The Application of Deep Kernel Machines to Various Types of Data

by

Xiaohui Wang

A research paper
presented to the University of Waterloo
in partial fulfillment of the
requirement for the degree of
Master of Mathematics
in
Computational Mathematics

Supervisor: Prof. Mu Zhu

Waterloo, Ontario, Canada, 2011

Abstract

Typically, kernel machines are linear classifiers in the implicit feature space. We argue that linear classification in the kernel's implicit feature space may sometimes be inadequate, especially in situations where the choices of kernel functions are limited. When this is the case, one naturally considers nonlinear classifiers in the feature space. We show that repeating this process produces something we call deep kernel machines. We apply this new algorithm to a various of data with different types of kernels. Results show that these deep kernel machines can make a tangible difference in classification performance.

Acknowledgements

I would like to thank my supervisor, Professor Mu Zhu, for his help and guidance with this research project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the last ten years, much attention has been paid to kernel machines and a number of powerful kernel based learning machines have been proposed [1], such as Support Vector Machines (SVMs) (Vapnik, 1995) [10], kernel Fisher discriminant (kFD) [4], kernel Principal Component Analysis (kPCA) (Scholkopf,et al. 1998) [11], Gaussian Process Classifier (Neal, 1998) [12], and so on.

However, kernel machines are linear classifiers in the implicit feature space which may be not sufficient in certain situations, especially when the choices of kernel functions are limited, for example, when dealing with more "exotic" types of data such as networks and texts. As deep networks, such as neural nets with many hidden layers [7], attract more and more amount of attention in the machine-learning community [5] - [8], Cho and Saul described a type of kernel machines [9] that "mimics" these deep networks.

In this research project, we present a novel kernel machine called Deep Kernel Machines which considers nonlinear classifiers in the feature space. We developed these composite (or deep) kernel functions while working with node classification problems on graphs. But, as can be expected, our motivation was rather different from Cho and Saul. We think our motivation is a lot more direct, and provides a valuable, alternative point of view for deep learning with kernels. In fact, we do not derive our motivation externally from attempting to mimic something else (namely multi-layer networks); instead, our motivation comes from internal considerations having to do with kernel machines themselves.

Kernel machines have been successfully applied to various areas, such as time-series prediction (Muller et al., 1997) [3], text categorization (Joachims, 1998) [13], prediction of protein-protein interactions (Huel et al., 2010) [22], etc. In the "Application" part of our research project, we will explain different types of kernels briefly and apply our new algorithm to some real-world data with adequate kernels.

The paper is organized as follows. In Chapter 2, we provide some background knowledge on kernel machines and introduce our novel algorithm - Deep Kernel Machines. In Chapter 3, we present some empirical evidence using a variety of types of kernels and data to show that DKMs are useful. In Chapter 4, we end with a short summary and some further remarks.

Chapter 2

Deep Kernel Machines

2.1 A Simple Kernel Machine

2.1.1 A General Form

For convenience, we simply discuss the two-class problems throughout the whole article. But it does not mean that our discussion is only limited to binary classification problems.

Suppose that there is a training set $X = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\} \subseteq \Re^d$. Associated with each vector in the training set, there is a label $y_i \in \{1, 2\}$, $i = 1, 2, ..., n$. Typically, a kernel machine has the form,

$$f(\mathbf{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \boldsymbol{K}(\mathbf{x}, \mathbf{x_i}) \tag{2.1}$$

for each vector $\mathbf{x}$ in the test set Z. Here, $\boldsymbol{K}(\cdot, \cdot)$ is a kernel function and the coefficient α_i often depends on the class labels.

2.1.2 A Simple Kernel Machine

In order to crystalize the gist of our ideas, we mostly work with a very simple kernel machine in this section. But when doing the experiments, we use the Support Vector Machines.

To classify $\mathbf{x_0} \in Z$, we can use the function,

$$f_X(\mathbf{x_0}) \equiv \frac{1}{n_1} \sum_{\mathbf{x}_i : y_i = 1} \boldsymbol{K}_X(\mathbf{x_0}, \mathbf{x_i}) - \frac{1}{n_2} \sum_{\mathbf{x}_i : y_i = 2} \boldsymbol{K}_X(\mathbf{x_0}, \mathbf{x_i}) \tag{2.2}$$

where n_1 and n_2 are total number of vectors in X with class label 1 and 2, respectively. For $\mathbf{x_0}$, this is simply the difference between its average similarity to class 1 and its average similarity to class 2. For example, we can classify $\mathbf{x_0}$ to the class which it is more similar to, i.e.,

$$\hat{y}_0 = \begin{cases} 1, & \text{if } f_X(\mathbf{x_0}) > c; \\ 2, & \text{otherwise.} \end{cases} \tag{2.3}$$

for some thresholding constant c.

It is easy to see that the function $f_X(\mathbf{x})$ in (2.2) is of the general form of (2.1), with $\alpha_0 = 0$, $\alpha_i = \frac{1}{n_1}$ or $-\frac{1}{n_2}$ depending on whether $y_i = 1$ or 2, and $\boldsymbol{K} = \boldsymbol{K}_X$. In other words, (2.2) is a simple kernel machine. This simple kernel machine can be constructed without invoking expensive optimization procedures in order to determine the coefficients, $\alpha_0, \alpha_1, \alpha_2, ..., \alpha_{n_1+n_2}$. A more sophisticated kernel machine such as the SVM requires quadratic programming to find these coefficients.

2.2 Deep Kernel Machines

2.2.1 The implicit feature space $\mathcal{F}$

A key idea behind kernel machines is that kernels can be regarded as calculating inner products in an implicit feature space, call it $\mathcal{F}$. That is,

$$\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}) = \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle, \quad \text{where} \quad \phi : X \mapsto \mathcal{F}. \tag{2.4}$$

We will write $\phi(\mathbf{x_i})$ and ϕ_i, $\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j})$ and $\boldsymbol{K}_X(i, j)$ interchangeably. This means the kernel $\boldsymbol{K}_X$ necessarily induces a distance function in $\mathcal{F}$,

$$\begin{aligned} & d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) \\ =& \|\phi(\mathbf{x_i}) - \phi(\mathbf{x_j})\| \\ =& \sqrt{\langle \phi_i, \phi_i \rangle - 2\langle \phi_i, \phi_j \rangle + \langle \phi_j, \phi_j \rangle} \\ =& \sqrt{\boldsymbol{K}_X(i, i) - 2\boldsymbol{K}_X(i, j) + \boldsymbol{K}_X(j, j)}. \end{aligned} \tag{2.5}$$

2.2.2 Linear classification in $\mathcal{F}$

Using the distance $d_{\mathcal{F}}$ — more specifically the squared distance $d^2_{\mathcal{F}}$, the decision rule (2.3) is equivalent to nearest-centroid classification in the feature space $\mathcal{F}$. To see this, notice that $\frac{1}{n_1}\sum \phi(\mathbf{x_i})$ and $\frac{1}{n_2}\sum \phi(\mathbf{x_i})$ are class centroids in $\mathcal{F}$. Nearest-centroid classification simply declares $\hat{y}_0 = 1$ if $\mathbf{x_0}$ is closer to the centroid of class 1, i.e., if

$$\left\| \phi(\mathbf{x_0}) - \frac{1}{n_1}\sum_{y_i=1} \phi(\mathbf{x_i}) \right\|^2 < \left\| \phi(\mathbf{x_0}) - \frac{1}{n_2}\sum_{y_i=2} \phi(\mathbf{x_i}) \right\|^2 . \tag{2.6}$$

The left-hand side is equivalent to

$$\langle \phi(\mathbf{x_0}), \phi(\mathbf{x_0}) \rangle - \frac{2}{n_1}\sum_{y_i=1} \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_0}) \rangle + \frac{1}{n_1^2}\sum_{y_i,y_j=1} \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_i}) \rangle \tag{2.7}$$

and similarly for the right-hand side:

$$\langle \phi(\mathbf{x_0}), \phi(\mathbf{x_0}) \rangle - \frac{2}{n_2}\sum_{y_i=2} \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_0}) \rangle + \frac{1}{n_2^2}\sum_{y_i,y_j=2} \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_i}) \rangle \tag{2.8}$$

Canceling out $\langle \phi(\mathbf{x_0}), \phi(\mathbf{x_0}) \rangle$ and dividing by -2, we obtain

$$\frac{1}{n_1}\sum_{y_i=1} \boldsymbol{K}_X\left(\mathbf{x_0}, \mathbf{x_i}\right) - \frac{1}{n_2}\sum_{y_i=2} \boldsymbol{K}_X\left(\mathbf{x_0}, \mathbf{x_i}\right) - c(X) > 0, \tag{2.9}$$

where

$$c(X) \equiv \frac{1}{2n_1^2}\sum_{y_i,y_j=1} \boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}) - \frac{1}{2n_2^2}\sum_{y_i,y_j=2} \boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}) \tag{2.10}$$

is a constant that depends only on X and not on $\mathbf{x_0}$. Clearly, this is equivalent to (2.3). Being equivalent to nearest-centroid classification, the kernel machine (2.2) is therefore a linear classifier in the feature space $\mathcal{F}$, just like all other kernel machines, including SVMs.

2.2.3 Nonlinear classification in $\mathcal{F}$

However, it is quite possible that a linear classifier in $\mathcal{F}$ is not sufficient. Being able to rely on a simple, linear classifier means the feature space $\mathcal{F}$ is, in some sense, “optimal”. But, in practice, there is no way to guarantee that this is the case, especially when the choices of kernel functions are limited, e.g., the node classification problem.

On the other hand, there is nothing in principle that prevents us from using other, more flexible classifiers in $\mathcal{F}$. For example, using the distance $d_{\mathcal{F}}$, we may consider a classifier based on kernel density estimates. Let

$$\hat{p}_k(\mathbf{x}) = \frac{1}{n_k} \sum_{\substack{\mathbf{x_i} \in X \\ y_i = k}} K_{h(\mathcal{F})} \left(d_{\mathcal{F}}(\mathbf{x}, \mathbf{x_i}) \right) \tag{2.11}$$

be a kernel density estimate of the distribution for class k. Many kernel functions can be used for density estimation, e.g.,

$$K_{h(\mathcal{F})}(d_{\mathcal{F}}) = \frac{1}{\sqrt{2\pi} h(\mathcal{F})} exp \left(- \frac{d_{\mathcal{F}}^2}{2h^2(\mathcal{F})} \right), \tag{2.12}$$

where $h(\mathcal{F})$ is a bandwidth parameter, which serves to scale the distance $d_{\mathcal{F}}$. We shall write

$$\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) \equiv K_{h(\mathcal{F})} \left(d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) \right). \tag{2.13}$$

Using (2.13) for $K_{h(\mathcal{F})}$, $\boldsymbol{K}_{\mathcal{F}}$ is nothing but the well-known radial-basis or Gaussian kernel, except it uses the distance function $d_{\mathcal{F}}$ rather than a distance defined on the original training set X. Therefore, (2.11) is a density estimate in the space $\mathcal{F}$ rather than on the original training set X. The subscript $\mathcal{F}$ and the notation $h(\mathcal{F})$ are used to emphasize this fact and to differentiate $\boldsymbol{K}_{\mathcal{F}}$ from $\boldsymbol{K}_X$, the kernel on the original training set X that induced the space $\mathcal{F}$.

Based on the kernel density estimates in (2.11), for each $\mathbf{x_0}$ in test set, we can predict its class label $\hat{y}_0$ depending on whether $\hat{p}_1(\mathbf{x_0}) - \hat{p}_2(\mathbf{x_0})$ is positive or negative. In other words, the decision function is simply

$$f_{\mathcal{F}}(\mathbf{x_0}) = \frac{1}{n_1} \sum_{\substack{\mathbf{x_i} \in X \\ y_i = 1}} \boldsymbol{K}_{\mathcal{F}} \left(\mathbf{x_0}, \mathbf{x_i} \right) - \frac{1}{n_2} \sum_{\substack{\mathbf{x_i} \in X \\ y_i = 2}} \boldsymbol{K}_{\mathcal{F}} \left(\mathbf{x_0}, \mathbf{x_i} \right). \tag{2.14}$$

It is easy to see that (2.14) is another kernel machine of the same form as (2.2). The only difference is that (2.14) uses the kernel $\boldsymbol{K}_{\mathcal{F}}$ whereas (2.2) uses the kernel $\boldsymbol{K}_X$.

2.2.4 Deep Kernel Machines

Let us summarize what we have said so far. The space $\mathcal{F}$ is the implicit feature space for $\boldsymbol{K}_X$. A kernel machine f_X (2.2) using the kernel $\boldsymbol{K}_X$ is a linear classifier in $\mathcal{F}$. If linear classifiers are not sufficient in $\mathcal{F}$, we can relax linearity and choose to work with a nonlinear classifier, e.g., by constructing kernel density estimates in $\mathcal{F}$ via the implied distance metric $d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j})$ — equation (2.5). This gives rise to a new kernel machine $f_{\mathcal{F}}$ (2.14), using the kernel $\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j})$ — equation (2.13). If we use (2.12) for kernel density estimation, the kernel updating formula from $\boldsymbol{K}_X$ to $\boldsymbol{K}_{\mathcal{F}}$ is simply, putting (2.5), (2.12), and (2.13) together,

$$\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) = \frac{1}{\sqrt{2\pi}h(\mathcal{F})} \times exp\left[-\frac{\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_i}) - 2\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}) + \boldsymbol{K}_X(\mathbf{x_j}, \mathbf{x_j})}{2h(\mathcal{F})^2}\right]. \tag{2.15}$$

The choice of $h(\mathcal{F})$ will be discussed in next section.

However, there is no reason why the process must end here. The kernel $\boldsymbol{K}_{\mathcal{F}}$ has its implicit feature space as well; let's call it $\mathcal{F}^2$. The kernel machine $f_{\mathcal{F}}$ (2.14) using the kernel $\boldsymbol{K}_{\mathcal{F}}$ is a linear classifier in $\mathcal{F}^2$. We can relax linearity in $\mathcal{F}^2$, if necessary, and choose to work with a nonlinear classifier, again, by constructing kernel density estimates in $\mathcal{F}^2$ via the implied distance metric,

$$d_{\mathcal{F}^2}(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\boldsymbol{K}_{\mathcal{F}}(i,i) - 2\boldsymbol{K}_{\mathcal{F}}(i,j) + \boldsymbol{K}_{\mathcal{F}}(j,j)}. \tag{2.16}$$

By the same argument, this would give us yet another kernel machine, say $f_{\mathcal{F}^2}$, of exactly the same form as f_X and $f_{\mathcal{F}}$, except it would be using the kernel

$$\begin{aligned}
&\boldsymbol{K}_{\mathcal{F}^2}(\mathbf{x_i}, \mathbf{x_j}) \\
=&K_{h(\mathcal{F}^2)}\left(d_{\mathcal{F}^2}(\mathbf{x_i}, \mathbf{x_j})\right) \\
=&\frac{1}{\sqrt{2\pi}h(\mathcal{F}^2)} \times exp\left[-\frac{\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_i}) - 2\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) + \boldsymbol{K}_{\mathcal{F}}(\mathbf{x_j}, \mathbf{x_j})}{2h(\mathcal{F}^2)^2}\right].
\end{aligned} \tag{2.17}$$

It is easy to see that this process can be repeated recursively. We refer to kernel machines generated by this recursive process as **Deep Kernel Machines** (DKMs). The one using the original kernel $\boldsymbol{K}_X$ is referred to as a level-0 DKM; the one using the kernel $\boldsymbol{K}_{\mathcal{F}}$, a level-1 DKM; the one using the kernel $\boldsymbol{K}_{\mathcal{F}^2}$, a level-2 DKM; and so on.

2.2.5 A heuristic for choosing $h(\mathcal{F})$

To carry out density estimation in $\mathcal{F}$, a bandwidth parameter $h(\mathcal{F})$ must be specified. While users are certainly free to optimize this parameter in practice, this can be tedious for DKMs because, as we go from X to $\mathcal{F}, \mathcal{F}^2, \mathcal{F}^3, ...$, there is a bandwidth parameter for each space, $h(\mathcal{F}), h(\mathcal{F}^2), h(\mathcal{F}^3), ...$, so a heuristic is desired. A reasonable heuristic is:

$$h(\mathcal{F}) = \frac{1}{n^2} \sum_{\mathbf{x_i}, \mathbf{x_j} \in X} \sum d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}). \tag{2.18}$$

That is, $h(\mathcal{F})$ can be chosen to be the average pair-wise distance in the space of $\mathcal{F}$. We use this heuristic in all of our experiments below.

2.3 Support Vector Machines

Even though, in our discussions so far, we have focused only on a specific simple kernel machine, it is easy to see that the recursively defined deep kernel function (2.15) can be plugged into any kernel machine. As we will use Support Vector Machines in all of our experiments below, we briefly introduce the idea of SVMs in this section.

In a two-class classification problem, SVM seeks an optimal hyperplane to separate the two classes [14]. A hyperplane in $\Re^d$ consists of all $\mathbf{x} \in \Re^d$ that satisfy the linear equation:

$$f(\mathbf{x}) = \beta' \mathbf{x} + \beta_0 \tag{2.19}$$

A hyperplane is called a separating hyperplane if there exists $c > 0$ such that

$$y_i(\beta' \mathbf{x} + \beta_0) \geq c, \forall i = 1, 2, ..., n. \tag{2.20}$$

We can scale the hyperplane so that (2.19) becomes

$$y_i(\beta' \mathbf{x} + \beta_0) \geq 1, \forall i = 1, 2, ..., n. \tag{2.21}$$

A separating hyperplane satisfying condition (2.21) is called a canonical separating hyperplane (CSHP).

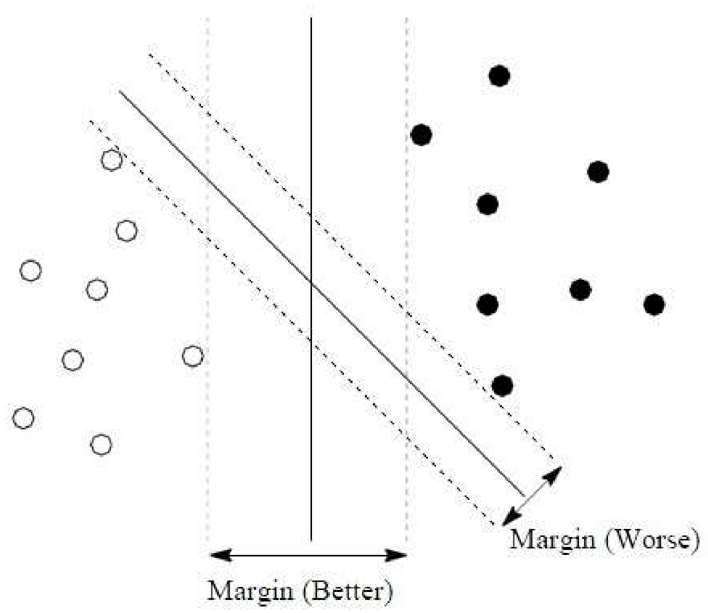

Figure 2.1: Two separating hyperplanes, one with a larger margin than the other.

If the two classes are perfectly separable, then there exist an infinite number of separating hyperplanes. Figure 1 shows two competing hyperplanes in such a situation. The SVM is based on the notion that the best canonical separating hyperplane to separate two classes is the one that is the farthest away from the training points. This notion is formalized mathematically by the margin of a hyperplane hyperplanes with larger margins are better. In particular, the margin of a hyperplane is equal to

$$margin = 2 \times \min\{y_i d_i, i = 1, 2, ..., n\} \tag{2.22}$$

where d_i is the signed distance between observation $\mathbf{x_i}$ and the hyperplane; see Figure (2.1) for an illustration. Figure 1 also shows to a certain extent why large margins are good on an intuitive level.

It can be shown that [15] d_i is equal to

$$d_i = \frac{1}{\parallel \beta \parallel}(\beta)'\mathbf{x_i} + \beta_0 \tag{2.23}$$

Then, equations (2.21) and (2.22) together imply that the margin of a CSHP is equal to

$$margin = 2 \times \min\{y_i d_i, i = 1, 2, ..., n\} = \frac{2}{\parallel \beta \parallel} \tag{2.24}$$

To find the best CSHP with the largest margin, we are interested in solving the following optimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2} \parallel \beta \parallel^2 + \gamma \sum_{i=1}^{n} \xi_i \\ subject\ to \quad & y_i(\beta' \mathbf{x} + \beta_0) \geq 1 - \xi_i, \forall i. \end{aligned} \tag{2.25}$$

where the extra variables ξ_i are introduced to relax the separability condition (2.21) because, in general, we cant assume the two classes are always perfectly separable. The term $\gamma \sum \xi_i$ acts as a penalty to control the degree of such relaxation, and γ is a tuning parameter.

The dual form of SVMs is as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x_i}' \mathbf{x_j} \\ s.t. \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad and \quad \alpha_i \geq 0, \forall i. \end{aligned} \tag{2.26}$$

The optimal β is

$$\beta = \sum_{i:\alpha_i > 0} \alpha_i y_i \mathbf{x_i} \tag{2.27}$$

The resulting hyperplane can be written as

$$f(\mathbf{x}) = \beta' \mathbf{x} + \beta_0 = \sum_{i:\alpha_i > 0} \alpha_i y_i \mathbf{x_i}' \mathbf{x} + \beta_0 \tag{2.28}$$

In order to obtain i, one solves (2.26), a problem that depends on the predictors $\mathbf{x_i}$ only through their inner-products $x_i' x_j$; once the $\alpha_i s$ are obtained, the ultimate decision function (2.28) is also just a function of inner-products in the predictor space.

Using “kernel tricks” [16], the kernel form of SVM is

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{K}_h(\mathbf{x_i}, \mathbf{x_j}) \\ s.t. \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad and \quad \alpha_i \geq 0, \forall i. \end{aligned} \tag{2.29}$$

and the decision function (2.28) becomes

$$f(\mathbf{x}) = \beta' \mathbf{x} + \beta_0 = \sum_{i:\alpha_i > 0} \alpha_i y_i \boldsymbol{K}_h(\mathbf{x_i}, \mathbf{x_j}) + \beta_0 \tag{2.30}$$

The boundary is linear in the space of $\phi(\mathbf{x})$ where $\phi(\cdot)$ is such that

$$\boldsymbol{K}_h(x; z) = \phi(\mathbf{x})' \phi(\mathbf{z}) \tag{2.31}$$

We dont even need to define the mapping $\phi(\cdot)$ explicitly; all we have to do is to pick a kernel function $\boldsymbol{K}_h(x; z)$. This makes the SVM very general.

Chapter 3

Application

In this chapter, we apply our novel algorithm Deep Kernel Machines to various data sets with different types kernel accordingly.

Table 3.1 summarizes a number of different data sets, which we used to perform empirical experiments in the following sections.

Table 3.1: Summary of data sets and classification tasks. The regular data type means the inputs are simply vectors in $\Re^d$

Name	Type of Data	$\boldsymbol{K}_X$	Class 1	Class 2
Breast	regular (10 features)	radial basis	212	357
Corel	regular (50 features)	radial basis	100	100
Digit38	image (16×16)	radial basis	658	542
Wave12	regular (21 features)	radial basis	175	152
Wave13	regular (21 features)	radial basis	175	173
Wave23	regular (21 features)	radial basis	152	173
Yeast	regular (8 features)	radial basis	457	243
Enron	network (182 nodes)	diffusion	16	166
Lawyer	network (36 notes)	diffusion	20	16
WebKB	text (indexed by 56,532 terms)	latent semantic	631	396
Protein	pairs of protein (Score matrix)	pairwise Mommoth	40	160

All experiments used the SVM as the base kernel machine, with the penalty on the sum of slack variables (often called the "cost" parameter in most SVM packages) tuned by cross validation on the training set alone. All results are averages from 25 repeated runs, each time using a random 50-50 split of the data set into training and test sets. The random splits are stratified by class label so that the fraction of data belonging to each class is roughly the same in the training and test sets.

3.1 Several Kinds of Data and RBF Kernels

RBF kernels are the most widely used kernels and have been extensively studied in neighbouring field. In this section, we introduce the RBF kernels and do some experiments with several kinds of data using this type of kernels.

3.1.1 RBF Kernels

Suppose that $\{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ is a set of data. The **Gaussian kernel** is defined as:

$$\boldsymbol{K}(\mathbf{x_i}, \mathbf{x_j}) = e^{-\frac{\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2}} \tag{3.1}$$

where $\sigma > 0$ is a parameter to be determined. Note that we are not restricted to using the Euclidean distance in the input space [17].

Those functions in (3.1) form the hidden units of a radial basis function network, and hence using this kernel will mean the hypotheses are radial basis function networks. It is therefore also referred to as the **radial basis function** (RBF) kernel.

The parameter σ controls the flexibility of the kernel. Small values of σ allow classifiers to fit any labels, hence risking overfitting. In this case, the kernel mattix becomes close to the identity matrix. On the other hand, large values of σ gradually reduce the kernel to a constant function, making it impossible to learn any non-trivial classifier.

3.1.2 Data

Totally seven experiments have been shown in this section. Their details are as follows.

Breast, Corel, and Yeast These data sets were obtained from the well-known UCI machine-learning repository, http://archive.ics.uci.edu/ml/datasets/. For the "Corel" data, we randomly downloaded 100 images of dinosaurs and another 100 images of mountains. The images were 384×256, and we used the first 50 principal components as features.

Digit38 This is a subset of the "ZIP code" data from http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html, consisting of only threes and eights.

Wave12, Wave13, and Wave23 These are subsets of the (simulated) "waveform" data from http://www-stat.stanford.edu/~tibs/ElemStatLearn/data.html. "Wave12" consists of only class 1 and class 2; "Wave13" consists of only class 1 and class 3; and so on.

3.1.3 Results

Figure (3.1) shows the test-set performance of DKMs on the data sets described aforementioned. In all cases, the tuning parameter for $\boldsymbol{K}_X$ is selected by cross validation. The horizontal axis stands for the data set and level of DKM. The vertical axis stands for the percent of possible improvement in classification accuracy over level-0, defined as $\frac{a_j - a_0}{1 - a_0}$ where a_j is the accuracy at level-j. Thus, a negative bar means the performance is worse than level-0.

Note that most bars are positive, which means that DKMs are beneficial.

3.2 Graph and Diffusion Kernels

In this section, we introduce the diffusion kernels and do some experiments with graph data using this type of kernels.

3.2.1 Diffusion Kernels

Suppose that $G = (V, E)$ is a graph, where $V = \{\mathbf{v_1}, \mathbf{v_2}, ..., \mathbf{v_n}\}$ is its set of vertices (nodes) and E is its set of edges. Associated with each node $\mathbf{v_i}$ is a class label $y_i \in \{1, 2, ..., C\}$.

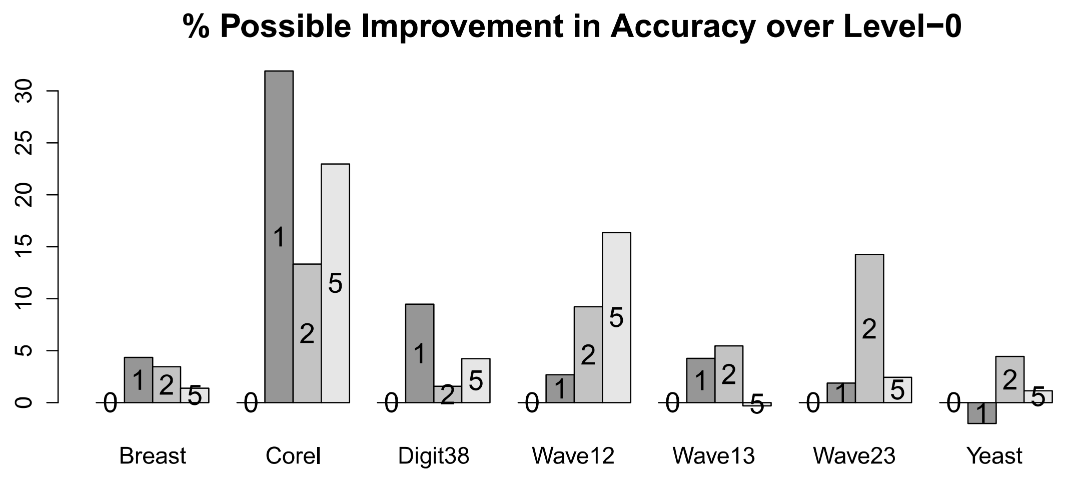

Figure 3.1: Results for the Datasets in this section

The **adjacency matrix** for graph G is defined as:

$$\mathbf{A}_G(i,j) = \begin{cases} 1, & \text{if there is an edge between } \mathbf{v_i} \text{ and } \mathbf{v_j}; \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Let

$$d_i = \sum_{j \neq i} \mathbf{A}_G(i,j) \tag{3.3}$$

be the **degree** of node. The **Graph Laplacian matrix** for G is defined by

$$\mathbf{L}_G = \mathbf{D}_G - \mathbf{A}_G \tag{3.4}$$

where $\mathbf{D}_G = diag\{d_1, d_2, ..., d_n\}$.

The **diffusion kernel** is defined as

$$\mathbf{K}_G = exp(-\beta \mathbf{L}_G) = \sum_{m=0}^{\infty} \frac{(-\beta)^m}{m!} \mathbf{L}_G^m \tag{3.5}$$

where $\beta > 0$ is a tuning parameter [29].

To measure the similarity between $\mathbf{v_i}$ and $\mathbf{v_j}$, the diffusion kernel takes into account the number of paths of length m between $\mathbf{v_i}$ and $\mathbf{v_j}$, for all m, and gives shorter paths more weight [31].

To compute the diffusion kernel, let $\mathbf{L}_G = \mathbf{U}\Sigma\mathbf{U}'$ be the spectral decomposition of the Laplacian matrix, where $\Sigma = diag(s_m)$. Using the fact that $\mathbf{L}_G^m$ has the same eigenvectors for all m, the diffusion kernel can be computed by

$$\mathbf{K}_G = \mathbf{U} diag\left(e^{-\beta s_m}\right) \mathbf{U}' \tag{3.6}$$

3.2.2 Data

Two experiments have been shown in this section. Their details are as follows.

Enron In 2001, a USA-based gas and electricity company named Enron was found guilty of serious accounting frauds. As part of the investigation, the US Federal Energy Regulatory Commission confiscated its corporate email database and made it publicly available. From the website http://cis.jhu.edu/~parky/Enron/enron.html, we obtained a 184×184 adjacency matrix, $\boldsymbol{A}_X$, that indicated whether there was email communication between any two of 184 unique email accounts, as well as the status of these 184 email account owners, e.g., CEO, employee, etc. We removed two accounts that never sent an email to another account.

Lawyer Lazega citeLazega:01 studied collaborative working relationships and social interactions in a New England law firm. Thirty-six (36) partners were interviewed and asked to express opinions about issues regarding how the law firm should be managed, such as whether the company should adopt a less flexible workflow. We obtained the data directly from Professor Lazega, and our initial diffusion kernel $\boldsymbol{K}_G$ was based on a similarity (rather than adjacency) matrix defined as

$$\mathbf{A}_G(i,j) = 0.5 \times \mathbf{I}_{friends}(i,j) + 0.5 \times \mathbf{I}_{collaborated}(i,j), \tag{3.7}$$

where $\mathbf{I}_{friends}(i,j) = 1$ if $\mathbf{v_i}$ and $\mathbf{v_j}$ were friends and 0 if not; and likewise for $\mathbf{I}_{collaborated}(i,j)$.

3.2.3 Results

Figure (3.2) and (3.3) display performance results of enron data set and lawyer data set respectively, for a range of tuning parameters. In these two data sets, the ratio of the two classes is about 1:10. In unbalanced situations like this, it is commonly held that classification accuracy (equivalently total misclassification error) is not the best measure of performance [19]. Thus, we use the area under the receiver-operating characteristic (ROC) curve [20], or simply AUC (for area under the curve) as the performance measure for this task instead.

3.3 Text and Latent Semantic Kernels

In the last decade, natural language text gradually took the place of multivariate data and became the most important data format for applications. We introduce the latent semantic kernels for text in this section and show the efficiency of DKMs to text data.

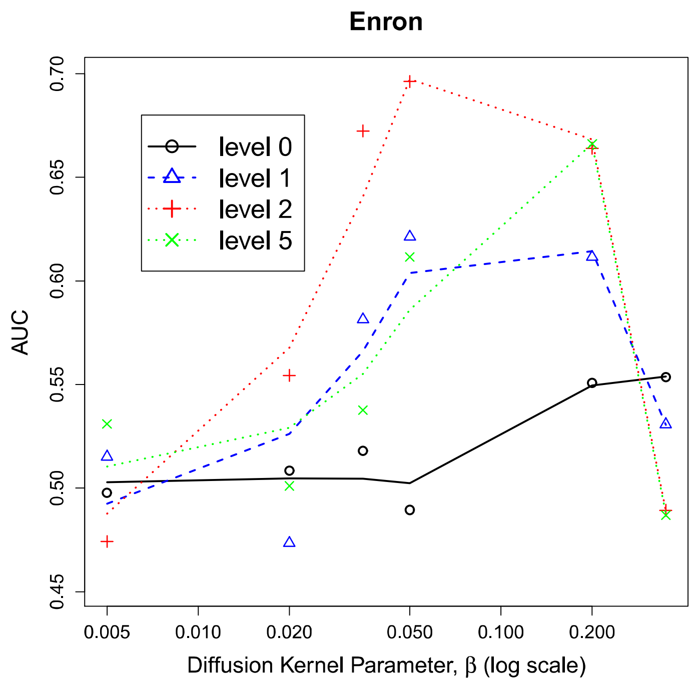

Figure 3.2: Results for the Enron Dataset

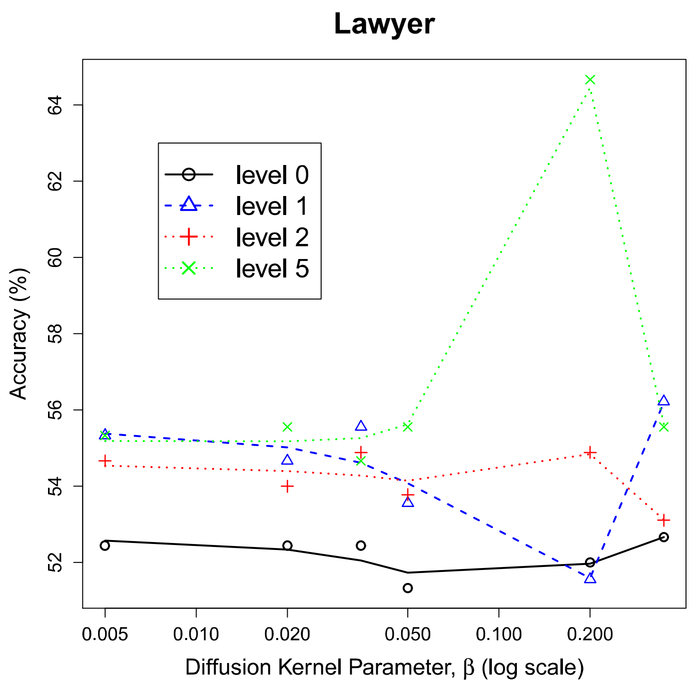

Figure 3.3: Results for the Lawyer Dataset

3.3.1 Latent Semantic Kernels

We introduce the definition of Latent Semantic Kernels according to Chapter 10 in [17].

Refer to the full set of documents $\{d_1, d_2, ..., d_l\}$ as the **corpus**, and the set of terms $\{t_1, t_2, ..., t_N\}$ occurring in the corpus as the **dictionary**. View a document as a bag of terms and represent it as a vector in a space in which each dimension is associated with one term from the dictionary

$$\phi : d \mapsto \phi(d) = (tf(t_1, d), tf(t_2, d), ..., tf(t_N, d)) \in \Re^N \tag{3.8}$$

where $tf(t_i, d)$ is the frequency of the term t_i in the document d. In this way, a document is mapped into a space of dimensionality N. We notice that the vector associated with a given document is sparse since it only contain a few number of terms in the dictionary.

Define the following matrix as the **document-term matrix** of a corpus:

$$\mathbf{D} = \begin{pmatrix} tf(t_1, d_1) & \cdots & tf(t_N, d_1) \\ \vdots & \ddots & \vdots \\ tf(t_1, d_l) & \cdots & tf(t_N, d_l) \end{pmatrix} \tag{3.9}$$

The **term-document matrix** is the transpose $\mathbf{D}'$ of the document-term matrix. The **term-by-term matrix** is given by $\mathbf{D}'\mathbf{D}$ and the **document-by-document matrix** is $\mathbf{D}\mathbf{D}'$.

Define **vector space kernel** as

$$\kappa(d_i, d_j) = \langle \phi(d_i), \phi(d_j) \rangle = \sum_{k=1}^{N} tf(t_k, d_i) tf(t_k, d_j) \tag{3.10}$$

Accordingly, the kernel matrix is:

$$\mathbf{K} = \mathbf{D}\mathbf{D}' \tag{3.11}$$

The vector space kernel ignores any semantic relation between words. To address this shortcoming, we consider a transformation of the type: $\tilde{\kappa}(d) = \kappa(d)\mathbf{S}$. We call $\mathbf{S}$ the **semantic matrix**. Accordingly, we have:

$$\tilde{\kappa}(d_i, d_j) = \kappa(d_i)\mathbf{S}\mathbf{S}'\kappa(d_j)' = \tilde{\kappa}(d_i)\tilde{\kappa}(d_j)' \tag{3.12}$$

Consider the singular value decomposition of the matrix D':

$$\mathbf{D}' = \mathbf{U}\Sigma\mathbf{V}' \tag{3.13}$$

where Σ is a diagonal matrix of the same dimensions as $\mathbf{D}$, and $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices whose columns are the eigenvectors of $\mathbf{D}'\mathbf{D}$ and $\mathbf{D}\mathbf{D}'$ respectively.

Let $\mathbf{S} = \mathbf{U}_k\mathbf{U}_k'$. Then the documents are projected into the space spanned by the first k columns of $\mathbf{U}$, using these new k-dimensional vectors for subsequent processing:

$$d \mapsto \phi(d)\mathbf{U}_k \tag{3.14}$$

where $\mathbf{U}_k$ is the matrix containing the first k columns of $\mathbf{U}$.

In this way, the eigenvectors for a set of documents can be viewed as concepts described by linear combinations of terms chosen in such a way that the documents are described as accurately as possible using only k such concepts.

Thus, we have the matrix of **Latent Semantic Kernels**:

$$\tilde{\kappa}(d_i, d_j) = \kappa(d_i)\mathbf{U}_k\mathbf{U}_k'\kappa(d_j)' \tag{3.15}$$

According to the dual representation, latent semantic kernels can be implemented as:

$$\phi(d)\mathbf{U}_k = \left(\lambda_i^{-\frac{1}{2}}\sum_{j=1}^{l}(\mathbf{v}_i)_j\kappa(d_j, d)\right)_{i=1}^{k} \tag{3.16}$$

where λ_i and $\mathbf{v}_i$ are eigenvalue and eigenvector pairs of the the kernel matrix.

3.3.2 Data

In this section, we use the WebKB data to do experiments. The WebKB data consist of 8282 web pages taken from various universities in 1997. These web pages were manually classified into 7 categories: staff, department, project, student, faculty, course, and other. Some preprocessing steps had been taken before running the experiments [21]. The final term-document matrix we used to perform our experiments consisted of 5893 documents (see Table 3.2) indexed by 56,532 terms.

Table 3.2: Summary of the WebKB data after the preprocessing steps

Category	Number	Percentage
Staff	74	1.2
Department	177	3.0
Project	396	6.7
Student	622	10.6
Faculty	631	10.7
Course	642	10.9
Other	3351	56.9
Total	5893	100.0

3.3.3 Results

Here, we also measure performance by classification accuracy. Figure (3.4), together with figure (3.2) and figure (3.3), show that SVMs using high-level kernels such as $\boldsymbol{K}_{\mathcal{F}}$ and $\boldsymbol{K}_{\mathcal{F}^2}$ often performed better than SVMs using the initial (level-0) kernel $\boldsymbol{K}_X$. At the same time, we can see that going up to higher levels such as level-5 can sometimes lead to over-fitting, as one would expect because the models become more complex. In practice, one can use cross validation to determine how deep to go.

3.4 Protein and Pairwise Kernels

The prediction of protein-protein interactions is essential to understanding the molecular mechanisms inside the cell. It can be viewed as a binary classification problem, "interacting" and "non-interacting". In 2005 Ben-Hur et al. proposed the tensor product pairwise kernel for protein-protein interactions [26] and then in 2006 Vert et al. proposed another k-ernel for pairs of proteins — the metric learning pairwise kernel[27]. In this section, we will introduce these two kinds of kernels and apply DKMs to the prediction of protein-protein interactions with them.

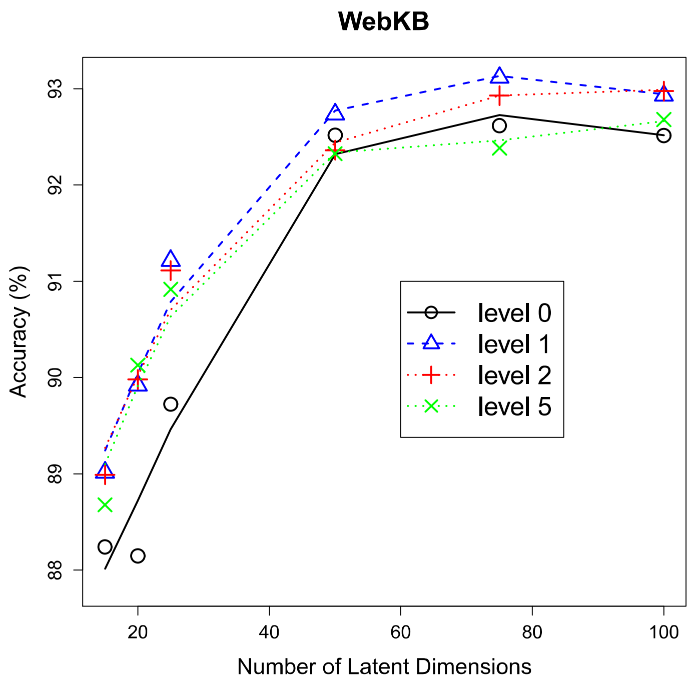

Figure 3.4: Results for WebKB data

3.4.1 Kernels for pairs of proteins

Suppose that $\{p_1, p_2, ..., p_n\}$ are a set of proteins and our objective is to predict whether each pair of these proteins, (p_i, p_j), is interacting or not.

First of all, we need to define a kernel for the single protein. There are many approaches to build kernels based on sequences of proteins, such as the spectrum kernel (Leslie et al., 2002) [33], the motif kernel (Ben-hur and Brutlag, 2003) [34] and the Pfam kernel (Gomez et al., 2003) [35]. But since proteins have 3D structures, kernels based on structures of proteins are undoubtedly more accurate. In 2010, Hue et al. proposed a method to build kernels protein structures [22]. They build kernels by alignment algorithms which create an alignment between two proteins and compute a score reflecting the alignment's quality, such as CE [23], DALI [24] and MAMMOTH [25]. Treat the output of these methods as an arbitrary score, denoted as $s(p, q)$. To convert the score to a kernel, we have to subtract the negative portion of the eigenvalue spectrum because the alignment quality score is not positive semidefinite and thus can not be used as a kernel function directly.

Suppose that M is the similarity score matrix. Consider its singular value decomposition:

$$\mathbf{M} = \mathbf{U}'\Sigma\mathbf{V} \tag{3.17}$$

Then the kernel matrix for the single protein can be defined as:

$$\mathbf{K} = \mathbf{U}'\Psi(\Sigma)\mathbf{V} \tag{3.18}$$

where $\Psi(\Sigma) = diag(\Psi(\lambda_1), \Psi(\lambda_2), ..., \Psi(\lambda_n))$, with $\Psi(\lambda) = 1 + \lambda$ if $\lambda > 0$ and $\Psi(\lambda) = 0$ if $\lambda \leq 0$.

Suppose that (p_1, p_2) and (q_1, q_2) are two pairs of proteins and $\boldsymbol{K}$ is the kernel between proteins. In order to derive the pairwise kernels $\boldsymbol{K}((p_1, p_2), (q_1, q_2))$ between two pairs of proteins from the single protein kernels $\boldsymbol{K}(p, q)$ between two proteins, we need to find a general method to build a kernel for pairs of objects from any kernel for objects. We introduce two of this kind of methods as follows.

The first one is called the **Tensor product pairwise kernel** (TPPK) [28]. It is obtained by a tensorization of the initial feature space. In this approach, two pairs of proteins are considered to be similar to one another when each protein from one pair is similar to one protein of the other pair. For example, if p_1 is similar to q_1 and p_2 is similar to q_2, then the pair (p_1, p_2) and (q_1, q_2) are similar. Thus, the TPPK between pairs of proteins can be translated as:

$$\mathbf{K}_{TPPK}((p_1, p_2), (q_1, q_2)) = \mathbf{K}(p_1, q_1)\mathbf{K}(p_2, q_2) + \mathbf{K}(p_1, q_2)\mathbf{K}(p_2, q_1) \tag{3.19}$$

However, TPPK has a counter-intuitive property that two protein pairs can be considered similar when the underlying protein pairs are strongly dissimilar [22]. Because if the base kernel function $\mathbf{K}$ can return negative values which yield a positive value for $\mathbf{K}_{TPPK}$ when multiplied together. To avoid this artifact, we can add 1 to the base kernel function.

The second one is called the **Metric learning pairwise kernel** (TPPK) [27]. It came from the idea of using distance metric learning to solving the problem of graph inference [30]. MLPK between pairs of proteins is defined as:

$$\mathbf{K}_{MLPK}((p_1, p_2), (q_1, q_2)) = [\boldsymbol{K}(p_1, q_1) + \boldsymbol{K}(p_2, q_2) - \boldsymbol{K}(p_1, q_2) - \boldsymbol{K}(p_2, q_1)]^2 \tag{3.20}$$

We will compare these two kinds of pairwise kernels in the following. The rationale behind TPPK is that the comparison between a pair (p_1, p_2) and (q_1, q_2) is done through the comparison of p_1 with q_1 and p_2 with q_2 using the kernel between individual proteins, on the one hand, and the comparisons of p_1 with q_2 and p_2 with q_1, on the other hand.

Since the formula of the MLPK might seem less intuitive than that of the TPPK, we use some simple algebra here to help better understand the MLPK. Any positive definite kernel can be written as an inner product:

$$\boldsymbol{K}(\mathbf{x_1}, \mathbf{x_2}) = \phi(\mathbf{x_1}) \cdot \phi(\mathbf{x_2}) \tag{3.21}$$

Hence the MLPK can be rewritten as follows by plugging 3.21 into 3.20:

$$\mathbf{K}_{MLPK}((p_1, p_2), (q_1, q_2)) = [(\phi(p_1) - \phi(p_2)) \cdot (\phi(q_1) - \phi(q_2))]^2 \tag{3.22}$$

So up to the square exponent, MLPK is an inner product between pairs after mapping the pairs (p_1, p_2) and (q_1, q_2) to $\phi(p_1) - \phi(p_2)$ and $\phi(q_1) - \phi(q_2)$, respectively. Thus, the major difference between TPPK and MLPK is that the former involves comparison between individual proteins of the first pair and individual proteins of the second pair, while the later compares pairs through the differences between their elements in the feature space.

Combining TPPK and MLPK by simply adding them together. In [27], they show that MLPK nearly always provides better prediction performance than TPPK, and that the combination of MLPK and TPPK together almost always leads to the best results. More kinds of combination of pairwise kernels can be found in [28].

3.4.2 Data

The data we use in this section comes from the Database of Interacting Proteins (DIP) [32]. The complete database contains 88,618 interactions among 27,496 proteins. Two different data sets are generated [22]: the "core" data set (It contains 6,175 proteins, with 1,581 interacting pairs involving 824 proteins and 4,743 non-interacting pairs. These interactions are considered reliable based on expression data and the presence of paralogous interacting protein pairs.) and the "small-scale" set (It contains 6,187 proteins, with 1,392 interacting pairs involving 1,175 proteins and 4,176 non-interacting pairs. These interactions are verified by small-scale experimental method, using techniques that reliably indicate direct physical interaction of proteins). Due to memory limited, in our experiment, we just choose a small subset from the "core" data set, which contains 200 proteins, with 50 interacting pairs and 150 non-interacting pairs. We download the PDB IDs and their labels ("DIP CORE") and kernel matrix for the single proteins ("MAMMOTH kernel matrix") from the website http://noble.gs.washington.edu/proj/pips.

3.4.3 Results

In our experiment, we use MAMMOTH [25] to build the single kernel based on protein structures and use the simple combination of TTPK and MLPK to build the pairwise kernel for pairs of proteins.

The proportion of non-interacting pairs with respect to interacting pairs is equal to $r = 4$ and this ratio will be much higher in realistic setting. Thus, we can not measure the result by accuracy. Besides, when dealing with highly skewed datasets, Precision-Recall (PR) curves give a more informative picture of an algorithm's performance than ROC curves do. It is because when the number of negative examples greatly exceeds the number of positives examples, a large change in the number of false positives can lead to a small change in the false positive rate used in ROC analysis. PR analysis, on the other hand, by comparing false positives to true positives rather than true negatives, captures the effect of the large number of negative examples on the algorithm's performance [36].

Therefore, we will use F-measure to measure the result in this section which is similar to what the authors did in [22].

First of all, we will introduce some related concepts of F-measure. For classification problems, the terms true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) compare the predicted class labels assigned by a classifier with the actual class labels. **Precision** is defined as:

$$precision = \frac{TP}{TP + FP} \tag{3.23}$$

and **Recall** is defined as:

$$recall = \frac{TP}{TP + FN} \tag{3.24}$$

F-measure is a measure which combines precision and recall:

$$F - measure = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.25}$$

Figure (3.25) is the F-measures for DKMs. As we expected, the results of level-1 to level-4 are better than that of level-0.

3.5 Additional Insight

Using the (smallest) "Lawyer" data set (see Section 3.2.2), Figure (3.6) shows an example of how the kernel matrix typically evolves as we move to higher-level feature spaces, starting from $\boldsymbol{K}_X$ with $\beta = 0.1$. Here, we see that the initial kernel matrix, $\boldsymbol{K}_X$, is close to being diagonal, which is not very informative. The subsequent kernel matrices, $\boldsymbol{K}_{\mathcal{F}}$ and $\boldsymbol{K}_{\mathcal{F}^2}$, are much more informative, which partially explains the improved classification performance. As we move to even higher levels, however, the kernel matrix starts to lose its "richness" again. To some extent, this explains the diminishing returns for going to higher levels, as well as the eventual deterioration in classification performance by going to very high levels (over-fitting). Though Figure (3.6) contains just one specific example, the kind of phenomenon depicted there is quite typical and representative.

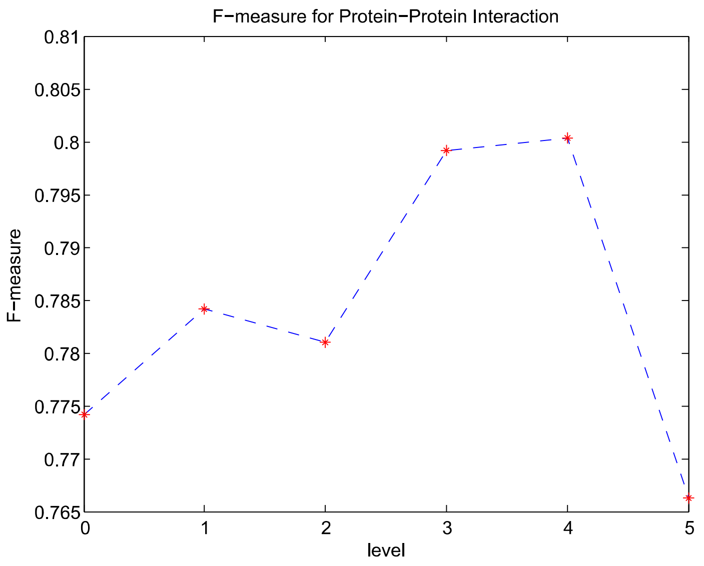

Figure 3.5: Results for Protein-Protein Interaction

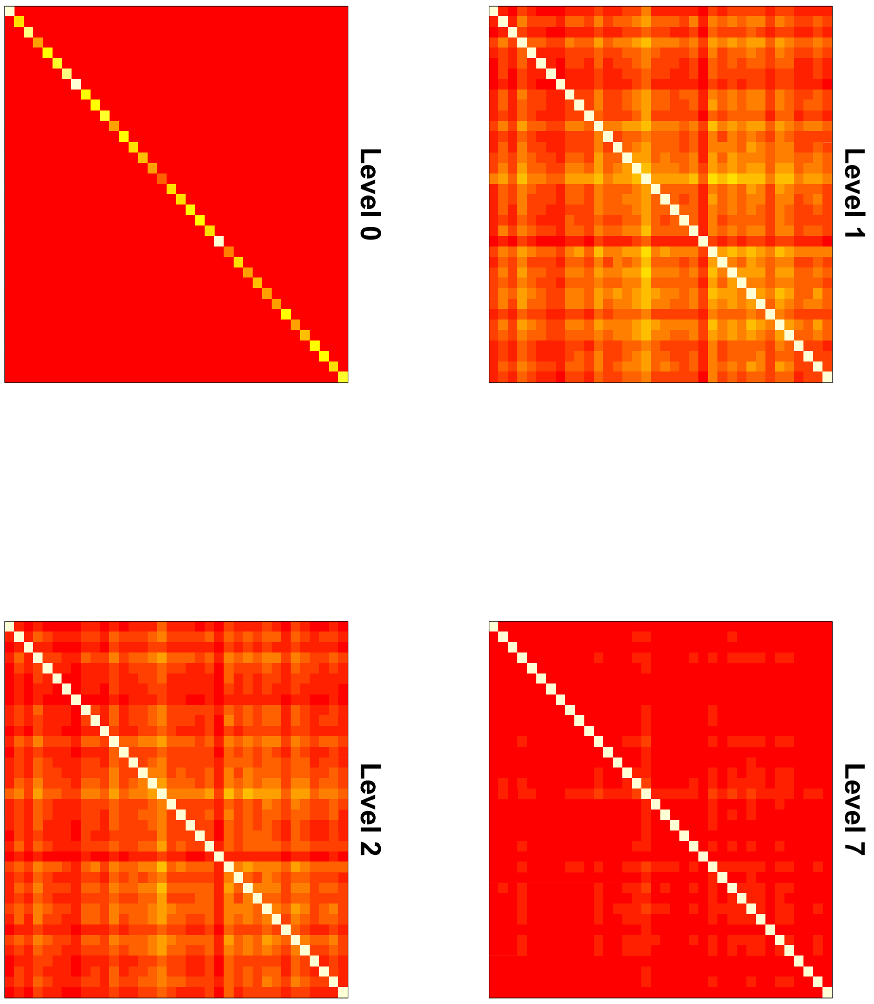

Figure 3.6: Visualizing the kernel matrices, using the (smallest) “Lawyer” data set.

3.6 More Explanation

From equation (2.13), we know that $\boldsymbol{K}_{\mathcal{F}}$ is a function of $d_{\mathcal{F}}$:

$$\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) = \psi_1(d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j})) \tag{3.26}$$

From equation (2.5), we know that $d_{\mathcal{F}}$ is a function of $\boldsymbol{K}_X$:

$$d_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) = \psi_2(\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_i}), \boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}), \boldsymbol{K}_X(\mathbf{x_j}, \mathbf{x_j})) \tag{3.27}$$

Thus,

$$\boldsymbol{K}_{\mathcal{F}}(\mathbf{x_i}, \mathbf{x_j}) = (\psi_1 \cdot \psi_2)(\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_i}), \boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j}), \boldsymbol{K}_X(\mathbf{x_j}, \mathbf{x_j})) \tag{3.28}$$

Then a natural question comes: why not writing our deep kernels as the composition of functions instead of generating them from the original kernel recursively?

The reason is there is no guarantee that every kernel can be written out as a function, for example, the diffusion kernels on graph nodes, which are defined in section (3.2.1) as equation (3.5). If the original kernel $\boldsymbol{K}_X$ is a diffusion kernel, there is no expression for us to calculate each element $\boldsymbol{K}_X(\mathbf{x_i}, \mathbf{x_j})$ in the kernel matrix. We have to compute the entire matrix by equation (3.6). In this case, it is impossible to calculate high level kernels directly from one composite function, without calculating the original kernel matrix in advance.

Besides, we note that from equation (2.18), in order to compute $h(\mathcal{F})$, we have to know the entire vector $d_{\mathcal{F}}$, which means that the kernel matrix $\boldsymbol{K}_X$ should be known in advance.

Chapter 4

Summary

Kernel Machines are very useful and popular in all kinds of areas. However, sometimes linear classification in the implied feature space $\mathcal{F}$ may be inadequate, especially when there are limited choices of kernel functions, e.g., when dealing with more "exotic" types of data such as networks and texts. Based on the premise, we apply the "kernel trick" again in the implicit feature space itself and repeat this process recursively. Then we can get our novel kind of kernel machines — Deep Kernel Machines (DKMs).

We apply this algorithm to various data sets with different types of kernels, such U-Ci data with RBF kernels, Proteins data with Pairwise kernels, and so on. The results supported our point of view. The same linear classifier often produced better results in high-level feature spaces such as $\mathcal{F}^2$ and $\mathcal{F}^3$ than in the initial feature space $\mathcal{F}$. As such, a DKM can be said to possess an "automatic kernel correction" capability. It essentially provides us with a recursive algorithm to find good feature spaces.

Two common challenges faced by all deep-learning algorithms are:

- the existence of many tuning parameters;
- the question of how deep one should go.

While these decisions are often necessarily problem-dependent and can always be made, in principle, by cross validation for lack of better procedures,we have attempted to provide some heuristics (Section 2.2.5) and guidelines (Section 3.5) for how to make these decisions in practice.

We are especially intrigued by Figure (3.6) and the possibilities it suggests. Given a kernel matrix $\boldsymbol{K}$, how do we quantify the amount of information it contains, and how does the notion of "information richness" correlate with classification performance? We are trying to answer these questions as part of our continuing research.

References

[1] Klaus-Robert Muller, K.-R. Muler, A. J. Smola, G. Rasch, B. Schokopf, J. Kohlmorgen, and V. N. Vapnik, *An Introduction to Kernel-Based Learning Algorithms*, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 12, NO. 2, MARCH 2001.

[2] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, Science, 313:504507, 2006.

[3] Klaus-Robert Muller, Sebastian Mika, Gunnar Rasch, Koji Tsuda and Bernhard Schokopf, *Predicting time series with support vector machines*, IArtificial Neural Networks-ICANN'97. ser. Springer Lecture Notes in Computer Science, vol. 1327, pp. 9991004, 1997.

[4] S. Mika, G. Rasch, J. Weston, B. Schokopf, and K.-R. Muler, *Fisher discriminant analysis with kernels*, IEEE, pp. 4148, 1999.

[5] I. Sutskever and G. E. Hinton, *Deep narrow sigmoid belief networks are universal approximators*, Neural Computation, 20:26292636, 2008.

[6] Y. Bengio, *Learning deep architectures for AI*, Foundations and Trends in Machine Learning, 2(1):1127, 2009.

[7] G. E. Hinton, S. Osindero, and Y. Teh, *A fast learning algorithm for deep belief nets*, Neural Computation, 18:15271554, 2006.

[8] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, *Exploring strategies for training deep neural networks*, Journal of Machine Learning Research, 10:140, 2009.

[9] Y. Cho and L. Saul, *Kernel methods for deep learnings*, In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, Advances in Neural Information Processing Systems 22, pages 342350. Curran Associates, Inc., 2009.

[10] C. Cortes and V. Vapnik, *Support-Vector Networks*, Machine Learning, 20, 1995.

[11] Bernhard Scholkopf, Sebastian Mika, Alex Smola, Gunnar Ratsch and Klaus-Robert Muller, *Kernel PCA Pattern Reconstruction via Approximate Pre-Images*, 1998.

[12] R. M. Neal, *Support-Vector Networks*, Regression and classification using gaussian process priors, In J. M. Bernardo et al. (Eds), Bayesian statistics 6, 475-501, Oxford University Press.

[13] T. Joachims, *Text categorization with support vector machines*, European Conferece on Machine Learning (ECML), Springer-Verlag, 1998.

[14] Mu Zhu, *Kernels and Ensembles*, The American Statistician, May 1, 2008, 62(2): 97-109.

[15] T. Hastie, R. Tibshirani and J. H. Friedman, *The Elements of Statistical Learning*, Springer, 2001.

[16] Nello Cristianini and John Shawe-Taylor, *An introduction to support Vector Machines: and other kernel-based learning methods*, Cambridge University Press, 2000.

[17] Nello Cristianini and John Shawe-Taylor, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[18] E. Lazega, *The Collegial Phenomenon: The Social Mechanisms of Cooperation Among Peers in a Corporate Law Partnership*, Oxford University Press, 2001.

[19] R. J. Bolton and D. J. Hand, *Statistical fraud detection: A review*, Statistical Science, 17(3): 235255, 2002.

[20] M. S. Pepe, *The Statistical Evaluation of Medical Tests for Classification and Prediction*, Oxford University Press, 2003.

[21] Alexandra Laflamme-Sanders and Mu Zhu, *LAGO on the unit sphere*, Neural Networks, 21: 1220-1223, 2008.

[22] Martial Huel, Michael Riffle, Jean-Philippe Vert and William S Noble, *Large-scale prediction of protein-protein interactions from structures*, BMC Bioinformatics, 11: 144, 2010.

[23] Shindyalov IN and Bourne PE, *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path*, Protein Engineering, 11:739-747, 1998.

[24] Holm L and Sander C, *Protein Structure Comparison by Alignment of Distance Matrices*, Journal of Molecular Biology, 233:123-138, 1993.

[25] Ortiz AR, Strauss CEM and Olmea O, *MAMMOTH (Matching molecular models obtained from theory): An automated method for model comparison*, Protein Science, 11:2606-2621 ,2002.

[26] Martin S, Roe D and Faulon JL, *Predicting protein-protein interactions using signature products*, Bioinformatics, 21(2):218-226, 2005.

[27] Vert JP, Qiu J and Noble WS, *A new pairwise kernel for biological network inference with support vector machines*, BMC Bioinformatics, 8(Suppl 10):S8, 2007.

[28] Ben-Hur A and Noble WS, *Kernel methods for predicting protein-protein interactions*, BMC Bioinformatics, 21(suppl 1):i38-i46, 2005.

[29] J. Lafferty and G. Lebanon, *Diffusion kernels on statistical manifolds*, Journal of Machine Learning Research, 6:129163, 2005.

[30] Vert JP and Yamanishi Y, *Supervised Graph Inference*, In Advances in Neural Information Processing Systems Volume 17, Edited by: Saul LK Weiss Y, Bottou L. Cambridge, MA: MIT Press: 1433-1440,2005.

[31] E. D. Kolaczyk, *Statistical Analysis of Network Data*, Springer-Verlag, 2009.

[32] Xenarios I, Rice DW, Salwinski L, Baron MK, Marcotte EM and Eisenberg D, *DIP: the Database of Interacting Proteins*, Nucleic Acids Research, 28:289-291, 2000.

[33] Leslie,C., Eskin,E. and Noble,W.S., *The spectrum kernel: A string kernel for SVM protein classification*, Proceedings of the Pacific Symposium on Biocomputing, New Jersey. World Scientific, Singapore, pp. 564575, 2002.

[34] Ben-hur,A. and Brutlag,D., *Remote homology detection: a motif based approach*, Bioinformatics, 19 (Suppl 1), i26-i33, 2003.

[35] Gomez,S.M., Noble,W.S. and Rzhetsky,A., *Learning to predict proteinprotein interactions*, Bioinformatics, 19, 18751881, 2003.

[36] Davis J and Goadrich M, *The relationship between precision-recall and ROC curves*, Proceedings of the International Conference on Machine Learning, 2006.