Learning Representations for Classification Problems in Reproducing Kernel Hilbert Spaces

Murilo Vale Ferreira Menezes

Graduate Program in Electrical Engineering

Federal University of Minas Gerais

Supervisor: Prof. Antônio de Pádua Braga

Co-Supervisor: Prof. Luiz Carlos Bambirra Torres

Dissertation

Master of Science

10/2020

Aprendendo Representações para Problemas de Classificação em Espaços de Hilbert do Kernel Reprodutivo

Murilo Vale Ferreira Menezes

Programa de Pós-Graduação em Engenharia Elétrica

Universidade Federal de Minas Gerais

Orientador: Prof. Antônio de Pádua Braga

Coorientador: Prof. Luiz Carlos Bambirra Torres

Dissertação de

Mestrado

10/2020

To my parents.

Acknowledgements

First of all, i want to thank my parents, Magda and Tarciso, my brothers, Marcelo and Guilherme, and sisters-in-law Mayesse and Renata, for all the support and for always believing in me. I would not be where I am if not for you.

Thank you to my nephews, Felipe, Pedro, and Joaquim, for bringing joy and love to the simple side of life. I learn more from you than you can imagine.

To Laura for all the love and support during all these years, in both calm and stressful moments. Thank you for helping me be the best version of myself.

To my advisor, Prof. Antônio de Pádua Braga, for all the guidance and teachings in the last four years. I am very grateful for having the opportunity of working with you. To my co-advisor, Prof. Luiz Carlos Bambirra Torres, for the friendship and advices during all the lunches in *bandejão*.

Thank you to all my friends from LITC, Hekima, and iFood, for the discussions and partnership in the everyday routine, and to my friends from Electrical Engineering. Your friendship is very important to me.

It's time we start smiling
What else should we do?

George Harrison

Abstract

The performance of a machine learning method, regardless of the task it is trying to solve, is dependent on the quality of the representations it receives. Not surprisingly, there is a wide class of methods that aim to leverage statistical properties of a dataset, either with raw or handcrafted features, to build more useful representations, from Principal Component Analysis to recent deep learning techniques.

Kernel methods are a very powerful family of models, which have the ability to map the input data into a space where otherwise hard tasks become easier to solve, such as linear classification. These methods have the ability to express their learning process only in terms of kernels, which are similarity functions between samples and can be interpreted as inner products in this mapped space, dismissing the need to explicitly map the data. However, these kernel functions often have a set of parameters that have to be chosen according to each task and have a great influence on the mapping, and, therefore, on the final task.

This work proposes two objective functions which can be used to learn these kernel parameters and achieve good classification results. Experiments with Gaussian, Laplacian, and sigmoid kernels are conducted. An interpretation of neural networks inside the kernel framework is also proposed, enabling these networks to be trained to learn representations using the proposed functions.

Based on empirical results and the analysis of each kernel function used in the experiments, properties of the proposed functions are discussed, along with how they can successfully be used in practice.

Resumo

O desempenho de um modelo de aprendizado de máquina, independentemente da tarefa, depende da qualidade das representações que o fornecemos. Há uma ampla classe de métodos que utilizam propriedades estatísticas de um conjunto de dados para aprender representações, da Análise de Componentes Principais (PCA) a técnicas de aprendizado profundo.

Métodos de kernel são uma família poderosa de modelos que têm a habilidade de mapear os dados para um espaço onde tarefas como classificação linear se tornam mais fáceis de serem resolvidas. Estes métodos têm a habilidade de expressar seu processo de aprendizado apenas em termos de funções de kernel, que são medidas de similaridade entre amostras e podem ser interpretadas como produtos internos neste espaço mapeado, não havendo necessidade do mapeamento explícito. Contudo, estas funções de kernel tipicamente têm um conjunto de parâmetros que devem ser ajustados de acordo com cada tarefa e têm grande influência no mapeamento, e, portanto, na tarefa final.

Este trabalho propõe duas funções objetivo com as quais podemos aprender estes parâmetros e atingir bons resultados em problemas de classificação. Conduzimos experimentos com kernels Gaussianos, Laplacianos e sigmoidais. Além disso, uma interpretação de redes neurais dentro do arcabouço de kernels é proposta, mostrando que estas redes podem ser treinadas para aprender representações de acordo com as funções propostas.

Com base em resultados empíricos e na análise das funções de kernel usadas, discutimos as propriedades das funções propostas e como usá-las na prática.

Contents

List of Symbols

$\mathbb{X}$	Set of vectors
$\mathcal{X}$	Vector space
$\mathbf{x}$	Vector
x	Scalar
$\mathcal{F}$	Function space
f	Function
P	Probability measure
X	Random variable
Θ	Parameter space
θ	Parameter vector
L	Loss function
k	Kernel function
N	Sample size
$\mathbf{w}$	Weight vector
W	Weight matrix
Σ	Covariance matrix
K	Gram matrix
I	Identity matrix
$\mathcal{H}$	Representation space
C	Soft-margin SVM trade-off parameter
σ	Radial basis function width
γ	Sigmoidal scaling parameter
ϕ	Function mapping to representation space
$\langle \cdot, \cdot \rangle$	Inner product
ψ	Kernel similarity
$\hat{\psi}$	Empirical kernel similarity
ξ	Kernel similarity of distributions
$\hat{\xi}$	Empirical kernel similarity of distributions
$\mathcal{D}$	Kernel distributional discrepancy
$\hat{\mathcal{D}}$	Empirical kernel distributional discrepancy

List of Abbreviations

AUC	Area Under the Curve
CAE	Contractive Autoencoder
ERM	Empirical Risk Minimization
GAN	Generative Adversarial Network
IPM	Integral Probability Metric
KDD	Kernel Distributional Discrepancy
KDE	Kernel Density Estimation
KSD	Kernel Similarity of Distributions
KSV	Kernel Similarity Variance
MLP	Multilayer Perceptron
MMD	Maximum Mean Discrepancy
PSD	Positive Semidefinite
RBF	Radial Basis Function
RKHS	Reproducing Kernel Hilbert Space
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SRM	Structural Risk Minimization
SVM	Support Vector Machine
t-SNE	t-distributed Stochastic Neighbor Embedding
VC	Vapnik-Chervonenkis

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning relies heavily on representations. Every time one is confronted with a learning task, data with some predefined structure is given, which determines not only the kind of algorithms to use, but also their performance. Consequently, great effort is made in extracting and processing these representations before the final learning task.

In order to reach relevant representations, one option is to handcraft features according to prior knowledge of the problem, which can be time-consuming and does not guarantee useful features for the task at hand. **Representation learning** methods aim to automatically find useful representations from input data, whether to make classification or regression problems easier to solve, for visualization, or just to capture the dynamics of our input data into a more representative space.

On top of the given input representations, one can, for instance, select features from input according to some criteria (Guyon & Elisseeff, 2003), which can help to rule out noisy and redundant features. Feature selection can be simple to execute and interpret. However, a broader class of algorithms consist of learning a representation that does not necessarily preserve the input features.

A myriad of representation learning methods are described in the literature, many of which are covered by Bengio *et al.* (2013). These methods can be probabilistic, such as Boltzmann machines (Hinton *et al.*, 1984; Salakhutdinov & Hinton, 2009), that try to find latent random variables that are able to explain our observed data. There are also methods that are set as learning a parametric map

from an input space $\mathcal{X}$ to a representation space $\mathcal{H}$, such as autoencoders (Hinton & Zemel, 1994), which aim to find a transformation that allows the reconstruction of the inputs with maximum accuracy. This distinction, nevertheless, is not disjoint, and some algorithms can be interpreted in both groups.

Kernel methods also take advantage of mapping the input instances into a feature space (Scholkopf & Smola, 2001). Here, the representations are defined according to **kernel functions**, which are similarity measures in the input space. By defining algorithms in terms of these functions, problems can be solved implicitly in the feature space, allowing very high-dimensional spaces to be used without having to store all the features. An example of such methods is the support vector machine (Boser *et al.*, 1992). SVMs are originally linear classifiers, and, using kernel functions, are capable of solving nonlinear problems with very high performance.

Using kernels, one can even solve problems using methods that would not be able to solve them in the original space. For instance, an SVM can be used to solve a classification problem in the space of unstructured text documents by only defining a kernel function that measures the similarity between two documents. Although general-purpose kernels, such as the Gaussian, are widely used, there is also the possibility of handcrafting them using prior knowledge.

Kernel functions usually have parameters to be set beforehand, for example the width σ in Gaussian and the degree d in polynomial kernels. These parameters have to be chosen carefully according to the task, since they directly determine the feature map, and consequently have great impact on the final performance. These parameters are typically chosen using grid-search and cross-validation (Friedman *et al.*, 2001), which can be resource-consuming.

There are, in the literature, methods that aim to optimize kernel parameters based on training data. Kim *et al.* (2008) formulates a convex optimization problem to learn the kernel in a classification task. Torres *et al.* (2014) use a graph structure in the input space in order to choose the width of a Gaussian for an RBF network. Wanderley *et al.* (2014) also focuses in RBF kernels, proposing methods to choose the width parameter for density estimation problems.

This work proposes two methods to optimize kernel parameters. The methods are proposed for general kernels and can be used to learn parameters of any

kernel with continuous parameters, as long as the norms of the representations are bounded. As a particular case, these methods can be used to learn representations using not only classic continuous kernels, but also parameters of neural networks. These methods are used to learn representations for a classification task.

The first method, the **kernel distributional distance**, is based on a measure of discrepancy between probability distributions. This measure is defined using a kernel function, and can be computed using only similarities between patterns. It can also be visualized in a similarity space of classes. By maximizing the pairwise distances between conditional distributions on the input space, we are able to learn representations that segregate different classes from each other, yielding good results with a linear classifier. This discrepancy function is similar to the **maximum mean discrepancy** (Gretton *et al.*, 2012) when defined on reproducing kernel Hilbert spaces.

The second method, called **kernel similarity variance**, is unsupervised. It was proposed with the goal of capturing density information on the input using only pairwise similarities between patterns. The parameters are maximized while searching for a descriptive representation space. This method yields good results for translation-invariant kernels that capture local structure, but not so much for other kernels. Possible explanations on why this happens are discussed.

Experiments using accuracy and AUC are executed in 19 real-world datasets, using Gaussian, Laplacian, and sigmoidal kernels, as well as multilayer perceptrons. Visualizations of the learned representations in a computer vision task are also provided. The results show that these methods can be used to learn class-dependent structure effectively.

1.1 Publications

- Menezes, M., Torres, L. C., & Braga, A. P. (2017). Otimização da Largura de Kernels RBF para Máquinas de Vetores de Suporte: Uma Abordagem Baseada em Estimativa de Densidades. In XIII Congresso Brasileiro de Inteligência Computacional.

- Menezes, M., Torres, L. C., & Braga, A. P. (2019, September). Learning Regularization Parameters of Radial Basis Functions in Embedded Likelihoods Space. In EPIA Conference on Artificial Intelligence (pp. 281-292). Springer, Cham.
- Menezes, M., Torres, L. C., & Braga, A. P. (2019). Width optimization of RBF kernels for binary classification of support vector machines: A density estimation-based approach. Pattern Recognition Letters, 128, 1-7.
- Menezes, M., Torres, L. C., & Braga, A. P. A Survey of Representation Learning Methods in Reproducing Kernel Hilbert Spaces. (In preparation)
- Menezes, M., Torres, L. C., & Braga, A. P. Learning Representations With Neural Networks Using Kernel-Based Functions. (In preparation)
- Menezes, M., Torres, L. C., & Braga, A. P. Unsupervised Learning of Kernel Parameters Using Global Similarity Measures. (In preparation)

1.2 Outline

This work is organized as follows.

Chapter 2 covers the theoretical background needed for this work. The general setting of statistical learning theory and inductive principles are defined, as well as representation learning basics. Kernel theory is also covered, with definitions of positive semidefinite kernels and the reproducing kernel map. Relevant kernel methods are also described, and the Chapter ends with the kernel interpretation of a neural network.

In Chapter 3 the main methods proposed in this work are described. Important definitions such as the kernel similarity and the similarity space are given, and the functions to be optimized are defined afterwards. This Chapter also discusses which properties of kernels are necessary for our functions to be optimized using continuous methods, with a subsequent analysis of the specific kernels used in this work under these requirements.

Chapter 4 contains the numerical and visual experiments, where it can be seen how our proposed functions perform under each type of kernel. The results are

also discussed, along with a possible reason why it works well on some settings but not so well on others.

Finally, Chapter 5 concludes the work and addresses possible sequels to further deepen the understanding of our framework.

Chapter 2

Statistical Learning, Representations, and Kernel Methods

This chapter overviews the core of the learning problem, presenting the setting of function estimation over samples generated by an unknown distribution. Then, the representation learning problem is covered, discussing how one can learn a useful vector space. Then, kernel methods are covered, giving the main framework on top of which this work is developed.

2.1 Statistical Learning Theory Basics

A system endowed with the property of learning can be described as a system that adapts its behavior with respect to some task based on former experience. In fact, Mitchell (1997) defines that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". In most cases, E consists of data collected in the past.

The problem of learning from data, as introduced by Vapnik (1995), can be seen as estimating a function using observations coming from unknown distributions. In this framework, the data belongs to an input vector space $\mathcal{X}$. Each element $\mathbf{x} \in \mathcal{X}$ is called a data point, an instance, or a sample.

A training set $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ is sampled from a well-defined but unknown probability distribution $P(\mathbf{x})$. When the problem is **supervised**, one also have access to a supervision vector $\mathbf{y}$, defined on output space $\mathcal{Y}$. For each input vector $\mathbf{x}$, a supervision vector y is sampled according to another unknown distribution $P(\mathbf{y}|\mathbf{x})$ defined over $\mathcal{Y}$. $\mathcal{Y}$ depends on the problem one is trying to solve. In a classification problem, the vectors $\mathbf{y}$ are discrete values representing the classes, and $\mathcal{Y}$ is the set of integers. For regression problems, $\mathcal{Y}$ is the real field $\mathbb{R}$.

Given samples from these distributions, the main goal is to find a function f mapping the input space $\mathcal{X}$ to an output space, typically $\mathcal{Y}$ for supervised problems. f is chosen from a set of functions $\mathcal{F}$, e.g. deep neural networks. In order to guide the search, a **loss function** L is defined, which measures the quality of a function f. Conventionally, it is called a **loss** function because the goal is to find functions that achieve low values of L[1]. Common losses for classification problems include the Cross Entropy (Goodfellow *et al.*, 2016), used when the output is a probability value, and the hinge loss (Rosasco *et al.*, 2004), used in Support Vector Machines.

The optimal function for a problem is defined as $f^* \in \mathcal{F}$ that minimizes the following **risk functional**:

$$R(f) = \int_{\mathcal{Z}} L(\mathbf{z}, f) dP(\mathbf{z}) \tag{2.1}$$

where $\mathcal{Z}$ is a general input space, which may consist of $\mathcal{X} \times \mathcal{Y}$ for supervised problems or $\mathcal{X}$ for unsupervised ones, and P is a probability measure defined over $\mathcal{Z}$.

The difficulty of finding f^* lies on the fact that one does not have access to the true distribution P, but only to a finite sample from it. In order to try to find the function that minimizes the risk functional of Equation 2.1, an **inductive principle** is needed[2].

[1] When the problem is to maximize a function, it can always be turned into a minimization problem by multiplying the function by -1.

[2] In inductive logic problems such as learning from data, one wants to reach global conclusions (the generating distribution, the input-output relation) from empirical observations (the samples). The conclusions are many times probabilistic and not provably true, in contrast to deductive reasoning, in which certainly true conclusions are reached when given true premises. Further discussions on inductive reasoning and how it applies to the learning problem is beyond the scope of this work.

One straightforward way to approximate the optimal solution is the Empirical Risk Minimization (ERM) principle. Given a finite sample $\mathbb{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_N\}$ from $\mathcal{Z}$, ERM consists on minimizing the following empirical estimate of the risk:

$$R_{emp}(f) = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{z}_i, f) \tag{2.2}$$

This principle is reasonable, especially because of the law of large numbers: when the sample size N grows, the empirical risk R_{emp} converges to the true risk R. However, with a finite sample and a broad class of functions $\mathcal{F}$, the ERM can lead to a common situation known as **overfitting**, in which one finds a function with low error in the training set but generalizes poorly outside of the observed points.

An extreme example from Scholkopf & Smola (2001) is reproduced here. In a classification problem with loss function 0-1, if $\mathcal{F}$ is allowed to be all the possible functions mapping a continuous input space $\mathcal{X}$ to $\mathcal{Y}$, one can minimize the empirical risk by choosing the following function:

$$f(x) = \begin{cases} y_i, & \text{if } \mathbf{x} = \mathbf{x}_i \text{ for } i \in \{1, \cdots, N\} \\ 1, & \text{otherwise.} \end{cases} \tag{2.3}$$

As $\mathcal{X}$ is continuous, the probability of a test point be exactly equal to one of the training points is zero (as the training set has measure zero). Thus, f would predict 1 to almost all of the test samples, and the true risk could be very high despite a very low empirical risk.

In fact, Vapnik (1995) derives a confidence interval for the risk functional. It is stated that $R(f)$ is bounded by above by a term that involves the empirical risk $R_{emp}(f)$ and a quantity that measures the **capacity** of f. For instance, when the loss function $L(z, f)$ is bounded $A \leq L(z, f) \leq B$, the following inequality holds with probability $1 - \eta$:

$$R(f) \leq R_{emp}(f) + \frac{B - A}{2} \sqrt{4 \frac{h\left(\ln(\frac{2N}{h}) + 1\right) - \ln(\frac{\eta}{4})}{N}} \tag{2.4}$$

where N is the sample size and h is called the Vapnik-Chervonenkis (VC) dimension of f, which is a measure of capacity.

Equality holds when $N \to \infty$ and that, for finite N, uncertainty rises for high-capacity functions. This happens because, when f has a very high VC dimension, one can not be sure if a low empirical risk happens because f lowers the true risk functional of if the learning machine is overfitting the training set. Figure 2.1 shows an example with two candidate functions in a regression setting.

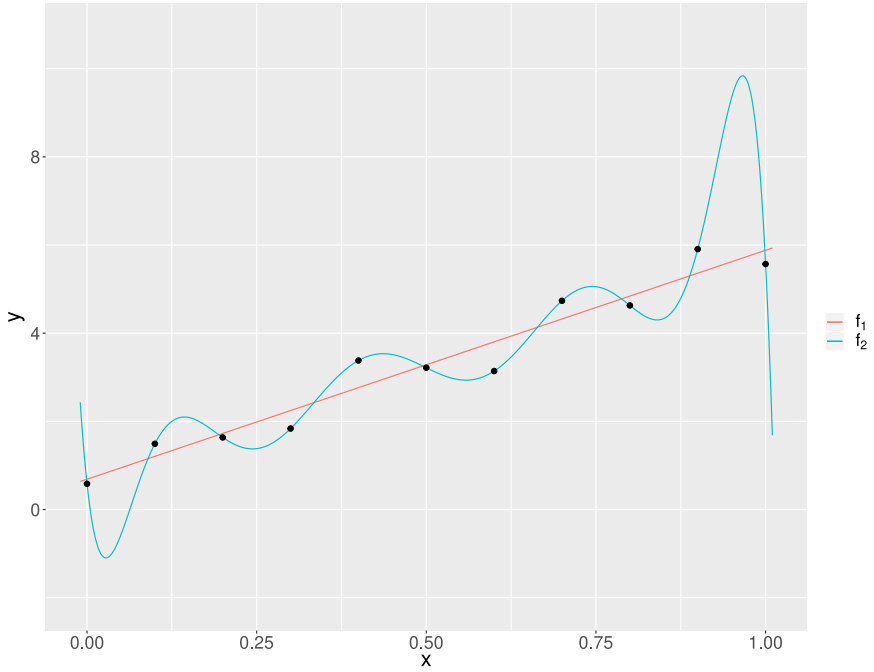


Figure 2.1: An example of two functions to fit a set of points. f_1 is a simple linear function found using least-squares, while f_2 is a polynomial of degree 10. f_2 fits all of the points exactly, and its empirical risk is zero. However, the test error may be higher than f_1's, as the function reaches more extreme values outside the training points. Empirical risk minimization alone would choose f_2 over f_1 unequivocally; structural risk minimization would take into account their complexities.

When dealing with limited data, an alternative to ERM is the Structural Risk Minimization (SRM). This inductive principle takes into account the capacity of the family of functions $\mathcal{F}$ by considering a sequence of sets of functions (the **structure**) S_i $(i = 1, 2, 3, \cdots)$, $S_1 \subset S_2 \subset S_3 \subset \cdots$, with increasing capacity. This method tries to minimize the sum of the empirical risk and a function capacity term, minimizing the bound on the true risk functional. The function structure is illustrated in Figure 2.2.

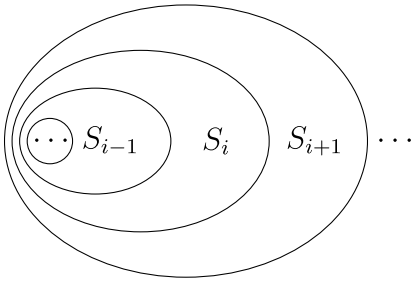


Figure 2.2: An illustration of the structural risk minimization principle. This inductive principle biases the learning machine towards simple functions that are able to explain the data sufficiently well.

One way of implementing SRM is by minimizing the regularized risk functional, which consists of the empirical risk added by a term that measures the complexity of f:

$$R_{reg}(f) = R_{emp}(f) + \lambda\Omega(f) \tag{2.5}$$

where Ω is an increasing function on the complexity of f and $\lambda > 0$ is a parameter to control the trade-off between the performance (the empirical risk) and the simplicity of f.

The regularization term usually penalizes the magnitude of f, enforcing smoothness. In neural networks, for instance, L_2 regularization (Goodfellow *et al.*, 2016; Tikhonov, 1963) is a very popular method, penalizing the sum of the L_2 norm of the weights.

2.2 Representation Learning

In a problem such as classification, a sample of inputs from space $\mathcal{X}$ is given, which may not be a relevant representation for the specific task. Patterns may come with lots of noise, or $\mathcal{X}$ may even be a space where classic pattern recognition algorithms are not defined.

The performance of any machine learning task, in fact, depends on the representation. This is why a range of methods, from kernel-based SVMs (Boser *et al.*, 1992) to deep neural networks (LeCun *et al.*, 2015), rely on a mapping from $\mathcal{X}$ to

a representation space (also called feature space) $\mathcal{H}$, where the problem becomes easier to solve. Figure 2.3 depicts an example of a representation mapping.

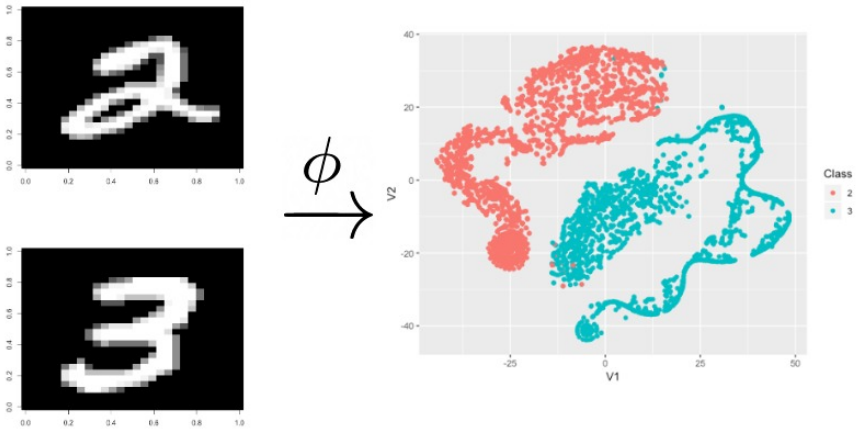


Figure 2.3: An example of a learned representation. Here, images from the MNIST handwritten digits dataset (LeCun *et al.*, 2010) are mapped into an Euclidean space, using t-SNE (Maaten & Hinton, 2008) to visualize them in a two-dimensional space.

Representation learning algorithms try to find a function ϕ that maps the points in $\mathcal{X}$ into useful representations in $\mathcal{H}$. The concept of usefulness is not self-evident, and is dependent on the task. For a classification problem, for instance, representations in which the classes are linearly separable are preferable.

However, when learning representations, one usually does not aim to approximate a given supervision vector, such as classification and regression problems. Even though representations can also be learned using information from supervision variables, there are general properties that make a representation helpful, which can be used as guidelines to conduct the learning process. Bengio *et al.* (2013) lists some of the desirable properties of a representation, which include:

- Disentanglement of latent factors: the observed variables are caused by many latent factors, and it is desirable that the representation disentangles them. For example, in speech data it may be possible to disentangle the frequency, loudness, and speaker emotion from one utterance to another.

Methods such as InfoGAN (Chen *et al.*, 2016) aim at learning disentangled features.

- Robustness: it is desirable that the features are robust with respect to small, noisy variations on the input data. One of the main concerns is to build representations that are robust to adversarial examples (Goodfellow *et al.*, 2015), in which the neural networks give entirely different outputs to very similar inputs. Contractive (Rifai *et al.*, 2011a) and denoising (Vincent *et al.*, 2008) autoencoders are built to learn robust encodings of the input data.

- Abstraction: causal factors of an observation are often linked to more abstract features. For instance, the presence or absence of a given object in an image tends to be more useful than the exact values of the pixels. This is one of the factors that explain the performance of deep learning methods: by learning hierarchical representations, one can reach very abstract final features, built on top of simpler ones.

When designing representation learning algorithms, there are also assumptions one may rely on. One of them is the **manifold hypothesis** Goodfellow *et al.* (2016), which assumes that data in high-dimensional spaces usually lie close to lower-dimensional manifolds embedded in that space. Another prior is assuming **natural clustering** (Bengio *et al.*, 2013), which means assuming that data with different categorical features, such as classes, lie on disconnected manifolds.

2.3 Kernel Methods

Kernels are, simply put, similarity functions between points in a given input space. This tool, albeit simple, is very powerful. Using kernels, one is able to solve many linear and nonlinear learning problems and take advantage of very high-dimensional representation spaces. This section present the basics of kernel theory, shedding light on the key concepts this work is based on.

2.3.1 Positive Semidefinite Kernels

The first thing to point here is that, in this work, only real-valued kernels are used, that is, functions that map to a real-valued scalar $\mathbb{R}$. Although kernels are defined in a more general form having also complex values, hereafter only real-valued kind is treated.

A real kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that represents an inner product $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$, where $\phi : \mathcal{X} \to \mathcal{H}$ is a mapping function from an input space $\mathcal{X}$ to a feature space $\mathcal{H}$ (Scholkopf & Smola, 2001). This work is focused on learning this feature space from adjusting the kernel parameters.

To build the reproducing kernel feature map framework, some definitions are necessary. All the definitions, although maybe not identical, are based on Scholkopf & Smola (2001).

Definition 1. *A positive semidefinite (PSD) kernel in an input space $\mathcal{X}$ is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that, for any set of points $\mathbf{x}_1, \cdots, \mathbf{x}_N \in \mathcal{X}$ and scalars $c_1, \cdots, c_N \in \mathbb{R}$, the following inequality holds:*

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{2.6}$$

Many of the most popular kernels are PSD, such as the Gaussian, the linear and the polynomial kernels. Now, the **Gram matrix** of k with respect to a set of input points is defined:

Definition 2. *Given a set of points $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ in $\mathcal{X}$ and a kernel k, the Gram matrix is an $N \times N$ matrix K with elements*

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.7}$$

If k is a positive semidefinite kernel, the Gram matrix K induced by k in a given set of N points is a positive semidefinite matrix, meaning that:

$$c^T K c \geq 0 \tag{2.8}$$

where $c \in \mathbb{R}^N$. This is a matrix notation equivalent to Equation 2.6.

Many of these functions, such as the Gaussian kernel, are defined depending on some parameters θ from a parameter space Θ. As the parameters of a given kernel are the main object of this work, sometimes a kernel is mentioned with its explicit parameters, k_θ.

2.3.2 Reproducing Kernel Hilbert Spaces

One way to interpret the feature space a kernel induces is via the reproducing kernel map. Here, each input point x is mapped to a function in a Hilbert space of functions defined by k.

Simply stated, a Hilbert space is an inner product space, that is, a vector space $\mathcal{H}$ endowed with an inner product $\langle \cdot, \cdot \rangle$, which is a function mapping $\mathcal{H} \times \mathcal{H} \to \mathbb{R}$. However, the converse is not true: for an inner product space to be considered a Hilbert space, it also needs to be **complete**, which means that it contains all the limits of all its Cauchy sequences (MacCluer, 2008).

Now, a reproducing kernel Hilbert space (RKHS) can be defined.

Definition 3. *Given an input vector space $\mathcal{X}$ and a Hilbert space $\mathcal{H}$, $\mathcal{H}$ is called a reproducing kernel Hilbert space if there exists a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:*

1. *k has the reproducing property:*

$$\langle f, k(\mathbf{x}, \cdot) \rangle_{\mathcal{H}} = f(\mathbf{x}) \ \forall f \in \mathcal{H} \tag{2.9}$$

   where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ represents the inner product in $\mathcal{H}$.

2. *k spans $\mathcal{H}$:*

$$\mathcal{H} = \overline{span\{k(\mathbf{x}, \cdot) | \mathbf{x} \in \mathcal{X}\}} \tag{2.10}$$

   where $\overline{X}$ denotes the completion of the set X, that is, the set X added to all the limit points of its Cauchy sequences.

In fact, the mapping of a point $\mathbf{x} \in \mathcal{X}$ to a space $\mathcal{H}$ that defines a kernel k is simply the kernel k with one of its arguments fixed in $\mathbf{x}$: $\phi(\mathbf{x}) = k(\mathbf{x}, \cdot)$ (Scholkopf & Smola, 2001).

The first condition of the definition means that, for every function in the RKHS, one can evaluate it in any point of $\mathcal{X}$ by simply taking the inner product of the function with the mapping of $\mathbf{x}$ using the kernel k. This makes possible the **kernel trick**: using $\mathcal{H}$ as a feature space in which other tasks such as classification can be performed. A function f in this space can be, for instance, a hyperplane which separates the mappings of the patterns $\phi(\mathbf{x})$, and can then be evaluated for any point in $\mathcal{X}$ to obtain a separation surface in the input space.

This is where the kernel trick comes handy in methods such as SVMs. By using a nonlinear kernel function, it is still possible to use linear methods in $\mathcal{H}$ in order to obtain a nonlinear outcome in $\mathcal{X}$.

The second condition means that the span of all the possible mappings of $\mathcal{X}$ into $\mathcal{H}$ is dense in $\mathcal{H}$.

If k is a kernel defined by a RKHS $\mathcal{H}$, the reproducing property assures that:

$$k(\mathbf{x}, \mathbf{x}') = \langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle = \langle k(\mathbf{x}', \cdot), k(\mathbf{x}, \cdot) \rangle = k(\mathbf{x}', \mathbf{x}) \tag{2.11}$$

for $\mathbf{x} \in \mathcal{X}$. It also follows that k is positive semidefinite (Scholkopf & Smola, 2001).

2.3.3 Kernel Density Estimation

One of the many applications kernel functions can be used is in density estimation, where, given a finite sample, one wants to estimate the density of their generating function. This section describes the Kernel Density Estimation, or KDE, a very important method on the conception of this work. KDE is a non-parametric method, meaning that it does not assume the function belongs to some closed family, and its complexity can grow as more data is sampled.

In KDE, translation-invariant kernels, such as Gaussians, are typically used. Given a positive definite kernel k, the normalized kernel is a function with the form:

$$\hat{k}(\mathbf{x}, \mathbf{x}') = \frac{1}{Z} k(\mathbf{x}, \mathbf{x}') \tag{2.12}$$

where

$$Z = \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}')d\mathbf{x}' \tag{2.13}$$

is a normalizing value that guarantees that the normalized kernel integrates to 1 and can then be considered a probability density in $\mathcal{X}$. It is important to note that the normalized kernel is only defined when k is Lebesgue integrable with respect to $\mathbf{x}'$ (and also with respect to $\mathbf{x}$, since k is symmetric):

$$\int_{\mathcal{X}} |k(\mathbf{x}, \mathbf{x}')|d\mathbf{x}' < \infty \tag{2.14}$$

Given a sample $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, one normalized kernel function is defined centered in each point. With N density functions, the final function can be estimated as their mixtures:

$$\hat{f}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \hat{k}(\mathbf{x}_i, \mathbf{x}) \tag{2.15}$$

Density estimation using Gaussian kernels were extensively studied by Silverman (1986). In this case, the normalized kernel is equivalent to a multivariate Gaussian with a diagonal covariance matrix $\Sigma = \sigma I$:

$$\begin{aligned} \hat{k}(\mathbf{x}, \mathbf{x}') &= \frac{1}{\sqrt{2\pi}^d \det(\Sigma)} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Sigma^{-1} (\mathbf{x} - \mathbf{x}')\right) \\ &= \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp\left(-\frac{1}{2}\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right) \end{aligned} \tag{2.16}$$

where d is the dimensionality of the input space $\mathcal{X} = \mathbb{R}^d$.

$\hat{f}$ is then estimated as:

$$\hat{f}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{(\sqrt{2\pi}\sigma)^d} \exp\left(-\frac{1}{2}\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right) \tag{2.17}$$

which can be written in terms of the original Gaussian kernel function:

$$\hat{f}(\mathbf{x}) = \frac{1}{N(\sqrt{2\pi}\sigma)^d} \sum_{i=1}^{N} \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{\sigma^2}\right) = \frac{1}{N(\sqrt{2\pi}\sigma)^d} \sum_{i=1}^{N} k_\sigma(\mathbf{x}, \mathbf{x}_i) \quad (2.18)$$

where k_σ denotes the kernel function with parameter σ.

The choice of σ is crucial to the performance of the estimator. If it is chosen to be too large, the KDE fails to capture the peculiarities of the original function, such as its peaks and valleys. However, a very small σ can also be bad, since one can end up with a function that fails to be smooth even where it should. Figure 2.4 shows four different values of σ in a one-dimensional synthetic dataset which consists of two normal distributions with means 3 and -3 and unitary variance.

To choose σ for univariate problems, Silverman (1986) suggests the following procedure as a general rule:

$$\sigma^* = \left(\frac{4}{3}\right)^{\frac{1}{5}} sN^{-\frac{1}{5}} \quad (2.19)$$

where s is the sample standard deviation and N is the number of points. However, there is no standard rule for general multidimensional problems, and the choice of σ is highly dependent on the problem.

2.3.3.1 Bayesian Classification and the Likelihood Space

One application in which the KDE can be used is in Bayesian classifiers (Duda *et al.*, 1973). These methods are generative models, relying on estimations of the probabilities $P(\mathbf{x}, y)$ to correctly classify the patterns.

The class y is treated as a latent variable. In order to estimate the probability of having a specific class y given an observation $\mathbf{x}$, $P(y|\mathbf{x})$, one can factor it using Bayes' theorem:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}, y)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \quad (2.20)$$

The probabilities $P(\mathbf{x})$ and $P(\mathbf{x}|y)$ can be estimated using KDE. To estimate the former, KDE is executed directly on the whole input data. To the latter,

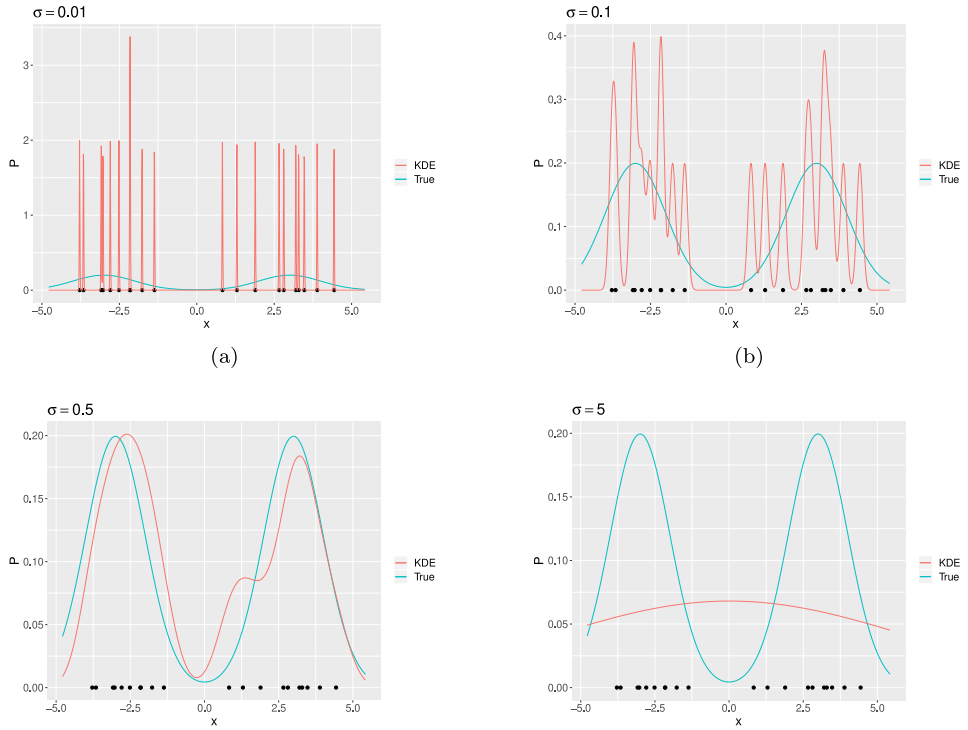
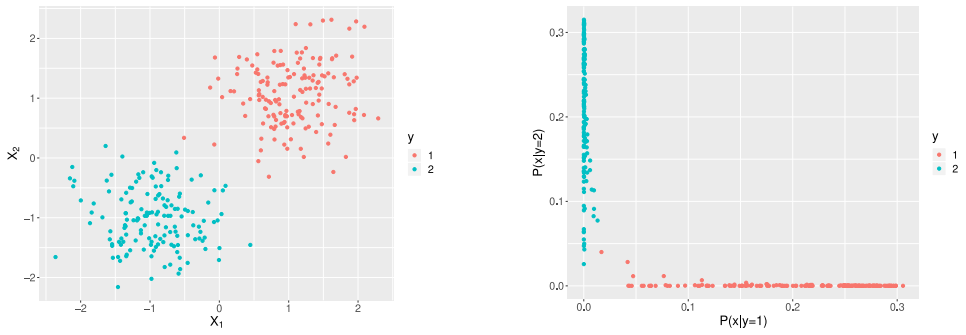


Figure 2.4: Estimation of densities in a synthetic problem using KDE. In 2.4a, σ is very small, and KDE not able to capture nothing from the generating distribution. In 2.4b, the KDE learns the two regions in some sense, but the estimate is still very rough. 2.4c shows an intermediary value of σ that is able to approach the original density, and 2.4d shows the effect of a very large σ, where the estimate is smoother than it should.

one has to estimate one model per class. Nevertheless, the estimation of $P(\mathbf{x})$ is usually not needed, since it is a normalization term shared between all classes. $P(y)$ can be estimated of a given class by simply taking the proportion of that class in the training set.

After $P(\mathbf{x}|y)$ is estimated for each class y, a very useful tool is the **likelihood space**, in which patterns can be mapped. In this space, each axis is the estimated probability of the pattern to a given class. This space can be visualized for a simple two-class problem in Figure 2.5.

With the likelihood space, one can visualize what is happening with the probability estimations and how they affect the classification ability. In this work, this



(a) (b) .

Figure 2.5: Example of similarity mapping using a Gaussian kernel.

same concept is used to define a similar space, in which separability measures in the feature space are computed.

2.3.4 Maximum Mean Discrepancy

In methods that rely on probability functions, such as generative modelling, it is useful to define divergence measures between distributions. Using a divergence, one can, for instance, train a model to maximize or minimize the difference between samples or detect distributional shift (Gama *et al.*, 2014).

A widely used family of divergences is the **ϕ-divergence** (Sriperumbudur *et al.*, 2009). Given two probability measures P and Q defined on a space $\mathcal{X}$, such that P is absolutely continuous with respect to Q, the ϕ-divergence between P and Q is defined as:

$$D_\phi(P, Q) = \int_{\mathcal{X}} \phi\left(\frac{dP}{dQ}\right) dQ \tag{2.21}$$

where $\phi : (0, \infty) \to (-\infty, \infty]$ is a convex function. A widely used instance of this family is the Kullback-Leibler divergence, which is defined with $\phi(x) = x \log x$ (Sriperumbudur *et al.*, 2009).

Another family of divergences between distributions is defined given not only a function, but a set of functions. The **integral probability metric** (IPM) between two probability measures P and Q is defined, in a general form, as:

$$\gamma_{\mathcal{F}}(P, Q) = \sup_{f \in \mathcal{F}} \left| \int_{\mathcal{X}} f dP - \int_{\mathcal{X}} f dQ \right| \tag{2.22}$$

where $\mathcal{F}$ is a space of real-valued, bounded, and measurable functions (Sriperumbudur *et al.*, 2009).

One instance of the IPM family is the **maximum mean discrepancy** (MMD) (Gretton *et al.*, 2012). In the MMD, the distance is defined using expected values of functions over the distributions:

$$MMD[\mathcal{F}, P, Q] = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_{\mathbf{x} \sim P}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{y} \sim Q}[f(\mathbf{y})] \right). \tag{2.23}$$

As a special case, when $\mathcal{F}$ is the unit ball in a reproducing kernel Hilbert space $\mathcal{H}$, the MMD can be rewritten in terms of the kernel k associated to $\mathcal{H}$. Specifically, the squared MMD can be defined as:

$$MMD^2[\mathcal{F}, P, Q] = \mathbb{E}_{\mathbf{x}, \mathbf{x}' \sim P}\left[k(\mathbf{x}, \mathbf{x}')\right] + \mathbb{E}_{\mathbf{y}, \mathbf{y}' \sim Q}\left[k(\mathbf{y}, \mathbf{y}')\right] - 2\mathbb{E}_{\substack{\mathbf{x} \sim P \\ \mathbf{y} \sim Q}}\left[k(\mathbf{x}, \mathbf{y})\right]. \tag{2.24}$$

Applications include a significance test, with which kernel functions can be used to compare two samples (Gretton *et al.*, 2012). Based on that, the MMD can also be used in generative adversarial networks, in which the generator has as an objective to minimize the MMD between the real and synthetic data (Li *et al.*, 2017, 2015).

In this work, a similar kernel-based discrepancy measure between two distributions is presented, which is then used to maximize the distributional discrepancy between different classes.

2.3.5 Support Vector Classification

The support vector machine (SVM) (Cortes & Vapnik, 1995) is possibly the most prominent kernel method. It implements, by design, the structural risk minimization principle. Originally proposed as a form to learn optimal linear classifiers, the SVM uses nonlinear kernel functions in order to implicitly map the points into a reproducing kernel Hilbert space. By solving the linear problem

in the RKHS using a nonlinear map, the SVM is able to learn nonlinear decision functions in the original space. The whole formalization in this section is based on Scholkopf & Smola (2001).

In the primal problem defined in an Euclidean space, the goal is to learn a hyperplane $f(x) = \langle \mathbf{x}, \mathbf{w} \rangle + b$ that separates the input patterns with the largest possible margin. Given a training set with input patterns $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \mathcal{X}$, and classes $\mathbb{Y} = \{y_1, \cdots, y_N\}$, $y_i \in \{-1, 1\}$, $i = 1, \cdots, N$, the hyperplane to be learned is defined to have distance $\frac{1}{\|\mathbf{w}\|}$ to its closest point.

Since the resulting hyperplane does not change if both $\mathbf{w}$ and b are multiplied by the same constant, this function, called the **canonical hyperplane**, is defined in order to rule out the possibility of representing the same linear function with many different sets of parameters. Furthermore, by this definition, the large margin hyperplane can be learned by minimizing $\|\mathbf{w}\|$, since this maximizes the distance of the hyperplane to the closest pattern.

The primal optimization problem is defined as follows:

$$\min_{\mathbf{w} \in \mathcal{X}} \frac{1}{2} \|\mathbf{w}\|^2 , \tag{2.25}$$

$$\text{subject to } y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) \geq 1 \text{ for all } i = 1, \cdots, N. \tag{2.26}$$

The constraints assure that a hyperplane that separates the two classes is achieved, with $sign \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) = y_i$ for all i.

To turn the problem into an unconstrained optimization problem, Lagrange multipliers α, $\alpha_i \geq 0$ for $i = 1, \cdots, N$, are introduced. The objective function turns into the following:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i \left(y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) - 1 \right) . \tag{2.27}$$

which is minimized with respect to $\mathbf{w}$ and b and maximized with respect to α.

By differentiating the Lagrangian with respect to $\mathbf{w}$ and b and setting the derivatives to zero, one reaches the dual formulation of the optimization problem. Training the SVM is equivalent to solving the following problem:

$$\max_{\alpha \in \mathbb{R}^N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.28}$$

subject to

$$\alpha_i \geq 0 \text{ for all } i = 1, \cdots, N, \tag{2.29}$$

and

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{2.30}$$

Using the dual formulation, it is no longer necessary to have explicit hyperplane parameters to solve the maximum margin problem. Furthermore, the input variables are only needed inside the inner product. This formulation of the SVM makes it straightforward to solve also nonlinear problems: by substituting $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ for a nonlinear kernel $k(\mathbf{x}_i, \mathbf{x}_j)$, the patterns are implicitly mapped into a reproducing kernel Hilbert space $\mathcal{H}$, where a linear classification problem can still be solved in the same manner. However, due to the nonlinear map, the linear problem in $\mathcal{H}$ becomes nonlinear in $\mathcal{X}$.

This classification problem, however, is only defined if the patterns are separable. In problems with overlapping classes, the hyperplane that solves the optimization problem does not exist. To use the SVM in such situations, some patterns have to violate the margin. Then, slack variables $\xi_i \geq 0$, $i = 1, \cdots, N$, are used to relax the constraint from Equation 2.26. The primal formulation of the flexible-margin SVM is as follows:

$$\min_{\substack{\mathbf{w} \in \mathcal{X}, \\ \xi \in \mathbb{R}^N}} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{C}{N} \sum_{i=1}^{N} \xi_i, \tag{2.31}$$

subject to

$$y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) \geq 1 - \xi_i \text{ for all } i = 1, \cdots, N, \tag{2.32}$$

and

$$\xi_i \geq 0 \text{ for all } i = 1, \cdots, N \tag{2.33}$$

where C is a predefined parameter which controls the trade-off between maximizing the margin and minimizing the training error.

The dual problem of the flexible-margin SVM, given a kernel k, is finally defined as:

$$\max_{\alpha \in \mathbb{R}^N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{2.34}$$

subject to

$$0 \leq \alpha_i \leq \frac{C}{N} \text{ for all } i = 1, \cdots, N, \tag{2.35}$$

and

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{2.36}$$

2.4 Kernel Interpretation of Neural Networks

Neural networks, especially multilayer perceptrons, are basically compositional functions that can be used to learn a function from an input space $\mathbb{R}^n$ to an output $\mathbb{R}^m$. An MLP with l layers, for instance, combines linear transformations and pointwise nonlinearities in a hierarchical manner, and has the following general form:

$$\phi(\mathbf{x}) = g\left(W_l g\left(\cdots W_2 g\left(W_1 \mathbf{x} + b_1\right) + b_2\right) + b_l\right) \tag{2.37}$$

where W_i is the weight matrix of the i-th layer, b_i is its bias vector, and $g(\cdot)$ a pointwise nonlinear function.

Part of their usefulness comes from their expressive power: a multilayer perceptron with only one hidden layer can approximate any given continuous function arbitrarily well, given enough neurons (Cybenko, 1989). Even so, one can benefit from networks with more than one hidden layer, as they can learn some kinds of data equally well, but with a less complex model (Mhaskar *et al.*, 2016).

In the literature, a number of works combine kernel theory and neural networks. Jacot *et al.* (2018), for instance, presents the Neural Tangent Kernel, which helps shed light on the training dynamics of deep neural networks using stochastic gradient descent. Montavon *et al.* (2011) uses RBF kernels to analyze a deep neural network on each layer. Other works, such as Lee *et al.* (2017)

and Garriga-Alonso *et al.* (2018), show equivalence between deep networks and Gaussian processes, a prominent kernel method.

Here, the neural network is interpreted as part of a kernel. Using the networks to map data into an Euclidean space, the linear inner product in that space is a kernel in input space.

Definition 4. *Let $\phi_\theta : \mathcal{X} \to \mathbb{R}^m$ be a neural network with parameters $\theta \in \Theta$. The neural kernel induced by ϕ_θ is a function $k_\theta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, defined as:*

$$k_\theta(\mathbf{x}, \mathbf{x}') = \langle \phi_\theta(\mathbf{x}), \phi_\theta(\mathbf{x}') \rangle \tag{2.38}$$

One important result to show for this work is that the neural kernel is positive semidefinite. In order to conclude this, it is important to show that the linear kernel is positive semidefinite.

Lemma 1. *Let the input space be a d-dimensional Euclidean space, $\mathcal{X} = \mathbb{R}^d$. Given $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, the linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$ is positive semidefinite.*

Proof. Recalling the definition of a positive semidefinite kernel (Definition 1), one needs to show that

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{2.39}$$

for all $c_i \in \mathbb{R}$ and $\mathbf{x}_i \in \mathbb{R}^d$, $i = 1, \cdots, N$. Regarding the linear kernel, the positive-semidefiniteness specifically implies that

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \geq 0. \tag{2.40}$$

In an Euclidean space of dimensionality d, the inner product can be obtained using Equation 2.41:

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{k=1}^{d} x_i^k x_j^k \tag{2.41}$$

where x^k is the k-th element of $\mathbf{x}$. Then, the following is true:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \sum_{k=1}^{d} x_i^k x_j^k \tag{2.42}$$

and the sums can be rearranged as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \sum_{k=1}^{d} x_i^k x_j^k = \sum_{k=1}^{d}\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j x_i^k x_j^k = \sum_{k=1}^{d}\sum_{i=1}^{N} c_i x_i^k \sum_{j=1}^{N} c_j x_j^k. \tag{2.43}$$

$\sum_{j=1}^{N} c_j x_j^k = \sum_{i=1}^{N} c_i x_i^k$, as i and j are just conventions. Finally:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \sum_{k=1}^{d}\sum_{i=1}^{N} c_i x_i^k \sum_{j=1}^{N} c_j x_j^k = \sum_{k=1}^{d} \left(\sum_{i=1}^{N} c_i x_i^k \right)^2 \geq 0 \tag{2.44}$$

□

Theorem 1. *Given a neural network $\phi_\theta : \mathcal{X} \to \mathbb{R}^m$, the neural kernel induced by ϕ_θ, $k_\theta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, is positive semidefinite.*

Proof. The first point to note is that ϕ_θ necessarily maps input points to an Euclidean space $\mathbb{R}^m$. Lemma 1 stated that the linear kernel is positive semidefinite for any set of points in an Euclidean space. Thus, it follows that the neural kernel is positive semidefinite. □

Using this tool, one can define kernels in any input space which can be used as input by a neural network: $\mathcal{X}$ can be a space where the inner product is not straightforward to compute or even does not exist, such as images, text, and speech.

Chapter 3

Learning Kernel Parameters

In this chapter, two loss functions for learning representations are proposed. As they are entirely based on inner products in the feature space to be learned, they can be seen as acting upon kernel functions.

The first function requires supervision which separates data points into discrete groups, making it suitable for classification problems. The second one is unsupervised and was first proposed as an alternative to the first after an extensive analysis of the Gaussian kernel while varying its parameter σ.

The chapter is organized as follows: first of all, both functions are defined, along with a discussion about they measure. Section 3.4 shows a couple of properties of these functions, which allow them to be optimized by well-known continuous optimization methods. Finally, Section 3.5, analyzes the behavior of both functions regarding the specific kernels used in this work.

3.1 Basic Definitions

Chapter 2.3.3 introduced the visualization of samples using likelihood values estimated by the KDE method. Inspired by this, an analogous mapping to a **similarity space** can be defined, which helps us interpret what the proposed functions learn.

Both kernel-optimizing functions are based on similarity values on the feature space $\mathcal{H}$. These values are obtained, naturally, using the inner product given by the kernel, $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}')$. These functions were first defined and tested

using Gaussian kernels. Because of this, the main examples and interpretations are given using these kernels.

First of all, a similarity measure from a point to a probability distribution in input space is defined.

Definition 5. *The **kernel similarity** $\psi_{k,\theta}(\mathbf{x}, P)$ of a point $\mathbf{x} \in \mathcal{X}$ to a probability distribution P over $\mathcal{X}$, given a kernel k with parameters θ, is given by:*

$$\psi_{k,\theta}(\mathbf{x}, P) = \mathbb{E}_{\mathbf{x}' \sim P}\left[k(\mathbf{x}, \mathbf{x}')\right] \tag{3.1}$$

Given a training set $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ sampled from P, it can be empirically estimated as:

$$\hat{\psi}_{k,\theta}(\mathbf{x}, \mathbb{X}) = \begin{cases} \frac{1}{N-1} \sum_{\substack{i=1 \\ \mathbf{x}_i \neq \mathbf{x}}}^{N} k(\mathbf{x}, \mathbf{x}_i) & \text{if } \mathbf{x} \in \mathbb{X} \\ \frac{1}{N} \sum_{i=1}^{N} k(\mathbf{x}, \mathbf{x}_i) & \text{otherwise} \end{cases} \tag{3.2}$$

Using this kernel similarity, one can compute similarities from points to specific sets, such as all of the samples belonging to a class, or even to the whole training set. Using this measure, it is possible to define more complex similarities and dissimilarities. It is used in definitions of both functions.

3.2 The Kernel Distributional Discrepancy Loss

Using the kernel similarity (Equation 3.1), one can compute how similar is a sample to a specific class, just as the likelihood values obtained using KDE[1] and expressed in Figure 2.4. For a classification problem with d classes, a point $\mathbf{x}$ in input space can be mapped into a similarity space $\mathbb{R}^d$, where each component would be its similarity to each one of the classes C_k, $k = \{1, \cdots, d\}$:

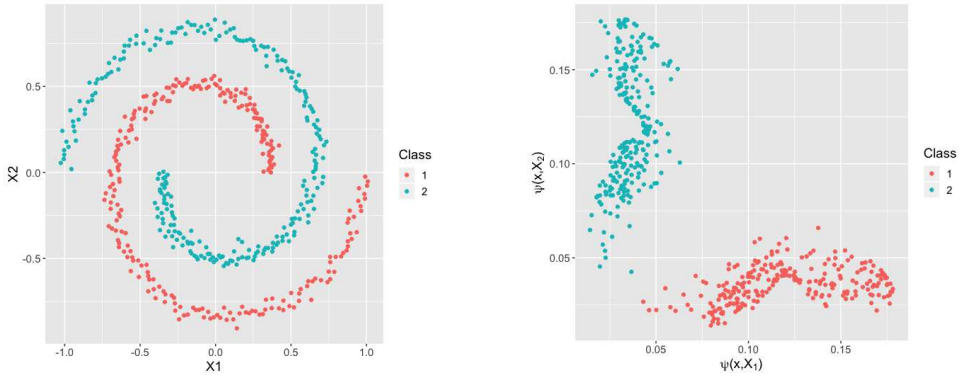
$$\mathbf{x} \rightarrow \left[\psi_{k,\theta}(\mathbf{x}, P_1), \psi_{k,\theta}(\mathbf{x}, P_2) \cdots, \psi_{k,\theta}(\mathbf{x}, P_d)\right]^T \tag{3.3}$$

where $P_i = P(X|Y = C_i)$.

To illustrate, Figure 3.1a shows a two-dimensional space, generated synthetically. This is a space related to a two-class classification setting. Each point is

[1] This similarity, however, does not need to be a probability value.

mapped according to its kernel similarity to both class 1 and 2 using a Gaussian kernel with parameter $\sigma = 0.2$.



(a) Samples in a two-dimensional input space

(b) Mapped samples in similarity space. Each axis represents the similarity of the point to each class.

Figure 3.1: Example of similarity mapping using a Gaussian kernel.

As the training data is generally a fixed sample and the type of kernel is chosen beforehand, this similarity mapping relies solely on the kernel parameters θ. As different values of θ yield different similarity mappings, it is possible, then, to define a function with which the effect of a specific θ on the separability of the classes can be assessed.

Nevertheless, before getting to define the supervised loss function, another definition is needed. Just as the kernel values can be used to measure how similar a point is to a distribution, they can also be used to measure how similar two distributions are with each other:

Definition 6. *The **kernel similarity of distributions** (KSD) $\xi_{k,\theta}(P, Q)$ between two distributions P and Q defined over the same topological space $\mathcal{X}$, given a kernel k with parameters θ, is the expected value of $k(\mathbf{x}, \mathbf{z})$ for $\mathbf{x} \sim P$ and $\mathbf{z} \sim Q$, given by:*

$$\xi_{k,\theta}(P, Q) = \mathbb{E}_{\substack{\mathbf{x}\sim P \\ \mathbf{z}\sim Q}} [k(\mathbf{x}, \mathbf{z})] \tag{3.4}$$

Given $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and $\mathbb{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_M\}$ sampled from P and Q respectively, the KSD can be empirically estimated as:

$$\hat{\xi}_{k,\theta}(\mathbb{X},\mathbb{Z}) = \begin{cases} \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} k(\mathbf{x}_i, \mathbf{z}_j) & \textit{if } \mathbb{X} = \mathbb{Z} \\ \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} k(\mathbf{x}_i, \mathbf{z}_j) & \textit{otherwise} \end{cases} \tag{3.5}$$

The similarity between two classes C_1 and C_2 can be defined by simply computing the KDS of $P(X|Y = C_1)$ and $P(X|Y = C_2)$. With two distributions P and Q, each of them can be defined as a vector in similarity space as follows:

$$V_P = [\xi_{k,\theta}(P,P), \xi_{k,\theta}(P,Q)]^T \tag{3.6}$$

$$V_Q = [\xi_{k,\theta}(Q,P), \xi_{k,\theta}(Q,Q)]^T \tag{3.7}$$

It is important to note that these vectors lie in the same similarity space as the point maps depicted in Figure 3.1b. Each component of this vector is nothing more than the expectation of the points belonging to a specific class with respect to each axis. Thus, in practice, only the midpoints of each class are computed in this space. Figure 3.2 shows the representations of both classes in the similarity space.

The last needed definition before presenting the supervised loss itself is the **kernel distributional discrepancy** between two distributions.

Definition 7. *Given two distributions P and Q defined over an input space $\mathcal{X}$ and a PSD kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with parameters θ, the kernel distributional discrepancy (KDD) between them is given by:*

$$\begin{aligned} \mathcal{D}_{k,\theta}(P,Q) = \big(&(\xi_{k,\theta}(P,P) - \xi_{k,\theta}(P,Q))^2 + \\ &(\xi_{k,\theta}(Q,Q) - \xi_{k,\theta}(Q,P))^2 \big)^{\frac{1}{2}} \end{aligned} \tag{3.8}$$

Substituting the KSD values for its definitions:

$$\begin{aligned} \mathcal{D}_{k,\theta}(P,Q) = \Bigg(&\left(\mathbb{E}_{\substack{\mathbf{x}\sim P \\ \mathbf{x}'\sim P}} [k(\mathbf{x},\mathbf{x}')] - \mathbb{E}_{\substack{\mathbf{x}\sim P \\ \mathbf{z}\sim Q}} [k(\mathbf{x},\mathbf{z})] \right)^2 + \\ &\left(\mathbb{E}_{\substack{\mathbf{z}\sim Q \\ \mathbf{z}'\sim Q}} [k(\mathbf{z},\mathbf{z}')] - \mathbb{E}_{\substack{\mathbf{z}\sim Q \\ \mathbf{x}\sim P}} [k(\mathbf{z},\mathbf{x})] \right)^2 \Bigg)^{\frac{1}{2}} \end{aligned} \tag{3.9}$$

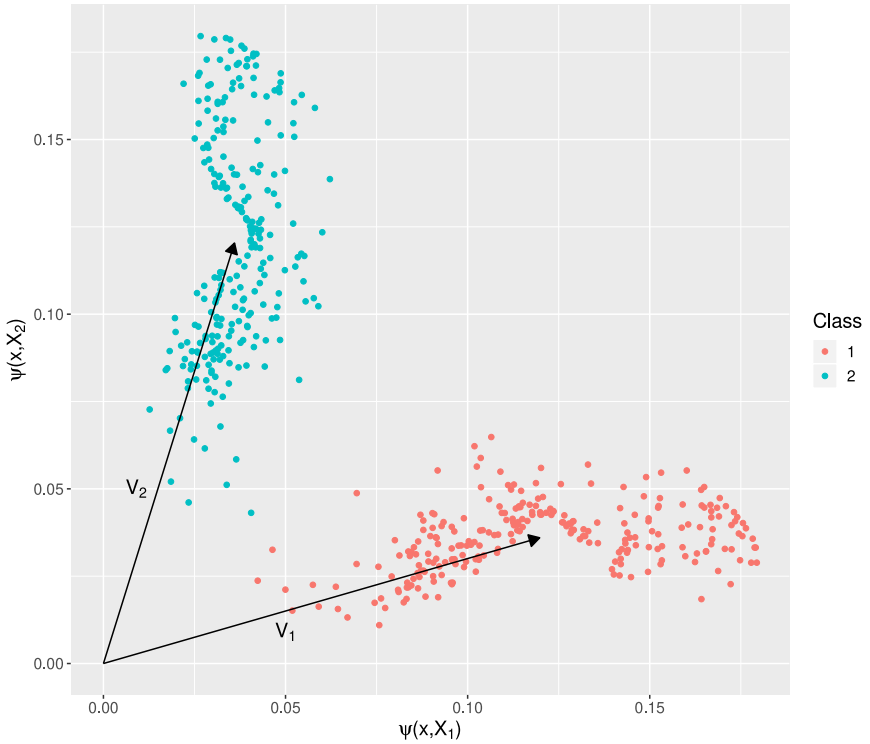


Figure 3.2: Similarity mapping along with the class vectors. The kernel distributional discrepancy is defined as the Euclidean distance between them, $\|V_1 - V_2\|_2$.

An empirical version of the distance can be obtained by using the sample means. If $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and $\mathbb{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_M\}$ are two i.i.d. finite samples from P and Q respectively, the empirical kernel distributional discrepancy is given by:

$$\hat{\mathcal{D}}_{k,\theta}(\mathbb{X}, \mathbb{Z}) = \left(\left(\left[\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} k(\mathbf{x}_i, \mathbf{x}_j) \right] - \left[\frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} k(\mathbf{x}_i, \mathbf{z}_j) \right] \right)^2 + \left(\left[\frac{1}{M(M-1)} \sum_{i=1}^{M} \sum_{\substack{j=1 \\ j \neq i}}^{M} k(\mathbf{z}_i, \mathbf{z}_j) \right] - \left[\frac{1}{NM} \sum_{i=1}^{M} \sum_{j=1}^{N} k(\mathbf{z}_i, \mathbf{x}_j) \right] \right)^2 \right)^{\frac{1}{2}} \tag{3.10}$$

When the probability distributions define classes over the input space, it becomes clear that Equation 3.8 is the Euclidean distance between the vectors given by Equations 3.6 and 3.7.

One interesting result arises when the Euclidean distance is substituted for the Manhattan distance. The distance between P and Q becomes:

$$\mathcal{D}^1_{k,\theta}(P,Q) = |\xi_{k,\theta}(P,P) - \xi_{k,\theta}(P,Q)| + |\xi_{k,\theta}(Q,Q) - \xi_{k,\theta}(Q,P)| \tag{3.11}$$

where the superscript 1 denotes the distance taken with relation to the L_1 norm.

If the kernel k is chosen such that it makes the KSD between a distribution to itself greater than or equal to the KSD between this distribution to another, the absolute values can be dropped, and Equation 3.11 is equivalent to the one-sided discrepancy:

$$\begin{aligned}\mathcal{D}^1_{k,\theta}(P,Q) = \xi_{k,\theta}(P,P) - 2\xi_{k,\theta}(P,Q) + \xi_{k,\theta}(Q,Q) = \\ \mathbb{E}_{\substack{\mathbf{x}\sim P \\ \mathbf{x}'\sim P}}\left[k(\mathbf{x},\mathbf{x}')\right] - 2\mathbb{E}_{\substack{\mathbf{x}\sim P \\ \mathbf{z}\sim Q}}\left[k(\mathbf{x},\mathbf{z})\right] + \mathbb{E}_{\substack{\mathbf{z}\sim Q \\ \mathbf{z}'\sim Q}}\left[k(\mathbf{z},\mathbf{z}')\right].\end{aligned} \tag{3.12}$$

where the squared MMD (Gretton *et al.*, 2012) in unit-norm reproducing kernel Hilbert spaces is retrieved.

In practice, it was usually the case that the KSD between two distinct distributions was lower than the KSD from a distribution to itself, although there exists exceptions in highly imbalanced datasets. In such cases, optimizing 3.11 is the same as optimizing the maximum mean discrepancy between P and Q.

Finally, the first loss function can be defined.

Definition 8. *The Kernel Distributional Discrepancy (KDD) loss function $L_{KDD}(C,k,\theta)$ for a problem with d classes, $C = \{C_1, \cdots, C_d\}$, is given by:*

$$L_{KDD}(C,k,\theta) = -\prod_{i=1}^{d}\prod_{\substack{j=1 \\ j\neq i}}^{d}\mathcal{D}_{k,\theta}(P_i,P_j) \tag{3.13}$$

where $P_i = P(X|Y = C_i)$.

In practice, the function can be estimated using the empirical kernel discrepancies between classes, pairwise. Given a training set $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ and a class set $\mathbb{Y} = \{y_1, \cdots, y_N\}$:

$$\hat{L}_{KDD}(\mathbb{X}, \mathbb{Y}, k, \theta) = -\prod_{i=1}^{d} \prod_{\substack{j=1 \\ j \neq i}}^{d} \hat{\mathcal{D}}_{k,\theta}(\mathbb{X}_i, \mathbb{X}_j) \tag{3.14}$$

where $\mathbb{X}_i$ are the elements of $\mathbb{X}$ belonging to class C_i.

The product of distances is used in order to give a naturally higher weight for lower distances: it is better to increase the distance between classes that are close to each other than increasing, by the same amount, the distance between classes that are further apart.

When using this function in real-world problems, the distance values can get very small if two classes are very close. The optimization procedure, then, becomes prone to numerical issues when optimizing the product between all the pairwise distances. To mitigate this effect, the assumption that no distance between different classes is exactly zero is made. As long as this happens, the loss can be optimized by minimizing the negative of the logarithm of the product instead:

$$\begin{aligned} \min_{\theta} \hat{L}_{KDD}(\mathbb{X}, \mathbb{Y}, k, \theta) = \min_{\theta} -\log \prod_{i=1}^{d} \prod_{\substack{j=1 \\ j \neq i}}^{d} \hat{\mathcal{D}}_{k,\theta}(\mathbb{X}_i, \mathbb{X}_j) = \\ \min_{\theta} -\sum_{i=1}^{d} \sum_{\substack{j=1 \\ j \neq i}}^{d} \log \hat{\mathcal{D}}_{k,\theta}(\mathbb{X}_i, \mathbb{X}_j) \end{aligned} \tag{3.15}$$

When kernel parameters are optimized by minimizing L_{KDD}, the assumption that a class will be more similar to itself than to the others is presumed. Equation 3.8 shows that the situation where the cross-similarities are maximized and

the self-similarities are minimized would still minimize the loss. However, this case is neither expected nor observed.

3.3 The Kernel Similarity Variance Loss

This section describes the unsupervised loss function, which also relies on the kernel similarity described in Definition 5.

With a fixed training set $\mathbb{X}$, each point $\mathbf{x} \in \mathbb{X}$ can be mapped into a scalar representing its similarity to the whole data-generating distribution P: $\psi_{k,\theta}(\mathbf{x}, P)$. Just as points were mapped into a two-dimensional similarity space, this can be seen as a map to the real line $\mathbb{R}$, a one-dimensional space.

Definition 9. *For a probability distribution P defined over $\mathcal{X}$ and a kernel $k : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ with parameters θ, the Kernel Similarity Variance (KSV) loss function $L_{KSV}(k, \theta)$ is the negative variance of the kernel similarities of P to itself:*

$$L_{KSV}(P, k, \theta) = -Var_{\mathbf{x}\sim P}\left[\psi_{k,\theta}(\mathbf{x}, P)\right] = -Var_{\mathbf{x}\sim P}\left[\mathbb{E}_{\mathbf{x}'\sim P}\left[k(\mathbf{x}, \mathbf{x}')\right]\right] \tag{3.16}$$

Given a training set $\mathbb{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, the empirical version of the unsupervised inner product loss can be defined as follows:

$$\hat{L}_{KSV}(\mathbb{X}, k, \theta) = -\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{1}{N-1}\sum_{\substack{j=1\\ j\neq i}}^{N} k(\mathbf{x}_i, \mathbf{x}_j) - \mu\right)^2 \tag{3.17}$$

where μ is the sample mean for the empirical kernel similarities:

$$\mu = \frac{1}{N(N-1)}\sum_{i'=1}^{N}\sum_{\substack{j'=1\\ j'\neq i'}}^{N} k(\mathbf{x}_{i'}, \mathbf{x}_{j'}) \tag{3.18}$$

This function was first proposed for the Gaussian kernel after the observation that, when σ gets very close to zero, the Gram matrix for that kernel gets closer to an identity matrix (provided that no two points in the training set are equal), in which the kernel value of a point to itself will be 1 and zero to all of the

other points. As the kernel values of points to themselves are ignored in order to have an unbiased estimator, the kernel similarities in this situation would be zero regardless of the input, and the variance is zero.

On the other hand, when the value of σ goes to infinity, the kernel value between any two samples gets closer and closer to 1. Then, in this limit, the average kernel value will be 1 for all samples, which also yields zero variance. Figure 3.3 shows what happens with the scalar projections of samples from the dataset shown in Figure 3.1a with four values of σ.

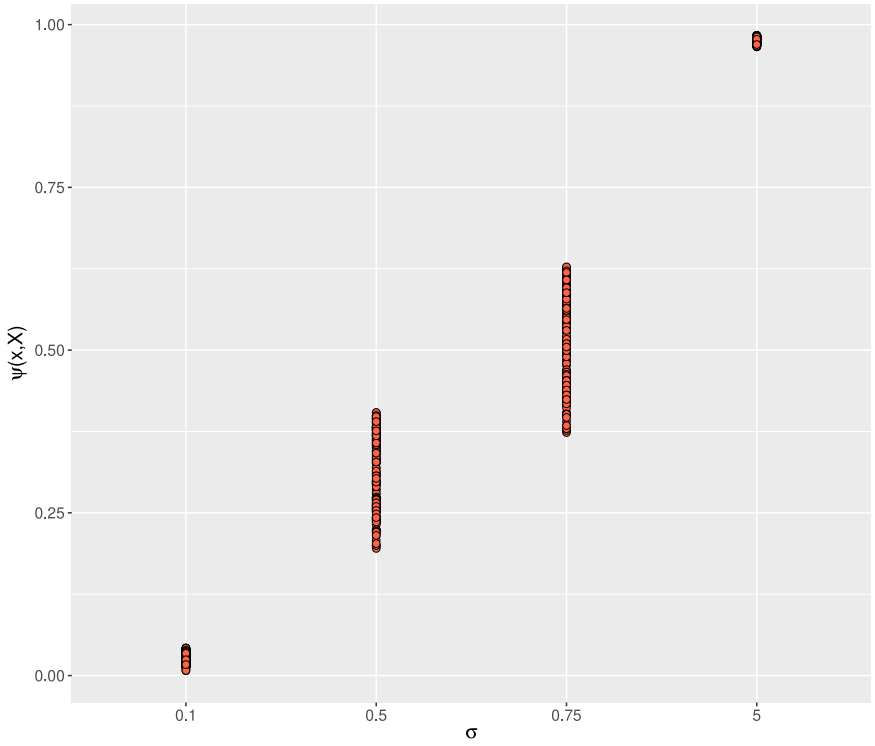


Figure 3.3: The unidimensional similarity map using a Gaussian kernel. The variance is maximized in intermediate values of σ.

The variance is maximized in intermediate σ values, which gives a minimum for the loss function. The intuition behind the use of the variance on the projections is to search for a more **descriptive** Hilbert space, where some samples are more similar to the whole data, while some are less. Using the kernel similarity using Gaussian kernels as a way to approximate local density, the representation is expected to capture the peaks and valleys of the generator distribution.

In the next chapter, it is shown that this yields satisfactory results when using some specific types of kernel, possible reasons are discussed.

3.4 Continuity and Boundedness

For classic optimization algorithms to be used on the proposed functions, some guarantees are needed. Here two of them are considered: **continuity** and **boundedness**. The first one tells whether one can safely search the parameter space using continuous methods. The second tells whether one can reach, or at least approximate, a point where this function reaches a minimum. This is important to be sure that, for instance, the function can not be minimized indefinitely to $-\infty$.

These properties are discussed separately for both functions and required characteristics for the kernel function are given.

3.4.1 KDD Loss

First of all, these characteristics are discussed regarding the supervised function. Under certain circumstances that are not hard to obtain, said function could be expected to reach, or at lest approximate, a minimum value in parameter space.

Theorem 2. *Given a kernel k with parameters $\theta \in \Theta$, the KDD loss $L_{KDD}(C, k, \theta)$ is continuous with respect to parameters θ if k is also continuous in parameter space Θ.*

Proof. The first point to note is that the operations involved in both kernel similarity (Equation 3.1) and kernel similarity for distributions (Equation 3.4) are purely continuous, which means that these values are continuous on the kernel values.

The KDD (Equation 3.8) is continuous on the KSD values. Then, the supervised loss is also continuous on them. If the kernel is continuous on its parameters, it can be seen that the supervised loss consists of a series of continuous operations from parameter space. Then, L_{KDD} is continuous in Θ.

□

Furthermore, all the operations done after the kernel are differentiable. Then, if the kernel is itself differentiable on its parameters, first-order methods such as gradient descent (Goodfellow *et al.*, 2016) can be used to search for minima.

Now the boundedness of the function is addressed.

Theorem 3. *For a d-class classification problem, if a kernel k_θ defines a feature map $\phi_\theta : \mathcal{X} \to \mathcal{H}$ that has its image inside the closed r-ball, i.e. $\|\phi_\theta(\mathbf{x})\| \leq r$ $\forall \mathbf{x} \in \mathcal{X}, \theta \in \Theta$, the values of the KDD loss function L_{KDD} lie inside the interval $[-\left(2\sqrt{2}r^2\right)^{d(d-1)}, 0]$.*

Proof. A Hilbert space with inner product $\langle \mathbf{x}, \mathbf{x}' \rangle_{\mathcal{H}}$ induces a norm $\|\mathbf{x}\|_{\mathcal{H}} = \langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{H}}^{\frac{1}{2}}$. For any reproducing kernel Hilbert space, the norm of $\phi_\theta(\mathbf{x})$ can be computed by Equation 3.19:

$$\|\phi_\theta(\mathbf{x})\| = k(\mathbf{x}, \mathbf{x})^{\frac{1}{2}} \tag{3.19}$$

If the norm of any vector in the RKHS induced by k is bounded, the inner product between any pair of vectors in that space will be bounded as well. The Cauchy-Schwarz inequality for kernels is needed. It can be stated as (Scholkopf & Smola, 2001):

$$k_\theta(\mathbf{x}, \mathbf{x}')^2 \leq k_\theta(\mathbf{x}, \mathbf{x}) k_\theta(\mathbf{x}', \mathbf{x}'). \tag{3.20}$$

Substituting Equation 3.19 into the inequality:

$$k_\theta(\mathbf{x}, \mathbf{x}')^2 \leq \|\phi_\theta(\mathbf{x})\|^2 \|\phi_\theta(\mathbf{x}')\|^2. \tag{3.21}$$

If the norm is at most r, the squared inner product is bounded:

$$k_\theta(\mathbf{x}, \mathbf{x}')^2 \leq r^2 r^2 \to |k_\theta(\mathbf{x}, \mathbf{x}')| \leq r^2 \tag{3.22}$$

If the inner product between any two vectors is bounded between r^2 and $-r^2$, both Equations 3.1 and 3.4 yield a value between $-r^2$ and r^2 for any inputs, since they are simple averages. Then, the vector describing a class (Equations 3.6 and 3.7) has components ranging between these two extrema and lies inside the closed square centered in the origin and with side length $2r^2$.

Any two vectors inside this square can have a distance of at most its diagonal, which is the Euclidean distance between the two extreme vectors $[r^2, r^2]^T$ and $[-r^2, -r^2]^T$:

$$d([r^2, r^2]^T, [-r^2, -r^2]^T) = \sqrt{2(r^2 - (-r^2))^2} = 2\sqrt{2}r^2 \tag{3.23}$$

Then, $\mathcal{D}_{k,\theta}(P, Q) \leq 2\sqrt{2}r^2$ regardless of P and Q.

The KKD loss is the product of all pairwise discrepancies between all of the d class-conditional distributions, excluding discrepancies from classes to themselves:

$$L_{KDD}(C, k, \theta) = -\prod_{i=1}^{d}\prod_{\substack{j=1 \\ j\neq i}}^{d} \mathcal{D}_{k,\theta}(P_i, P_j) \geq -\prod_{i=1}^{d}\prod_{\substack{j=1 \\ j\neq i}}^{d} 2\sqrt{2}r^2 = -\left(2\sqrt{2}r^2\right)^{d(d-1)} \tag{3.24}$$

which proves the lower bound.

The upper bound is trivial: by definition, the lowest possible value of the kernel distributional discrepancy between two distributions is 0. Then, the highest possible value for the KDD loss is zero, as it is the negative of the product of multiple kernel discrepancies.

□

Under some types of kernels, the function can never be minimized nor maximized to infinity. This does not mean, however, that a single point of minimum can be always found, since the parameter space may not be (and it usually is not) compact: it may be the case that the minima values can be arbitrarily approximated as the parameters go to infinity, but no specific finite parameter value will achieve it.

3.4.2 KSV Loss

The unsupervised loss is now analyzed. The analysis is very similar to the one done with the supervised variation, as are the conclusions.

Theorem 4. *Given a kernel k with parameters $\theta \in \Theta$, the unsupervised inner product loss $L_{KSV}(P, k, \theta)$ is continuous with respect to parameters θ if k is also continuous in Θ.*

Proof. This proof is as straightforward as the proof of Theorem 2. As noted, the kernel similarity (Equation 3.1) is a continuous operation in the kernel values.

Once taken the kernel similarity values of all the training points, one can see that the loss function consists simply in taking the variance over these values, which is itself a continuous operator. Then, if k is continuous on Θ, L_{KSV} consists of a series of continuous operators on Θ and the loss is continuous on it as well. □

Theorem 5. *If a kernel k_θ is defines a feature map $\phi_\theta : \mathcal{X} \to \mathcal{H}$ that has its image inside the closed r-ball, i.e. $\|\phi_\theta(\mathbf{x})\| \leq r \ \forall \mathbf{x} \in \mathcal{X}, \theta \in \Theta$, the values of the KSV loss function L_{KSV} lie inside the interval $[-\frac{1}{2}r^2, 0]$.*

Proof. This proof follows a similar path than the one for Theorem 3. As already shown, if $\|\phi_\theta(\mathbf{x})\| \leq r$, the kernel values necessarily lie between $-r^2$ and r^2. The kernel similarity of any sample to the whole training set would necessarily lie in this same interval.

Once the values $\psi_{k,\theta}$ are obtained for each point, the only operation performed is the variance of these values. Popoviciu's inequality on variances (Popoviciu, 1935) states that, if a random variable X has its support inside the bounded interval $[m, M]$, its variance is bounded by above:

$$Var[X] \leq \frac{1}{4}(M - m) \tag{3.25}$$

In Equation 3.16, the kernel similarity is a random variable, since the input vector is itself a random variable X:

$$\psi_{k,\theta}(X, P) = \mathbb{E}_{X' \sim P}\left[k(X, X')\right] \tag{3.26}$$

Then, Equation 3.25 can be used to derive a lower bound on the loss function value. Substituting its limits:

$$L_{KSV}(P, k, \theta) = -Var[\psi_{k,\theta}] \geq -\frac{1}{4}(r^2 - (-r^2)) = -\frac{1}{2}r^2 \tag{3.27}$$

Once again, the upper bound is trivial, since the variance is non-negative by definition.

□

3.5 Analysis in Specific Kernels

Here an analysis of the behavior of the functions on the specific kernels used in this work is conducted. These kernels were chosen because all their parameters are continuous.

3.5.1 Gaussian and Laplacian

Gaussian and Laplacian kernels are **translation invariant**. This means that the value of the kernel between two vectors $k(\mathbf{x}, \mathbf{x}')$ will not be changed if these vectors are translated by the same quantity: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} + \delta, \mathbf{x}' + \delta)\ \forall \delta \in \mathcal{X}$.

Equations 3.28 and 3.29 give the expressions for Gaussian and Laplacian kernels, respectively.

$$k_\sigma(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right) \tag{3.28}$$

$$k_\sigma(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{\sigma}\right) \tag{3.29}$$

In the case of these two functions, this happens because the kernels are merely functions of the distance between patterns and do not take into account the absolute value of the vectors. $k(\mathbf{x}, \mathbf{x}')$ can be seen as a function centered in $\mathbf{x}'$, and all $\mathbf{x}$ located in the sphere of radius r will have the same function value. That is why these two kernels can be specifically called radial basis function (RBF) kernels. Hereafter, this work refers to both Gaussian and Laplacian kernels as RBF.

Since close patterns will have a higher similarity value than further away patterns, these functions tend to capture local information.

Both the kernels take the same parameter σ, which can range inside the **open** interval $(0, \infty)$, and the functions are optimized accordingly inside it. The kernels are undefined for $\sigma = 0$.

Both functions are continuous on σ. Furthermore, it is clear to see that the reproducing kernel Hilbert space induced by them lies inside the unit ball:

$\|\phi_\sigma(\mathbf{x})\| = k_\sigma(\mathbf{x},\mathbf{x})^{\frac{1}{2}} = 1$ for every $\mathbf{x} \in \mathcal{X}$. This means that Theorems 2, 3, 4 and 5 hold.

Another important remark is that these functions converge to specific values in the limits of the parameter space:

$$\lim_{\sigma\to 0} k_\sigma(\mathbf{x},\mathbf{x}') = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{x}' \\ 0, & \text{otherwise} \end{cases} \tag{3.30}$$

$$\lim_{\sigma\to\infty} k_\sigma(\mathbf{x},\mathbf{x}') = 1 \tag{3.31}$$

This means that the limits of the similarity measures proposed here are very well defined. First, this is shown regarding the empirical kernel similarity.

It is assumed that no two elements are the same in the training set. This means that, given a point $\mathbf{x}$ and a training set $\mathbb{X}$, $\mathbf{x}$ appears at most once in $\mathbb{X}$. If $\mathbb{X}$ has size N, it is clear to see that Equation 3.32 holds.

$$\lim_{\sigma\to 0} \hat{\psi}_{k,\sigma}(\mathbf{x},\mathbb{X}) = 0 \tag{3.32}$$

Regarding the infinite-width limit, any two samples have similarity 1, such that:

$$\lim_{\sigma\to\infty} \hat{\psi}_{k,\sigma}(\mathbf{x},\mathbb{X}) = 1 \tag{3.33}$$

for any pattern $\mathbf{x}$ and set X.

It is possible now to analyze what happens with the empirical KSV loss, $\hat{L}_{KSV}$, in the same limits in RBF kernels. As the similarity of every pattern to the training set, excluding to themselves, the similarity of any point to the whole set will converge to 0 when σ approaches 0. Then, it becomes clear that:

$$\lim_{\sigma\to 0} \hat{L}_{KSV}(\mathbb{X},k,\sigma) = -\frac{1}{N}\sum_{i=1}^{N}(0-0)^2 = 0 \tag{3.34}$$

for any training set $\mathbb{X}$.

Something similar happens in the opposite limit. As $\sigma \to \infty$, the similarity of any pattern to the training set tends to 1. Thus:

$$\lim_{\sigma \to \infty} \hat{L}_{KSV}(\mathbb{X}, k, \sigma) = -\frac{1}{N} \sum_{i=1}^{N} (1-1)^2 = 0 \tag{3.35}$$

for any training set $\mathbb{X}$. In both limits of the parameter space, the unsupervised function takes its upper bound. There are some settings in which the value of the function will stay zero for all σ: if all the N data points are supported in the vertices of a standard simplex in $\mathbb{R}^{N-1}$ (e.g. an equilateral triangle in $\mathbb{R}^2$), the average kernel for each point will always have the same value. In practice, however, it is observed that the variance is positive for any intermediate σ, making the function minimizable. Figures 3.4 and 3.5 show the behavior of the negative KSV loss for the spirals problem of Figure 3.1a in Gaussian and Laplacian kernels, respectively.

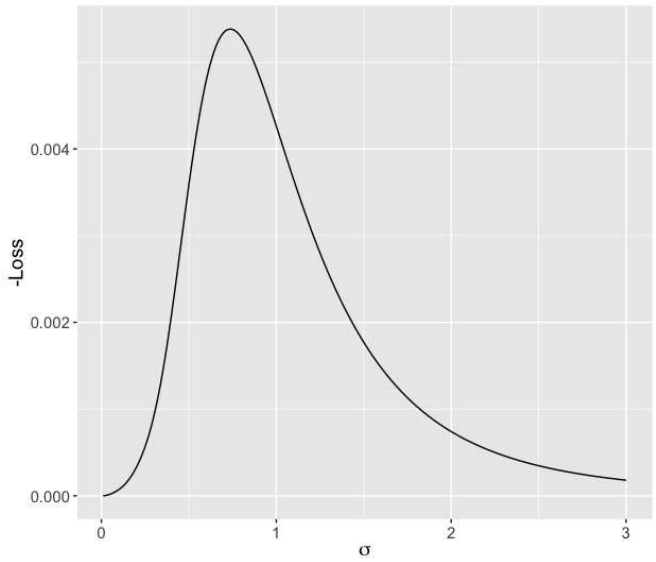


Figure 3.4: Negative KSV loss function behavior for the Gaussian kernel.

To analyze the supervised function in these kernels, the KSD has to be analyzed on the limits. The assumption that each pattern $\mathbf{x}$ appears at most once in the training set is kept.

The empirical KDD between two sets is analyzed.

When $\sigma \to 0$, it is already shown that $\psi_{k,\sigma}(\mathbf{x}, \mathbb{X})$ is 0. If $\psi_{k,\sigma}(\mathbf{x}, \mathbb{X})$ for every $\mathbf{x} \in \mathbb{X}$ is zero, every pattern will be mapped into the zero vector in similarity

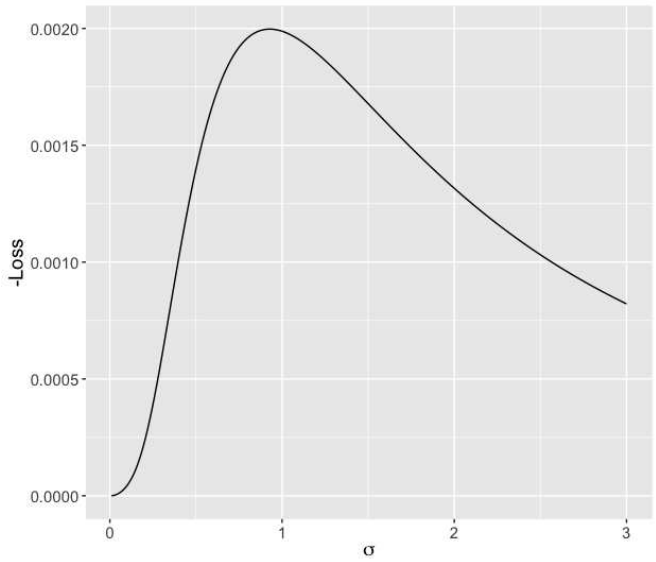


Figure 3.5: Negative KSV loss function behavior for the Laplacian kernel.

space, and the KSD will be zero as well.

When $\sigma \to \infty$, $\psi_{k,\sigma}(\mathbf{x}, \mathbb{X})$ takes the value of 1 regardless of $\mathbf{x}$ and $\mathbb{X}$. In this case, all the class vectors V_i will have ones in all of their entries and will converge to a single point, letting the distances go to zero.

In both limits, the kernel discrepancies will approach zero. Generally, the function is minimized between these extremes. Figures 3.6 and 3.7 show the behavior of the negative supervised loss L_{KDD} when varying σ in the dataset shown in Figure 3.1a, for Gaussian and Laplacian kernels, respectively.

3.5.2 Sigmoidal

The sigmoidal kernel is very popular in practice. An SVM with the sigmoidal kernel is equivalent to a neural network with tanh activation functions (Vapnik, 1995). It is expressed in Equation 3.36:

$$k_\theta(\mathbf{x}, \mathbf{x}') = \tanh\left(\gamma\langle \mathbf{x}, \mathbf{x}'\rangle + b\right) \tag{3.36}$$

for $\theta = \{\gamma, b\}$ and $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$.

This kernel does not have the same properties as the RBF ones. It is not

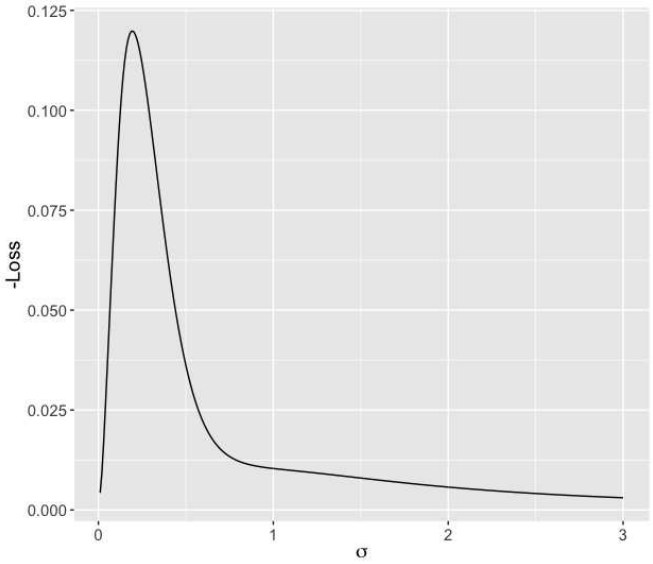


Figure 3.6: Negative KDD loss function behavior for the Gaussian kernel.

translation-invariant, as it is a function of the inner product in input space, which is dependent on the norms of $\mathbf{x}$ and $\mathbf{x}'$.

Furthermore, this kernel is not always positive semidefinite (PSD) (Burges, 1998). The Gram matrix induced by it is PSD only for some choices of parameters

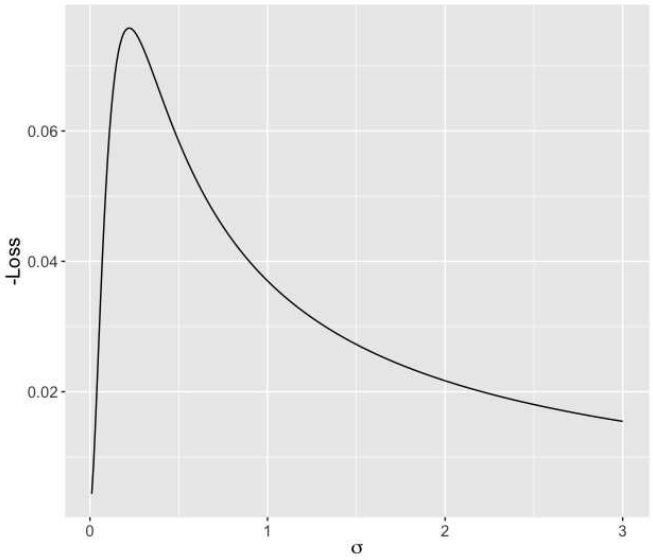


Figure 3.7: Negative KDD loss function behavior for the Laplacian kernel.

γ and b. It also depends on the data.

This is a very important result, since the reproducing kernel map is dependent on the positive semidefiniteness. Burges (1998) states that $\gamma \geq 0$ and $b \geq 0$ are necessary, but not sufficient, conditions for the sigmoidal kernel to be PSD. Then, the search is limited to non-negative parameters. The effect of (conditional) positive semi-definiteness was not explored in this work.

The functions are suitable to optimize the sigmoidal kernel parameters, since it is continuous in them and the output space is bounded ($-1 < k_\theta(\mathbf{x}, \mathbf{x}) < 1 \; \forall \mathbf{x}$). For the problem depicted in Figure 3.1a, the functions are computed for a large range of pairs (γ, b). The results of negative loss are in Figures 3.8 and 3.9.

When optimizing both parameters jointly, both functions seem not to have a single minimum value, and the parameters tend to infinity. However, in practice, this maximum can be arbitrarily approximated, and optimization algorithms tend to stop when the norm of the gradient, $\|\nabla_\theta L\|$, gets small enough.

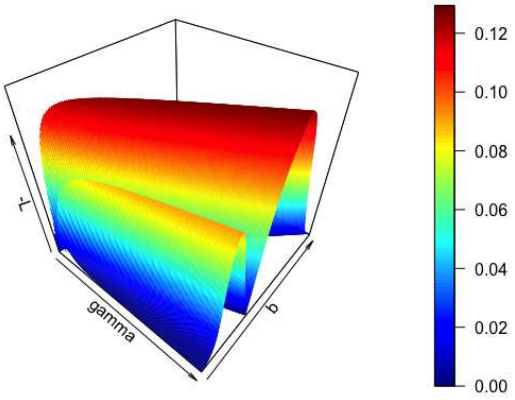


Figure 3.8: Negative KDD loss function surface for γ and b in the sigmoidal kernel.

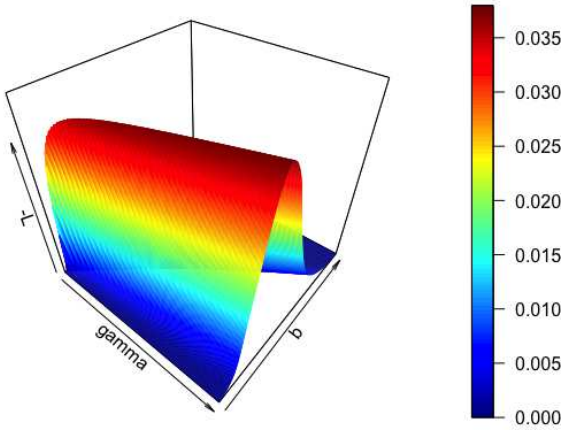


Figure 3.9: Negative KSV loss function surface for γ and b in the sigmoidal kernel.

3.5.3 Neural Networks

As shown in Section 2.4, inner products between outputs of neural networks are PSD kernels. For the functions to be suitable for training the networks with methods such as SGD, there are two points to assure. First of all, that the kernel is not only continuous, but differentiable. The Euclidean inner product, the kernel similarities, their variances, and distances are, indeed, differentiable, so are multilayer perceptrons with differentiable activation functions. Thus, the gradient of the network weights with respect to the inner product losses are computable.

The boundedness of the outputs' norms also need to be guaranteed to avoid the functions to be optimized towards infinity. In order to do this, one only use the logistic pointwise activation function in the output. By doing this, the magnitude of the real numbers that constitute the output vectors are limited, consequently limiting their norms.

3.5.3.1 Output Normalization

The output space of the network is an Euclidean vector space: $\mathcal{H} = \mathbb{R}^d$. In this space, one can compute the distance between two points using their norms and inner product, shown in Equation 3.37 below.

$$d(\mathbf{x}, \mathbf{x}') = (\|\mathbf{x}\| + \|\mathbf{x}'\| - 2\langle \mathbf{x}, \mathbf{x}' \rangle)^{\frac{1}{2}} . \tag{3.37}$$

In output space $\mathcal{H}$, it becomes:

$$d(\phi_\theta(\mathbf{x}), \phi_\theta(\mathbf{x}')) = (\|\phi_\theta(\mathbf{x})\| + \|\phi_\theta(\mathbf{x}')\| - 2\langle \phi_\theta(\mathbf{x}), \phi_\theta(\mathbf{x}') \rangle)^{\frac{1}{2}} . \tag{3.38}$$

The Euclidean distance is increasing with respect to the norms and decreasing with respect to the inner product. The kernel function is expected to indicate distance information: the closer they are in output space, the higher the kernel value $k_\theta(\mathbf{x}, \mathbf{x}') = \langle \phi_\theta(\mathbf{x}), \phi_\theta(\mathbf{x}') \rangle$. The output vectors are then normalized to have unit norm:

$$\mathbf{h} = \phi_\theta(\mathbf{x}) = \frac{\hat{\mathbf{h}}}{\left\| \hat{\mathbf{h}} \right\|} \tag{3.39}$$

where $\hat{\mathbf{h}}$ is the network's output right before normalization.

One effect of the normalization layer would be to introduce a discontinuity when $\|h\| = 0$, as the function would become undefined. This possibility is eliminated by using the logistic activation function before the normalization step, as the values of the components of h would never be zero. Then, the continuity on the parameters is assured, since no output vector would ever be mapped to the zero vector.

Chapter 4

Experimental Results

This chapter presents the experiments and results with the two proposed functions. Two main experiments were conducted, both in classification problems.

In the experiment described in Section 4.1, Support Vector Machines were used to compare the performance of all the four types of kernels used in this work: Gaussian, Laplacian, sigmoidal, and the Multi-Layer Perceptron. Section 4.2 focuses solely on the MLP kernels, comparing them to ordinarily-trained MLPs. Section 4.4 discusses the results.

The experiments were made using Python's Scikit-Learn package (Pedregosa *et al.*, 2011), which is based on the LIBSVM implementation of support vector machines (Chang & Lin, 2011). The neural networks were implemented using PyTorch (Paszke *et al.*, 2017).

4.1 Comparing Kernels in SVMs

The first experiment uses Support Vector Machines to assess the classification performances of the four kernel families trained with each function proposed here.

Test were performed using 18 real-world datasets consisting of binary classification settings, most of which were obtained from the UCI Machine Learning Repository (Lichman, 2013). The exceptions are: "Appendicitis" was obtained from the KEEL datasets repository (Alcalá-Fdez *et al.*, 2011); "Breast Cancer Hess Probes" (**breastHess**) was obtained from Hess *et al.* (2006); and the dataset

"Gene expression dataset" (**golub**) was obtained from Golub *et al.* (1999). Originally, the dataset "Glass Identification" (**glass**) has seven distinct classes. It was treated as a one-versus-all problem, as done in Castro & Braga (2013).

All the datasets went through the same preprocessing steps: remotion of rows with missing data and normalization of the columns to zero mean and unit variance.

All four kernels had their parameters chosen using as criteria the KDD and the KSV losses. It's worth reminding that the KDD for binary problems boils down to kernel distance (Equation 3.8) maximization between the two classes. MMD maximization was also used as a criterion to compare the results. For the Gaussian, Laplacian, and sigmoidal kernels, tests were made choosing their parameters using grid-search with five-fold cross-validation to estimate scores. The search spaces are defined in Table 4.1.

Table 4.1: Hyperparameter ranges for grid search

Kernel	Parameters	Search Space
Gaussian	σ	$\sigma = \{2^{-15}, \cdots, 2^{4}\}$
Laplacian	σ	$\sigma = \{2^{-15}, \cdots, 2^{4}\}$
Sigmoidal	γ, b	$\gamma = \{2^{-15}, \cdots, 2^{4}\}$ $b = \{2^{-15}, \cdots, 2^{4}\}$

The capacity control parameter C has to be defined as well. For the grid search setting, the grid consisted of the Cartesian product between the kernel parameters search spaces and the C search space. For the experiments where the kernel-specific parameters were chosen using the proposed functions or MMD, C was chosen using grid-search, once the kernel parameters were already determined. For all cases, C was searched between $\{2^{-5}, \cdots, 2^{16}\}$.

Only one neural network architecture is used on all tests, regardless of the dataset. It is a multilayer perceptron with two layers: the hidden one with 20 units and the output with 50. A linear SVM was then executed in this output space. As it is completely infeasible to choose the dozens of weights of a neural network using grid search and cross-validation, the next section is dedicated to comparing with a regularly-trained neural network.

Accuracy results, separated by kernel type, are presented on Tables 4.2 to 4.5. The area below the ROC curve, commonly known as AUC (Area Under Curve), was also used as a comparison metric. These results are on Tables 4.6 to 4.9. Each experiment was executed 10 times for each dataset using cross-validation. The confidence intervals are for a significance of 0.05.

Table 4.2: Accuracy using the Gaussian kernel

Basename	Gaussian-Grid Search	Gaussian-MMD	Gaussian-KDD	Gaussian-KSV
fertility	0.870 ± 0.035	0.880 ± 0.030	0.880 ± 0.030	0.880 ± 0.030
appendicitis	0.885 ± 0.077	0.876 ± 0.079	0.876 ± 0.079	0.876 ± 0.079
australian	0.867 ± 0.037	0.861 ± 0.038	0.861 ± 0.033	0.859 ± 0.032
german	0.762 ± 0.031	0.766 ± 0.030	0.760 ± 0.028	0.766 ± 0.031
golub	0.821 ± 0.042	0.809 ± 0.060	0.795 ± 0.062	0.723 ± 0.158
banknote	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
glass	0.972 ± 0.017	0.963 ± 0.014	0.963 ± 0.014	0.972 ± 0.017
ILPD	0.701 ± 0.032	0.712 ± 0.008	0.715 ± 0.006	0.715 ± 0.006
haberman	0.729 ± 0.043	0.735 ± 0.048	0.725 ± 0.048	0.722 ± 0.041
sonar	0.855 ± 0.057	0.870 ± 0.064	0.870 ± 0.044	0.879 ± 0.047
breastHess	0.805 ± 0.063	0.835 ± 0.066	0.827 ± 0.062	0.812 ± 0.058
breastcancer	0.968 ± 0.014	0.963 ± 0.016	0.962 ± 0.015	0.966 ± 0.017
parkinsons	0.933 ± 0.054	0.908 ± 0.037	0.902 ± 0.040	0.887 ± 0.042
heart	0.826 ± 0.042	0.844 ± 0.035	0.844 ± 0.035	0.848 ± 0.038
climate	0.954 ± 0.013	0.950 ± 0.013	0.950 ± 0.013	0.957 ± 0.011
diabetes	0.769 ± 0.025	0.772 ± 0.022	0.773 ± 0.020	0.775 ± 0.020
ionosphere	0.934 ± 0.031	0.952 ± 0.024	0.946 ± 0.020	0.937 ± 0.029
bupa	0.715 ± 0.045	0.684 ± 0.057	0.698 ± 0.047	0.707 ± 0.046
average	0.854 ± 0.016	0.854 ± 0.016	0.853 ± 0.015	0.849 ± 0.017

Table 4.3: Accuracy using the Laplacian kernel

Basename	Laplacian-Grid Search	Laplacian-MMD	Laplacian-KDD	Laplacian-KSV
fertility	0.880 ± 0.030	0.880 ± 0.030	0.880 ± 0.030	0.880 ± 0.030
appendicitis	0.838 ± 0.065	0.876 ± 0.072	0.876 ± 0.072	0.876 ± 0.072
australian	0.854 ± 0.044	0.864 ± 0.042	0.865 ± 0.044	0.867 ± 0.041
german	0.719 ± 0.011	0.768 ± 0.021	0.768 ± 0.021	0.766 ± 0.021
golub	0.654 ± 0.048	0.795 ± 0.062	0.795 ± 0.062	0.795 ± 0.062
banknote	0.999 ± 0.002	0.999 ± 0.002	0.999 ± 0.002	1.000 ± 0.000
glass	0.963 ± 0.014	0.967 ± 0.016	0.967 ± 0.016	0.963 ± 0.014
ILPD	0.712 ± 0.011	0.706 ± 0.017	0.706 ± 0.022	0.710 ± 0.017
haberman	0.719 ± 0.036	0.725 ± 0.031	0.725 ± 0.031	0.725 ± 0.033
sonar	0.620 ± 0.032	0.822 ± 0.067	0.817 ± 0.072	0.817 ± 0.072
breastHess	0.746 ± 0.048	0.812 ± 0.069	0.812 ± 0.069	0.812 ± 0.069
breastcancer	0.963 ± 0.016	0.963 ± 0.015	0.963 ± 0.017	0.963 ± 0.017
parkinsons	0.907 ± 0.039	0.943 ± 0.047	0.943 ± 0.047	0.943 ± 0.047
heart	0.837 ± 0.047	0.833 ± 0.034	0.833 ± 0.034	0.837 ± 0.036
climate	0.915 ± 0.007	0.944 ± 0.009	0.944 ± 0.009	0.944 ± 0.009
diabetes	0.751 ± 0.027	0.769 ± 0.029	0.775 ± 0.022	0.772 ± 0.023
ionosphere	0.929 ± 0.022	0.946 ± 0.020	0.943 ± 0.019	0.943 ± 0.019
bupa	0.675 ± 0.048	0.698 ± 0.040	0.696 ± 0.035	0.698 ± 0.048
average	0.816 ± 0.018	0.851 ± 0.016	0.851 ± 0.016	0.851 ± 0.016

Table 4.4: Accuracy using the sigmoidal kernel

Basename	Sigmoidal-Grid Search	Sigmoidal-MMD	Sigmoidal-KDD	Sigmoidal-KSV
fertility	0.870 ± 0.035	0.880 ± 0.030	0.870 ± 0.035	0.880 ± 0.030
appendicitis	0.905 ± 0.065	0.781 ± 0.077	0.772 ± 0.071	0.820 ± 0.049
australian	0.865 ± 0.043	0.848 ± 0.036	0.846 ± 0.030	0.772 ± 0.041
german	0.760 ± 0.031	0.709 ± 0.026	0.708 ± 0.021	0.666 ± 0.014
golub	0.780 ± 0.079	0.762 ± 0.070	0.762 ± 0.070	0.734 ± 0.077
banknote	0.991 ± 0.005	0.704 ± 0.047	0.706 ± 0.049	0.622 ± 0.027
glass	0.958 ± 0.030	0.865 ± 0.010	0.865 ± 0.010	0.865 ± 0.010
ILPD	0.696 ± 0.029	0.706 ± 0.028	0.719 ± 0.015	0.720 ± 0.015
haberman	0.738 ± 0.063	0.729 ± 0.029	0.722 ± 0.031	0.729 ± 0.013
sonar	0.635 ± 0.091	0.645 ± 0.062	0.639 ± 0.053	0.576 ± 0.081
breastHess	0.732 ± 0.065	0.797 ± 0.072	0.797 ± 0.080	0.721 ± 0.057
breastcancer	0.963 ± 0.014	0.965 ± 0.015	0.969 ± 0.015	0.902 ± 0.035
parkinsons	0.850 ± 0.057	0.754 ± 0.014	0.754 ± 0.014	0.753 ± 0.041
heart	0.822 ± 0.043	0.819 ± 0.047	0.830 ± 0.036	0.811 ± 0.054
climate	0.954 ± 0.014	0.909 ± 0.013	0.911 ± 0.010	0.915 ± 0.007
diabetes	0.759 ± 0.032	0.737 ± 0.026	0.742 ± 0.026	0.637 ± 0.032
ionosphere	0.880 ± 0.019	0.695 ± 0.036	0.695 ± 0.036	0.556 ± 0.049
bupa	0.692 ± 0.060	0.583 ± 0.055	0.548 ± 0.038	0.562 ± 0.025
average	0.825 ± 0.018	0.771 ± 0.016	0.770 ± 0.017	0.736 ± 0.019

Table 4.5: Accuracy using an MLP kernel

Basename	MLP-MMD	MLP-KDD	MLP-KSV
fertility	0.880 ± 0.030	0.870 ± 0.035	0.880 ± 0.030
appendicitis	0.800 ± 0.091	0.821 ± 0.076	0.839 ± 0.055
australian	0.861 ± 0.037	0.861 ± 0.037	0.662 ± 0.045
german	0.723 ± 0.047	0.722 ± 0.034	0.693 ± 0.008
golub	0.764 ± 0.096	0.762 ± 0.109	0.725 ± 0.137
banknote	1.000 ± 0.000	1.000 ± 0.000	0.687 ± 0.062
glass	0.967 ± 0.032	0.967 ± 0.032	0.976 ± 0.024
ILPD	0.701 ± 0.022	0.712 ± 0.011	0.715 ± 0.006
haberman	0.722 ± 0.045	0.722 ± 0.039	0.680 ± 0.124
sonar	0.836 ± 0.084	0.831 ± 0.069	0.552 ± 0.105
breastHess	0.834 ± 0.062	0.804 ± 0.058	0.700 ± 0.071
breastcancer	0.971 ± 0.012	0.971 ± 0.012	0.889 ± 0.042
parkinsons	0.887 ± 0.047	0.872 ± 0.072	0.749 ± 0.021
heart	0.837 ± 0.040	0.841 ± 0.035	0.674 ± 0.048
climate	0.956 ± 0.017	0.948 ± 0.014	0.915 ± 0.007
diabetes	0.745 ± 0.020	0.729 ± 0.033	0.686 ± 0.021
ionosphere	0.877 ± 0.039	0.889 ± 0.034	0.701 ± 0.091
bupa	0.672 ± 0.041	0.643 ± 0.050	0.574 ± 0.016
average	0.835 ± 0.017	0.831 ± 0.018	0.739 ± 0.021

Table 4.6: AUC using the Gaussian kernel

Basename	Gaussian-Grid Search	Gaussian-MMD	Gaussian-KDD	Gaussian-KSV
fertility	0.570 ± 0.156	0.674 ± 0.222	0.663 ± 0.214	0.708 ± 0.156
appendicitis	0.847 ± 0.119	0.819 ± 0.134	0.819 ± 0.134	0.824 ± 0.129
australian	0.929 ± 0.022	0.928 ± 0.021	0.929 ± 0.021	0.932 ± 0.020
german	0.789 ± 0.035	0.782 ± 0.034	0.783 ± 0.034	0.785 ± 0.034
golub	0.803 ± 0.122	0.788 ± 0.144	0.770 ± 0.135	0.740 ± 0.147
banknote	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
glass	0.970 ± 0.063	0.976 ± 0.050	0.976 ± 0.050	0.976 ± 0.050
ILPD	0.609 ± 0.063	0.663 ± 0.034	0.665 ± 0.037	0.668 ± 0.036
haberman	0.668 ± 0.089	0.728 ± 0.061	0.685 ± 0.088	0.604 ± 0.114
sonar	0.948 ± 0.025	0.954 ± 0.023	0.946 ± 0.023	0.951 ± 0.024
breastHess	0.889 ± 0.073	0.894 ± 0.079	0.888 ± 0.081	0.887 ± 0.078
breastcancer	0.995 ± 0.005	0.992 ± 0.007	0.991 ± 0.009	0.990 ± 0.011
parkinsons	0.980 ± 0.019	0.918 ± 0.041	0.931 ± 0.040	0.931 ± 0.033
heart	0.925 ± 0.027	0.921 ± 0.026	0.921 ± 0.026	0.922 ± 0.025
climate	0.955 ± 0.040	0.947 ± 0.045	0.948 ± 0.045	0.952 ± 0.045
diabetes	0.832 ± 0.035	0.832 ± 0.030	0.834 ± 0.029	0.833 ± 0.031
ionosphere	0.979 ± 0.017	0.984 ± 0.016	0.983 ± 0.015	0.983 ± 0.014
bupa	0.744 ± 0.052	0.726 ± 0.070	0.746 ± 0.061	0.739 ± 0.054
average	0.857 ± 0.024	0.863 ± 0.022	0.860 ± 0.023	0.857 ± 0.023

Table 4.7: AUC using the Laplacian kernel

Basename	Laplacian-Grid Search	Laplacian-MMD	Laplacian-KDD	Laplacian-KSV
fertility	0.651 ± 0.212	0.646 ± 0.224	0.635 ± 0.234	0.679 ± 0.229
appendicitis	0.864 ± 0.115	0.899 ± 0.097	0.899 ± 0.097	0.910 ± 0.085
australian	0.914 ± 0.024	0.929 ± 0.021	0.929 ± 0.021	0.930 ± 0.022
german	0.775 ± 0.034	0.795 ± 0.033	0.795 ± 0.033	0.795 ± 0.034
golub	0.790 ± 0.153	0.862 ± 0.115	0.862 ± 0.115	0.862 ± 0.115
banknote	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
glass	0.985 ± 0.029	0.978 ± 0.046	0.978 ± 0.046	0.978 ± 0.046
ILPD	0.680 ± 0.051	0.683 ± 0.041	0.685 ± 0.041	0.686 ± 0.040
haberman	0.653 ± 0.065	0.667 ± 0.067	0.671 ± 0.067	0.661 ± 0.061
sonar	0.934 ± 0.034	0.917 ± 0.073	0.917 ± 0.073	0.916 ± 0.075
breastHess	0.886 ± 0.070	0.902 ± 0.079	0.900 ± 0.079	0.902 ± 0.079
breastcancer	0.992 ± 0.008	0.993 ± 0.006	0.993 ± 0.007	0.993 ± 0.007
parkinsons	0.982 ± 0.019	0.988 ± 0.019	0.988 ± 0.019	0.989 ± 0.017
heart	0.910 ± 0.037	0.921 ± 0.027	0.921 ± 0.027	0.921 ± 0.027
climate	0.939 ± 0.039	0.954 ± 0.041	0.953 ± 0.040	0.954 ± 0.040
diabetes	0.819 ± 0.026	0.832 ± 0.031	0.834 ± 0.029	0.832 ± 0.031
ionosphere	0.982 ± 0.015	0.987 ± 0.014	0.986 ± 0.013	0.986 ± 0.014
bupa	0.745 ± 0.047	0.759 ± 0.051	0.760 ± 0.051	0.769 ± 0.050
average	0.861 ± 0.023	0.873 ± 0.022	0.872 ± 0.023	0.876 ± 0.022

Table 4.8: AUC using the sigmoidal kernel

Basename	Sigmoidal-Grid Search	Sigmoidal-MMD	Sigmoidal-KDD	Sigmoidal-KSV
fertility	0.490 ± 0.276	0.515 ± 0.209	0.442 ± 0.230	0.550 ± 0.267
appendicitis	0.837 ± 0.149	0.584 ± 0.184	0.523 ± 0.107	0.698 ± 0.141
australian	0.923 ± 0.028	0.900 ± 0.025	0.900 ± 0.023	0.869 ± 0.037
german	0.785 ± 0.040	0.735 ± 0.043	0.729 ± 0.045	0.673 ± 0.037
golub	0.842 ± 0.105	0.773 ± 0.119	0.775 ± 0.122	0.748 ± 0.094
banknote	1.000 ± 0.000	0.813 ± 0.039	0.815 ± 0.038	0.711 ± 0.031
glass	0.924 ± 0.095	0.838 ± 0.067	0.848 ± 0.072	0.792 ± 0.077
ILPD	0.682 ± 0.051	0.621 ± 0.066	0.630 ± 0.072	0.628 ± 0.053
haberman	0.626 ± 0.072	0.620 ± 0.084	0.608 ± 0.077	0.499 ± 0.044
sonar	0.716 ± 0.074	0.706 ± 0.050	0.687 ± 0.057	0.602 ± 0.100
breastHess	0.830 ± 0.065	0.833 ± 0.087	0.852 ± 0.065	0.713 ± 0.094
breastcancer	0.991 ± 0.006	0.990 ± 0.007	0.990 ± 0.007	0.960 ± 0.022
parkinsons	0.885 ± 0.056	0.667 ± 0.072	0.681 ± 0.084	0.772 ± 0.079
heart	0.900 ± 0.023	0.895 ± 0.027	0.899 ± 0.026	0.901 ± 0.027
climate	0.955 ± 0.038	0.816 ± 0.054	0.817 ± 0.071	0.739 ± 0.096
diabetes	0.821 ± 0.038	0.787 ± 0.035	0.792 ± 0.032	0.667 ± 0.058
ionosphere	0.887 ± 0.049	0.772 ± 0.050	0.770 ± 0.051	0.427 ± 0.081
bupa	0.701 ± 0.061	0.618 ± 0.052	0.530 ± 0.064	0.493 ± 0.063
average	0.822 ± 0.026	0.749 ± 0.025	0.738 ± 0.027	0.691 ± 0.028

Table 4.9: AUC using an MLP kernel

Basename	MLP-MMD	MLP-KDD	MLP-KSV
fertility	0.558 ± 0.258	0.542 ± 0.259	0.573 ± 0.205
appendicitis	0.720 ± 0.207	0.794 ± 0.128	0.740 ± 0.152
australian	0.929 ± 0.020	0.923 ± 0.021	0.699 ± 0.055
german	0.740 ± 0.066	0.742 ± 0.034	0.560 ± 0.067
golub	0.765 ± 0.130	0.797 ± 0.110	0.718 ± 0.163
banknote	1.000 ± 0.000	1.000 ± 0.000	0.819 ± 0.062
glass	0.980 ± 0.042	0.978 ± 0.046	0.974 ± 0.050
ILPD	0.734 ± 0.057	0.731 ± 0.045	0.644 ± 0.061
haberman	0.679 ± 0.089	0.656 ± 0.086	0.618 ± 0.102
sonar	0.878 ± 0.069	0.891 ± 0.063	0.579 ± 0.143
breastHess	0.875 ± 0.083	0.870 ± 0.074	0.792 ± 0.085
breastcancer	0.991 ± 0.007	0.991 ± 0.007	0.946 ± 0.030
parkinsons	0.937 ± 0.060	0.933 ± 0.050	0.604 ± 0.076
heart	0.883 ± 0.042	0.904 ± 0.050	0.696 ± 0.079
climate	0.946 ± 0.055	0.941 ± 0.047	0.525 ± 0.124
diabetes	0.805 ± 0.040	0.806 ± 0.046	0.648 ± 0.058
ionosphere	0.949 ± 0.033	0.937 ± 0.038	0.707 ± 0.076
bupa	0.687 ± 0.047	0.695 ± 0.077	0.568 ± 0.098
average	0.836 ± 0.026	0.841 ± 0.025	0.690 ± 0.027

4.2 Comparison with regularly-trained MLPs

As said in the last section, only one architecture of MLP was used during the tests. However, this topology may not be ideal for the problems used here. Then, experiments were made to see how this neural network would perform on the same datasets when trained regularly, using cross-entropy loss (Goodfellow *et al.*, 2016).

For this, a linear classification layer was added to the network, making then a three-layer neural network.

Besides the end-to-end training with cross-entropy classification loss, the network was trained using each criterion (KDD, KSV, and MMD), excluding the last layer's linear classifier, which was trained afterwards, separately, using the cross-entropy loss. In this situation, the hidden layer representations were defined beforehand, and the linear classification layer was tuned while the rest of the network remained static.

Accuracy and AUC results are shown in Tables 4.10 and 4.11, respectively. Each experiment was executed 10 times for each dataset using cross-validation. The confidence intervals are for a significance of 0.05.

Table 4.10: Accuracy in comparison with an end-to-end trained MLP

Basename	MLP-CE	MLP-MMD	MLP-KDD	MLP-KSV
fertility	0.800 ± 0.048	0.880 ± 0.030	0.880 ± 0.030	0.880 ± 0.030
appendicitis	0.838 ± 0.081	0.838 ± 0.082	0.819 ± 0.085	0.857 ± 0.074
australian	0.852 ± 0.033	0.859 ± 0.038	0.862 ± 0.035	0.633 ± 0.031
german	0.728 ± 0.029	0.719 ± 0.051	0.718 ± 0.038	0.701 ± 0.006
golub	0.750 ± 0.080	0.748 ± 0.117	0.764 ± 0.096	0.752 ± 0.101
banknote	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.686 ± 0.022
glass	0.962 ± 0.035	0.958 ± 0.033	0.958 ± 0.030	0.957 ± 0.044
ILPD	0.718 ± 0.024	0.700 ± 0.025	0.715 ± 0.006	0.715 ± 0.006
haberman	0.712 ± 0.042	0.732 ± 0.040	0.738 ± 0.050	0.738 ± 0.041
sonar	0.807 ± 0.033	0.826 ± 0.073	0.841 ± 0.067	0.500 ± 0.085
breastHess	0.797 ± 0.071	0.849 ± 0.063	0.826 ± 0.063	0.745 ± 0.023
breastcancer	0.969 ± 0.014	0.972 ± 0.014	0.972 ± 0.014	0.836 ± 0.031
parkinsons	0.918 ± 0.031	0.913 ± 0.039	0.892 ± 0.062	0.749 ± 0.011
heart	0.811 ± 0.063	0.833 ± 0.047	0.841 ± 0.043	0.700 ± 0.056
climate	0.957 ± 0.011	0.952 ± 0.022	0.939 ± 0.021	0.913 ± 0.009
diabetes	0.766 ± 0.030	0.741 ± 0.019	0.729 ± 0.025	0.698 ± 0.026
ionosphere	0.900 ± 0.024	0.880 ± 0.021	0.894 ± 0.032	0.661 ± 0.025
bupa	0.715 ± 0.042	0.672 ± 0.057	0.611 ± 0.051	0.563 ± 0.027
average	0.833 ± 0.016	0.837 ± 0.017	0.833 ± 0.018	0.738 ± 0.019

Table 4.11: AUC in comparison with an end-to-end trained MLP

Basename	MLP-CE	MLP-MMD	MLP-KDD	MLP-KSV
fertility	0.620 ± 0.235	0.591 ± 0.238	0.644 ± 0.221	0.647 ± 0.151
appendicitis	0.753 ± 0.144	0.784 ± 0.136	0.828 ± 0.117	0.759 ± 0.140
australian	0.924 ± 0.022	0.918 ± 0.029	0.919 ± 0.025	0.705 ± 0.062
german	0.758 ± 0.049	0.768 ± 0.045	0.758 ± 0.037	0.560 ± 0.054
golub	0.733 ± 0.120	0.752 ± 0.114	0.775 ± 0.135	0.693 ± 0.176
banknote	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.818 ± 0.065
glass	0.984 ± 0.022	0.987 ± 0.025	0.988 ± 0.023	0.983 ± 0.019
ILPD	0.749 ± 0.031	0.739 ± 0.062	0.744 ± 0.044	0.625 ± 0.058
haberman	0.690 ± 0.069	0.696 ± 0.091	0.648 ± 0.139	0.633 ± 0.095
sonar	0.896 ± 0.044	0.905 ± 0.059	0.910 ± 0.048	0.522 ± 0.125
breastHess	0.844 ± 0.072	0.877 ± 0.088	0.872 ± 0.092	0.748 ± 0.123
breastcancer	0.991 ± 0.008	0.986 ± 0.010	0.987 ± 0.011	0.897 ± 0.040
parkinsons	0.962 ± 0.039	0.958 ± 0.029	0.945 ± 0.049	0.543 ± 0.098
heart	0.905 ± 0.027	0.910 ± 0.045	0.903 ± 0.045	0.739 ± 0.041
climate	0.937 ± 0.042	0.932 ± 0.051	0.939 ± 0.040	0.499 ± 0.109
diabetes	0.826 ± 0.029	0.815 ± 0.030	0.812 ± 0.034	0.669 ± 0.040
ionosphere	0.958 ± 0.017	0.935 ± 0.037	0.935 ± 0.044	0.504 ± 0.093
bupa	0.721 ± 0.059	0.707 ± 0.068	0.692 ± 0.082	0.511 ± 0.042
average	0.847 ± 0.023	0.848 ± 0.024	0.850 ± 0.024	0.670 ± 0.027

4.3 Visual experiments with MNIST dataset

Visual experiments were also conducted to explore the final aspect of the representations. A simple convolutional neural network (LeCun *et al.*, 1995) was trained in the MNIST handwritten digits dataset (LeCun *et al.*, 2010). The model used had two convolutional layers and one fully connected, mapping to an output space with 128 dimensions. A logistic function is applied pointwise in the elements of every output vector, and the vectors are normalized to have unit norm. This architecture achieved an accuracy of 98.84% on the test set. Both KDD and KSV were used.

After training, samples from both training and test sets were mapped into the network's output space, and, subsequently, into a two-dimensional space using t-SNE (Maaten & Hinton, 2008) to facilitate visualization.

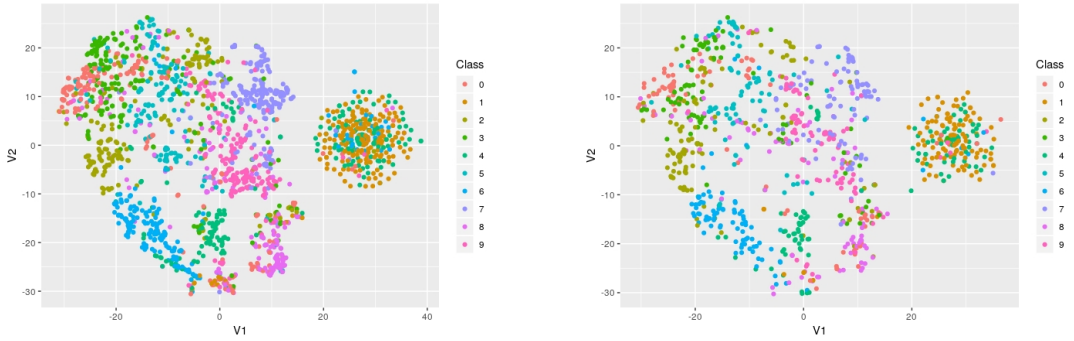
The mappings learned using KDD are depicted in Figure 4.1. Patterns representing the same digits are clustered into compact groups in both training and test sets, showing that samples from the same class tend to be more similar, while samples from different classes tend to be more distant.

Mappings learned with the unsupervised KSV function are shown in Figure 4.2. Patterns from the same class tend to be close as well. However, the network does not separate them into clear clusters, and most classes overlap with

another one. The accuracy of the test set using KSV was 23.21%.

(a) Training set (b) Test set

Figure 4.1: Visualization of MNIST representations learned with KDD maximization. Maximizing the discrepancy between classes in the training set still yields coherent groups in the test set. t-SNE was used to facilitate visualization.



(a) Training set (b) Test set

Figure 4.2: Visualization of MNIST representations learned with KSV maximization. There is a lot of overlap between classes, which helps explain the poor classification performances. However, points of the same class still tend to get mapped close to each other. t-SNE was used to facilitate visualization.

4.4 Result Analysis and Discussions

4.4.1 Supervised KDD Function

The results unquestionably show the usefulness of the KDD function in learning class-coherent representations. By maximizing the pairwise distances between the class distributions, distinct classes can be distinguished, and patterns from the same class are aggregated into cohesive clusters.

For the two RBF kernels, both accuracy and AUC results show that the KDD function is equivalent to finding kernel parameters using exhaustive search and cross-validation to estimate performance. This is not true for the sigmoidal kernel: grid-search SVM performed consistently better. This kernel, however, is not always positive semidefinite, which means that the reproducing kernel map is not guaranteed to exist. Thus, it is no surprise that the proposed methods do not always achieve satisfactory results using this function.

The numeric results also show that the Maximum Mean Discrepancy maximization is statistically equivalent to maximizing the kernel distance in all settings and kernels used. Both functions can be computed with very close forms, taken with respect to the same terms, and can even represent distances in the same space.

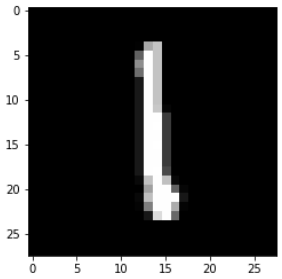
4.5 Unsupervised KSV Function

On the other hand, the unsupervised function did not perform very well on average, being consistently the worst between the compared groups. However, the Gaussian and Laplacian kernels are exceptions, in which the accuracy and AUC results were statistically equivalent to the others, including the fully grid-search chosen parameters.

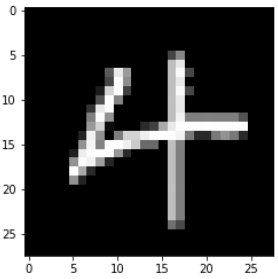
One possible explanation for this is that the RBF kernels have the **locality** prior. Since these kernels are mere functions of the distance, they capture local information of the data. Furthermore, they are nonlinear functions, with which the dot product in feature space $\mathcal{H}$ decreases exponentially when the distance increases.

Neural networks are a very expressive family of functions, and, in theory, can implement functions that capture these local aspects. Even so, they are not limited to that, as are RBF kernels. Besides, the Kernel Similarity Variance alone does not enforce these locality principles: there are maxima and minima points on the function that violate the low distance/high inner product principle. For instance, the digits 1 and 4 in Figure 4.2 are part of the same cluster, with no distinction between them whatsoever. Although both of them typically have one vertical line, the fours have additional structure and are further away from the ones than the ones are to themselves, as a human can quickly recognize (examples are depicted in Figure 4.3). Thus, there should be at least some distinction between them in the output space.

Without supervision or strong priors and inductive biases, it is possible that neural networks will not learn satisfactory representations and can learn functions as complex as necessary to minimize the loss.



(a) A digit from class 1.



(b) A digit from class 4.

Figure 4.3: Examples of images from the MNIST dataset. Digits 1 and 4 were mixed in the same cluster by the KSV-trained MLP.

A possible solution is to embed this local prior in the training process of the network, forcing it to implement a local function, where close points in input space will also be close in representation space. This can be done, for instance, by penalizing the norm of the network's Jacobian using a contractive term, such as contractive autoencoders (CAEs) (Rifai *et al.*, 2011b). It is known that CAEs actually learn a vector field with information on the generator probability density

function (Alain & Bengio, 2014). A simple regularization term, such as the norm of the weights, can also be useful by enforcing smoothness.

The sigmoidal kernel did not work well with the KSV function. The numeric results using it were even worse than the KDD results. Here, the two problems add up: the kernel is only conditionally PSD, and may not always have the reproducing kernel map the function relies on. Moreover, the sigmoidal kernel does not capture the local information the RBF kernels do. With that in mind, the poor performance is no surprise.

Chapter 5

Conclusions and Future Work

In this work, two methods to learn representations with a given parametric map were proposed. These methods were built over kernel theory, and criteria are computed based entirely on inner products between points in the feature space. Here, classification problems were the focus, aiming to reach a space where the classes are as separable as possible.

The two methods consisted of a supervised and an unsupervised function. The supervised one, called the kernel distributional distance function, took into account the class information to maximize a distance measure between class-conditional probabilities on the inputs. However, this measure is defined in a general form and can be applied to any pair of distributions using a suitable kernel. The second method tries to find a map that is more expressive, with some samples in high-density regions while others stay in low-density ones.

Experiments optimizing parameters of Gaussian and Laplacian kernels show that both methods achieve good accuracy and AUC results when used in support vector classification. Choosing the kernel parameters via grid-search as a baseline, the results were statistically equivalent using confidence intervals for $p = 0.05$. They were also equivalent to choosing the parameters using MMD.

Using sigmoidal kernels, however, the results were not good. In fact, the grid-search baseline was unequivocally superior. This kernel, however, is not positive semidefinite, which does not guarantee the reproducing kernel map exists. As the proposed functions rely on the existence of this space, it is understandable that the performance was not as good as in the RBF kernels.

Experiments were also carried using multilayer perceptrons in the kernel framework. Using the KDD function to train the network's representations prior to the classification step, it can be seen that it is possible to reach spaces with separable classes, reflecting on the numerical results. However, the KSV function did not repeat KDD's performance. Even though the network could minimize the loss and find convergent regions, the learned representations had lots of class overlap. A possible explanation for this is that the neural network, albeit a continuous function, does not enforce locality when mapping to the representation space. In contrast, the map defined by RBF kernels is based, by design, on information of the immediate neighborhood of the points.

5.1 Future Work

Regarding the kernel distributional distance, more fundamental work has to be done. First of all, an exploration regarding the possibility of the kernel discrepancy between two distributions is a metric in the space of distributions. For that, one needs to check if it satisfies all the axioms for such. Another important work regarding the KDD is to analyze and interpret it inside the broader framework of measures of divergence between distributions, since it is not yet clear whether the KDD can be considered an integral probability metric.

The kernel similarity variance function achieved great results with translation-invariant kernels, but not when used in MLPs. As it is hypothesized that this function works better with operators that preserve local information other than global, this hypothesis could be tested by using a suitable inductive bias when training the network. One idea for a possible extension is to use a contractive term in the loss function, just as contractive autoencoders (CAEs) Rifai *et al.* (2011b), which penalize the norm of the function's Jacobian matrix evaluated over the training points.

Although classification problems were the main focus, the proposed methods can be applied to other types of problems. For instance, these functions can be used to learn embeddings in $\mathbb{R}^N$ of data such as images, text, and speech. A work where it is used to embed speech data based on speaker information is already in progress. Another possible application is in generative models. For

example, generative adversarial networks could be trained to minimize the kernel discrepancy between original and generated data, just as Li *et al.* (2017) with maximum mean discrepancy.

From a computational cost point of view, each evaluation of the functions demands kernel values of all pairwise training points. Thus, the processing cost can escalate quadratically with the sample size. To mitigate these effects, it may be possible to optimize them dividing the training set in mini-batches. This brings a trade-off: smaller-sized batches makes the function evaluations computationally cheaper, but may affect the confidence in their estimation. This effect was not extensively studied.

References

Alain, G. & Bengio, Y. (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, **15**, 3563–3593. 58

Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L. & Herrera, F. (2011). Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, **17**. 47

Bengio, Y., Courville, A. & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**, 1798–1828. 1, 11, 12

Boser, B.E., Guyon, I.M. & Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, 144–152, ACM, New York, NY, USA. 2, 10

Burges, C.J.C. (1998). Geometry and Invariance in Kernel Based Methods. **3**, 54–67. 43, 44

Castro, C.L. & Braga, A.P. (2013). Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data. *IEEE transactions on neural networks and learning systems*, **24**, 888–899. 48

Chang, C.C. & Lin, C.J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, **2**, 1–27. 47

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I. & Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning. *30th Conference on Neural Information Processing Systems (NIPS 2016)*, 2172–2180. 12

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, **20**, 273–297. 20

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, **2**, 303–314. 23

Duda, R.O., Hart, P.E. & Stork, D.G. (1973). *Pattern classification*. Wiley, New York. 17

Friedman, J., Hastie, T. & Tibshirani, R. (2001). *The elements of statistical learning*, vol. 1. Springer series in statistics New York. 2

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, **46**, 1–37. 19

Garriga-Alonso, A., Rasmussen, C.E. & Aitchison, L. (2018). Deep convolutional networks as shallow gaussian processes. *arXiv preprint arXiv:1808.05587*. 24

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A. *et al.* (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, **286**, 531–537. 48

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT press. 7, 10, 12, 36, 53

Goodfellow, I.J., Shlens, J. & Szegedy, C. (2015). Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–11. 12

Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B. & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, **13**, 723–773. 3, 20, 31

Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, **3**, 1157–1182. 1

Hess, K.R., Anderson, K., Symmans, W.F., Valero, V., Ibrahim, N., Mejia, J.A., Booser, D., Theriault, R.L., Buzdar, A.U., Dempsey, P.J. *et al.* (2006). Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer. *Journal of clinical oncology*, **24**, 4236–4244. 47

Hinton, G.E. & Zemel, R.S. (1994). Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*, 3–10. 2

Hinton, G.E., Sejnowski, T.J. & Ackley, D.H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA. 1

Jacot, A., Gabriel, F. & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, **2018-Decem**, 8571–8580. 23

Kim, S.J., Zymnis, A., Magnani, A., Koh, K. & Boyd, S. (2008). Learning the kernel via convex optimization. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, 1997–2000, IEEE. 2

LeCun, Y., Bengio, Y. *et al.* (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, **3361**, 1995. 54

LeCun, Y., Cortes, C. & Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, **2**. 11, 54

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, **521**, 436–444. 10

Lee, J., Bahri, Y., Novak, R., Schoenholz, S.S., Pennington, J. & Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*. 23

Li, C.L., Chang, W.C., Cheng, Y., Yang, Y. & Póczos, B. (2017). MMD GAN: Towards deeper understanding of moment matching network. *Advances in Neural Information Processing Systems*, **2017-Decem**, 2204–2214. 20, 61

Li, Y., Swersky, K. & Zemel, R. (2015). Generative moment matching networks. In *International Conference on Machine Learning*, 1718–1727. 20

Lichman, M. (2013). UCI machine learning repository. 47

Maaten, L.v.d. & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, **9**, 2579–2605. 11, 54

MacCluer, B. (2008). *Elementary functional analysis*, vol. 253. Springer Science & Business Media. 14

Mhaskar, H., Liao, Q. & Poggio, T. (2016). Learning Functions: When Is Deep Better Than Shallow. 1–12. 23

Mitchell, T.M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, **45**, 870–877. 6

Montavon, G., Braun, M.L. & Müller, K.R. (2011). Kernel analysis of deep networks. *Journal of Machine Learning Research*, **12**, 2563–2581. 23

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. & Lerer, A. (2017). Automatic differentiation in pytorch. 47

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. *et al.* (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, **12**, 2825–2830. 47

Popoviciu, T. (1935). Sur les équations algébriques ayant toutes leurs racines réelles. *Mathematica*, **9**, 129–145. 38

Rifai, S., Vincent, P., Muller, X., Glorot, X. & Bengio, Y. (2011a). Contractive auto-encoders: Explicit invariance during feature extraction. *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 833–840. 12

Rifai, S., Vincent, P., Muller, X., Glorot, X. & Bengio, Y. (2011b). Contractive auto-encoders: Explicit invariance during feature extraction. 57, 60

Rosasco, L., De Vito, E., Caponnetto, A., Piana, M. & Verri, A. (2004). Are Loss Functions All the Same? *Neural Computation*, **16**, 1063–1076. 7

Salakhutdinov, R. & Hinton, G. (2009). Deep Boltzmann machines. *Journal of Machine Learning Research*, **5**, 448–455. 1

Scholkopf, B. & Smola, A.J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press. 2, 8, 13, 14, 15, 21, 36

Silverman, B.W. (1986). *Density estimation for statistics and data analysis*, vol. 26. CRC press. 16, 17

Sriperumbudur, B.K., Fukumizu, K., Gretton, A., Schölkopf, B. & Lanckriet, G.R.G. (2009). On integral probability metrics, ϕ-divergences and binary classification. 1–18. 19, 20

Tikhonov, A.N. (1963). Solution of incorrectly formulated problems and the regularization method. *Soviet Math.*, **4**, 1035–1038. 10

Torres, L.C., Lemos, A.P., Castro, C.L. & Braga, A.P. (2014). A geometrical approach for parameter selection of radial basis functions networks. In *International Conference on Artificial Neural Networks*, 531–538, Springer. 2

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer science & business media. 6, 8, 42

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, 1096–1103. 12

Wanderley, M.F.B., Torres, L.C.B., Natowicz, R. & Braga, A.P. (2014). A maximum margin-based kernel width estimator and its application to the response to neoadjuvant chemotherapy. *Revista Brasileira de Engenharia Biomédica*, **30**, 17–26. 2