levels. On the one hand, scientific data mining applications operate on data that are generated based on extensive and computationally intense algorithms. An example of the hard computing algorithms, are the extensive filtering and pre-processing algorithms in the case of the heart disease example. Other examples of hard computing in scientific data mining occur when the purpose of the soft computing model is to mimic a more traditional computationally demanding hard computing problem.

In this chapter the fusion of soft and hard computing occurs on a different level as well. The direct-kernel modeling methods highlighted in this chapter are in essence neural networks, a soft computing method. On the other hand, the kernel transform itself, and the way how the weights of support vectors machines, kernel ridge regression, and kernel-PLS are determined are hard computing methods. Nevertheless, the optimal choice for the hyper-parameters in these models, e.g., the kernel σ, and the λ for the regularization parameter in ridge regression, is often based on a soft computing approach. By this we mean that the model performance is rather insensitive to the exact (hard) optimal choice, and that approximate procedures for determining the hyperparameters suffice for most applications.

10.8 CONCLUSIONS

In this chapter, we provided an introduction to predictive data mining, introduced the standard data mining problem and some basic terminology, and developed direct kernel methods as a way-out of the machine learning dilemma and as a true fusion between hard and soft computing. Direct kernel methods were then applied to three different case studies for predictive data mining. In this introduction, a rather narrow view on data mining was presented. We did not address explicitly how to feed back novel and potentially useful information to the expert. While feature selection is definitely and important step to provide such meaningful feedback, the final discovery and rule formulation phase is often highly application dependent. The use of innovative visualization methods, such as the so-called pharmaplots in Fig. 10.18, and self-organizing maps is often informative and helpful for the knowledge discovery process.

ACKNOWLEDGEMENT

The authors thank Thanakorn Naenna, Robert Bress and Ramathilagam Bragaspathi for the computational support for this work. The authors also thank Profs. Kristin Bennett and Curt Breneman, for interesting and stimulating discussions during developing the framework of direct kernel methods for data mining applications. The helpful and substantial comments of Prof. Ovaska for improving the presentation are greatly appreciated.

The authors acknowledge the National Science Foundation support of this work (IIS-9979860) and SBIR Phase I #0232215.

REFERENCES

[1] Nello Cristianini and John Shawe-Taylor [2000] *Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press.
[2] Vladimir Vapnik [1998] *Statistical Learning Theory*, John Wiley & Sons.
[3] Bernhard Schölkopf and Alexander J. Smola [2002] *Learning with Kernels*, MIT Press.
[4] Teuvo Kohonen [1997] *Self-Organizing Maps, 2nd Edition*, Springer.
[5] Guido Deboeck and Teuvo Kohonen (Eds.) [1998] *Visual Explorations in Finance with Self-Organizing* Maps, Springer.
[6] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, Eds. [1996] *Advances in Knowledge Discovery and Data mining,* MIT Press.
[7] Mj. A. Berry and G. Linoff [1997] *Data Mining Techniques for Marketing, Sales, and Customer Support*, John Wiley & Sons, Inc.
[8] Joseph P. Bigus [1996] *Data Mining with Neural Networks,* McGraw-Hill.
[9] Sholom M. Weiss and Nitin Indurkhya [1998] *Predictive Data Mining: A practical Guide,* Morgan Kaufmann Publishers, Inc., San Fransisco.
[10] Zhengxin Chen [2001] *Data Mining and Uncertain Reasoning*, John Wiley & Sons, Inc.
[11] Curt M. Breneman, Kristin P. Bennett, Mark Embrechts, Steven Cramer, Minghu Song, and Jinbo Bi [2003] "Descriptor Generation, Selection and Model Building in Quantitative Structure-Property Analysis" Chapter 11 in *QSAR Developments*, James N. Crawse, Ed., John Wiley.
[12] Robert H. Kewley, and Mark J. Embrechts [2000] "Data Strip Mining for the Virtual Design of Pharmaceuticals with Neural Networks," *IEEE Transactions on Neural Networks,* Vol.11 (3), pp. 668-679.
[13] Svante Wold, Michael Sjöström, and Lennart Eriksson [2001] "PLS-Regression: a Basic Tool of Chemometrics," Chemometrics and Intelligent Laboratory Systems, Vol. 58, pp. 109-130.
[14] Alexander Golbraikh and Alexander Tropsha [2002] "Beware of q^2!" *Journal of Molecular Graphics and Modelling*, Vol 20, pp. 269-276.
[15] Richard A. Johnson and Dean W. Wichern [2000] *Applied Multivariate Statistical Analysis (second edition),* Prentice Hall
[16]Fredric M. Ham and Ivica Kostanic [2001] *Priciples of Neurocomputing for Science & Engineering,* McGraw-Hill.
[17] Vladimir Cherkassky and Filip Mulier [1998] *Learning from Data: Concepts, Theory, and Methods*, John Wiley & Sons, Inc.
[18] A. E. Hoerl, and R. W. Kennard [1970] "Ridge Regression: Biased Estimation for Non-Orthogonal Problems," *Technometrics,* Vol. 12, pp. 69-82.
[19] Simon Haykin [1999] *Neural Networks: A Comprehensive Foundation (2nd Ed.),* Prentice Hall.
[20] José Principe, Neil R. Euliano, and W. Curt Lefebre [2000] *Neural and Adaptive Systems: Fundamentals through Simulations*, John Wiley & Sons, Inc.
[21] B. Schölkopf, A. Smola, and K-R Müller [1998] "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation*, Vol. 10, 1299-1319, 1998.
[22] W. Wu, D. L. Massarat and S. de Jong [1997] "The Kernel PCA Algorithm for Wide Data. Part I: Theory and Algorithms," *Chemometrics and Intelligent Laboratory Systems*, Vol. 36, pp. 165-172.

[23] W. Wu, D. L. Massarat and S. de Jong [1997] "The Kernel PCA Algorithm for Wide Data. Part II: Fast Cross-Validation and Application in Classification of NIR Data," *Chemometrics and Intelligent Laboratory Systems*, Vol. 37, pp. 271-280.

[24] Glenn Fung and Olvi L. Mangasarian, "Proximal Support Vector Machine Classifiers," in *Proceedings KDD 2001*, San Francisco, CA.

[25] Evgeniou, T., Pontil, and M. Poggio, T. [2000] "Statistical Learning Theory: A Primer," *International Journal of Computer Vision*, Vol. 38(1), pp. 9-13.

[26] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio [2000] "Regularization Networks and Support Vector Machines," in *Advances in Large Margin Classifiers*, MIT Press.

[27] Poggio, T., and Smale S., [2003] "The Mathematics of Learning: Dealing with Data," To appear in *Notices of the AMS*, May 2003.

[28] Suykens, J. A. K. and Vandewalle, J. [1999] "Least-Squares Support Vector Machine Classifiers," *Neural Processing letters*, Vol. 9(3), pp. 293-300, Vol. 14, pp. 71-84.

[29] Suykens, J. A. K., van Gestel, T. de Brabanter, J. De Moor, M., and Vandewalle, J. [2003] *Least Squares Support Vector Machines*, World Scientific Pub Co, Singapore.

[30] Ilse C. F. Ipsen, and Carl D. Meyer [1998] "The Idea behind Krylov Methods," *American Mathematical Monthly*, Vol. 105, 889-899.

[31] The Analyze/StripMiner code is available on request for academic use, or can be downloaded from www.drugmining.com.

[32] Møller, M. F., [1993] "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning," *Neural Networks*, Vol. 6, pp.525-534.

[33] Keerthi, S. S., and Shevade S. K. [2003] "SMO Algorithm for Least Squares SVM Formulations," *Neural Computation*, Vol. 15, pp. 487-507.

[34] A. N. Tikhonov [1963] "On Solving Incorrectly Posed Problems and Method of Regularization," *Doklady Akademii Nauk USSR*, Vol. 151, pp. 501-504.

[35] A. N. Tikhonov and V. Y. Arsenin [1977] *Solutions of ill-Posed Problems*, W.H. Winston, Washinton D.C.

[36] Bennett, K. P., and Embrechts, M. J. [2003] "*An Optimization Perspective on Kernel Partial Least Squares Regression,*" Chapter 11 in *Advances in Learning Theory: Methods, Models and* Applications, Suykens J.A.K. et al., Eds., NATO-ASI Series in Computer and System Sciences, IOS Press, Amsterdam, The Netherlands.

[37] M. Forina and C. Armanino [1981] "Eigenvector Projection and Simplified Non-Linear Mapping of Fatty Acid Content of Italian Olive Oils," *Ann. Chim. (Rome)* Vol. 72, pp. 127-155.

[38] Jure Zupan and Johann Gasteiger [1999] *Neural Networks in Chemistry and Drug Design*, Wiley-CH.

[39] Rosipal R., and Trejo, L. J. [2001] "Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Spaces," *Journal of Machine Learning Research*, Vol. 2, pp. 97-128.

[40] Chih-Chung Chang and Chih-Jen Lin [2001] *LIBSVM: A library for Support Vector Machines*, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[41] Daubechies, I. [1992], *Ten Lectures on Wavelets*, Siam, Philadelphia, PA.

[42] J. A. Swets, R. M. Dawes, and J. Monahan [2000] "Better Decisions through Science," *Scientific American*, pp. 82-87.

Froelicher, V., Shetler, K., and Ashley, E. [2002] "Better Decisions through Science: Exercise Testing Scores." *Progress in Cardiovascular Diseases*, Vol. 44(5), pp. 385-414.

[43] M.S. Thaler [1999] *The only ECG Book You'll Ever Need*, Lippincott Williams & Wilkins, third edition, 1999.