



Cassiano Otávio Becker

HYPER-PARAMETER OPTIMIZATION FOR MANIFOLD
REGULARIZATION LEARNING

OTIMIZAÇÃO DE HIPERPARÂMETROS PARA APRENDIZADO DO
COMPUTADOR POR REGULARIZAÇÃO EM VARIEDADES

Campinas
2013



UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Cassiano Otávio Becker

HYPER-PARAMETER OPTIMIZATION FOR MANIFOLD
REGULARIZATION LEARNING

OTIMIZAÇÃO DE HIPERPARÂMETROS PARA APRENDIZADO DO
COMPUTADOR POR REGULARIZAÇÃO EM VARIEDADES

Master dissertation presented to the School of Electrical and Computer Engineering in partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

Concentration area: Automation.

Dissertação de Mestrado apresentada ao Programa de Pós- Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Automação.

Orientador (Advisor): Prof. Dr. Paulo Augusto Valente Ferreira

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Cassiano Otávio Becker, e orientada pelo Prof. Dr. Paulo Augusto Valente Ferreira.

Campinas
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

B388h Becker, Cassiano Otávio, 1977-
Hyper-parameter optimization for manifold regularization learning / Cassiano Otávio Becker. – Campinas, SP : [s.n.], 2013.

Orientador: Paulo Augusto Valente Ferreira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Aprendizado do computador. 2. Aprendizado semi-supervisionado. 3. Otimização matemática. 4. Seleção de modelo (Estatística). I. Ferreira, Paulo Augusto Valente, 1958-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Otimização de hiperparâmetros para aprendizado do computador por regularização em variedades

Palavras-chave em inglês:

Machine learning

Semi-supervised learning

Mathematical optimization

Model selection

Área de concentração: Automação

Titulação: Mestre em Engenharia Elétrica

Banca examinadora:

Paulo Augusto Valente Ferreira [Orientador]

Fernando Antônio Campos Gomide

Tiago Agostinho de Almeida

Data de defesa: 12-08-2013

Programa de Pós-Graduação: Engenharia Elétrica

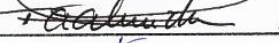
COMISSÃO JULGADORA - TESE DE MESTRADO

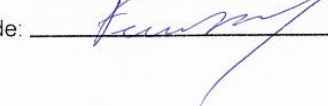
Candidato: Cassiano Otávio Becker

Data da Defesa: 12 de agosto de 2013

Título da Tese: "Otimização de Hiperparâmetros para Aprendizado do Computador por Regularização em Variedades"

Prof. Dr. Paulo Augusto Valente Ferreira (Presidente): 

Prof. Dr. Tiago Agostinho de Almeida: 

Prof. Dr. Fernando Antônio Campos Gomide: 

Abstract

This dissertation investigates the problem of hyper-parameter optimization for regularization based learning models. A review of different learning algorithms is provided in terms of different losses and learning tasks, including Support Vector Machines, Regularized Least Squares and their extension to semi-supervised learning models, more specifically, Manifold Regularization. A gradient based optimization approach is proposed, using an efficient calculation of the Leave-one-out Cross Validation procedure. Datasets and numerical examples are provided in order to evaluate the methods proposed in terms of their generalization capability of the generated models.

Key-words: Machine Learning. Semi-Supervised Learning. Optimization. Support Vector Machines. Regularized Least Squares. Manifold Regularization. Model Selection.

Resumo

Esta dissertação investiga o problema de otimização de hiperparâmetros para modelos de aprendizado do computador baseados em regularização. Uma revisão destes algoritmos é apresentada, abordando diferentes funções de perda e tarefas de aprendizado, incluindo Máquinas de Vetores de Suporte, Mínimos Quadrados Regularizados e sua extensão para modelos de aprendizado semi-supervisionado, mais especificamente, Regularização em Variedades. Uma abordagem baseada em otimização por gradiente é proposta, através da utilização de um método eficiente de cálculo da função de validação por exclusão unitária. Com o intuito de avaliar os métodos propostos em termos de qualidade de generalização dos modelos gerados, uma aplicação deste método a diferentes conjuntos de dados e exemplos numéricos é apresentada.

Palavras-chave: Aprendizado do Computador. Aprendizado Semi-Supervisionado. Otimização Matemática. Máquinas de Vetores de Suporte. Regularização por Mínimos Quadrados. Regularização em Variedades. Seleção de Modelo.

Contents

Introduction	1
1 Machine Learning and the Regularization Framework	4
1.1 Supervised, Unsupervised and Semi-supervised Machine Learning	4
1.1.1 Learning Tasks	5
1.1.2 Regression	5
1.1.3 Classification	6
1.1.4 Other Loss Functions and Tasks	7
1.2 Regularization Learning	8
1.2.1 Statistical Machine Learning Interpretation	9
1.2.2 Solution in Reproducing Kernel Hilbert Spaces	11
1.2.3 From Infinite-dimensional to l -Dimensional Optimization Problem	13
1.3 Regularized Least Squares	16
1.4 Support Vector Machines	17
1.5 Computational Considerations	18
2 Hyper-parameter Optimization	20
2.1 Introduction	20
2.2 Model Assessment	21
2.2.1 Validation and Test Functions	21
2.2.2 Bias-Variance Decomposition	23
2.2.3 Partition Strategies	25
2.2.4 Cross-Validation	26
2.2.5 Leave-One-Out Bounds	27
2.3 Formulation as an Optimization Problem	28
2.4 Hyper-parameter Search Strategies	29
2.4.1 Grid Search	29

2.4.2	Random Search	29
2.4.3	Gradient Optimization	30
2.5	Multiple Kernel Learning	32
3	Semi-supervised Learning: Manifold Regularization	33
3.1	Semi-supervised Learning	33
3.1.1	Transductive and Inductive Learning	34
3.1.2	The Manifold Assumption	34
3.2	Manifold Regularization	35
3.2.1	The Laplacian and its Hyper-parameters	38
3.2.2	Point Cloud Kernel	40
3.3	Laplacian Regularized Least Squares	42
3.4	Laplacian Support Vector Machines	43
3.5	Computational Considerations	44
4	Hyper-parameter Optimization for Manifold Regularization Learning Models	45
4.1	Hyper-parameter Optimization Formulation	45
4.2	Combination of Laplacians	47
4.2.1	Computational Considerations	48
4.3	Computational Experiments	49
4.3.1	Experimental Setup	49
4.3.2	Gradient Optimization of Regularized Least Squares	51
4.3.3	Influence of Unlabeled Data	53
4.3.4	Laplacian RLS Hyper-parameter Optimization with Limited Function Evaluations	56
5	Conclusion	61
	References	64
A	Parameters and Hyper-parameters	69
B	Experimental Code	70

Acknowledgement

I would like to thank:

Prof. Dr. Paulo Valente, for his openness and availability.

My wife Mariana, for her unconditional support and continuous interest.

My parents Angela and Mario, for their wisdom and incentive.

The members of the Committee for their comments and thorough revision.

Venturus Innovation Center for allowing me to attend classes while being their employee.

The professors of FEEC/Unicamp for their quality and interest in teaching.

All differences in this world are of degree, and not of kind, because oneness is the secret of everything.

Swami Vivekananda

List of Figures

1.1	The ϵ -insensitive loss function	6
1.2	The hinge loss function	7
1.3	Different approximation functions with zero loss at training points	8
3.1	Manifold Regularization learning classification function for the Two-moons dataset	38
3.2	Point Cloud Kernel for the Two-circles dataset	42
4.1	Hyper-parameter level sets and feval graph for the COIL20 dataset	52
4.2	Hyper-parameter level sets and feval graph for the USPST dataset	53
4.3	COIL20 dataset boxplots for LapRLS test set error performance	54
4.4	USPST dataset boxplots for LapRLS test set error performance	55
4.5	Point Cloud Kernel equivalence with LapRLS	56
4.6	COIL20 boxplots for test set error comparing different search methods.	58
4.7	USPST boxplots for test set error comparing different search methods.	59

Acronyms and Notation

SVM	Support Vector Machine
RLS	Regularized Least Squares
LOOCV	Leave-one-out Cross Validation
RKHS	Reproducing Kernel Hilbert Space

\mathbf{M}	notation for matrices (latin capital letters)
\mathbf{M}^T	$(^T)$, post posed to a vector or matrix, indicates transposition
$\mathbf{M} \geq 0$	matrix \mathbf{M} is symmetric positive semi-definite
$\ \cdot\ _K$	norm K of a vector, matrix or function
\mathbf{x}^*	optimal value for vector \mathbf{x} under criteria defined in the context
\mathbb{R}	the real set of numbers
\mathbb{Z}	the integer set of numbers
\mathbb{Z}_+	the non-negative integer set of numbers
\mathbb{N}	the natural set of numbers (including zero)
\mathbf{I}	identity matrix of appropriate dimension
\mathbf{e}	vector of all ones of appropriate dimension

Introduction

Machine Learning is increasingly present in many engineering domains. Bringing together elements from statistics, neuroscience and computer science, it enables systems to improve their performance by implementing mechanisms that process data they collect from their environment. With the growth of interconnected systems and communications technology, the availability of data is a significant resource and opportunity to be explored.

The list of relevant applications of Machine Learning can be as large as one might wish it to be. Examples can be found from medicine (medical imaging, protein folding, drug discovery) to social security and fraud detection. From business intelligence (infrastructure monitoring, recommender systems) to highly elaborate autonomous vehicles and robotics. With the pressing needs of society for more efficient systems to address the challenges of natural resources and equality of opportunities for an increasing population, such list can only increase in number and depth.

The field of Machine Learning as it is found today can be understood to have begun, on one side, with the modeling of neurons as the Rosenblatt perceptron (Rosenblatt 1958), which later enabled the field of Artificial Neural Networks. On the other side, it received continuous contributions from statistical techniques for approximating and estimating mathematical functions from data collections, such as Minimum Least Squares and Linear Regression. A major development was achieved in the 1990's when the problems received a more general formulation with Support Vector Machines and Kernel Functions, which at the same time relied on results from the field of Convex Optimization, assuring it a solid and well behaved mathematical base. Such formulation was later further integrated and generalized with the current approach of Regularization Theory.

One aspect that has not been entirely addressed in Regularization Theory is the definition of the hyper-parameters of the learning function being estimated, even though such parameters are often found to have a significant influence over the final algorithm performance. These

variables are defined as exogenous to the learning problem and are either assumed to have been defined previously, from prior knowledge, or left open to be determined ad-hoc. In the latter, usually a blind search is employed (grid search), which linearly traverses the dimensions of the parameters. Such strategies tend to be sub-optimal and become prohibitive as the number of hyper-parameters grow. The increase in knowledge and eventual improvement in the search and optimization of such parameters constitutes a path to higher efficiency and quality in the determination of computational models in Machine Learning.

Another significant dimension of improvement in the problem of learning from data has been the development of semi-supervised algorithms, which are able to take advantage of both labeled and unlabeled data (data whose meaning or judgement is not provided). Such algorithms are of relevance because unlabeled data is usually generated at much lower cost, as they forego the labeling stage typically performed by a specialist, or demand additional energy and knowledge. In many cases, unlabeled data is readily available or can be autonomously and continuously collected by the system. Manifold Regularization is a semi-supervised learning approach which extends the regularization framework by using the unlabeled data distribution to present additional factors to the learning optimization objective function. Such terms, however, introduce additional hyper-parameters, whose values affect the algorithm's predictive performance. The work in this dissertation attempts to bring together these two problems by looking at possibilities for hyper-parameter optimization for Manifold Regularization learning algorithms. It is structured as follows:

- **Chapter 1** presents a review of the regularization framework, which gives rise to learning algorithms which are approached in this dissertation, such as the Support Vector Machines and Regularized Least Squares, both also known as Kernel Learning Machines. In the review, a connection with some statistical learning principles are established. The chapter begins with the basic definitions of common Machine Learning tasks, such as classification and regression.
- In **Chapter 2**, the problem of hyper-parameter optimization is addressed, as part of the broader model selection problem. Different learning model assessment strategies are described, which differ in the way the available training data is used. The chapter then presents some of the most relevant hyper-parameter search strategies, with special attention to the gradient derivation of the closed form leave-one-out cross validation value. An alternative approach that involves the optimization of combination weights of kernel functions is also briefly presented.

- **Chapter 3** presents an important extension to the regularization framework which enables the algorithm to benefit from unlabeled data, leading to a semi-supervised setting. The Manifold Regularization technique derivation is described, along with its main mathematical components and training algorithms. An important result, the Point Cloud Kernel, which enables the use of previously not semi-supervised algorithms as becoming such, is also included.
- **Chapter 4** is devoted to explore the application of some of the hyper-parameter optimization techniques presented in Chapter 2 to the Manifold Regularization learning models presented in Chapter 3. Such models present an additional challenge as they introduce additional hyper-parameters, as compared to non semi-supervised models. In the case where parameters are integers, the employment of a weighted combination of base components is proposed. Finally, computational experiments are presented which illustrate the main ideas presented in the text.

Finally, conclusions and suggestions for future work are presented in **Chapter 5**.

Machine Learning and the Regularization Framework

1.1 Supervised, Unsupervised and Semi-supervised Machine Learning

The overall goal of machine learning can be described as the one of producing algorithms which are able to establish (or learn) a functional relationship between two variables, $f : \mathcal{X} \rightarrow \mathcal{Y}$, based on an underlying phenomenon $p(\mathbf{x}, y)$, which is probabilistic in nature. In order to achieve it, the learning system is given a set of samples of $p(\mathbf{x}, y)$, which are assumed to be independent and identically distributed (i.i.d), as is expected to produce a valid approximation. The “independent” part of the assumption means that one example being observed does not affect the value or nature of the next example, and by “identically”, the assumption means that all samples are sampled from the same underlying probability distribution. With these samples, an estimator $f(\mathbf{x})$ for future realizations of the process is sought, based on the additional induction assumption that future data will be also drawn from the same originating distribution. Assigning variables to the samples constituting the training set, we can say that it might be composed by a combination of:

- l labeled examples $S_l = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_l, y_l)\}$, with $\mathbf{x}_i \in \mathcal{X}$, a general set, for example $\mathcal{X} = \mathbb{R}^N$ and $y_i \in \mathcal{Y}$ the output prediction variable set.
- u unlabeled examples $S_u = \{\mathbf{x}_1, \dots, \mathbf{x}_u\}$, with $\mathbf{x}_i \in \mathcal{X}$, and to which their corresponding y_i values are not provided.

The set \mathcal{X} can be some subset of a non-Euclidean input domain, such as trees, strings, graphs and other structured objects, as long as an appropriate similarity measure between two examples

can be defined. If only labeled samples are provided, the learning is said to be *supervised*. If both labeled and unlabeled sets are provided, the setting is said to be *semi-supervised*, with typically $u > l$ to $u \gg l$. If only unlabeled examples are provided, the algorithms are referred to as *unsupervised*, and their objective is typically to estimate the underlying probability density $p(\mathbf{x})$.

The set \mathcal{Y} defines the prediction variable $y = \{y_1, \dots, y_l\}$, and its domain characterizes the different types of functional relationships, which are grouped as different learning tasks. For example, if y is allowed only to assume two different integer values, such as $y_i \in \{-1, 1\}$, the task is referred to as the one of binary classification.

Given the training sets S_l, S_u and the type of functional relationship $f(\mathbf{x})$ expected, each learning task is also associated with a loss function, $V(y_i, f(\mathbf{x}_i))$, which provides a quantitative measure of the approximation quality of the function being estimated. The next section describes some learning tasks and their loss functions.

1.1.1 Learning Tasks

In this section we provide some detail for two of the main learning tasks involved in supervised and semi-supervised learning, Regression and Classification.

1.1.2 Regression

In Regression one is interested in establishing an approximation of $f : \mathcal{X} \rightarrow \mathcal{Y}$, with $\mathcal{Y} = \mathbb{R}$. It means that $y \in \mathcal{Y}$ can assume any continuous value in the range of the reals, and the discrepancy in the approximation can be measured by the difference between the predicted value $f(\mathbf{x}_i)$ and the actual training value y_i . The elaboration of this idea is found in two commonly adopted functions, the first being the square loss function

$$V(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2. \quad (1.1)$$

One of the reasons for squaring the difference is that it always provides positive error values that can be directly summed, and that its first derivative is usually a continuous function, amenable to standard optimization techniques.

The second loss function is the ϵ -insensitive loss (Rifkin 2002), which is written as

$$V(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|_\epsilon \quad (1.2)$$

with the $|\cdot|_\epsilon$ function defined as

$$|x|_\epsilon \equiv \begin{cases} 0, & \text{if } |x| < \epsilon; \\ |x| - \epsilon, & \text{otherwise.} \end{cases} \quad (1.3)$$

This function is also referred to as the dead-zone loss, meaning that deviations less than tolerance value ϵ are disregarded, as can be seen in the Figure 1.1.

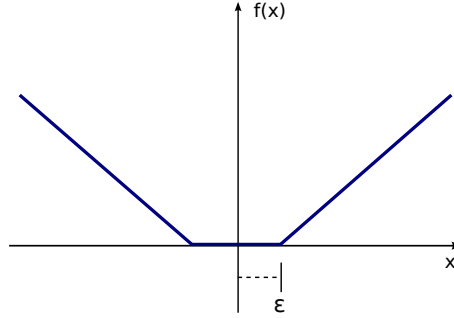


Figure 1.1: The ϵ -insensitive loss function

The ϵ -insensitive loss is relevant because of its similarity with an important loss function used in classification, the hinge loss function, and because it tends to yield sparse solutions in term of the computational complexity of the learned function.

If multiple output variables are desired $f : \mathcal{X} \rightarrow \mathbb{R}^N$, a common approach is to treat each dimension in the output variable as an independent variable, and define a separate learning algorithm for each variable.

1.1.3 Classification

In Classification, one is interested in establishing a prediction function in a restricted domain, where the output variable y is assumed to be an integer. The most common case is where y is restricted to assume two values, for example $y \in \{0, 1\}$ or $\{-1, 1\}$. In this case, the learned function f is usually defined to produce a continuous output as an intermediate step, and the final prediction is taken after applying a hard separation envelope to its output value, such as $f_{\text{class}}(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$, with

$$\text{sign}(t) \equiv \begin{cases} 0, & \text{if } t < 0; \\ 1, & \text{otherwise.} \end{cases} \quad (1.4)$$

The same function is also the first intuitive form for the classification loss function. Its non-differentiability and non-convexity characteristics, however, have complicating consequences in most algorithms and are typically substituted by equivalent convex surrogate functions. A common such function is the hinge loss (Rifkin 2002), which introduces a linear penalty if the

predicted function value and the label have opposite signs (error), and is zero when they have same signs and $f(\mathbf{x}_i) \geq 1$. Mathematically,

$$V(y_i, f(\mathbf{x}_i)) = (1 - y_i f(\mathbf{x}_i))_+ = \max(0, 1 - y_i f(\mathbf{x}_i)). \quad (1.5)$$

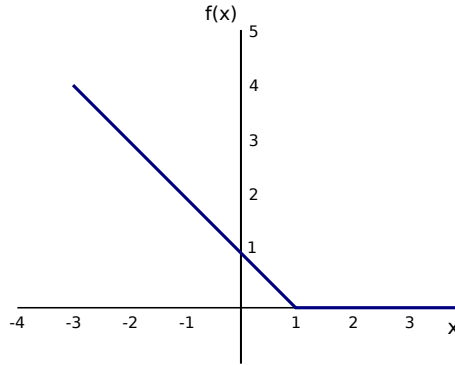


Figure 1.2: The hinge loss function

When this function is squared, it becomes continuously differentiable, and is referred to as the quadratic hinge loss. A hybrid function between the hinge and squared hinge loss is the smooth hinge loss (or Huber loss), which defines a region of quadratic transition between the zero and linear loss sections (Huber 1964).

In multi-class classification one defines the prediction variable to be $y_i \in \{1, \dots, C\}$ for $C \in \mathbb{N}$ and $C > 1$. A common approach is to define C classifying functions, trained pairwise, and assign the output class according to a choice function of the type $c_i = \arg \max f_i(\mathbf{x})$, $i = 1, \dots, C$. This approach is known as one-versus-all multi-class classification; other approaches are described in (Rifkin 2002). Another related task, but in the opposite direction, is one-class classification, or novelty detection. This is an unsupervised learning task, concerned with predicting whether a certain point is likely to have been generated by the underlying distribution of reference, as described in (Schölkopf & Smola 2002).

1.1.4 Other Loss Functions and Tasks

When the loss function is of the type of the logistic function $1/(1 + e^{-x})$, called logistic loss, the learning algorithms are referred to as Logistic Regression (Bishop 2006), and share many characteristics with the algorithms further detailed in this dissertation.

Other relevant learning tasks are those of dimensionality reduction and compression, which attempt to find equivalent sparser representations to a certain functional relationship, such as Principal Component Analysis and its kernelized and non-linear variants (Shawe-Taylor

& Cristianini 2004). Yet another important learning task is the one of clustering, which is an unsupervised learning task that seeks to find natural groupings of data points, given a multivariate distribution and a certain proximity criteria, with a pre-established or open number of reference groups definitions.

1.2 Regularization Learning

Following our objective of learning a functional relationship $f : \mathcal{X} \rightarrow \mathcal{Y}$ from the available samples and their associated loss functions, a first approach would be to attempt to minimize the value of such loss function for all the available examples, assuming a given candidate function. Such candidate function could be, for example, a piecewise linear composition, a parameterized sum of polynomials, or any other function allowing its form to be adjusted according to the examples and the loss function output. It is easy to see that, given a finite set of examples, there might be multiple solutions that could approximate the data given, and at the same time lead to equivalent overall resulting loss values (sometimes zero loss), as illustrated in Figure 1.3.

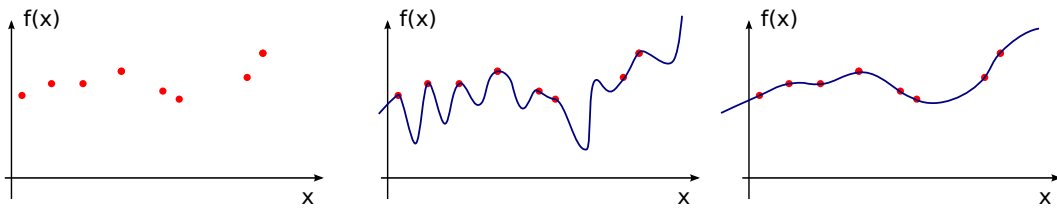


Figure 1.3: Different approximation functions with zero loss at training points

However, having in mind the overall goal that the functional relationship established should yield a valid approximation of the real underlying phenomenon being learned for future and unseen data, such a variety of functions are likely to produce different, distorted and for that purpose, invalid approximations. The reason is that, for such a finite set of training data, the degrees of freedom provided by the parameterized template function are likely to allow for infinite equally valid solutions with the same loss functional minimum value. Such problems have been referred to as *ill-posed problems*. According to (Evgeniou, Pontil & Poggio 1999), “A well-posed problem (in the sense of Hadamard, (Tikhonov and Arsenin, 1977)), is a problem for which a solution (a) exists, (b) is unique, and (c) depends continuously on the data. A problem for which at least one of the above conditions does not hold is ill-posed”.

The regularization approach adopted in this dissertation was first proposed by (Tikhonov & Arsenin 1977) and consists essentially of *including an additional penalty term in the loss function to restore the well-posedness of the solution*. Such term is commonly dependent of the

number and the module of the components of the candidate solution function, and has the goal of measuring the level of change (energy or capacity) required from the learning function in order to approximate the data presented. This regularization term, apart from reducing the set of valid solutions, encourages smoothness of the resulting function, and in turn improves the generalization capability of the function being learned. A probabilistic interpretation of such approach will be provided in the next section. The general regularization problem is formalized as:

$$\mathbf{f}^* = \arg \min_{\mathbf{f} \in \mathcal{H}} \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \Omega(\|\mathbf{f}\|_{\mathcal{H}}). \quad (1.6)$$

In this formulation, the function \mathbf{f}^* is a point in a possibly infinite-dimensional Hilbert vector space \mathcal{H} ; $V(y_i, f(\mathbf{x}_i))$ is the loss function, which is dependent of the data and the candidate function being learned; and $\Omega(\|\mathbf{f}\|_{\mathcal{H}})$ is the regularizer, an increasing functional of the norm of the candidate function. These components will be further addressed in section 1.2.2.

We now move to a brief analysis and justification of the regularization framework from the statistical point of view.

1.2.1 Statistical Machine Learning Interpretation

Statistical Learning Theory, as developed by (Vapnik & Sterin 1977, Cortes & Vapnik 1995) provides an approach to quantify the effects of using a limited amount of examples when attempting to estimate a probabilistic functional relation between the two variables of interest. For that purpose, it begins with a statement that the goal of a learning task is the minimization of the risk functional

$$R(\mathbf{f}) = \int_{\mathbf{x}, y} V(y_i, f(\mathbf{x}_i)) p(\mathbf{x}, y) d\mathbf{x} dy. \quad (1.7)$$

The risk functional is a reference mathematical construct that weights the loss functional, weighted by the underlying probability distribution. Because learning the joint probability distribution between both variables is the goal of the learning task in itself, the actual calculation of this functional cannot be performed. Instead, statistical learning theory proposes an induction principle, based on the sampled data, in order to build an empirical estimator of the expected risk functional, defined as

$$R_{emp}(\mathbf{f}) = \frac{1}{l} \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)). \quad (1.8)$$

If the number of examples provided increases to infinity, the empirical risk converges to the expected risk value. To the other end, a key result established by this theory is the derivation of a set of probability bounds for the difference between the expected and empirical risks for any loss function. One of such bounds is defined to be dependent on the number of examples provided, l , and on a quantity called the capacity h of the target function, with a given probability η . The bound depends on an increasing function ϕ , whose exact format is defined in (Vapnik & Sterin 1977), and can be written compactly as

$$R(\mathbf{f}) < R_{emp}(\mathbf{f}) + \phi\left(\sqrt{\frac{h}{l}}, \eta\right). \quad (1.9)$$

Taking the bound to the extremes on its variables, we can see that the more data we have available (l), the less the difference between both quantities. Also, the higher the capacity allowed for the candidate function (h), the higher the distance between both approximations, and the evaluated empirical risk might be disconnected from the actual expected risk, leading to a meaningless approximated function, often associated to the effect of overfitting. Such effect is referred to when the approximation function achieves low error when evaluated at the example points, but at the expense of a being highly distorted in other regions, leading to poor generalization results.

Minimizing the empirical risk functional alone is therefore an incomplete objective. An approach named Structural Risk Minimization (SRM) was proposed as a response, and consists of looking for the best trade-off between empirical risk and the capacity related term, which together compose the right hand side of (1.9). Such trade-off is also referred to as the *bias-variance* trade-off, with *bias* being related to the loss of predicting ability when a too restrictive function space is imposed to the data (estimation error, or underfitting). The term *variance* is associated with excess capacity allowed to the approximation function, which tend to render it prone to the influence of noise and individual measurements (approximation error). The SRM principle proposes a gradual search approach, in which the empirical risk functional is minimized for increasing upper bounds on learning function capacities, as measured by the quantity h . The SRM search strategy can be therefore described as a sequential minimization of the empirical risk over a gradual nesting of growing hypotheses spaces $H_1 \subset H_2 \subset \dots \subset H_M$:

$$\begin{aligned} & \underset{\mathbf{f}}{\text{minimize}} && R_{emp}(y_i, \mathbf{f}) \\ & \text{subject to} && f \in H_m \\ & && m = 1, \dots, M. \end{aligned} \quad (1.10)$$

A function \mathbf{f} that minimizes the bound (1.9) for a space with given capacity h_m is therefore sought as the solution to the SRM problem. Calculating the capacity h of a given candidate function is however not straightforward, as discussed in (Vapnik & Sterin 1977). As such, the formulation (1.10) is essentially more useful as a theoretical rather than computational framework. Nevertheless, the work of (Evgeniou et al. 1999) established a monotonic relation between a type of measure of function capacity A_m and its norm $\|\cdot\|_K$, if defined over a Reproducing Kernel Hilbert Space \mathcal{H}_K (a concept to be introduced in the next section). Norms in this space can be readily computed under some conditions. Therefore, in this space, Structural Risk Minimization can be formulated as a set of optimization problems as the one following, for increasing values of A_m :

$$\begin{aligned} & \underset{\mathbf{f} \in \mathcal{H}_K}{\text{minimize}} && \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) \\ & \text{subject to} && \|f\|_K \leq A_m \\ & && m = 1, \dots, M. \end{aligned} \tag{1.11}$$

The formulation above can be still re-arranged by defining the Lagrange multiplier λ_m and squaring the inequality constraint for computational convenience, yielding:

$$\begin{aligned} & \underset{\mathbf{f} \in \mathcal{H}_K}{\text{minimize}} && \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \lambda_m (\|\mathbf{f}\|_K^2 - A_m^2) \\ & \text{subject to} && \lambda_m \geq 0 \\ & && m = 1, \dots, M, \end{aligned} \tag{1.12}$$

which can be equivalently written, for other increasing functions of the norm of \mathbf{f} , as the regularization problem (1.6).

Solving this equivalent SRM problem still involves finding minimizing solutions for a set of λ_m values. SRM does not provide practical guidance on how to conduct such joint search, and in practice one usually resorts to cross-validation strategies such as described in Chapter 2, which are subject of investigation in this dissertation.

For more details on the characterization of the equivalence between SRM and Regularization Theory, please refer to (Evgeniou et al. 1999, Evgeniou, Poggio, Pontil & Verri 2002).

1.2.2 Solution in Reproducing Kernel Hilbert Spaces

It turns out that the formulation in 1.12 admits a convenient solution approach, one that leads to many learning task algorithms in an extensible framework. It allows for different data

types as their input and different types of output predictions (depending on the loss function). The condition to be fulfilled is that the target function should belong to a space of functions referred to as Reproducing Kernel Hilbert Space.

A brief description of a Hilbert space is that it is a vector space, possibly infinite-dimensional, which is complete and endowed with a dot product. By allowing infinite dimensional vectors, a Hilbert space includes the possibility of treating functions as its elements. The completeness property means that all Cauchy sequences in the space are converging sequences. One standard Hilbert space is the N -dimensional Euclidean space \mathbb{R}^N . Another commonly employed Hilbert space is the space of all square integratable functions, \mathcal{L}_2 . More details and reviews of these concepts are provided in (Schölkopf & Smola 2002, Rifkin 2002) and references therein.

Going one step further, a Reproducing Kernel Hilbert Space, RKHS, is a Hilbert Space in which the dot product can be computed by a kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, $k = k(\mathbf{x}, \mathbf{x}')$ that has the reproducing property. Such kernel function is defined as a dot product between maps from input space \mathcal{X} to a feature space \mathcal{H} , typically in higher dimensions (even infinite dimensional). Such mapping functions $\Phi(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{H}$ are referred to as feature maps. An important property is that most kernel functions can be computed efficiently directly in the data space \mathcal{X} . The kernel function can therefore be written as

$$k(\mathbf{x}, \mathbf{x}') \equiv \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle. \quad (1.13)$$

The reproducing property, in turn, requires that an evaluation functional $\mathcal{F}_{\mathbf{x}} : \mathcal{H}_K \rightarrow \mathbb{R}$ for a point \mathbf{x} given the vector $\mathbf{k}(\mathbf{x}, \cdot) = \mathbf{k}_{\mathbf{x}}(\cdot)$ be defined such that

$$\mathcal{F}_{\mathbf{x}}[\mathbf{f}] \equiv f(\mathbf{x}) = \langle \mathbf{f}, \mathbf{k}_{\mathbf{x}}(\cdot) \rangle \quad \forall \mathbf{f} \in \mathcal{H}_K. \quad (1.14)$$

This property is guaranteed by the Riesz Representation Theorem (Rudin 1986), and requires that $\mathbf{k}_{\mathbf{x}} \in \mathcal{H}$ is a bounded functional satisfying

$$\exists M \in \mathbb{R} < \infty \text{ such that } \forall \mathbf{x} \in \mathcal{X}, |\mathcal{F}_{\mathbf{x}}[\mathbf{f}]| = |f(\mathbf{x})| \leq M \|\mathbf{f}\|_K. \quad (1.15)$$

The reproducing property can be also understood from an analogy with the Dirac evaluation operator employed in linear time invariant systems theory. In that analogy, the dot product is equivalent to the convolution operator, the kernel function $\mathbf{k}_{\mathbf{x}}(\cdot)$ is the Dirac delta function at \mathbf{x} , and $\mathbf{f} = f(\cdot)$ is the system input signal.

The Kernel function has a key role as providing a similarity metric between any two points. Such output provides essential information for the training algorithms and target functions to

operate. Perhaps the most widely used kernel function is the Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}}, \quad \sigma > 0. \quad (1.16)$$

The Gaussian kernel, also known as a radial basis function (RBF), operates on Euclidian distance between two points, being therefore translation invariant.

Another important kernel function is the sigmoid kernel, which is dependent of the Euclidian dot product between two vectors, operating therefore on the projection between both vectors, and being rotation invariant:

$$k(\mathbf{x}, \mathbf{x}_i) = \tanh(a\langle \mathbf{x}, \mathbf{x}_i \rangle + b), \quad a > 0, b > 0. \quad (1.17)$$

Algorithms employing the sigmoid kernel are similar to multi-layer perceptron (MLP) networks with one hidden layer.

A third canonical kernel is the linear kernel, which is defined as the direct dot product between two vectors, $\langle \mathbf{x}, \mathbf{x}_i \rangle$. Powers of the type $(\langle \mathbf{x}, \mathbf{x}_i \rangle + a)^b$ are also valid polynomial kernels.

A quite remarkable property of kernel functions is that they do not necessarily have to be defined in terms of \mathbb{R}^N Euclidean vectors as their input. Data types such as strings, trees (Schölkopf & Smola 2002, Shawe-Taylor & Cristianini 2004) and graphs (Vishwanathan, Schraudolph, Kondor & Borgwardt 2010) can be also be defined as input elements, as long as a valid kernel similarity function is defined. The learning algorithms defined by the regularization framework are independent of the specific kernel employed, and different kernels functions for different data types provide an important dimension of modularity to this machine learning framework.

1.2.3 From Infinite-dimensional to l -Dimensional Optimization Problem

The regularization problem (1.6) has up to now been defined as having a function \mathbf{f} in a Hilbert Space \mathcal{H} as its target variable, and, as such, defines an infinite-dimensional optimization problem. By endowing this space with a valid kernel function, and therefore defining \mathbf{f} to belong to an RHKS \mathcal{H}_K , an extremely important property can be derived from the so called Representer Theorem (Wahba 1990). Applying this theorem allows the target function to be expressed in terms of a combination of the l training examples, and reduces the problem from an infinite dimensional space to an l -dimensional one, as we will see next. This property had historically been employed in the context of spline approximation problems; later it has been extended by

(Schölkopf, Herbrich & Smola 2001) to more general regularization formulations.

Theorem 1. Representer Theorem The optimal solution \mathbf{f}^* of the regularization problem defined for \mathbf{f} in a RKHS space \mathcal{H}_k as ¹

$$\underset{\mathbf{f} \in \mathcal{H}_k}{\text{minimize}} \quad \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \Omega(\|\mathbf{f}\|_k) \quad (1.18)$$

can be written as a finite linear combination of kernel functions:

$$f^*(\mathbf{x}) = \sum_{i=1}^l \alpha_i k(\mathbf{x}, \mathbf{x}_i) \quad (1.19)$$

or, for \mathbf{f}^* written in the feature space,

$$\mathbf{f}^* = \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i). \quad (1.20)$$

Put in words, as pointed out by (Rifkin 2002), the solution to the regularization problem can be expressed as a hyperplane in feature space, corresponding to a certain linear combination of the projection of the training examples.

Proof. The proof is by contradiction, and follows (Rifkin 2002) with a slight change in notation. Suppose \mathbf{f}^* cannot be expressed as in (1.19). In this case, it must have an orthogonal complement \mathbf{f}_0^\perp , written as

$$\mathbf{f}^* = \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i) + \mathbf{f}_0^\perp = \mathbf{f}_0 + \mathbf{f}_0^\perp, \quad \text{with} \quad \langle \mathbf{f}_0, \mathbf{f}_0^\perp \rangle = 0. \quad (1.21)$$

We will now evaluate this new candidate solution by means of the objective function of the regularization problem (1.18). Take its first term, $V(y_i, f^*(\mathbf{x}_i))$, and consider the application of the candidate function to any training point \mathbf{x}_j , which is defined as the dot product between the two vectors in feature space. By the reproducing property, we have

¹The subscript k on the norm of the function means that the norm is defined by the Kernel k.

$$\begin{aligned}
f^*(\mathbf{x}_j) &= \left\langle \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i) + \mathbf{f}_0^\perp, \Phi(\mathbf{x}_j) \right\rangle \\
&= \left\langle \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \right\rangle + \langle \mathbf{f}_0^\perp, \Phi(\mathbf{x}_j) \rangle \\
&= \left\langle \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \right\rangle \\
&= f_0(\mathbf{x}_j).
\end{aligned} \tag{1.22}$$

This shows that \mathbf{f}_0^\perp does not affect \mathbf{f}^* nor $V(y_i, f^*(\mathbf{x}_i))$, because of the orthogonality between \mathbf{f}_0^\perp and any feature space representation of a training point \mathbf{x}_j , by the definition of \mathbf{f}_0^\perp . As (Rifkin 2002) points out, this does not mean that \mathbf{f}^* and \mathbf{f}_0 are the same, but only that they assume the same values at every training point \mathbf{x}_j , $j \in 1, \dots, l$.

Now, for the second term $\Omega(\|\mathbf{f}\|_K)$, by the Pythagoras theorem and the orthogonality between \mathbf{f}_0 and \mathbf{f}_0^\perp , we know that

$$\|\mathbf{f}^*\|_K = \|\mathbf{f}_0 + \mathbf{f}_0^\perp\|_K = \|\mathbf{f}_0\|_K + \|\mathbf{f}_0^\perp\|_K, \tag{1.23}$$

and by virtue of the triangle inequality we have

$$\|\mathbf{f}^*\|_K \geq \|\mathbf{f}_0\|_K, \tag{1.24}$$

with equality only holding for $\|\mathbf{f}_0^\perp\|_K = 0$, that is, $\mathbf{f}_0^\perp = \mathbf{0}$.

Therefore, the minimum of the objective function requires that the orthogonal component be absent, which shows that the optimal function can be expressed as (1.19). This result is valid for any increasing function $\Omega(\cdot)$ of $\|\mathbf{f}\|_K$, and also implies that optimal solution is unique when both $V(y_i, f^*(\mathbf{x}_i))$ and $\Omega(\|\mathbf{f}\|_K)$ are convex functions.

□

Because the solution is expressed as a linear combination of the training vectors, the solution is often said to be in the subspace \mathcal{H}_0 spanned by the data $k_{\mathbf{x}_i}$, $i = 1, \dots, l$. The vectors or examples with non-zero combination coefficients are also known as the support vectors of the solution, which are referred to as Support Vector Machines (Cortes & Vapnik 1995).

As a useful corollary of Theorem 1, when the regularization term is defined as $\Omega(\|\mathbf{f}\|) = \|\mathbf{f}\|^2$, the squared norm can be written as

$$\|\mathbf{f}\|^2 = \langle \mathbf{f}, \mathbf{f} \rangle = \left\langle \sum_{i=1}^l \alpha_i \Phi(\mathbf{x}_i), \sum_{j=1}^l \alpha_j \Phi(\mathbf{x}_j) \right\rangle = \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) = \alpha^T \mathbf{K} \alpha. \quad (1.25)$$

Therefore, for the target solution in the form $f^*(\mathbf{x}) = \sum_{i=1}^l \alpha_i k(\mathbf{x}, \mathbf{x}_i)$, we can rewrite the regularization problem as

$$\underset{\alpha \in \mathbb{R}^l}{\text{minimize}} \quad \sum_{i=1}^l V\left(\mathbf{x}_i, \sum_{j=1}^l \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)\right) + \lambda \alpha^T \mathbf{K} \alpha, \quad (1.26)$$

which is a convex optimization problem defined in terms of the l -dimensional weight variable $\alpha = (\alpha_1, \dots, \alpha_l)$.

1.3 Regularized Least Squares

The regularization problem has been defined over an RKHS, which, through the Representer Theorem, allowed it to be reduced to an l -dimensional convex problem. Now, by using the different loss functions (defined in section 1.1.1), practical machine learning algorithms can be derived. For the loss function $V(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$ and knowing that $f^*(\mathbf{x}) = \sum_{i=1}^l \alpha_i k(\mathbf{x}, \mathbf{x}_i)$, we obtain

$$\alpha^* = \arg \min \frac{1}{l} (\mathbf{y} - \mathbf{K} \alpha)^T (\mathbf{y} - \mathbf{K} \alpha) + \lambda \alpha^T \mathbf{K} \alpha, \quad (1.27)$$

where K is an $l \times l$ matrix defined by $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, known as the Gram matrix, and \mathbf{y} is the label vector $\mathbf{y} = [y_1, \dots, y_l]$. The right hand side on (1.27) is a quadratic problem in α also known by the names of Ridge Regression and Least Square SVM (Suykens & Vandewalle 1999). This problem can actually be solved in closed form, by calculating its derivative with respect to α and setting it to zero:

$$\frac{1}{l} (\mathbf{y} - \mathbf{K} \alpha^*)^T (-\mathbf{K}) + \lambda \mathbf{K} \alpha^* = 0. \quad (1.28)$$

Solving 1.28 for α we obtain:

$$\alpha^* = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}. \quad (1.29)$$

Therefore the solution is defined as the inverse of the kernel Gram matrix added by a constant diagonal matrix for better conditioning. Such diagonal matrix is referred to as the *Ridge*.

1.4 Support Vector Machines

The application of the regularization framework to a different loss function leads to one of the most widely known and adopted machine learning algorithms, Support Vector Machines - SVM. By applying the hinge loss function $V(y_i, f(\mathbf{x}_i)) = (1 - y_i f(\mathbf{x}_i))_+$ and the expansion $f^*(\mathbf{x}) = \sum_{i=1}^l \alpha_i k(\mathbf{x}, \mathbf{x}_i)$, the regularization objective function can be rewritten as

$$\alpha^* = \arg \min_{\alpha} \frac{1}{l} \sum_{i=1}^l |1 - y_i f(\mathbf{x}_i)|_+ + \lambda \alpha^T \mathbf{K} \alpha. \quad (1.30)$$

Each hinge loss function $|\cdot|_+ = \max(0, \cdot)$ can be transformed into a linear inequality constraint for each training point, by introducing a positive slack variable ξ_i , which leads to the equivalent problem:

$$\begin{aligned} & \underset{\alpha \in \mathbb{R}^l}{\text{minimize}} && \frac{1}{l} \sum_{i=1}^l \xi_i + \lambda \alpha^T \mathbf{K} \alpha \\ & \text{subject to} && y_i \alpha^T \mathbf{k}_i \geq 1 - \xi_i, \\ & && \xi_i \geq 0, \end{aligned} \quad (1.31)$$

where \mathbf{k}_i is the i -th column (or row) of the symmetric Gram matrix \mathbf{K} .

Looking at the resulting formulation, it can be seen that (1.31) is a convex problem, with a quadratic objective function and linear inequality constraints. In that case, optimization theory tells us that strong duality holds with have zero duality gap between the primal and dual optimal values. It turns out (see (Rifkin 2002) for details) that the dual program is also a quadratic program, however with simpler constraints:

$$\begin{aligned} & \underset{\beta \in \mathbb{R}^l}{\text{maximize}} && \mathbf{1}^T \beta + \beta^T Q \beta \\ & \text{subject to} && \sum_{i=1}^l y_i \beta_i = 0, \\ & && 0 \leq \beta_i \leq \frac{1}{l}, \end{aligned} \quad (1.32)$$

where

$$\begin{aligned} Q &= \mathbf{Y} \left(\frac{\mathbf{K}}{2\lambda} \right) \mathbf{Y}, \\ \alpha^* &= \frac{\mathbf{Y} \beta^*}{2\lambda}, \\ \mathbf{Y} &= \text{diag}(\mathbf{y}). \end{aligned} \quad (1.33)$$

Historically, SVMs have been firstly proposed following a geometrically inspired derivation with the objective of finding a maximal margin separating hyperplane (Cortes & Vapnik 1995) between the training data. In that case, the maximization of the margin was equivalent to minimizing a term proportional to the square of the normal vector weights of the hyperplanes, $\|\alpha\|^2 = \alpha^T \alpha$, which corresponds to the regularization term in the case of a linear kernel. The approach was then extended to allow for the use of nonlinear kernel functions to include penalty terms for misclassified training points. This later formulation can now be interpreted as a special case of the regularization framework.

When compared to Regularized Least Squares, Support Vector Machines tend to produce solutions which are more sparse in terms of the expansion coefficients of the optimal function in its training examples. This fact is known to be caused by the 1-norm characteristic of the hinge loss function.

1.5 Computational Considerations

Resolving the RLS problem involves the inversion of a matrix of order l - the number of labeled examples. This operation has complexity of $O(n^3)$ with the Gauss-Jordan elimination method as the straight forward implementation, being reduced to $O(n^{2.3})$ with more modern implementations, such as the one presented in (Coppersmith & Winograd 1990). Because of the growing sizes of datasets being targeted in learning problems (for example resulting from computational vision applications), there has been new research in methods for defining matrix approximations to reduce the equivalent order of the dataset matrix. Such research focuses in methods for optimally defining column sampling methods and other types of spectral decompositions, and looks at the effects caused by such reduction in the resulting function prediction capability (Talwalkar 2010). These methods become even more relevant in the semi-supervised setting, where the resulting matrix is of order $l + u$, with u , the size of the unlabeled examples, typically much larger than l .

For the SVM solution computation, there has been a large volume of specific research, focusing at different optimization strategies and algorithms. Traditionally, the solution has been predominantly calculated in the dual formulation. The naïve computation in that format results in $O(l^3)$ time complexity, and $O(l^2)$ space complexity, as the kernel matrix is typically pre-computed. The first significant improvement in SVM computation was proposed by (Platt 1999) and achieved performance $O(n^2 d)$, where d is the number of dimensions of the data, via a decomposition method that calculates a pair of Lagrange multipliers at each iteration. This

method, and the ones of (Joachims 1999), with similar complexity characteristics, were the reference solvers for SVM problems during a significant part of SVM history.

The solution for SVM in its primal formulation was overlooked until (Chapelle 2007), that argued that both forms led to essentially the same complexity. The dual solution is thought to have been preferred because, while the primal formulation involves l inequality constraints, the dual formulation restructured the problem with simpler box restrictions. The primal approach, as proposed by (Chapelle 2007) reformulated the problem as an unconstrained problem, and allowed achieving control in the suboptimality of the primal α variable via early stopping, which is a desirable characteristic in many large-scale problems. Additional methods involving other approaches, such as stochastic gradient descent method, have been proposed (Shalev-Shwartz, Singer & Srebro 2007), which allow trading off accuracy against computational time with beneficial rates (even sublinear in the number of examples for some cases), while reducing drastically the space complexity requirement. A review these methods and algorithms, with a focus on their performance characteristics when applied to large scale problems, along with a summary of their complexity, is given in (Menon 2009).

Hyper-parameter Optimization

2.1 Introduction

We have seen that the previous algorithms and their optimization formulations provide a well defined way to determine the optimal combination weights of basis functions that minimize the regularized objective function as a consequence of the representer theorem. However, these formulations take important parameters as given, such as the regularization constant λ and any of the parameters of the kernel basis functions. Because these parameters are not optimized in the training stage, they are referred to as *hyper-parameters*.

Although the weight coefficients α are assured to be optimally determined according to the regularization function at the training stage, the influence of the variation of the hyper-parameter values θ on the performance of the algorithm can be easily observed. Therefore, in the process of finding a good function approximation for the problem at hand, a complementary stage, which we refer to as hyper-parameter optimization, is required. Taking this into account, the function we need to determine depends not only on the data samples, but also on the weight coefficients and the hyper-parameters, $f(\mathbf{x}_i, \alpha_i, \theta)$.

Hyper-parameter optimization can also be understood as part of a larger problem, referred to as *model selection* (Hastie, Tibshirani & Friedman 2001). Model selection is a broad term which might include the initial definition of the learning algorithm to be adopted, as well as the definition of the types of kernel functions to be considered as basis for the function approximation expansion. In this text, we are considering that these definitions have been established a priori, as a result of prior knowledge of the problem at hand.

To guide the search for optimal hyper-parameters, one needs to define an objective function and a search strategy. As with the training stage, the overall hyper-parameter optimization process is essentially guided by information available from the underlying process which we want

to approximate, provided as a collection of data samples. Because the ultimate objective of the learning process is to assure that the function learned will be able to generalize to future and unseen data, one approach to model selection is to test the performance of the function against data to which it did not have access to during the training stage. In order to assure such level of independence, one usually splits the examples set in different partitions, such as *training, validation and test sets*. In the most strict case, the training set is used for the optimization of the weights, the validation set is aimed at optimizing hyper-parameters, and the test set is used for the evaluation of the generalization of performance of the final function model. The actual allocation of such partition will mostly depend on the amount of data available, the computational complexity supported, and the statistical properties desired for the final estimate. In the next sections, we will describe different aspects to this problem, attempting to provide an optimization approach for the selection of hyper-parameter values with some candidate objective functions, while looking at search strategies that make the most efficient use of the available data.

2.2 Model Assessment

The search for better hyper-parameters can be conducted if one provides an estimate for the performance of the learned function over independent data. The proposition of such estimate faces two main questions: “what are the metrics to quantify the predictive quality of the function?” And “how does one define the data partition between the training, validation and test activities?” This is the subject of the next sections.

2.2.1 Validation and Test Functions

The definition of the validation function follows essentially the same general approach used for the loss function for the training stage, where one typically accounts for the errors or deviations in the prediction of the candidate function, as compared to the actual value of the samples target variable. As we have seen, such functions are slightly different for Regression and Classification tasks.

Regression

In Regression, one wishes to evaluate the function prediction difference to the actual prediction variable, for each future data point, as a continuous variable. Therefore, most functions accumulate deviations from the true value to the prediction function according to

a distance norm. For the one-dimensional space of the output variable y , the squared error, $(f(\mathbf{x}) - y)^2$, or the absolute error, $|f(\mathbf{x}) - y|$, are commonly adopted.

Variations of such error functions can be defined according to special characteristics of the problems at hand, as, for example, providing ϵ -tolerance regions, or introducing *max* norms.

Classification

In Classification, for the reference binary case, a common characteristic is that the positive and negative classes might (and commonly do) have different penalties for errors associated to each situation. For example, erroneously assigning a class as negative (false negative error) might be much more harmful (or costly) than an false positive error. A practical example is in medical diagnosis, where the error of not detecting a disease in a screening test must incur a much higher penalty. Therefore, it is usual that both types of errors are accounted for with different weights, as will be described shortly.

The loss function $L_{\text{val}}(f(\mathbf{x}), y) = f_{\text{val}}(t_p, f_p)$ is defined as function of the following key quantities:

- n_p , number of positive examples;
- n_n , number of negative examples;
- t_p , number of positive examples correctly classified: $t_p = \sum_{i:y=+1}^n s(f(\mathbf{x}_i), y_i)$, with n being the total number of examples;
- and f_p , number of negative examples incorrectly classified: $f_p = \sum_{i:y=-1}^n [1 - s(f(\mathbf{x}_i), y_i)]$.

In the last two cases, the function $s(f(\mathbf{x}_i), y_i)$ is 1 if the data is correctly classified, and 0 if it is incorrectly classified. With these quantities, some additional validation functions can be defined:

Error Rate	$\text{er} = \frac{n_p - t_p + f_p}{n}$	the percentage of incorrect prediction
Weighted Error Rate	$\text{wer} = \frac{(n_p - t_p) + \eta f_p}{n_p + \eta n_p}$	defined in terms of the ratio η of the cost of negative to positive misclassification
Precision	$\text{pr} = \frac{t_p}{t_p + f_p}$	percentage of positive classifications that are correct
Accuracy	$\text{acc} = \frac{t_p + t_n}{t_p + f_p + t_n + f_n}$	percentage of all classifications that are correct
Recall	$\text{re} = \frac{t_p}{n_p}$	percentage of positive examples that are correctly classified
F-Measure	$\text{F} = \frac{2 \text{pr re}}{\text{pr} + \text{re}} = \frac{2t_p}{n_p + t_p + f_p}$	is the harmonic mean of precision and recall

Table 2.1: Validation Functions for Classification

Another validation function that is sometimes referred to is the *Area under ROC Curve*, where ROC stands for Receiver Operating Characteristics. Such metric yields a number between 0.5 and 1, and is statistically equivalent to the probability that a classifier will assign a higher value to a positive instance, higher than a randomly chosen negative. An extensive description of this function is presented in (Fawcett 2006).

Smooth Sigmoidal Approximation for Classification Validation

The functions defined for classification evaluation in the Table 2.1 are functions of the integer count of correct and incorrect predictions, as provided by the discrimination function $s(f(\mathbf{x}_i), y_i)$. This function is not continuously differentiable, a fact that might exclude some continuous search algorithms. In some cases, a smooth approximation for the s function,

$$\tilde{s}(f(\mathbf{x}_i), y_i) = \frac{1}{1 + e^{-\rho y_i f(\mathbf{x}_i)}} \quad , \quad (2.1)$$

is adopted, where the parameter ρ regulates the smoothness of the approximation.

2.2.2 Bias-Variance Decomposition

The second question to be answered is related to the different ways of partitioning the data between training, validation and testing activities. The analysis of a fundamental characteristic of the generalization error, the bias-variance decomposition, provides important elements to guide such partition, as described in (Hastie et al. 2001). To build a reference model, we assume a square loss function, with input measurements modeled as $y = f(\mathbf{x}) + \epsilon$, with $Var(\epsilon) = \sigma_\epsilon^2$. That is, input values are the true function values plus a measurement error following a Gaussian distribution.

The expected error is therefore defined as the expectation of the squared difference between the two quantities. By expanding the square and applying mathematical equivalences for the variance, a decomposition of the error expression can be produced:

$$\begin{aligned} R_{sq}(\mathbf{x}_0) &= \mathbb{E}[\mathbf{y} - f(\mathbf{x}_0)]^2 \\ &= \sigma_\epsilon^2 + [\mathbb{E}f(\mathbf{x}_0) - f(\mathbf{x}_0)]^2 + \mathbb{E}[f(\mathbf{x}_0) - \mathbb{E}f(\mathbf{x}_0)]^2 \\ &= \sigma_\epsilon^2 + \text{Bias}^2[f(\mathbf{x}_0)] + \text{Var}[f(\mathbf{x}_0)] \\ &= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}. \end{aligned} \quad (2.2)$$

The first term, the variance of the measurements, cannot be eliminated unless $\sigma_\epsilon = 0$. The second term is the squared bias, that is, the amount that the estimation is expected to differ

from its true mean. The last term is the expected squared deviation around its true mean. The bias term is related to what is defined as the *approximation* error, which is caused by possible constraints on the function model imposed by the approximation. The variance term is related to the *estimation* error, which is caused by an occasional excess of adaptation of the model to peculiarities of the sample, and the additional curvature introduced in other regions of the function image as a result of that effort. Generally, we can say that the more complex the function model is allowed to be, the lesser will be its bias, and the higher will be its variance component.

Optimism in Training Error

When empirically evaluating the error performance of the approximation function over its training set, it can be shown that such estimate will almost invariably be an optimistic estimate of the true generalization error. An intermediate approximation can be statistically derived if one defines the quantity R_{in} , the in-sample error, to be the expectation over additional y values (mathematically) taken at the training points. The optimistic excess can be estimated through the definition of a factor R_{gap} such that $R_{in} = R_{emp} + R_{gap}$. Such decomposition is analogous to the Structural Risk Minimization bound seen in Chapter 1. According to (Hastie et al. 2001), using R_{in} thus defined, the expectation of the optimistic factor can be defined, for different loss functions including square loss and 0-1 losses, as:

$$\mathbb{E}[R_{gap}] = \frac{2}{n} \sum_{i=1}^n \text{Cov}[f(\mathbf{x}, y)]. \quad (2.3)$$

This result has an interesting interpretation (Hastie et al. 2001): it tells that the amount by which the empirical risk underestimates the true error depends on how strongly y_i affects its own prediction. As a reference model, such term can be obtained for the simplified case where the approximation is defined as a linear fit with b basis functions as being equivalent to $b\sigma_\epsilon^2$.

$$\mathbb{E}[R_{in}] = \mathbb{E}[R_{emp}] + 2\frac{b}{n}\sigma_\epsilon^2. \quad (2.4)$$

This approximation, referred to as the C_p (conceptual predictive) statistic, although derived for the restricted class of linear functions, is in qualitative agreement with the risk bounds defined in Chapter 1. It is increasingly dependent of a quantity related to function complexity, and decreasingly related to the number of examples provided for the training.

This statistic is also the basis for the In-Sample class model assessment criteria that will be mentioned next.

2.2.3 Partition Strategies

We now look at different partition strategies and how they can be used to allow for the estimation of the generalization error.

Hold-out

The simplest strategy for partitioning the data set is to randomly assign a given percentage of samples for the three tasks we have mentioned (training, validation and test), for example, 50, 25 and 25 percent, respectively. One can make a quick qualitative analysis on the bias and variance characteristics of such split: as the number of data samples available for training in this case is only half of the total, a significant bias can be incurred, as compared to a case where the majority of data would be used. In terms of variance, because only one trial or random split is employed, one might expect that the output hyper-parameters and performance estimates to be reasonably dependent on the specific data partition that has been drawn.

In response to these characteristics, and seeking to better explore the information available at any example set, two other broad strategies have been devised, *In-Sample Estimation* and *Cross-Validation*. A common characteristic to both is that they attempt to eliminate the need for a separate validation set.

In-Sample Prediction Error Estimation

In-sample estimation criteria look at statistical properties of the models and error loss functions to arrive at an estimate of the optimism factor between the empirical error observed and the expected generalization error. Such factors are similar to the one mentioned previously for the C_p statistic. Therefore, for equivalent empirical error performances obtained using a given function, in-sample estimation criteria will favor the adoption of simpler function classes.

One widely referred of such criteria, being recognized as the first for this purpose, is the *Akaike Information Criterion - AIC* (Akaike 1973). The AIC, in its general form, is defined in terms of a log likelihood loss function and a term that is increasing with the number of parameters. For a Gaussian model, it can be expressed as

$$\text{AIC}(\theta) = R_{\text{emp}}(\theta) + 2 \frac{b(\theta)}{n} \sigma_{\epsilon}^2 \quad (2.5)$$

which coincides with the C_p statistic. While for linear models $b(\theta)$ is the direct number of basis functions used to fit the data, for nonlinear or more complex models, a corresponding effective number of parameters quantity needs to be defined, which might depend on the hyper-parameter

vector θ . A brief discussion on the calculation of effective number of parameters is found in Section 7.6 of (Hastie et al. 2001).

Other widely mentioned criteria are the BIC (Bayesian Information Criterion) and the MDL (Minimum Description Length). Both are defined from different framework approaches (the MDL is derived from an information theoretic optimal coding principle), but arrive at similar formulations. Their main difference lies in relative emphasis towards simpler or more complex functions. Compared to AIC, BIC and MDL provide more severe penalties for complexity, therefore favoring simpler models. Please refer to (Hastie et al. 2001) for more detailed exposition of each criterion.

Although these criteria are valuable for suggesting insights and providing some degree of relative merit between different models, their full application to models with non-linear dependencies on their parameters is limited, as their derivation in such cases have to be specific and are usually non-trivial.

2.2.4 Cross-Validation

A simple and general method for estimating generalization performance is based on assigning rotating partitions from the same overall training set sub-partitions for training and validation. This method effectively uses training data playing both the training and validation roles, while assuring the independence condition. The method is also generally and independently applicable to any type of non-linear dependency between the function and its hyper-parameters. More formally, for the set of n training examples one can define a mapping $\kappa(t) : \{1, \dots, n\} \rightarrow \{1, \dots, T\}$ which will assign each data to a given validation partition t , given an initial random assignment. The negative sign on the partition index denotes the exclusion of that element from the set. The T partitions, or *folds*, are assumed to be of the same size. The procedure consists in, for a given vector of hyper-parameters θ , execute T trainings with training subsets, each composed of $\frac{T-1}{T}n$ examples, and $\frac{n}{T}$ examples as validation elements, where the validation elements at each fold are defined by the $\kappa(t)$ mapping. The cross validation estimate can be computed by averaging the accumulated loss over the T partitions over which it was applied, as

$$CV(f(\mathbf{x}_i, \alpha, \theta)) = \frac{1}{T} \sum_{t=1}^T L_{\text{val}}(y_i, f^{-\kappa(i)}(\mathbf{x}_i, \alpha, \theta)). \quad (2.6)$$

This procedure is known as *k-Fold Cross Validation* (Hastie et al. 2001). We have replaced k with the index t to avoid notation conflict with the kernel function variable. In the extreme case where the number of folds T is equal to the number of examples n , the procedure will be

equivalent to running n model trainings with $n - 1$ data examples, which are then tested against the single excluded example at each fold. In this case, the cross validation procedure is referred to as *Leave-One-Out Cross Validation* (LOOCV) (Hastie et al. 2001). This method is known to provide an almost unbiased estimation, as it consistently uses the largest amount of training data available for each training, being closest to the final model in terms of use of training data. A side effect is, however, felt in opposite direction, when one looks at the resulting variance behavior. Because the $T = n$ training sets are so similar amongst themselves, there will be no averaging or compensatory effects between different training split characteristics, as compared to what one would have with larger validation (and smaller training) splits. Moreover, a concern that is intuitive, and often follows the presentation of LOOCV, is its computational cost, as its calculation would in theory involve running the same number of training sessions as the number of examples. Fortunately, as we will see in the next session, there are closed-form estimates for some cross validation models which can be obtained after a single training session with all the data.

2.2.5 Leave-One-Out Bounds

One important fact about some regularization based models is that their optimality conditions, either before or after the optimization problem has been solved, can be written as a system of linear equations. This is immediately verifiable for the Regularized Least Squares algorithms (and their related Least Square SVM method), and is also true for SVM's (Keerthi, Sindhvani & Chapelle 2007). It turns out that due to the linearity, and given the canonical leave-one-out split structure, where a single example is extracted at each turn, a closed form expression for the difference in prediction can be derived. See (Cawley & Talbot 2004, Rifkin 2002, Rifkin & Lippert 2007). We will present the resulting expression for the case of Regularized Least Squares.

LOOCV Bounds for Regularized Least Squares

Remembering the notation for Regularized Least Squares, we have that a solution to the training problem is defined by the linear system $[\mathbf{K} + \lambda \mathbf{I}] \alpha = \mathbf{C} \alpha = \mathbf{y}$. We will denote the resulting function trained over the whole training set (without excluding an example) as f_S , and as $f_S^{(-i)}$ the function trained with the training set, but excluding the i_{th} example.

The i_{th} residue, or the difference between the target variable and the predicted value for the i_{th} LOOCV round can be shown to be calculated exactly as

$$L_{\text{val}}^{\text{LOOCV}}(\mathbf{x}_i, y_i) = y_i - f_S^{(-i)}(\mathbf{x}_i) = \frac{y_i - f_S(\mathbf{x}_i)}{1 - C_{ii}^{-1}} = r_i, \quad (2.7)$$

which, with further manipulation, as described in (Rifkin & Lippert 2007), can be also written as

$$r_i = \frac{\alpha_i}{C_{ii}^{-1}}. \quad (2.8)$$

Thus, the i_{th} residue is the ratio of its kernel weight component to C_{ii}^{-1} , the i_{th} element of the diagonal of the inverse of the training matrix. An extension of this result for the case of k -Fold Cross Validation (for $T < n$) is provided in (Pahikkala, Boberg & Salakoski 2006), with an increase in computational complexity as the size of the validation set at each fold increases.

PRESS Criterion

The PRESS criterion (Predicted Residual Sum of Squares) (Allen 1974) is defined as the sum of the squared differences from the target variable to its leave-one-out prediction. It is a continuously differentiable function, closely related to the leave-one-out partition, defined as

$$\text{PRESS}(\theta, y_i, f(\mathbf{x}_i)) = CV(f(\mathbf{x}_i, \alpha, \theta)) = \sum_{i=1}^n [y_i - f_S^{(-i)}(\mathbf{x}_i)]^2 = \sum_{i=1}^n r_i^2. \quad (2.9)$$

It can be applied to both regression and classification problems if the output function for the latter is used before the application of the hard discriminant function.

2.3 Formulation as an Optimization Problem

The validation process, with the objective of searching for appropriate hyper-parameters according to a validation merit function, can be also viewed as an optimization problem. In (Bergstra & Bengio 2012), such problem is addressed from an expectation point of view, as

$$\theta^{(*)} = \arg \min_{\theta \in \Theta} \mathbb{E}_x [L_{\text{val}}(y, f(\mathbf{x}, \theta, \alpha))]. \quad (2.10)$$

Because the predicting function is typically found as a result of an optimization procedure during training, problem (2.10) ultimately involves optimizing an output from another optimization process. If the above expectation is replaced with its empirical approximation through cross-validation, this problem can be generally represented in a bi-level optimization form, as

$$\begin{aligned}
& \underset{\alpha_t, \theta}{\text{minimize}} && \sum_{t=1}^T CV(f^{-\kappa(i)}(\mathbf{x}_i, \alpha_t, \theta)) \\
& \text{subject to} && \\
& && \alpha_t \in \arg \min_{\alpha_t} \{V(\alpha_t, \theta, \mathbf{x}_{\kappa(i)}, y_{\kappa(i)})\} \\
& && \theta \in \Theta \\
& && i = 1 \dots n; \ t = 1 \dots T.
\end{aligned} \tag{2.11}$$

Such formulation, despite not being usually amenable for a direct numerical implementation or solution, is useful as a reference framework for alternative castings of the hyper-parameter optimization problem that we are addressing.

2.4 Hyper-parameter Search Strategies

In the following sections we present some relevant search strategies that can be applied to problem (2.11), namely exhaustive grid search, random search, and gradient optimization-based approaches.

2.4.1 Grid Search

Grid search is the simplest and perhaps the mostly used search strategy when the number of hyper-parameters is low. This is usually the case for the simplest SVM classifier models, where the regularization constant λ and one or two kernel parameters need to be optimized. Usually, the scale variation of hyper-parameters is taken to be logarithmic, and a grid with a pre-specified resolution is defined, for example $\lambda \in \{2^{-15}, 2^{-13}, \dots, 2^3, 2^5\}$; $\sigma \in \{2^{-15}, 2^{-13}, \dots, 2^1, 2^3\}$. Therefore, for the general search space $\Theta = \Theta_1 \times \dots \times \Theta_h$, with corresponding cardinality $|\Theta| = \prod_{i=1}^h |\Theta_i|$ it can be easily seen that the computation complexity for grid search is exponential in the number of hyper-parameters. Therefore, grid search is an uninformed or blind-search algorithm that has its application limited to lower dimensional problems.

2.4.2 Random Search

The use of random search has been recently proposed by (Bergstra & Bengio 2012) as an alternative to higher dimensional problems where simplicity of implementation is still a foremost concern. It is based on the key observation that a uniform distribution of points in a higher dimensional space is more likely to evenly explore such space, given a limited number of points.

This is in contrast with grid search, as in this later case, the search is conducted by iterating sequentially over each dimension, and the algorithm will usually exhaust the search in a given dimension before starting another. In (Bergstra & Bengio 2012) the authors observe that, in higher dimensional hyper-parameter spaces, it is likely that there will be a number of parameters which will have lower impact in the function performance. Such spaces, therefore, are spaces with a lower effective dimensionality, and by exploring it according to uniform distributions of points, “Random Search has the same efficiency in the relevant subspace as if it had been used to search only the relevant dimensions”. Randomly generated points provide a more dense exploration of a dimension that is more relevant to the objective being optimized, or for the dimensions where variables have a lower level of coupling. The generation of random points is simply performed according to a uniform distribution random generator, scaled in each hyper-parameter dimension by the parameter limits:

$$\theta_j = \theta_{min_j} + (\theta_{max_j} - \theta_{min_j}) U(0, 1); \quad j = 1 \dots h. \quad (2.12)$$

The quasi-uniform random exploration of the search space has the additional advantage of allowing a more even balance between computational effort and exploration in each dimension, thus permitting more control and early stopping in the search. Because the generation and traversal through the points is independent, it also enables a simple parallel computing solution, as no state control or coordination needs to be implemented.

2.4.3 Gradient Optimization

Gradient optimization uses information from the rate of decay of the objective function in each dimension, allowing the definition of a direction of assured local descent in the search space. The convergence to a global optimal solution is not generally guaranteed. There is a wide body of methods and literature on different gradient-based methods, which depend, amongst other factors, on the mathematical properties of its objective function (convex, quadratic, etc), types of constraints (box, linear, semi-definite matrices, etc). The reader is referred to the excellent books (Boyd & Vandenberghe 2004, Nocedal & Wright 2006) for further information about this important class of optimization methods. In the following we proceed to the derivation of the gradients for the leave-one-out validation bound function with respect to its regularization and kernel hyperparameters. These gradients can be provided to different general optimization solvers, as done in the Computational Experiments section presented in Chapter 4.

LOOCV Gradients

Taking advantage of the facts that a closed-form, exact bound, can be derived for the leave-one-out cross validation strategy, and that PRESS is a continuously differentiable criterion, we now derive its gradient with respect to its hyper-parameters θ , according to the exposition in (Cawley & Talbot 2007). Re-stating the definition of the criterion, we have

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n r_i^2, \quad \text{with } r_i = \frac{\alpha_i}{C_{ii}^{-1}}. \quad (2.13)$$

Applying the chain rule and the product (division) rule on the residue, yields

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^l \frac{\partial J(\theta)}{\partial r_i} \frac{\partial r_i}{\partial \theta_j}, \quad \text{with} \quad (2.14)$$

$$\frac{\partial r_i}{\partial \theta_j} = \frac{\partial \alpha_i}{\partial \theta_j} \frac{1}{C_{ii}^{-1}} - \frac{\alpha_i}{(C_{ii}^{-1})^2} \frac{\partial C_{ii}^{-1}}{\partial \theta_j} \quad \text{and} \quad \frac{\partial J(\theta)}{\partial r_i} = r_i^{-i} = \frac{\alpha_i}{C_{ii}^{-1}}. \quad (2.15)$$

The derivatives needed are now $\partial \alpha_i / \partial \theta_j$ and $\partial C_{ii}^{-1} / \partial \theta_j$. For the inverse matrix derivative, we obtain¹

$$\frac{\partial \mathbf{C}^{-1}}{\partial \theta_j} = -\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \theta_j} \mathbf{C}^{-1}. \quad (2.16)$$

For the α derivative, knowing that $\alpha = \mathbf{C}^{-1} \mathbf{Y}$, it is possible to write

$$\frac{\partial \alpha}{\partial \theta_j} = \frac{\partial \mathbf{C}^{-1} \mathbf{Y}}{\partial \theta_j} = -\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \theta_j} \mathbf{C}^{-1} \mathbf{Y} = -\mathbf{C}^{-1} \frac{\partial \mathbf{C}}{\partial \theta_j} \alpha, \quad (2.17)$$

so that all the derivatives are functions of the \mathbf{C} matrix derivative.

Going one step further, for the regularization parameter $\theta_j = \lambda$ we have:

$$\frac{\partial \mathbf{C}}{\partial \lambda} = \mathbf{I}, \quad \text{and} \quad (2.18)$$

$$\frac{\partial \alpha}{\partial \lambda} = -\mathbf{C}^{-1} \alpha. \quad (2.19)$$

For the kernel parameters $\theta_j = \theta_K$ we can generally write:

$$\frac{\partial \mathbf{C}}{\partial \theta_K} = \frac{\partial \mathbf{K}}{\partial \theta_K} \quad (2.20)$$

¹This identity can be easily derived by applying the product differentiation rule to the product $(\mathbf{C} \mathbf{C}^{-1})' = (\mathbf{I})'$ and isolating $(\mathbf{C}^{-1})'$.

and taking this definition, for example $\theta_j = \sigma$, the parameter for the Gaussian kernel,

$$\frac{\partial k(\mathbf{x}_i, \mathbf{x})}{\partial \sigma} = k(\mathbf{x}_i, \mathbf{x}) \frac{\|\mathbf{x}_i - \mathbf{x}\|^2}{\sigma^3}, \quad (2.21)$$

which concludes the derivation of the LOOCV validation function gradient with respect to its hyper-parameters.

2.5 Multiple Kernel Learning

A different approach to selecting or searching for the optimal parameter values for a given kernel function is called Multiple Kernel Learning (MKL). In MKL, the kernel function k_η is defined as a combination f_η of other base kernel functions k_m , which might have different functional formats and parameter values, according to

$$k_\eta = f_\eta(\{k_m(\mathbf{x}_i^m, \mathbf{x}_j^m)\}_{m=1}^q). \quad (2.22)$$

The problem in MKL is defined as finding the optimal combination of such base kernels such that a target function related to the validation error is minimized. One possible function is the linear combination, written as

$$k_\eta = \sum_{m=1}^q \eta_m k_m(\mathbf{x}_i^m, \mathbf{x}_j^m), \quad \eta \in \mathbb{R}^q. \quad (2.23)$$

The MKL problem relies on finding the values of $\eta = (\eta_1, \dots, \eta_q) \in \mathbb{R}^q$. A wide variety of combination strategies has been recently proposed by a number of authors. A comprehensive review and a taxonomy of the aspects explored in different combination strategies is provided in (Gönen & Alpaydın 2011). An application of this approach will be provided in more detail in Chapter 4.

Semi-supervised Learning: Manifold Regularization

3.1 Semi-supervised Learning

As briefly introduced in Chapter 1, semi-supervised learning is defined as the ability to use both labeled *and* unlabeled data in order to improve the predictive or adaptive ability of an algorithm. If one looks at the practical aspects of collecting and providing training data, it is easy to conclude that the cost of obtaining unlabeled data is much smaller than its labeled counterpart. Not only unlabeled data do not require a supervisor or expert to define their labels, such data might be autonomously collected by the machine itself during its operation. That idea leads to the notion of continuous learning, and alludes to the way humans are believed to process data in order to improve their experience and understanding, being constantly exposed to natural stimuli. As a result, as pointed by (Belkin, Niyogi & Sindhwani 2006), “if we are to make progress in understanding on how natural learning comes about, we need to think about the basis of semi-supervised learning”.

Moving to a mathematical interpretation, we have already modeled the learning problem as that of establishing a probabilistic functional relationship between two variables $f : \mathcal{X} \rightarrow \mathcal{Y}$. Equivalently, we are interested in learning the joint probability distribution $P(\mathbf{x}, \mathbf{y})$, which can be decomposed as $P(\mathbf{x}, \mathbf{y}) = P(\mathbf{x})P(\mathbf{y}|\mathbf{x})$. In that case, one might attempt to use the information present on the unlabeled data to learn the base distribution $P(\mathbf{x})$. The validity of such principle, however, relies on some assumptions, formulated by most semi-supervised learning approaches. They are:

- Cluster assumption: points in the same cluster are likely to be of the same class.

- Low density separation: the decision boundary should lie in a low-density region.
- Semi-supervised smoothness assumption: if two points $\mathbf{x}_1, \mathbf{x}_2$ are close in a high-density region, then so should be the corresponding outputs y_1, y_2 .
- Manifold assumption: high-dimensional data lie (roughly) on a low-dimensional manifold.

The Manifold Regularization approach followed in this text relies on the last assumption, and is addressed in the following sections. For an interesting discussion on each of the other assumptions, please refer to (Chapelle, Schölkopf & Zien 2006).

3.1.1 Transductive and Inductive Learning

Semi-supervised learning algorithms can also be described as being transductive or inductive, according to the way they are expected to make their predictions. In inductive type algorithms, the function learned is presented with both labeled and unlabeled data, and it is expected to be able to generalize data to which it did not have access at training time. Its performance is measured fundamentally on its success on the unseen data. Transductive type algorithms, in turn, are given both labeled and unlabeled data, but are only expected to achieve good performance in making correct predictions on the class of the unlabeled data to which they had access at training time. In transductive algorithms, the output function learned does not even need to be (and often is not) defined for unseen data. Transductive algorithms are defined as such in response to what became to be known as Vapnik’s principle (Chapelle et al. 2006): “When trying to solve some problem, one should not solve a more difficult problem as an intermediate step”. Therefore, one can understand the goal of transductive learning as a less complex one, to be adopted if the application under consideration and its prediction regime allows for its use. In this work, however, we are interested in Manifold Regularization as an inductive type of algorithm, capable of generalizing well for future or unseen examples. The definitions that follow will be according to the work presented in (Belkin et al. 2006).

3.1.2 The Manifold Assumption

The Manifold Assumption is at the core of the Manifold Regularization approach. One should therefore take a moment to examine its validity. According to (Belkin et al. 2006), “in many natural situations, it is clear that the data are supported on a low-dimensional manifold. This is often the case when points are generated by some physical process. For example, in speech production the articulatory organs can be modeled as a collection of tubes. The space of

speech sounds is therefore a low-dimensional manifold parameterized by lengths and widths of the tubes. Photographs of an object from various angles form a three dimensional submanifold of the image space. In other cases, such as in text retrieval tasks, it may be less clear whether a low-dimensional manifold is present”. This is the same reasoning behind the approach of the inverse problem, where a given set of natural laws are supposed to restrict the space of information generated by the phenomenon under study. Such Manifold Assumption therefore appears to have place in many applications.

The Manifold Assumption is also related to a key challenge in statistical learning, which is the curse of dimensionality. As the number of dimensions considered for a given problem grows, the “volume of ignorance” in that space grows in exponential proportion. It means that the number of data points required to maintain a given information density also grows exponentially. As pointed out in (Chapelle et al. 2006), “this is a problem that directly affects generative approaches that are based on density estimates in input space. A related problem of high dimensions, which may be more severe for discriminative methods, is that pairwise distances tend to become more similar, and thus less expressive. If the data happen to lie on a low-dimensional manifold, however, then the learning algorithm can essentially operate in a space of corresponding dimension, thus avoiding the curse of dimensionality”.

3.2 Manifold Regularization

Manifold Regularization was proposed by (Belkin et al. 2006) and consists fundamentally in extending the Regularization Learning approach to take in account both labeled and unlabeled data in an inductive way. It achieves this goal by including an additional regularizer which imposes the function being learned to be smooth on the estimated manifold of the data, thereby enforcing the manifold and the semi-supervised smoothness learning assumptions. In this setting, the regularization problem is expressed as

$$\underset{\mathbf{f} \in \mathcal{H}_K}{\text{minimize}} \quad \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \gamma_A \|\mathbf{f}\|_K^2 + \gamma_I \|\mathbf{f}\|_I^2 \quad (3.1)$$

where the $\|\cdot\|_I$ norm refers to the norm of the function in the *intrinsic* subspace, which is the space defined by the manifold approximation, and γ_A and γ_I are hyperparameters. The original space, over which the predicting function is defined, is called the *ambient* space, and its norm, once the function is assumed to be in a RKHS, can be calculated as previously defined.

Two main questions should arise while looking at the Manifold Regularization problem and trying to reach a solution which can be computed from data. How does one estimate the

manifold from the data? And how does one evaluate the smoothness of the function over such manifold?

According to (Belkin et al. 2006) the manifold structure can be empirically estimated from available data through the graph Laplacian associated with the data. The graph Laplacian is a matrix which maps the data as a graph, according to a data neighborhood association criterion, thereby producing a distance criteria over the manifold space. The next subsections detail the computation of such matrices.

The smoothness of the candidate function over that manifold can be evaluated by associating the value of the function according to neighborhood information given by the manifold approximation graph (Laplacian matrix). As an intermediate step, a data adjacency matrix \mathbf{W} is defined, whose elements are non-zero for pairs of points which are evaluated to be neighbors (according to a criterion to be defined soon). Then, the smoothness penalty can be estimated by summing the contribution of all pairs of data considered to be neighbors (the non-zero matrix elements w_{ij}), weighted by the difference of the target function value for the two points of the pair. The penalty thus defined tells the function not to assume highly different values along its neighbor points, and, as such, enforces the function smoothness over the approximated manifold. Adopting such penalty definition as the Manifold Regularization term, the regularization problem becomes

$$\underset{\mathbf{f} \in \mathcal{H}_k}{\text{minimize}} \quad \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \gamma_A \|\mathbf{f}\|_K^2 + \frac{\gamma_I l}{(u+l)^2} \sum_{i,j=1}^{l+u} (f(\mathbf{x}_i) - f(\mathbf{x}_j))^2 w_{ij} \quad (3.2)$$

where the third term corresponds to the norm of the function in the intrinsic space. In the original work where this formulation is proposed, (Belkin et al. 2006) shows that such expression is indeed a valid norm in a given (Reproducing Kernel) Hilbert Space. By a re-arrangement of the components of the third term, and defining \mathbf{f}_{l+u} as the vector containing the evaluation of the target function $f(\mathbf{x}_i)$ in all labeled and unlabeled points \mathbf{x}_i , along with a the Laplacian matrix \mathbf{L} derived from the adjacency matrix \mathbf{W} , the above expression can be equivalently written as:

$$\underset{\mathbf{f} \in \mathcal{H}_k}{\text{minimize}} \quad \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \gamma_A \|\mathbf{f}\|_K^2 + \frac{\gamma_I l}{(u+l)^2} \mathbf{f}_{l+u}^T \mathbf{L} \mathbf{f}_{l+u}. \quad (3.3)$$

We can see that the \mathbf{L} matrix defines a quadratic form, which is, by its equivalence to the third term sum in (3.2), positive semi-definite. This quadratic form is always positive for arbitrary $f(\mathbf{x})$ values, provided the working definition that w_{ij} are always positive distance quantities.

The effect of the inclusion of the manifold smoothness penalty is illustrated in Figure 3.1. In this figure, an experiment using the *Two moons* dataset is presented (Belkin et al. 2006). In this dataset, only one labeled example is provided for each class, represented by the circle and diamond markers. The remaining points are provided without their labels. In the leftmost plot, the algorithm does not take in account unlabeled data (what is equivalent to setting the γ_I hyperparameter to zero). The resulting separating surface, although being capable of achieving general nonlinear forms, becomes in this case a straight line located midway between the labeled training points, which is the form that minimizes the given regularization functional. This separation will however display poor classification performance if tested against the unlabeled data. If the manifold smoothness penalty is included (by setting a nonzero value to γ_I), it will encourage the approximating function not to vary along regions with high density of data, as induced by the neighborhood measure encoded in the Laplacian matrix. Such is the role of the $(f(\mathbf{x}_i) - f(\mathbf{x}_j))^2 w_{ij}$ penalty term. This phenomenon is presented in the center and rightmost plots. The resulting separation region is molded (according to the minimization of the regularized functional) to the distribution of the data, while maintaining the separation between labeled data examples. The relative strength of these two effects will be controlled by the ratio of the γ_I and γ_A regularization hyper-parameters.

Problem (3.3) can be used for defining computable prediction functions once we verify that the Representer Theorem can be also applied to reduce the problem from infinite dimensional to the n -dimensional determination of $\alpha \in \mathbb{R}^{l+u}$, as done previously.

Theorem 2. *Representer Theorem for Manifold Learning*

The objective function of the manifold learning problem

$$\underset{\mathbf{f} \in \mathcal{H}_k}{\text{minimize}} \quad \sum_{i=1}^l V(y_i, f(\mathbf{x}_i)) + \gamma_A \|\mathbf{f}\|_K^2 + \frac{\gamma_I l}{(u+l)^2} \mathbf{f}_{l+u}^T \mathbf{L} \mathbf{f}_{l+u} \quad (3.4)$$

admits a solution \mathbf{f}^* as the expansion of the $l+u$ examples:

$$f^*(\mathbf{x}) = \sum_{i=1}^{l+u} \alpha_i k(\mathbf{x}, \mathbf{x}_i). \quad (3.5)$$

Proof. The proof is a variation of the orthogonality argument developed for the case of standard regularization learning, Theorem 1, where \mathbf{f}^* is decomposed into an expansion over the training set and its orthogonality complement. Note that the objective function of (3.4) is formed by a sum of three components. For the first two components, the loss function and the norm of the function in ambient space, the demonstration follows exactly like in the proof for Theorem 1. It

turns out that the same reasoning can be extended for the third component, the regularization term over the empirical manifold approximation, as follows:

$$\begin{aligned}
 \frac{\gamma_I l}{(u+l)^2} \mathbf{f}_{l+u}^T \mathbf{L} \mathbf{f}_{l+u} &= \gamma_I' \mathbf{f}_{l+u}^T \mathbf{L} \mathbf{f}_{l+u}, \text{ with } \gamma_I' > 0 \\
 &= \gamma_I' (\sqrt{\mathbf{L}} \mathbf{f}_{l+u})^T \sqrt{\mathbf{L}} \mathbf{f}_{l+u} \\
 &= \gamma_I' \|\sqrt{\mathbf{L}} \mathbf{f}_{l+u}\|^2,
 \end{aligned} \tag{3.6}$$

which is valid because \mathbf{L} is positive semi-definite. It means that the third component is also a norm of \mathbf{f}^* . The working assumption that $f^*(\mathbf{x}) = \sum_{i=1}^{l+u} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + f_0^\perp$ is contradicted by showing that \mathbf{f}_0^\perp is equal to zero when the objective function in (3.4) is minimized.

□

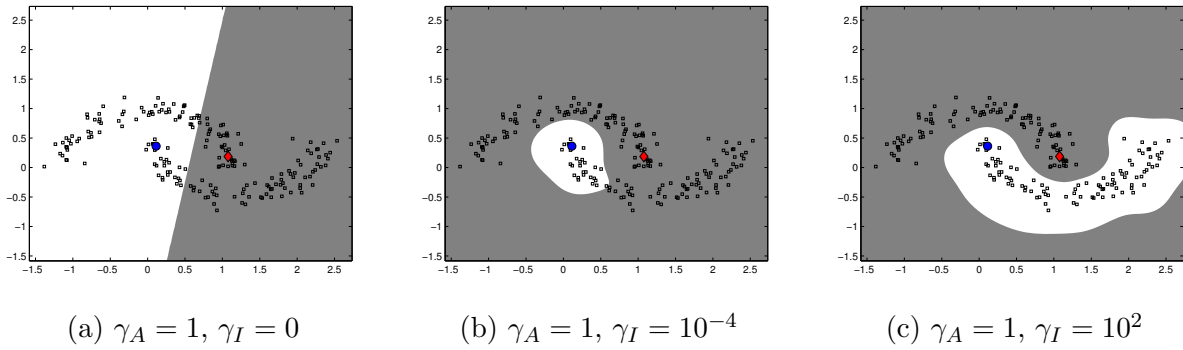


Figure 3.1: Manifold Regularization learning classification function for the Two-moons dataset for different values of γ_A and γ_I using the Laplacian Regular Least Squares Algorithm

3.2.1 The Laplacian and its Hyper-parameters

As stated previously, the estimation of the manifold can be achieved through the calculation of the Laplacian matrix, which is calculated from an adjacency matrix representation of the data according to some neighborhood relationship criterion. We first look at two types of such criterion.

The ϵ -NN Adjacency Matrix

The ϵ -NN Adjacency Matrix is created by considering that any two given points are neighbors if their Euclidean distance in the ambient space is less than or equal to a given prescribed distance ϵ . This adjacency matrix is therefore an $(l+u) \times (l+u)$ matrix defined as

$$w_{ij} = \begin{cases} \omega_{ij}, & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| \leq \epsilon; \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

A common drawback of this definition is that it cannot assure that the resulting graph is connected, depending on the value of parameter ϵ and the distance of a given point to its closest neighbors (since the distance of a given point to its closest neighbor might be more than ϵ). This disadvantage is not present in the next adjacency matrix construction method, k -NN.

The k -NN Adjacency Matrix

The k -NN Adjacency Matrix determines, for each of its elements, what are the k other points that have smaller Euclidian distance to it, defining its adjacency assignment. For disambiguation of notation, letting the number of nearest neighbors be represented by the parameter ν , the k -NN adjacency matrix $\mathbf{W} = w_{ij}$ can be defined as

$$w_{ij} = \begin{cases} \omega_{ij}, & \text{if } \|\mathbf{x}_i - \mathbf{x}_{iv}^*\| \leq \|\mathbf{x}_i - \mathbf{x}_j\|, \quad v = 1, \dots, \nu; \quad i, j = 1, \dots, n; \\ 0, & \text{otherwise,} \end{cases} \quad (3.8)$$

where \mathbf{x}_{iv}^* are the ν nearest neighbors for each point \mathbf{x}_i , and ω_{ij} is the adjacency weight, next defined. It is worthwhile to emphasize here that the ν is an integer parameter.

Adjacency Weight ω_{ij}

The adjacency relation assignment can be either a binary one, in which case $\omega_{ij} = 1$, or it can be weighted according to some measure of the strength of the relationship. Typically, the non-binary adjacency weight can be calculated as an exponential radial decay, such as

$$\omega_{ij} = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\tau}}. \quad (3.9)$$

where τ is a parameter for the decay sensitivity, sometimes referred to as “spread”. In both cases we have that $\omega_{ij} \geq 0$, granting the positive semi-definite assumption stated previously.

The Laplacian Matrix

The Laplacian matrix carries essentially the same information of the adjacency matrix. The difference lies in its diagonal entries. The Laplacian matrix is defined as:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}, \quad (3.10)$$

where \mathbf{D} is a diagonal matrix with:

$$d_{ii} = \sum_{j=1}^{l+u} w_{ij}, \quad i = 1, \dots, n, \quad j = 1, \dots, n. \quad (3.11)$$

The Laplacian matrix can be normalized for some theoretical guarantees pointed by (Belkin et al. 2006), in which case it assumes the form

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2}. \quad (3.12)$$

Finally, one has the possibility of working with an integer power of the Laplacian matrix, which replaces the notion of adjacency with that of the existence of p -closed paths (iterations) between any two points as graph vertices. This can be understood as an alternative notion of neighborhood derived from the adjacency matrix. This fact will be used complementarily in defining composite Laplacian matrices in the next chapter. The iterated graph Laplacian matrix \mathbf{M} can be defined as

$$\mathbf{M} = \frac{\gamma_I}{\gamma_A} \mathbf{L}^p, \quad p \in \mathbb{N}_+. \quad (3.13)$$

It is known that \mathbf{L}^p , a product of \mathbf{L} positive semi-definite matrices is also positive semi-definite. This can be shown by iteratively applying the standard eigenvalue decomposition expression for \mathbf{L}^p . For the case $p = 2$, for example, we have $\mathbf{L} \mathbf{v} = \lambda \mathbf{v}$, where (λ, \mathbf{v}) is an eigenvalue-eigenvector pair of \mathbf{L} . Hence,

$$\mathbf{L}^2 \mathbf{v} = \mathbf{L} \mathbf{L} \mathbf{v} = \lambda \mathbf{L} \mathbf{v} = \lambda \lambda \mathbf{v} = \lambda^2 \mathbf{v} \quad (3.14)$$

It is readily seen that, because $\mathbf{L} \geq 0$, all its eigenvalues are non-negative, and therefore all the eigenvalues of \mathbf{L}^p will be also non-negative.

The use of Manifold Regularization models, including their iterated and weighted Laplacians matrices, as well as the manifold penalty multiplier, introduce additional hyperparameters θ_{MR} to the model selection problem, $\theta_{MR} = (\gamma_I, \nu, p, \tau)$.

3.2.2 Point Cloud Kernel

As an alternative and powerful formulation to the Manifold Regularization problem, (Sindhwani, Niyogi & Belkin 2005) have worked on the formulation (3.3) to allow the manifold penalty term to be defined for standard Regularization and Kernel Learning algorithms, thereby enabling existing algorithms to be extended to the manifold semi-supervised setting. The main idea is that the Laplacian and its associated manifold smoothness penalty term can be incorporated into a single kernel function definition. It is proven that such new composite kernel is indeed a valid RKHS kernel. The so called Point Cloud Kernel, is based on a new

dot product definition that relates any two \mathcal{H}_K elements with their plain dot product, but to which an addition product scaled by the Laplacian matrix is summed, weighted by the penalty parameters. The Point Cloud dot product is written as

$$\langle \mathbf{f}, \mathbf{g} \rangle_{\tilde{\mathcal{H}}_K} = \langle \mathbf{f}, \mathbf{g} \rangle_{\mathcal{H}} + \frac{\gamma_I}{\gamma_A} \mathbf{f}^T \mathbf{L} \mathbf{g}. \quad (3.15)$$

The proof that this product induces a norm that defines a valid RKHS can be found in (Sindhwani et al. 2005). The application of such norm definition defines the Point Cloud Kernel, which, for each point, is given by:

$$\tilde{k}(\mathbf{x}, \mathbf{z}) = k(\mathbf{x}, \mathbf{z}) - \mathbf{k}_x (\mathbf{I} + \mathbf{M} \mathbf{K})^{-1} \mathbf{M} \mathbf{k}_z. \quad (3.16)$$

where \mathbf{k}_x denotes the $l + u$ vector of kernel function evaluations between the current point \mathbf{x} and all the examples. The equivalent definition applies to \mathbf{k}_z and \mathbf{z} .

It is worth to note that such kernel is compatible with any of the RKHS-based regularization learning schemes (such as RLS and SVM), a fact that enforces the notion of modularity enabled by this framework. When looking at the Gram matrix of all training data, the Point Cloud Kernel matrix, together with its component matrix dimensions, is represented as:

$$\begin{aligned} \underbrace{\tilde{\mathbf{K}}_{train}}_{l \times l} &= \underbrace{\mathbf{K}}_{l \times l} - \underbrace{\mathbf{K}_x}_{l \times (l+u)} \underbrace{(\mathbf{I} + \mathbf{M} \mathbf{K})^{-1} \mathbf{M}}_{(l+u) \times (l+u)} \underbrace{\mathbf{K}_z}_{(l+u) \times l} \\ &= \mathbf{K} - \mathbf{K}_x \mathbf{G} \end{aligned} \quad (3.17)$$

$$\text{where } \mathbf{G} = (\mathbf{I} + \mathbf{M} \mathbf{K})^{-1} \mathbf{M} \mathbf{K}_z.$$

An interesting interpretation of the Point Cloud Kernel is that it can be thought of as a kernel over the labeled data deformed by the unlabeled data, so as to alter (or correct) the notion of similarity between two vectors in ambient space with information from the intrinsic space. Figure 3.2 depicts the level sets of the Point Cloud Kernel for the similarity between itself and all potential surrounding points in the two-circles dataset, where the influence of the unlabeled data can be clearly perceived. The plot displays the level sets for the kernel similarity function between the labeled point in the outer circle, represented by a diamond marker, and every other point in the input space. One can see that although there are points in the inner circle which

are geometrically closer (in ambient space) to the labeled point, the Point Cloud kernel function assigns them a lower similarity measure than to those points along the outer circle. That is, points along the outer circle are evaluated to be relatively closer to the labeled point in intrinsic space than point in the inner circle. The intensity of such deformation effect is controlled by the ratio of the intrinsic and ambient regularization hyper-parameters.

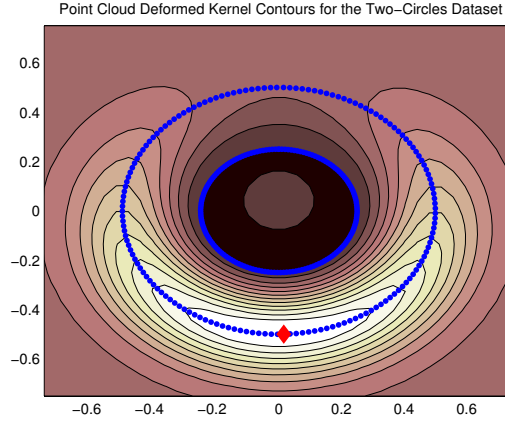


Figure 3.2: Data-deformed kernel for a manifold derived from the Two-circles dataset. The two concentric circles correspond to the unlabeled data distribution of two different classes. One class is the inner circle, having one labeled data point as the solid circle. The other class is the outer circle, having the painted diamond as its labeled example. The contour level sets show how the similarity value is influenced by the unlabeled data distribution.

We now derive the Manifold Regularization counterparts for the two algorithms defined in Chapter 1 as RLS and SVM.

3.3 Laplacian Regularized Least Squares

As done for the case of Regularized Least Squares, the Laplacian Regularized Least Squares (LapRLS) is defined with the use of the square loss function over the labeled data. By applying the Representer Theorem and knowing that the solution is of the form $f^*(\mathbf{x}) = \sum_{i=1}^{l+u} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$, problem (3.3) can be written as

$$\alpha^* = \underset{\alpha \in \mathbb{R}^{l+u}}{\operatorname{argmin}} \frac{1}{l} (\mathbf{y} - \mathbf{J}\mathbf{K}\alpha)^T (\mathbf{y} - \mathbf{J}\mathbf{K}\alpha) + \gamma_A \alpha^T \mathbf{K}\alpha + \frac{\gamma_I l}{(u+l)^2} \alpha^T \mathbf{K}\mathbf{L}\mathbf{K}\alpha, \quad (3.18)$$

where \mathbf{y} is an $(l+u)$ -dimensional vector with $\mathbf{y} = [y_1, \dots, y_l, 0, \dots, 0]$; \mathbf{J} is an $(l+u) \times (l+u)$ matrix $\mathbf{J} = \operatorname{diag}(1, \dots, 1, 0, \dots, 0)$ with first l entries 1 and remaining entries zero, and \mathbf{K} is the $(l+u) \times (l+u)$ Gram matrix over labeled and unlabeled data. Proceeding in similar fashion as done for the RLS case, by equalling the gradient of the objective function of (3.18) to zero,

yields

$$\alpha^* = (\mathbf{JK} + \gamma_A \mathbf{I} + \frac{\gamma_I l}{(u+l)^2} \mathbf{LK})^{-1} \mathbf{y} = \mathbf{C}^{-1} \mathbf{y}, \quad (3.19)$$

which, similarly to the RLS case, is computable by direct inversion of the composite matrix, now of order of $(l+u)$.

3.4 Laplacian Support Vector Machines

The application of the hinge loss function leads to what (Belkin et al. 2006) refers to as the Laplacian Support Vector Machine (LapSVM). Its derivation is quite similar in its main steps to the case of the SVM, and follows his exposition. For details and intermediate steps, please refer to (Belkin et al. 2006).

After applying the hinge loss function and expressing the solution in the form prescribed by the Representer Theorem, the optimization problem is written as:

$$\alpha^* = \arg \min \frac{1}{l} \sum_{i=1}^l |1 - y_i f(\mathbf{x}_i)|_+ + \gamma_A \|\mathbf{f}\|_K^2 + \frac{\gamma_I l}{(u+l)^2} \mathbf{f}_{l+u}^T \mathbf{L} \mathbf{f}_{l+u}. \quad (3.20)$$

Each term of the hinge loss function can be replaced by an equivalent slack variable ξ_i and two inequality constraints. The reformulation, in its primal form, becomes:

$$\begin{aligned} & \underset{\alpha \in \mathbb{R}^{l+u}, \xi \in \mathbb{R}^l}{\text{minimize}} && \frac{1}{l} \sum_{i=1}^l \xi_i + \gamma_A \alpha^T \mathbf{K} \alpha + \frac{\gamma_I l}{(u+l)^2} \alpha^T \mathbf{K} \mathbf{L} \mathbf{K} \alpha \\ & \text{subject to} && y_i \alpha^T \mathbf{k}_i \geq 1 - \xi_i \\ & && \xi_i \geq 0 \\ & && i = 1, \dots, l. \end{aligned} \quad (3.21)$$

By the same duality arguments used in the SVM case, (3.21) is a convex problem which can be cast in its dual form, with zero duality gap, in order to obtain simpler constraints. Obtaining the dual problem requires the introduction of the Lagrange multipliers β and a Lagrangian function to be minimized, which leads to the following relation between primal and dual variables:

$$\alpha^* = \left(2\gamma_A \mathbf{I} + 2\frac{\gamma_I}{(l+u)^2} \mathbf{L} \mathbf{K} \right)^{-1} \mathbf{J}^T \mathbf{Y} \beta^*. \quad (3.22)$$

The dual problem can then be written in terms of the dual variables β as

$$\begin{aligned}
& \underset{\beta \in \mathbb{R}^l}{\text{maximize}} && \mathbf{1}^T \beta + \beta^T Q \beta \\
& \text{subject to} && \sum_{i=1}^l y_i \beta_i = 0 \\
& && 0 \leq \beta_i \leq \frac{1}{l}
\end{aligned} \tag{3.23}$$

where:

$$\begin{aligned}
\mathbf{Q} &= \mathbf{Y} \mathbf{J} \mathbf{K} \left(2\gamma_A \mathbf{I} + 2 \frac{\gamma_I}{(l+u)^2} \mathbf{L} \mathbf{K} \right)^{-1} \mathbf{J}^T \mathbf{Y}, \\
\mathbf{Y} &= \text{diag}(y_1, \dots, y_l), \\
\mathbf{J} &= [\mathbf{I} \ \mathbf{0}] \in \mathbb{R}^{l \times (l+u)}.
\end{aligned} \tag{3.24}$$

Note that after finding the optimal solution β^* of (3.23) for the dual problem, an additional linear system needs to be solved in order to obtain the values for α^* and the value of the target function.

3.5 Computational Considerations

There are two key differences when comparing the RLS and SVM computational characteristics to its manifold extensions. The first is that the manifold algorithms require the Laplacian matrix to be calculated, which for the k -NN variant is of order $O((l+u)d)$. If one compares this to the baseline complexities for the RLS and SVM (and their LapRLS and LapSVM), this is not a dominant factor. The second difference is that the manifold algorithms operate with $l+u$ data points, with typically $u \gg l$. It means that the complexities for LapRLS and LapSVM are now of the order of $O[(l+u)^3]$ in time, and $O[(l+u)^2]$ in space. The same strategies that have been applied for reducing the training time in SVM can be used, such as looking for controlled approximate solutions in the primal, as recently proposed in (Melacci & Belkin 2011). Additionally, other initiatives propose a reduction in the effective dimension of the Laplacian and associated kernel matrices through sampling and decomposition, such as the one presented in (Talwalkar, Kumar & Rowley 2008, Talwalkar 2010). A third aspect is caused by the additional hyper-parameters required by the manifold models, and their influence in the final performance achieved by the predicting model. This effect will be addressed in the next chapter.

Hyper-parameter Optimization for Manifold Regularization Learning Models

In this chapter we use concepts and results of Chapters 2 and 3 to the higher dimensional problem of optimizing hyper-parameters for Manifold Regularization learning models. The author proposes a new formulation for the manifold hyper-parameter optimization problem, which enables the use of continuous optimization algorithm. Computational experiments are performed, which illustrate the main ideas presented throughout this text, including a comparison of different hyper-parameter optimization algorithms.

4.1 Hyper-parameter Optimization Formulation

We have seen that the PRESS validation function, when combined with the leave-one-out cross validation partition strategy, and in the case where the kernel weights are expressed as a linear system of the class label vector $\mathbf{C}\alpha = \mathbf{y}$, can be expressed in closed form as

$$\text{PRESS}(\theta) = J(\theta) = \frac{1}{2} \sum_{i=1}^l r_i^2 \quad \text{where} \quad r_i = y_i - f_S^{(-i)}(\mathbf{x}_i) = \frac{y_i - f_S(\mathbf{x}_i)}{1 - C_{ii}^{-1}} \quad (4.1)$$

and $f_S(\mathbf{x}_i)$ being the prediction function trained with the complete training set:

$$f_S(\mathbf{x}_i) = \sum_{i=1}^{l+u} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = \alpha^T \mathbf{k}_i \quad \text{and} \quad \alpha = \mathbf{C}^{-1} \mathbf{y} \quad (4.2)$$

The \mathbf{C} matrix has been defined in 3.19 for the case of Laplacian Least Squares. Note that although the training of the semi-supervised learning function uses $l+u$ examples, the validation value is computed over the errors incurred over the l labeled examples only. If we specialize the

general optimization formulation presented in (2.11), we arrive at the following formulation of the hyper-parameter optimization problem:

$$\begin{aligned}
& \underset{\theta}{\text{minimize}} \quad \frac{1}{2} \sum_{i=1}^l r_i^2 \\
& \text{subject to} \\
& \quad r_i = \frac{y_i - \alpha^T \mathbf{k}_i}{1 - C_{ii}^{-1}}, \\
& \quad \alpha = \mathbf{C}^{-1} \mathbf{y}, \\
& \quad \mathbf{C} = \mathbf{J}\mathbf{K}(\theta_{\mathbf{k}}) + \gamma'_A \mathbf{I} + \gamma'_I \mathbf{L}^p(\nu) \mathbf{K}(\theta_k), \\
& \quad \theta \in \Theta;
\end{aligned} \tag{4.3}$$

with α and the \mathbf{C} matrix being a function of the vector of hyper-parameters $\theta = (\gamma'_A, \theta_k, \gamma'_I, \nu, p)$ with $\gamma'_I = \frac{l}{(u+l)^2} \gamma_I$ and $\gamma'_A = l \gamma_A$. Recall that:

- γ_A is the ambient space regularizer (same as λ in RLSC), controlling the amount of fitting allowed in the approximation.
- θ_k is a vector of kernel hyper-parameters. For the case of the Gaussian kernel, we have $\theta_k = (\sigma)$, the exponential width.
- γ_I is the intrinsic space regularizer, controlling the amount of influence of unlabeled data.
- ν is the number of nearest neighbors considered in the adjacency matrix for the Laplacian matrix.
- p is the iteration power of the Laplacian matrix.

For the last two hyper-parameters, ν and p , we note that they are integer quantities. In order to apply continuous optimization methods, one could relax this constraint and define their derivatives in order to determine a direction of descent, and later re-apply the integrality constraint at each. A similar strategy has been applied in a recent article by (Yuan, Liu & Liu 2012), which used a method without derivatives, the Nelder-Mead optimization algorithm (Nelder & Mead 1965). In this dissertation, however, we have chosen to preserve the integrality of these parameters, adopting an alternative approach, described next.

4.2 Combination of Laplacians

As discussed in Chapter 2, one can define a linear combination of kernel functions which result in a valid composite kernel. There have been many studies exploring variations of this approach, surveyed in (Gönen & Alpaydm 2011). Such approach has also been recently extended to combination of Laplacian matrices in the work of (Geng, Xu, Tao, Yang & Hua 2009). According to this study, the ensemble Laplacian matrix \mathbf{E} is defined as a convex combination of base Laplacian matrices, written as

$$\mathbf{E} = \sum_{j=1}^q \mu_j \mathbf{L}_j, \quad \sum_{j=1}^q \mu_j = 1, \quad \mu_j \geq 0. \quad (4.4)$$

We adopt this approach in this dissertation, and develop it further so as to make use of its differentiability characteristic for hyper-parameter optimization. The ensemble Laplacian can be defined to span, for example, a two-dimensional combination of the nearest neighbor number ν and the iteration coefficient p , leading to a continuous simplex hyper-parameter $\mu \in \mathbb{R}^{\nu \times p}$. As analyzed in (Geng et al. 2009), because each \mathbf{L}_j is a positive semi-definite matrix, \mathbf{E} is a positive semi-definite matrix. In this case, the LapRLS formulation can be rephrased as:

$$\alpha^* = \underset{\alpha \in \mathbb{R}^{l+u}}{\operatorname{argmin}} \frac{1}{l} (\mathbf{y} - \mathbf{J}\mathbf{K}\alpha)^T (\mathbf{y} - \mathbf{J}\mathbf{K}\alpha) + \gamma_A \alpha^T \mathbf{K}\alpha + \frac{\gamma_I l}{(u+l)^2} \alpha^T \mathbf{K} \mathbf{E} \mathbf{K} \alpha, \quad (4.5)$$

leading to the equivalent definition of the \mathbf{C}_{LC} matrix. After differentiating the objective function with respect to α , equaling the result to zero, and rearranging, we obtain

$$\mathbf{C}_{LC} \alpha^* = \mathbf{y}, \quad \text{where} \quad \mathbf{C}_{LC} = \mathbf{J}\mathbf{K} + \gamma_A l \mathbf{I} + \frac{\gamma_I l}{(u+l)^2} \mathbf{E} \mathbf{K}. \quad (4.6)$$

We can then update our hyper-parameter optimization formulation and write:

$$\begin{aligned}
& \underset{\theta}{\text{minimize}} \quad \frac{1}{2} \sum_{i=1}^n r_i^2 \\
& \text{subject to} \\
& \quad r_i = \frac{y_i - \alpha^T \mathbf{k}_i}{1 - C_{ii}^{-1}}, \\
& \quad \alpha = \mathbf{C}_{LC}^{-1} \mathbf{y} \\
& \quad \mathbf{C}_{LC} = (\mathbf{J}\mathbf{K} + \gamma'_A \mathbf{I} + \gamma'_I \mathbf{E} \mathbf{K}) \\
& \quad \sum_{j=1}^q \mu_j = 1, \mu_j \geq 0 \\
& \quad \theta \in \Theta
\end{aligned} \tag{4.7}$$

where we have included the hyper-parameters μ and the vector of real hyper-parameters $\theta_R = (\gamma_A, \theta_k, \gamma_I, \mu)$ and fixed the integer parameters in $\theta_I = (\nu, p)$. The overall training and validation procedure, as directed by this optimization formulation, is summarized in Algorithm 1.

Algorithm 1 Gradient-based Hyper-parameter Optimization for Manifold Regularization with Combination of Laplacians

Input :

- l labeled training samples (\mathbf{x}_i, y_i) and u unlabeled training points (\mathbf{x}_i)
- default and boxed limit values for hyper-parameters $\theta = (\gamma_A, \theta_k, \gamma_I, \mu)$
- base integer values for Laplacian parameters $\{\nu_1, \dots, \nu_{qn}\} \times \{p_1, \dots, p_{qp}\}$

Procedure :

- 1: compute the base set of Laplacian matrices \mathbf{L}_j for the range of ν, p
- 2: do until $\|\nabla J(\theta_n)\| \leq \epsilon_{\text{tol}}$:
- 3: compute \mathbf{C} and its inverse
- 4: compute the gradient $\nabla J(\theta)$
- 5: $\theta_{n+1} = \theta_n + a_n \mathbf{d}_n$ with a_n the step size and \mathbf{d}_n a descent direction dependent on $\nabla J(\theta)$

Output :

- optimal vector of hyperparameters θ^*
-

4.2.1 Computational Considerations

Leave-one-out Cross Validation, in its naive implementation, requires n training rounds of $n - 1$ examples, where $n = l$ for the supervised case, and $n = l + u$ for the semi-supervised case. With the closed form calculation for LOOCV here presented, only 1 training round is required. Each training round is of cost $O(n^3)$ for the RLS and LapRLS cases, and between n^3 and down to sublinear for SVM and LapSVM cases, depending on the specific SVM solver implementation and desired accuracy, as described in section 1.5. The hyper-parameter gradient calculation

requires multiplications between the training matrices for each hyper-parameter dimension, and is therefore of cost $O(hn^3)$, where h is the number of hyper-parameters. Quasi-newton algorithms are also typically configured to stop in a number of iterations which is roughly proportional to the number of dimensions of the search space.

4.3 Computational Experiments

The experiments performed illustrate the main concepts addressed in this dissertation. Firstly, an application of hyper-parameter optimization using the closed-form PRESS validation function is presented. A second experiment shows the influence of the use of unlabeled data in improving prediction accuracy for Laplacian Regularized Least Squares and its Point Cloud Kernel equivalent. Lastly, the Manifold Regularization hyper-parameter optimization problem is addressed, using different optimization methods. While this last experiment does not intend to be in any way exhaustive or generalizable, it hopes to raise questions and interest for future work.

4.3.1 Experimental Setup

The execution of experiments requires the consideration of different aspects involving the selection of datasets, training algorithms, optimization solvers and reporting procedures. They are briefly described next.

Datasets

The two datasets selected have also been used in other semi-supervised learning experiments (Belkin et al. 2006, Sindhwani et al. 2005, Melacci & Belkin 2011). These datasets are described below:

- *COIL20*: according to (Nene, Nayar & Murase 1996), the “Columbia Object Image Library (COIL-20) is a database of gray-scale images of 20 objects. The objects were placed on a motorized turntable against a black background. The turntable was rotated through 360 degrees to vary object pose with respect to a fixed camera. Images of the objects were taken at pose intervals of 5 degrees. This corresponds to 72 images per object”. The 20 classes have been grouped in two target classes to enable the use of binary classification, thereby isolating the influence of multiclass strategies. The elements of COIL20 are 1024 dimensional.

- *USPST*: is a collection of handwritten digits from the United States Postal Service. It consists of images of 0 to 9 digits with a resolution of 16 to 16 pixels. This collection has been adopted in many machine learning studies, and is available in (Bache & Lichman 2013). Target classes have been grouped in two classes of ten objects to allow for binary classification. The elements in USPST are 256 dimensional.

Test Execution Environment

To run the experiments with different configurations and experiment scenarios, a modular test harness has been developed for this study, using Matlab (MATLAB 2009). The set-up allows the use of different learning algorithms, optimizers, kernel functions and validation strategies. The main modules of the set-up are:

- *Trainer*: Implements different supervised and semi-supervised learning algorithms, with the following being available: RLS, LapRLS, C-LapRLS, and Least Squares SVM. Other SVM training variants could be integrated if needed;
- *Optimizer*: Integration of the Trainer with different solution algorithms for hyper-parameter optimization, with the “Grid Search”, “Random Search”, “Gradient (Quasi Newton)”, “Nelder-Mead” and “Literature” implementations. The “Literature” optimizer is a trivial solver with returns optimal values provided from external sources;
- *Validation Strategies*: The implementation of the actual objective function to be optimized, available as the LOOCV closed form, LOOCV iterative (where each fold is explicitly calculated), and k -Fold iterative;
- *Partition*: Provides the indexing of different random partitions (labeled, unlabeled, test) on the datasets, enabling partition selections to be saved between executions, in order to allow for reproducibility between experiment runs;
- *Kernel*: Allows the use of different kernel functions, including the Point Cloud Kernel, which uses unlabeled data and references a second, inner, kernel function.
- *Datasets and Parameters*: allows the mapping of different datasets and hyper-parameter values to be used in the experiments.

Apart from these modules, a centrally configured main execution loop to collect the data generated for the output generation module (boxplots, contours, scatter figures) was also implemented by the author. For the learning algorithms, the main solution step is performed

with the linear system solver for matrix inversion available in Matlab. For the optimization solvers, the gradient Quasi-Newton solver is the one implemented in the *fmincon* procedure of the Matlab Optimization Toolbox, which is a Sequential Quadratic Programming (SQP) algorithm (Bazaraa & Shetty 1979). For Nelder-Mead optimization, an external code implemented by (Cawley 2006) was used. Experiments were executed on Matlab 7.8.0 Student Version on an Intel i5 Quad-core 2.5 GHz 6 GB memory machine, running Ubuntu Linux 64-bit Kernel version 3.5.0-23.

4.3.2 Gradient Optimization of Regularized Least Squares

The objectives of the computation experiments in this section are two-fold. The first goal is to observe the applicability of the PRESS validation function. We then look at the performance of gradient search in finding optimal points in the space of that function, as the second goal.

An RLS classifier with a Gaussian (RBF) kernel has been applied to both datasets, with two hyper-parameters subject to variation and optimization: the Gaussian radial parameter (σ) and the regularization parameter (λ). The Gaussian kernel is a typical decision in the absence of specific information.

The 2-dimensional vector of hyper-parameters was subject to the gradient optimization of the PRESS objective function, and its optimization trajectory was recorded. For each point in the trajectory, its corresponding PRESS function value was stored, as well as its test performance evaluated with an independent hold-out set, computed over t examples. Figures 4.1 and 4.2 present the optimization trajectories plotted against both the validation function and the independent test set contours for the hyper-parameter space. Hyper-parameters are presented on a \log_2 base, the vertical axis corresponding to the kernel parameter, and the horizontal to the regularization parameter. One can observe that regions of lower (darker) validation function values are in strong correspondence with regions of lower test set error performance, for both datasets. This behavior was observed for many splits and different numbers of labeled examples used as training conditions. It can also be seen that the optimization trajectory is relatively well guided and arrives at regions of minima with under 40 function evaluations, in both cases. In comparison, if this number were assigned to a uniform grid in two dimensions, a resolution of 6 points per parameter dimension would be attained. As seen in Section 4.2, in higher dimensional parameter spaces, coarser representations of hyper-parameter space would be obtained.

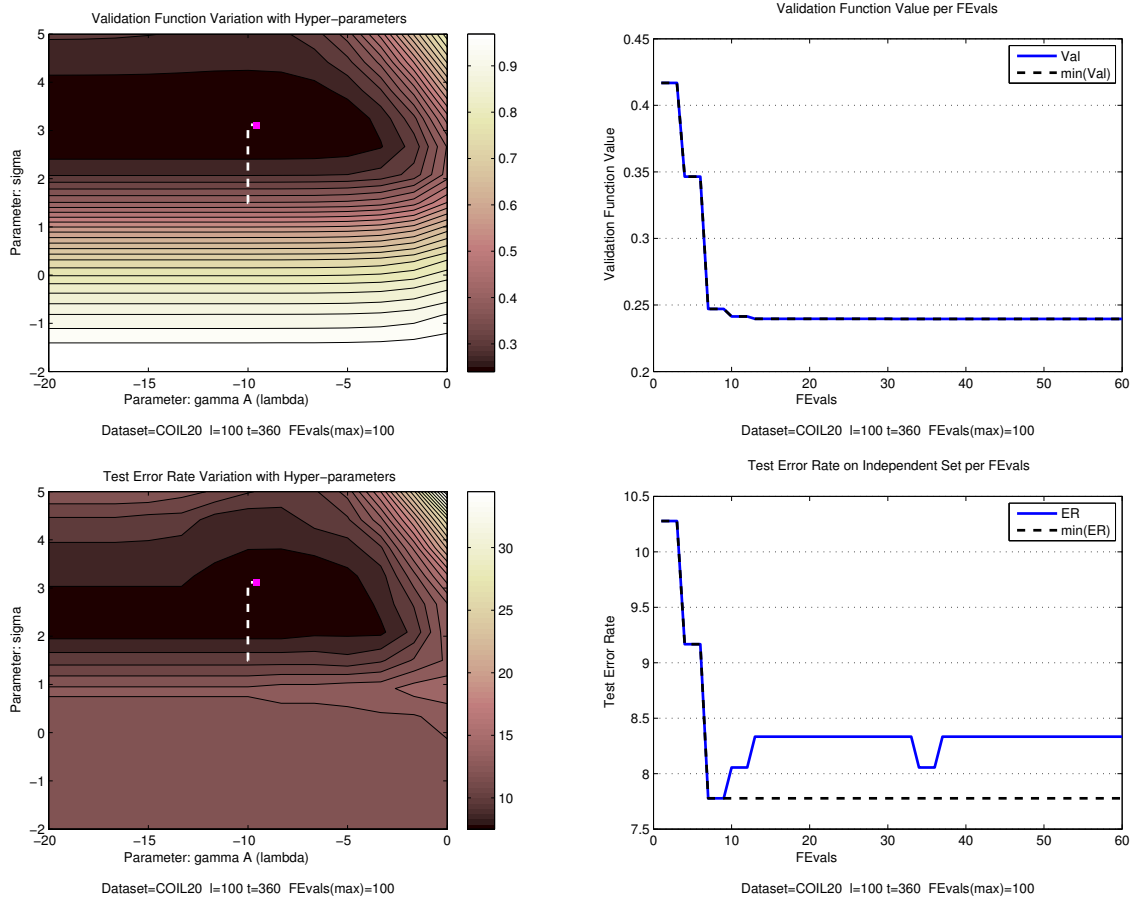


Figure 4.1: Hyper-parameter trajectory and level sets (left column), and decay over validation function evaluations (right) for the PRESS validation function (top row) and test error rate (bottom) for the COIL20 dataset trained with 100 labeled samples.

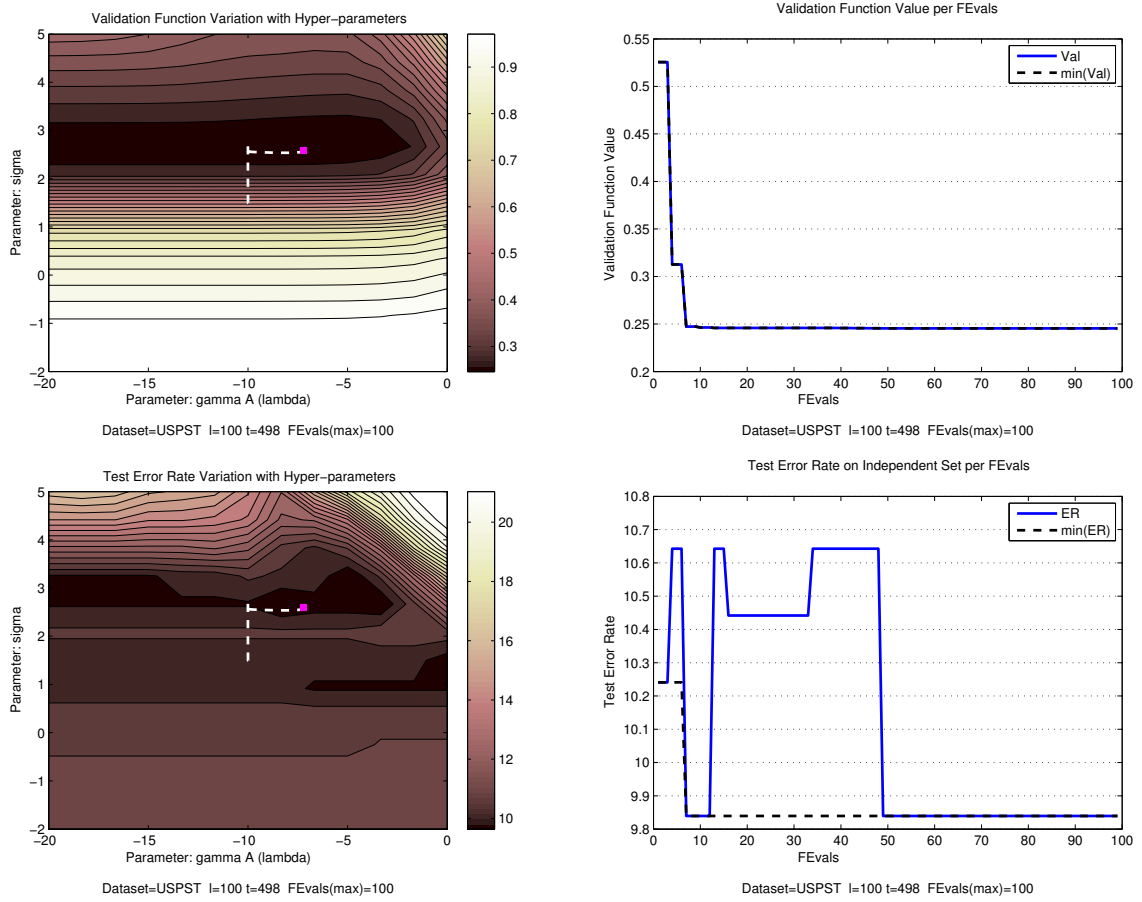


Figure 4.2: Hyper-parameter trajectory and level sets (left column), and decay over function evaluations (right) for the PRESS validation function (top row) and test error rate (bottom) for the USPST dataset trained with 100 labeled samples.

4.3.3 Influence of Unlabeled Data

The purpose of this experiment is to observe the accuracy improvement provided by the availability of unlabeled data, as processed by Manifold Regularization. The training algorithm employed was the Laplacian Regularized Least Squares. As pointed out by (Belkin et al. 2006) in his experiments, no significant gain was observed from LapSVM over LapRLS in the case of manifold learning, and the LapRLS version has therefore been adopted for simplicity of implementation.

In the experiments reported below, the amount of labeled data defined as training points to the algorithm was restricted to smaller sets, and the remaining samples were used as unlabeled training points. The amount of unlabeled training examples was, in turn, defined as being zero, being the full remainder of the dataset, and being an intermediate amount between the later two.

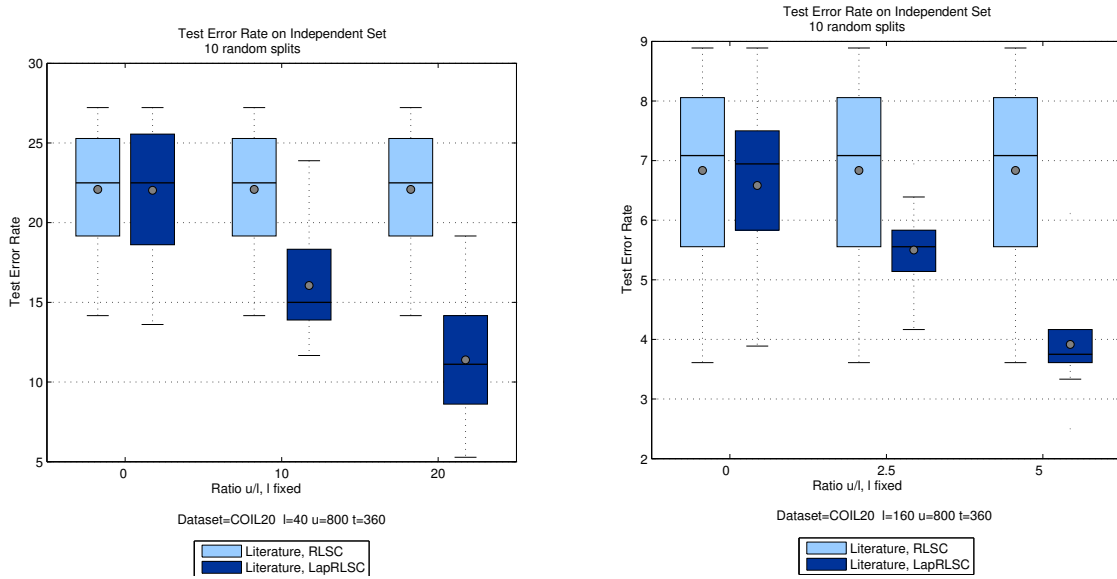


Figure 4.3: COIL20 dataset boxplots for test set error performance comparing supervised RLS (light color) to semi-supervised LapRLS (darker color) for increasing availability of unlabeled samples (their ratio being represented on the horizontal scale). The first plot (left) uses 40 labeled samples. The second plot uses 160 labeled samples.

The results are presented in the form of boxplot graphs (see explanation in separate frame) in Figure 4.3. The vertical distribution is related to the variation of performance as a result of 10 different data splits being used as the labeled training points. The 5-dimensional hyper-parameter vector $\theta = \{\gamma_A, \sigma, \gamma_I, \nu, p\}$ was defined with the values equal to those presented in (Melacci & Belkin 2011).

Examining the results, one can see that the both the average classification error and its limits are consistently improved as the algorithm is run with increasing numbers of unlabeled data points. This was observed not only in the situation where the amount of labeled samples was scarce (40 and 50 points), but also when the number of labeled points was comparatively high, in regions of better overall performance, for both datasets.

In a **boxplot**, each collection of data is represented by a vertical column, whose shaded portion corresponds to the second and third quartile of the data. Each quartile corresponds to an ordered interval of the variable which spans 25% of the collection. The shaded area therefore corresponds to the central location of 50% of the data. The horizontal line in the center is the median location, and the circle is the mean value. The thinner dotted line has as its limits (whiskers) the points corresponding to a distance 1.5 of the interquartile range (IQR) added to the upper and lower quartiles, IQR being the interval spanned by the second and third quartiles. The whisker location might however be defined differently, depending on author preferences. Occasional outliers not belonging to this interval are plotted independently as isolated data points.

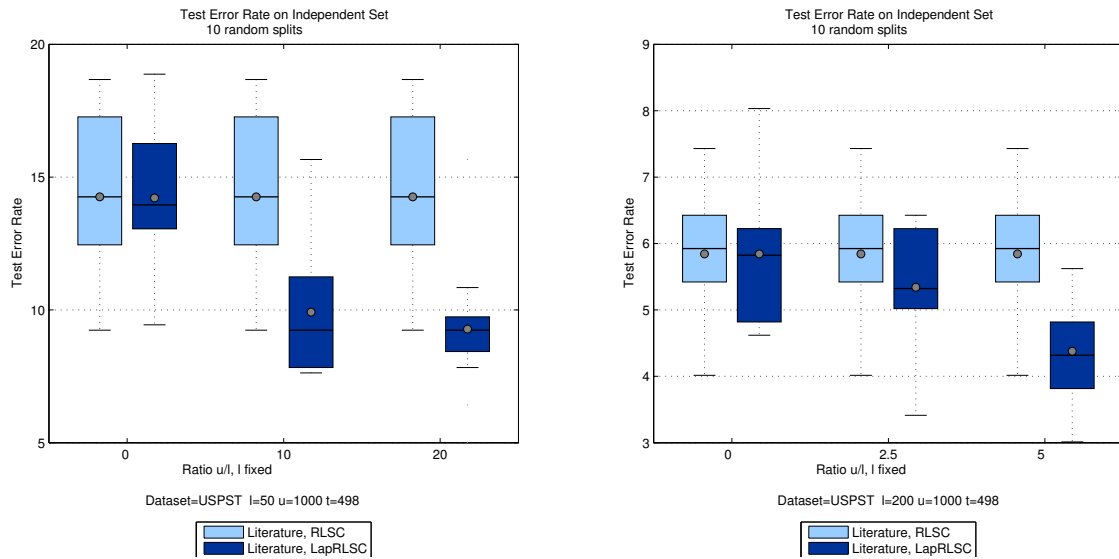


Figure 4.4: USPST dataset boxplots for test set error performance comparing supervised RLS (light color) to semi-supervised LapRLS (darker color) for increasing availability of unlabeled samples (their ratio being represented on the horizontal scale). The first plot (left) uses 50 labeled samples. The second plot uses 200 labeled samples.

Point Cloud Kernel Equivalence

The Point Cloud Kernel formulation is intended to enable the use of standard supervised learning algorithms, such as Regularized Least Squares and Support Vector Machines, for semi-supervised learning tasks. For that purpose, it defines a data dependent kernel, which, as presented in Section 3.2.2, depends on the Laplacian matrix, but whose training kernel matrix and the linear system matrix \mathbf{C} turn out to be of the order of the number of labeled examples. The prediction function that uses the Point Cloud Kernel is also a weighted combination of Point Cloud Kernel function evaluations, but spanning the labeled examples only. To illustrate the equivalence of the prediction functions generated by Point Cloud Kernels that employs a

supervised algorithm (RLS) and its Manifold Regularization counterpart (LapRLS), a simple experiment aimed at comparing test rate results for different numbers of unlabeled examples, for our two working datasets, was run.

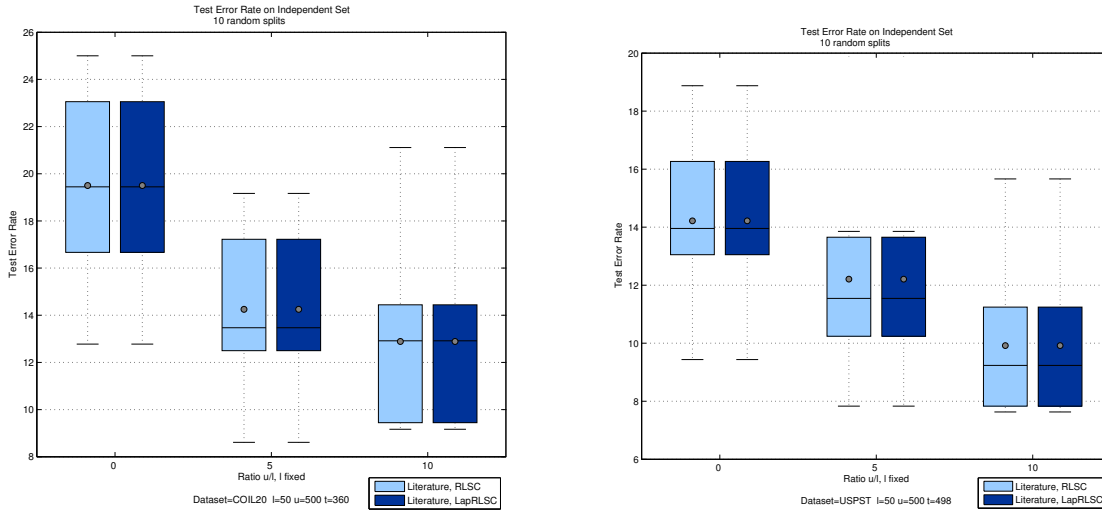


Figure 4.5: Comparison of test error performance for growing ratios of unlabeled data for manifold regularization models trained with Laplacian Regularized Least Squares and Regularized Least Squares with Point cloud kernel for the COIL20 and USPST datasets.

Although having been produced by a standard supervised training solver, it can be seen that the Point Cloud Kernel prediction output is equivalent to the one generated by a Laplacian Regularized Least Square algorithm, as expected.

4.3.4 Laplacian RLS Hyper-parameter Optimization with Limited Function Evaluations

The final set of experiments provides a comparison of different search strategies for the optimization with respect to the set $\theta = \{\gamma_A, \sigma, \gamma_I, \nu, p\}$ of hyper-parameters when applied to Laplacian Regularized Least Squares learning algorithms. The criterion for comparison is the resulting Test Error Rate on the independent dataset when the hyper-parameters are optimized under a limitation on the number of function evaluations (**feval**). One **feval** corresponds to one full validation cycle, which in turn involves the training and evaluation of the model on the training data, given the cycle data partition and the current hyper-parameter vector. In the case of iterative K-Fold cross validation, that means running K training and validation test execution cycles. In the case of the closed-form leave-one-out cross validation, each **feval** corresponds to one training execution, with the validation function value being estimated through (4.1). Therefore, in the problem of hyper-parameter optimization, **feval** is a costly task which is

likely to be the dominant effort in the computation.

The experiment comprised the use of five different optimization methods: “Quasi-Newton”, “Grid Search”, “Random Search”, “Nelder-Mead” and “Literature”. “Quasi-newton” implements the model and algorithm described in Section 4.2, using an SQP solver, as mentioned previously. All other methods use the LapRLS formulation with integer parameters for ν and p . The “Nelder-Mead” method (Nelder & Mead 1965) is a heuristic-based derivative free search algorithm which has been included for its complementary characteristics to the other methods. All the methods had the possibility to keep track and limit their execution to a given maximum number \mathbf{feval}_{\max} of function evaluations. The definition of such value took in consideration the high relative cost characteristic of each \mathbf{feval} and looked for a lower bound criteria, taking “Grid Search” as a baseline.

In this case, if three candidate values are defined per each hyper-parameter dimension (lower, intermediate, upper), given that the hyper-parameter vector for this problem has five components, the traversal of the grid requires a total of $3^5 = 243$ \mathbf{fevals} , which was the limit adopted. The actual upper and lower values for the variables in the grid had also to be properly defined. We referred to a recent and closely related article (Melacci & Belkin 2011), which presents a comprehensive experimental section where different datasets and their cross-validated hyper-parameters, generated through a much finer grid-based cross validation search, are listed. In the present study, the upper and lower values of the variables are made equal to the rounded maximum and minimum values of the optimal parameters across all datasets. Such definition emulates prior ignorance over (or an educated guess for) the Manifold Regularization hyper-parameter values given the space of their typical values for common datasets. The resulting grid values are presented in Table 4.1. Using information from the same article, the “Literature” methods returns the specific (optimal) hyper-parameter values found for each dataset.

The values adopted for the grid in the “Grid Search” method are also used to define initial conditions and the search space for the other methods. For the case of “Quasi-Newton” and “Nelder-Mead”, the initial point was defined as the arithmetic mean hyper-parameter vector defined by the lower and upper values at each vector component. For “Random Search” \mathbf{feval}_{\max} points are generated according to a uniform distribution limited to the upper and lower values of each component.

Hyper-param.	lower		upper
$\log_2(\gamma'_A)$	-20	-10	0
$\log_2(\sigma)$	-2	2	5
ν	2	10	50
p	1	2	5
$\log_2(\gamma'_I)$	-20	0	10

Table 4.1: Hyper-parameters grid values.

The experiments were run for both the COIL20 and USPST datasets, using $l=100$ for the labeled examples and $u=500$ unlabeled ones. The test error rate results for 10 splits for each method can be seen in the boxplots in Figures 4.6 and 4.7. Average and standard deviation (in brackets) values are also presented in Table 4.2, with results for additional tests with a higher number of unlabeled examples, as well as results obtained for the case of supervised RLS (zero unlabeled samples), which are included for reference.

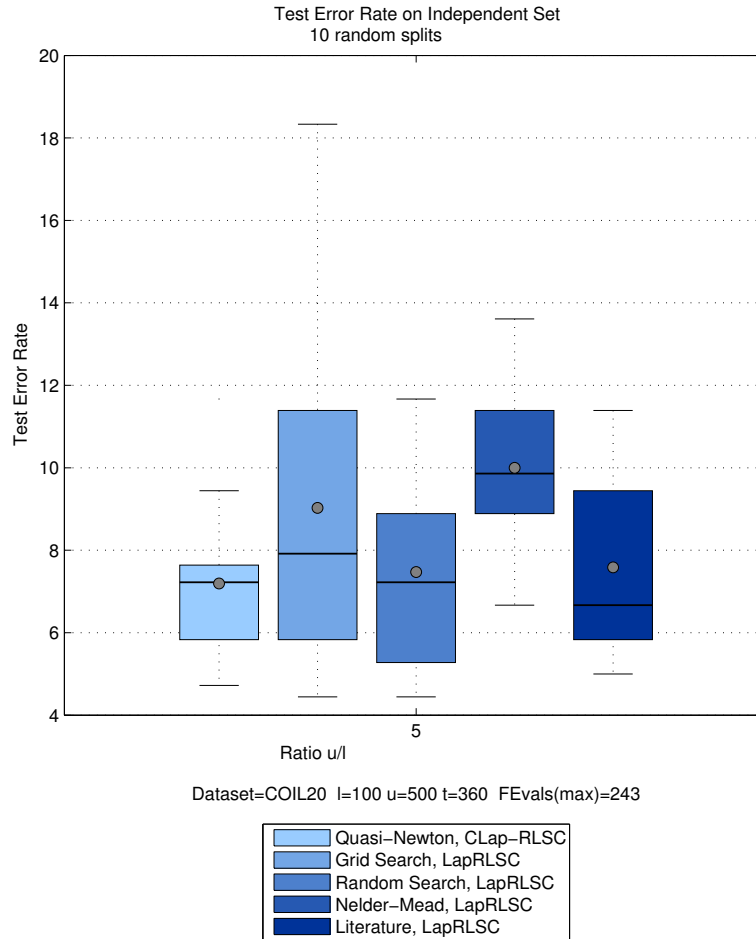


Figure 4.6: COIL20 dataset boxplots for test set error performance comparing different search methods.

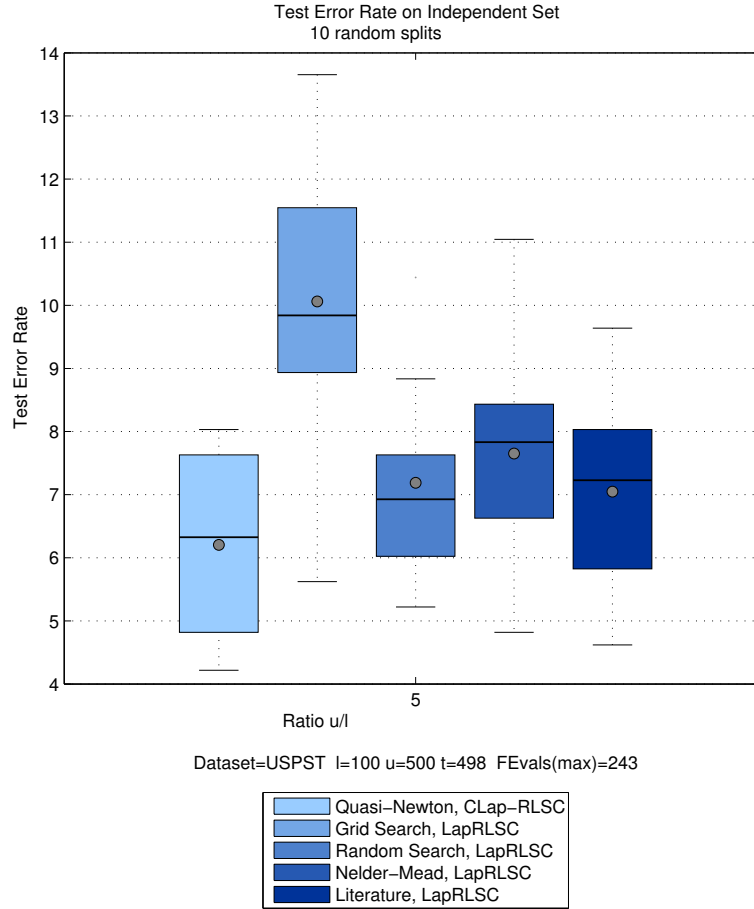


Figure 4.7: USPST dataset boxplots for test set error performance comparing different search methods.

Analyzing the results, it can be seen that, “Quasi-Newton” presented best average performance in most cases. This could be attributed to the freedom it enjoys in searching for locally optimal solutions with potentially large improvements in its first iterations, and fine increments as it progresses. Another reason could be the use of the combination of Laplacian matrices, which might have endowed the algorithm with a slightly higher learning capability. This effect would need to be clarified in a future study. “Quasi-netwon” was however closely followed by “Literature” and “Random”. One explanation for the slightly lower performance of “Literature” is that the parameters considered resulted from a search performed in a more general distribution of labeled and unlabeled examples. The poorer results exhibited by “Grid” (except for the last case) are more easily explained by the few points per dimension it was allowed. With the increase in sophistication of prediction models, as the recent multi-view and vector-valued regularization models (Rosenberg, Sindhwani, Bartlett & Niyogi 2009, Minh, Bazzani & Murino 2013), the dimensionality of the hyper-parameter

Data	Quasi-Newton	Grid	Random	Nelder-Mead	Literature
COIL20					
l=100 u=940	7.00 (2.38)	8.39 (4.18)	6.86 (2.10)	8.58 (2.47)	5.83 (2.67)
l=100 u=500	7.19 (2.14)	9.03 (4.20)	7.58 (2.44)	10.00 (1.94)	7.58 (2.35)
l=100 u=0 (RLSC)	8.28 (2.22)	8.97 (2.92)	8.33 (2.07)	8.33 (1.90)	10.81 (2.25)
USPST					
l=100 u=1000	4.60 (1.45)	6.55 (2.13)	4.74 (1.07)	6.35 (1.70)	5.68 (1.46)
l=100 u=500	6.20 (1.42)	10.06 (2.36)	7.09 (2.12)	7.65 (1.67)	7.05 (1.48)
l=100 u=0 (RLSC)	8.17 (1.64)	7.63 (1.64)	8.18 (1.61)	8.13 (1.73)	9.64 (2.41)

Table 4.2: Mean and standard deviation test error rate for five different methods of hyper-parameter search for Manifold Regularization compared to the supervised case. The best values found are shown in boldface.

vector is expected to increase and pose even stronger limitations on such type of search space optimization. “Nelder-Mead”, despite being a robust method which does not rely on the calculation of derivatives, typically needs a relatively larger number of function evaluations to accurately converge, and might have had its performance diminished by the truncation on \mathbf{feval}_{\max} . Nevertheless, both “Quasi-Newton” and “Nelder-Mead” were observed to commonly reach \mathbf{feval}_{\max} before fulfilling other stopping criteria. The performance of “Random”, when its simplicity of implementation is taken in balance, needs to be highlighted. One conjecture for its good performance is that the validation function landscape, although being highly dimensional, and in many cases multimodal, might typically contain regions of plateaus with a lower sensitivity to small changes in the hyper-parameters. This characteristic, if present, might potentially allow globally sub-optimal results with limited \mathbf{feval} calculations. To explore such possibility, a more thorough characterization of validation function landscapes for different datasets, as well as their correlation to actual test rates for Manifold Regularization models, is suggested as a future research topic.

Conclusion

This dissertation looked at two relevant aspects of recent machine learning topics, hyper-parameter optimization and semi-supervised learning. Apart from the review of their (from the eyes of the author) most important theoretical aspects and results from recent research, computational experiments have been developed over two practical datasets. Finally, a hyper-parameter optimization technique for Manifold Regularization models was proposed and applied, which encompassed most of the concepts presented. More specifically, the following conclusions and suggestions for future work can be highlighted.

The PRESS validation function appears to be a good indicator of out of sample performance, and can be used as a proxy function for model selection. It is also a convenient estimator, as it can be calculated as a by-product of training calculations, through the elements of the diagonal of the inverse of a known matrix. Furthermore, it is a continuously differentiable function, allowing for the definition of directions of local descent.

Manifold Regularization allows an increase in performance through the complementary use of unlabeled data. The improvement is greater and significant when the number of labeled examples is relatively small. The Manifold Regularization problem, as formulated, remains a convex problem and can be efficiently solved by a number of available optimization solvers. The uses of unlabeled data can also be beneficial when combined with standard supervised kernel methods, such as Laplacian Regularized Least Squares and Support Vector Machines, through the use of the Point Cloud Kernel. The development of such a kernel function, which embeds unlabeled information in its distance definition, illustrates the modular characteristic of the regularization and kernel machines framework.

Gradients for the Manifold Regularization are obtainable using by-product data of the training phase. The use of gradient optimization led to obtention of competitive performance rates when compared to the ones reported in the literature. In the case of the Laplacian

matrix for the manifold approximation, the adoption of a weighted combination composed by different numbers of nearest neighbors and iteration matrix exponents enabled additional gains in prediction performance.

Random search is a competitive search strategy which becomes more efficient than grid search when the number of parameters to be determined is large. Such behavior can be attributed to the fact that the uniform distribution produces points that tend to be uniformly spaced in each dimension individually; the coverage of the search space increases linearly, on average, with the number of points in each dimension. Such behavior is not obtainable with grid search, which traverses each dimension completely before producing an increment on the next nested dimension.

Given the apparent intractability of the hyper-parameter optimization problem, there have been many proposals for the adoption of metaheuristic-based search algorithms, for example (Friedrichs & Igel 2005, Guo, Yang, Wu, Wang & Liang 2008). These are probabilistic search algorithms inspired by analogies with natural processes, such as genetic algorithms, particle-swarm and ant colony optimizations. These algorithms usually do not take advantage of local information available with the gradient, and often require a number of parameter adjustments. There is likely to be room for a combination of both gradient and metaheuristic approaches through the application of what is known as local search techniques in the field of metaheuristic optimization.

Another possibility is to apply the generalized k -Fold cross validation definition as presented in (Pahikkala et al. 2006) and look for possible gains in the variance of the solution obtained. A related approach, which has been proposed by (Cawley & Talbot 2007) is to add a regularization term on the hyper-parameter objective function level. Work would be needed to determine appropriate forms for such regularizers, and how to determine the optimal overall regularization weights. The work presented in (Cawley & Talbot 2007, Cawley & Talbot 2010) is a good starting point.

Finally, Manifold Regularization algorithms face the high cost demanded by matrix operations on the size of the Laplacian matrix, of order $l + u$. There has been some research focusing on the use of numerical techniques such as Nyström decomposition or other sparse methods (Liu, He & Chang 2010, Talwalkar et al. 2008), which select a subset of the elements of the Laplacian matrix. Their main concern is estimating the impact of the loss of information resulting from the approximation, and providing criteria on how to best select a data subset. Progress in this area is likely to be directly transferred to the hyper-parameter optimization stage, as the overall problem size, and therefore its complexity, will be correspondingly reduced.

Publication Submissions

The following paper has been produced as a result of this work: *Gradient Hyper-parameter Optimization for Manifold Regularization*, submitted to the *ICMLA 2013 - 12th International Conference on Machine Learning and Applications 2013 Miami, Florida, USA - Workshop: Machine Learning Algorithms, Systems and Applications*. <http://icmla-conference.org/icmla13>

References

- Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle, *Second international symposium on information theory*, Akademinai Kiado, pp. 267–281.
- Allen, D. M. (1974). The relationship between variable selection and data agumentation and a method for prediction, *Technometrics* **16**(1): 125–127.
- Bache, K. & Lichman, M. (2013). UCI machine learning repository.
URL: <http://archive.ics.uci.edu/ml>
- Bazaraa, M. S. & Shetty, C. M. (1979). *Nonlinear Programming: Theory and Algorithms*, Wiley, New York.
- Belkin, M., Niyogi, P. & Sindhwani, V. (2006). Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, *The Journal of Machine Learning Research* **7**: 2399–2434.
- Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization, *The Journal of Machine Learning Research* **13**: 281–305.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.
- Boyd, S. & Vandenberghe, L. (2004). *Convex Optimization*, Cambridge University Press.
- Cawley, G. C. (2006). Leave-one-out cross-validation based model selection criteria for weighted ls-svms, *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, IEEE, pp. 1661–1668. Software retrieved from <http://theoval.cmp.uea.ac.uk/matlab/> on 16-Jul-2013.

- Cawley, G. C. & Talbot, N. L. (2004). Fast exact leave-one-out cross-validation of sparse least-squares support vector machines, *Neural networks* **17**(10): 1467–1476.
- Cawley, G. C. & Talbot, N. L. (2007). Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters, *The Journal of Machine Learning Research* **8**: 841–861.
- Cawley, G. C. & Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation, *The Journal of Machine Learning Research* **99**: 2079–2107.
- Chapelle, O. (2007). Training a support vector machine in the primal, *Neural Computation* **19**(5): 1155–1178.
- Chapelle, O., Schölkopf, B. & Zien, A. (eds) (2006). *Semi-Supervised Learning*, MIT Press, Cambridge, MA.
- Coppersmith, D. & Winograd, S. (1990). Matrix multiplication via arithmetic progressions, *Journal of symbolic computation* **9**(3): 251–280.
- Cortes, C. & Vapnik, V. (1995). Support-vector networks, *Machine learning* **20**(3): 273–297.
- Evgeniou, T., Poggio, T., Pontil, M. & Verri, A. (2002). Regularization and statistical learning theory for data analysis, *Computational Statistics & Data Analysis* **38**(4): 421–432.
- Evgeniou, T., Pontil, M. & Poggio, T. (1999). A unified framework for regularization networks and support vector machines.
- Fawcett, T. (2006). An introduction to roc analysis, *Pattern Recogn. Lett.* **27**(8): 861–874.
- Friedrichs, F. & Igel, C. (2005). Evolutionary tuning of multiple svm parameters, *Neurocomputing* **64**: 107–117.
- Geng, B., Xu, C., Tao, D., Yang, Y. & Hua, X.-S. (2009). Ensemble manifold regularization, *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 2396–2402.
- Gönen, M. & Alpaydın, E. (2011). Multiple kernel learning algorithms, *Journal of Machine Learning Research* **12**: 2211–2268.
- Guo, X., Yang, J., Wu, C., Wang, C. & Liang, Y. (2008). A novel ls-svms hyper-parameter selection based on particle swarm optimization, *Neurocomputing* **71**(16): 3211–3215.

- Hastie, T., Tibshirani, R. & Friedman, J. (2001). *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA.
- Huber, P. J. (1964). Robust estimation of a location parameter, *The Annals of Mathematical Statistics* **35**(1): 73–101.
- Joachims, T. (1999). Making large-scale svm learning practical., *Advances in Kernel Methods - Support Vector Learning*, MIT Press.
- Keerthi, S., Sindhvani, V. & Chapelle, O. (2007). An efficient method for gradient-based adaptation of hyperparameters in svm models, *NIPS 2006* .
- Liu, W., He, J. & Chang, S.-F. (2010). Large graph construction for scalable semi-supervised learning, *Proceedings of the 27th International Conference on Machine Learning*, pp. 679–686.
- MATLAB (2009). *version 7.8.0 (R2009a)*, The MathWorks Inc., Natick, Massachusetts.
- Melacci, S. & Belkin, M. (2011). Laplacian support vector machines trained in the primal, *Journal of Machine Learning Research* **12**: 1149–1184.
- Menon, A. K. (2009). Large-scale support vector machines: algorithms and theory, *Research Exam, University of California, San Diego* .
- Minh, H. Q., Bazzani, L. & Murino, V. (2013). A unifying framework for vector-valued manifold regularization and multi-view learning, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, p. 100–108.
- Nelder, J. A. & Mead, R. (1965). A simplex method for function minimization, *Computer Journal* **7**: 308–313.
- Nene, S. A., Nayar, S. K. & Murase, H. (1996). Columbia object image library (coil-20), *Dept. Comput. Sci., Columbia Univ., New York*. [Online] <http://www.cs.columbia.edu/CAVE/coil-20.html> **62**.
- Nocedal, J. & Wright, S. (2006). *Numerical optimization*, Springer series in operations research and financial engineering, 2. ed. edn, Springer, New York, NY.
- Pahikkala, T., Boberg, J. & Salakoski, T. (2006). Fast n-fold cross-validation for regularized least-squares, *Proceedings of the ninth Scandinavian conference on artificial intelligence (SCAI 2006)*, Citeseer, pp. 83–90.

- Platt, J. C. (1999). Advances in kernel methods, MIT Press, Cambridge, MA, USA, chapter Fast training of support vector machines using sequential minimal optimization, pp. 185–208.
- Rifkin, R. M. (2002). *Everything old is new again: a fresh look at historical approaches in machine learning*, PhD thesis, Massachusetts Institute of Technology.
- Rifkin, R. M. & Lippert, R. A. (2007). Notes on regularized least squares.
- Rosenberg, D., Sindhwani, V., Bartlett, P. & Niyogi, P. (2009). Multiview point cloud kernels for semisupervised learning [lecture notes], *Signal Processing Magazine, IEEE* **26**(5): 145–150.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review* **65**(6): 386.
- Rudin, W. (1986). *Real and complex analysis (3rd)*, New York: McGraw-Hill Inc.
- Schölkopf, B., Herbrich, R. & Smola, A. J. (2001). A generalized representer theorem, *Computational learning theory*, Springer, pp. 416–426.
- Schölkopf, B. & Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization and beyond*, the MIT Press.
- Shalev-Shwartz, S., Singer, Y. & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm, *Proceedings of the 24th international conference on Machine learning*, ACM, pp. 807–814.
- Shawe-Taylor, J. & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*, Cambridge University Press, New York, NY, USA.
- Sindhwani, V., Niyogi, P. & Belkin, M. (2005). Beyond the point cloud: from transductive to semi-supervised learning, *Proceedings of the 22nd international conference on Machine learning*, ACM, pp. 824–831.
- Suykens, J. A. & Vandewalle, J. (1999). Least squares support vector machine classifiers, *Neural processing letters* **9**(3): 293–300.
- Talwalkar, A. (2010). *Matrix approximation for large-scale learning*, PhD thesis, New York University.
- Talwalkar, A., Kumar, S. & Rowley, H. (2008). Large-scale manifold learning, *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, pp. 1–8.

- Tikhonov, A. & Arsenin, V. Y. (1977). *Solution of ill-posed problems*, Washington: Winston & Sons.
- Vapnik, V. & Sterin, A. (1977). On structural risk minimization or overall risk in a problem of pattern recognition, *Automation and Remote Control* **10**(3): 1495–1503.
- Vishwanathan, S., Schraudolph, N. N., Kondor, R. & Borgwardt, K. M. (2010). Graph kernels, *The Journal of Machine Learning Research* **99**: 1201–1242.
- Wahba, G. (1990). *Spline models for observational data*, Vol. 59, Society for industrial and applied mathematics.
- Yuan, J., Liu, X. & Liu, C.-L. (2012). Leave-one-out manifold regularization, *Expert Systems with Applications* **39**(5): 5317–5324.

Parameters and Hyper-parameters

The following table lists the main variables for the formulation of the hyper-parameter optimization problem, including the split of the data in labeled and unlabeled sets, the adoption of multiple validation partitions and multiple Laplacian combinations.

S	dataset
d	input dimension
\mathbf{x}_i	input data point
y_i	output data point
o	prediction function output
n	number of all input points
l	number of labeled input points
u	number of unlabeled input points
$\theta \in \Theta$	vector of hyper-parameters in hyper-parameter space
h	number of hyper-parameters
γ_A, λ	ambient space regularization coefficient
γ_I	intrinsic space regularization coefficient
σ	Gaussian kernel width parameter
ν	adjacency matrix number of nearest neighbors
p	power of Laplacian matrix
α	coefficient weights
q	number of Laplacians in combination
μ	Laplacian combination coefficients
$t \in 1 \dots T$	validation folds
C	training matrix
K	kernel matrix (Gram matrix)
L	Laplacian matrix
W	adjacency matrix

Table A.1: Some symbols and variables used in the text

Experimental Code

An excerpt from the test harness code used in the experiments is given below. The main execution loop and a central configuration file example were selected for the listing.

```
1 % Main Optimization Loop for Manifold Hyper-parameter Optimization
2 % Master Thesis – FEEC Unicamp – 2013
3 % Cassiano Otavio Becker
4 function hyperoptloop()
5 % select configuration file
6 conf_funcstr = 'conf_GradRBF';
7 % load function pointers
8 conf_func = str2func(conf_funcstr);
9 [conf, allexprs] = conf_func();
10 % load dataset description from configuration information
11 datasets = loadDatasets();
12 clc;
13 % iterate over selected datasets
14 for di=conf.d.data_sel
15     conf.c.di = di;
16     d = datasets(conf.c.di);
17     disp(strcat('***** DATASET: ',d.name, ...
18         '*****'));
19     % load dataset file
20     f= load(strcat(conf.d.datasetPath, '/', d.name, '.mat'));
21     % iterate over methods
22     exprs = conf.e.expr;
23     for e=1:size(exprs,2)
24         model = loadParams(d);
25         expr{e} = catstruct(conf,allexprs{exprs(e)});
26         expr{e} = setConf(expr{e});
27         % iterate over data splits for given dataset and method
```

```

28     for t=1:expr{e}.c.nsplits
29         expr{e}.c.t = t ;
30         % iterate over unlabeled number parameter
31         for up=1:expr{e}.c.npoints
32             % prepare dataset split training and test partition data
33             expr{e}.c.up = up;
34             nl = d.partition.l;
35             nu = d.partition.u;
36             umin = max(0, expr{e}.d.min_u);
37             umax = min(nu, expr{e}.d.max_u);
38             nus=floor(linspace(umin,umax,expr{e}.c.npoints));
39             [d, dt] = getPartitioned(t, f , up, nus, d, conf);
40
41             fprintf(['_____']...
42                 '_____\\n'])
43             fprintf(['# Method (%d of %d): %s - %s - %s |'...
44                 'Split %3d of %3d - U: %5.2f of %5.2f | %s \\n'],...
45                 e, ...
46                 length(exprs),...
47                 expr{e}.trainer,...
48                 expr{e}.optimizer,...
49                 expr{e}.kernel,...close
50                 expr{e}.c.t,...
51                 expr{e}.c.nsplits,...
52                 nus(expr{e}.c.up)/d.partition.l,...
53                 nus(expr{e}.c.npoints)/d.partition.l,...
54                 d.name...
55                 );
56             model= resetValHistory(model);
57             % main optimization loop, according to function pointers
58             % from configuration file
59             [er,model] = optimizeHyper(d,dt,expr{e}, model);
60
61             % populate history with test set evaluation performance
62             fval_path = model.val.history.fval;
63             tr_path = getTestPerformanceHistory(...
64                 dt,model,expr{e},expr{e}.p.path);
65             % format data for box plot
66             ern(e,t,up,:) = tr_path;
67             ern(e,t,up,end) = expr{e}.evalFunc(dt ,model,expr{e});
68             vrn(e,t,up,:) = fval_path;
69             vrn(e,t,up,end) = expr{e}.validationFunc(...
70                 model.best_param, d ,expr{e},model);
71         end
72     fprintf(['_____']...

```

```

73         '_____\\n'])
74     end
75     fprintf(['=====\\n'])
76     '=====\\n'])
77 end
78 % run boxplots
79 if conf.p.semi == 1
80     lims.nus = nus;
81     lims.nl = nl;
82     plotSemiBoxPlot(d,ern, vrn, lims, expr);
83 end
84 % run multiple split decay plot
85 if conf.p.feval == 1
86     plotFeval(ern, vrn);
87 end
88 % save statistics
89 filename = strcat('out/data/data_', conf.e.name, '_', d.name,...
90     '_', datestr(now,'yyyymmddTHHMMSS'));
91 save(filename, 'd', 'ern', 'vrn', 'lims', 'expr');
92 end
93 % close all figures
94 disp('--- > End of experiment - press any key to close all windows');
95 pause
96 close all;
97 end
98 end

```

```

1 function [conf, expr] = conf_All()
2 %% Experiment Configurations
3 i = 1;
4 expr{i}.optimizer = 'gridsearch';
5 expr{i}.kernel = 'rbf';% 'point_cloud';%
6 expr{i}.trainer = 'laprlsc';% 'lapmkrisc';% 'rlsc';% 'lssvm';%
7 expr{i}.validation = 'press';
8 i = 2;
9 expr{i}.optimizer = 'randsearch';
10 expr{i}.kernel = 'rbf';% 'point_cloud';
11 expr{i}.trainer = 'laprlsc';% 'lapmkrisc';% 'lssvm';% 'rlsc';%
12 expr{i}.validation = expr{1}.validation;% 'press';
13 i = 3;
14 expr{i}.optimizer = 'neldermead';
15 expr{i}.kernel = 'rbf';% 'point_cloud';
16 expr{i}.trainer = 'laprlsc';% 'laprlsc';% 'lapmkrisc';% 'lssvm';%
17 expr{i}.validation = expr{1}.validation;% 'press';

```

```

18 i = 4;
19 expr{i}.optimizer = 'gradsearch';
20 expr{i}.kernel = 'rbf';%; 'point_cloud';
21 expr{i}.trainer = 'laprlsc';%; 'laprlsc';%; 'lapmkrisc';%; 'lssvm';%
22 expr{i}.validation = expr{1}.validation;%'press';
23 i = 5;
24 expr{i}.optimizer = 'gradsearch';
25 expr{i}.kernel = 'rbf';%; 'point_cloud';
26 expr{i}.trainer = 'lapmkrisc';%; 'lssvm';%; 'laprlsc';%; 'rlsc';%
27 expr{i}.validation = expr{1}.validation;%'press';
28 i = 6;
29 expr{i}.optimizer = 'literature';
30 expr{i}.kernel = 'rbf';%; 'point_cloud';
31 expr{i}.trainer = 'laprlsc';%; 'laprlsc';%; 'lapmkrisc';%; 'lssvm';%
32 expr{i}.validation = expr{1}.validation;%'press';
33 % name for saving output files
34 conf.e.name = 'LapRLSC_150u500_all_g3';
35 % selection of methods to run
36 conf.e.expr = [5 1 2 3 6];
37 % number of folds for validation function (0 = closed form loocv)
38 conf.e.tfold = 0;
39 % validation function
40 conf.e.err = 'sos';%; 'err_count_smooth';%; 'err_count';%
41 conf.e.errfun = str2func(conf.e.err);
42 %% Dataset Configurations
43 conf.d.datasetPath = '/home/cassiano/msc/thesisCode/dataset';
44 % datasets to use
45 conf.d.data_sel = [1 2];
46 % load previously save partition indexes
47 conf.d.loadPartition = 1 ;
48 % number of unlabeled stations
49 conf.c.npoints = 1;
50 % number of random splits
51 conf.c.nsplits = 10;
52 % maximum number of fevals
53 conf.c.ntries = 243;
54 %% Optimization Configurations
55 % select which parameters will be optimized
56 conf.o.par_sel = [1:14];%11];
57 %% Plot Configurations
58 % enable plot parameter path on 2 dimensions
59 conf.p.path = 0;
60 % select which 2 parameters to plot
61 conf.p.path_sel = [1 2];
62 % enable plot parameter sensitivity

```

```
63 conf.p.par = 0;
64 % selection of parameters for which sensitivity will be plotted
65 conf.p.par_load = 1;
66 conf.p.par_sel = [2:5];
67 % enable boxplot for comparison of methods
68 conf.p.semi = 1;
69 % plot validation function box_plot if above option is one
70 conf.p.semi_val = 1;
71 % enable plot of error and validation against validation fevals
72 conf.p.semi_lines = 0;
73 % enable of multiple split decay plots
74 conf.p.feval = 0;
75 % enable saving plot
76 conf.p.save = 0;
77 %% Set Environment Structure
78 conf = setEnviron(conf);
79 end
80 end
```