A Regularization Framework For Active Learning From Imbalanced Data

by

Hristo Spassimirov Paskov

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2010

[June 2010]

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2010

Certified by .....
Tomaso A. Poggio
Eugene McDermott Professor in the Brain Sciences
Thesis Supervisor

Certified by .....
Lorenzo A. Rosasco
IIT–MIT Visiting Faculty
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

A Regularization Framework For Active Learning From Imbalanced Data

by

Hristo Spassimirov Paskov

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2010, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

We consider the problem of building a viable multiclass classification system that minimizes training data, is robust to noisy, imbalanced samples, and outputs confidence scores along with its predications. These goals address critical steps along the entire classification pipeline that pertain to collecting data, training, and classifying. To this end, we investigate the merits of a classification framework that uses a robust algorithm known as Regularized Least Squares (RLS) as its basic classifier. We extend RLS to account for data imbalances, perform efficient active learning, and output confidence scores. Each of these extensions is a new result that combines with our other findings to give an altogether novel and effective classification system.

Our first set of results investigates various ways to handle multiclass data imbalances and ultimately leads to a derivation of a weighted version of RLS with and without an offset term. Weighting RLS provides an effective countermeasure to imbalanced data and facilitates the automatic selection of a regularization parameter through exact and efficient calculation of the Leave One Out error. Next, we present two methods that estimate multiclass confidence from an asymptotic analysis of RLS and another method that stems from a Bayesian interpretation of the classifier. We show that while the third method incorporates more information in its estimate, the asymptotic methods are more accurate and resilient to imperfect kernel and regularization parameter choices. Finally, we present an active learning extension of RLS (ARLS) that uses our weighting methods to overcome imbalanced data. ARLS is particularly adept to this task because of its intelligent selection scheme.

Thesis Supervisor: Tomaso A. Poggio
Title: Eugene McDermott Professor in the Brain Sciences

Thesis Supervisor: Lorenzo A. Rosasco
Title: IIT–MIT Visiting Faculty

Acknowledgments

Special thanks to my amazing wife for her undying support and patience throughout the making of this thesis. My work would still be in a Word document were it not for your help with LaTeX.

I would like to thank Professor Tomaso Poggio and Dr. Lorenzo Rosasco for all of their help, instruction, and inspiration. I have learned more about Machine Learning and Learning Theory from my discussions with you than I could ever have by myself.

I would like to thank Dr. David Brock for being my mentor throughout MIT. You helped me find my passion for Machine Learning and have inspired so many ideas along the way.

I would like to thank all of the members of CBCL for all of the entertaining, relaxing, and thought inspiring discussions: Jim, Cheston, Andre, Nick, Gadi, Sharat, Charlie, Joel, Ethan, and Huei-han.

I would like to thank my mom and dad and brothers Ivan and Alex for their support and reassurance through my most difficult moments at MIT.

I would like to thank my friends Yue, Alex, Daniel, Phil, Francisco, Evan, Salil, Saureen, Ammar, and Greg for all of the good times and distractions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Computer-Based Classification

Classification is a fundamental learning problem in which a learner is tasked with finding a rule to separate data that is consistent with a set of labeled examples. The ultimate goal of this learning procedure is not to memorize the training data, but rather to find an accurate way to predict the labels of new examples. As such, the focus of classification is on prediction and a successful classifier is one that generalizes well to unseen data.

It is particularly interesting to see how well we can replicate and automate such intelligence with computer-based classifiers. Exploring and trying to implement the facets of human intelligence is a fascinating endeavor in itself because it drives at the fundamental questions of consciousness. On a more pragmatic level, classifiers are invaluable analysis tools for exploring, understanding, and predicting data in science and industry. To this end, the success of computer-based classification systems relies on overcoming the full gamut of issues that occur when collecting data, training classifiers, and using them.

1.2 Collecting Training Data

Building a viable classification system begins at the data collection phase, which is closely related to the problem of experiment design. The most expensive part of acquiring training data is often labeling it. Finding sample points can often be automated, but labeling each of these points requires human intervention. A canonical example of this effect is that it is easy to scan the internet for millions of images, but it would require an intractable number of man hours to label the objects in them. Sadly, traditional approaches to training classifiers are directly at odds with this requirement because they rely on large data sets for good generalization performance.

Complimentary to the issue of minimizing the number of labeled examples is that of choosing effective points to train on. Oftentimes all that is necessary to correctly classify a cluster of points is a single labeled example. However, bias, correlations, or bad luck in the sampling process used to generate training data may create a number of points which give little information beyond the first sample. At the very least, such spurious examples waste labels. In the worst case they mislead the classifier into overemphasizing some aspect of the data and hurt generalization.

Active learning addresses these issues by making learning interactive so that the learner may ask questions as it trains. In the context of classification, an active learner is one which starts with an initially unlabelled or partly labeled data set and is allowed to ask for the labels of any points it is interested in. The grand hope of active learning is to reduce the cost of training a learner by having it ask for the labels of only the relevant points it needs. A secondary effect of this procedure is that the active learner picks the most informative points and does not waste time training on spurious ones. Taken together, these goals can be seen as addressing the problem of experiment design. The learner is trying to perform as few experiments as possible, each of which must be properly informative if it is to classify and generalize correctly.

1.3 Training in the Real World

The second part of an effective classification pipeline is training a robust algorithm in an effective and timely manner. We stress robust because real world data is inevitably noisy and unlikely to be perfectly separable by a simple rule. Any classifier that is not built upon these assumptions is almost certain to generalize poorly because it will spend its efforts modeling noise and over-fitting the data. Relying on noise for its predictive qualities is a foolish endeavor because any "correlations" with the actual classification of a point are usually due to pure chance.

Adding to the difficulties of noisy data is that the training data may be - heavily - imbalanced. Imbalances occur when the relative proportions of some parts of the training data are different than what we wish to train on. Broadly speaking, data imbalance occurs largely as a result of:

1. Explicit and implicit faults in creating the training set that lead to a mismatch between the distribution of points we train on and what the classifier is likely to encounter when used.

2. The cost of mislabeling certain points is higher than the cost of erring on others.

A concrete example of both of these problems comes from training a classifier to detect certain kinds of rare cosmic phenomena. It may be that there are many thousands of times as many negative training examples as positive ones and that mislabeling a positive occurrence costs a researcher years of waiting. Without any knowledge of the imbalance, a reasonable classification rule is to label everything as negative because it is certain to be correct the overwhelming majority of the time. However, this defeats the entire purpose of using the classifier and is demonstrative of how imbalances can ruin a classifier's performance.

A particularly important instance of data imbalance comes from adapting binary classification algorithms to multiclass classification. While some of the most powerful classification algorithms focus exclusively on binary labels, real world problems often involve several if not many classes. Luckily, we can reuse these classifiers by trans-

forming the multiclass classification problem into a series of binary tasks. One of the most intuitive and effective techniques creates a classifier for each possible label and tasks it with classifying points as either belonging to the class or not.

This technique is useful in that it handles any number of classes easily, but it also creates a large data imbalance when there are many classes. Each "one vs. all" classifier obtains its positive examples from a single class and labels the rest of the training data as negative. As such, the imbalance between positive and negative samples increases with the number of classes. When there are one hundred classes, only one percent of the points are positive and the classifier faces the same cosmic confusion as our earlier example. It is clear that a classifier's success with the one vs. all regime and generally imbalanced data is predicated on its compensatory abilities.

1.4 Tuning Performance

Beyond dealing with the vagaries of real world data, the usability of a classifier is also heavily reliant on the ease and speed with which any manually adjusted parameters can be tuned. As discussed in more detail in the Preliminaries section, the classifier we use learns its decision rule by balancing its performance on the training data with how likely it is to be over-fitting. It critically relies on a regularization parameter that, roughly speaking, determines how cautious the classifier should be in fitting the data. A value that is too high will not separate the data properly, while a value that is too low will generalize poorly. As there is little intuition to be had about what a value of 0.1 or 0.6 really means for a particular data set, it is imperative that there be a fast and easy method to select the regularization parameter. In the ideal case, such a parameter would be determined automatically by the system.

1.5 Practical Classification

The final criticality of a classification system is how informative its results actually are. There is no guarantee that a classifier fits all of its training data perfectly and

there is even less of a guarantee that it will label every new point correctly. In fact, it is safe to assume that the classifier will make some mistakes with certainty; the goal of learning is best effort in that it minimizes the number of mistakes a classifier is likely to make. It is therefore necessary to know the confidence of each prediction so that the user may decide whether to trust the classifier or to resort to a more expensive means of classifying a difficult example. This confidence must be applicable to both, binary and multiclass classification.

An ideal classifier will know the conditional probability of each class given a point so that a confidence score corresponds to the likelihood of its decision. In reality, the confidence score must also take into account the classifier's imperfect knowledge of these conditional class probabilities. It is important to realize that the accuracy of the confidence score is therefore largely related to the quality of the classifier's probability model. As such, the accuracy and speed with which this model can be queried are vital; confidence is useful only if it provides a good estimate of the truthfulness of a classification and can be found without adding substantial overhead.

All in all, building an effective classification system involves a number of difficulties starting with the data collection phase and extending to the use of the resulting classifier. Properly resolving of each of these problems is essential to the success of the entire system and the usefulness of classification in general. To this end, we investigate the merits of a classification framework for binary and multiclass classification that uses a robust algorithm known as Regularized Least Squares (RLS) as its basic classifier. We tackle the problems associated with training and using our system by demonstrating an effective countermeasure to imbalanced data and several methods for estimating confidence. These findings motivate an active learning extension of RLS (ARLS) that effectively completes our classification pipeline by addressing the data collection phase.

1.6 Background

1.6.1 Classification

Some of the most successful algorithms for binary classification determine a simple line that separates positive and negative examples in a Euclidean space $\Re^n$. The forerunner of modern linear separators appeared in 1950's as the Perceptron algorithm [23] and remains influential some sixty years later. Today, Vapnik's Support Vector Machine (SVM) [28] is one of the most ubiquitous classifiers. It extends the original Perceptron by finding a linear separator with maximum margins between the decision boundary and training data. The SVM effectively projects data into a higher dimensional space and finds a linear separator there. Using kernels increases discriminative power because it corresponds to considering higher order interactions of the data's features that translate into finding a non-linear separator in the original space.

An alternative to the SVM is RLS, an algorithm initially designed for continuous regression because it minimizes the mean squared error of its training data in a stable manner [8]. As [21] discusses, RLS holds computational advantages over the SVM in that its solution can be found by solving a system of linear equations. Moreover, it is easy to tune the algorithm's regularization parameter because a classifier's exact LOO error can be found quickly. RLS can be adapted to classification by using binary labels for the function values it models. Indeed, recent work by [21] and [1] demonstrates its efficacy as a binary classifier and in one vs. all multiclass classification. The usual kernel tricks that the SVM exploits can also be applied to RLS, so it too can learn non-linear separators.

An important connection between SVMs and RLS elucidates the key to their generalization abilities. In particular, [8] show that the problem of learning is ill-posed because it has multitudes of unstable solutions. They then demonstrates that both, SVMs and RLS, overcome these issues through regularization. Indeed, both algorithms are instances of Tikhonov regularization - a powerful method for enforcing stability by choosing functions that have minimal training error and complexity. A further study into learning theory by [3] provides the critical link between stability and

learning: a stable solution is likely to generalize well beyond training data. Taken together, these studies provide a theoretical justification for the SVM and RLS as robust classifiers.

1.6.2 Classification Confidence

The success of SVMs has engendered several studies into estimating the confidence of these classifiers. It is worth noting that the SVM classifier relies on a subset of its training data for classification, i.e. the support vectors, and therefore only models the conditional class probabilities of these support vectors. All points that lie outside of the range of the support vectors receive "probabilities" that are greater than 1 and are not interpretable as meaningful measures of the likelihood of class membership.

Nonetheless, a variety of ad-hoc methods exist that try to convert the SVM's predictions into probabilities by effectively squashing the output to be between 0 and 1. The most popular method due to [18] converts a linear transformation of the SVM output into a probability by passing it into a sigmoid function. A summary of other methods for confidence estimation are presented in [20]. The author demonstrates that Platt's method is the best way to convert SVM outputs to confidences, but also that estimating the class conditional probabilities directly gives significantly better scores.

In contrast to the SVM, RLS uses all of its training data in the resulting classifier and should therefore be able to give better confidence estimates. As [1] demonstrates, the output of the RLS classifier directly converges to the probability of class membership as the training data increases. This property is desirable because it shows that RLS estimates confidence in a principled manner. An interpretation of RLS as a Gaussian process [19] presents an alternate way to estimate confidence through normal distributions. This estimate takes into account the classifier's variance and therefore uses more information in its score.

1.6.3 Active Learning

Active learning extends the traditional classification paradigm by making learning interactive in the hopes of using less training data. Algorithms in this area can be categorized as either pool based or streaming. The former assumes that all training data is given at once as a partially - or completely - unlabeled pool of examples while the latter operates on examples that arrive individually in a stream. It is worth noting that purely streaming active learners operate in a more pessimistic framework than pool based ones because a particular point may never come up again if it is not queried. However, a pool of points can be approximated in the streaming scenario by aggregating examples before training. Conversely, a stream is easily simulated by selecting points from a given pool at random. This close connection between active learning regimes gives credence to a discussion of algorithms from both realms.

One of the earliest active learning algorithms due to [26] queries a committee of classifiers with every point it receives. The algorithm improves its estimate of the correct hypothesis space, i.e. set of classifiers that fit the training data, by asking for the label of any new point its classifiers disagree on. Interestingly, a later analysis of the algorithm given in [9] shows its uses exponentially fewer training points than passive algorithms under certain data distributions. The merits of active learning are also explored by [10] who proves that an active filter can detect certain kinds of sparse signals that a passive scheme has no hope of finding.

Since the original query by committee algorithm, [27] has extended the notion of actively improving the hypothesis space to Transductive SVMs over a pool of unlabelled data. This algorithm is significant in its use of the powerful SVM classifier. A series of streaming active learning algorithms are given in [5] and [6] that can also use this classifier. The first algorithm determines whether querying a point will be useful through statistical complexity bounds applied to the "disagreement coefficient" of its classifier. Roughly speaking, this coefficient estimates how much classifier changes when the incoming point has different labels. The second algorithm employs an importance weighting scheme that asks for a point's label based on an

estimated probability of its usefulness.

Finally, [29] presents an algorithm that select which points to query from an initially unlabelled data pool by greedily minimizing a risk. The authors use an unregularized algorithm that propagates known labels to all points in the data pool via an estimate of underlying manifold of the data. We extend this algorithm by replacing the underlying classifier with RLS and exploring its behavior on imbalanced data sets.

1.7 Contributions

We consider the problem of building a viable multiclass classification system that minimizes training data, is robust to noisy, imbalanced samples, and outputs confidence scores along with its predications. These goals address critical steps along the entire classification pipeline that pertain to collecting data, training, and classifying. Our system uses RLS as its base classifier, which we extend to account for data imbalances, perform efficient active learning, and output confidence scores. Each of these extensions is a new result that combines with our other findings to give an altogether novel and effective classification system.

Our first set of results investigates various ways to handle multiclass data imbalances and ultimately leads to a derivation of a weighted version of RLS with and without an offset term. Weighting RLS provides an effective countermeasure to imbalanced data and facilitates the automatic selection of a regularization parameter through exact and efficient calculation of the Leave One Out (LOO) error. These derivations follow the reasoning in [21] and they present a unification of all relevant and otherwise disparate results that pertain to weighting RLS.

Next, we present two methods that estimate multiclass confidence from an asymptotic analysis of RLS [1] and another method that stems from a Bayesian interpretation of the classifier [19]. To the best of our knowledge, there has been no formal study investigating the applicability of these views to multiclass confidence estimation with RLS.

Our final contribution presents an active learning extension of RLS (ARLS) that uses our weighting methods to overcome imbalanced data. ARLS builds on [29] by replacing their harmonic classifier with RLS and showing how to attain a worst case complexity that matches that of standard RLS. This study is significant in its introduction of ARLS and its investigation of the algorithm in the face of imbalanced data.

1.8 Thesis Outline

The remainder of this thesis is structured as follows. Section 2 discusses preliminary material including our notation, relevant learning theory, and RLS for binary and multiclass settings. Our contributions are presented in section 3 where we first focus on the adapting RLS to imbalanced data and ultimately derive weighted RLS with and without an offset term. Next, we discuss three methods to estimate multiclass confidence with RLS and then use our results to derive an active learning extension of RLS. We then present experiments with synthetic and real data sets for all of our findings in section 4. Finally, we conclude in section 5 and discuss future directions.

Chapter 2

Preliminaries

In the spirit of keeping this thesis as self contained as possible, it is useful to go over the major concepts and notation we will be using throughout the remainder of the paper. The majority of the concepts presented in this paper require a good understanding of linear algebra and basic familiarity with functional analysis. We recommend [15] as a reference for the relevant functional analysis used in learning theory.

Section 2.1 opens with a brief discussion of the matrix notation used throughout this paper. We then introduce the kernel trick in Section 2.2 to motivate a quick discussion of Reproducing Kernel Hilbert Spaces in Section 2.3. The heart of the material is presented in Section 2.4 where we introduce the general theory behind learning and apply it to derive RLS. Finally, Section 2.5 shows a fast way to find a good regularization parameter and Section 2.6 extends binary classification to multiple classes.

2.1 Matrix Notation

Our matrix notation follows the standards used throughout much of linear algebra and we specify indices in a manner reminiscent of Matlab [12]. In particular, column and row vectors as well as constants are denoted by lower case letters while matrices will always be capitalized. The i^{th} element of a vector x is given by x_i, and the

element of matrix M located at the i^{th} row and j^{th} column is simply M_{ij}. The i^{th} row vector of M is given by M_i and the j^{th} column vector is similarly $M_{:,j}$.

At times we extend the aforementioned notation by using an index vector to specify a subset of the elements in a vector or matrix. In the context of an n dimensional vector x, and index vector l is a vector of integers between 1 and n such that if $z = x_l$ then $z_i = x_{l_i}$ for $i = 1, \ldots, |l|$. We can also apply l to the rows or columns of a matrix so that, for example, M_l returns a submatrix of the $n \times n$ matrix M whose rows are given by l. Finally, the statement $Z = M_{lk}$ where l and k are appropriately defined index vectors should be interpreted as a $|l| \times |k|$ matrix with $Z_{ij} = M_{l_i k_j}$ for $i, j \in \{1, \ldots, |l|\} \times \{1, \ldots, |k|\}$.

Our final piece of notation concerns the evaluation of functions on sets of points. In particular, if $f : \Re^d \to \Re$ a real valued function over points in $\Re^d$, then $f(X)$, where each row of X is a point in $\Re^d$, returns a vector of values, y, such that $y_i = f(X_i)$. In the case of a function with two arguments, $k : \Re^d \times \Re^d \to \Re$, $k(X, \cdot)$ denotes a vector of single valued functions $\bar{f}$ where $\bar{f}_i(z) = k(X_i, z)$. Here the $\cdot$ notation is used to indicate an unspecified argument. Finally, the function call $k(X, Z)$, where each row of X and Z is a point in $\Re^d$, returns a matrix M of values such that $M_{ij} = k(X_i, Z_j)$.

2.2 Kernels

The general problem of classification is one of inductive learning in that a learner must draw conclusions from a training set that can then be applied to new data. In this context, it is reasonable to compare new data points with ones in the training set, and kernels define a powerful way to do this. Given two points $x, y \in \Re^d$, a kernel k is a positive definite function $k : \Re^d \times \Re^d \to \Re$ that maps x and y to a real number serving as a comparison between the two. The simplest example of a kernel is the linear kernel which is the dot product of x and y:

$$k_1(x, y) := x \cdot y \tag{2.1}$$

The linear kernel is weak in the sense that it only compares points along corresponding

dimensions. Oftentimes, it is desirable to consider higher order interactions by comparing corresponding sets of dimensions. For instance, the second order interactions of x and y are given by finding

$$k_2(x,y) = \sum_{i=1}^{d}\sum_{j=1}^{d} x_i x_j y_i y_j \tag{2.2}$$

A careful reader will notice that k_2 corresponds to taking a dot product in the feature space $\Re^{d'}$ where $d' = \frac{d(d+1)}{2}$. Points are mapped to $\Re^{d'}$ by multiplying all pairs of dimensions in the original feature space. This approach is perfectly valid but it has to compute quadratically many terms and quickly becomes intractable as we consider even higher order interactions. The kernel trick remedies situation by noting that

$$k_2(x,y) = (x \cdot y)(x \cdot y) = k_1^2(x,y) \tag{2.3}$$

In general, the n^{th}-order interactions of two points are given by

$$k_n(x,y) := (x \cdot y)^n \tag{2.4}$$

and all possible interactions up to degree n can by computed via

$$k_{\leq n}(x,y) = (1 + x \cdot y)^n \tag{2.5}$$

A particularly interesting kernel that corresponds to an infinite dimensional space of all possible interactions is the RBF kernel.

$$k_{RBF}(x,y) := e^{\frac{||x-y||^2}{2\sigma^2}} \tag{2.6}$$

This kernel has an added width parameter σ that determines which degree of interactions we pay more attention to; as σ becomes large the RBF kernel behaves like the linear kernel.

2.3 Reproducing Kernel Hilbert Spaces

Thus far, we have motivated the discussion of various kernels as computationally efficient ways to compute dot products in arbitrarily high dimensional spaces. It turns out that this interpretation can be formalized through the use of Reproducing Kernel Hilbert Spaces (RKHS), leading to a powerful mathematical machinery with which to reason about learning. While we defer a formal and in depth discussion of RKHS to one of the suggested textbooks, we briefly discuss them to give a sense of their usefulness in learning theory.

An RKHS $\mathcal{H}$ is a Hilbert space of real valued functions over a compact set X such that for every $x \in X$, every $f \in \mathcal{H}$ is bounded by $|f(x)| \leq M||f||_{\mathcal{H}}$ where M is nonnegative. Keeping with our running example of $\Re^d$, X can be any compact set within the Euclidean space. $\mathcal{H}$ would be any set of real valued functions that take an element from X as input and satisfy the aforementioned boundedness requirements.

A consequence of this definition shows that there is a one to one correspondence between kernels and RKHS so that a kernel k induces an RKHS $\mathcal{H}_k$ when used over a compact set X. The kernel plays an important role in $\mathcal{H}_k$ because the set of single input functions $\{k(x, \cdot)|x \in X\}$ obtained from using $x \in X$ as an argument to k form a basis for $\mathcal{H}_k$. As such, any function $f \in \mathcal{H}_k$ is a (possibly infinite) linear combination of k_x where $x \in X' \subseteq X$.

Another illuminative theorem due to Mercer shows that if k is continuous and $x, y \in X$

$$k(x, y) = \sum_{i=1}^{\infty} \sigma_i \psi_i(x) \psi_i(y) \tag{2.7}$$

where σ_i and ψ_i are the eigenvalues and eigenfunctions of k. This demonstrates that k is a dot product in some feature space induced by ψ. For example, ψ maps $\Re^d \rightarrow \Re^{d'}$ by computing all second order interactions in our earlier example with k_2. Taking our property of induced RKHS with Mercer's theorem demonstrates that any function in $\mathcal{H}_k$ is actually a weighted dot product with some subset of X in a feature space induced by k. This last result foreshadows an important theorem that allows us to

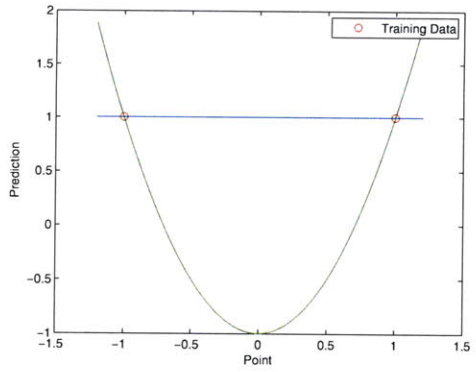


Figure 2-1: Without any restrictions, there are multiple ways to fit the positively labeled points at -1 and 1. In particular, the label of 0 can be either positive or negative.

compute a powerful set of classifiers.

2.4 Classification, Tikhonov Regularization, and RLS

Suppose that we are given a set of n examples $X = \{x_1, x_2, \ldots, x_n\}$ with corresponding binary labels $y = (y_1, y_2, \ldots, y_n)^T$ where each $x_i \in \Re^d$ and $y_i \in L$. In the binary case, $L = \{-1, 1\}$, whereas the more general multiclass setting of T possible classes is obtained by letting $L = \{1, 2, \ldots, T\}$. Unless explicitly stated, we assume that labels are binary valued for simplicity.

Binary classification can be formalized as the problem of finding a function $c : \Re^d \to \{-1, 1\}$ that will correctly classify unseen points given a training set X, y. The classifier c is usually constructed by thresholding a real valued function $f : \Re^d \to \Re$ such that $f(x) \geq 0$ is considered a positive label and $f(x) < 0$ a negative one. We will abuse terminology by calling f a classifier and distinguish between its value and thresholded classification as the "soft" and "hard" classification, respectively. Classification thus focuses on finding a real valued function that will lead to useful predications when thresholded in a simple manner.

It turns out that it is impossible to find such a function without imposing some

kind of restrictions. To see why, consider using the set of all polynomials as possible classifiers. As Figure 2-1 demonstrates, we can find two polynomials that perfectly fit a training set, yet disagree on the label of another point. In the absence of other information, how should we choose between classifiers?

The answer to this question comes from Occam's Razor and lies at the heart of much learning theory: the simplest answer is usually the best one. A particularly powerful approach to enforcing this requirement is neatly formulated in Tikhonov regularization [8].The solution to Tikhonov regularization is a function that balances minimizing a loss function on the training set with minimizing its norm in the RKHS:

$$f^* = \arg\min_{f \in \mathcal{H}_k} \frac{1}{n} \sum_{i=1}^{n} V(y_i, f(x_i)) + \lambda ||f||_k^2 \tag{2.8}$$

Here V is a nonnegative and monotonic loss function that penalizes errors on the training set while the second term penalizes complicated functions and therefore mitigates over-fitting. It is worth mentioning that using the hinge loss [28] leads to SVMs while the squared loss results in RLS. Thus, the RLS objective is expressed as

$$f^{RLS} = \arg\min_{f \in \mathcal{H}_k} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda ||f||_k^2 \tag{2.9}$$

While the functional formulation of RLS is a useful starting point because it gives a good interpretation of regularization, it does not reveal an easy way to solve the objective. Luckily, much of the heavy lifting in finding f^{RLS} is done by the Representer Theorem which transforms the RLS problem into a simple optimization procedure over $\Re^n$. As hinted at in our discussion of RKHS, the solution to Tikhonov regularization - and hence RLS - can be expressed as a linear combination of n kernel functions, each taking an argument from the training set X. Formally, for some $c^* \in \Re^n$

$$f^{RLS}(\cdot) = \sum_{i=1}^{n} c_i k(x_i, \cdot) \tag{2.10}$$

This representation reveals that the RLS objective is tantamount to finding an optimal value of c^*. In fact, the problem reduces to a convex quadratic optimization problem

by defining the kernel matrix K to be an $n \times n$ matrix with $K_{ij} = k(x_i, x_j)$:

$$c^* = \arg\min_{c \in \Re^n} ||y - Kc||^2 + \lambda c^T K c \tag{2.11}$$

The minimizing value c^* can be obtained through simple differentiation and is given by

$$c^* = G^{-1} y \tag{2.12}$$

where $G = K + \lambda I$ is the original kernel matrix with λ added to its main diagonal. Thus, the optimal RLS classifier for a specific choice of kernel and regularization parameter can be found by simply solving a system of linear equations. We continue our derivations to show that RLS' solution is particularly amenable to finding a good λ through LOO cross validation.

2.5 LOO Error and the Regularization Parameter

Once we fix the loss function, there are two parameters that remain unspecified in the RLS objective. The first is implicitly stated as the kernel and it defines the manner in which we compare points and the RKHS from which f^{RLS} can be selected. As mentioned earlier, selecting an appropriate kernel is an entire field of study onto itself and is largely dependent on the kind of data that is being modeled. The second parameter, λ, appears explicitly in the formula and it forces us to select smoother functions that are less prone to over-fitting as it increases. Luckily, an appropriate regularization parameter can easily be found by considering the LOO error of f^{RLS} for different values of λ.

Expressing the RLS solution as a matrix inversion neatly demonstrates the role of λ. Equation (2.12) shows that RLS is a kind of spectral filter that modifies the eigenvalues of the original kernel matrix. In particular, c^* is obtained by adding λ to every eigenvalue of the kernel matrix before inverting it. It follows that any eigenvalues that are much smaller than λ will be filtered out because the regularization

parameter mitigates their role in the final solution. An overbearing choice of λ will filter out too much of the information stored in the kernel matrix, while an overly small value will lead the classifier to model noise.

The spectral interpretation of λ's effect on the RLS solution demonstrates that it is critical to select a good value based on the eigenvalues of the kernel matrix. It is reasonable to assume that we will not want to filter beyond the extreme eigenvalues of the kernel matrix so that an optimal λ should occur within the range of its eigenvalues. To this end, an easy way to find this value is to search over a geometric series that starts and ends at the smallest and largest eigenvalues, respectively. The best regularization parameter is the one which minimizes either the LOO squared error or LOO classification error - the choice of which error to use is merely a matter of preference.

Computing the LOO error naively takes $O(n^4)$ for each trial because we must compute n different RLS solutions, each of which requires $O(n^3)$ calculations. It turns out that we can reuse our efforts spent in solving a single solution to compute the LOO error $L_i^E = y_i - f_{S^i}(x_i)$. Letting f_{S^i} be the RLS classifier trained on all points but x_i

$$L^E = \frac{c}{diag(G^{-1})} \tag{2.13}$$

where division is performed element-wise. We can further optimize our ability to calculate L^E by computing the eigenvalue decomposition of K. Since regularization affects only the eigenvalues of K, we can compute the numerator and denominator of L^E in $O(n^2)$ steps. This trick allows for $O(n)$ candidate values of λ to be tried while maintaining the usual RLS runtime of $O(n^3)$ calculations.

2.6 Multiclass Classification

Our discussion of classification has thus far been relegated to binary labels where a single threshold is applied to RLS to obtain the class label. However, this reasoning breaks down with multiple classes because using two or more threshold values enforces

an undesirable ordering over the classes. Consider a simple three class example in which we threshold the classifier as

$$c(x) = \begin{cases} 1 & f(x) < 0 \\ 2 & 0 \leq f(x) < 1 \\ 3 & 1 \leq f(x) \end{cases} \tag{2.14}$$

However, this choice of thresholds implies that class 1 is more to similar to class 2 than it is to 3. In general we have no reason to believe that this is the case, so the implied ordering is undesirable and likely to be detrimental to performance.

This situation can be rectified by encoding each class as a series of binary labels. While there are a number of ways to do this, we focus on the simple one vs. all regime. Here a separate function f^k is trained for each class by setting the labels of all points belonging to k as positive and the labels of all other points as negative. Thus, the encoded one vs. all labels for class k are given by

$$y_i^k = \begin{cases} 1 & y_i = k \\ -1 & y_i \neq k \end{cases} \tag{2.15}$$

Each f^k serves to distinguish its own class from all other classes. This scheme leads to a simple maximum decision rule in that the label of a point is chosen as the corresponding highest voting classifier. Note that a label is chosen even if all of the classifiers evaluate to negative values.

Chapter 3

Algorithms and Analysis

The Algorithms and Analysis chapter presents the new contributions of this thesis. We investigate the merits of a classification framework that uses a RLS as its base classifier. Our results address the various problems that arise when building a multiclass classification system for use on real world data. Specifically, we focus on the issues of minimizing training data, overcoming noisy, imbalanced samples, and outputting accurate confidence scores.

This chapter's structure follows the various stages of the classification pipeline. Section 3.1 tackles the issue of training and tuning RLS on imbalanced data and ultimately ends with a derivation of weighted RLS. We then discuss three different ways to estimate the confidence of multiclass predications in section 3.2. Finally, section 3.3 focuses on minimizing training data by presenting an active extension of RLS that we adapt to imbalanced data.

3.1 Strategies for Correcting Imbalanced Data

We begin our exploration of effective classification systems by examining various ways to adapt RLS to account for imbalanced data. These techniques do not require additional data collection and are initially motivated by considering the simple binary case. We then explore their behavior in the context of a one vs. all multiclass regime and show that the only simple and effective way to deal with imbalanced data is by

reweighting the squared loss.

Data imbalance occurs in binary classification when the relative proportion of positive and negative training examples does not match the proportions we wish to train our classifier on. Such disparities can be created by implicit or explicit faults in the training set that misrepresent the actual distribution of positive and negative examples. A second cause of imbalance is that the cost of mislabeling a positive example may be different than the cost of erring on a negative one.

To formalize our discussion, suppose that we wish to train an RLS classifier on a set of n binary labeled examples X, y. Let n_+ and n_- be the number of positive and negative training examples, respectively, so that $n = n_+ + n_-$. Without loss of generality, assume that a data imbalance exists because there are too few positive training examples so that we wish to train as if there were αn_+ of them, where $\alpha > 1$. There are a number of modifications we might make to the RLS objective to try to correct this imbalance:

1. Duplicate positive examples to obtain αn_+ of them.

2. Weight the squared loss on positive examples as $w_1 = \frac{\alpha}{\alpha n_+ + n_-}$ and $w_{-1} = \frac{1}{\alpha n_+ + n_-}$ for negative points. This strategy leads to a loss of the form

$$\sum_{i=1}^{n} w_{y_i}(y_i - f(x_i))^2 \tag{3.1}$$

3. Use a smaller regularization term on coefficients that correspond to positive training samples. Letting X_+ and X_- correspond to the sets of positive and negative training examples, respectively, the RLS classifier can be expressed as

$$f(\cdot) = \sum_{i=1}^{n} c_i k(\cdot, x_i) = k(\cdot, X_+)c_+ + k(\cdot, X_-)c_- = f^+(\cdot) + f^-(\cdot) \tag{3.2}$$

We can express its norm as two separate components so that $||f||_K^2 = ||f^+||_K^2 + ||f^-||_K^2$. Setting $\lambda_- > \lambda_+ \geq 0$, our last strategy leads to an RLS objective with

regularization term

$$\lambda_+ ||f^+||^2 + \lambda_- ||f^-||^2 \tag{3.3}$$

4. Increase the magnitude of the positive labels to some $\alpha > 1$.

It turns out that the first three strategies are equivalent. To see this, modify the RLS objective in equation (2.11) so that it has a weighted squared loss:

$$J(c) = (y - Kc)^T A(y - Kc) + \lambda c^T Kc \tag{3.4}$$

Here A a diagonal weight matrix whose entries are appropriately w_1 or w_{-1}. The minimizer of this equation is given by taking derivatives and solving to obtain

$$(AK + \lambda I)c = Ay \tag{3.5}$$

$$\rightarrow c = (AK + \lambda I)^{-1} Ay \tag{3.6}$$

However, if we multiply the left side by A^{-1}, we get

$$(K + \lambda A^{-1})c = y \tag{3.7}$$

$$\rightarrow c = (K + \lambda A^{-1})^{-1} y = (AK + \lambda I)^{-1} Ay \tag{3.8}$$

This latter formulation corresponds to choosing $\lambda_+ = \frac{\lambda}{w_1} < \lambda_{-1} = \frac{\lambda}{w_{-1}}$. As such, strategies (2) and (3) are equivalent. Indeed all three are the same, although we defer the proof of the equivalence of (1) and (2) to the appendix for brevity.

The equivalence of strategies (1) through (3) decreases the number ways we can counter data imbalances; our options are either to weight the loss or to change the relative magnitudes of labels. Both of these methods work well as long as RLS is used strictly for binary data. However, the next subsection examines data imbalance in multiclass classification and demonstrates that relabeling is completely ineffective in this setting.

Data Imbalance in Multiclass Settings

Multiclass classification is strictly harder than binary classification from the perspective of imbalanced data because it is unavoidable in the former and may never occur in the latter. In particular, one vs. all regimes force each classifier to train on significantly fewer positive examples than negative ones. This data imbalance is warranted because it correctly demonstrates that any individual class is unlikely in the face of so many others. However, imperfect knowledge of the data's generative probability distribution often leads the classifier to label everything as a negative and to give undesirable LOO error curves that favor no regularization. Such strange behavior ultimately hurts classification accuracy and leads to suboptimal multiclass predictions.

We first consider using strategy (4) to overcome the data imbalance because it can be accomplished with little effort. Indeed at first blush, changing the relative magnitudes of positive and negative labels seems to be an effective technique for correcting data imbalance. However, its efficacy in binary classification is misleading. Relabeling fails with multiclass classification because of how the winning class is chosen.

To show this, suppose that we are given n training examples in a matrix X with labels between 1 and T specified in vector y. Assuming that there are exactly $\frac{n}{T}$ examples from each class, using a one vs. all regime corresponds to training classifiers $f^1, f^2, \ldots, f^T \in \mathcal{H}_k$ each on $n_+ = \frac{n}{T}$ positive examples and $n_- = \frac{T-1}{T}n$ negative examples.

We can save on processing time by using the same regularization parameter for each class and finding $G^{-1} = (K + \lambda I)^{-1}$ once. The coefficients of f^t can easily be found by $c^t = G^{-1}y^t$, where y^t is the one vs. all encoding of labels for class k. Our balancing scheme is therefore tantamount to using labels a and $-b$, where $a > b > 0$, for positive and negative examples, respectively. Let $[k]$ be the index vector of all elements that belong to class k. Then

$$c_i^t = G_i^{-1}y^t = \sum_{j=1}^{n} G_{ij}^{-1}y_j^t = \sum_{k=1}^{T} G_{i[k]}^{-1}y_{[k]}^t = \sum_{k=1}^{T} y_k G_{i[k]}^{-1}\bar{1} \tag{3.9}$$

By assumption, $y_k = -b$ for all classes but t so that

$$c_i^t = aG_{i[t]}^{-1}\bar{1} - b\sum_{k=1,k\neq t}^{T} G_{i[k]}^{-1}\bar{1} = (a+b)G_{i[t]}^{-1}\bar{1} - b\sum_{k=1}^{T} G_{i[k]}^{-1}\bar{1} = (a+b)G_{i[t]}^{-1}\bar{1} - bG_i^{-1}\bar{1} \quad (3.10)$$

It follows that when we classify a point, we take

$$f^t(x) = \sum_{i=1}^{n} c_i^t(x_i, x) = \sum_{i=1}^{n} k(x_i, x)[(a+b)G_{i[t]}^{-1}\bar{1} - bG_i^{-1}\bar{1}] \quad (3.11)$$

A one vs. all multiclass scheme chooses the best class as the one with the highest scoring classifier f^*. Put another way, the winning class is one for which $f^*(x) - f^j(x) \geq 0$ for $j = 1, \ldots, T$. Comparing the difference between any two classes s and t,

$$f^s(x) - f^t(x) = \sum_{i=1}^{n} k(x_i, x)[(a+b)G_{i[s]}^{-1}\bar{1} - bG_i^{-1}\bar{1}] - \sum_{i=1}^{n} k(x_i, x)[(a+b)G_{i[t]}^{-1}\bar{1} - bG_i^{-1}\bar{1}] \quad (3.12)$$

$$= (a+b)\sum_{i=1}^{n} k(x_i, x)[G_{i[s]}^{-1}\bar{1} - G_{i[t]}^{-1}\bar{1}] \quad (3.13)$$

It follows that as long as $a, b > 0$, changing their magnitudes will have no effect on the difference, only its magnitude. As such, the multiclass prediction will stay the same and no correction for imbalance is actually performed. Hence, strategy (4) is ineffective in the multiclass setting and should be avoided in binary RLS implementations that might be used for multiclass classification.

We thus turn to reweighting the squared loss so as to equilibrate the number of positive and negative training examples. This popular procedure does not require special handling of the classifiers' outputs when using the maximum decision rule because each classifier faces the same reapportionment of positive training examples. Equilibrating the training data forces each classifier to spend as much effort minimizing loss on positive examples as on negative ones and therefore gives a clearer

estimate of the distribution of positive samples. Continuing with our earlier multiclass problem, weights that fix the data imbalance are given by

$$w_1 = \frac{\left(\frac{n_-}{n_+}\right)}{2n_-} = \frac{T-1}{2n_-} \tag{3.14}$$

$$w_{-1} = \frac{1}{2n_-} \tag{3.15}$$

It turns out that equilibrating positive and negative training examples in this manner can be justified through a multiclass objective. Suppose that we wish to maximize the difference between the value of the correct classifier of each training point and the $T-1$ incorrect classifiers. The objective we wish to maximize can be expressed as

$$J(f^1, f^2, \ldots, f^T) = \sum_{i=1}^{n} \sum_{j=1, j \neq y_i}^{T} [f^{y_i}(x_i) - f^j(x_i)] \tag{3.16}$$

Sadly J is not well defined because we can select arbitrarily high values for $f^{y_i}(x_i)$. However, we can make it well posed by forcing each $f^{y_i}(x_i)$ to be as close as possible to 1, and conversely, each $f^j(x_i)$ for $j \neq y_i$ to be as close as possible to -1. The new objective that we wish to *minimize* is

$$J'(f^1, f^2, \ldots, f^T) = \sum_{i=1}^{n} \sum_{j=1, j \neq y_i}^{T} [V(f^{y_i}(x_i), 1) + V(f^j(x_i), -1)] \tag{3.17}$$

where $V(\cdot, \cdot)$ is a loss function such as the squared loss. This objective retains our goal of maximizing the difference between correct and incorrect classifiers on the training data. Multiplying the objective by $\frac{1}{2n_-}$ does not change the optimization problem and we can rearrange terms to see that

$$\begin{aligned}
\frac{1}{2n_-}J'(f^1, f^2, \ldots, f^T) &= \frac{1}{2n_-}\sum_{i=1}^{n}(T-1)V(f^{y_i}(x_i), 1) + \frac{1}{2n_-}\sum_{j=1}^{T}\sum_{i\in\{i|y_i\neq j\}} V(f^j(x_i), -1) \\
&= \frac{1}{2n_-}\sum_{j=1}^{T}\left[\sum_{i\in\{i|l_i=j\}}(T-1)V(f^j(x_i), -1) + \sum_{i\in\{i|y_i\neq j\}} V(f^j(x_i), -1)\right] \\
&= \sum_{j=1}^{T}\left[\sum_{i=1}^{n} w_{y_i^j} V(f^j(x_i), y_i^j)\right] \qquad (3.18)
\end{aligned}$$

Hence J' can be minimized by minimizing a weighted loss for each one vs. all classifier separately. If we replace V with the squared loss, we recover the weighted RLS loss for each one vs. all classifier. As such, using w_1 and w_{-1} corresponds to minimizing a multiclass loss whose empirical success we demonstrate with our experiments.

In conclusion, there are a number of reasonable strategies we might employ to correct imbalanced data. It turns out that duplicating points and using different regularization parameters for each class are equivalent to weighting the squared loss. The two primary methods for dealing with imbalances in binary classification are to weight the squared loss or to use different magnitudes for positive and negative labels. The latter is risky since it does not translate to multiclass classification because of the maximum decision rule. Since multiclass classification induces its own data imbalances, we can only recommend weighting RLS' loss as an effective technique for dealing with imbalance. We thus apply our findings to derive weighted versions of RLS in the next two subsections.

3.1.1 Weighted RLS

The inefficacy of relabeling and the equivalence of our aforementioned balancing methods give impetus to the derivation of a weighted version of RLS. In particular, consider generalizing the usual RLS objective to

$$J(f) = \sum_{i=1}^{n} a_i(y_i - f(x_i))^2 + \lambda||f||_k^2 \qquad (3.19)$$

where $a_i \geq 0$ and $\sum_{i=1}^{n} a_i = 1$. Appealing to the Representer Theorem and letting A be a positive definite diagonal matrix with $A_{ii} = a_i$ allows us to rewrite the objective as

$$J(c) = (y - Kc)^T A(Y - yc) + \lambda c^T Kc \tag{3.20}$$

It turns out that we can recover the usual RLS problem by defining $W = A^{\frac{1}{2}}$ and

$$Z = Wy \tag{3.21}$$

$$M = WKW \tag{3.22}$$

$$d = W^{-1}c \tag{3.23}$$

Note that M is symmetric and positive semidefinite, so it too is a valid kernel matrix. These substitutions lead to

$$J(c) = J'(d) = (z - Md)^T(z - Md) + \lambda d^T Md \tag{3.24}$$

Following the standard definition of $G = M + \lambda I$, we know the solution to J' is

$$d = (M + \lambda I)^{-1}z = G^{-1}z \tag{3.25}$$

$$c = WG^{-1}z \tag{3.26}$$

Next, suppose that we wish to find the LOO error of weighted RLS quickly. We achieve a similar result to equation (2.13) that allows us to calculate the LOO error in $O(n^2)$ time. In particular, the *un-weighted* LOO error $L_i^E = y_i - f_{S^i}(x)$ is derived in the appendix and is given by

$$L^E = \frac{c}{diag(AG^{-1})} \tag{3.27}$$

We have hereby demonstrated a simple extension to RLS that overcomes data

imbalances by weighting the squared loss in RLS' objective. However, even this classifier is unlikely to perform well because it lacks an offset term with which to overcome limitations discussed in the next subsection. Thus, the next subsection completes our RLS derivations by defining the offset RLS problem and showing how to calculate all necessary parameters with a weighted loss.

Weighted Offset RLS

The need for an unpenalized offset in RLS is easily motivated by considering the RLS solution obtained from using a linear or polynomial kernel. In particular, any linear RLS classifier evaluates to zero when its input is zero and is thus constrained to go through the origin. This constraint forces points on opposite sides of the origin, as divided by the classifier's hyperplane, to have different labels. However, if all points around the origin have the same label, RLS will always misclassify some of them. This limitation can unnecessarily compromise performance and thereby invalidate any attempts to correct data imbalances. We thus "complete" RLS' predictive abilities by allowing it to use an unpenalized offset term.

In order to set the stage for our derivations, consider the usual binary classification setting in which we are given a training set of n binary labeled examples X, y. Our extension changes the RLS prediction to be of the form $f^{RLS}(x) = k(x, X)c + b$. In order to determine appropriate values for c and b, we modify the RLS objective to be

$$J(f, b) = \sum_{i=1}^{n} a_i(y_i - f(x_i) - b)^2 + \lambda ||f||_{\mathcal{H}}^2 \tag{3.28}$$

We again appeal to the Representer Theorem to write our objective as a quadratic matrix problem

$$J(c, b) = (y - Kc - b\bar{1})^T A(y - Kc - b\bar{1}) + \lambda c^T Kc \tag{3.29}$$

Here $\bar{1}$ is the column vector all ones. Making the usual definitions $W = A^{\frac{1}{2}}$, $z = Wy$, $M = WKW$, $d = W^{-1}c$ and letting $w = diag(W)$, we get that

$$J'(d,b) = (z - Md - bw)^T(z - Md - bw) + \lambda d^T Md \tag{3.30}$$

Relegating all further calculations to the appendix, the optimal values of c and b are given by

$$c = W(M + \lambda I)^{-1}(z - bw) = WG^{-1}(z - bw) \tag{3.31}$$

$$b = \frac{z^T r}{w^T r} \tag{3.32}$$

where $r = G^{-1}w$. If we let R be the diagonal matrix such that $R_{ii} = r_i$, the corresponding leave one out error for offset weighted RLS is

$$L_E = \frac{c}{\left(diag(AG^{-1}) - \frac{ARr}{w^T r}\right)} \tag{3.33}$$

The derivation of a weighted offset RLS concludes our study of imbalanced data in the context of standard classification. It is clear that the RLS framework leads to effective extensions for correcting data imbalances in the binary and multiclass settings. Our weighted extensions retain the robust properties of RLS and afford simple and fast solutions to calculating all necessary parameters and LOO errors. As such, RLS demonstrates its merits as a base classifier in any effective classification system. We now turn to several ways to estimate the confidence of the RLS classification.

3.2 Estimating Confidence

Once training data has been gathered and a classifier properly trained, it is useful to have some measure of the confidence with which a point is classified. This information can be used to determine whether to trust the classifier or to resort to some more expensive and accurate method of classification. In order for a confidence estimate to be useful in our classification pipeline, we require that it be accurate, easy to estimate, and extendable to the multiclass setting. To this end, we present three methods of

confidence estimation that stem from asymptotic and Bayesian analyses of RLS.

3.2.1 Asymptotic Methods

Asymptotic methods for confidence estimation arise from the observation that the Bayes decision rule for the squared loss results in a quantity that is proportional to the probability of class membership. More formally, suppose that we wish to label every point in some input space $\mathcal{X}$ with label a if it is a positive example and b otherwise, where $a, -b > 0$. Let $p(1|x)$ be the probability that $x \in \mathcal{X}$ is a positive example. We can minimize the expected squared loss

$$R(c) = \int_{\mathcal{X}} p(1 \mid x)(a - c(x))^2 + (1 - p(1| \mid x))(b - c(x))^2 dp(x) \tag{3.34}$$

by minimizing $R(c)$ on each point. Taking derivatives and solving, we see that

$$c(x) = (a - b)p(1 \mid x) + b \tag{3.35}$$

It follows that RLS, endowed with a sufficiently rich kernel[1], will converge to this optimal c as the amount of training data increases. Following our discussion of multiclass RLS invariance to the magnitudes of a and b, we assume that $a = 1$ and $b = -1$. Given an RLS solution f, our estimate that a point x should have a positive label is

$$\hat{p}(1 \mid x) = \frac{f(x) - 1}{2} \tag{3.36}$$

We can easily extend this reasoning to the multiclass setting by letting $\hat{p}^i(1 \mid x)$ be the confidence estimate obtained from the i^{th} one vs. all classifier. This probability estimate corresponds to the likelihood that x belongs to class i and no others. As such, our first confidence estimation technique directly uses the soft classification of each one vs. all classifier to determine the certainty of the classification.

The next method improves upon our initial estimate by directly focusing on multiclass classification. In particular, one vs. all schemes employ the maximum decision

[1]The RBF kernel has this property.

rule so that the winning classifier, f^{α}, is more clearly identified as the difference between it and the second highest scoring classifier, f^{β}, increases. This intuition defines our second confidence estimation technique, which we aptly name the "gap" confidence, as

$$\hat{p}^{\alpha}(1 \mid x) - \hat{p}^{\beta}(1 \mid x) \tag{3.37}$$

3.2.2 Bayesian Method

While asymptotic methods ultimately *converge* to the probability of class membership, there is no guarantee that this estimate is a real probability between 0 and 1 when obtained from a finite data set. Such a normalized probability estimate can be achieved by treating RLS as a Gaussian process. In order to define this view, consider the usual binary classification setting in which we are given a labeled dataset X, y as well as regularization parameter λ and kernel k that we use to train an RLS classifier f^{RLS}. Furthermore, suppose that we wish to predict the label of a new point $x \in \Re^d$. Before training, we can interpret the joint distribution of the labels of X and x as a multivariate Gaussian with zero mean and covariance matrix

$$\Sigma = \begin{bmatrix} k(X,X) + \lambda I & k(X,x) \\ k(x,X) & k(x,x) \end{bmatrix} \tag{3.38}$$

Hence, $\begin{bmatrix} y \\ y_x \end{bmatrix}$ is drawn from $\mathcal{N}(0, \Sigma)$ training is tantamount to asking for the distribution of $y_x \mid X, y, x,$ once we are given the labels of the training set. Standard results for the conditional distribution of a multivariate Gaussian [19] lead to $y_x \mid X, y, x$ $\mathcal{N}(\mu_x, \sigma_x^2)$ where

$$\mu_x = f^{RLS}(x) \tag{3.39}$$

$$\sigma_x^2 = K(x,x) + \lambda - K(x,X)(k(X,X) + \lambda I)^{-1} K(X,x) \tag{3.40}$$

Since a Gaussian distribution's mean is also its mode, using the RLS solution is equivalent to taking the maximum a posteriori estimate of y_x. We can apply these results to classification by noting that x is classified as positive if $y_x \geq 0$ so that an estimate of the likelihood that x belongs to the positive class is given by

$$p(y_x \geq 0 \mid X, y, x) = \int_0^\infty p(y_x \mid X, y, x) dy_x = 1 - \Phi\left(\frac{-\mu_x}{\sigma_x}\right) \tag{3.41}$$

Here Φ is the cumulative density function of the normal distribution which we used by normalizing our threshold of zero with the mean and variance of y_x. These probability estimates can be extended to the multiclass setting with T classes by realizing that the totality of outputs from every one vs. all classifier defines its own multivariate Gaussian distribution with mean and covariance

$$m_x = \begin{bmatrix} f^1(x) \\ \vdots \\ f^T(x) \end{bmatrix} \Sigma_x = \begin{bmatrix} (\sigma_x^1)^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & (\sigma_x^T)^2 \end{bmatrix} \tag{3.42}$$

The diagonal variances $(\sigma_x^i)^2$ are differentiated for each class because they may differ if a weighting scheme is employed. Noting that Σ_x is diagonal, the likelihood that classifier i is the highest scoring is given by

$$p(y_x^i > y_x^j, \forall j \neq i \mid X, y, x) = \int_{-\infty}^\infty p(y_x^i \mid X, y, x) \left(\prod_{j=1, j\neq i}^{T} \int_{-\infty}^{y_x^i} p(y_x^j \mid X, y, x) dy_x^j \right) dy_x^i \tag{3.43}$$

Thus, equation (3.43) defines our Bayesian estimate of the multiclass confidence.

We conclude our study of multiclass classification confidence by noting that the asymptotic confidence estimation methods rely solely on the output of each classifier and therefore have the same complexity as prediction, namely $O(ndT)$. However, the Bayesian method's reliance on a variance estimate requires the calculation of σ_x^2 which takes $O(ndT + n^2T)$ time for all classes. Furthermore, the calculation of the

Bayesian confidence is nontrivial because it requires numerical integration. We now turn to an active learning extension of RLS that incorporates our asymptotic confidence estimates and weighting results. This final result completes the classification pipeline by addressing the need to minimize labeled data.

3.3 Active RLS

Thus far, we have concentrated on the training and classification stages of our classification pipeline by adapting RLS to overcome imbalanced data and to output confidence scores. Our final result caters to the data collection phase of our classification system by presenting an active learning extension of RLS that minimizes the number of training samples we need to label. We motivate the algorithm as a greedy risk minimization procedure in the first section and then show how to adapt this risk to imbalanced data with weights. Our last section focuses on an efficient implementation of ARLS that uses the Matrix Inversion Lemma.

3.3.1 Algorithm

Consider a variant of the usual classification problem in which we are given a pool of n *unlabeled* training points X with each $x_i \in \Re^d$ for $i = 1, \ldots, n$. Instead of corresponding labels, we are given access to a labeler $L : X \to \{-1, 1\}$ that returns the label of any point in our pool. Our goal is to train a classifier with good generalization performance using as few calls to L as possible. This pool-based active learning problem complicates the usual classification paradigm in that we must choose which points to select and have to decide when to stop asking for labels.

We address the active learning problem by greedily minimizing a risk as presented in [29]. Our discussion of this risk is identical to the original paper, except that we replace their classification scheme with RLS. The Active RLS (ARLS) algorithm builds upon the authors' work by using a regularized classifier that should give good generalization properties. At each iteration t, ARLS partitions X into disjoint sets X_U and X_L of unlabelled and labeled points, respectively. It starts with $X_U = X$ and

ends after a sufficiently low risk has been attained or, in the worst case, after every point has been queried so that $X_L = X$. To set the stage for our risk, let f^L be the classifier we obtain from training RLS on X_L with labels given by y_L. Furthermore, define $f^{L+(x,y)}$ to be the resulting classifier after training on $X_L \cup \{x\}$ when $x \in X_U$ has label $y \in \{\pm 1\}$. The empirical Bayes classification risk of $f^{L+(x,y)}$ is given by

$$\mathfrak{R}(f^{L+(x,y)}) = \sum_{z \in X} \sum_{l \in \{-1,1\}} p(l \mid z) \delta(l, sign[f^{L+(x,y)}(z)]) \tag{3.44}$$

where $\delta(a,b) = 0$ if $a = b$ and 1 otherwise. In an ideal world, ARLS would select the next point to query at each step as the minimizer of the expected risk

$$\begin{aligned} x^* &= \min_{x \in X_U} \mathbb{E}_y[\mathfrak{R}(f^{L+(x,y)}] \\ &= \min_{x \in X_U} p(1 \mid x)\mathfrak{R}(f^{L+(x,1)}) + p(-1 \mid x)\mathfrak{R}(f^{L+(x,-1)}) \end{aligned} \tag{3.45}$$

However, since we do not have access to $p(y \mid x)$, we must rely on the probability estimates given by f^L. In particular, we use the asymptotic estimate

$$\hat{p}^L(1 \mid x) = \frac{f^L(x) - 1}{2} \tag{3.46}$$

as the probability estimate. The estimated Bayes classification risk is thus given by

$$\hat{\mathfrak{R}}(f^{L+(x,y)}) = \sum_{z \in X} \sum_{l \in \{-1,1\}} \hat{p}^{L+(x,y)}(l \mid z) \delta(l, sign[f^{L+(x,y)}(z)]) \tag{3.47}$$

and

$$\mathbb{E}_Y[\hat{\mathfrak{R}}(f^{L+(x,y)})] = \hat{p}^L(1 \mid x)\mathfrak{R}(f^{L+(x,1)}) = \hat{p}^L(-1 \mid x)\mathfrak{R}(f^{L+(x,-1)}) \tag{3.48}$$

It is worthwhile to note that we use the asymptotic estimate over the Bayesian one even though $\hat{p}^L$ may not be a true probability. This preference stems from our expe-

rience with the algorithm. With a correct kernel and regularization parameter, the performance difference between both methods is negligible. However, the Bayesian method is significantly more sensitive to imperfect kernel and regularization parameter choices. When faced with such a scenario, its performance quickly degrades to requiring labels from every point in X. Moreover, $\hat{p}^L$ tends to remain between 0 and 1 even though it is not constrained to do so.

Another key observation comes from the fact that replacing p with its estimate $\hat{p}^L$ leads to a selection scheme that changes its focus as $\hat{p}^L$ increases in accuracy. In particular, when ARLS has too few data points to accurately guess the distribution of labels, it resorts to selecting points that give maximal information about the clusters of X. These clusters are initially chosen at random, but as $\hat{p}^L$ improves, ARLS directly chooses points that improve its classification accuracy. This selection scheme makes sense because in the absence of any information, ARLS does not worry about the particular labels of any points and it simply chooses ones that will many other points. However, once it has some idea of the division of classes, it focuses on refining its estimate of the decision boundary.

It turns out that ARLS can also be adapted to imbalanced data, although special care must be taken in estimating our risk. To this end, the next section extends our risk estimate to be compatible with the bias introduced from weighting RLS.

3.3.2 Active Learning with Imbalanced Data

In keeping with our goal of building a classification system that can handle the vagaries of real world data, it is necessary to consider ARLS under imbalanced data. Experimental evidence suggests that the risk estimate that guides our classifier's selection of data points is inherently resilient to such imbalances. However, the risk is sensitive to weighting schemes in that using probability estimates from f^L when the kernel matrix is weighted biases ARLS to make worse choices than if it had chosen points randomly. To make matters worse, weighting is unavoidable in certain heavily imbalanced datasets, irrespective of how points are selected.

We can reconcile the use of weights with our risk estimate by inversely weighting

each component of $\mathfrak{R}$. In particular, suppose that we are training the ARLS classifier using weights w_y. Removing bias from our risk corresponds to calculating

$$\hat{\mathfrak{R}}^w(f^{L+(x,y)}) = \sum_{z \in X} \sum_{l \in \{-1,1\}} \frac{\hat{p}^{L+(x,y)}(l \mid z)}{w_l} \delta(l, sign[f^{L+(x,y)}(z)]) \tag{3.49}$$

$$\mathbb{E}_y^w[\hat{\mathfrak{R}}^w(f^{L+(x,y)})] = \frac{\hat{p}^L(1 \mid x)}{w_1} \hat{\mathfrak{R}}^w(f^{L+(x,1)}) + \frac{\hat{p}^L(-1 \mid x)}{w_{-1}} \hat{\mathfrak{R}}^w(f^{L+(x,-1)}) \tag{3.50}$$

These modified risk estimates allow us to use weighted RLS as the classifier in ARLS. Moreover, the new risk estimates successfully mitigate the effects of weighting and thereby allow ARLS to be used as an effective component in our classification pipeline. The next section discusses how to implement ARLS in an efficient manner that allows for online updates and fast minimization of $\mathbb{E}_y^w[\hat{\mathfrak{R}}^w(f^{L+(x,y)})]$.

3.3.3 Fast Estimation and Updates

Our implementation of ARLS combines careful bookkeeping with the Matrix Inversion Lemma to demonstrate a computationally tractable algorithm that has the same worst case complexity as RLS. While a naive approach to estimating $\hat{\mathfrak{R}}^w$ and updating f^L requires computing $O(n^2)$ RLS solutions, we can reuse our work done in finding f^L to achieve a quadratic speedup.

To this end, suppose that we are using ARLS with kernel k, regularization parameter λ, and weight w_y for positive and negative classes. Here w_y be used to weight the kernel matrix, although we prefer to divide λ because it leads to a more elegant solution. Let W^L be the diagonal matrix with w_y^{-1} on its main diagonal for every labeled point. We require an initial computation of the kernel matrix $K = k(X, X)$, maintenance of a covariance matrix

$$V^L = k(X, X_L)(k(X_L, X_L) + \lambda W^L)^{-1} k(X_L, X) = K_L^T (K_{LL} + \lambda W^L)^{-1} K_L \tag{3.51}$$

and the latest predicted values of X given by

$$m^L = f^L(X) = K_L^T(K_{LL} + \lambda W^L)^{-1} y_L \tag{3.52}$$

In order to update m^L to $m^{L+(x_i,y_i)}$ quickly, we appeal to Theorem A.4.1 from the appendix to get that

$$m^{L+(x_i,y_i)} = m^L + \frac{m_i^L - y_i}{K_{ii} + \lambda w_{yi}^{-1} - V_{ii}^L}(V_i^L - K_i)^T \tag{3.53}$$

Furthermore, an application of Corollary A.4.1 shows that the coefficients of f^L are given by

$$c^L = (W^L)^{-1}\frac{y_L - m_L^L}{\lambda} \tag{3.54}$$

Equation (3.53) can be used to update m^L and to estimate $\mathbb{E}_y^w[\hat{\mathfrak{R}}^w(f^{L+(x,y)})]$ in $O(n)$ time. It remains to show how V^L can be updated to $V^{L+(x_i,y_i)}$ once we include x_i in our labeled set. Theorem A.4.1 applies to each column of $V_{:,j}^L$ leading to

$$V_{:,j}^{L+(x_i,y_i)} = V_{:,j}^L + \frac{V_{i,j}^L - K_{ij}}{K_{ii} + \lambda w_{y_i}^{-1} - V_{ii}^L}(V_i^L - K_i)^T \tag{3.55}$$

It is clear that finding the minimizer of $\mathbb{E}_y^w[\hat{\mathfrak{R}}^w(f^{L+(x,y)})]$ requires only $O(n^2)$ computations. Furthermore, once we select the next point to query and train on, the brunt of the calculation is spent in updating V^L which takes $O(n^2)$. The overall runtime of ARLS is thus given by $O(tn^2)$, which can be significantly less than that of RLS if a fraction of X is used to achieve reliable predictions. Even in the worst case when active learning is of no use, our runtime collapses to $O(n^3)$, which is identical to that of computing the RLS solution over X.

Chapter 4

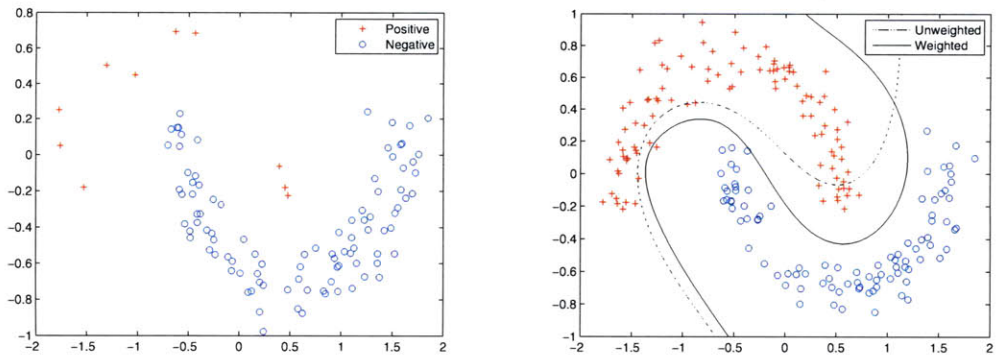
Experiments

The Experiments chapter presents synthetic and real data experiments with our findings from the Algorithms and Analysis chapter. Each set of experiments begins with synthetic data that provides a controlled environment with which to demonstrate important properties. We then subject our methods to real world data so as to establish their validity in the "wild".

The organization of this chapter follows that of the previous one in that section 4.1 demonstrates the positive effects of weighting RLS on imbalanced data set. Section 4.2 extends the real world experiments of the previous section to show the performance of our three confidence estimation schemes. Finally, section 4.3 provides an in depth discussion of the behavior of ARLS on balanced and imbalanced data sets.

4.1 Weighted RLS

Our first set of experiments demonstrates the effects of weighting on RLS with synthetic and real data. In particular, Figure 4-1(a) shows an imbalanced training set from the "Two Moons" dataset when there are 10 times fewer positive training examples than negative ones. The classifiers we obtain from using no weights and equilibrating the two classes are plotted against a test set which contains equal numbers of examples from both classes in Figure 4-1(b). We use a Gaussian kernel with width parameter 2 and automatically select the regularization parameter from the



(a) Training Set

(b) Test Set

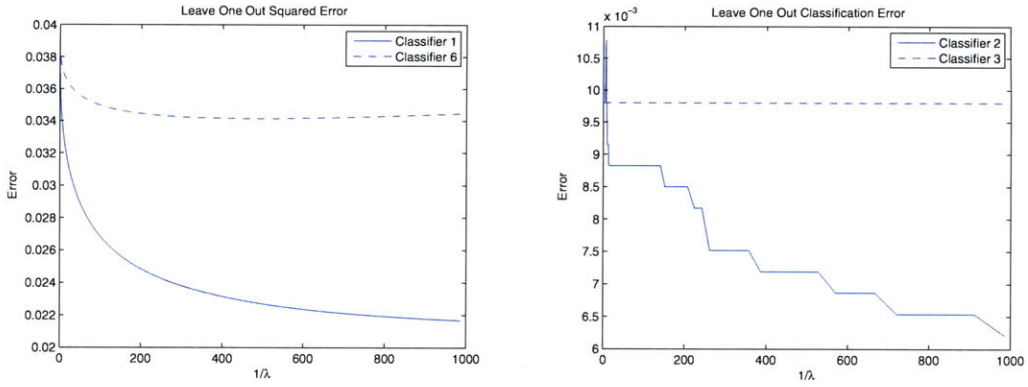
Figure 4-1: Left: The Two Moons dataset with an imbalanced training set. Right: Decision boundaries of weighted and unweighted RLS solutions obtained from the training set and plotted against the test set.

LOO classification error.

While the training set suggests a separable boundary, the class imbalance leads unweighted RLS to shrink its estimate of the volume of the positive class. Indeed the data imbalance is severe enough that the decision boundary does not even classify the training set correctly. Weighting successfully corrects this situation and decreases the unweighted classifier's error from 7.5% to 0.5% on the test set. That the weighted classifier does not completely separate the test set is due to the paucity of positive examples from which to refine its decision boundary.

Next, we apply our multiclass weighting scheme to the Caltech 101 data set using features extracted from the CBCL hierarchical model as described in [25] [17] [16]. We use 30 training examples per category for a total of 3060 training examples in a one vs. all multiclass regime. It is important to note that the CBCL model outputs 8075 features per training vector so that it is easy to over-fit the training data. It is essential to use a heavily regularized linear kernel - anything more powerful will immediately fit the data and generalize poorly.

Our one. vs. all scheme creates a large data imbalance in which less than 1% of the training examples are positive. Figure 4-2 demonstrates the LOO squared and classification error for various regularization parameters on our training set. While



(a) LOO Squared Error

(b) LOO Classification Error

Figure 4-2: Left: Examples of malformed LOO squared error curves from training on the Caltech 101 dataset. Right: Sample LOO classification error curves from the same dataset and trials.

these plots are taken from specific classes, they actually represent the overwhelming majority of LOO error curves obtained from each of our 102 classifiers. It is clear that the data imbalance makes it impossible to use either error estimate to determine a proper regularization value. In the case of squared error, the error is either a bottomless curve that suggests no regularization or has a minimum that recommends a regularization parameter that surely over-fits. The classification error curves are also bottomless and in some cases are completely flat so that no amount of regularization will help performance. This latter situation is particularly troublesome because it indicates that the classifier will always ignore its positive examples.

Our results demonstrate the phenomenon that heavily imbalanced training data favors over-fitting. The classifier is heavily biased towards classifying everything as negative so that less regularization increases its chances of labeling an example as positive because of over-fitting. However, such improvements should be taken with caution because they are unlikely to generalize well. The classifier is faced with a hard situation in which any amount of regularization forces it to mislabel most positive examples. Conversely, a lack of regularization allows it to fit its training data correctly, but is unlikely to lead to a good decision boundary because there are too few positive examples.

Next, we demonstrate the effects of equilibrating the loss of positive and negative examples in Figure 4-3. We present a histogram of the logarithm of the automatically selected regularization parameter for each one vs. all classifier, with and without reweighting, from the Caltech training set. The "optimal" regularization parameter is selected as the largest parameter that minimizes either the LOO squared or classification error. It is clear that the unweighted case selects hopelessly low regularization values - we were forced to use a logarithmic scale because these regularization parameters are always counted as zero otherwise. However, weighting shows preference towards some regularization and picks values between 0.4 and 0.01.

Finally, we present accuracy and training time results of three different trials on the Caltech 101 using a test set of over 6000 points in Table **??**. Each trial uses both, weighted and unweighted RLS, with regularization parameters selected by our automatic LOO error method. We also use a seemingly optimal regularization value of 0.4 which is obtained by selecting the highest regularization value automatically selected among any of the 102 classifiers. The weighted RLS solutions are computed using the Cholesky decomposition of the weighted kernel matrix for each one vs. all classifier while the unweighted trials perform a single eigenvalue decomposition. These results are compared against the performance of a tuned SVM which is trained using LibSVM.

Our results indicate that the automatic methods tend to select overly low regularization parameters. Indeed training all of the classifiers with $\lambda = 0.4$ gives a significant performance boost that exceeds the performance of the SVM. The performance of automatically regularized unweighted RLS is even worse because it opts for effectively no regularization. It interesting to see that unweighted RLS performs comparatively well when used with our optimal regularization value. Nonetheless, weighting gives an improvement across all three trials and, more importantly, it saves our automatic λ selection method. Finally, it is worth noting that training the RLS classifier is over 70 times faster than training the SVM. If we are willing to forego weighting, we can get away with computing a single eigenvalue decomposition that is reused across all one vs. all classifiers, even if they differ in their regularization

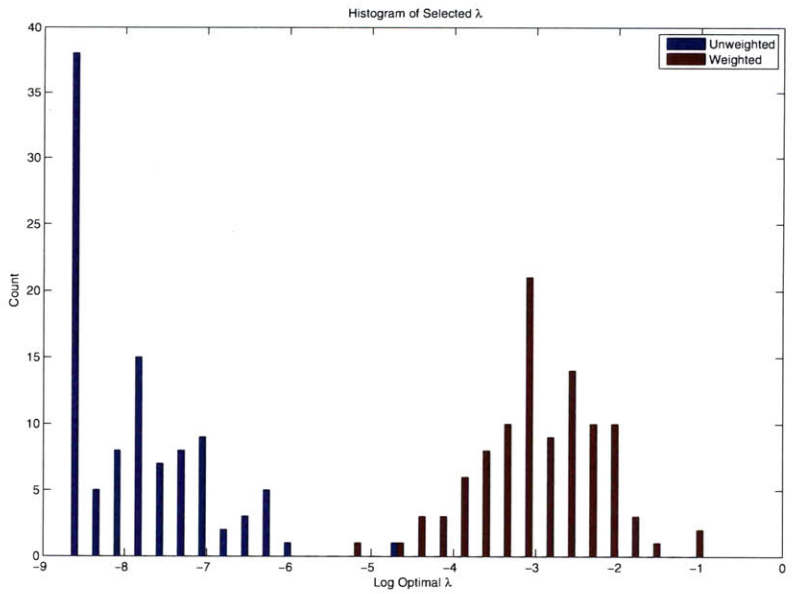


Figure 4-3: A histogram of the logarithms of regularization parameters automatically selected from the Caltech 101 dataset for each one vs. all classifier. Blue corresponds to parameters selected from unweighted RLS while red corresponds to weighted RLS.

parameter. These increases in performance and training speed give credence to using RLS in a classification setting.

All in all, weighting is an effective countermeasure to imbalanced data that it is easy to implement and works reliably. This corrective strategy not only enhance a classifier's performance on misrepresentative training sets, but it also saves our automatic parameter tuning. The only caveat of this method is that it requires separate processing of each kernel matrix when used for one vs. all classification.

4.2 Confidence

We continue our experiments with the Caltech 101 dataset to compare our methods for estimating confidence. All confidence estimates are derived from weighted RLS with $\lambda = .4$. In an ideal setting, confidence would range between 0 and 1 and correspond exactly to the probability of success. At the bare minimum, we require that accuracy be a monotonically increasing function of confidence which we can then transform

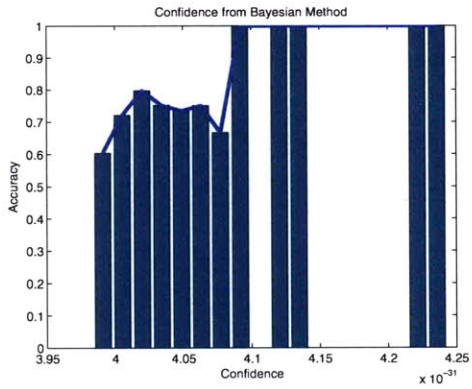


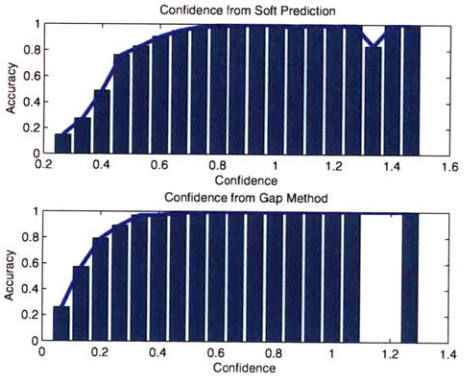
Figure 4-4: A histogram of accuracy as a function of the Bayes' confidence score. The value of each bin is equal to the accuracy attained on all points whose confidence falls in the bin.

into a probability. Indeed, this is the approach taken to estimate confidence from SVM's [20].

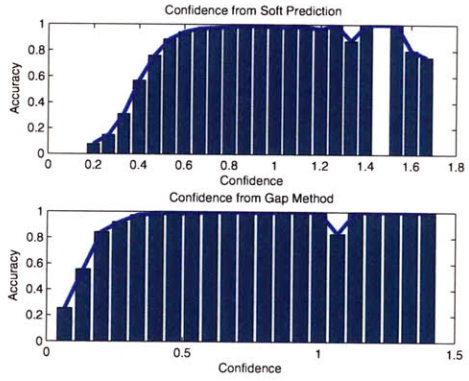
Figure 4-4 shows a histogram of the accuracy attained on all test samples that fall within a specific confidence bin using our Bayesian method. Sadly, the accuracy of the confidence estimates fluctuates wildly and is therefore unusable. This fluctuation is likely due to sensitivity to kernel choice and regularization parameter. Indeed, we discounted using the Bayesian probability estimate for ARLS partly because it was not resilient to imperfect parameter choices. A second contributor to this method's poor performance is that even a minor imperfection in the probability estimate is magnified 100 fold because of the large number of classes. All in all, the Bayesian method's poor performance and hefty processing requirements make it a poor choice for confidence estimation.

We thus turn to our asymptotic methods for more reliable confidence estimates, which are given in Figure 4-5. The histogram bins are obtained using the three Caltech 101 trials discussed in our weighted RLS experiments. It is worthwhile to note that data gaps as in bin 1.2 in the gap method on trial 1 occur because no points attain that confidence, not because of poor accuracy. Both methods achieve our desired monotonicity, although the gap method does not suffer from the soft classification's decrease in accuracy on trial 2. It is clear that the asymptotic methods

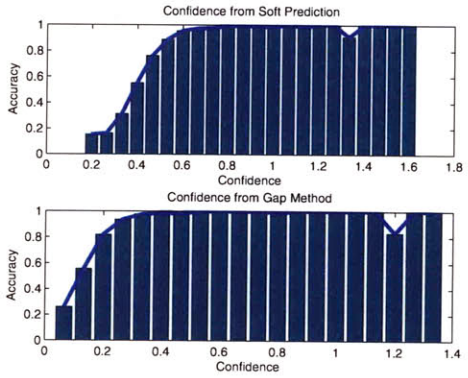
are desirable over our Bayesian method because they achieve better monotonicity and use quadratically less processing time.



(a) Confidence Trial 1



(b) Confidence Trial 2



(c) Confidence Trial 3

Figure 4-5: A histogram of accuracy as a function of the asymptotic confidence score. The value of each bin is equal to the accuracy attained on all points whose confidence falls in the bin.

We conclude our confidence experiments with the observation that using more information and processing power as in the case of our Bayesian confidence estimate is not necessarily better. While one would expect this method to be superior over our simple asymptotic methods, it turns out to be overly sensitive to RLS' probability model. Thus, using the soft RLS classification directly or taking the gap difference works well because it is computationally trivial and accurate.

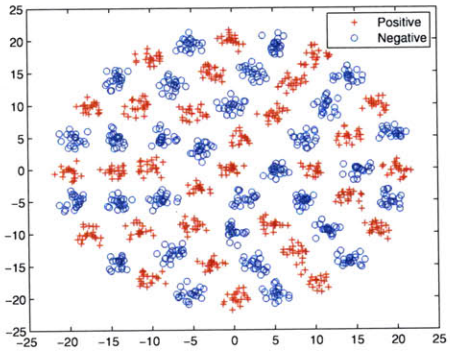


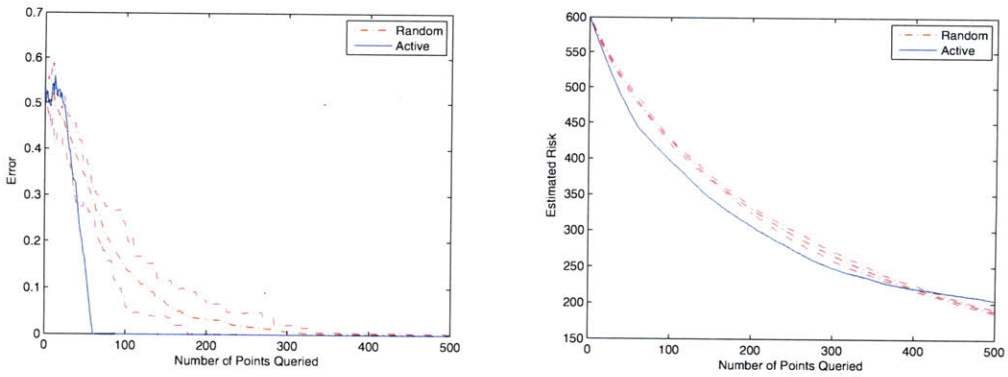
Figure 4-6: The Spheres dataset

4.3 Active Learning

Our final set of experiments investigates the performance of ARLS on balanced and imbalanced data sets. Active learning tests RLS' classification and confidence estimation performance by using the latter to inform point selection and improve the former. We compare ARLS against an online version of RLS that selects points at random. This passive selection scheme matches the usual paradigm of data collection for classification in that points are not distinguished a priori so that good generalization relies on a sizeable training set.

4.3.1 Balanced Data

Figure 4-6 shows the "Spheres" dataset which consists of 1200 points that are generated by selecting 20 points from each of 60 uniformly distributed and equally spaced spheres. The corresponding Figure 4-7 plots the estimated risk and validation error on a validation set of 1200 similarly distributed points as a function of the number of queried points. We compare the performance of active selection to randomly selecting points using a Gaussian kernel with width 1.7 and regularization parameter 0.5. The error bars around the random method's error indicate one standard deviation; ARLS always selects the same points so it has no variation. Our active learning scheme achieves 100% accuracy on the validation set after querying exactly 60 points. It turns out that ARLS correctly clusters the points into their respective spheres and



(a) Validation Error

(b) Estimated Risk

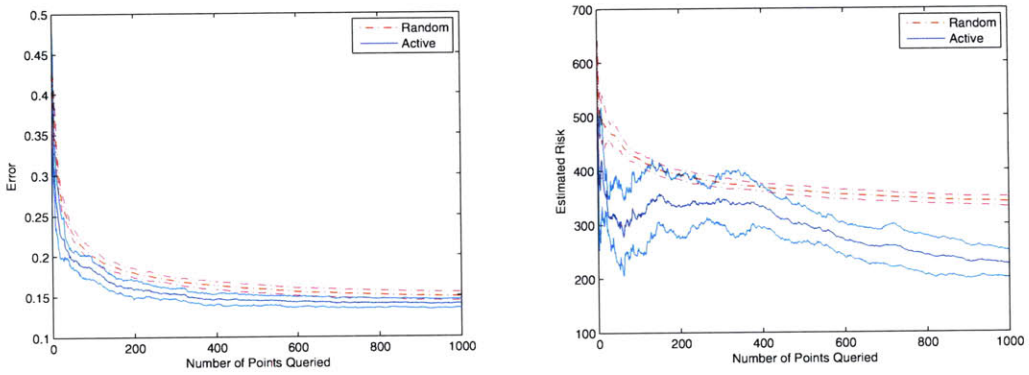
Figure 4-7: Estimated risk and validation error of active and passive selection on the Spheres dataset. Error bars indicate 1 standard deviation.

queries the center of each cluster once. This example depicts the behavior of ARLS' risk estimate well. The active selection scheme chooses the center of each sphere because it minimizes a global risk over all points - this is different than always selecting the most uncertain point. Once ARLS knows the label of the center of a sphere, it temporarily assumes that the label applies to all nearby points and moves on to discover the label of a different cluster.

Next, we investigate ARLS' performance on a real data set by tasking it with distinguishing high energy gamma particles from background noise in the MAGIC Gamma Telescope dataset. Each of 10 trials uses 1% of the 19,000 10-dimensional training vectors in the training set and estimates error on a separate validation set consisting of 2000 vectors. We use a Gaussian kernel with width 3 and regularization parameter 0.2.

4.3.2 Imbalanced Data

Similarly to Figure 4-7, Figure 4-8 shows the estimated risk and validation set error as a function of the number of points queried. Error bars around the error and risk indicate one standard deviation. We see that ARLS consistently achieves better performance than random selection. On average, it takes ARLS approximately 350



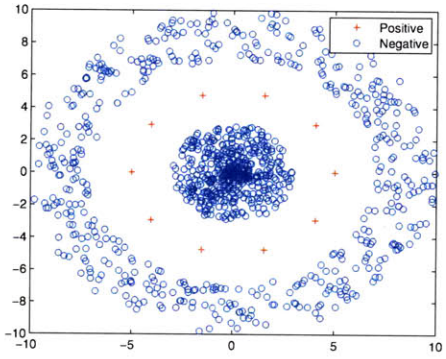
(a) Validation Error

(b) Estimated Risk

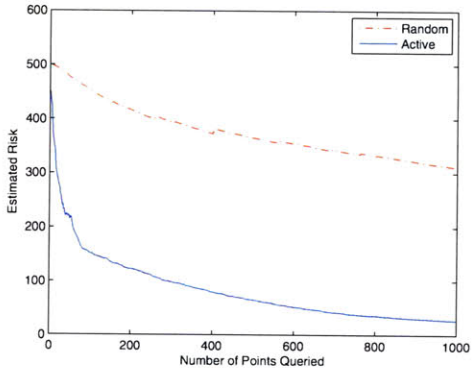
Figure 4-8: Estimated risk and validation error of active and passive selection on the MAGIC dataset. Error bars indicate 1 standard deviation.

points to achieve the accuracy RLS gets with 1000 points. Thus, ARLS achieves its objective of using fewer training points than passive selection would allow. However, its risk estimate does not convey the marginal returns of using more than 350 points. Instead, the risk decreases even more sharply after this "optimal" point. This effect is undesirable because it suggests that the estimated risk is not a good indicator of when to stop querying.

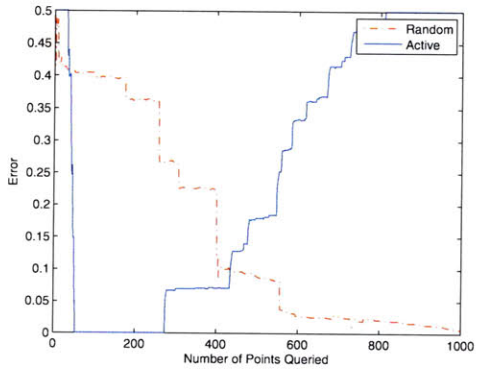
We begin our exploration of ARLS' performance on imbalanced data with a synthetic benchmark that shows the resilience of our risk to data imbalances. The "Circles" dataset, shown in Figure 4-9(a) simulates a heavy data imbalance in which there are 100 times more negative examples than positive ones. Performance is measured on a separate validation set with equal numbers of positive and negative examples so that a completely nave classifier will never attain good performance. We use a Gaussian kernel with a width of 1.7 and regularization parameter of 0.2. The validation error of unweighted ARLS is compared against randomly selection with equilibrating weights in Figure 4-9. We see that ARLS' point selection scheme is impervious to the imbalance because it achieves perfect classification after querying approximately 50 data points. However, its error rate increases after 270 data points because its training set becomes saturated with negative examples. This effect demonstrates the paradoxical result that, in the absence of weights, ARLS can train on too much data



(a) Circles Dataset



(b) Estimated Risk



(c) Validation Error

Figure 4-9: Top Left: The imbalanced Circles dataset. Top Right: The estimated risk of unweighted ARLS and weighted random selection. Bottom: The validation error of unweighted ARLS and weighted random selection. The validation set has equal proportions of positive and negative samples.

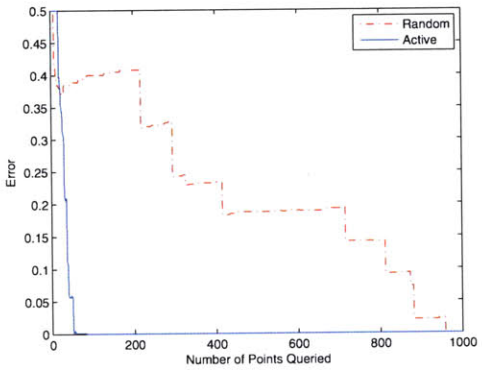


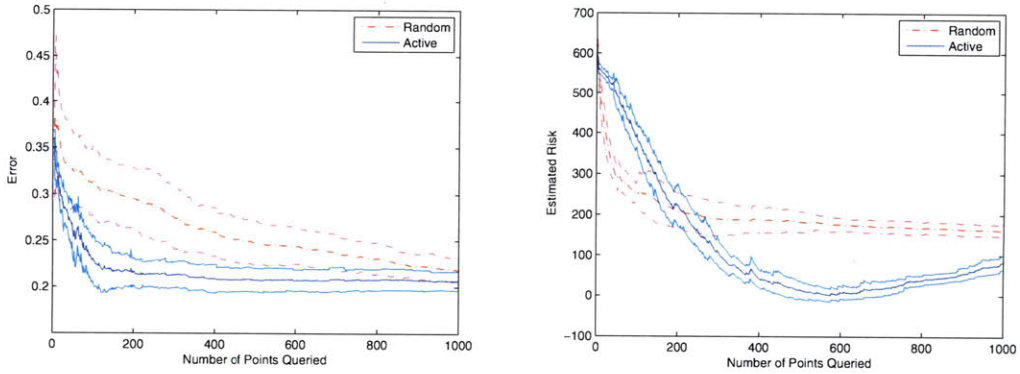
Figure 4-10: The validation error of weighted, unbiased ARLS on the Circles dataset. The validation set has equal proportions of positive and negative samples.

so that early stopping is essential. In the limit where ARLS uses the entire training set, it is identical to unweighted RLS and therefore is expected to misclassify positive test samples.

Weighted ARLS using a bias corrected risk allows us to combine the intelligence of ARLS with the corrective weighting scheme of the random method so as to find relevant points quickly and then remain at a low error. The validation error of this method on the Circles dataset is shown in Figure 4-10. It reaches near perfect performance after 50 queries and maintains a zero error rate thereafter. It is worth noting that this corrected method prefers tighter parameters in that we use a Gaussian with width 1 and regularization parameter 0.1.

Finally, we apply weighted, bias corrected ARLS to the MAGIC data set when we use 10 times more examples of background noise than gamma signals and a total of 1300 training points. Similarly to our synthetic trials, the imbalanced data favors tighter parameters so that we use a Gaussian width of 1.5 and regularization parameter of 0.1. We conduct 10 trials that estimate error on a separate validation set with the original proportions of examples. As usual, Figure 4-11 demonstrates the estimated risk and validation error as a function of points queried. Error bars on the active and random selection curves denote one standard deviation.

Active selection outperforms random selection by an even wider margin than in



(a) Validation Error

(b) Estimated Risk

Figure 4-11: Left: The validation error of weighted, unbiased ARLS when trained on an imbalanced MAGIC dataset. The validation set is balanced and error bars denote 1 standard deviation. Right: The estimated risk on the training set.

the balanced case. The former manages to reach near optimal performance after approximately 200 queries, which the latter attains only after querying nearly all of the training data. It is also interesting to note that the estimated risk reaches a minimum at around 600 queries. While the risk still overestimates the number of points that should be queried, it does not mislead the user into querying all of them as in the balanced case. Moreover, the minimum of the risk curve approximately corresponds to the number of queries where the error stops improving even marginally.

In conclusion, ARLS is useful in reducing the number of labeled points needed to train a classifier. The algorithm can be adapted to use weights and is therefore applicable to imbalanced data. Its selection scheme gives it a significant advantage over random selection under such adverse conditions, making it particularly applicable to one vs. all multiclass regimes. Sadly, the algorithm's risk estimate does not appear to be a reliable indicator of when to stop querying, so alternative methods should be used to decide this.

	Trial 1		Trial 2		Trial 3	
	Accuray	Training Time	Accuracy	Training Time	Accuracy	Training Time
WRLS $\lambda = 0.4$	64.73	0:01:59	63.05	0:02:00	63.59	0:01:58
RLS $\lambda = 0.4$	62.46	0:00:22	61.84	0:00:20	62.26	0:00:20
SVM	62.67	2:27:48	61.75	2:35:50	62.45	2:34:20
WRLS $\lambda = Auto$	60.75	-	59.45	-	59.20	-
RLS $\lambda = Auto$	54.41	-	52.28	-	52.08	-

Table 4.1: Accuracy and processing time of weighted RLS (WRLS) and unweighted RLS on Caltech 101 using different regularization settings.

Chapter 5

Conclusion

We conclude by summarizing our findings and relating them to the three stages of the classification pipeline:

1. An effective classification system should work to minimize the amount of labeled training data it needs. We address this requirement by deriving an active learning extension of RLS that queries the user for the labels of particularly relevant points. The algorithm greedily minimizes a risk estimate of its unlabelled training data and incorporates new labels in an efficient, online fashion. We demonstrate that our algorithm is particularly adept to handling imbalanced data because of its intelligent selection scheme.

2. Real world data is multiclass and inherently noisy and imbalanced. A viable classifier must handle these issues but also be fast and easy to train. We derive a weighted version of RLS that successfully counters imbalanced data and affords a fast way to estimate the LOO error. This result allows for automatic tuning of the classifier's regularization parameter. Weighted RLS' performance and training time in a one vs. all multiclass regime on the Caltech 101 dataset makes it an excellent candidate for our classification system.

3. Finally, it is necessary to estimate an accurate confidence score along with a classifier's prediction. We demonstrate two fast methods for doing this that

work well in multiclass regimes. Indeed, RLS' soft predications ultimately converge to the probability of class membership and are good confidence scores by themselves.

Appendix A

Appendix

A.1 Equivalence of Reweighting and Point Duplication

We assume the usual RLS setting in which we are given a training set consisting of n points and their corresponding labels X, y. Suppose that we try to fix a data imbalance by double counting some points in the RLS solution, so that we deal with $n_{eff} > n$ points. The RLS objective becomes

$$\min_{f \in \mathcal{H}_K} \frac{1}{2n_{eff}} \sum_{i=1}^{n_{eff}} (f(x_i) - y_i)^2 + \frac{\lambda}{2} ||f||_K^2 \tag{A.1}$$

The Representer Theorem allows us to express the RLS solution as a linear combination of kernels that reuse any duplicated points from the training set:

$$f(\cdot) = \sum_{i=1}^{n_{eff}} c_i k(x_i, \cdot) \tag{A.2}$$

In addition, the RKHS norm is given by

$$||f||_K^2 = \sum_{i=1}^{n_{eff}} \sum_{j=1}^{n_{eff}} c_i c_j k(x_i, x_j) \tag{A.3}$$

Note that the above indicates that if $x_i = x_j$, $i \neq j$, then $c_i = c_j$. To see this, fix

some point x_j for which we have $t > 1$ copies with indices $H = \{i \mid x_i = x_j\}$. The Representer theorem allows us to represent f as

$$f(\cdot) = \sum_{i=1, i \notin H}^{n_{eff}} c_i k(x_i, \cdot) + \sum_{i \in H} c_i k(x_j, \cdot) = \sum_{i=1, i \notin H}^{n_{eff}} c_i k(x_i, \cdot) + k(x_j, \cdot) \sum_{i \in H} c_i \tag{A.4}$$

Treating H as an index vector, we see that for any $c_H \in \mathcal{R}^t$ such that $\sum_{i \in H} c_i = d_j$, f remains the same function. However, the norm of f depends quadratically on c_H so it follows that the optimal f has all of the components of c_H equal to each other.

Using this property, our norm collapses to

$$||f||_K^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} n_i c_i n_j c_j k(x_i, x_j) = (Nc)^T K (Nc) \tag{A.5}$$

where N is a diagonal matriz and $N_{ii} \in \mathbb{Z}^+$ is the number of copies of x_i we wish to use. Our loss becomes

$$\begin{aligned}
\sum_{i=1}^{n_{eff}} (f(x_i) - y_i)^2 &= \sum_{i=1}^{n_{eff}} \left(\sum_{j=1}^{n_{eff}} c_i k(x_j, x_i) - y_i \right)^2 \\
&= \sum_{i=1}^{n} n_i \left(\sum_{j=1}^{n} n_i c_i k(x_j, x_i) - y_i \right)^2 \\
&= (KNc - y)^T N (KNc - y) = (KNc - y)^T (NKNc - Ny) \\
&= c^T NKNKNc - c^T NKNy - y^T NKNc + y^T Ny
\end{aligned} \tag{A.6}$$

Thus, the objective we wish to minimize is

$$J(c) = \frac{1}{2n_{eff}} (c^T NKNKNc - c^T NKNy - y^T NKNc + y^T Ny) = \frac{\lambda}{2} c^T NKNc \tag{A.7}$$

Taking derivatives and setting equal to 0, we find

$$\frac{\partial J}{\partial c} = \frac{1}{n_{eff}}(NKNKNc - NKNy) + \lambda NKNc = 0$$
$$\rightarrow (KNc - y) + n_{eff}\lambda c = 0$$
$$\rightarrow (KN + n_{eff}\lambda I)c = (K + n_{eff}\lambda N^{-1})Nc$$
$$\rightarrow c = N^{-1}(K + n_{eff}\lambda N^{-1})^{-1}y = N^{-1}(AK + \lambda I)^{-1}Ay \tag{A.8}$$

where $A_{ii} = \frac{n_i}{n_{eff}}$. Next, note that

$$f(x) = \sum_{i=1}^{n_{eff}} c_i k(x_i, x) = \sum_{i=1}^{n} n_i c_i k(x_i, x) = k(x, X)Nc$$
$$= k(x, X)(AK + \lambda I)^{-1}Ay \tag{A.9}$$

This last form matches our results from 3.1.1.2, and hence reweighting and duplicating points are equivalent.

A.2 Weighted RLS and Leave One Out Error Derivation

Suppose that we wish to solve a weighted version of the RLS problem:

$$J(f) = \sum_{i=1}^{n} a_i(y_i - f(x_i))^2 + \lambda||f||_{\mathcal{H}}^2 \tag{A.10}$$

where $a_i > 0$ and $\sum_{i=1}^{n} a_i = 1$. The Representer Theorem allows us to express the minimizer of the above a

$$f(\cdot) = \sum_{j=1}^{n} c_j k(x_j, \cdot) \tag{A.11}$$

So that our minimization problem can be written as

$$J(c) = (y - Kc)^T A(y - Kc) + \lambda c^T Kc \tag{A.12}$$

where A is a positive definite diagonal matrix. Let $W = A^{\frac{1}{2}}$ so that we can define

$$z = Wy \tag{A.13}$$

$$M = WKW \tag{A.14}$$

$$d = W^{-1}c \tag{A.15}$$

Our minimization problem becomes

$$\begin{aligned} J(c) &= (y - Kc)^T W^T W(y - Kc) + \lambda c^T W^{-1} WKWW^{-1} c \\ &= (z - Md)^T (z - Md) + \lambda d^T Md \end{aligned} \tag{A.16}$$

$$J(d) = (z - Md)^T (z - Md) + \lambda d^T Md \tag{A.17}$$

Noting that M must be symmetric and positive semidefinite, it is a valid kernel matrix and we can use the standard results for the RLS solution to obtain that

$$\rightarrow d = (M + \lambda I)^{-1} z = G^{-1} z \tag{A.18}$$

$$G = M + \lambda I \tag{A.19}$$

$$c = WG^{-1} z \tag{A.20}$$

Next, we find a convenient and fast method to find the leave one out crossvalidation error. A few definitions will be useful. Let f_{S^i} be the classifier we get from training on all points except x_i, and let c^i be its associated parameters. Furthermore, define

$$y_j^i = \begin{cases} y_j & i \neq j \\ f_{S^i}(x_i) & i = j \end{cases} \tag{A.21}$$

$$z^i = Wy^i \tag{A.22}$$

As [21] demonstrates, it turns out that f_{S^i} is the same as the function we would get by training on all input points using y^i for labels. To see why, let f_R be the classifier we get by training on all points using labels

$$\mathcal{Y}_j = \begin{cases} y_j & i \neq j \\ f_R(x_i) & i = j \end{cases} \tag{A.23}$$

so that $f_R(x_i) - \mathcal{Y}_i = 0$. Assuming $\lambda > 0$, Corollary A.4.1 shows that the coefficients of f_R are given by

$$c^R = A\frac{\mathcal{Y} - f_R(X)}{\lambda} \tag{A.24}$$

It follows that $c_i^R = 0$ so that x_i does not play a role in either the squared loss or the regularization term. As such, training f_R is equivalent to training over all points but x_i, and we know the minimizer of this problem to be f_{S^i}. Hence, $f_R = f_{S^i}$ and $f_R(X_i) = y_i^i$.

We can use the close resemblance between our original problem and that of finding f_{S^i} to save on computation. Note that

$$f_S(x_i) = (Kc)_i = (KWG^{-1}z)_i = \frac{(MG^{-1}z)_i}{w_i} \tag{A.25}$$

So that the difference between our original and leave one out classifier is

$$
\begin{aligned}
f_{S^i}(x_i) - f_s(x_i) &= \frac{1}{w_i}\sum_{j=1}^{n}(MG^{-1})_{ij}(z_j^i - z_j) \\
&= (MG^{-1})_{ii}(y_i^i - y_i) \\
f_{S^i}(x_i)(1 - (MG^{-1})_{ii}) &= f_s(x_i) - (MG^{-1})_{ii}y_i \\
f_{S^i}(x_i) &= \frac{f_S(x_i) - (MG^{-1})_{ii}y_i}{1 - (MG^{-1})_{ii}} \qquad \text{(A.26)}
\end{aligned}
$$

The unweighted leave one out error is given by

$$
\begin{aligned}
y_i - f_{S^i}(x_i) &= y_i - \frac{f_S(x_i) - (MG^{-1})_{ii}y_i}{1 - (MG^{-1})_{ii}} \\
&= \frac{y_i - (MG^{-1})_{ii}y_i - f_S(x_i) + (MG^{-1})_{ii}y_i}{1 - (MG^{-1})_{ii}} \\
&= \frac{y_i - f_S(x_i)}{1 - (MG^{-1})_{ii}} \qquad \text{(A.27)}
\end{aligned}
$$

At this point a simple lemma will help simplify our calculations.

Lemma A.2.1

$$L - MG^{-1}L = \lambda G^{-1}L \qquad \text{(A.28)}$$

Proof

$$
\begin{aligned}
MG^{-1} &= M(M + \lambda I)^{-1} = (M + \lambda I - \lambda I)(M + \lambda I)^{-1} \\
&= I - \lambda G^{-1} \qquad \text{(A.29)} \\
L - MG^{-1}L &= (I - MG^{-1})L = \lambda G^{-1}L \quad \blacksquare \qquad \text{(A.30)}
\end{aligned}
$$

Using our lemma,

$$
\begin{aligned}
y_i - f_S(x_i) = y_i - \frac{(MG^{-1}z)_i}{w_i} = \frac{(z - MG^{-1}z)_i}{w_i} = \frac{\lambda(G^{-1}z)_i}{w_i} = \frac{d_i}{w_i} \qquad \text{(A.31)} \\
1 - (MG^{-1})_{ii} = (I - MG^{-1})_{ii} = \lambda(G^{-1})_{ii} \qquad \text{(A.32)}
\end{aligned}
$$

Our unweighted leave one error for all points is thus given by

$$L_E = \frac{d}{Wdiag(G^{-1})} = \frac{c}{W^2 diag(G^{-1})} \tag{A.33}$$

Where division is performed element-wise and $diag(G^{-1})$ is a column vector of the diagonal elements of G^{-1}.

In the way of error statistics we may wish to calculate:

1. The weighted leave one out squared error is given by

$$L_{WE}^{SQ} = L_E^T A L_E \tag{A.34}$$

2. The misclassification error requires the leave one out values, which can be computed by

$$L_V = y - L_E \tag{A.35}$$

3. The weighted empirical squared error is given by

$$R^{SQ} = (y - Kc)^T A(y - Kc) = (z - Md)^T (z - Md) \tag{A.36}$$

$$z - Md = z - MG^{-1}z = \lambda G^{-1} z = \lambda d \tag{A.37}$$

$$R^{SQ} = \lambda^2 d^T d = \lambda^2 c^T A^{-1} c \tag{A.38}$$

4. Finally, the empirical misclassification error requires calculating

$$\begin{aligned} Kc &= y - y + Kc = y - W^{-1}(z - MG^{-1}z) = y - \lambda W^{-1} d \\ &= y - \lambda A^{-1} c \end{aligned} \tag{A.39}$$

A.3 Weighted Offset RLS and Leave One Out Error Derivation

Suppose that in addition to weighting we also use an unpenalized offset parameter b. In this case our optimization problem minimizes

$$J(f,b) = \sum_{i=1}^{n} a_i(y_i - f(x_i) - b)^2 + \lambda||f||_{\mathcal{H}}^2 \tag{A.40}$$

The Representer Theorem again allows us to express J as a quadratic matrix problem

$$J(c,b) = (y - Kc - b\bar{1})^T A(y - Kc - b\bar{1}) + \lambda c^T Kc \tag{A.41}$$

Making the usual definitions of $W = A^{\frac{1}{2}}$, $z = Wy$, $M = WKW$, $d = W^{-1}c$ we get

$$\begin{aligned} J(d,b) =& (z - Md - bw)^T(z - Md - bw) + \lambda d^T Md \\ =& z^T z - z^T Md - bz^T w - d^T Mz + d^T MMd \\ &+ bd^T Mw - bw^T z + bw^T Md + b^2 w^T w + \lambda d^T Md \\ =& z^T z + d^T MMd + b^2 w^T w - 2d^T Mz \\ &- 2bw^T z + 2bd^T Mw + \lambda d^T Md \end{aligned} \tag{A.42}$$

Next, we take derivatives and set equal to 0 to solve for b and d

$$\begin{aligned} 0 = \frac{1}{2}\frac{\partial J(d,b)}{\partial d} &= MMd - Mz + bMw + \lambda Md \\ &\rightarrow MMd + \lambda Md = Mz - bMw \\ &\rightarrow (M + \lambda I)d = z - bw \end{aligned} \tag{A.43}$$

The optimal d is thus given by

$$d = (M + \lambda I)^{-1}(z - bw) = G^{-1}(z - bw) \tag{A.44}$$

We can also differentiate to solve for b.

$$\begin{aligned} 0 = \frac{1}{2}\frac{\partial J(d,b)}{\partial b} &= bw^T w - w^T z + d^T M w \\ &= bw^T w - w^T z + w^T M G^{-1}(z - bw) \\ b(w^T w - w^T M G^{-1} w) &= w^T z - w^T M G^{-1} z \end{aligned} \tag{A.45}$$

Defining $r = G^{-1}w$, the solution to b is

$$b = \frac{w^T(I - MG^{-1})z}{w^T(I - MG^{-1})w} = \frac{z^T G^{-1} w}{w^T G^{-1} w} = \frac{z^T r}{w^T r} \tag{A.46}$$

We can use our earlier reasoning to quickly calculate the leave one out error. Again, let f_{S^i} be the classifier we get from training on all training points except X_i, and let c_i and b_i be its associated parameters. Furthermore, define

$$y_j^i = \begin{cases} y_j & i \neq j \\ f_{S^i}(x_i) & i = j \end{cases} \tag{A.47}$$

$$z^i = W y^i \tag{A.48}$$

Note that this time our definition of f_S is complicated by the addition of b. However, our proof that f_{S^i} is equivalent to training over all points using labels y^i remains the same. Writing out f_{S^i} explicitly, we get

$$\begin{aligned} f_S(x_i) = (Kc)_i + b &= \frac{(Md)_i}{w_i} + b = \frac{(MG^{-1}(z - bw))_i}{w_i} + b \\ &= \frac{(MG^{-1}z)_i}{w_i} + \frac{b}{w_i}(w_i - (MG^{-1}w)_i) \end{aligned} \tag{A.49}$$

We can use Lemma A.2.1 to simplify the above:

$$w_i - (MG^{-1}w)_i = (w - MG^{-1}w)_i = \lambda r_i \tag{A.50}$$

$$\rightarrow f_S(x_i) = \frac{(MG^{-1}z)_i}{w_i} + \frac{\lambda b}{w_i} r_i \tag{A.51}$$

Taking the difference of values on x_i we get

$$f_{S^i}(x_i) - f_S(x_i) = \frac{1}{w_i} \sum_{j=1}^{n} (MG^{-1})_{ij}(z_j^i - z_j) + \frac{\lambda}{w_i} r_i (b^i - b) \tag{A.52}$$

which can be simplified by noting that

$$b^i - b = \frac{z^{i^T} r}{w^T r} - \frac{z^T r}{w^T r} = \frac{(z^i - z)^T r}{w^T r} = \frac{(w_i f_{S^i}(x_i) - z_i) r_i}{w^T r} \tag{A.53}$$

Hence,

$$\begin{aligned} f_{S^i}(x_i) - f_S(x_i) &= \frac{1}{w_i}(MG^{-1})_{ii}(w_i f_{S^i}(x_i) - z_i) + \frac{\lambda}{w_i}\frac{r_i^2}{w^T r}(w_i f_{S^i}(x_i) - z_i) \\ &= \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r} \right) (f_{S^i}(x_i) - y_i) \end{aligned} \tag{A.54}$$

Solving for $f_{S^i}(x_i)$, we get

$$\begin{aligned} f_{S^i}(x_i) \left(1 - (MG^{-1})_{ii} - \frac{\lambda r_i^2}{w^T r} \right) &= f_S(x_i) - \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r} \right) y_i \\ f_{S^i}(x_i) &= \frac{f_S(x_i) - \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r} \right) y_i}{1 - (MG^{-1})_{ii} - \frac{\lambda r_i^2}{w^T r}} \end{aligned} \tag{A.55}$$

Next, the leave one out error affords further simplifications:

$$
\begin{aligned}
y_i - f_{S^i}(x_i) &= y_i - \frac{f_S(x_i) - \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r}\right) y_i}{1 - (MG^{-1})_{ii} - \frac{\lambda r_i^2}{w^T r}} \\
&= \frac{y_i - \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r}\right) y_i - f_S(x_i) + \left((MG^{-1})_{ii} + \frac{\lambda r_i^2}{w^T r}\right) y_i}{1 - (MG^{-1})_{ii} - \frac{\lambda r_i^2}{w^T r}} \\
&= \frac{y_i - f_S(x_i)}{1 - (MG^{-1})_{ii} - \frac{\lambda r_i^2}{w^T r}} \qquad \text{(A.56)}
\end{aligned}
$$

The numerator can be simplified by

$$
y_i - f_S(x_i) = y_i - \frac{(MG^{-1}z)_i}{w_i} - \frac{\lambda b}{w_i} r_i = \frac{z_i - (MG^{-1}z)_i - \lambda b r_i}{w_i} \qquad \text{(A.57)}
$$

$$
z_i - (MG^{-1}z)_i - \lambda b r_i = ((I - MG^{-1})z - \lambda b r)_i = \lambda (G^{-1}z - bG^{-1}w)_i = \lambda d_i \qquad \text{(A.58)}
$$

Combining our results,

$$
y_i - f_{S^i}(x_i) = \frac{\lambda d_i}{w_i \left(\lambda G_{ii}^{-1} - \frac{\lambda r_i^2}{w^T r}\right)} \qquad \text{(A.59)}
$$

so that the unweighted leave one out error for all points is

$$
L_E = \frac{d}{W\left(diag(G^{-1}) - \frac{r * r}{w^T r}\right)} = \frac{c}{W^2\left(diag(G^{-1}) - \frac{r * r}{w^T r}\right)} \qquad \text{(A.60)}
$$

where division and $*$ are performed element-wise.

Finally, we consider the 4 kinds of error statistics we may wish to estimate:

1. The weighted leave one out squared error is given by

$$
L_{WE}^{SQ} = L_E^T A L_E \qquad \text{(A.61)}
$$

2. The leave one out misclassification error requires the leave one out values, which are

$$
L_V = y - L_E \qquad \text{(A.62)}
$$

3. The weighted empirical squared can be calculated via

$$\begin{aligned} R^{SQ} &= (y - Kc - b)^T A(y - Kc - b) \\ &= (z - Md - bw)^T (z - Md - bw) \qquad \text{(A.63)} \\ z - Md - bw &= z - MG^{-1}(z - bw) - bw \\ &= z - MG^{-1}z - b(w - MG^{-1}w) \\ &= \lambda G^{-1}z - \lambda b G^{-1}w = \lambda d = \lambda W^{-1}c \qquad \text{(A.64)} \\ R^{SQ} &= \lambda^2 c^T A^{-1} c \qquad \text{(A.65)} \end{aligned}$$

4. Finally, the empirical misclassification error requires calculation of $Kc + b$:

$$Kc + b = y - y + Kc + b = y - (y - Kc - b) = y - \lambda A^{-1}c \qquad \text{(A.66)}$$

A.4 Active Learning Update Theorems

Let X be a pool of points and L an index set such that $X_L \subset X$ is a subset for which we know extra information, encoded in a $|L|$ dimensional column vector a_L. Furthermore, suppose that we are given a kernel k, regularization parameter λ, and diagonal weight matrix W^L which we use to form an approximate inverse $G_L^{-1} = (k(X_L, X_L) + \lambda W^L)^{-1}$. We can propagate our knowledge of a^L to all of X via

$$b^L = k(X, X_L) G_L^{-1} a^L \qquad \text{(A.67)}$$

Consider updating b^L to $b^{L+\{i\}}$ so that it includes new information, a_i, about some $x_i \notin X_L$ which is in our original pool. Rather than computing $G_{L+\{i\}}^{-1}$ we would like to reuse our work done in calculating b^L. Theorem A.4.1 shows how to perform this update with minimal computation.

Theorem A.4.1 *Suppose we are in a scenario as described above. Then*

$$b^{L+\{i\}} = b^L + \frac{b_i^L - a_i}{m}(V_i^L - K(x_i, X))^T \tag{A.68}$$

$$V^L = K(X, X_L)G_L^{-1}K(X_L, X) \tag{A.69}$$

$$m = K(x_i, x_i) + \lambda W^i - V_{ii}^L \tag{A.70}$$

Proof Let $z = G_L^{-1}K(X_L, x_i)$, then by the Matrix Inversion Lemma,

$$\begin{aligned} b^{L+\{i\}} &= \begin{bmatrix} K(X, X_L) & K(X, x_i) \end{bmatrix} \begin{bmatrix} G_L^{-1} + \frac{zz^T}{m} & -\frac{z}{m} \\ -\frac{z^T}{M} & \frac{1}{m} \end{bmatrix} \begin{bmatrix} a_L \\ a_i \end{bmatrix} \\ &\quad \begin{bmatrix} K(X, X_L) & K(X, x_i) \end{bmatrix} \begin{bmatrix} G_L^{-1}a_L + \frac{z}{m}(z^T a_L - a_i) \\ -\frac{z^T a_L - a_i}{m} \end{bmatrix} \qquad \text{(A.71)} \\ &= K(X, X_L)G_L^{-1}a_L + (K(X, X_L)z - K(X, x_i))\left(\frac{z^T a_L - a_i}{m}\right) \qquad \text{(A.72)} \\ &= b^L + \frac{b_i^L - a_i}{m}(V_i^L - K(X, x_i))^T \quad \blacksquare \qquad \text{(A.73)} \end{aligned}$$

We also derive a corollary from Lemma A.2.1 that is useful for online and active learning.

Corollary A.4.2 *Suppose that we wish to compute some vector* $c = G_L^{-1}(p - q_L)$ *and we are given* $d = K(X, X_L)G_L^{-1}(p - q_L) + q_L$. *Then*

$$\begin{aligned} d_L &= K(X_L, X_L)G_L^{-1}(p - q_L) \\ &= p - q - \lambda W^L G_L^{-1}(p - q_L) + q_L = p - \lambda W^L c \qquad \text{(A.74)} \\ &\rightarrow c = (W^L)^{-1}\frac{p - d_L}{\lambda} \qquad \text{(A.75)} \end{aligned}$$

Bibliography

[1] Baldassarre, Rosasco, Barla, and Verri. Multi-output learning via spectral filtering. *Preprint*, 2010.

[2] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. Importance weighted active learning. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 49–56, 2009.

[3] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, pages 499–526, 2002.

[4] Cucker and Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 2002.

[5] Sanjoy Dasgupta, Daniel Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. *Advances in Neural Information Processing Systems (NIPS)*, 20, 2007.

[6] Sanjoy Dasgupta, Adam Tauman Kalai, and Claire Monteleoni. Analysis of perceptron-based active learning. *Journal of Machine Learning Research*, 2009.

[7] Luc Devroye, Lázló Győrfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31 of *Applications of Mathematics*. Springer, New York, 1996.

[8] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 1999.

[9] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:7138–168, 1997.

[10] J. Haupt, R. Castro, and R. Nowak. Distilled sensing: Adaptive sensing for sparse detection and estimation. submitted, 2010.

[11] Jarvis Haupt, Rui Castro, and Robert Nowak. Distilled sensing: Adaptive sampling for sparse detection and estimation. *AI and Statistics (AISTATS)*, 2009.

[12] The MathWorks Inc. Matlab, 2010.

[13] Donald E. Knuth. *Seminumerical Algorithms.* Addison-Wesley, 1981.

[14] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models.* The MIT Press, Cambridge, Massachusetts, 2009.

[15] A. N. Kolmogorov and S. V. Fomin. *Elements of the Theory of Functions and Functional Analysis.* Addison-Dover Publications, Inc., Mineola, New York, 1957.

[16] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. Cns: a gpu-based framework for simulating cortically-organized networks. *MIT-CSAIL-TR-2010-013*, CBCL-286, 2010.

[17] Jim Mutch and David G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision (IJCV)*, 80:45–57, 2008.

[18] John Platt. *Advances in Large Margin Classifiers.* MIT Press, 1999.

[19] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, Cambridge, Massachusett, 2006.

[20] Stefan Rűping. A simple method for estimating conditional probabilities for svms, 2004.

[21] Rifkin. *Everything old is new again: A fresh look at historical approaches in machine learning.* PhD thesis, Massachusetts Institute of Technology, 2002.

[22] R. Rifkin and R. A. Lippert. Notes on regularized least-squares. *AI Technical Report*, 268, 2002.

[23] Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[24] Bernhard Schőlkopf and Alexander J. Smola. *Learning with Kernels.* The MIT Press, Cambridge, Massachusetts, 2002.

[25] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine*, 29, 2007.

[26] H. S. Seung, M. Opper, and H. Sompolinksy. Query by committee. *Proceedings of the fifth annual workshop on Computational Learning Theory*, 1992.

[27] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, pages 45–66, 2001.

[28] V. N. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[29] Xiaojin Zhu, John Lafferty, and Zoubin Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.