On the Role of Feature Selection in Machine Learning

Thesis submitted in partial fulfillment of the degree of
Doctor of Philosophy
by

Amir Navot

Submitted to the Senate of the Hebrew University

December 2006

This work was carried out under the supervision of Prof. Naftali Tishby.

Acknowledgments

Many people helped me in many ways over the course of my Ph.D. studies and I would like to take this opportunity to thank them all. A certain number of people deserve special thanks and I would like to express my gratitude to them with a few words here. The first is my supervisor Naftali (Tali) Tishby who taught me a great deal and always supported me, even when I put forward extremely wild and ill-thought out ideas. Tali played a major part in shaping my scientific point of view. The second person is Ran Gilad-Bachrach, who was both like a second supervisor and a good friend. He guided my first steps as a Ph.D. student, and was always ready to share ideas and help with any issue. Our many discussions on everything were the best part of my Ph.D. studies. My other room mates, Amir Globerson and Gal Chechik deserve thanks for all the help and the inspiring discussions. Lavi Shpigelman, a good friend and a great classmate, was always ready to help on everything from a worthy scientific question to a very technical problem with my computer. Aharon Bar-Hillel challenged me with tough questions but also helped me find the answers. My brother, Yiftah Navot, has been my longstanding mentor and is always there to help me with any mathematical problem. There is no way I can express how much I value his assistance. My mother always made it very clear that studies are the most important thing. Of course my wife, Noa, without whose endless support I never would have finished (or probably have never even started) my Ph.D. studies. Finally I would also like to thank Esther Singer for her efforts to improve my English, and all the administrative staff of both the ICNC and CS who were always kind to me and were always willing to lend a hand. I also thank the Horowitz foundation for the generous funding they have provided me.

Abstract

This thesis discusses different aspects of feature selection in machine learning, and more specifically for supervised learning. In machine learning the learner (the machine) uses a training set of examples in order to build a model of the world that enables reliable predictions. In supervised learning each training example is an (instance, label) pair, and the learner's goal is to be able to predict the label of a new unseen instance with only a small chance of erring. Many algorithms have been suggested for this task, and they work fairly well on many problems; however, their degree of success depends on the way the instances are represented. Most learning algorithms assume that each instance is represented by a vector of real numbers. Typically, each number is a result of a measurement on the instance (e.g. the gray level of a given pixel of an image). Each measurement is a *feature*. Thus a key question in machine learning is how to represent the instances by a vector of numbers (features) in a way that enables good learning performance. One of the requirements of a good representation is conciseness, since a representation that uses too many features raises major computational difficulties and may lead to poor prediction accuracy. However, in many supervised learning tasks the input is originally represented by a very large number of features. In this scenario it might be possible to find a smaller subset of features that can lead to much better performance.

Feature selection is the task of choosing a small subset of features that is sufficient to predict the target labels well. Feature selection reduces the computational complexity of learning and prediction algorithms and saves on the cost of measuring non selected features. In many situations, feature selection can also enhance the prediction accuracy by improving the signal to noise ratio. Another benefit of feature selection is that the identity of the

selected features can provide insights into the nature of the problem at hand. Therefore *feature selection* is an important step in efficient learning of large multi-featured data sets. On a more general level the feature selection research field clearly enters into research on the fundamental issue of data representation.

In chapter 2 we discuss the necessity of feature selection. We raise the question of whether a separate stage of feature selection is indeed still needed, or whether modern classifiers can overcome the presence of huge number of features. In order to answer this question we present a new analysis of the simple two-Gaussians classification problem. We first consider the maximum likelihood estimation as the underlying classification rule. We analyze its error as a function of the number of features and number of training instances, and show that while the error may be as poor as chance when using too many features, it approaches the optimal error if we chose the number of features wisely. We also explicitly find the optimal number of features as a function of the training set size for a few specific examples. Then, we test SVM [14] empirically in this setting and show that its performance matches the predictions in the analysis. This suggests that feature selection is still a crucial component in designing an accurate classifier, even when modern discriminative classifiers are used, and even if computational constraints or measuring costs are not an issue.

In chapter 3 we suggest new methods of feature selection for classification which are based on the maximum margin principle. A margin [14, 100] is a geometric measure for evaluating the confidence of a classifier with respect to its decision. Margins already play a crucial role in current machine learning research. For instance, SVM [14] is a prominent large margin algorithm. The novelty of the results presented in this chapter lies in the use of large margin principles for feature selection. The use of margins allows us to devise new feature selection algorithms as well as to prove a PAC (Probably Approximately Correct) style generalization bound. The bound is on the generalization accuracy of 1-NN on a selected set of features, and guarantees good performance for any feature selection scheme which selects a small set of features while keeping the margin large. On the algorithmic side, we use a margin based criterion to measure the quality of sets of features. We present two new feature selection algorithms, *G-flip* and *Simba*, based on this criterion. The merits of these algorithms are demonstrated on various datasets.

In chapter 4 we discuss feature selection for regression (aka function estimation). Once again we use the Nearest Neighbor algorithm and an evaluation function which is similar in its nature to the one used for classification in chapter 3. This way we develop a non-linear, simple, yet effective feature subset selection method for regression and use it in analyzing cortical neural activity. This algorithm is able to capture complex dependency of the target function upon its input and makes use of the leave-one-out error as a natural regularization. We explain the characteristics of our algorithm with synthetic problems and use it in the context of predicting hand velocity from spikes recorded in motor cortex of a behaving monkey. By applying feature selection we are able to improve prediction quality and we suggest a novel way of exploring neural data.

Finally, chapter 5 extends the standard framework of feature selection to consider generalization in the features axis. The goal of standard feature selection is to select a subset of features from a given set of features. Here, instead of trying to directly determine which features are better, we attempt to learn the properties of good features. For this purpose we assume that each feature is represented by a set of properties, referred to as *meta-features*. This approach has three main advantages. First, the selection problem can be considered as a standard learning problem in itself. This novel viewpoint enables derivation of better generalization bounds for the joint learning problem of selection and classification. Second, it allows us to devise selection algorithms that can efficiently explore for new good features in the presence of a huge number of features. Finally, it contributes to a better understanding of the problem. We also show how this concept can be applied in the context of *inductive transfer*. We show that transferring the properties of good features between tasks might be better than transferring the good features themselves. We illustrate the use of meta-features in the different applications on a handwritten digit recognition problem.

Contents

Acknowledgments v

Abstract vii

1 Introduction **1**

1.1 Machine Learning 1

1.1.1 Supervised Learning 2

1.1.2 Representation 5

1.1.3 Machine Learning, Artificial Intelligence and Statistics 7

1.1.4 Machine Learning and Neural Computation 8

1.1.5 Other Learning Models 9

1.2 Feature Selection 14

1.2.1 Common paradigms for feature selection 16

1.2.2 The Biological/Neuroscience Rationale 21

1.3 Notation 22

2 Is Feature Selection Still Necessary? **24**

2.1 Problem Setting and Notation 26

2.2 Analysis 27

2.2.1 Observations on The Optimal Number of Features 28

2.3 Specific Choices of $\boldsymbol{\mu}$ 31

2.4 SVM Performance 33

2.5 Summary 36

3 Margin Based Feature Selection **38**

3.1 Nearest Neighbor classifiers 40

3.2 Margins 41

3.3 Margins for 1-NN 42

3.4 Margin Based Evaluation Function 44

3.5 Algorithms 46

3.5.1 Greedy Feature Flip Algorithm (G-flip) 46

3.5.2 Iterative Search Margin Based Algorithm (Simba) 47

3.5.3 Comparison to Relief 48

3.5.4 Comparison to R2W2 49

3.6 Theoretical Analysis 50

3.7 Empirical Assessment 52

3.7.1 The Xor Problem 52

3.7.2 Face Images 54

3.7.3 Reuters 55

3.7.4 Face Images with Support Vector Machines 59

3.7.5 The NIPS-03 Feature Selection Challenge 62

3.8 Relation to Learning Vector Quantization 63

3.9 Summary and Discussion 65

A Complementary Proofs for Chapter 3 66

4 Feature Selection For Regression **69**

4.1 Preliminaries 71

4.2 The Feature Selection Algorithm 72

4.3 Testing on synthetic data 74

4.4 Hand Movement Reconstruction from Neural Activity 76

4.5 Summary 79

5 Learning to Select Features **81**

5.1 Formal Framework 83

5.2 Predicting the Quality of Features 84

5.3 Guided Feature Extraction 86
5.3.1 Meta-features Based Search 87
5.3.2 Illustration on Digit Recognition Task 88
5.4 Theoretical Analysis 92
5.4.1 Generalization Bounds for *Mufasa* Algorithm 92
5.4.2 VC-dimension of Joint Feature Selection and Classification 96
5.5 Inductive Transfer 99
5.5.1 Demonstration on Handwritten Digit Recognition 100
5.6 Choosing Good Meta-features 101
5.7 Improving Selection of Training Features 104
5.8 Summary 105
A Notation Table for Chapter 5 106

6 Epilog **108**

List of Publications **111**

Bibliography **113**

Summary in Hebrew **I**

Chapter 1

Introduction

This introductory chapter provides the context and background for the results discussed in the following chapters and defines some crucial notation. The chapter begins with a brief review of machine learning, which is the general context for the work described in this thesis. I explain the goals of machine learning, present the main learning models currently used in the field, and discuss its relationships to other related scientific fields. Then, in section 1.2, I review the field of feature selection, which is the subfield of machine learning that constitutes the core of this thesis. I outline the rationale for feature selection, the different paradigms that are used and survey some of the most important known algorithms. I show the ways in which research in feature selection is related to biology in section 1.2.2.

1.1 Machine Learning

Machine learning deals with the theoretic, algorithmic and applicative aspects of l*earning from examples*. In a nutshell, learning from examples means that we try to build a machine (i.e. a computer program) that can learn to perform a task by observing examples. Typically, the program uses the *training examples* to build a model of the world that enables reliable predictions. This contrasts with a program that can make predictions using a set of predefined rules (the classical Artificial Intelligence (AI) approach). Thus in *machine learning* the machine must learn from its own experience, and in that sense it adheres to the very

old - but still wise - proverb of our Sages of Blessed Memory (Hazal):

אין חכם כבעל נסיון

which translates to: "experience is the best teacher". This proverb implies that a learner needs to acquire experience on his or her own in order to achieve a good level of performance and not be satisfied with the explanations given by the teacher (pre-defined rules in our case). I elaborate on the advantages of this approach in section 1.1.3.

1.1.1 Supervised Learning

While there are a few different learning frameworks in machine learning (see section 1.1.5), we focus here on *supervised learning*. In supervised learning we get a labeled sample as input and use it to predict the label of a new unseen instance. More formally, we have a training set $S^m = \{\mathbf{x}^i, y^i\}_{i=1}^m$, $\mathbf{x}^i \in \mathbb{R}^N$ and $y^i = c\left(\mathbf{x}^i\right)$ where c is an unknown, but fixed function. The task is to find a mapping h from $\mathbb{R}^N$ to the label set with a small chance of erring on a new unseen instance, $\mathbf{x} \in \mathbb{R}^N$, that was drawn according to the same probability function as the training instances. c is referred to as the *target concept* and the N coordinates are called *features*. An important special case is when c has only a finite number of categorical values. In this case we have a *classification* problem. In most of this dissertation we focus on classification (except in chapter 4).

The conventional assessment of the quality of a learning process is the generalization ability of the learned rule, i.e. how well it can predict the value of the target concept on a new unseen instance. We focus on *inductive batch* learning[1], and specifically on the *Probably Approximately Correct* (PAC) learning model [118, 13], where it is assumed that the training instances are drawn iid according to a fixed (but unknown) distribution $\mathcal{D}$. Thus the generalization ability of a classification rule h is measured by its *generalization error*, which is the chance to err on a new unseen instance that was drawn according to the same distribution $\mathcal{D}$. Formally it is defined as follows:

$$e_{\mathcal{D}}(h) = Pr_{\mathbf{x}\sim\mathcal{D}}\left(h\left(\mathbf{x}\right) \neq c\left(\mathbf{x}\right)\right)$$

However, we usually cannot measure this quantity directly. Thus we look at the *training*

[1] see section 1.1.5 for a short review of other common learning models

error, where the training error of a classifier h with respect to a training set S of size m is the percentage of the training instances that h err on; formally defined as follows:

$$e_S(h) = \frac{1}{m} \sum_i \left(1 - \delta_{h(\mathbf{x}_i), c(\mathbf{x}_i)}\right)$$

where $\mathbf{x}_i$ is the i's training instance and δ is Kronecker's delta.

From the theoretic point of view, machine learning attempts to characterize what is *learnable*, i.e. under which conditions a small training error guarantees a small generalization error. Such a characterization can give us insights into the theoretic limitations of learning that any learner must obey. It is clear that if h is allowed to be any possible function, there is no way to bound the gap between generalization error and training error. Thus we have to introduce some restrictive conditions. The common assumption is that we choose our hypothesis from a given class of hypotheses $\mathcal{H}$. The classical result here is the characterization of the learnable hypotheses classes in the PAC model [118, 13]. Loosely speaking, a hypotheses class is PAC-learnable if there is an algorithm that ensures that the gap between the generalization error and the training error is arbitrarily small when the number of training instances is large enough. The characterization theorem states that a class is learnable if and only if it has a finite VC-dimension. The VC-dimension is a combinatorial property which measures the complexity of the class. The bound is tighter when the VC-dimension is smaller, i.e. when the hypotheses class is simpler. Thus, this result is a demonstration of principle of Occam's razor: lex parsimoniae (law of succinctness):

entia non sunt multiplicanda praeter necessitatem,

which translates to:

entities should not be multiplied beyond necessity.

Which often rephrased as: "The simplest explanation is the best one" .

Another kind of bounds on the generalization error of a classifier are *data dependent* bounds [103, 9]. Whereas standard bounds depends only on the *size* of the training set, data dependent bounds take advantage of the fact that some training sets are better than others. That is, the better the training set, the better bounds it gives on the generalization error. This way, the data give bounds which are tighter than the standard bounds. A main component in data dependent bounds is the concept of *margins*. Generally speaking,

a margin [14, 100] is a geometric measure for evaluating the confidence of a classifier with respect to its decision. We elaborate on the definition and usage of margins in chapter 3. We make extensive use of data dependent bounds in our theoretical results in chapter 3 and chapter 5.

Algorithmically speaking, machine learning tries to develop algorithms that can find a good rule (approximation of the target concept) using the given training examples. Ideally, it is preferable to find algorithms for which it is possible to prove certain guarantees on the running time and the accuracy of the result rule. However, heuristic learning algorithms that "just work" are also abundant in the field, and sometimes in practice they work better than the alternative "provable" algorithms. One possible algorithmic approach to (classification) supervised learning is to build a probabilistic model for each class (using standard statistic tools) and then classify a new instance into the class with highest likelihood. However, since many different probabilistic models imply the same decision boundary, it can be easier to learn the decision boundary directly (the discriminative approach). Two such classic algorithms are the Perceptron [98] and the One Nearest Neighbor (1-NN) [32] that were introduced half a century ago and are still popular. The Perceptron directly finds a hyper-plane that correctly separates the training instances by going over the instances iteratively and updating the hyper-plane direction whenever the current plane errs. 1-NN on the other hand simply stores the training instances and then assigns a new instance with the label of the closest training instance.

The most important algorithm for supervised learning in current machine learning is the Support Vector Machine (SVM) [14, 116], which finds the hyper-plane that separates the training instances with the largest possible margin, i.e., the separating plane which is as far from the training instances as possible. By an implicit mapping of the input vectors to a high dimension Hilbert space (using a kernel function) SVM provides a systematic tool for finding non-linear separations using linear tools. Another prominent algorithm for supervised classification is the AdaBoost [33], which uses a weak-learner to build a strong classifier. It builds a set of classifiers by re-running the weak classifier on the same training set, but each time putting more weight on instances where the previous classifiers erred. Many other algorithms for supervised learning exist and I only mention some of the most

important ones.

In terms of applications,, machine learning tries to adapt algorithms to a specific task. Learning algorithms are widely used for many tasks both in industry and in academia, e.g., face recognition, text classification, medical diagnosis and credit card fraud detection, just to name a few. Despite the fact that many learning algorithms are general and can be applied to any domain, "out-of-the-box" learning algorithms do not generally work very well on given real-world problems. Aside from the required tuning, each domain has its own specific difficulties and further effort may be required to find the right representation of instances (see section 1.1.2).

1.1.2 Representation

A key question in machine learning is how to represent the instances. In most learning models it is assumed that the instances are given as a vector in $\mathbb{R}^N$ (where N is any finite dimension), and the analysis starts from there. There is a general consensus that once we have a good representation, most reasonable learning methods will perform well after a reasonable tuning effort. On the other hand, if we choose a poor representation achieving a good level of performance is hopeless. But how do we choose the best way to represent an abstract object (e.g. image) by a vector of numbers? A good representation should be compact and meaningful at the same time. Is there a general method to find such a representation? Choosing a representation means choosing a set of *features* to be measured on each instance. In real life, this set of features is usually chosen by a human expert in the relevant domain who has a good intuition of what might work. The question is whether it is possible to find algorithms that use the training sample (and possibly other external knowledge) in order to find a good representation automatically.

If the instances are physical entities (e.g. a human patients), choosing the features means choosing which physical measurements to perform on them. In other cases the instances are given as a vectors of numbers in the first place (e.g. the gray level of pixels of digital image) and then the task of finding good representation (i.e. good set of features) is the task of finding a transformation that convert the original representation into a better one. This can

be done without using the labels, in unsuprevised manner (see section 1.1.5) or using the labels. If the instances originally described by a large set of raw features, one way to tackle this is by using *dimensionality reduction*. In dimensionality reduction we look for a small set of functions (of the raw features) that capture the relevant information. In the context of supervised learning, dimensionality reduction algorithms try to find a small number of functions that preserve the information on the labels.

Feature selection is a special form of dimensionality reduction, where we limit ourselves to choosing only a subset out of the given set of raw features. While this might seems to be strong limitation, feature selection and general dimensionality reduction are not that different, considering that we can always first generate many possible functions of the raw features (e.g. many kinds of filters and local descriptors of an image) and then use feature selection to choose only some of them. This process of generating complex features by applying functions on the raw features is called *feature extraction*. Thus, in other words, using feature extraction followed by feature selection we can get a general dimensionality reduction. Nevertheless, in feature extraction we have to select which features to generate out of the huge (or even infinite) number of possible functions of the raw features. We tackle this issue in chapter 5. As will be explained in more detail in the section 1.2, the main advantages of feature selection over other dimensionality reduction methods are interpretabilty and economy (as it saves the cost of measuring the non- selected features).

The "holy grail" is to find a representation which is concise and allow classification by a simple rule at the same time, since a simple classifier over low dimension generalize well. However, this is not always possible, and therefore the is a tradeoff between conciseness the representation and complexity of the classifier. Different methods may choose different work point of this tradeoff. While dimensionality reduction methods focus on conciseness, SVM[2] (and other kernel machines) convert the data into a very sparse representation in order to allow very simple (namely linear) classification rule and control overfitting by maximizing the margin. However, In chapter 2 we suggest that the ability of SVM to avoid overfitting on high dimensional data is limited to scenarios where the data is laying on a low dimensional manifold.

[2] See section 1.1.1 for a short description of SVM

Another related issue in data representation is the tradeoff between the conciseness of the representation and its potential prediction accuracy. One principled way for quantifying this tradeoff, known as the *Information Bottleneck* [111], is to measure both the complexity of the model and its prediction accuracy by using Shannon's mutual information, measuring both complexity and accuracy by bits. During my PhD studies I also took major part in research on the relation between the Information Bottleneck framework and some classical problems in Information Theory (IT) such as a version of *source coding with side information* presented by Wyner, Ahlswede and Korner (WAK) [121, 3], Rate-Distortion and Cost-Capacity [102]. In this research we took advantage of the similarities to obtain new results and insights both on the IB and the classical IT problems. These results are not included in this thesis due to space limitations. They where published in [39] and are under review process for publication in IEEE-IT.

1.1.3 Machine Learning, Artificial Intelligence and Statistics

Machine learning can also be considered a broad sub-field of Artificial Intelligence (AI). AI refers to the ability of an artificial entity (usually a computer) to exhibit intelligence. While AI in general tends to prompt profound philosophical issues, as a research field in computer science it is confined to dealing with the ability of a computer to perform a task that is usually done by humans and is considered as a task that requires intelligence. A classical AI approach tackles such a task by using a set of logical rules (which were designed by a human expert) that constitute a flow chart. Such systems are usually referred to as *Expert Systems*, *Rule-Based Systems* or *Knowledge-Based Systems* [51]. The first such systems were developed during the 1960s and 1970s and became very popular and applied commercially during the 1980s [76]. On the other hand, as already mentioned, in machine learning the computer program tries to derive the rules by itself using a set of input-output pairs (aka *training set*). Thus machine learning focuses on the *learning process*.

For example, take the task of *text classification*. Here the task is to classify free-language texts into one or more classes from a predefined set of classes. Each class can represent a topic, an author or any other property. A rule-based system for such a task usually

consists of pre-defined rules that query the existence of some words or phrases in the text to be classified. The rules are extracted by humans, i.e., the programmer together with an expert in the relevant field who knows how to predict which phrases characterize each class. There are several main drawbacks to this approach. First, the task of defining the set of rules requires a great deal of expert human work and becomes virtually impossible for large systems. Moreover, even if we already have such a system that actually works, it is very hard to maintain it. Imagine that we want to add a new class to a system with a thousand rules. It is very hard to predict the effect of changing even one rule. This is particularly crucial because the environment changes over time, e.g., new classes appear and old classes disappear, the language changes (who used the word "tsunami" before December 26, 2004?), and it is very awkward to adapt such a system to such changes. The machine learning approach overcomes the above drawbacks since the system is built automatically from a labeled training sample and the rule can be adapted automatically with any feedback (i.e., an additional labeled instance). However, machine learning raises other problems such as the interpretability of the decision and the need for a large amount of labeled instances. These issues will be discussed in more detail in the following.

Machine learning is also closely related to *statistics*. Both fields try to estimate an unknown function from a finite sample; thus there is a large overlap between them. Indeed many statisticians claim that some of the results that are considered new in machine learning are well known in the statistics community. However, while the formal framework might be similar, the applicative point of view is very different. While classical statistics focuses on hypothesis testing and density estimation, machine learning focuses on how to create computer programs that can learn to perform an intelligent task. One additional difference is that machine learning, as sub-field of computer science, is also concerned with the algorithmic complexity of computational implementations.

1.1.4 Machine Learning and Neural Computation

The brain of any animal and especially the human brain is the best known learner. Over the course of a lifetime the brain has to learn how to perform a huge number of new, com-

plicated tasks and has to solve computational problems continuously and instantaneously. For example the brain has to resolve the visual input that comes through the visual system and produce the correct scene. This involves such problems as segmentation (which pixel belongs to which object), detection of objects and estimation of objects' movements. Correct resolving is crucial for the creature's survival. There are several different approaches to study the brain as a computational machine. One is to look inside the brain and try to "see" how it works (Physiology); another is to try to build an abstract model of the brain (regardless of the specific hardware) that fits the observations and can predict the brain's behavior (Cognitive Psychology). The common sense behind using machine learning for brain research is the belief that if we are able build a machine that can deal with the same computational problems the brain has to solve, it will teach us something about how the brain works. As regards theory, machine learning can provide the limitations on learning process that any learner must obey , including the brain.

1.1.5 Other Learning Models

In life, different learning tasks can be very different from each other in many respects, including the kind of input/feedback we get (instances alone or instances with some "hints") , the way we are tested (during the learning process or only at the end, number of allowed mistakes) and the level of control we have over the learning process (can we affect the world or just observe it?). Thus, many different models have been suggested for formalizing learning. As already explained, this work focuses on passive-inductive-batch-supervised learning that was presented in section 1.1.1. However, we refer to some other models as well. For this reason and in order to give a more complete picture of the machine learning field I briefly overview the other main models and point out the differences between them.

Supervised vs. Unsupervised Learning

As already mentioned, in supervised learning the task is to learn an unknown concept c from examples, where each instance $\mathbf{x}$ comes together with the value of the target function for this instance, $c(\mathbf{x})$. Thus the task is to find an approximation of c, and performance

is measured by the quality of the approximation for points that did not appear in the training set. On the other hand, in unsupervised learning the training sample includes only instances, without any labels, and the task is to find "interesting structures" in the data. Typical approaches to unsupervised learning include *clustering*, building probabilistic g*enerative models* and finding *meaningful transformations* of the data. Clustering algorithms try to cluster the instances into a few clusters such that instances inside the same cluster are "similar" and instances in two different clusters are "different". A classical clustering algorithm is the k-means [77] which clusters instances according to the Euclidean distance. It starts with random locations of k cluster centers and then iteratively assigns instances to the cluster of the closest center, and updates the centers' locations to the center of gravity of the assigned instances. Clustering is also one of the more obvious applications of the Information-Bottleneck [111], which was already mentioned in section 1.1.2. The Information-Bottleneck is an information theoretical approach for extracting the relevant information in one variable with respect to another variable, and thus can be used for finding a clustering of one variable that preserves the maximum information on the other variable. Generative models methods assume that the data were drawn from a distribution of a certain form (e.g. mixture of Gaussians) and looks for the parameters that maximize the *likelihood* of the data. The prominent algorithm here is the *Expectation Maximization* (EM) [26] family of algorithms. *Principal Component Analysis* (PCA) [56] is a classic example of an algorithm that looks for an interesting transformation of the data. PCA finds the linear transformation into a given dimension that preserves the maximal possible variance. Other prominent algorithms of this type are Multidimensional Scaling (MDS) [67], *Projection Pursuit* [35, 50, 36, 57], *Independent Component Analysis* (ICA) [11] and the newer *Local Linear Embedding* (LLE) [99].

Inductive vs transductive

Supervised learning can be divided into *inductive* learning and *transductive* learning [117]. In inductive learning we want our classifier to perform well on *any* instance that was drawn from the same distribution as the training set. On the other hand, in transductive learning it is assumed that the test instances were known at training time (only the instances, not their

labels of course), and thus we want to find a classifier that performs well on these predefined test instances alone. These two models comply with different real life tasks. The difference can be illustrated on the task of text classification. Assume that we want to build a system that will classify emails that will arrive in the future using a training set of classified emails we received in the past. This fits the inductive learning model, since we do not have the future emails at the training stage. On the other hand if we have an archive of a million documents, where only one thousand of them are classified and we want to use this subset to build a classifier for classifying the rest of the archive, this fits the transductive model. In other words, transductive learning fits situations where we have the test questions while we are studying for the exam.

Batch vs. Online Learning

We can also divide supervised learning into *batch* learning and *online* learning [72]. In batch learning it is assumed that the learning phase is separate from the testing phase, i.e., we first get a batch of labeled instances, we use them to learn a model (e.g. classifier) and then we use this model for making predictions on new instances. When analyzing batch models it is usually assumed that the instances were drawn iid from an underlying distribution and that the test instances will be drawn from the same distribution. The accuracy of a model is defined as the expected accuracy with respect to this distribution. However, this model does not fit many real life problems, where we have to learn "on the go" and do not have a sterile learning phase, i.e., we must be able to make predictions from the very beginning and we pay for errors in the learning phase as well. Online learning assumes that we get the instances one by one. We first get the instance without a label and have to make a prediction. Then we get the real label, and suffer a loss if our prediction was wrong. Now we have the chance to update our model before getting another instance and so on. In analyzing online algorithms, performance is measured by the cumulative loss (i.e., the sum of losses we suffered so far), and the target is to achieve a loss which is not "much more" than the loss made by the best model (out of the class of models we work with). Thus, in online analysis, the performance measure is relative, and it is not necessary to assume any assumptions on the way that the instances were produced. See [25] for a discussion on the

differences, analogies and conversions between batch and online analysis.

Passive vs. Active Learning

In *passive* learning, the learner can only observe the instances given by the teacher. On the other hand, in *active learning* models the learner has the opportunity to guide the learning process in some way. Different active learning models differ in the way the learner can affect the learning process (i.e. the instances s/he will get). For example in *membership query* [6] model the learner can generate instance by him or herself and ask the teacher for their label. This may be problematic in some domains, as the learner can generate meaningless instances [68]. Another example is the *selective sampling* model [21]. Here the learner observes the instances given by the teacher and decides which of them s/he wants to get the label for. This model is most useful when it is cheap to get instances, but it is expensive to label them. It can be shown that under some conditions active learning can reduce exponentially the number of labels required to ensure a given level of accuracy (compared to passive learning) ([34]). See [38] for an overview on active learning.

Reinforcement Learning

In many real life situations things are not so simple. The game is not just to predict some label. Sometimes you are required to choose from a set of many actions, and you may be rewarded or be penalized for your action and the action may affect the environment. The *Reinforcement learning* (see e.g. [59]) model tries to capture the above observations and to present a more "complete" model. It is assumed that the world has a state. At each step the learner (aka agent in this context) takes an action which is chosen out of a set of possible actions. The action may affect the world state. In each step the agent gets a reward which depends on the world state and the chosen action. The agent's goal is to maximize the cumulative reward or *discounted* reward, where discounted reward put more emphasis on the reward that will be given in the near future. The world's current state may be known to the agent (the "fully observed" model) or hidden (the "partially observed" model). When the world's state is hidden, the agent can only observe some noisy measurement of it. Under some Markovian assumptions it is possible to find a good strategy efficiently.

The reinforcement learning model is popular in the context of robotics, where people try to create a robot that can behave reasonably in a changing environment.

Inductive transfer

Another observation about human learning is that a human does not learn isolated tasks, but rather learns many tasks in parallel or sequentially. *Inductive transfer* [10, 110, 16] is a learning model that captures this kind of characteristic of human learning. Thus, in inductive transfer (a.k.a *learning to learn*, *transfer learning* or *task2task*) one tries to use knowledge gained in previous tasks in order to enhance performance on the current task. In this context, the main question is the kind of knowledge we can transfer between tasks. This is not trivial, as it is not clear how we can use, for an example, images which are labeled "cat or not cat" in order to learn to classify images to "table or not table". One option, which is very popular, is to share the knowledge about the representation, i.e. to assume that the same kind of features or distance measure is good for all the classes. For example, [110] uses the previous tasks to learn a distance measure between instances. This is done by constructing a training set of *pairs of instances*, where a pair is labeled 1 if and only if we can be sure that they have the same label (i.e. they are from the same task and have a positive label for the class of interest in this task). This make sense as the goal is to find a distance measure for which instances of the same class are close and instances of different classes are well apart. They use Neural Network to learn the distance measure from the pairs' training set. Another option, which interests us, is to share the knowledge on which features are useful. We elaborate on this in chapter 5, where we show how our new approach to feature selection can be applied to inductive transfer. Several authors have noted that sometimes transferring knowledge can hurt performance on the target problem. Two problems are considered as related if the transfer improves the performance and non-related otherwise. However, it is clear that this notion is not well defined as the behavior depends on the kind of information we choose to transfer, or more generally on the learning algorithm [16].

1.2 Feature Selection

In many supervised learning tasks the input is represented by a very large number of features, many of which are not needed for predicting the labels. Feature selection (variously known as *subset selection*, *attribute selection* or *variable selection*) is the task of choosing a small subset of features that is sufficient to predict the target labels well. The four main reasons to use feature selection are:

1. **Reduced computational complexity.** Feature selection reduces the computational complexity of learning and prediction algorithms. Many popular learning algorithms become computationally intractable in the presence of huge numbers of features, both in the training step and in the prediction step. A preceding step of feature selection can solve the problem.

2. **Economy.** Feature selection saves on the cost of measuring non selected features. Once we have found a small set of features that allows good prediction of the labels, we do not have to measure the rest of the features any more. Thus, in the prediction stage we only have to measure a few features for each instance. Imagine that we want to predict whether a patient has a specific disease using the results of medical checks. There are a huge number of possible medical checks that might be predictive; let's say that there are 1000 potential checks and that each of them costs ten dollars to perform. If we can find a subset of only 10 features that allows good performance, it saves a lot of money, and may turn the whole thing from an infeasible into a feasible procedure.

3. **Improved accuracy.** In many situations, feature selection can also enhance prediction accuracy by improving the signal to noise ratio. Even state-of-the-art learning algorithms cannot overcome the presence of a huge number of irrelevant or weakly relevant features. On the other hand once a small set of good features has been found, even very simple learning algorithms may yield good performance. Thus, in such situations, an initial step of feature selection may improve accuracy dramatically.

4. **Problem understanding.** Another benefit of feature selection is that the identity of

the selected features can provide insights into the nature of the problem at hand. This is significant since in many cases the ability to point out the most informative features is more important than the ability to make a good prediction in itself. Imagine we are trying to predict whether a person has a specific type of cancer using gene expression data. While we can know whether the individual is sick or not in other ways, the identity of the genes which are informative for prediction may give us a clue to the disease mechanism involved, and help in developing drugs.

Thus *feature selection* is an important step in efficient learning of large multi-featured data sets. Regarding reasons 2 and 4 above, feature selection has an advantage over other general dimensionality reduction methods. Computing general functions of all the input features means that we must always measure all the features first, even if in the end we calculate only a few functions of them. In many problems, different features vary in terms of their nature and their units (e.g. body temperature in Celsius, yearly income in dollars). In these cases feature selection is the natural formulation and it enables a better interpretation of the results, as the meaning of the combination of the features is not very clear.

Many works define the task of feature selection is "detecting which features are relevant and which are irrelevant". In this context we need to define *relevancy*. This is not straightforward, because the effect of a feature depends on which other features we have selected. Almuallim and Dietterich [4] provide a definition of relevance for the binary noise-free case. They define a feature to be relevant if it appears in any logical formula that describes the target concept. Gennari et al. [37] suggest a probabilistic definition of relevancy, which defines a feature to be relevant if it affects the conditional distribution of the labels. John et al [55, 62] define the notion of strong and weak relevance of a feature. Roughly speaking, a strongly relevant feature is a useful feature that cannot be replaced by any other feature (or set of features) whereas a weakly relevant feature is a feature which is useful, but can be replaced by another feature (or set of features). They also show that, in general, relevance does not imply optimality and that optimality does not imply relevance. Moreover, as we show in chapter 2, even if a feature is relevant and useful, it may detract in the presence of other features. Thus, when the concern is prediction accuracy, the best definition of the

task of feature selection as "choosing a small subset of features that allows for good prediction of the target labels". When the goal is "problem understanding" (reason 4 above), the relevancy of each feature alone might be important. However, in real life problems features are rarely completely irrelevant and thus it might be better to inquire about the importance of each feature. Cohen et al. [20] suggested the Shapley value of a feature as a measure of its importance. The Shapley value is a quantity taken from game theory; where it is used to measure the importance of a player in a cooperative game.

Below I review the main feature selection paradigms and some of the immense number of selection algorithms that have been presented in the past. However, a complete review of feature selection methods is beyond the scope of this thesis. See [45] or [43] for a comprehensive overview of feature selection methodologies. For a review of (linear) feature selection for regression see [82].

1.2.1 Common paradigms for feature selection

Many different algorithms for the task of feature selection have been suggested over the last few decades both in the Statistics and in the Learning community. Different algorithms present different conceptual frameworks. However, in the most common selection paradigm an **evaluation function** is used to assign scores to subsets of features and a **search algorithm** is used to search for a subset with a high score. See figure 1.1 for an illustration of this paradigm. Different selection methods can be different both in the choice of evaluation function and the search method. The evaluation function can be based on the performance of a specific predictor (*wrapper* model, [62]) or on some general (typically cheaper to compute) relevance measure of the features to the prediction (*filter* model, [62]). The evaluation function is not necessarily a "black box" and in many cases the search method can use information on the evaluation function in order to perform an efficient search.

In most common wrappers, the quality of a given set of features is evaluated by testing the predictor performance on a validation set. The main advantage of a wrapper is that you optimize what really interest you - the predictor accuracy. The main drawback of such methods is their computational deficiency that limits the number of sets that can be

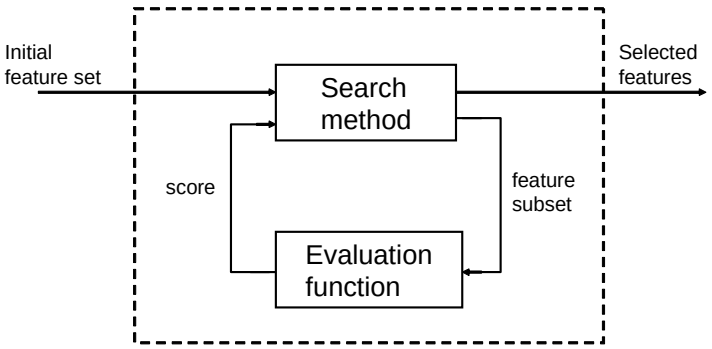

Figure 1.1: The most common selection paradigm: An evaluation function is used to evaluate the quality of subsets of features and a search engine is used for finding a subset with high score.

evaluated. The computational complexity of wrappers is high since we have to re-train the predictor in each step. Common filters use quantities like conditional Variance (of the features given the labels), Correlation Coefficients or Mutual Information as a general measure. In any case (wrapper or filter), an exhaustive search over all feature sets is generally intractable due to the exponentially large number of possible sets. Therefore, search methods are employed which apply a variety of heuristics. Two classic search methods are [78]:

1. **Forward selection:** start with an empty set of features and greedily add features one at a time. In each step, the feature that produces the larger increase of the evaluation function (with respect to the value of the current set) is added.

2. **Backward Elimination:** Start with a set of features that contains all the features and greedily remove features one at a time. In each step the feature whose removal results in the larger increase (or smaller decrease) in the evaluation function value is removed.

Backward elimination has the advantage that when it evaluates the contribution of a feature it takes into consideration all the other potential features. On the other hand, in forward selection, a feature that was added at one point can become useless later on and vice versa. However, since evaluating small sets of features is usually faster than evaluating large sets, forward selection is much faster when we are looking for small number of features to select. Moreover, if the initial number of features is very large, backward elimination become infeasible. A combination of the two is also possible of course, and has been used in many works. Many other search methods such as stochastic hill climbing, random permutation [87] and genetic algorithms [48] can be used as search methods as well.

I turn now to review a few examples of some of the more famous feature selection algorithms. Almuallim and Dietterich [4] developed the FOCUS family of algorithms that performs an exhaustive search in the situation where both the features and labels are binary (or at least have small number of possible values). All feature subsets of increasing size are evaluated, until a *sufficient* set is found. A set is *sufficient* if there are no conflicts, i.e. if there is no pair of instances with the same feature values but different labels. They also presented heuristic versions of FOCUS that look only at "promising" subsets, and thus make it feasible when the number of features is larger. Kira and Rendell [61] presented the RELIEF algorithm, a distance based filter that uses a 1-Nearest Neighbor (1-NN) based evaluation function (see chapter 3 for more details). Koller and Shamai [64] proposed a selection algorithm that uses a Markov blanket [89]. Their algorithm uses backward elimination, where a feature is removed if it is possible to find a Markov blanket for it, or in other words, if it possible to find a set of features such that the feature is independent of the labels given this set. Pfahringer [91] presented a selection method which is based on the Minimum Description Length principle [97]. Vafaie and De Jong [115], used the average Euclidean distance between instances in different classes as an evaluation function and genetic algorithms [48] as a search method.

The above algorithms are all filters. The following are examples of wrappers. Aha and Bankert [1] used a wrapper approach for instance-based learning algorithms (instance-based algorithms [2] are extensions of 1-Nearest Neighbor [32, 22]) combined with a "beam" search strategy, using a kind of backward elimination. Instead of starting with an empty or a full

set, they select a fixed number of subsets (with a bias toward small sets) and start with the best one among them. Then they apply backward elimination while maintaining a queue of features which is ordered according to the contribution of a feature the last time it was evaluated. Skalak [106] used a wrapper for the same kind of classifiers, but with random permutation instead of deterministic search. Caruana and Freitag [17] developed a wrapper for decision trees [93, 94] with a search which is based on adding and removing features randomly, but also removing in each step all the features which were not included in the induced tree. Bala et al. [8] and Cherkauer and Shavlik [19] used a wrapper for decision trees together with genetic algorithms as search strategies. Weston et al. [120] developed a sophisticated wrapper for SVMs. Their algorithm minimizes a bound on the error of SVM and uses gradient descent to fasten the search, but it still has to solve the SVM quadratic optimization problem in each step, and in this sense it is a wrapper.

Individual feature ranking

Other feature selection methods simply rank individual features by assigning a score to each feature independently. These methods are usually very fast, but inevitably fail in situations where only a combined set of features is predictive of the target function. Two of the most common rankers are:

1. Infogain [93], which assigns to each feature the Mutual Information between the feature value and the label, considering both the feature and the label as random variables and using any method to estimate the joint probability. Formally, let P be an estimation of the joint probability for a feature, i.e., P_{ij} represents the probability that both f equal its i's possible value and the label is j, then we get the following formula for the infogain of the feature f:

$$IG(f) = \sum_{i,j} P_{ij} log \frac{P_{ij}}{\left(\sum_i P_{ij} \sum_j P_{ij}\right)}$$

   The most common way to estimate P is by simply counting the percent of the instances that present each value pair ("the empiric distribution") with some kind of zero correction (e.g., adding a small constant to each value, and re-normalizing).

2. Correlation Coefficients (corrcoef), which assign each feature the Pearson correlation between the vector of values the feature got in the training set (v) and the vector of the labels (y) [92] i.e.,

$$CC(f) = \frac{E(v - E(y)(y - E(y)))}{\sqrt{Var(v)}Var(y)}$$

where the expectation and the variance are estimated empirically of course.

Corrcoef has the advantage (over infogain) that it can be used with continuous values (of features or the label) without any need of quantization. On the other hand, Infogain has the advantage that it does not assume any geometric meaning of the values, and thus can work for any kind of features and label, even if they are not numerical values.

Many other individual feature rankers have been proposed. Among them are methods which use direct calculations on the true-positive, true-negative, false-negative and false-positive which are obtained using the given features. Other methods use different kinds of distance measures between distributions to calculate the distance between the empirical joint distribution and the expected one (assuming independency between the feature and the label). Infogain is an example of such a method, with Kullback & Leibler divergence (D_{KL}) as the "distance measure". Other methods use statistical tests including the chi-square ([73, 74]), t-test ([107]) or the ratio of between-class variance to within-class variance, known as the Fisher criterion ([29]). See [45], chapter 3 for a detailed overview of individual feature ranking methods.

Embedded approach

The term *Embedded methods* is usually used to describe selection which is done automatically by the learning algorithm. Note that almost any wrapper can be considered as an embedded method, because selection and learning are interleaved. Decision tree [93, 94] learning can also be considered to be an embedded method, as the construction of the tree and the selection of the features are interleaved, but the selection of the feature in each step is usually done by a simple filter or ranker. Other authors use the term "embedded method" to refer only to methods where the selection is done by the learning algorithm implicitly. A prominent example of such a method is the L1-SVM [86].

1.2.2 The Biological/Neuroscience Rationale

Feature selection is carried out by many biological systems. A sensory system is exposed to an infinite number of possible features, but has to select only some of them. For example, the number of potential details in a visual scene is infinite and only some (albeit a large number) of them are perceived, even in the low level of the reticulum and V1. Moreover, many sensory systems perform feature selection as part of the processing of the data. For example, higher levels in the visual path have to choose from among the huge number of features that the receptive fields in V1 produce. Ullman et al. [114] state that the problem of feature selection is a fundamental question in the study of visual processing. They show that intermediate complexity (IC) features are the optimal choice for classification. Serre et al. [101] show that selecting object class-specific features improves object recognition ability in a model of biological vision. This suggests that feature selection in biological systems is not something that done only once, but rather is a dynamic process which is a crucial part of the never-ending learning process of a living creature.

Another example is selecting relevant spectral cues in the auditory system. When a sound is perceived by the ear, the level of modulation of each frequency depends on the direction the sound came from. These changes are called spectral cues and the auditory system of a human uses them to determine the elevation of the sound source [47]. Feature selection occurs in the olfactory pathway as well. As described by Mori et al. [83], the mammalian olfactory sensory neurons detect many different odor molecules and send information (using their axons) to the olfactory bulb (OB). Each glomerulus in the OB is connected to sensory neurons of the same type and can be considered as representing a feature. Lateral inhibition among glomerular modules is a kind of feature selection mechanism. Perera et al. [90] developed a feature selection method which is inspired by the olfactory system. This is an example of something that happens frequently, where researchers adopt ideas from the biological system when developing computerized algorithms.

Feature Selection in service of biological research

Nowadays biological experiments produce huge amounts of data that cannot be analyzed without the aid of computerized tools. Feature selection is a useful tool for data analysis. For example, assume that we want to measure the activity of many neurons during a given task and then produce a set of features that represents many different kinds of measurements of the neurons' activity. Now we can use feature selection to find which kind of neuron and which kind of activity characteristic are most informative for predicting the behavior. We demonstrate such use of feature selection in chapter 4, where we use feature selection on data recorded from the motor cortex of a behaving monkey.

1.3 Notation

Although specific notation is introduced in the chapters as needed, the following general notation is used throughout this thesis. Vectors in $\mathbb{R}^N$ are denoted by boldface small letters (e.g. $\mathbf{x}$, $\mathbf{w}$). Scalars are denoted by small letters (e.g. x, y). The i'th element of a vector $\mathbf{x}$ is denoted by x_i. log is the base 2 logarithm and ln is the natural logarithm. m is the number of training instances. N denotes the number of all potential features while n denotes the number of selected features. Subsets of features are denoted by F and a specific feature is usually denoted by f. A labeled training set of size m is denoted by S^m (or by S when m is clear from the context). The table below summarizes the main notation we keep along the thesis for a quick reference.

Notation	short description
m	number of training instances
N	total number of features
n	number of selected features
F	set of features
f	a feature
$\mathbf{w}$	a weight vector over the features
S^m	training set of size m
c	the target classification rule
h	a hypothesis
$\mathcal{D}$	a distribution over instances (and labels) space
$\hat{\mathrm{er}}_S^{\gamma}(h)$	γ- sensitive training error of hypothesis h
$\mathrm{er}_{\mathcal{D}}(h)$	generalization error of hypothesis h w.r.t. $\mathcal{D}$

Chapter 2

Is Feature Selection Still Necessary?[1]

As someone who has researched feature selection for the last several years, I have often heard people doubt its relevancy with the following argument: "a good learning algorithm should know to handle irrelevant or redundant features correctly and thus an artificial split of the learning process into two stages imposes excessive limits and can only impede performance". There are several good rebuttals to this claim. First, as explained in section 1.2, improved classification accuracy is not the only rationale for feature selection. Feature selection is also motivated by improved computational complexity, economy and problem understanding. Second, research on feature selection is a part of the broader issue of data representation. However, to provide a complete answer, in this chapter we directly discuss the issue of whether there is a need to do feature selection solely for improving classification accuracy (ignoring other considerations), even when using current state-of-art learning algorithms.

Obviously, if the true statistical model is known, or if the sample is unlimited (and the number of features is finite), any additional feature can only improve accuracy. However, when the training set is finite, additional features can degrade the performance of many

[1] The results presented in this chapter were first presented as a chapter in the book "Subspace, Latent Structure and Feature Selection", edited by Saunders, C. and Grobelnik, M. and Gunn, S. and Shawe-Taylor, J. [84]

classifiers, even when all the features are statistically independent and carry information on the label. This phenomenon is sometimes called "the peaking phenomenon" and was already demonstrated more than three decades ago by [54, 113, 96] and in other works (see references there) on the classification problem of two Gaussian-distributed classes with equal covariance matrices (LDA). Recently, [49] analyzed this phenomenon for the case where the covariance matrices are different (QDA); however, this analysis is limited to the case where all the features have equal contributions. On the other hand [53] showed that, in the Bayesian setting, the optimal Bayes classifier can only benefit from using additional features. However, using the optimal Bayes classifier is usually not practical due to its computational cost and the fact that the true *prior* over the classifiers is not known. In their discussion, [53] raised the problem of designing classification algorithms which are computationally efficient and robust with respect to the feature space. Now, three decades later, it is worth inquiring whether today's state-of-the-art classifiers, such as *Support Vector Machine* (SVM) [14], achieve this goal.

Here we re-visit the two-Gaussian classification problem, and concentrate on a simple setting of two spherical Gaussians. We present a new, simple analysis of the optimal number of features as a function of the training set size. We consider the maximum likelihood estimation as the underlying classification rule. We analyze its error as function of the number of features and number of training instances, and show that while the error may be as bad as chance when using too many features, it approaches the optimal error if we chose the number of features wisely. We also explicitly find the optimal number of features as a function of the training set size for a few specific examples. We test SVM empirically in this setting and show that its performance matches the predictions of the analysis. This suggests that feature selection is still a crucial component in designing an accurate classifier, even when modern discriminative classifiers are used, and even if computational constraints or measuring costs are not an issue.

2.1 Problem Setting and Notation

First, let us introduce some notation. Vectors in $\mathbb{R}^N$ are denoted by bold face lower case letter (e.g. $\mathbf{x}, \boldsymbol{\mu}$) and the j'th coordinate of a vector $\mathbf{x}$ is denoted by x_j. We denote the restriction of a vector $\mathbf{x}$ to the first n coordinates by $\mathbf{x}_n$.

Assume that we have two classes in $\mathbb{R}^N$, labeled $+1$ and -1. The distribution of the points in the positive class is $Normal\left(\boldsymbol{\mu}, \Sigma = I\right)$ and the distribution of the points in the negative class is $Normal\left(-\boldsymbol{\mu}, \Sigma = I\right)$, where $\boldsymbol{\mu} \in R^N$, and I is the $N \times N$ unit matrix. To simplify notation we assume, without loss of generality, that the coordinates of $\boldsymbol{\mu}$ are ordered in descending order of their absolute value and that $\mu_1 \neq 0$; thus if we choose to use only $n < N$ features, the best choice would be the first n coordinates. The optimal classifier, i.e., the one that achieves the maximal accuracy, is $h\left(\mathbf{x}\right) = \text{sign}\left(\boldsymbol{\mu} \cdot \mathbf{x}\right)$. If we are restricted to using only the first n features, the optimal classifier is $h\left(\mathbf{x}, n\right) = \text{sign}\left(\boldsymbol{\mu}_n \cdot \mathbf{x}_n\right)$.

We assume that the model is known to the learner (i.e. that there two antipode spherical Gaussian classes) , but the model parameter, $\boldsymbol{\mu}$, is not known. Thus, in order to analyze this setting we have to consider a specific way to estimate $\boldsymbol{\mu}$ from a training sample $S^m = \left\{\mathbf{x}^i, y^i\right\}_{i=1}^m$, where $\mathbf{x}^i \in R^N$ and $y^i \in \{+1, -1\}$ is the label of $\mathbf{x}^i$. We consider the maximum likelihood estimator of $\boldsymbol{\mu}$:

$$\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\mu}}\left(S^m\right) = \frac{1}{m}\sum_{i=1}^{m} y^i \mathbf{x}^i$$

Thus the estimated classifier is $\hat{h}\left(\mathbf{x}\right) = \text{sign}\left(\hat{\boldsymbol{\mu}} \cdot \mathbf{x}\right)$. For a given $\hat{\boldsymbol{\mu}}$ and number of features n, we look on the generalization error of this classifier:

$$\text{error}\left(\hat{\boldsymbol{\mu}}, n\right) = P\left(\text{sign}\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n\right) \neq y\right)$$

where y is the true label of $\mathbf{x}$. This error depends on the training set. Thus, for a given training set size m, we are interested in the average error over all the possible choices of a sample of size m:

$$\text{error}\left(m, n\right) = E_{S^m}\text{error}\left(\hat{\boldsymbol{\mu}}\left(S_m\right), n\right) \tag{2.1}$$

We look for the optimal number of features, i.e. the value of n that minimizes this error:

$$n_{opt} = \arg\min_n \text{error}\left(m, n\right)$$

2.2 Analysis

For a given $\hat{\boldsymbol{\mu}}$, and n the dot product $\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n$ is a *Normal* random variable on its own and therefore the generalization error can be explicitly written as (using the symmetry of this setting):

$$\text{error}\left(\hat{\boldsymbol{\mu}}, n\right) = P\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n < 0| + 1\right) = \Phi\left(-\frac{E_{\mathbf{x}}\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n\right)}{\sqrt{V_{\mathbf{x}}\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n\right)}}\right) \tag{2.2}$$

here Φ is the Gaussian cumulative density function: $\Phi\left(a\right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-\frac{1}{2}z^2} dz$. We denote by $E_{\mathbf{x}}$ and $V_{\mathbf{x}}$ expectation and variance with respect to the true distribution of $\mathbf{x}$.

For a given number of features n, and a given sample S, we have

$$E_{\mathbf{x}}\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n\right) = \hat{\boldsymbol{\mu}}_n \cdot \boldsymbol{\mu}_n = \sum_{j=1}^{n} \hat{\mu}_j \mu_j$$

and

$$V_{\mathbf{x}}\left(\hat{\boldsymbol{\mu}}_n \cdot \mathbf{x}_n\right) = \sum_{j=1}^{n} \hat{\mu}_j^2 V_{\mathbf{x}}\left(x_j\right) = \sum_{j=1}^{n} \hat{\mu}_j^2$$

substituting in equation (2.2) we get:

$$\text{error}\left(\hat{\boldsymbol{\mu}}, n\right) = \Phi\left(-\frac{\sum_{j=1}^{n} \hat{\mu}_j \mu_j}{\sqrt{\sum_{j=1}^{n} \hat{\mu}_j^2}}\right) \tag{2.3}$$

Now, for a given training set size m, We want to find n that minimizes the average error term $E_{S^m}\left(\text{error}\left(\hat{\boldsymbol{\mu}}, n\right)\right)$, but instead we look for n that minimizes an approximation of the average error:

$$n_{opt} = \arg\min_{n} \Phi\left(-\frac{E_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j \mu_j\right)}{\sqrt{E_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j^2\right)}}\right) \tag{2.4}$$

We first have to justify why the above term approximates the average error. We look at the variance of the relevant terms (the numerator and the term in the square root in the denominator). $\hat{\mu}_j$ is a Normal random variable with expectation μ_j and variance $1/m$, thus

$$V_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j \mu_j\right) = \sum_{j=1}^{n} \mu_j^2 V_{S^m}\left(\hat{\mu}_j\right) = \frac{1}{m} \sum_{j=1}^{n} \mu_j^2 \xrightarrow[m \to \infty]{} 0 \tag{2.5}$$

$$V_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j^2\right) = \frac{2n}{m^2} + \frac{4}{m} \sum_{\alpha=1}^{n} \mu_j^2 \xrightarrow[m \to \infty]{} 0 \tag{2.6}$$

where in the last equality we used the fact that if $Z \sim Normal\left(\mu, \sigma^2\right)$, then $V\left(Z^2\right) = 2\sigma^4 + 4\sigma^2\mu^2$. We also note that:

$$\begin{aligned} E_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j^2\right) &= \sum_{j=1}^{n}\left(V_{S^m}\left(\hat{\mu}_j\right) + E\left(\hat{\mu}_j\right)^2\right) \\ &= \sum_{j=1}^{n}\left(\mu_j^2 + \frac{1}{m}\right) \xrightarrow[m\to\infty]{} \sum_{j=1}^{n} \mu_j^2 > 0 \end{aligned}$$

therefore the denominator is not zero for any value of m (including the limit $m \to \infty$). Combining this with (2.5) and (2.6) and recalling that the derivative of Φ is bounded by 1, we conclude that at least for a large enough m, it is a good approximation to move the expectation inside the error term. However, we need to be careful with taking m to ∞, as we know that the problem becomes trivial when the model is known, i.e., when n is finite and $m \to \infty$. Additionally, in our analysis we also want to take the limit $n \to \infty$, and thus we should be careful with the order of taking the limits. In the next section we see that the case of bounded $\sum_{j=1}^{n} \mu_j^2$ is of special interest. In this situation, for any $\epsilon > 0$, there are finite m_0 and n_0 such that the gap between the true error and the approximation is less than ϵ with a high probability for any (m, n) that satisfies $m > m_0$ and $n > n_0$ (including the limit $n \to \infty$). Thus, for any $m > m_0$, the analysis of the optimal number of features n_{opt} (under the constraint $n > n_0$) as a function of m using the approximation is valid. When we are interested in showing that too many features can hurt the performance, the constraint $n > n_0$ is not problematic. Moreover, figure 2.1 shows numerically that for various choices of $\boldsymbol{\mu}$ (including choices such that $\sum_{j=1}^{n} \mu_j^2 \xrightarrow[n\to\infty]{} \infty$), moving the expectation inside the error term is indeed justified, even for small value of m.

Now we turn to finding the n that minimizes (2.4), as function of the training set size m. This is equivalent to finding n that maximizes

$$f(n,m) = \frac{E_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j \mu_j\right)}{\sqrt{E_{S^m}\left(\sum_{j=1}^{n} \hat{\mu}_j^2\right)}} = \frac{\sum_{j=1}^{n} \mu_j^2}{\sqrt{\sum_{j=1}^{n}\left(\mu_j^2 + \frac{1}{m}\right)}} = \frac{\sum_{j=1}^{n} \mu_j^2}{\sqrt{\frac{n}{m} + \sum_{j=1}^{n} \mu_j^2}} \tag{2.7}$$

2.2.1 Observations on The Optimal Number of Features

First, we can see that, for any finite n, when $m \to \infty$, $f\left(n,m\right)$ reduces to $\sqrt{\sum_{j=1}^{n} \mu_j^2}$ and thus, as expected, using all the features maximizes it. It is also clear that adding a

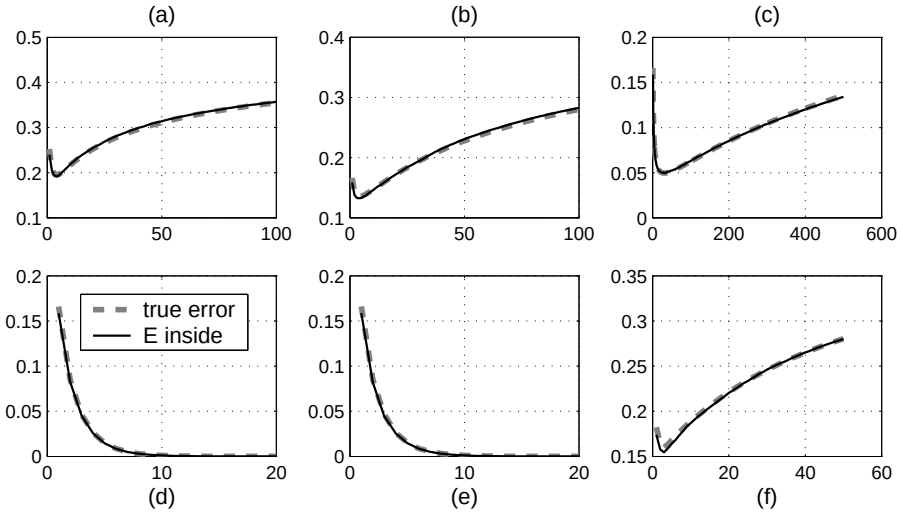

Figure 2.1: **Numerical justification for using equation 2.4**. The true and approximate error ("E-inside") as function of the number of features used for different choices of $\boldsymbol{\mu}$: (a) $\mu_j = 1/\sqrt{2^j}$, (b) $\mu_j = 1/j$, (c) $\mu_j = 1/\sqrt{j}$, (d) $\mu_j = 1$, (e) $\mu_j = rand$ (sorted) and (f) $\mu_j = rand/j$ (sorted). The training set size here is $m = 16$. The true error was estimated by averaging over 200 repeats. The approximation with the expectation inside error term is very close to the actual error term (2.1), even for small training set ($m = 16$).

completely non-informative feature μ_j ($\mu_j = 0$) will decrease $f(n, m)$. We can also formulate a sufficient condition for the situation where using too many features is harmful, and thus feature selection can improve the accuracy dramatically:

Statement 1 *For the above setting, if the partial sum series* $s_n = \sum_{j=1}^{n} \mu_j^2 < \infty$ *then for any finite* m *the error of the ML classifier approaches to* $1/2$ *when* $n \to \infty$ *and there is* $n_0 = n_0(m) < \infty$ *such that selecting the first* n_0 *features is superior to selecting* k *features for any* $k > n_0$.

Proof. Denote $\lim_{n\to\infty} s_n = s < \infty$, then the numerator of (2.7) approaches s, while the denominator approaches ∞, thus $f(n, m) \xrightarrow[n\to\infty]{} 0$ and the error $\Phi(-f(n, m)) \to 1/2$. On the other hand $f(n, m) > 0$ for any finite n, thus there exists n_0 such that $f(n_0, m) > f(k, m)$ for any $k > n_0$.

□

Note that it is not possible to replace the condition in the above statement by $\mu_j \xrightarrow[j\to\infty]{} 0$ (see example 2.4). The following consistency statement gives a sufficient condition on the number of features that ensure asymptotic optimal error:

Statement 2 *For the above setting, if we use a number of features $n = n(m)$ that satisfies (1) $n \xrightarrow[m\to\infty]{} \infty$ and (2) $\frac{n}{m} \xrightarrow[m\to\infty]{} 0$ then the error of the ML estimator approaches to the optimal possible error (i.e. the error when μ is known and we use all the features) when $m \to \infty$. Additionally, if $\sum_{j=1}^{n} \mu_j \xrightarrow[n\to\infty]{} \infty$, condition (2) above can be replaced with $\frac{n}{m} \xrightarrow[m\to\infty]{} c$, where c is any finite constant.*

Proof. Recalling that the optimal possible error is given by $\Phi\left(-\sqrt{\sum_{j=1}^{\infty} \mu_j}\right)$, the statement follows directly from equation 2.7.

□

Corollary 2.1 *Using the optimal number of features ensure consistency (in the sense of the above statement).*

Note as well that the effect of adding a feature depends not only on its value, but also on the current value of the numerator and the denominator. In other words, the decision whether to add a feature depends on the properties of the features we have added so far. This may be surprising, as the features here are statistically independent. This apparent dependency comes intuitively from the signal-to-noise ratio of the new feature to the existing ones.

Another observation is that if all the features are equal, i.e. $\mu_j = c$ where c is a constant,

$$f(n,m) = \frac{c^2}{\sqrt{\frac{1}{m} + c^2}} \sqrt{n}$$

and thus using all the features is always optimal. In this respect our situation is different from the one analyzed by [54]. They considered Anderson's W classification statistic [5] for the setting of two Gaussians with same covariance matrix, but both the mean and the covariance were not known. For this setting they show that when all the features have equal contributions, it is optimal to use $m - 1$ features (where m is the number of training examples).

2.3 Specific Choices of $\boldsymbol{\mu}$

Now we find the optimal number of features for a few specific choices of $\boldsymbol{\mu}$.

Example 2.1 *Let* $\mu_j = \frac{1}{\sqrt{2^j}}$, *i.e.,* $\boldsymbol{\mu} = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{4}}, \frac{1}{\sqrt{8}}, \ldots, \frac{1}{\sqrt{2^N}}\right)$, *thus* $\|\boldsymbol{\mu}\| \xrightarrow[n\to\infty]{} 1$. *An illustration of the density functions for this case is given in figure 2.2(a). Substituting this* $\boldsymbol{\mu}$ *in (2.7), we obtain:*

$$f(n,m) = \frac{1 - \frac{1}{2}2^{-n}}{\sqrt{\frac{n}{m} + 1 - \frac{1}{2}2^{-n}}}$$

Taking the derivative with respect to n *and equating to zero we get*[2] *:*

$$-\frac{1}{4}2^{-2n}\ln 2 + 2^{-n}\ln 2\left(1 + \frac{n}{m} + \frac{1}{2m}\right) - \frac{1}{m} = 0$$

Assuming n *is large, we ignore the first term and get:*

$$m = \frac{1}{\ln 2}2^n - n - \frac{1}{2\ln 2}$$

Ignoring the last two lower order terms, we have:

$$\frac{1}{\sqrt{m}} \cong \frac{\ln 2}{\sqrt{2^n}} = (\ln 2)\,\mu_n$$

This makes sense as $1/\sqrt{m}$ *is the standard deviation of* μ_j, *so the above equation says that we only want to take features with a mean larger than the standard deviation. However, we should note that this is true* ***only in order of magnitude****. No matter how small the first feature is, it is worth to take it. Thus, we have no hope to be able to find optimal criterion of the form: take feature* j *only if* $\mu_j > f(\sqrt{m})$.

Example 2.2 *Let* $\mu_j = 1/j$. *An illustration of the density functions for this case is given in figure 2.2(b). Since* $\sum_{j=1}^{\infty}\left(\frac{1}{j}\right)^2 = \frac{\pi^2}{6}$,

$$\sum_{j=1}^{n}\left(\frac{1}{j}\right)^2 > \frac{\pi^2}{6} - \int_n^{\infty}\frac{1}{x^2}dx = \frac{\pi^2}{6} - \frac{1}{n}$$

$$\sum_{j=1}^{n}\left(\frac{1}{j}\right)^2 < \frac{\pi^2}{6} - \int_{n+1}^{\infty}\frac{1}{x^2}dx = \frac{\pi^2}{6} - \frac{1}{n+1}$$

[2] The variable n is an integer, but we can consider $f(n,m)$ to be defined for any real value.

thus the lower bound $\frac{\pi^2}{6}-\frac{1}{n}$ *approximates the finite sum up to* $\frac{1}{n}-\frac{1}{n+1}=\frac{1}{n(n+1)}$. *Substituting this lower bound in (2.7), we obtain:*

$$f(n,m) \cong \frac{\frac{\pi^2}{6}-\frac{1}{n}}{\sqrt{\frac{n}{m}+\frac{\pi^2}{6}-\frac{1}{n}}}$$

taking the derivative with respect to n *and equating to zero yields* $m \cong \frac{n^3\pi^2-18n^2}{n\pi^2-6}$ *and thus for a large* n *we obtain the power law* $m \cong n^2$, *or* $n \cong \sqrt{m} = m^{\frac{1}{2}}$, *and again we have* $\frac{1}{\sqrt{m}} \cong \frac{1}{n} = \mu_n$.

Example 2.3 *Let* $\mu_j = \frac{1}{\sqrt{j}}$, *i.e* $\boldsymbol{\mu} = \left(1, \frac{1}{\sqrt{2}}, \ldots \frac{1}{\sqrt{N}}\right)$. *An illustration of the separation between classes for a few choices of* n *is given in figure 2.2(c). Substituting* $\sum_{j=1}^{n}\mu_j^2 \cong \log n$ *in (2.7) we get:*

$$f(n,m) = \frac{\log n}{\sqrt{\frac{n}{m}+\log n}} \tag{2.8}$$

taking the derivative with respect to n *and equating to zero, we obtain:*

$$m \cong \frac{n(\log n - 2)}{\log n}$$

thus for a large n *we have* $m \cong n$, *and once again we have* $\frac{1}{\sqrt{m}} \cong \frac{1}{\sqrt{n}} = \mu_n$.

This example was already analyzed by Trunk ([113]) who showed that for any finite number of training examples, the error approaches one half when the number of features approaches ∞. *Here we get this results easily from equation (2.8), as for any finite* m, $f(n,m) \xrightarrow[n\to\infty]{} 0$, *thus the error term* $\Phi(-f(n,m))$ *approaches* $1/2$. *on the other hand, when* $\boldsymbol{\mu}$ *is known the error approaches zero when* n *increases, since* $\|\boldsymbol{\mu}\| \xrightarrow[n\to\infty]{} \infty$ *while the variance is fixed. Thus, from corollary 2.1 we know that by using* $m = n$ *the error approaches to zero when* m *grows, and our experiments show that it drops very fast (for* $m \sim 20$ *it is already below* 5% *and for* $m \sim 300$ *below* 1%*).*

Example 2.4 *In this example we show that the property* $\mu_j \xrightarrow[j\to\infty]{} 0$ *does not guarantee that feature selection can improve classification accuracy. Define* $s_n = \sum_{j=1}^{n}\mu_j^2$. *Let* μ_j *be such that* $s_n = n^\alpha$ *with* $\frac{1}{2} < \alpha < 1$. *Since* $\alpha < 1$, *it follows that indeed* $\mu_j \to 0$. *On the other hand, by substituting in (2.7) we get:*

$$f(n,m) = \frac{n^\alpha}{\sqrt{\frac{n}{m}+n^\alpha}} = \frac{n^{\alpha/2}}{\sqrt{\frac{n^{1-\alpha}}{m}+1}}$$

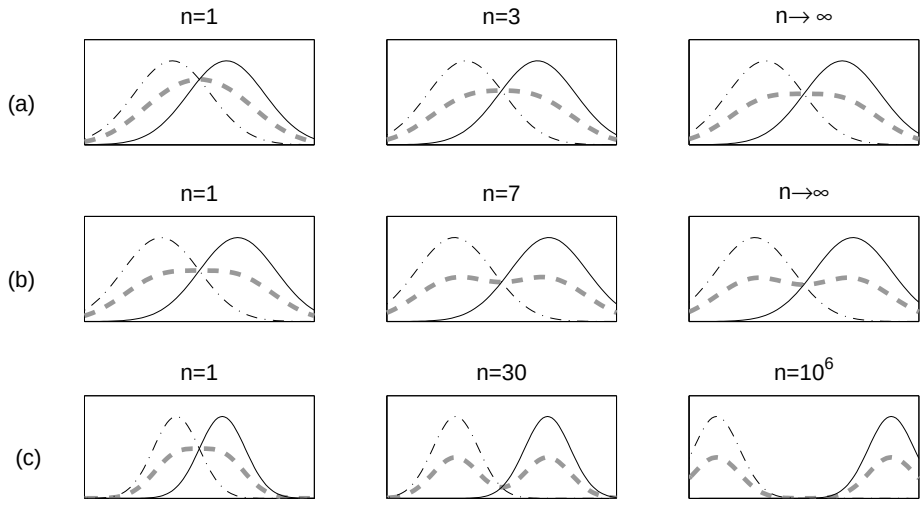

Figure 2.2: Illustration of the separation between classes for different number of features (n), for different choices of $\boldsymbol{\mu}$. The projection on the prefix of $\boldsymbol{\mu}$ of the density function of each class and the combined density (in gray) are shown. (a) for example 2.1, (b) for example 2.2 and (c) for example 2.3.

thus for $n > m^{\alpha-1}$,

$$f(n,m) \geq \frac{n^{\alpha/2}}{\sqrt{2\frac{n^{1-\alpha}}{m}}} = n^{\alpha-1/2}\sqrt{\frac{m}{2}}$$

and since $\alpha > 1/2$, *we have that* $f(n,m) \xrightarrow[n\to\infty]{} \infty$ *and thus using all the features is optimal. We can see that the intuitive relation between the value of the smallest feature we want to take and* $\frac{1}{\sqrt{m}}$ *that raised in the previous examples does not hold here. this demonstrate that thinking on this value as proportional to* $\frac{1}{\sqrt{m}}$ *may be misleading.*

2.4 SVM Performance

So far we have seen that the naive maximum likelihood classifier is impeded by using too many weak features, even when all the features are relevant and independent. However it is worth testing whether a modern and sophisticated classifier such as SVM that was designed to work in very high dimensional spaces can overcome the "peaking phenomenon". For this purpose we tested SVM on the above two Gaussian setting in the following way.

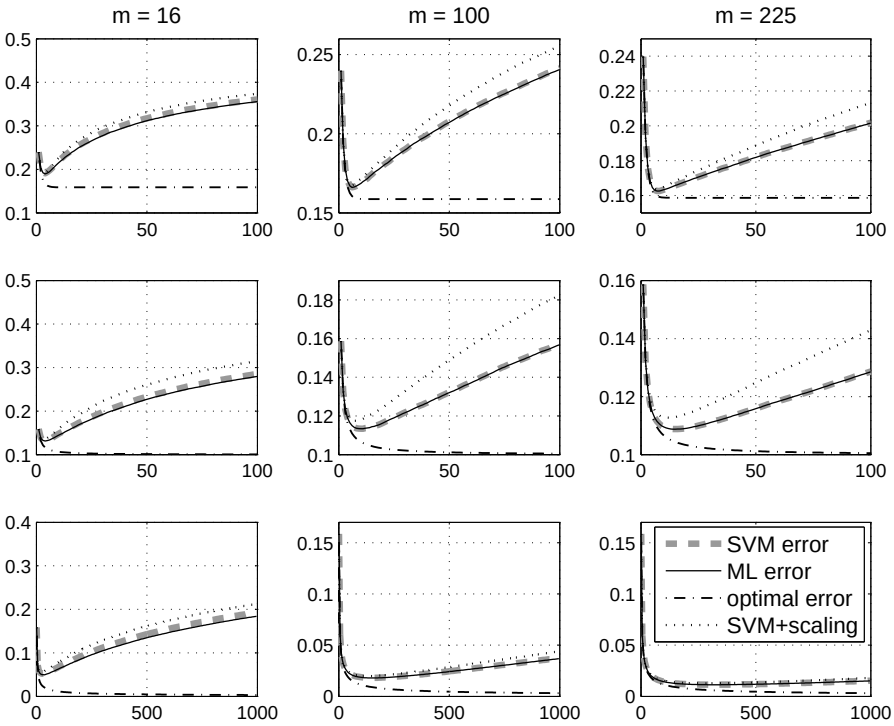

Figure 2.3: **The error of SVM as a function of the number of features used.** The top, middle and bottom rows correspond to μ_j equals $1/\sqrt{2^j}$, $1/j$ and $1/\sqrt{j}$ respectively. The columns correspond to training set sizes of 16, 100 and 225. The SVM error was estimated by averaging over 200 repeats. The graphs show that SVM produces the same qualitative behavior as the ML classifier we used in the analysis and that pre-scaling of the features strengthen the effect of too many features on SVM.

We generated a training set with 1000 features, and trained linear[3] SVM on this training set 1000 times, each time using a different number of features. Then we calculated the generalization error of each returned classifier analytically. The performance associated with a given number of features n is the generalization error achieved using n features, averaged over 200 repeats. We used the SVM tool-box by Gavin Cawley [18]. The parameter C was tuned manually to be $C = 0.0001$, the value which is favorable to SVM when all (1000) features are used. The results for the examples described in section 2.3 for three different choices of training set size are presented in figure 2.3.

We can see that in this setting SVM suffers from using too many features just like the

[3] The best classifier here is linear, thus linear kernel is expected to give best results.

maximum likelihood classifier[4]. On the other hand, it is clear that in other situations SVM does handle huge number of features well, otherwise it could not be used together with kernels. Therefore, in order to understand why SVM fails here, we need to determine in what way our high dimensional scenario is different from the one caused by using kernels. The assumption which underlies the usage of the large margin principle, namely that the density around the true separator is low, is violated in our first example (example 2.1), but not for the other two examples (see figure 2.2). Moreover, if we multiply the $\boldsymbol{\mu}$ of example 2.1 by 2, then the assumption holds and the qualitative behavior of SVM does not change. Hence this cannot be a major factor.

One might suggest that a simple pre-scaling of the features, such as dividing each features by an approximation of its standard deviation[5], might help SVM. Such normalization is useful in many problems, especially when different features are measured in different scales. However, in our specific setting it is not likely that such normalization can improve the accuracy, as it just suppress the more useful features. Indeed, our experiments shows (see figure 2.3, dotted lines) that the pre-scaling strengthen the effect of too many features on SVM.

One significant difference is that in our setting the features are statistically independent, whereas when the dimension is high due to the usage of kernels, the features are highly correlated. In other words, the use of kernels is equivalent to deterministic mapping of low dimensional space to a high dimensional space. Thus there are many features, but the actual dimension of the embedded manifold is low whereas in our setting the dimension is indeed high. We ran one initial experiment which supported the assumption that this difference was significant in causing the SVM behavior. We used SVM with a polynomial kernel of increasing degrees in the above setting with $\boldsymbol{\mu} = (1, 0)$, i.e. one relevant feature and one irrelevant feature. The results are shown in figure 2.4. The accuracy declines as the degree increases as expected (since the best model here is linear). However, the effect of the kernel degree on the difference in accuracy using one or both features is not significant, despite the

[4] This strengthen the observation in [46] (page 384) that additional noise features hurts SVM performance.

[5] Given the class, the standard deviation is the same for all the features (equals to 1), but the overall standard deviation is larger for the first features, as in the first coordinates the means of the two classes a more well apart.

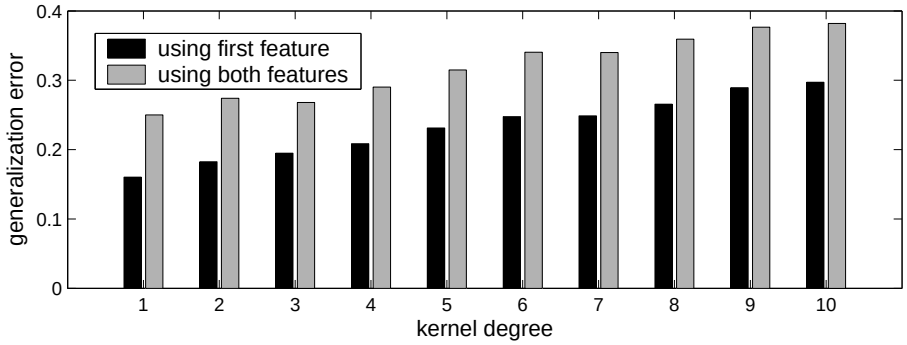

Figure 2.4: **The effect of the degree of a polynomial kernel on SVM error** $\boldsymbol{\mu} = (1, 0)$ and the training set size is $m = 20$. The results were averaged over 200 repeats. The effect of the kernel degree on the difference in accuracy using one or both features is not significant.

fact that the number of irrelevant features grows exponentially with the kernel degree[6].

2.5 Summary

We started this chapter by asking the question whether feature selection is still needed even when we only interested in classification accuracy, or maybe modern classifiers can handle the presence of a huge number of features well and only lose from the split of the learning process into two stages (feature selection and learning of a classifier). Using a simple setting of two spherical Gaussians it was shown that in some situations using too many features results in error as bad as chance while a wise choice of the number of features results in almost optimal error. We showed that the most prominent modern classifier, SVM, does not handle large numbers of weakly relevant features correctly and achieves suboptimal accuracy, much like a naive classifier. We suggest that the ability of SVM to work well in the presence of huge number of features may be restricted to cases where the underlying distribution is concentrated around a low dimensional manifold, which is the case when kernels are used. However, this issue should be further investigated. Thus we conclude that feature selection is indeed needed, even if nothing but classification accuracy interests us.

This chapter focused on feature selection, but the fundamental question addressed here

[6] When a polynomial kernel of degree k is used, only k features are relevant whereas $2^k - k$ are irrelevant.

is relevant to the broader question of general dimensionality reduction. In this setting one looks for any conversion of the data to a low dimensional subspace that preserves the relevant properties. Thus one should ask in what way the optimal dimension depends on the number of training instances. We expect that a similar trade-off can be found.

Chapter 3

Margin Based Feature Selection[1]

[1] The results presented in this chapter were first presented in our NIPS02 paper titled "Margin Analysis of the LVQ algorithm" [24], our ICML04 paper titled "Margin Based Feature Selection" [40] and a chapter titled "Large Margin Principles for Feature Selection" in the book "Feature extraction, foundations and applications", edited by Guyon, I. and Gunn, S. and Nikravesh, M. and Zadeh, L. [41]

After describing the main concepts in feature selection and the prime rationales, (see section 1.2 and chapter 2), this chapter discusses the ways feature selection is carried out. New methods of feature selection for classification based on the maximum margin principle are introduced. A margin [14, 100] is a geometric measure for evaluating the confidence of a classifier with respect to its decision. Margins already play a crucial role in current machine learning research. For instance, SVM [14] is a prominent large margin algorithm. The main novelty of the results presented in this chapter is the use of large margin principles for feature selection. Along the way we also present the new observation that there are two types of margins *(sample margin* and *hypothesis margin*), and define these two types of margins for the *One Nearest Neighbor* (1-NN) [32] algorithm. We also develop a theoretical reasoning for the 20 year old LVQ [63] algorithm (see section 3.8)

Throughout this chapter the 1-NN is used as the "study-case" predictor, but most of the results are relevant to other distance based classifiers (e.g. LVQ [63], SVM-RBF [14]) as well. To demonstrate this, we compare our algorithms to the R2W2 algorithm [120], which was specifically designed as a feature selection scheme for SVM. We show that even in this setting our algorithms compare favorably to all the contesters in the running..

The use of margins allows us to devise new feature selection algorithms as well as prove a PAC (Probably Approximately Correct) style generalization bound. The bound is on the generalization accuracy of 1-NN on a selected set of features, and guarantees good performance for any feature selection scheme which selects a small set of features while keeping the margin large. On the algorithmic side, we use a margin based criterion to measure the quality of sets of features. We present two new feature selection algorithms, *G-flip* and *Simba*, based on this criterion. The merits of these algorithms are demonstrated on various datasets. Finally, we study the *Relief* feature selection algorithm [61] in the large margin context. While *Relief* does not explicitly maximize any evaluation function, it is shown here that implicitly it maximizes the margin based evaluation function.

Before we present the main results, we briefly review the key concepts in 1-NN and margins.

3.1 Nearest Neighbor classifiers

Though fifty years have passed since the introduction of *One Nearest Neighbor* (1-NN) [32] it is still a popular algorithm. 1-NN is a simple and intuitive algorithm but at the same time may achieve state of the art results [105]. During the training process 1-NN simply stores the given training instances (also referred to as *prototypes*) without any processing. When it required to classify a new instance, it returns the label of the closest instance in the training set. Thus, 1-NN assumes that there is a way to measure distance between instances. If the instances are given as vectors in $\mathbb{R}^n$, we can simply use a standard norm (e.g. l_2 norm). However in many cases using more sophisticated distance measures that capture our knowledge of the problem may improve the results dramatically. It is also possible to *learn* a good distance measure from the training set itself. Many distance measures have been suggested for different problems in the past. One example of the usage of such sophisticated distance measure is the *tangent distance* that was employed by [104] for digit recognition. This distance measure takes into account the prior knowledge that the digit identity is invariant to small transformations such as rotation, translation etc. Using 1-NN together with this distance measure [104] achieved state-of-the-art results on the USPS data set [69].

However in large, high dimensional data sets it often becomes unworkable. One approach to cope with this computational problem is to approximate the nearest neighbor [52] using various techniques. An alternative approach is to choose a small data-set (aka prototypes) which represents the original training sample, and apply the nearest neighbor rule only with respect to this small data-set. This solution maintains the "spirit" of the original algorithm, while making it feasible. Moreover, it might improve the accuracy by reducing noise overfitting. We elaborate on these methods in section 3.8.

Another possible solution for noise overfitting is to use k-NN, which considers the k nearest neighbors and takes the majority (for classification) or the mean (for regression) over the labels of the k nearest neighbors. Here it is also possible to weigh the effect of each of the k neighbors according to its distance. 1-NN ensures error which is not more than twice the optimal possible error in the limit of infinite number of training instances

[22]. k-NN, when is used with $k = k(m)$ (where m is the number of training instances) that satisfies $k \xrightarrow[m\to\infty]{} \infty$ and $\frac{k}{m} \xrightarrow[m\to\infty]{} 0$ ensures optimal error in the limit $m \to \infty$ [28]. In this sense k-NN is consistent. See [112] for a recent review on Nearest Neighbor and ways to approximate it.

In the following we will use 1-NN as the "study case" predictor in developing our feature selection methods and will show how feature selection can improve the performance of 1-NN.

3.2 Margins

The concept of *margins* play an important role in current research in machine learning. A margin measures the confidence a classifier when making its predictions. Margins are used both for theoretic generalization bounds and as guidelines for algorithm design. Theoretically speaking, margins are a main component in data dependent bounds [103, 9] on the generalization error of a classifier. Data dependent bounds use the properties of a specific training data to give bounds which are tighter than the standard bounds that hold for any training set and depend only on its size (and the classifier properties). Since in many cases the margins give a better prediction of the generalization error than the training error itself, they can be used by learning algorithms as a better measure for a classifier quality. We can consider margins as a stronger and more delicate version of the training error. Two of the most prominent algorithms in the field, *Support Vector Machines* (SVM) [116] and *AdaBoost* [33] are motivated and analyzed by margins. Since the introduction of these algorithms dozens of papers have been published on different aspects of margins in supervised learning [100, 81, 15].

The most common way to define the margin of an instance with respect to a classification rule is as the distance between the instance and the decision boundary induced by the classification rule. SVM uses this kind of margin. We refer to this kind of margin as *sample margin*. However, as we show in [24], an alternative definition, *Hypothesis Margin*, exists. In this definition the margin is the distance that the *classifier* can travel without changing the way it labels a sample instance. Note that this definition requires a distance measure between classifiers. This type of margin is used in AdaBoost [33]. The difference between

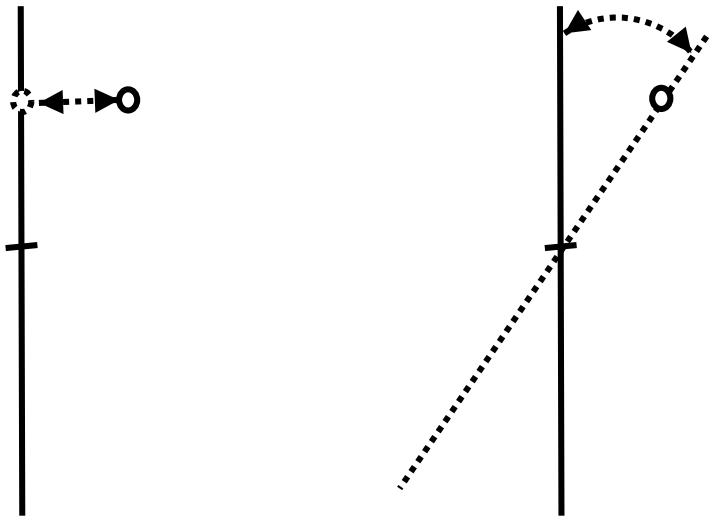

Figure 3.1: Illustration of the two types of margins for homogeneous linear classifiers with angle as distance measure between classifiers. (a) *sample margin* measures how much an **instance** can travel before it hits the decision boundary. (b) on the other hand *a hypothesis margin* measures how much the **hypothesis** can travel before it hits an instance.

these two definitions of margin is illustrated in figure 3.1.

While the margins themselves are a non-negative, the standard convention is to add a sign to them that indicates whether the instance was correctlly calssified, where a negative sign indicates wrong classification. We use the terms *signed margin* or *unsigned margin* to indicate whether the sign is considered or not, but this prefix is omitted in the following whenever it is clear from the context. Note that the signed margin depends on the true label and therefore is only available during the training stage.

3.3 Margins for 1-NN

Throughout this chapter we will be interested in margins for 1-NN. As described in section 3.1, a 1-NN classifier is defined by a set of training points (prototypes) and the decision boundary is the Voronoi tessellation. The sample margin in this case is the distance between the instance and the Voronoi tessellation, and therefore it measures the sensitivity to small

changes of the instance position. We define the hypothesis margin for 1-NN as follows:

Definition 3.1 *The* (unsigned) hypothesis margin *for 1-NN with respect to an instance* $\mathbf{x}$ *is the maximal distance* θ *such that the following condition holds: if we draw a ball with radius* θ *around each prototype, any change of the location of prototypes inside their* θ *ball will not change the assigned label of the instance* $\mathbf{x}$.

Therefore, the hypothesis margin measures stability with respect to small changes in the prototype locations. See figure 3.2 for illustration. The sample margin for 1-NN can be unstable since small relocations of the prototypes might lead to a dramatic change in the sample margin. Thus the hypothesis margin is preferable in this case. Furthermore, the hypothesis margin is easy to compute (lemma 3.1) and lower bounds the sample-margin (lemma 3.2).

Lemma 3.1 *Let* $\nu = (\nu_1, \ldots, \nu_k)$ *be a set of prototypes. Let* $\mathbf{x}$ *be an instance then the (unsigned) hypothesis margin of* ν *with respect to* $\mathbf{x}$ *is* $\theta = \frac{1}{2}\left(\|\nu_j - \mathbf{x}\| - \|\nu_i - \mathbf{x}\|\right)$ *where* ν_i *is the closest prototype to* $\mathbf{x}$ *with the same label as* $\mathbf{x}$ *and* ν_j *is the closest prototype with alternative label.*

The proof of this lemma is straightforward from the definition.

Lemma 3.2 *Let* $\mathbf{x}$ *be an instance and* $\nu = (\nu_1, \ldots, \nu_k)$ *be a set of prototypes. Then the (unsigned) sample margin of* $\mathbf{x}$ *with respect to* ν *is greater or equal to the (unsigned) hypothesis margin of* ν *with respect to* $\mathbf{x}$.

Proof. Let θ be larger than the sample margin, i.e. there exists $\hat{\mathbf{x}}$ such that $\|\mathbf{x} - \hat{\mathbf{x}}\| \leq \theta$ but $\mathbf{x}$ and $\hat{\mathbf{x}}$ are labeled differently by ν. Since they are labeled differently, the closest prototype to $\mathbf{x}$ is different than the closest prototype to $\hat{\mathbf{x}}$. W.L.G let ν_1 be the closest prototype to $\mathbf{x}$ and ν_2 be the closest prototype to $\hat{\mathbf{x}}$.

Let $w = \mathbf{x} - \hat{\mathbf{x}}$ then $\|w\| \leq \theta$. Define $\forall j \quad \hat{\nu}_j = \nu_j + w$. We claim that $\hat{\nu}_2$ is the closest prototype to $\mathbf{x}$ in $\hat{\nu}$. This follows since

$$\|\mathbf{x} - \hat{\nu}_j\| = \|\mathbf{x} - (\nu_j + w)\| = \|\hat{\mathbf{x}} - \nu_j\|$$

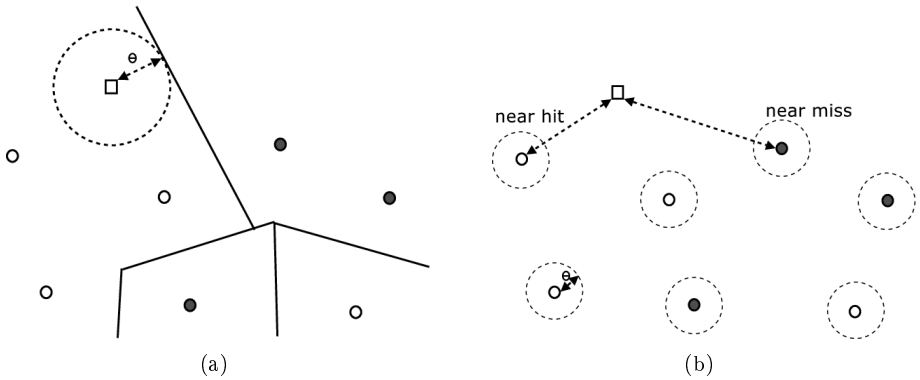

Figure 3.2: The two types of margins for the Nearest Neighbor rule. We consider a set of prototypes (the circles) and measure the margin with respect to an instance (the square). (a) the *(unsigned) sample margin* is the distance between the instance and the decision boundary (the Voronoi tessellation). (b) the *(unsigned) hypothesis margin* is the largest distance the prototypes can travel without altering the label of the new instance. In this case it is half the difference between the distance to the **nearmiss** and the distance to the **nearhit** of the instance. Note that in this example the white prototype governs the margin is not the same for both types of margins: for the hypothesis margin it is the one closest to the new instance, whereas for the sample margin it is the one that defines the relevant segment of the Voronoi tessellation.

Hence $\hat{\nu}$ assigns $\mathbf{x}$ a different label and also no prototype traveled a distance which is larger than θ, thus the hypothesis margin is less than θ.

Since this result holds for any θ greater than the sample margin of ν we conclude that the hypothesis margin of ν is less than or equal to its sample margin. □

Lemma 3.2 shows that if we find a set of prototypes with a large hypothesis margin then it has a large sample margin as well.

3.4 Margin Based Evaluation Function

Our selection algorithm works on the most common selection paradigm that consists of an evaluation function which assigns a score to a given subset of features and a searches method that search for a set with a high score (see section 1.2.1 for more details on this paradigm). In this section we introduce our margin based evaluation function.

A good generalization can be guaranteed if many instances have a large margin (see

section 3.6). We introduce an evaluation function, which assigns a score to sets of features according to the margin they induce. First we formulate the margin as a function of the selected set of features.

Definition 3.2 *Let P be a set of prototypes and $\mathbf{x}$ be an instance. Let $\mathbf{w}$ be a weight vector over the feature set, then the* margin *of $\mathbf{x}$ is*

$$\theta_P^{\mathbf{w}}(\mathbf{x}) = \frac{1}{2}\left(\|\mathbf{x} - \textbf{\textit{nearmiss}}(\mathbf{x})\|_{\mathbf{w}} - \|\mathbf{x} - \textbf{\textit{nearhit}}(\mathbf{x})\|_{\mathbf{w}}\right) \tag{3.1}$$

where $\|\mathbf{z}\|_{\mathbf{w}} = \sqrt{\sum_i w_i^2 z_i^2}$, $\textbf{\textit{nearhit}}(\mathbf{x})$ is the closest prototype to $\mathbf{x}$ with same label as $\mathbf{x}$ and $\textbf{\textit{nearmiss}}(\mathbf{x})$ is the closest prototype to $\mathbf{x}$ with alternative label

Definition 3.2 extends beyond feature selection and allows weight over the features. When selecting a set of features F we can use the same definition by identifying F with its indicating vector. Therefore, we denote by $\theta_P^F(\mathbf{x}) := \theta_P^{I_F}(\mathbf{x})$ where I_F is one for any feature in F and zero otherwise.

Since $\theta^{\lambda w}(\mathbf{x}) = |\lambda|\theta^{\mathbf{w}}(\mathbf{x})$ for any scalar λ, it is natural to introduce some normalization factor. The natural normalization is to require $\max w_i^2 = 1$, since it guarantees that $\|\mathbf{z}\|_{\mathbf{w}} \leq \|\mathbf{z}\|$ where the right hand side is the Euclidean norm of z.

Now we turn to defining the evaluation function. The building blocks of this function are the margins of all the instances. The margin of each instance $\mathbf{x}$ is calculated with respect to the sample excluding $\mathbf{x}$ ("leave-one-out margin").

Definition 3.3 *Let $u(\cdot)$ be a utility function. Given a training set S and a weight vector w,* the evaluation function *is:*

$$e(\mathbf{w}) = \sum_{\mathbf{x} \in S} u\left(\theta_{S \setminus \mathbf{x}}^{\mathbf{w}}(\mathbf{x})\right) \tag{3.2}$$

The utility function controls the contribution of each margin term to the overall score. It is natural to require the utility function to be non-decreasing; thus larger margin introduce larger utility. We consider three utility functions: linear, zero-one and sigmoid. The linear utility function is defined as $u(\theta) = \theta$. When the linear utility function is used, the evaluation function is simply the sum of the margins. The zero-one utility is equals 1 when the margin is positive and 0 otherwise. When this utility function is used the utility function

is proportional to the leave-one-out error. The sigmoid utility is $u(\theta) = 1/(1 + exp(-\beta\theta))$. The sigmoid utility function is less sensitive to outliers than the linear utility, but does not ignore the magnitude of the margin completely as the zero-one utility does. Note also that for $\beta \to 0$ or $\beta \to \infty$ the sigmoid utility function becomes the linear utility function or the zero-one utility function respectively. In the *Simba* algorithm we assume that the utility function is differentiable, and therefore the zero-one utility cannot be used.

It is natural to look at the evaluation function solely for weight vectors $\mathbf{w}$ such that $\max w_i^2 = 1$. However, formally, the evaluation function is well defined for any $\mathbf{w}$, a fact which we make use of in the *Simba* algorithm. We also use the notation $e(F)$, where F is a set of features to denote $e(I_F)$.

3.5 Algorithms

In this section we present two algorithms which attempt to maximize the margin based evaluation function. Both algorithms can cope with multi-class problems. Our algorithms can be considered as filter methods for general classifiers. They also have much in common with wrappers for 1-NN. A Matlab implementation of these algorithms is available at `http://www.cs.huji.ac.il/labs/learning/code/feature_selection/`. I also wrote a Matlab GUI application that was designed to make it easy to perform feature selection and assess the effect of the selection on classification accuracy. The tool is called *Feature Selection Tool* (FST) and can be downloaded from `http://www.cs.huji.ac.il/~anavot/feature_selection_tool/fst.htm`. The tool is mainly aimed at make it easier for researchers who are not programmers to use and evaluate feature selection on their data. The tool supports the selection algorithms which are presented in this chapter and some other related or classic selection algorithms. The classification algorithms which are currently supported are 1-Nearest-Neighbors, Naive Bayes and SVM.

3.5.1 Greedy Feature Flip Algorithm (G-flip)

The *G-flip* (algorithm 1) is a greedy search algorithm for maximizing $e(F)$, where F is a set of features. The algorithm repeatedly iterates over the feature set and updates the set of chosen

Algorithm 1 Greedy Feature Flip (G-flip)

1. Initialize the set of chosen features to the empty set: $F = \phi$
2. for $t = 1, 2, \ldots$
   (a) pick a random permutation s of $\{1 \ldots N\}$
   (b) for $i = 1$ to N,
      i. evaluate $e_1 = e\left(F \cup \{s(i)\}\right)$ and $e_2 = e\left(F \setminus \{s(i)\}\right)$
      ii. if $e_1 > e_2$, $F = F \cup \{s(i)\}$
      else-if $e_2 > e_1$, $F = F \setminus \{s(i)\}$
   (c) if no change made in step (b) then break

features. In each iteration it decides to remove or add the current feature to the selected set by evaluating the margin term (3.2) with and without this feature. This algorithm is similar to the zero-temperature Monte-Carlo (Metropolis) method. It converges to a local maximum of the evaluation function, as each step increases its value and the number of possible feature sets is finite. The computational complexity of one pass over all features of a naive implementation of *G-flip* is $\Theta\left(N^2 m^2\right)$ where N is the number of features and m is the number of instances. However the complexity can be reduced to $\Theta\left(Nm^2\right)$ since updating the distance matrix can be done efficiently after each addition/deletion of a feature from the current active set. Empirically *G-flip* converges in a few iterations. In all our experiments it converged after less than 20 epochs, in most of the cases in less than 10 epochs. A nice property of this algorithm is that once the utility function is chosen, it is *parameter free*. There is no need to tune the number of features or any type of threshold.

3.5.2 Iterative Search Margin Based Algorithm (Simba)

The *G-flip* algorithm presented in section 3.5.1 tries to find the feature set that maximizes the margin directly. Here we take another approach. We first find the weight vector $\mathbf{w}$ that maximizes $e(\mathbf{w})$ as defined in (3.2) and then use a threshold in order to get a feature set. Of course, it is also possible to use the weights directly by using the induced distance measure instead. Since $e(\mathbf{w})$ is smooth almost everywhere, whenever the utility function is smooth, we use gradient ascent in order to maximize it. The gradient of $e(\mathbf{w})$ when evaluated on a

Algorithm 2 Simba

1. initialize $\mathbf{w} = (1, 1, \ldots, 1)$
2. for $t = 1 \ldots T$
   (a) pick randomly an instance $\mathbf{x}$ from S
   (b) calculate $\mathbf{nearmiss}(\mathbf{x})$ and $\mathbf{nearhit}(\mathbf{x})$ with respect to $S \setminus \{\mathbf{x}\}$ and the weight vector $\mathbf{w}$.
   (c) for $i = 1, \ldots, N$
   calculate
   $$\triangle_i = \frac{1}{2} \frac{\partial u(\theta(\mathbf{x}))}{\partial \theta(\mathbf{x})} \left(\frac{(x_i - \mathbf{nearmiss}(\mathbf{x})_i)^2}{\|\mathbf{x} - \mathbf{nearmiss}(\mathbf{x})\|_{\mathbf{w}}} - \frac{(x_i - \mathbf{nearhit}(\mathbf{x})_i)^2}{\|\mathbf{x} - \mathbf{nearhit}(\mathbf{x})\|_{\mathbf{w}}} \right) w_i$$
   (d) $\mathbf{w} = \mathbf{w} + \triangle$
3. $\mathbf{w} \leftarrow \mathbf{w}^2 / \left\|\mathbf{w}^2\right\|_\infty$ where $(\mathbf{w}^2)_i := (w_i)^2$.

sample S is:

$$\begin{aligned} (\bigtriangledown e(\mathbf{w}))_i &= \frac{\partial e(\mathbf{w})}{\partial w_i} = \sum_{\mathbf{x} \in S} \frac{\partial u(\theta(\mathbf{x}))}{\partial \theta(\mathbf{x})} \frac{\partial \theta(\mathbf{x})}{\partial \mathbf{w}_i} \\ &= \frac{1}{2} \sum_{\mathbf{x} \in S} \frac{\partial u(\theta(\mathbf{x}))}{\partial \theta(\mathbf{x})} \left(\frac{(x_i - \mathbf{nearmiss}(\mathbf{x})_i)^2}{\|\mathbf{x} - \mathbf{nearmiss}(\mathbf{x})\|_{\mathbf{w}}} - \frac{(x_i - \mathbf{nearhit}(\mathbf{x})_i)^2}{\|\mathbf{x} - \mathbf{nearhit}(\mathbf{x})\|_{\mathbf{w}}} \right) w_i \end{aligned} \tag{3.3}$$

In *Simba* (algorithm 2) we use a stochastic gradient ascent over $e(\mathbf{w})$ while ignoring the constraint $\|\mathbf{w}^2\|_\infty = 1$. In each step we evaluate only one term in the sum in (3.3) and add it to the weight vector $\mathbf{w}$. The projection on the constraint is done only at the end (step 3).

The computational complexity of *Simba* is $\Theta(TNm)$ where T is the number of iterations, N is the number of features and m is the size of the sample S. Note that when iterating over all training instances, i.e. when $T = m$, the complexity is $\Theta\left(Nm^2\right)$.

3.5.3 Comparison to Relief

Relief [61] (algorithm 3) is a feature selection algorithm, which was shown to be very efficient for estimating feature quality. The algorithm holds a weight vector over all features and updates this vector according to the instances presented. [61] proved that under some assumptions, the expected weight is large for relevant features and small for irrelevant ones. They also explain how to choose the relevance threshold τ in a way that ensures the prob-

Algorithm 3 RELIEF [61]

1. initiate the weight vector to zero: $\mathbf{w} = 0$
2. for $t = 1 \dots T$,
   (a) pick randomly an instance $\mathbf{x}$ from S
   (b) for $i = 1 \dots N$,
      i. $w_i = w_i + (x_i - \mathbf{nearmiss}(\mathbf{x})_i)^2 - (x_i - \mathbf{nearhit}(\mathbf{x})_i)^2$
3. the chosen feature set is $\{i | w_i > \tau\}$ where τ is a threshold

ability that a given irrelevant feature chosen is small. *Relief* was extended to deal with multi-class problems, noise and missing data by [65]. For multi-class problems [65] also presents a version called *Relief-F* that instead of using the distance to the nearest point with an alternative label, looks at the distances to the nearest instance of any alternative class and takes the average. In the experiments we made *Relief-F* was inferior to the standard *Relief*.

Note that the update rule in a single step of *Relief* is similar to the one performed by *Simba* when the utility function is linear, i.e. $u(\theta) = \theta$ and thus $\partial u(\theta)/\partial\theta = 1$. Indeed, empirical evidence shows that *Relief* does increase the margin (see section 3.7). However, there is a major difference between *Relief* and *Simba*: *Relief* does not re-evaluate the distances according to the weight vector $\mathbf{w}$ and thus it is inferior to *Simba*. In particular, *Relief* has no mechanism for eliminating redundant features. *Simba* may also choose correlated features, but only if this contributes to the overall performance. In terms of computational complexity, *Relief* and *Simba* are equivalent.

3.5.4 Comparison to R2W2

R2W2 [120] is the state-of-the-art feature selection algorithm for the *Support Vector Machines* (SVM) classifier. This algorithm is a sophisticated wrapper for SVM and therefore uses the maximal margin principle for feature selection indirectly. The goal of the algorithm is to find a weights vector over the features, which will minimize the objective function of the SVM optimization problem. This objective function can be written as R^2W^2 where

R is the radius of a ball containing all the training data and W is the norm of the linear separator. The optimization is done using gradient descent. After each gradient step a new SVM optimization problem is constructed and solved. Thus it becomes cumbersome for large scale data.

The derivation of R2W2 algorithm assumes that the data are linearly separable. Since this cannot be guaranteed in the general case we use the "ridge" trick of adding a constant value to the diagonal of the kernel matrix. Note also that R2W2 is designed for binary classification tasks only. There are several ways in which it can be extended to multi class problems. However, these extensions will make the algorithm even more demanding than its original version.

As in SVM, R2W2 can be used together with a kernel function. We chose to use the Radial Basis Function (RBF) kernel. The RBF kernel is defined to be

$$K(\mathbf{x}_1, \mathbf{x}_2) = e^{-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|}{2\sigma^2}}$$

where σ is a predefined parameter. The choice of the RBF kernel is due to the similarity between SVM with RBF kernel and the nearest-neighbor rule. Our implementation is based on the one in the *Spider package* [119].

3.6 Theoretical Analysis

In this section we use feature selection and large margin principles to prove finite sample generalization bounds for *One Nearest Neighbor* (1-NN), combined with the preceding step of feature selection. [22] showed that asymptotically the generalization error of 1-NN can exceed the generalization error of the Bayes optimal classification rule by at most a factor of 2. However, on finite samples, nearest neighbor can over-fit and exhibit poor performance.

1-NN gives zero training error, on almost any sample. Thus the training error is too rough to provide information on the generalization performance of 1-NN. We therefore need a more detailed measure to provide meaningful generalization bounds and this is where margins become useful. It turns out that in a sense 1-NN is a maximum margin algorithm. Once our proper definition of margin is used, i.e. sample-margin, it is easy to verify that

1-NN generates the classification rule with the largest possible margin.

The combination of a large margin and a small number of features provides enough evidence to obtain a useful bound on the generalization error. The bound we provide here is data-dependent [103, 9]. Therefore, the value of the bound depends on the specific sample. The bound holds simultaneously for any possible method to select a set of features. Thus, if a selection algorithm selects a small set of features with a large margin, the bound guarantees that 1-NN that is based on this set of features will generalize well. This is the theoretical rationale behind *Simba* and *G-flip*.

We use the following notation in our theoretical results:

Definition 3.4 *Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \{\pm 1\}$ and $h : \mathcal{X} \longrightarrow \{\pm 1\}$ a classification function. We denote by $er_{\mathcal{D}}(h)$ the generalization error of h with respect to $\mathcal{D}$:*

$$er_{\mathcal{D}}(h) \quad = \quad \Pr_{\mathbf{x},y\sim\mathcal{D}}[h(\mathbf{x}) \neq y]$$

For a sample $S = \{(\mathbf{x}_k, y_k)\}_{k=1}^m \in (\mathcal{X} \times \{\pm 1\})^m$ and a constant $\gamma > 0$ we define the γ-sensitive training error to be

$$\hat{er}_S^{\gamma}(h) = \frac{1}{m}\left|\left\{ (k : h(\mathbf{x}_k) \neq y_k) \quad or \quad (\mathbf{x}_k \text{ has sample-margin } < \gamma) \right\}\right|$$

Our main result is the following theorem[2]:

Theorem 3.1 *Let $\mathcal{D}$ be a distribution over $\mathbb{R}^N \times \{\pm 1\}$ which is supported on a ball of radius R in $\mathbb{R}^N$. Let $\delta > 0$ and let S be a sample of size m such that $S \sim \mathcal{D}^m$. With probability $1 - \delta$ over the random choice of S, for any set of features F and any $\gamma \in (0, 1]$*

$$er_{\mathcal{D}}(h) \leq \hat{er}_S^{\gamma}(h) + \sqrt{\frac{2}{m}\left(d \ln\left(\frac{34em}{d}\right)\log_2(578m) + \ln\left(\frac{8}{\gamma\delta}\right) + (|F|+1)\ln N\right)}$$

Where h is the nearest neighbor classification rule when distance is measured only on the features in F and $d = (64R/\gamma)^{|F|}$.

A few notes about this bound; First the size of the feature space, N, appears only logarithmically in the bound. Hence, it has a minor effect on the generalization error of 1-NN. On the other hand, the number of selected features, F, appears in the exponent. This

[2] Note that the theorem holds when sample-margin is replaced by hypothesis-margin since the later lower bounds the former.

is another realization of the "curse of dimensionality" [12]. See appendix A for the proof of theorem 3.1.

A large margin for many instances will make the first term of the bound small, while using a small set of features will make the second term of the bound small. This gives us the motivation to look for small sets of features that induce large margin, and that is what *G-flip* and *Simba* do. As this bound is a worst case bound, like all the PAC style bounds, it is very loose in most of the cases, and the empirical results are expected to be much better.

3.7 Empirical Assessment

We first demonstrate the behavior of *Simba* on a small synthetic problem. Then we compare the different algorithms on image and text classification tasks. The first task is pixel (feature) selection for discriminating between male and female face images. The second task is a word (feature) selection for multi-class document categorization. In order to demonstrate the ability of our algorithms to work with other classifiers (beside of Nearest Neighbor) we also report results with SVM with RBF kernel (see section 3.7.4). We also report the results obtained on some of the datasets of the *NIPS-2003 feature selection challenge* [44]. For these comparisons we have used the following feature selection algorithms: *Simba* with both linear and sigmoid utility functions (referred as *Simba(lin)* and *Simba(sig)* respectively), *G-flip* with linear, zero-one and sigmoid utility functions (referred as *G-flip(lin)*, *G-flip(zero-one)* and *G-flip(sig)* respectively), *Relief*, *R2W2* and *Infogain*[3].

3.7.1 The Xor Problem

To demonstrate the quality of the margin based evaluation function and the ability of the *Simba* algorithm[4] to deal with dependent features we use a synthetic problem. The problem consisted of 1000 instances with 10 real valued features. The target concept is a *xor* function over the first 3 features. Hence, the first 3 features are relevant while the other features are irrelevant. Note that this task is a special case of parity function learning and is considered

[3] Recall that *Infogain* ranks features according to the mutual information between each feature and the labels (see chapter 1, section 1.2.1)

[4] The linear utility function was used in this experiment.

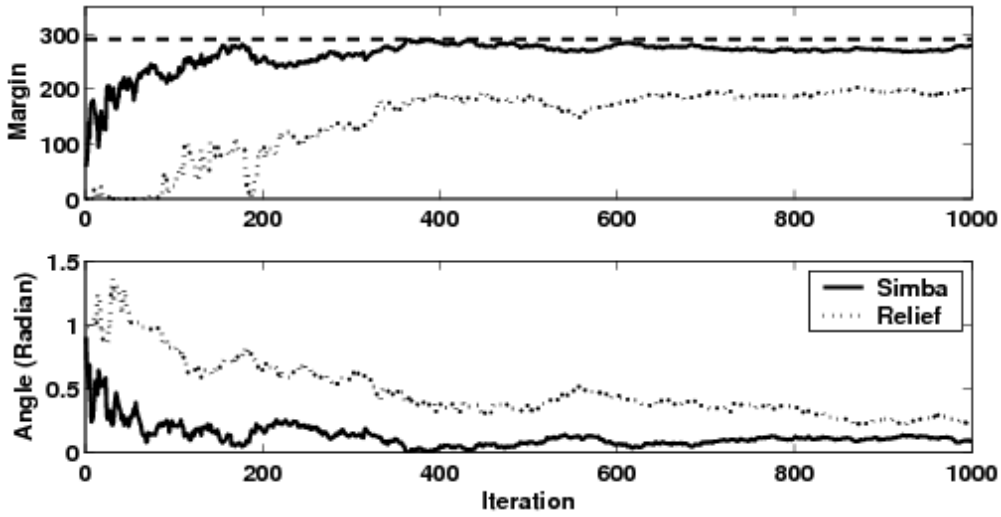

Figure 3.3: The results of applying *Simba* (solid) and *Relief* (dotted) on the *xor* synthetic problem. **Top:** The margin value, $e(\mathbf{w})$, at each iteration. The dashed line is the margin of the correct weight vector. **Bottom:** the angle between the weight vector and the correct feature vector at each iteration (in Radians).

hard for many feature selection algorithms [43]. Thus for example, any algorithm which does not consider functional dependencies between features fails on this task. The simplicity (some might say over-simplicity) of this problem, allows us to demonstrate some of the interesting properties of the algorithms studied.

Figures 3.3 present the results we obtained on this problem. A few phenomena are apparent in these results. The value of the margin evaluation function is highly correlated with the angle between the weight vector and the correct feature vector (see figures 3.3 and 3.4). This correlation demonstrates that the margins characterize correctly the quality of the weight vector. This is quite remarkable since our margin evaluation function can be measured empirically on the training data whereas the angle to the correct feature vector is unknown during learning.

As suggested in section 3.5.3 *Relief* does increase the margin as well. However, *Simba*, which maximizes the margin directly, outperforms *Relief* quite significantly. as shown in figure 3.3.

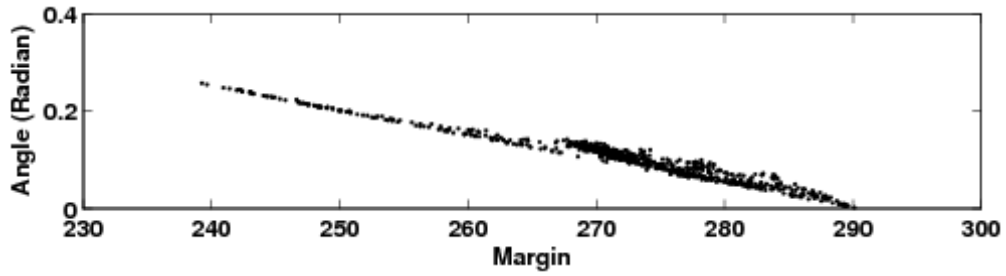

Figure 3.4: The scatter plot shows the angle to the correct feature vector as function of the value of the margin evaluation function. The values were calculated for the *xor* problem using *Simba* during iterations 150 to 1000. Note the linear relation between the two quantities.

Figure 3.5: Excerpts from the face images dataset.

3.7.2 Face Images

We applied the algorithms to the AR face database [80], which is a collection of digital images of males and females with various facial expressions, illumination conditions, and occlusions. We selected 1456 images and converted them to gray-scale images of 85×60 pixels, which are taken as our initial 5100 features. Examples of the images are shown in figure 3.5. The task we tested is classifying the male vs. the female faces.

In order to improve the statistical significance of the results, the dataset was partitioned independently 20 times into training data of 1000 images and test data of 456 images. For each such partitioning (split) *Simba*[5], *G-flip*, *Relief*, *R2W2* and *Infogain* were applied to select optimal features and the 1-NN algorithm was used to classify the test data points. We used 10 random starting points for *Simba* (i.e. random permutations of the train data) and selected the result of the single run which reached the highest value of the evaluation function.

[5] *Simba* was applied with both linear and sigmoid utility functions. We used $\beta = 0.01$ for the sigmoid utility.

The average accuracy versus the number of features chosen, is presented in figure 3.6. *G-flip* gives only one point on this plot, as it chooses the number of features automatically, although this number changes between different splits, it does not change significantly.

The features *G-flip (zero-one)* selected enabled 1-NN to achieve accuracy of 92.2% using about 60 features only, which is better than the accuracy obtained with the whole feature set (91.5%). *G-flip (zero-one)* outperformed any other alternative when only few dozens of features were used. *Simba(lin)* significantly outperformed *Relief*, *R2W2* and *Infogain*, especially in the small number of features regime. If we define a difference as significant if the one algorithm is better than the other in more than 90% of the partition, we can see that *Simba* is significantly better than *Relief*, *Infogain* and *R2W2* when fewer than couple of hundreds of features are being used (see figure 3.7). Moreover, the 1000 features that *Simba* selected enabled 1-NN to achieve an accuracy of 92.8%, which is better than the accuracy obtained with the whole feature set (91.5%).

A closer look on the features selected by *Simba* and *Relief* (figure 3.8) reveals the clear difference between the two algorithms. *Relief* focused on the hair-line, especially around the neck, and on other contour areas in a left-right symmetric fashion. This choice is suboptimal as those features are highly correlated to each other and therefore a smaller subset is sufficient. *Simba* on the other hand selected features in other informative facial locations but mostly on one side (left) of the face, as the other side is clearly highly correlated and does not contribute new information to this task. Moreover, this dataset is biased in the sense that more faces are illuminated from the right. Many of them are saturated and thus *Simba* preferred the left side over the less informative right side.

3.7.3 Reuters

We applied the different algorithms on a multi-class text categorization task. For these purpose we used a subset of the Reuters-21578 dataset[6]. We have used the documents, which are classified to exactly one of the following 4 topics: *interest*, *trade*, *crude* and *grain*. The obtained dataset contains 2066 documents, which are approximately equally distributed between the four classes. Each document was represented as the vector of counts of the

[6]The dataset can be found at *http://www.daviddlewis.com/resources/*

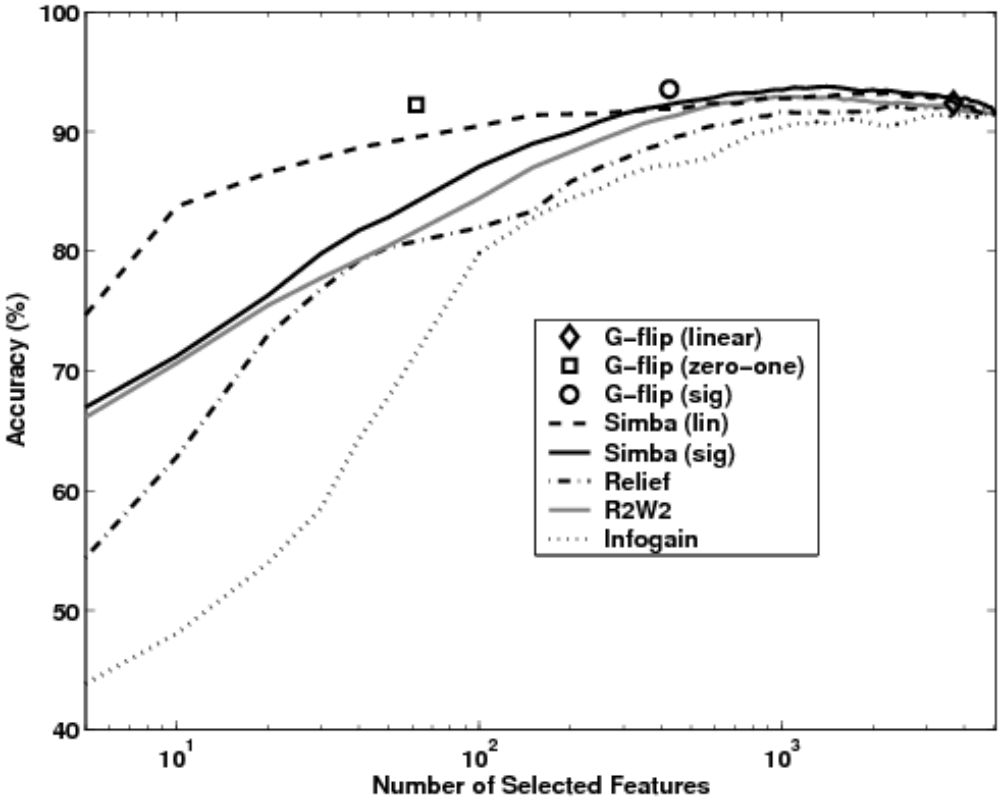

Figure 3.6: **Results for AR faces dataset.** The accuracy achieved on the AR faces dataset when using the features chosen by the different algorithms. The results were averaged over the 20 splits of the dataset. For the sake of visual clarity, error bars are presented separately in figure 3.7.

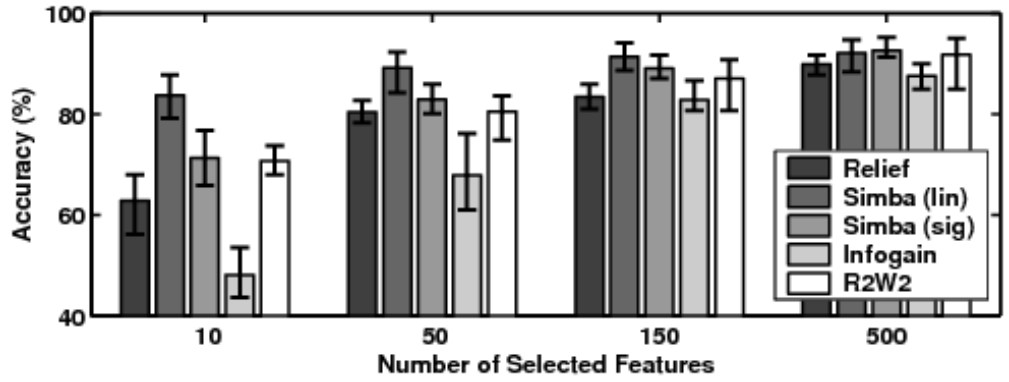

Figure 3.7: **Error intervals for AR faces dataset.** The accuracy achieved on the AR faces dataset when using the features chosen by the different algorithms. The error intervals show the area were 90% of the results (of the 20 repeats) fell, i.e., the range of the results after eliminating the best and the worse iterations out of the 20 repeats.

(a) (b) (c) (d) (e) (f)

Figure 3.8: The features selected (in black) by *Simba(lin)* and *Relief* for the face recognition task. 3.8(a), 3.8(b) and 3.8(c) shows 100, 500 and 1000 features selected by *Simba*. 3.8(d), 3.8(e) and 3.8(f) shows 100, 500 and 1000 features selected by *Relief*.

different words. Stop-words were omitted and numbers were converted to a predefined special character as a preprocessing.

To improve the statistical significance of our results, the corpus was partitioned 20 times into training set of 1000 documents and test set of 1066 documents. For each such partitioning, the words that appear less than 3 times in the training set were eliminated, which left ∼4000 words (features). For each partitioning *G-flip*, *Simba*, *Relief*, *Relief-F* and *Infogain* were applied to select an optimal set of features[7] and the 1-NN was used to classify the test documents. *Simba* and *G-flip* were applied with both linear and sigmoid utility functions[8]. *G-flip* was also applied with the zero-one utility function. We have used 10 random starting points for *Simba* and selected the results of the single run that achieved the highest value for the evaluation function. The average (over the 20 splits) accuracy versus the number of chosen features is presented in figure 3.9. *G-flip* gives only one point on this plot, as it chooses the number of features automatically, although this number changes between different splits, it does not change significantly. Another look on the results is given in figure 3.10. This figure shows error intervals around the average that allow appreciating the statistical significance of the differences in the accuracy. The top ranked twenty features are presented in table 3.1.

The best overall accuracy (i.e. when ignoring the number of features used) was achieved by *G-flip(sig)*. *G-flip(sig)* got 94.09% generalization accuracy using ∼350 features. *Infogain* and *Simba(sig)* are just a little behind with 92.86% and 92.41% , that was achieved using ∼40 and ∼30 features only (respectively). *Relief* is far behind with 87.92% that was achieved using 250 features.

The advantage of *Simba(sig)* is very clear when looking on the accuracy versus the number of features used. Among the algorithms that were tested, *Simba(sig)* is the only algorithm that achieved (almost) best accuracy over the whole range. Indeed, when less than few dozens of features are used, *Infogain*, achieved similar accuracy, but when more features are used, it's accuracy drops dramatically. While *Simba(sig)* achieved accuracy above 90% for any number of features between ∼20 and ∼3000, the accuracy achieved by

[7] *R2W2* was not applied for this problem as it is not defined for multi-class problems.
[8] The sigmoid utility function was used with $\beta = 1$ for both *G-flip* and *Simba*.

Simba(sig)		Simba(lin)		Relief		Infogain	
oil	rate	oil	corn	##	s	oil	trade
wheat	bpd	##	brazil	#	bank	tonnes	wheat
trade	days	tonnes	wish	###	rate	crude	bank
tonnes	crude	trade	rebate	oil	dlrs	rate	agriculture
corn	japanese	wheat	water	trade	barrels	grain	barrels
bank	deficit	bank	stocks	####	rates	petroleum	rates
rice	surplus	rice	certificates	billion	bpd	pct	corn
grain	indonesia	rates	recession	opec	pct	tariffs	japan
rates	interest	billion	got	tonnes	barrel	mt	deficit
ec	s	pct	ve	wheat	cts	energy	bpd

Table 3.1: The first 20 words (features) selected by the different algorithms for the Reuters dataset. Note that *Simba(sig)* is the only algorithm which selected the titles of all four classes (interest, trade, crude, grain) among the first twenty features

Infogain dropped below 90% when more than ∼400 features are used, and below 85% when more than ∼1500 are used.

The advantage of *Simba(sig)* over relief is also very clear. The accuracy achieved using the features chosen by *Simba(sig)* is notably and significantly better than the one achieved using the features chosen by *Relief*, for any number of selected features. Using only the top 10 features of *Simba(sig)* yields accuracy of 89.21%, which is about the same as the accuracy that can be achieved by any number of features chosen by *Relief*.

3.7.4 Face Images with Support Vector Machines

In this section we show that our algorithms work well also when using another distance based classifier, instead of the Nearest Neighbor classifier. We test the different algorithms together with SVM with RBF kernel classifier and show that our algorithms works as good as, and even better than the *R2W2* algorithm that was tailored specifically for this setting and is much more computationally demanding.

We have used the AR face database [80]. We have repeated the same experiment as described in section 3.7.2, the only difference being that once the features were chosen, we used SVM-RBF to classify the test data points. The sigma parameter used in the RBF kernel was selected to be 3500, the same parameter used in the R2W2 feature selection algorithm. The value for this parameter was tuned using cross-validation. See figure 3.11

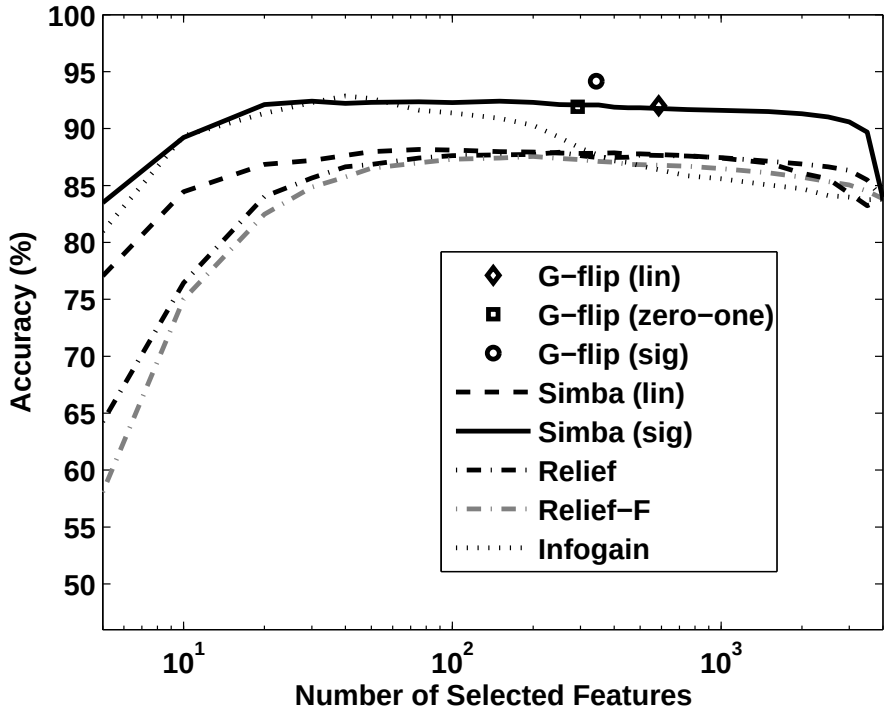

Figure 3.9: **Results for Reuters dataset.** The accuracy achieved on the Reuters dataset when using the features chosen by the different algorithms. The results were averaged over the 20 splits of the dataset. For the sake of visual clarity, error bars are presented separately in figure 3.10.

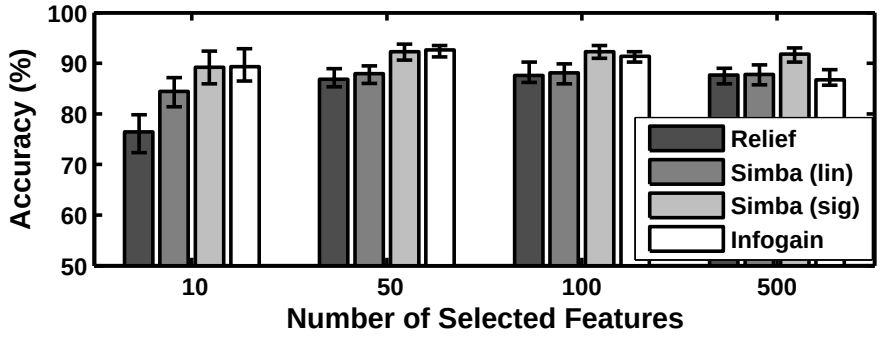

Figure 3.10: **Error intervals for Reuters dataset.** The accuracy achieved on the Reuters dataset when using the features chosen by the different algorithms. The error intervals show the area were 90% of the results (of the 20 repeats) fell, i.e., the range of the results after eliminating the best and the worse iterations out of the 20 repeats.

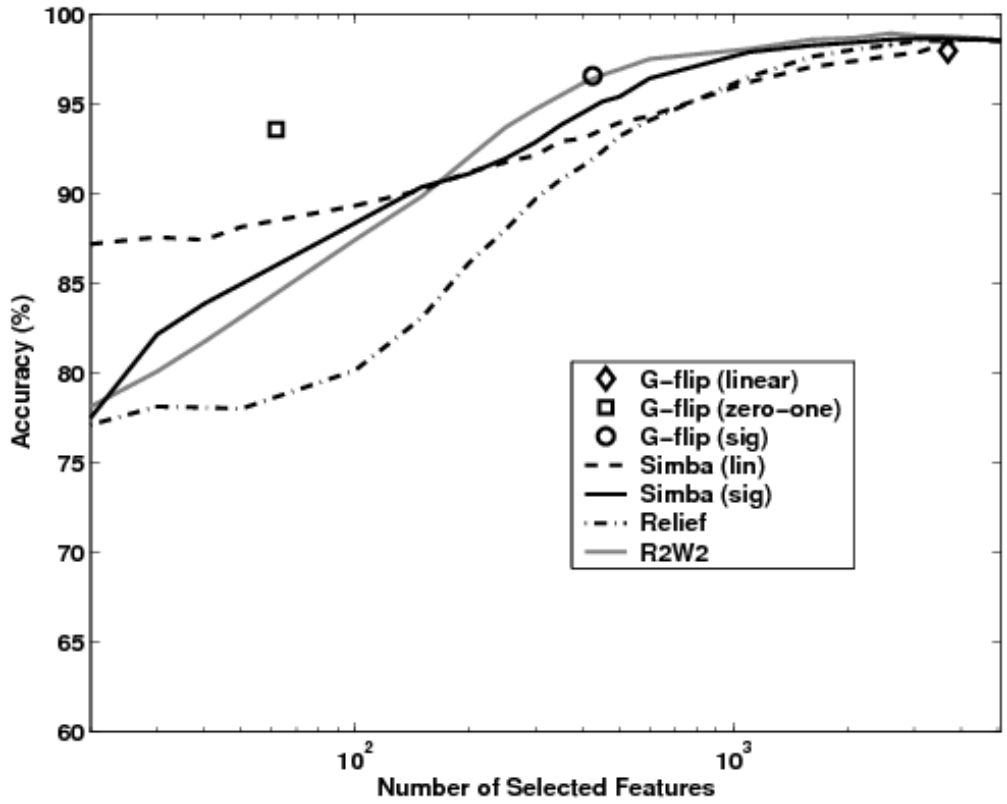

Figure 3.11: **Results for AR faces dataset with SVM-RBF classifier.** The accuracy achieved on the AR faces dataset when using the features chosen by the different algorithms and SVM-RBF classifier. The results were averaged over the 20 splits of the dataset

for a summary of the results.

Both *Simba* and *G-flip* perform well, especially in the small number of features regime. The results in the graph are the average over the 20 partitions of the data. Note that the only winnings which are 90% significant are those of *Simba (lin)* and *G-flip (zero-one)* when only few dozens of features are used.

3.7.5 The NIPS-03 Feature Selection Challenge

The *NIPS-03 feature selection challenge* [44] was the prime motivator to develop the selection algorithms presented in this chapter. The first versions of *G-flip* were developed during the challenge and *Simba* was developed following the challenge. We applied *G-flip(lin)* as part

of our experiments in the challenge. The two datasets on which we used *G-flip* are *ARCENE* and *MADELON*.

In *ARCENE* we first used *Principal Component Analysis* (PCA) as a preprocessing and then applied *G-flip* to select the principal components to be used for classification. *G-flip* selected only 76 features. These features were fed to a *Support Vector Machine* (SVM) with a RBF kernel and yielded a 12.66% balanced error (the best result on this dataset was a 10.76% balanced error).

In *MADELON* we did not apply any preprocessing. *G-flip* selected only 18 out of the 500 features in this dataset. Feeding these features to SVM-RBF resulted in a 7.61% balanced error, while the best result on this dataset was a 6.22% error.

Although *G-flip* is a very simple and naive algorithm, it ranks as one of the leading feature selection methods on both *ARCENE* and *MADELON*. It is interesting to note that when 1-NN is used as the classification rule instead of SVM-RBF the error degrades only by $\sim$1% on both datasets. However, on the other datasets of the feature selection challenge, we did not use *G-flip* either due to its computational requirements or due to poor performance. Note that we tried only the linear utility function for *G-flip* and did not try *Simba* on any of the challenge datasets, since it was developed after the challenged ended.

3.8 Relation to Learning Vector Quantization

As already mentioned in section 3.1, a possible way to improve the computational complexity of 1-NN and avoid noise overfitting for large high dimensional datasets is by replacing the training set with a small number of *prototypes*. The goal of the training stage in this context is to find good small set of prototypes, i.e. a set that induces a small generalization error. The most prominent algorithm for this task is the 20 year old *Learning Vector Quantization* (LVQ) algorithm [63]. Several variants of LVQ have been presented in the past, but all of them share the following common scheme. The algorithm maintains a set of prototypes, $\nu = (\nu_1, \ldots, \nu_k)$, each is assigned with a predefined label, which is kept constant during the learning process. It cycles through the training data $S^m = \{(\mathbf{x}_l, y_l)\}_{l=1}^m$ and in each iteration modifies the set of prototypes in accordance with one instance $(\mathbf{x}_t, y_t)$. If the prototype ν_j

has the same label as y_t it is attracted to $\mathbf{x}_t$ but if the label of ν_j is different it is repelled from it. Hence LVQ updates the closest prototypes to $\mathbf{x}_t$ according to the rule:

$$\nu_j \leftarrow \nu_j \pm \alpha_t(\mathbf{x}_t - \nu_j) \ , \tag{3.4}$$

where the sign is positive if the label of $\mathbf{x}_t$ and ν_j agree, and negative otherwise. The parameter α_t is updated using a predefined scheme and controls the rate of convergence of the algorithm. The variants of LVQ differ as to which prototypes they choose to update in each iteration and as regards the specific scheme used to modify α_t.

LVQ was presented as a heuristic algorithm. However, in [24] we show that it emerges naturally from the margin based evaluation function presented in section 3.4. Up to very minor differences LVQ is a stochastic gradient ascent over this evaluation function, and the different variants of LVQ correspond to different choices of utility function. In this sense LVQ is "dual" to our margin-based selection algorithms. Both algorithms optimize the same target function, but with respect to different parameters. In the selection algorithms the position of prototypes is fixed and the free parameter is the set of selected features, whereas in LVQ the set of features is fixed and the free parameters are the positions of the prototypes. This new point of view on the veteran LVQ as a maximum margin algorithm enables us to derive margin-based bounds on the generalization error of LVQ. This bound is similar in its form to the bound presented in section 3.6. We do not go into detail in this section, as it not directly related to feature selection and thus goes beyond the scope of this dissertation. The details can be found in [24]. However, another nice observation is the following: SVM looks for the linear classifier with the maximal minimum margin over the training set. On the other hand, among *all* the possible classifiers, 1-NN has the maximal minimum margin over the training set, but it has a very complicated structure. We show that LVQ looks for the classifier with maximal minimum margin for a given number of prototypes k. Recalling that for $k = 2$ we get a linear classifier, and for $k = \#instances$ we get 1-NN, we can consider LVQ as a family of large margin classifiers with a parameter k that controls the complexity of the hypothesis class.

3.9 Summary and Discussion

In this chapter, a margin-based criterion for measuring the quality of a set of features has been presented. Using this criterion we derived algorithms that perform feature selection by searching for the set that maximizes it. We suggested two new methods for maximizing the margin based-measure, *G-flip*, which does a naive local search, and *Simba*, which performs a gradient ascent. These are just some representatives of the variety of optimization techniques (search methods) which can be used. We have also shown that the well-known *Relief* algorithm [61] approximates a gradient ascent algorithm that maximizes this measure. The nature of the different algorithms presented here was demonstrated on various feature selection tasks. It was shown that our new algorithm *Simba*, which is a gradient ascent on our margin based measure, outperforms *Relief* on all these tasks. One of the main advantages of the margin based criterion is the high correlation that it exhibits with feature quality. This was demonstrated in figures 3.3 and 3.4.

The margin based criterion was developed using the 1-Nearest-Neighbor classifier but we expect it to work well for any distance based classifier. Additionally to the test we made with 1-NN, we also tested our algorithms with SVM-RBF classifier and showed that they compete successfully with the state-of-the-art algorithm that was designed specifically for SVM and is much more computationally demanding.

Our main theoretical result in this chapter is a new rigorous bound on the finite sample generalization error of the 1-*Nearest Neighbor* algorithm. This bound depends on the margin obtained following the feature selection.

In the experiments we have conducted, the merits of the new algorithms were demonstrated. However, our algorithms use the Euclidean norm and assume that it is meaningful as a measure of similarity in the data. When this assumption fails, our algorithms might not work. Coping with other similarity measures will be an interesting extension to the work presented here.

The user of *G-flip* or *Simba* should choose a utility function to work with. Here we have demonstrated three such functions: linear, zero-one and sigmoid utility functions. The linear utility function gives equal weight to all points and thus might be sensitive to outliers. The

sigmoid utility function suppresses the influence of such outliers. We have also experimented with the fact that *G-flip* with zero-one utility uses fewer features than *G-flip* with linear utility, while the sigmoid utility lies in between. It is still an open problem how to adapt the right utility for the data being studied. Nevertheless, much like the choice of kernel for SVM, using a validation set it is possible to find a reasonable candidate. A reasonable initial value for the parameter β of the sigmoid utility is something on the same order of magnitude as one over the average distance between training instances. As for the choice between *G-flip* and *Simba*; *G-flip* is adequate when the goal is to choose the best feature subset, without a need to control its precise size. *Simba* is more adequate when ranking the features is required.

A Complementary Proofs for Chapter 3

We begin by proving a simple lemma which shows that the class of nearest neighbor classifiers is a subset of the class of 1-Lipschitz functions. Let $\mathbf{nn}_F^S(\cdot)$ be a function such that the sign of $\mathbf{nn}_F^S(\mathbf{x})$ is the label that the nearest neighbor rule assigns to $\mathbf{x}$, while the magnitude is the sample-margin, i.e. the distance between $\mathbf{x}$ and the decision boundary.

Lemma 3.3 *Let F be a set of features and let S be a labeled sample. Then for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^N$:*

$$\left|\boldsymbol{nn}_F^S(\mathbf{x}_1) - \boldsymbol{nn}_F^S(\mathbf{x}_2)\right| \leq \|F(\mathbf{x}_1) - F(\mathbf{x}_2)\|$$

where $F(\mathbf{x})$ is the projection of $\mathbf{x}$ on the features in F.

Proof. Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$. We split our argument into two cases. First assume that $\mathbf{nn}_F^S(\mathbf{x}_1)$ and $\mathbf{nn}_F^S(\mathbf{x}_2)$ have the same sign. Let $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^{|F|}$ be the points on the decision boundary of the 1-NN rule which are closest to $F(\mathbf{x}_1)$ and $F(\mathbf{x}_2)$ respectively. From the definition of $\mathbf{z}_{1,2}$ it follows that $\mathbf{nn}_F^S(\mathbf{x}_1) = \|F(\mathbf{x}_1) - \mathbf{z}_1\|$ and $\mathbf{nn}_F^S(\mathbf{x}_2) = \|F(\mathbf{x}_2) - \mathbf{z}_2\|$ and thus

$$\begin{aligned} \mathbf{nn}_F^S(\mathbf{x}_2) &\leq \|F(\mathbf{x}_2) - \mathbf{z}_1\| \\ &\leq \|F(\mathbf{x}_2) - F(\mathbf{x}_1)\| + \|F(\mathbf{x}_1) - \mathbf{z}_1\| \\ &= \|F(\mathbf{x}_2) - F(\mathbf{x}_1)\| + \mathbf{nn}_F^S(\mathbf{x}_1) \end{aligned} \tag{3.5}$$

By repeating the above argument while reversing the roles of $\mathbf{x}_1$and $\mathbf{x}_2$ we get

$$\mathbf{nn}_F^S(\mathbf{x}_1) \leq \|F(\mathbf{x}_2) - F(\mathbf{x}_1)\| + \mathbf{nn}_F^S(\mathbf{x}_2) \tag{3.6}$$

Combining (3.5) and (3.6) we obtain

$$\left|\mathbf{nn}_F^S(\mathbf{x}_2) - \mathbf{nn}_F^S(\mathbf{x}_1)\right| \leq \|F(\mathbf{x}_2) - F(\mathbf{x}_1)\|$$

The second case is when $\mathbf{nn}_F^S(\mathbf{x}_1)$ and $\mathbf{nn}_F^S(\mathbf{x}_2)$ have alternating signs. Since $\mathbf{nn}_F^S(\cdot)$ is continuous, there is a point $\mathbf{z}$ on the line connecting $F(\mathbf{x}1)$ and $F(\mathbf{x}_2)$ such that $\mathbf{z}$ is on the decision boundary. Hence,

$$\begin{aligned}\left|\mathbf{nn}_F^S(\mathbf{x}_1)\right| &\leq \|F(\mathbf{x}_1) - \mathbf{z}\| \\ \left|\mathbf{nn}_F^S(\mathbf{x}_2)\right| &\leq \|F(\mathbf{x}_2) - \mathbf{z}\|\end{aligned}$$

and so we obtain

$$\begin{aligned}\left|\mathbf{nn}_F^S(\mathbf{x}_2) - \mathbf{nn}_F^S(\mathbf{x}_1)\right| &= \left|\mathbf{nn}_F^S(\mathbf{x}_2)\right| + \left|\mathbf{nn}_F^S(\mathbf{x}_1)\right| \\ &\leq \|F(\mathbf{x}_2) - \mathbf{z}\| + \|F(\mathbf{x}_1) - \mathbf{z}\| \\ &= \|F(\mathbf{x}_2) - F(\mathbf{x}_1)\|\end{aligned}$$

□

The main tool for proving theorem 3.1 is the following:

Theorem 3.2 *[9] Let $\mathcal{H}$ be a class of real valued functions. Let S be a sample of size m generated i.i.d. from a distribution $\mathcal{D}$ over $\mathcal{X} \times \{\pm 1\}$ then with probability $1-\delta$ over the choices of S, every $h \in \mathcal{H}$ and every $\gamma \in (0,1]$ let $d = fat_{\mathcal{H}}(\gamma/32)$:*

$$er_{\mathcal{D}}(h) \leq \hat{er}_S^{\gamma}(h) + \sqrt{\frac{2}{m}\left(d\ln\left(\frac{34em}{d}\right)\log(578m) + \ln\left(\frac{8}{\gamma\delta}\right)\right)}$$

We now turn to prove theorem 3.1:

Proof (of theorem 3.1): Let F be a set of features such that $|F| = n$ and let $\gamma > 0$. In order to use theorem 3.2 we need to compute the fat-shattering dimension of the class of nearest neighbor classification rules which use the set of features F. As we saw in lemma 3.3 this class is a subset of the class of 1-Lipschitz functions on these features. Hence we can

bound the fat-shattering dimension of the class of NN rules by the dimension of Lipschitz functions.

Since $\mathcal{D}$ is supported in a ball of radius R and $\|\mathbf{x}\| \geq \|F(\mathbf{x})\|$, we need to calculate the fat-shattering dimension of Lipschitz functions acting on points in $\mathbb{R}^n$ with norm bounded by R. The fat_γ-dimension of the 1-NN functions on the features F is thus bounded by the largest γ packing of a ball in $\mathbb{R}^n$ with radius R, which in turn is bounded by $(2R/\gamma)^{|F|}$.

Therefore, for a fixed set of features F we can apply to theorem 3.2 and use the bound on the fat-shattering dimension just calculated. Let $\delta_F > 0$ and we have according to theorem 3.2 with probability $1 - \delta_F$ over sample S of size m that for any $\gamma \in (0, 1]$

$$\begin{aligned} \text{er}_{\mathcal{D}}(\text{nearest-neighbor}) \quad \leq \quad & \hat{\text{er}}_S^{\gamma}(\text{nearest-neighbor}) + \\ & \sqrt{\frac{2}{m}\left(d \ln\left(\frac{34em}{d}\right) \log(578m) + \ln\left(\frac{8}{\gamma\delta_F}\right)\right)} \end{aligned} \tag{3.7}$$

for $d = (64R/\gamma)^{|F|}$. By choosing $\delta_F = \delta / \left(N\binom{N}{|F|}\right)$ we have that $\sum_{F \subseteq [1 \ldots N]} \delta_F = \delta$ and so we can apply the union bound to (3.7) and obtain the stated result. □

Chapter 4

Feature Selection For Regression and its Application to Neural Activity[1]

In this chapter we discuss feature selection for regression (aka function estimation). Once again we use the Nearest Neighbor algorithm and an evaluation function which is similar in nature to the one used for classification in chapter 3. This way we develop a non-linear, simple yet effective feature subset selection method for regression. Our algorithm is able to capture complex dependency of the target function on its input and makes use of the leave-one-out error as a natural regularization. We explain the characteristics of our algorithm on synthetic problems and use it in the context of predicting hand velocity from spikes recorded in motor cortex of a behaving monkey. By applying feature selection we are able to improve prediction quality and suggest a novel way of exploring neural data.

The selection paradigm presented in section 1.2.1 of the introduction which involves an *evaluation function* and a *search method* is adequate for regression as well, but the evaluation function should be suitable for regression; i.e., consider the continuous properties

[1] The results presented in this chapter were first presented in our NIPS05 paper titled "Nearest Neighbor Based Feature Selection for Regression and its Application to Neural Activity" [85].

of the target function. A possible choice of such an evaluation function is the leave-one-out (LOO) mean square error (MSE) of the *k-Nearest-Neighbor* (kNN) estimator ([27, 7]). This evaluation function has the advantage that it both gives a good approximation of the expected generalization error and can be computed quickly. [79] used this criterion on small synthetic problems (up to 12 features). They searched for good subsets using *forward selection*, *backward elimination* and an algorithm (called *schemata*) that *races* feature sets against each other (eliminating poor sets, keeping the fittest) in order to find a subset with a good score. All these algorithms perform a local search by flipping one or more features at a time. Since the space is discrete the direction of improvement is found by trial and error, which slows the search and makes it impractical for large scale real world problems involving many features.

We extend the LOO-kNN-MSE evaluation function to assign scores to *weight vectors* over the features, instead of just to feature subsets. This results in a smooth ("almost everywhere") function over a continuous domain, which allows us to compute the gradient analytically and to employ a stochastic gradient ascent to find a locally optimal weight vector. The resulting weights provide a ranking of the features, which we can then threshold in order to produce a subset. In this way we can apply an easy-to-compute, gradient directed search, without relearning a regression model at each step but while still employing a strong non-linear function estimate (kNN) that can capture complex dependency of the function on its features.

Our original motivation for developing this method was to address a major computational neuroscience question: which features of the neural code are relevant to the observed behavior? This is an important key to the interpretability of neural activity. Feature selection is a promising tool for this task. Here, we apply our feature selection method to the task of reconstructing hand movements from neural activity, which is one of the main challenges in implementing brain computer interfaces [109]. We look at neural population spike counts, recorded in motor cortex of a monkey while it performed hand movements and locate the most informative subset of neural features. We show that it is possible to improve prediction results by wisely selecting a subset of cortical units and their time lags relative to the movement. Our algorithm, which considers feature subsets, outperforms methods that

consider features on an individual basis, suggesting that complex dependency on a set of features exists in the code.

4.1 Preliminaries

Let $g(\mathbf{x})$, $g : \mathbb{R}^N \longrightarrow \mathbb{R}$ be a function that we wish to estimate. Given a set $S \subset \mathbb{R}^N$, the empiric *mean square error* (MSE) of an estimator $\hat{g}$ for g is defined as $MSE_S(\hat{g}) = \frac{1}{|S|}\sum_{x\in S}(g(\mathbf{x}) - \hat{g}(\mathbf{x}))^2$.

kNN Regression As already explained in the previous chapter, *k-Nearest-Neighbor* (kNN) is a simple, intuitive and efficient way to estimate the value of an unknown function in a given point using its values in other (training) points. In section 3.1 we described the the classification version of kNN. Here we use the kNN for a regression problem, so instead of using majority over the labels of the k nearest neighbors, we take the average of the function value. Formally, Let $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ be a set of training points. The kNN estimator is defined as the mean function value of the nearest neighbors: $\hat{g}(\mathbf{x}) = \frac{1}{k}\sum_{\mathbf{x}'\in eyalkr@cs.huji.ac.il\mathcal{N}(\mathbf{x})}\hat{g}(x')$ where $eyalkr@cs.huji.ac.il\mathcal{N}(\mathbf{x}) \subset S$ is the set of k nearest points to $\mathbf{x}$ in S and k is a parameter([27, 7]). A softer version takes a *weighted* average, where the weight of each neighbor is proportional to its proximity. One specific way of doing this is

$$\hat{g}(\mathbf{x}) = \frac{1}{Z}\sum_{\mathbf{x}'\in eyalkr@cs.huji.ac.il\mathcal{N}(\mathbf{x})} g(\mathbf{x}')e^{-d(\mathbf{x},\mathbf{x}')/\beta} \tag{4.1}$$

where $d(\mathbf{x},\mathbf{x}') = \|\mathbf{x}-\mathbf{x}'\|_2^2$ is the ℓ_2 norm, $Z = \sum_{\mathbf{x}'\in eyalkr@cs.huji.ac.il\mathcal{N}(\mathbf{x})} e^{-d(\mathbf{x},\mathbf{x}')/\beta}$ is a normalization factor and β is a parameter. The soft kNN version will be used in the remainder of this paper. This regression method is a special form of *locally weighted regression* (See [7] for an overview of the literature on this subject). It has the desirable property that no learning (other than storage of the training set) is required for the regression. Also note that the Gaussian Radial Basis Function has the form of a *kernel* ([116]) and can be replaced with any operator on two data points that decays as a function of the difference between them (e.g. kernel induced distances). As will be seen in the next section, we use the MSE of a modified kNN regressor to guide the search for a set of features $F \subset \{1, \ldots n\}$ that

achieves a low MSE. However, the MSE and the Gaussian kernel can be replaced by other loss measures and kernels (respectively) as long as they are differentiable almost everywhere.

4.2 The Feature Selection Algorithm

In this section we present our selection algorithm called *RGS* (Regression, Gradient guided, feature Selection). It can be seen as a filter method for general regression algorithms or as a wrapper for estimation by the kNN algorithm.

Our goal is to find subsets of features that induce a small estimation error. As in most supervised learning problems, we wish to find subsets that induce a small generalization error, but since it is not known, we use an *evaluation function* on the training set. This evaluation function is defined not only for subsets but for any weight vector over the features. This is more general because a feature subset can be represented by a binary weight vector that assigns a value of one to features in the set and zero to the rest of the features.

For a given weights vector over the features $\mathbf{w} \in \mathbb{R}^n$, we consider the weighted squared ℓ_2 norm induced by $\mathbf{w}$, defined as $\|z\|_{\mathbf{w}}^2 = \sum_i z_i^2 w_i^2$ and use the k nearest neighbors according to the distance induced by this norm. We use $\mathcal{N}_{\mathbf{w}}(\mathbf{x})$ to denote the set of these k neighbors. Given a training set S, we denote by $\hat{g}_{\mathbf{w}}(\mathbf{x})$ the value assigned to $\mathbf{x}$ by a weighted kNN estimator, defined in equation 4.1, using the weighted squared ℓ_2-norm as the distances $d(\mathbf{x}, \mathbf{x}')$ and the nearest neighbors, $\mathcal{N}_{\mathbf{w}}(\mathbf{x})$, are found among the points of S excluding $\mathbf{x}$:

$$\hat{g}_{\mathbf{w}}(\mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{x}' \in \mathcal{N}_{\mathbf{w}}(\mathbf{x})} g(\mathbf{x}') e^{-\sum_i (x_i - x_i')^2 w_i^2 / \beta} \;, \quad \mathcal{N}_{\mathbf{w}}(\mathbf{x}) \subset S \setminus \mathbf{x}$$

The evaluation function is defined as the negative MSE of the weighted kNN estimator:

$$e(\mathbf{w}) = -\frac{1}{2} \sum_{\mathbf{x} \in S} \left(g(\mathbf{x}) - \hat{g}_w(\mathbf{x}) \right)^2 . \tag{4.2}$$

This evaluation function scores weight vectors ($\mathbf{w}$). A change of weights will cause a change in the distances and, possibly, the identity of each point's nearest neighbors, which will change the function estimates. A weight vector that induces a distance measure in which neighbors have similar labels receives a high score. Note that there is no explicit regularization term in $e(\mathbf{w})$. This is justified by the fact that for each point, the estimate of

Algorithm 4 $RGS(S, k, \beta, T)$

1. initialize $\mathbf{w} = (1, 1, \ldots, 1)$
2. for $t = 1 \ldots T$
   (a) pick randomly an instance $\mathbf{x}$ from S
   (b) evaluate the gradient of $e(\mathbf{w})$ on $\mathbf{x}$:

$$\begin{aligned} \nabla e(\mathbf{w}) &= -\left(g(\mathbf{x}) - \hat{g}_w(\mathbf{x})\right) \nabla_{\mathbf{w}} \hat{g}_{\mathbf{w}}(\mathbf{x}) \\ \nabla_{\mathbf{w}} \hat{g}_{\mathbf{w}}(\mathbf{x}) &= \frac{-\frac{4}{\beta} \sum_{\mathbf{x}'', \mathbf{x}' \in \mathcal{N}_{\mathbf{w}}(\mathbf{x})} g(x'') a(x', x'') \, \mathbf{u}(x', x'')}{\sum_{\mathbf{x}'', \mathbf{x}' \in \mathcal{N}_{\mathbf{w}}(\mathbf{x})} a(x', x'')} \end{aligned}$$

   where $a(x', x'') = e^{-\left(||x - x'||^2_{\mathbf{w}} + ||x - x''||^2_{\mathbf{w}}\right)/\beta}$
   and $\mathbf{u}(x', x'') \in R^N$ is a vector with $u_i = w_i \left[(x_i - x'_i)^2 + (x_i - x''_i)^2\right]$.
   (c) $\mathbf{w} = \mathbf{w} + \eta_t \nabla e(\mathbf{w}) = \mathbf{w}\left(1 + \eta_t \nabla_{\mathbf{w}} \hat{g}_{\mathbf{w}}(\mathbf{x})\right)$ where η_t is a decay factor.

its function value does not include that point as part of the training set. Thus, equation 4.2 is a leave-one-out cross validation error. Clearly, it is impossible to go over all the weight vectors (or even over all the feature subsets), and therefore some search technique is required.

Our method finds a weight vector $\mathbf{w}$ that locally maximizes $e(\mathbf{w})$ as defined in (4.2) and then uses a threshold in order to obtain a feature subset. The threshold can be set either by cross validation or by finding a natural cutoff in the weight values. However, we later show that using the distance measure induced by $\mathbf{w}$ in the regression stage compensates for taking too many features. Since $e(\mathbf{w})$ is defined over a continuous domain and is smooth almost everywhere we can use gradient ascent in order to maximize it. *RGS* (algorithm 4) is a stochastic gradient ascent over $e(\mathbf{w})$. In each step the gradient is evaluated using one sample point and is added to the current weight vector. *RGS* considers the weights of all the features at the same time and thus it can handle dependency on a group of features. This is demonstrated in section 4.3. In this respect, it is superior to selection algorithms that score each feature independently. It is also faster than methods that try to find a good subset directly by trial and error. Note, however, that convergence to a global optimum is not guaranteed, and standard techniques to avoid local optima can be used.

The parameters of the algorithm are k (number of neighbors), β (Gaussian decay factor), T (number of iterations) and $\{\eta_t\}_{t=1}^{T}$ (step size decay scheme). The value of k can be tuned

by cross validation, however a proper choice of β can compensate for a k that is too large. It makes sense to tune β to a value that places most neighbors in an active zone of the Gaussian. In our experiments, we set β to half of the mean distance between points and their k neighbors. It usually makes sense to use η_t that decays over time to ensure convergence, however, on our data, convergence was also achieved with $\eta_t = 1$.

The computational complexity of *RGS* is $\Theta(TNm)$ where T is the number of iterations, N is the number of features and m is the size of the training set S. This is correct for a naive implementation which finds the nearest neighbors and their distances from scratch at each step by measuring the distances between the current point to all the other points. RGS is basically an on-line method which can be used in batch mode by running it in epochs on the training set. When it is run for only one epoch, $T = m$ and the complexity is $\Theta\left(m^2 N\right)$. Matlab code for this algorithm (and those that we compare it with) is available at `www.cs.huji.ac.il/labs/learning/code/fsr/`

4.3 Testing on synthetic data

The use of synthetic data, where we can control the importance of each feature, allows us to illustrate the properties of our algorithm. We compare our algorithm to other common selection methods: *infoGain* [93], correlation coefficients (*corrcoef*) and *forward selection* (see [43]). *infoGain* and *corrcoef* simply rank features according to the mutual information[2] or the correlation coefficient (respectively) between each feature and the labels (i.e. the target function value). Forward selection (*fwdSel*) is a greedy method in which features are iteratively added into a growing subset. In each step, the feature showing the greatest improvement (given the previously selected subset) is added. This is a search method that can be applied to any evaluation function and we use our criterion (equation 4.2 on feature subsets). This well known method has the advantages of considering feature subsets and that it can be used with non linear predictors. Another algorithm we compare with scores each feature independently using our evaluation function (4.2). This helps us in analyzing *RGS*, as it may help single out the respective contributions to performance of the properties

[2] Feature and function values were "binarized" by comparing them to the median value.

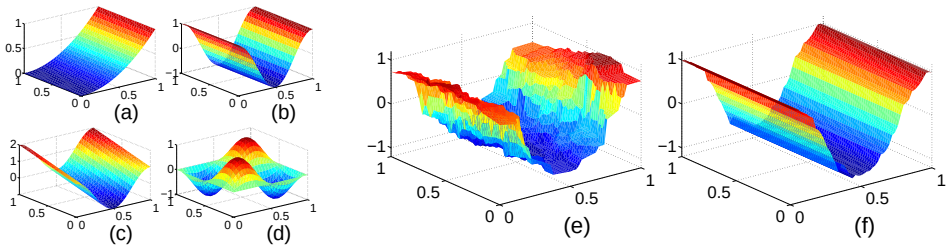

Figure 4.1: (a)-(d): Illustration of the four synthetic target functions. The plots shows the function's value as function of the first two features. (e),(f): demonstration of the effect of feature selection on estimating the second function using kNN regression ($k = 5$, $\beta = 0.05$). (e) using both features ($mse = 0.03$), (f) using the relevant feature only ($mse = 0.004$)

of the evaluation function and the search method. We refer to this algorithm as *SKS* (Single feature, kNN regression, feature Selection).

We look at four different target functions over $\mathbb{R}^{50}$. The training sets include 20 to 100 points that were chosen randomly from the $[-1, 1]^{50}$ cube. The target functions are given in the top row of figure 4.2 and are illustrated in figure 4.1(a-d). A random Gaussian noise with zero mean and a variance of $1/7$ was added to the function value of the training points. Clearly, only the first feature is relevant for the first two target functions, and only the first two features are relevant for the last two target functions. Note also that the last function is a smoothed version of parity function learning and is considered hard for many feature selection algorithms [43].

First, to illustrate the importance of feature selection on regression quality we use kNN to estimate the second target function. Figure 4.1(e-f) shows the regression results for target (b), using either only the relevant feature or both the relevant and an irrelevant feature. The addition of one irrelevant feature degrades the MSE ten fold. Next, to demonstrate the capabilities of the various algorithms, we run them on each of the above problems with varying training set size. We measure their success by counting the number of times that the relevant features were assigned the highest rank (repeating the experiment 250 times by re-sampling the training set). Figure 4.2 presents the success rate as function of training set size.

We can see that all the algorithms succeeded on the first function which is monotonic and depends on one feature alone. *infoGain* and *corrcoef* fail on the second, non-monotonic

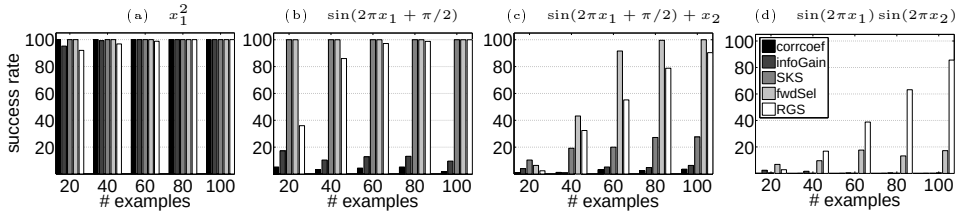

Figure 4.2: Success rate of the different algorithms on 4 synthetic regression tasks (averaged over 250 repetitions) as a function of the number of training examples. Success is measured by the percent of the repetitions in which the relevant feature(s) received first place(s).

function. The three kNN based algorithms succeed because they only depend on local properties of the target function. We see, however, that RGS needs a larger training set to achieve a high success rate. The third target function depends on two features but the dependency is simple as each of them alone is highly correlated with the function value. The fourth, XOR-like function exhibits a complicated dependency that requires consideration of the two relevant features simultaneously. *SKS* which considers features separately sees the effect of all other features as noise and, therefore, has only marginal success on the third function and fails on the fourth altogether. *RGS* and *fwdSel* apply different search methods. *fwdSel* considers subsets but can evaluate only one additional feature in each step, giving it some advantage over *RGS* on the third function but causing it to fail on the fourth. *RGS* takes a step in all features simultaneously. Only such an approach can succeed on the fourth function.

4.4 Hand Movement Reconstruction from Neural Activity

To suggest an interpretation of neural coding we apply *RGS* and compare it with the alternatives presented in the previous section[3] on the hand movement reconstruction task. The data sets were collected while a monkey performed a planar center-out reaching task with one or both hands [88]. 16 electrodes, inserted daily into novel positions in primary motor

[3] *fwdSel* was not applied due to its intractably high run time complexity. Note that its run time is at least r times that of *RGS* where r is the size of the optimal set and is longer in practice.

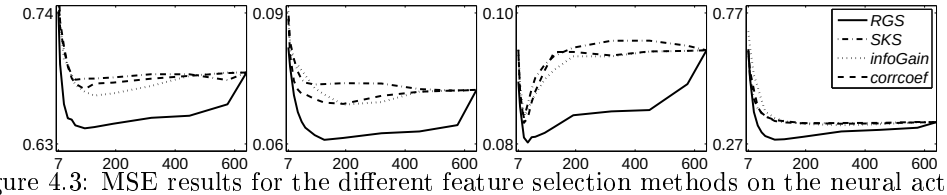

Figure 4.3: MSE results for the different feature selection methods on the neural activity data sets. Each sub figure is a different recording day. MSEs are presented as a function of the number of features used. Each point is a mean over all 5 cross validation folds, 5 permutations on the data and the two velocity component targets. Note that some of the data sets are harder than others.

cortex were used to detect and sort spikes in up to 64 channels (4 per electrode). Most of the channels detected isolated neuronal spikes by template matching. Some, however, had templates that were not tuned, producing spikes during only a fraction of the session. Others (about 25%) contained unused templates (resulting in a constant zero producing channel or, possibly, a few random spikes). The rest of the channels (one per electrode) produced spikes by threshold passing. We construct a labeled regression data set as follows. Each example corresponds to one time point in a trial. It consists of the spike counts that occurred in the 10 previous consecutive $100ms$ long time bins from all 64 channels ($64 \times 10 = 640$ features) and the label is the X or Y component of the instantaneous hand velocity. We analyze data collected over 8 days. Each data set has an average of 5050 examples collected during the movement periods of the successful trials.

In order to evaluate the different feature selection methods we separate the data into training and test sets. Each selection method is used to produce a ranking of the features. We then apply kNN (based on the training set) using different size groups of top ranking features to the test set. We use the resulting MSE (or correlation coefficient between true and estimated movement) as our measure of quality. To test the significance of the results we apply 5-fold cross validation and repeat the process 5 times on different permutations of the trial ordering. Figure 4.3 shows the average (over permutations, folds and velocity components) MSE as a function of the number of selected features on four of the different data sets (results on the rest are similar and omitted due to lack of space)[4]. It is clear that RGS achieves better results than the other methods throughout the range of feature numbers.

[4]We use $k = 50$ (approximately 1% of the data points). β is set automatically as described in section 4.2. These parameters were manually tuned for good kNN results and were not optimized for any of the feature selection algorithms. The number of epochs for RGS was set to 1 (i.e. $T = m$).

To test whether the performance of *RGS* was consistently better than the other methods we counted winning percentages (the percent of the times in which *RGS* achieved lower MSE than another algorithm) in all folds of all data sets and as a function of the number of features used. Figure 4.4 shows the winning percentages of *RGS* versus the other methods. For a very low number of features, while the error is still high, *RGS* winning scores are only slightly better than chance but once there are enough features for good predictions the winning percentages are higher than 90%. In figure 4.3 we see that the MSE achieved when using only approximately 100 features selected by *RGS* is better than when using all the features. This difference is indeed statistically significant (win score of 92%). If the MSE is replaced by correlation coefficient as the measure of quality, the average results (not shown due to lack of space) are qualitatively unchanged.

RGS not only ranks the features but also gives them weights that achieve locally optimal results when using kNN regression. It therefore makes sense not only to select the features but to weigh them accordingly. Figure 4.5 shows the winning percentages of *RGS* using the weighted features versus *RGS* using uniformly weighted features. The corresponding MSEs (with and without weights) on the first data set are also displayed. It is clear that using the weights improves the results in a manner that becomes increasingly significant as the number of features grows, especially when the number of features is greater than the optimal number. Thus, using weighted features can compensate for choosing too many by diminishing the effect of the surplus features.

To take a closer look at what features are selected, figure 4.6 shows the 100 highest ranking features for all algorithms on one data set. Similar selection results were obtained in the rest of the folds. One would expect to find that well isolated cells (template matching) are more informative than threshold based spikes. Indeed, all the algorithms select isolated cells more frequently within the top 100 features (*RGS* does so in 95% of the time and the rest in 70%-80%). A human selection of channels, based only on looking at raster plots and selecting channels with stable firing rates was also available to us. This selection was independent of the template/threshold categorization. Once again, the algorithms selected the humanly preferred channels more frequently than the other channels. Another and more interesting observation that can also be seen in the figure is that while *corrcoef, SKS* and

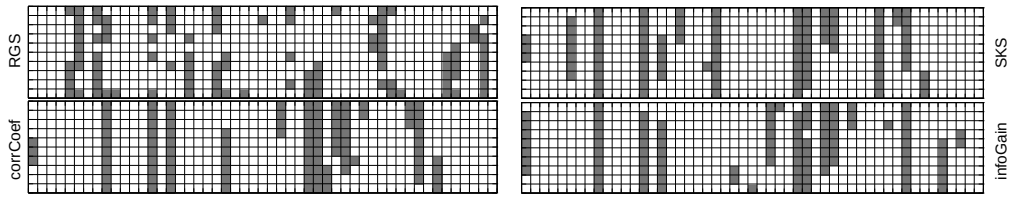

Figure 4.6: 100 highest ranking features (grayed out) selected by the algorithms. Results are for one fold of one data set. In each sub figure the bottom row is the ($100ms$) time bin with least delay and the higher rows correspond to longer delays. Each column is a channel (silent channels omitted).

infoGain tend to select all time lags of a channel, RGS's selections are more scattered (more channels and only a few time bins per channel). Since RGS achieves the best results, we

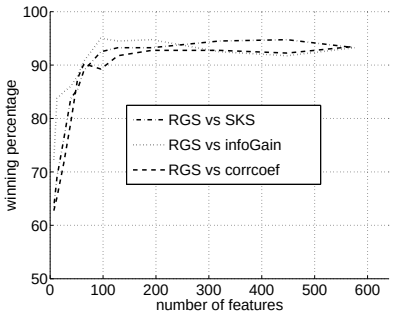

Figure 4.4: Winning percentages of RGS over the other algorithms. RGS achieves better MSEs consistently.

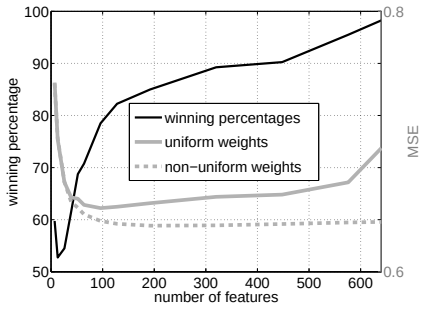

Figure 4.5: Winning percentages of RGS with and without weighting of features (black). Gray lines are corresponding MSEs of these methods on the first data set.

conclude that this selection pattern is useful. Apparently RGS found these patterns thanks to its ability to evaluate complex dependency on feature subsets. This suggests that such dependency of the behavior on the neural activity does exist.

4.5 Summary

In this chapter we presented a new method of selecting features for function estimation and use it to analyze neural activity during a motor control task . We used the leave-one-out mean squared error of the kNN estimator and minimize it using a gradient ascent on an

"almost" smooth function. This yields a selection method which can handle a complicated dependency of the target function on groups of features yet can be applied to large scale problems. This is valuable since many common selection methods lack one of these properties. By comparing the result of our method to other selection methods on the motor control task, we showed that consideration of complex dependency helps to achieve better performance. These results suggest that this is an important property of the code.

Chapter 5

Learning to Select Features[1]

In the standard framework of feature selection discussed in previous chapters, the task is to find a (small) subset of features out of a given set of features that is sufficient to predict the target labels well . Thus, the feature selection methods discussed so far of tell us which features are better. However, they do not tell us what *characterizes* these features or how to judge new features which were not evaluated using the labeled data. In this chapter we extend the standard framework and present a novel approach to the task of feature selection that attempts to learn *properties* of good features. We claim that in many cases it is natural to represent each feature by a set of properties, which we call *meta-features*. As a simple example, in image related tasks where the features are gray-levels of pixels, two meta-features can be the (x, y) position of each pixel. We use the training set in order to learn the relation between the meta-feature values and feature usefulness. This in turn enables us to predict the quality of unseen features. This may be useful in many applications, e.g., for predicting the future usefulness of a new word (that did not appear in the training set) for text classification. Another possible application is for predicting the worthiness of a candidate medical examination based on the properties of this examination and the properties and the worthiness of previous examinations. It is also a very useful tool for feature extraction. As described in section 1.1.2, in feature extraction the original features

[1] The results presented in this chapter were first presented in a paper titled "Learning to Select Features using their Properties" submitted to JMLR at 29 August, 2006.

are used to generate new "more complex" features. One example for such extracted features in image related tasks are all the products of sets of 3 pixels. Any function of the original features can be used as an extracted feature, and there are a virtually infinite number of such functions. Thus the learner has to decide which potential complex features have a good chance of being the most useful. For this task we derive a selection algorithm (called *Mufasa*) that uses meta-features to explore a huge number of candidate features efficiently. We also derive generalization bounds for the joint problem of feature selection (or extraction) and classification when the selection is made using meta-features. These bounds are better than the bounds obtained for direct selection.

We also show how our concept can be applied in the context of inductive transfer ([10, 110, 16]). As described in section 1.1.5, in inductive transfer one tries to use knowledge acquired in previous tasks to enhance performance on the current task. The rationale lies in the observation that a human does not learn isolated tasks, but rather learns many tasks in parallel or sequentially. In this context, one key question is the kind of knowledge we can transfer between tasks. One option, which is very popular, is to share the knowledge about the representation, or more specifically, knowledge about feature usefulness. Here we suggest and show that it might be better to share knowledge about the *properties* of good features instead of knowledge about which features are good. The problem of handwritten digit recognition is used throughout this chapter to illustrate the various applications of our novel approach.

Related work: [108] used meta-features of words for text classification when there are features (words) that are unseen in the training set, but appear in the test set. In their work the features are words and the meta-features are words in the neighborhood of that word. They used the meta-features to predict the role of words that are unseen in the training set. Generalization from observed (training) features to unobserved features is discussed in [66]. Their approach involves clustering the instances based on the observed features. What these works and ours have in common is that they all extend the learning from the standard instance-label framework to learning in the feature space. Our formulation here, however, is different and allows a mapping of the feature learning problem onto the standard supervised

learning framework (see Table 5.1). Another related model is *Budget Learning* ([75, 42]), that explores the issue of deciding which is the most valuable feature to measure next under a limited budget. Other ideas using feature properties to produce or select good features can be found in the literature and have been used in various applications. For instance [71] used this rationale in the context of inductive transfer for object recognition. Very recently, [95] also used this approach in the same context for text classification. They use a property of pairs of words which indicates whether they are synonyms or not for the task of estimating the words' covariance matrix. [58] used property-based clustering of features for handwritten Chinese recognition and other applications. Our formulation encompasses a more general framework and suggests a systematic way to use the properties as well as derive algorithms and generalization bounds for the combined process of feature selection and classification.

5.1 Formal Framework

Recall that in the standard supervised learning framework (see section 1.1.1) it is assumed that each instance is a vector $\mathbf{x} \in \mathbb{R}^N$ and the N coordinates are the features. However, we can also consider the instances as *abstract entities* in the space $\mathcal{S}$ and think of the features as *measurements* on the instances. Thus each feature f can be considered as a function from $\mathcal{S}$ to $\mathbb{R}$, i.e., $f : \mathcal{S} \to \mathbb{R}$. We denote the set of all the features by $\{f_j\}_{j=1}^N$. In the following we use the term *feature* to describe both raw input variables (e.g., pixels in an image) and variables constructed from the original input variables using some function (e.g., product of 3 pixels in the image).

Also, recall that the standard task of feature selection is to select a subset of the given N features that enables good prediction of the label (see section 1.2). As described in the previous chapters, this is done by looking for features which are more useful than the others. However, as already mentioned, in this chapter we want to extend this framework and to find what *characterizes* the better features. Thus we further assume that each feature is described by a set of "properties" $\mathbf{u}(\cdot) = \{u_r(\cdot)\}_{r=1}^k$ which we call *meta-features*. Formally, each $u_r(\cdot)$ is a function from the space of possible measurements to $\mathbb{R}$. Thus each feature f is described by a vector $\mathbf{u}(f) = (u_1(f), \ldots, u_k(f)) \in \mathbb{R}^k$. Note that the meta-features are

Table 5.1: Feature learning by meta-features as a standard supervised learning

Training set	Features described by meta-features
Test set	Unobserved features
Labels	Feature quality
Hypothesis class	Class of mappings from meta-features to quality
Generalization in feature selection	Predicting the quality of new features
Generalization in the joint problem	Low classification error

not dependent on the instances, a fact which is important for understanding our analysis in what follows. We also denote a general point in the image of $\mathbf{u}(\cdot)$ by $\mathbf{u}$. The new framework we introduce in this chapter forces us to introduce some not very standard notations. In order to make it easier for the reader to become familiarized with this notation we provide a summary table in the appendix A at the end of this chapter.

5.2 Predicting the Quality of Features

Before we describe the more useful applications, we start by describing the most obvious use of the framework presented in this chapter. Here we assume that we observe only a subset of the N features; i.e., that in the training set we see only the value of some of the features. We can directly measure the quality (i.e. usefulness) of these features using the training set, but we also want to be able to predict the quality of the unseen features. Thus we want to think of the training set not only as a "training set of instances", but also as a "training set of features".

Thus we want to predict the quality of a new unseen feature. At this stage we evaluate the quality of each feature alone[2]. More formally, our goal is to use the training set S^m and the set of meta-features for learning a mapping $\hat{Q} : \mathbb{R}^k \longrightarrow \mathbb{R}$ that predicts the quality of a feature using the values of its meta-features. For this we assume that we have a way to measure the quality of each feature that does appear in the training set. This measure can be any kind of standard evaluation function that uses the labeled training set to evaluate features (e.g., Infogain or wrapper based). Y_{MF} denotes the vector of measured

[2] We discuss the evaluation of subsets in sections 5.3 and 5.6.

Algorithm 5 $\hat{Q}$ =quality_map(S^m, featquality, regalg)

1. measure the feature quality vector: $Y_{MF} = \text{featquality}(S^m)$
2. calculate the $N \times k$ meta features matrix X_{MF}
3. use the regression alg. to learn a mapping from meta feature value to quality: $\hat{Q} = \text{regalg}(X_{MF}, Y_{MF})$

qualities, i.e. $Y_{MF}(j)$ is the measured quality of the j's feature in the training set. Now we have a new supervised learning problem, with the **original features** as **instances**, the **meta-features** as **features** and Y_{MF} as the (continuous) target **label**. The analogy to the standard supervised problem is summarized in Table 5.1. Thus we can use any standard regression learning algorithm to find the required mapping from meta-features to quality. The above procedure is summarized in algorithm 5. Note that this procedure uses a standard regression learning procedure. That is, the generalization ability to new features can be derived using standard generalization bounds for regression learning.

We now present a toy illustration of the ability to predict the quality of unseen features by the above procedure using a handwritten digit recognition task (OCR). For this purpose, we used the MNIST ([70]) dataset which contains images of 28×28 pixels of centered digits $(0 \ldots 9)$. We converted the pixels from gray-scale to binary by thresholding. Here we use the 784 original pixels as the input features and the (x, y) location of the pixel as meta-features. Recall that we do not try to generalize along the instance dimension, but instead try to generalize along the feature dimension, using the meta-features. Thus we first choose a fixed set of 2000 images from the dataset. Then we put aside a random subset of the 392 features as a test set of features. Then we use a growing number of training features, which were chosen randomly out of the remaining 392 features. Our measure of quality here is Infogain. Now, given a training set of features we use a linear regression[3] to learn a mapping from meta-features to quality. Using this mapping we predict the quality of the 392 testing features.

We check the accuracy of the prediction by comparing it to the quality that was measured

[3]The linear regression is done over an RBF representation of the (x, y) location. We used 49 Gaussians (with std=3 pixels) located on a grid over the image and represent the location by a 49-dimensional vector of the responses of the Gaussians in this location.

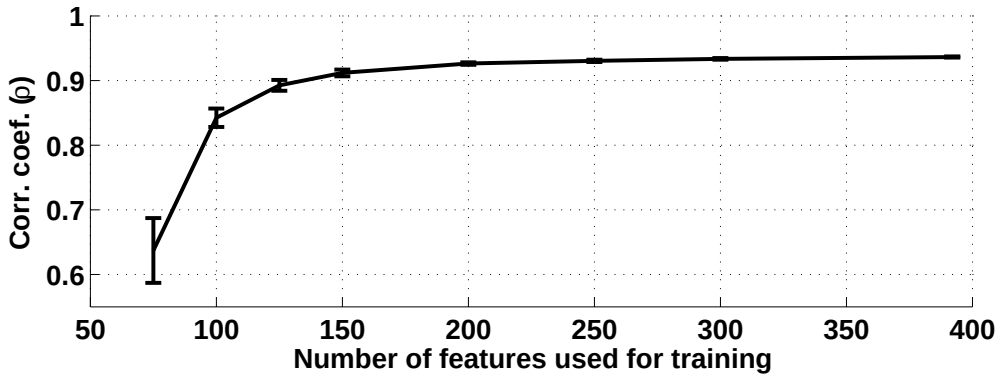

Figure 5.1: The correlation coefficient between the predicted quality and the real quality of testing features as a function of the number of training features. Error bars show the standard deviation.

by directly applying Infogain on the testing features. We do the comparison in two different ways: (1) by correlation coefficient between the two. (2) By checking the accuracy of the 1-Nearest-Neighbor (1-NN) classifier on a test set of 2000 instances, using a growing number of selected features out of the 392 testing features. The features were sorted by either the predicted quality or the direct Infogain. In order to check the statistical significance, we repeat the experiment 20 times by re-splitting into train and test sets. The results are presented in Figure 5.1 and Figure 5.2 respectively. It is clear from the graphs that if we have enough training features, we can predict new feature qualities with high accuracy. In Figure 5.2 we see that the classification error of using the predicted quality is similar to the error obtained by direct use of Infogain. This occurs even though our prediction was done using only $\sim 1/8$ of the features.

5.3 Guided Feature Extraction

Here we show how the low dimensional representation of features by a relatively small number of meta-features enables efficient selection even when the number of potential features is very large or even infinite. This is highly relevant to the feature extraction scenario. We tackle the problem of choosing which features to generate out of the huge number of potential features in the following way. Instead of evaluating all features, we evaluate a relatively

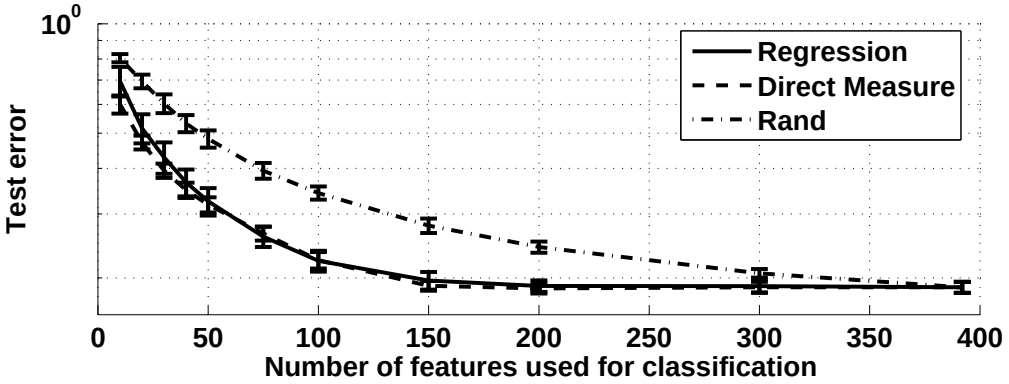

Figure 5.2: The classification error of 1-NN on a testing set of instances as a function of the number of the top features ranked by either prediction using 100 training features or by direct Infogain on test features. The results of random ranking is also presented. Error bars show the standard deviation.

small number of features and learn to predict which other features are better. This way we can guide our search for good features.

5.3.1 Meta-features Based Search

Assume that we want to select (or extract) a set of n features out of the large number of N potential features. We define a stochastic mapping from values of meta-features to selection (or extraction) of features. More formally, let V be a random variable that indicates which feature is selected. We assume that each point $\mathbf{u}$ in the meta-feature space induces density $p(v|\mathbf{u})$ over the features. Our goal is to find a point $\mathbf{u}$ in the meta-feature space such that drawing n features (independently) according to $p(v|\mathbf{u})$ has a high probability of giving us a good set of n features. For this purpose we suggest the *Mufasa* (for Meta-Features Aided Search Algorithm) (algorithm 6) which uses stochastic local search in the meta-feature space.

Note that *Mufasa* does not use explicit prediction of the quality of unseen features as we did in section 5.2, but it is clear that it cannot work unless the meta-features are informative on the quality. Namely, *Mufasa* can only work if the chance of drawing a good set of features from $p(v|\mathbf{u})$ is some continuous function of $\mathbf{u}$, i.e., a small change in $\mathbf{u}$ results in a small change in the chance of drawing a good set of features. If, in addition, the meta-features

Algorithm 6 F_{best} =Mufasa(n, J)

1. Initialization: $q_{best} = maxreal$, $\mathbf{u} \leftarrow \mathbf{u}_0$, where $\mathbf{u}_0$ is the initial guess of $\mathbf{u}$.
2. For $j = 1 \ldots J$
   (a) Select (or generate) a new set F_j of n random features according to $p(w|\mathbf{u})$.
   (b) q_j = quality(F_j). (Any measure of quality, e.g. cross-validation classification accuracy)
   (c) If $q_j \geq q_{best}$
   $F_{best} = F_j$, $\mathbf{u}_{best} = \mathbf{u}$, $q_{best} = q_j$
   (d) Randomly select new $\mathbf{u}$ which is near $\mathbf{u}_{best}$. For example: $\mathbf{u} \leftarrow \mathbf{u}_{best} + noise$.
3. return F_{best}

space is "simple"[4] we expect it to find a good point in a small number of steps. The number of steps plays an important role in the generalization bounds we present in section 5.4.1. Like any local search over a non-convex target function, a convergence to a global optimum is not guaranteed, but any standard technique to avoid local maxima can be used. In section 5.3.2 we demonstrate the ability of *Mufasa* to efficiently select good features in the presence of a huge number of candidate (extracted) features on the handwritten digit recognition problem. In section 5.4.1 we present a theoretical analysis of *Mufasa*.

5.3.2 Illustration on Digit Recognition Task

In this section we revisit the handwritten digit recognition problem described in section 5.2 and use it to demonstrate how *Mufasa* works for feature extraction. Once again we use the MNIST dataset, but this time we deal with more complicated extracted features. These extracted features are in the following form: logical AND of 1 to 8 pixels (or their negation), which are referred to as *inputs*. In addition, a feature can be shift invariant for shifts of up to *shiftInvLen* pixels. That means we take a logical OR of the output of all the different locations we can get by shifting.

Thus a feature is defined by specifying the set of *inputs*, which of the inputs are negated, and the value of *shiftInvLen*. Similar features were already used by [30] on the MNIST dataset, but with a fixed number of inputs and without shift invariance. The idea of using

[4] We elaborate on the meaning of "simple" in sections 5.4 and 5.6.

shift invariance for digit recognition is also not new, and was used for example by [104]. It is clear that there are a huge number of such features; thus we have no practical way to measure or use all of them. Therefore we need some guidance for the extraction process, and here is the point where the meta-features framework comes in.

We use the following four meta-features:

1. *numInputs*: the number of inputs (1-8).
2. *precentPos*: percent of logic positive pixels (0-100, rounded).
3. *shiftInvLen*: maximum allowed shift value (1-8).
4. *scatter*: average distance of the inputs from their center of gravity (COG) (1-3.5).

In order to find a good value for meta-features we use *Mufasa* (algorithm 6), with different values of allowed budget. We use a budget (and not just the number of features) since features with large *shiftInvLen* are more computationally expensive. We defined the cost of measuring (calculating) a feature as $0.5\left(1+a^2\right)$, where a is the *shiftInvLen* of the feature; this way the cost is proportional to the number of locations we measure the feature. We use 2000 images as a training set, and the number of steps, J, is 50. We choose specific features, given a value of meta-features, by re-drawing features randomly from a uniform distribution over the features that satisfy the given value of the meta-features until the full allowed budget is used up. We use 2-fold cross validation of the linear Support Vector Machine (SVM) ([14]) to check the quality of the set of selected features in each step. We use the multi-class SVM toolbox developed by [23]. Finally, for each value of allowed budget we check the results obtained by the linear SVM (that uses the selected features) on a test set of another 2000 images using the selected features.

We compare the results with those obtained using the features selected by Infogain as follows. We first draw features randomly using a budget which is 50 times larger, then we sort them by Infogain normalized by the cost[5] and take a prefix that uses the allowed budget (referred as MF+Norm Infogain). As a sanity check, we also compare the results to those obtained by doing 50 steps of choosing features of the allowed budget randomly, i.e. over all possible values of the meta-features. Then we use the set with the lowest 2-fold cross-validation error (referred as MF+Rand). We also compare our results with

[5]Infogain without normalization gives worse results.

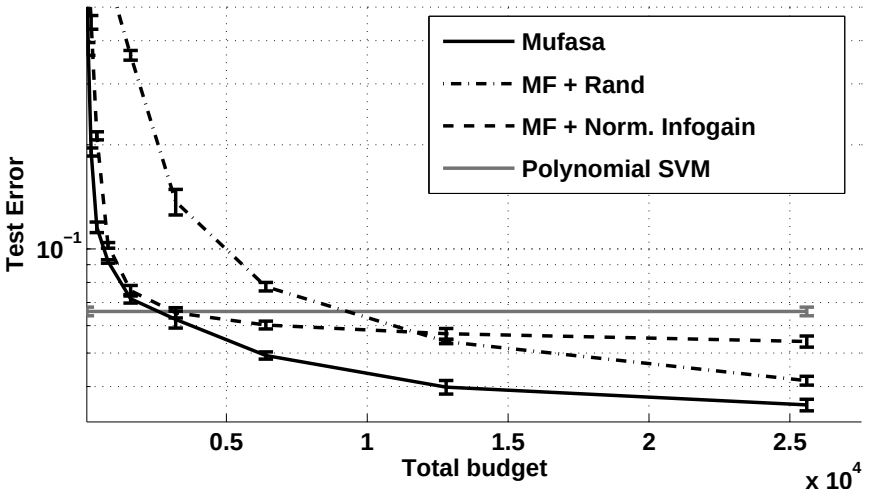

Figure 5.3: **Guided feature extraction for digit recognition**. The generalization error rate as a function of the available budget for features, using different selection methods. The number of training instances is 2000. Error bars show a one standard deviation confidence interval. SVM is not limited by the budget, and always implicitly uses all the products of features. We only present the results of SVM with a polynomial kernel of degree 2, the value that gave the best results in this case.

SVM with a polynomial kernel of degree 1-4, that uses the original pixels as input features. This comparison is relevant since SVM with a polynomial kernel of degree k implicitly uses ALL the products of up to k pixels, and the product is equal to AND for binary pixels. To evaluate the statistical significance, we repeat each experiment 10 times, with different partitions into train and test sets. The results are presented in Figure 5.3. It is clear that *Mufasa* outperforms the budget-dependent alternatives, and outperforms SVM for budgets larger than 3000 (i.e. about 600 features). It is worth mentioning that our goal here is not to compete with the state-of-art results on MNIST, but to illustrate our concept and to compare the results of the same kind of classifier with and without using our meta-features guided search. Note that our concept can be combined with most kinds of classification, feature selection, and feature extraction algorithms to improve them as discussed in section 5.8.

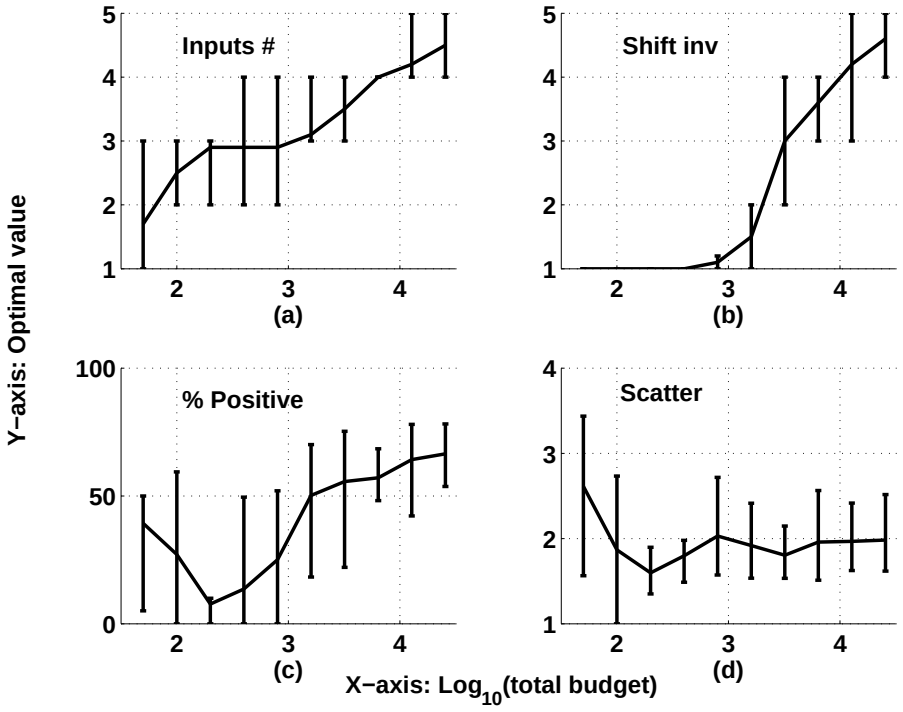

Figure 5.4: **Optimal value of the different meta features as a function of the budget**. Error bars indicate the range where values fall in 90% of the runs. We can see that the size of the optimal shift invariance, the optimal number input and optimal percent of positive inputs grow with the budget.

Another benefit of the meta-features guided search is that it helps understand the problem. To see this we need to take a closer look at the chosen values of the meta-features ($\mathbf{u}_{best}$) as a function of the available budget. Figure 5.4 presents the average chosen value of each meta-feature as a function of the budget. We can see (in Figure 5.4b) that when the budget is very limited, it is better to take more cheap features rather than fewer more expensive shift invariant features. On the other hand, when we increase the budget, adding these expensive complex features is worth it. We can also see that when the budget grows, the optimal number of inputs grows as does the optimal percent of positive inputs . This occurs because for a small budget, we prefer features that are less specific, and have relatively high entropy, at the expense of "in class variance". For a large budget, we can permit ourselves to use sparse features (low probability of being 1), but gain specificity. For the scatter meta-features, there is apparently no correlation between the budget and the optimal value.

The vertical lines (error bars) represent the range of selected values in the different runs. It gives us a sense of the importance of each meta-feature. A smaller error bar indicates higher sensitivity of the classifier performance to the value of the meta-feature. For example, we can see that performance is sensitive to *shiftInvLen* and relatively indifferent to *percentPos*.

5.4 Theoretical Analysis

In this section we derive generalization bounds for the combined process of selection and classification, when the selection process is based on meta-features. We show that in some cases, these bounds are far better than the bounds that assume each feature can be selected directly. This is because we can significantly narrow the number of possible selections, and still find a good set of features. In section 5.4.1 we perform the analysis for the case where the selection is made using *Mufasa* (algorithm 6). In section 5.4.2 we present a more general analysis, which is independent of the selection algorithm, but instead assumes that we have a given class of mappings from meta-features to a selection decision.

5.4.1 Generalization Bounds for *Mufasa* Algorithm

The bounds presented in this section assume that the selection is made using *Mufasa* (algorithm 6), but they could be adapted to other meta-feature based selection algorithms. Before presenting the bounds, we need some additional notations. We assume that the classifier that is going to use the selected features is chosen from a hypothesis class $\mathcal{H}_c$ of real valued functions and the classification is done by taking the sign.

We also assume that we have a hypothesis class $\mathcal{H}_{fs}$, where each hypothesis is one possible way to select the n out of N features. Using the training set, our feature selection is limited to selecting one of the hypotheses that is included in $\mathcal{H}_{fs}$. As we show later, if $\mathcal{H}_{fs}$ contains all the possible ways of choosing n out of N features, then we get an unattractive generalization bound for large values of n and N. Thus we use meta-features to further restrict the cardinality (or complexity) of $\mathcal{H}_{fs}$. We have a combined learning scheme of choosing both $h_c \in \mathcal{H}_c$ and $h_{fs} \in \mathcal{H}_{fs}$. We can view it as choosing a single classifier from $\mathcal{H}_{fs} \times \mathcal{H}_c$. In the following paragraphs we analyze the equivalent size of hypothesis space

$\mathcal{H}_{fs}$ of *Mufasa* as a function of the number of steps of the algorithm.

For the theoretical analysis, we can reformulate *Mufasa* into an equivalent algorithm that is split into two stages, called *Mufasa1* and *Mufasa2*. The first stage (*Mufasa1*) is randomized, but does not use the training set. The second stage (*Mufasa2*) is deterministic, and performs the described search. Since *Mufasa2* is deterministic, we must generate all potentially required hypotheses in *Mufasa1*, using the same probability distribution of feature generating that may be required by step 2a of the algorithm. Then, during the search, the second stage can deterministically select hypotheses from the hypotheses created in the first stage, rather than generate them randomly. Eventually, the second stage selects a single good hypothesis out of the hypotheses created by *Mufasa1* using the search. Therefore, the size of $\mathcal{H}_{fs}$, denoted by $|\mathcal{H}_{fs}|$, is the number of hypotheses created in *Mufasa1*.

The following two lemmas upper bound $|\mathcal{H}_{fs}|$, which is a dominant quantity in the generalization bound. The first one handles the case where the meta features has discrete values, and there are a relatively small number of possible values for the meta-features. This number is denoted by $|MF|$.

Lemma 5.1 *Any run of* Mufasa *can be duplicated by first generating* $J|MF|$ *hypotheses by* Mufasa1 *and then running* Mufasa2 *using these hypotheses only, i.e. using* $|\mathcal{H}_{fs}| \leq J|MF|$, *where* J *is the number of iterations made by* Mufasa *and* $|MF|$ *is the number of different values the meta-features can be assigned.*

Proof. Let *Mufasa1* generate J random feature sets for each of the $|MF|$ possible values of meta-features. The total number of sets we get is $J|MF|$. We have only J iterations in the algorithm, and we generated J feature sets for each possible value of the meta-features. Therefore, it is guaranteed that all hypotheses required by *Mufasa2* are available. □

Note that in order to use the generalization bound of the algorithm, we cannot consider only the subset of J hypotheses that was tested by the algorithm. This is because this subset of hypotheses is affected by the training set (just as one cannot choose a single hypothesis using the training set, and then claim that the hypotheses space of the classifier includes only one hypothesis). However, from lemma 5.1, we get that the algorithm search within

no more than $J|MF|$ feature selection hypotheses, that were determined *without* using the training set.

The next lemma handles the case where the cardinality of all possible values of meta-features is large relative to 2^J, or even infinite. In this case we can get a tighter bound that depends on J but not on $|MF|$.

Lemma 5.2 *Any run of* Mufasa *can be duplicated by first generating* 2^J *hypotheses by* Mufasa1 *and then running* Mufasa2 *using these hypotheses only, i.e. using* $|\mathcal{H}_{fs}| \leq 2^J$, *where* J *is the number of iterations* Mufasa *performs.*

Proof. In *Mufasa1,* we build the entire tree of possible random updating of meta-features (step 2d of algorithm 6) and generate the features. Since we do not know the label, in the jth iteration we have 2^{j-1} hypotheses we need to generate. The total number of hypotheses is then $\left(2^J - 1\right)$. □

To state our theorem we also need the following standard definitions:

Definition 5.1 *Let* $\mathcal{D}$ *be a distribution over* $\mathcal{S} \times \{\pm 1\}$ *and* $h : \mathcal{S} \to \{\pm 1\}$ *a classification function. We denote by* $er_{\mathcal{D}}(h)$ *the generalization error of* h *w.r.t* $\mathcal{D}$*:*

$$er_{\mathcal{D}}(h) = Pr_{s,y\sim\mathcal{D}}[h(s) \neq y]$$

For a sample $S^m = \{(s_k, y_k)\}_{k=1}^m \in (\mathcal{S} \times \{\pm 1\})^m$ *and a constant* $\gamma > 0$, *the* γ*-sensitive training error is:*

$$\hat{er}_S^{\gamma}(h) = \frac{1}{m}|\{i : y_i h(s_i) < \gamma\}|$$

Now we are ready to present our main theoretical results:

Theorem 5.1 *Let* $\mathcal{H}_c$ *be a class of real valued functions. Let* S *be a sample of size* m *generated i.i.d from a distribution* $\mathcal{D}$ *over* $\mathcal{S} \times \{\pm 1\}$. *If we choose a set of features using* Mufasa, *with a probability of* $1 - \delta$ *over the choices of* S, *for every* $h_c \in \mathcal{H}_c$ *and every* $\gamma \in (0, 1]$*:*

$$er_{\mathcal{D}}(h_c) \leq \hat{er}_S^{\gamma}(h_c) +$$

$$\sqrt{\frac{2}{m}\left(d\ln\left(\frac{34em}{d}\right)\log(578m)+\ln\left(\frac{8}{\gamma\delta}\right)+g\left(J\right)\right)}$$

where $d = fat_{\mathcal{H}_c}(\gamma/32)$ *and* $g(J) = \min(J\ln 2, \ln(J|MF|))$ *(where* J *is the number of steps* Mufasa *does and* $|MF|$ *is the number of different values the meta-features can be assigned, if this value is finite, and* ∞ *otherwise).*

Our main tool in proving the the above theorem is the following theorem:

Theorem 5.2 *(Bartlett, 1998)*

Let $\mathcal{H}$ *be a class of real valued functions. Let* S *be a sample of size* m *generated i.i.d from a distribution* $\mathcal{D}$ *over* $\mathcal{S}\times\{\pm 1\}$ *; then with a probability of*$1-\delta$ *over the choices of* S*, every* $h\in\mathcal{H}$ *and every* $\gamma\in(0,1]$*:*

$$er_{\mathcal{D}}(h) \leq \hat{er}_S^{\gamma}(h)+$$

$$\sqrt{\frac{2}{m}\left(d\ln\left(\frac{34em}{d}\right)\log(578m)+\ln\left(\frac{8}{\gamma\delta}\right)\right)}$$

where $d = fat_{\mathcal{H}}(\gamma/32)$

Proof. (of theorem 5.1)

Let $\left\{F_1,...,F_{|\mathcal{H}_{fs}|}\right\}$ be all possible subsets of selected features. From theorem 5.2 we know that

$$er_{\mathcal{D}}(h_c,F_i) \leq \hat{er}_S^{\gamma}(h_c,F_i)+$$

$$\sqrt{\frac{2}{m}\left(d\ln\left(\frac{34em}{d}\right)\log(578m)+\ln\left(\frac{8}{\gamma\delta_{F_i}}\right)\right)}$$

where $er_{\mathcal{D}}(h_c,F_i)$ denote the generalization error of the selected hypothesis for the fixed set of features F_i.

By choosing $\delta_F = \delta/|\mathcal{H}_{fs}|$ and using the union bound, we get that the probability that there exist F_i $(1\leq i\leq|\mathcal{H}_{fs}|)$ such that the equation below does not hold is less than δ

$$er_{\mathcal{D}}\left(h_c\right) \leq \hat{er}_S^{\gamma}\left(h_c\right)+$$

$$\sqrt{\frac{2}{m}\left(d\ln\left(\frac{34em}{d}\right)\log(578m)+\ln\left(\frac{8}{\gamma\delta}\right)+\ln|\mathcal{H}_{fs}|\right)}$$

Therefore, with a probability of $1-\delta$ the above equation holds for *any* algorithm that selects one of the feature sets out of $\left\{F_1,...,F_{|H_{fs}|}\right\}$. Substituting the bounds for $|\mathcal{H}_{fs}|$ from lemma 5.1 and lemma 5.2 completes the proof. □

An interesting point in this bound is that it is independent of the number of selected features and of the total number of possible features (which may be infinite in case of feature generation). Nevertheless, it can select a good set of features out of $O\left(2^J\right)$ candidate sets. These sets may be non-overlapping, so the potential number of features that are candidates is $O\left(n2^J\right)$. For comparison, in chapter 3 we presented a same kind of bound but for direct feature selection. The bound we presented there has the same form as the bound we present here, but where $g(J)$ is replaced by a term of $O(\ln N)$, which is typically much larger than $J\ln 2$. If we substitute $N=n2^J$, then for the experiment described in section 5.3.2, $n\ln N = Jn(\ln 2n) \cong 375000$ while $\ln(J|MF|) \cong 11$.

5.4.2 VC-dimension of Joint Feature Selection and Classification

In the previous section we presented an analysis which assumes that the selection of features is made using *Mufasa*. In this section we turn to a more general analysis, which is independent of the specific selection algorithm, and rather assumes that we have a given class $\mathcal{H}_s$ of mappings from meta-features to a selection decision. Formally, $\mathcal{H}_s$ is a class of mappings from a meta-features value to $\{0,1\}$, that is, for each $h_s \in \mathcal{H}_s$, $h_s : \mathbb{R}^k \to \{0,1\}$. h_s defines which features are selected as follows:

$$f \text{ is selected} \iff h_s(\mathbf{u}(f)) = 1$$

where, as usual, $\mathbf{u}(f)$ is the value of the meta-features for feature f. Given the values of the meta-features of all the features together with h_s we get a single feature selection hypothesis. Therefore, $\mathcal{H}_s$ and the set of possible values of meta-features indirectly defines our feature selection hypothesis class, $\mathcal{H}_{fs}$. Since we are interested in selecting exactly n features (n is predefined), we use only a subset of $\mathcal{H}_s$ where we include only functions that imply selection of n features[6]. For simplicity, in the analysis we use the VC-dim of $\mathcal{H}_s$

[6] Note that one valid way to define $\mathcal{H}_s$ is by applying a threshold on a class of mappings from meta-features value to feature quality, $\hat{Q} : \mathbb{R}^k \to \mathbb{R}$. See, e.g., example 2 at the end of this section.

without this restriction, which is an upper bound of the VC-dim of the restricted class.

Our goal is to calculate an upper bound on the VC-dimension ([117]) of the joint problem of feature-selection and classification. To achieve this, we first derive an upper bound on $|\mathcal{H}_{fs}|$ as a function of VC-dim $(\mathcal{H}_s)$ and the number of features N.

Lemma 5.3 *Let $\mathcal{H}_s$ be a class of mappings from the meta-feature space ($\mathbb{R}^k$) to $\{0,1\}$, and let $\mathcal{H}_{fs}$ be the induced class of feature selection schemes; the following inequality holds:*

$$|\mathcal{H}_{fs}| \leq \left(\frac{eN}{\textit{VC-dim}\,(\mathcal{H}_s)}\right)^{\textit{VC-dim}(\mathcal{H}_s)}$$

Proof. The above inequality follows directly from the well known fact that a class with VC-dim d cannot give more than $\left(\frac{em}{d}\right)^d$ different partitions of a sample of size m (see, for example, [60] pp. 57).

□

The next lemma relates the VC dimension of the classification concept class (d_c), the cardinality if the selection class ($|\mathcal{H}_{fs}|$) and the VC-dim of the joint learning problem.

Lemma 5.4 *Let $\mathcal{H}_{fs}$ be a class of the possible selection schemes for selecting n features out of N and let $\mathcal{H}_c$ be a class of classifiers over $\mathbb{R}^n$. Let $d_c = d_c\,(n)$ be the VC-dim of $\mathcal{H}_c$. If $d_c \geq 11$ then the VC-dim of the combined problem (i.e. choosing $(h_{fs}, h_c) \in \mathcal{H}_{fs} \times \mathcal{H}_c$) is bounded by $(d_c + \log|\mathcal{H}_{fs}| + 1)\log d_c$.*

Proof. For a given set of selected features, the possible number of classifications of m instances is upper bounded $\left(\frac{em}{d_c}\right)^{d_c}$ (see [60] pp. 57). Thus, for the combined learning problem, the total number of possible classifications of m instances is upper bounded by $|\mathcal{H}_{fs}|\left(\frac{em}{d_c}\right)^{d_c}$. The following chain of inequalities shows that if $m = (d_c + \log|\mathcal{H}_{fs}| + 1)\log d_c$ then $|\mathcal{H}_{fs}|\left(\frac{em}{d_c}\right)^{d_c} < 2^m$:

$$
\begin{aligned}
|\mathcal{H}_{fs}| \left(\frac{e\,(d_c + \log|\mathcal{H}_{fs}| + 1)\log d_c}{d_c} \right)^{d_c} &= |\mathcal{H}_{fs}|\,(e \log d_c)^{d_c} \left(1 + \frac{\log|\mathcal{H}_{fs}| + 1}{d_c} \right)^{d_c} \\
&\leq e\,(|\mathcal{H}_{fs}|)^{1+\log e}\,(e \log d_c)^{d_c} && (5.1) \\
&\leq (|\mathcal{H}_{fs}|)^{2+\log e}\,(e \log d_c)^{d_c} && (5.2) \\
&\leq (|\mathcal{H}_{fs}|)^{2+\log e}\, d_c^{d_c+1} && (5.3) \\
&\leq d_c^{d_c+1}\,(|\mathcal{H}_{fs}|)^{\log d_c} && (5.4) \\
&= d_c^{d_c+1} d_c^{(\log|\mathcal{H}_{fs}|)} && (5.5) \\
&= 2^{(d_c+1+\log|\mathcal{H}_{fs}|)\log d_c}
\end{aligned}
$$

where we used the following equations / inequalities:

(5.1) $(1 + a/d)^d \leq e^a \qquad \forall a, d > 0$

(5.2) here we assume $|\mathcal{H}_{fs}| > e$, otherwise the lemma is trivial

(5.3) $(e \log d)^d \leq d^{d+1} \qquad \forall d \geq 1$

(5.4) $\log d_c > 2$ (since $d_c \geq 11$)

(5.5) $a^{\log b} = b^{\log a} \qquad \forall a, b > 1$

Therefore, $(d_c + \log|\mathcal{H}_{fs}| + 1)\log d_c$ is an upper bound on VC-dim of the combined learning problem.

□

Now we are ready to state the main theorem of this section.

Theorem 5.3 *Let $\mathcal{H}_s$ be a class of mappings from the meta-feature space ($\mathbb{R}^k$) to $\{0, 1\}$, let $\mathcal{H}_{fs}$ be the induced class of feature selection schemes for selecting n out of N features and let $\mathcal{H}_c$ be a class of classifiers over $\mathbb{R}^n$. Let $d_c = d_c(n)$ be the VC-dim of the $\mathcal{H}_c$. If $d_c \geq 11$, then the VC-dim of the joint class $\mathcal{H}_{fs} \times \mathcal{H}_c$ is upper bounded as follows*

$$VC\text{-}dim\,(\mathcal{H}_{fs} \times \mathcal{H}_c) \leq \left(d_c + d_s \log \frac{eN}{d_s} + 1 \right) \log d_c$$

where d_s is the VC-dim of $\mathcal{H}_s$.

The above theorem follows directly by substituting lemma 5.3 in lemma 5.4.

To illustrate the gain of the above theorem we calculate the bound for a few specific choices of $\mathcal{H}_s$ and $\mathcal{H}_c$:

1. First, we should note that if we do not use meta-features, but consider all the possible ways to select n out of N features the above bound is replaced by

$$\left(d_c + \log \left(\begin{array}{c} N \\ n \end{array} \right) + 1 \right) \log d_c \tag{5.6}$$

   which is very large for reasonable values of N and n.

2. Assuming that both $\mathcal{H}_s$ and $\mathcal{H}_c$ are classes of linear classifiers on $\mathbb{R}^k$ and $\mathbb{R}^n$ respectively, then $d_s = k+1$ and $d_c = n+1$ and we get that the VC of the combined problem of selection and classification is upper bounded by

$$O\left((n + k \log N) \log n \right)$$

   If $\mathcal{H}_c$ is a class of linear classifiers, but we allow any selection of n features the bound is (by substituting in 5.6):

$$O\left((n + n \log N) \log n \right)$$

   which is much larger if $k \ll n$. Thus in the typical case where the number of meta-features is much smaller than the number of selected features (e.g. in section 5.3.2) the bound for meta-feature based selection is much smaller.

3. Assuming that the meta-features are binary and $\mathcal{H}_s$ is the class of all possible functions from meta-feature to $\{0, 1\}$, then $d_s = 2^k$ and the bound is

$$O\left(\left(d_c + 2^k \log N \right) \log d_c \right)$$

   which is still might be much better than the bound in equation 5.6 if $k \ll \log n$.

5.5 Inductive Transfer

Here we show how the usage of meta features can be employed for *inductive transfer* ([10, 110, 16]). As explained in section 1.1.5 of the introduction, inductive transfer refers to the

problem of applying the knowledge learned in one or more tasks to the learning of a new task (the target task). Here, in the context of feature selection, we interested in better prediction of feature quality for a new task using its quality in other tasks. We assume that the different tasks use the same set of features. We also assume that during the stage we required to estimate feature quality we did not yet find any instances of the new task. A straightforward prediction is to use the average quality of the feature over the previous tasks. However we suggest using the quality in previous tasks of other features with similar properties, i.e. with similar values of meta-features. This make sense as usually not the exact same features are good for different tasks, but rather, good features of different tasks share similar properties. In practice this is done by first using the training tasks to learn a regression from meta-features to feature quality (similar to the procedure described in algorithm 5), and then using the regression to predict the quality of the features for the new task.

The notion of *related tasks* arises in many works on inductive transfer. Task is usually defined as related to the task of interest if using the knowledge gathered in this task improves performance on the current task. However, this notion is tricky as it may depend on a variety of parameters such as the choice of the learning algorithm ([16]). Here we show that the level of improvement depends highly on the kind of knowledge we choose to transfer. It is much better to transfer which *kind* of features are good rather than which specific features are good. This is achieved by using meta-features.

5.5.1 Demonstration on Handwritten Digit Recognition

To demonstrate the use of of meta-features for inductive transfer we consider the following collection of tasks: a task is defined by a pair of digits (e.g., 3 and 7), and the task is to classify images of these two digits. Thus we have $\begin{pmatrix} 10 \\ 2 \end{pmatrix} = 45$ different tasks to choose from. To avoid any possible overlap between the training tasks and the testing tasks, we first randomly divide the digits $0 \ldots 9$ into two disjoint sets of five digits. Then, we choose pairs of digits for the training tasks from the first set, and for the testing tasks from the second set. For each training task we calculate the quality of each feature (pixel) by infogain, and take

the average over all the training tasks. Then, we predict the quality of each feature for the test task in two different ways: (1) directly, as equal to the average quality on the training tasks. (2) indirectly, by learning a regression from the meta features to the quality. Finally, we check the correlation coefficient between the two different predicted qualities and the "real" quality, i.e., the infogain calculated using many examples of the test task. To evaluate the statistical significance of our results, we repeat this experiment for all possible choices of training tasks and a test tasks. We try to use two different kinds of meta features: exact (x, y) coordinates and only the distance from the center of the image, r. The results are presented in Figure 5.5. We can see that the indirect prediction that uses the meta-features achieves much better prediction, and that using r as a meta feature outperforms using (x, y) as meta-features. This also suggests that the relatedness of tasks depends on the kind of knowledge we choose to transfer and on the way we transfer it.

5.6 Choosing Good Meta-features

Choosing good meta-features for a given problem is obviously crucial. It is also worth inquiring why the problem of finding good meta-features is easier than the problem of finding good features. As we show in section 5.3.2, in the presence of a huge number of candidate features it is easier to guess which properties might be indicative of feature quality than to guess which exact features are good. In this section we give general guidelines on good choices of meta-features and mappings from meta-feature values to selection of features. First, note that if there is no correlation between meta-features and quality, the meta-features are useless. In addition, if any two features with the same value of meta-features are redundant (highly correlated), we gain almost nothing from using a large set of them. In general, there is a trade-off between two desired properties:

1. Features with the same value of meta-features have similar quality.
2. Low redundancy between features with the same value of meta-features.

When the number of features we select is small, we should not be overly concerned about redundancy and rather focus on choosing meta-features that are informative on quality. On

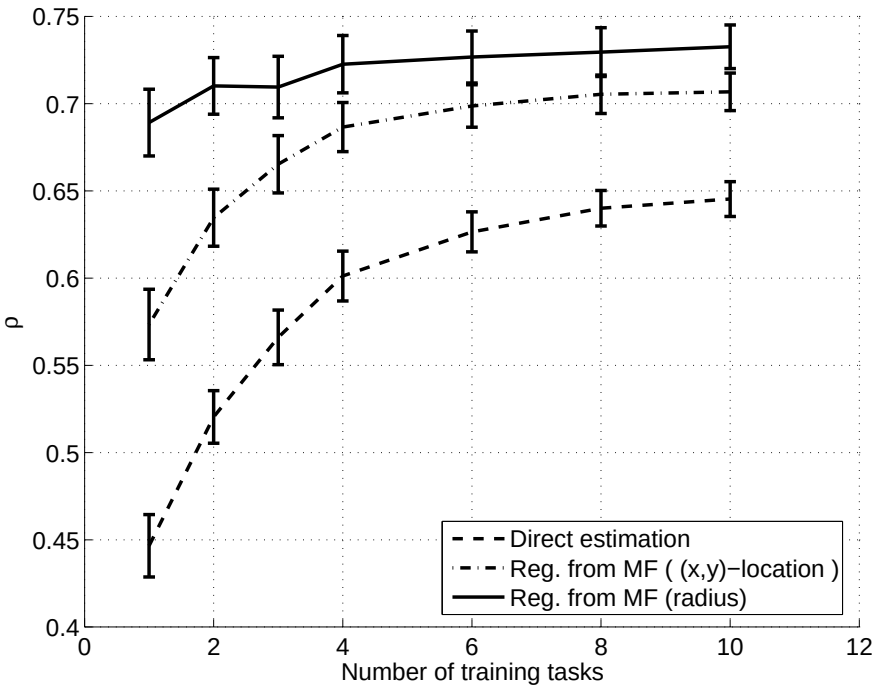

Figure 5.5: **Inductive Transfer by meta-features for OCR** . The correlation coefficient between the predicted and "real" quality of features as a function of number of training tasks. The results are averaged over all the possible choices of training tasks and test task repetitions. It is clear that indirect prediction (using meta-features) significantly outperforms the direct prediction.

the other hand, if we want to select many features, redundancy may be dominant, and this requires our attention. Redundancy can also be tackled by using distribution over meta-features instead of a single point.

In order to demonstrate the above trade-off we carried out one more experiment using the MNIST dataset. We used the same kind of features as in section 5.3.2, but this time without shift-invariance and with fixed scatter. The task was to discriminate between 9 and 4 and we used 200 images as the training set and another 200 as the test set. Then we used *Mufasa* to select features, where the meta-features were either the (x, y)-location or the number of inputs. When the meta-feature was (x, y)-location, the distribution of selecting the features, $p(v|\mathbf{u})$, was uniform in a 4×4 window around the chosen location (step 2a in *Mufasa*). Then we checked the classification error on the test set of a linear SVM (which uses the selected features). We repeated this experiment for different numbers of features[7]. The results are presented in Figure 5.6. When we use a small number of features, it is better to use the (x, y)-location as a meta-feature whereas when using many features it is better to use the number of inputs as a meta-feature. This supports our prediction about the redundancy-homogeneity trade-off. The (x, y)-locations of features are good indicators of their quality, but features from similar positions tend to be redundant. On the other hand, constraints on the number of inputs are less predictive of feature quality but do not cause redundancy.

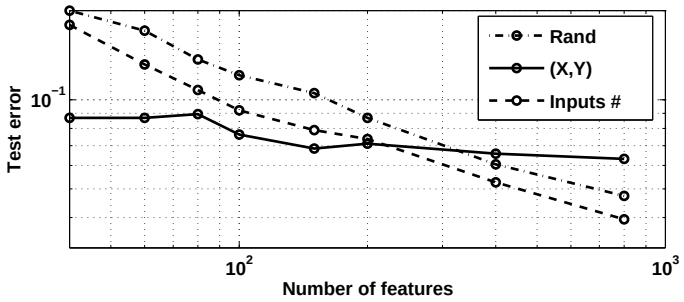

Figure 5.6: **Different choices of meta-features**. The generalization error as a function of the number of selected features. The two lines correspond to using different meta-features: (x, y)-location or number of inputs. The results of random selection of features are also presented.

[7] We do not use shift invariance here, thus all the features have the same cost.

5.7 Improving Selection of Training Features

The applications of meta-features presented so far involve predicting the quality of unseen features. However, the meta-features framework can also be used to improve the estimation of the quality of features that we do see in the training set. We suggest that instead of using direct quality estimation, we use some regression function on the meta-features space (as in algorithm 5). When we have only a few training instances, direct approximation of the feature quality is noisy, thus we expect that smoothing the direct measure by using a regression function of the meta-features may improve the approximation. One initial experiment we conducted in this way shows the potential of this method, but also raises some difficulties.

Once again we used 2000 images of the MNIST dataset with the 784 pixels as input features, and the (x, y) location as meta-features. We used algorithm 5 with Infogain and linear regression over RBF (see section 5.2 for details) to produce a prediction of the feature quality, and compared it to the direct measure of the Infogain in the following way: we calculated the Infogain of each feature using all 2000 instances as a reference quality measure, and compared its correlation coefficient with the two alternatives. The results are presented in Figure 5.7. It is clear that when the number of training instances is small enough (below 100) the indirect prediction that uses meta-features outperforms the direct measure in predicting the real Infogain. However, there was no significant improvement in the error rate of classification. We believe that this was due to the limitations of Infogain which ignores the redundancy between the features and the fact that in this choice of meta-features, features with similar values of meta-features are usually redundant. It might be possible to solve this problem by a different choice of meta-features. Another more fundamental problem is that if, by chance, a good feature is constant on the training set, our indirect method may predict correctly that it is good quality feature, but this feature is useless for classification using the current training set . The question of how to overcome this problem is still open.

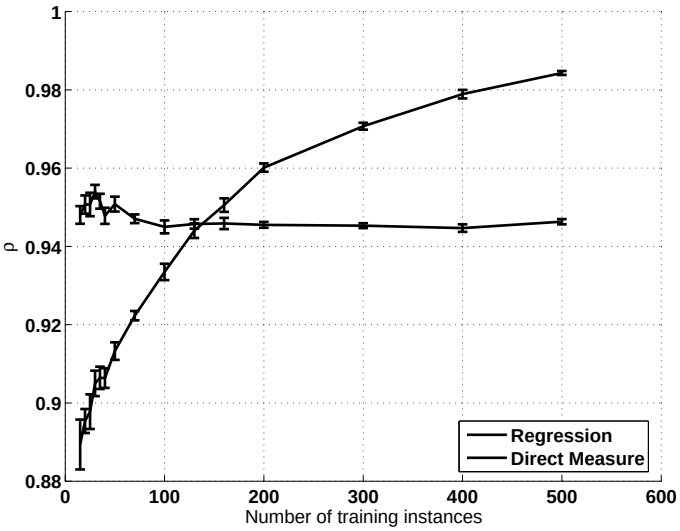

Figure 5.7: **Improving quality estimation from a few training instances for the OCR problem**. The correlation coefficient of direct Infogain and prediction using meta-features with the true Infogain, as a function of the number of training instances. It is clear that when there are less than 100 training instances, the indirect measure is better.

5.8 Summary

In this chapter we presented a novel approach to feature selection. Instead of just selecting a set of better features out of a given set, we suggest learning the properties of good features. We demonstrated how this technique may help in the task of feature extraction, or feature selection in the presence of a huge number of features and in the setting of inductive transfer. We showed that when the selection of features is based on meta-features it is possible to derive better generalization bounds on the combined problem of selection and classification. We illustrated our approach on a digit recognition problem, but we also expect our methods to be very useful in many other domains. For an example, in the problem of tissue classification according to a gene expression array where each gene is one feature, ontology-based properties may serve as meta-features. In most cases in this domain there are many genes and very few training instances; therefore standard feature selection methods tend to over-fit and thus yield meaningless results ([31]). A meta-feature based selection can help as it may reduce the complexity of the class of possible selections.

In section 5.3 we used meta-features to guide feature extraction. Our search for good

features is computationally efficient and has good generalization properties because we do not examine each individual feature. However, avoiding examination of individual features may also be considered as a disadvantage since we may include some useless individual features. This can be solved by using a meta-features guided search as a fast but rough filter for good features, and then applying more computationally demanding selection methods that examine each feature individually.

A Notation Table for Chapter 5

The following table summaries the notation and definitions introduced in chapter 5 for quick reference.

Notation	Short description	Sections
meta-feature	a property that describes a feature	
k	number of meta-features	
$\mathcal{S}$	(abstract) instances space	5.1, 5.4.1
f	a feature. formally, $f : \mathcal{S} \to \mathbb{R}$	
c	a classification rule	
S^m	S^m is a labeled set of instances (a training set)	5.1, 5.2, 5.4.1
$u_i(f)$	the the value of the$i$'s meta-feature on feature f	5.1, 5.3
$\mathbf{u}(f)$	$\mathbf{u}(f) = (u_1(f), \ldots, u_k(f))$, a vector that describes the feature f	5.1, 5.3, 5.4.1
$\mathbf{u}$	a point (vector) in the meta-feature space	5.1, 5.3
$\hat{Q}$	a mapping from meta-features value to feature quality	5.2, 5.4.2
Y_{MF}	measured quality of the features (e.g. Infogain)	5.2
X_{MF}	$X_{MF}(i,j) =$ the value of the j's meta-feature on the i's feature	5.2
F, F_j	a set of features	5.3
V	a random variable indicating which features are selected	5.3
$p(v\|\mathbf{u})$	the conditional distribution of V given meta-features values ($\mathbf{u}$)	5.3, 5.6
h_c	a classification hypothesis	5.4.1
$\mathcal{H}_c$	the classification hypothesis class	5.4.1
d_c	the VC-dimension of $\mathcal{H}_c$	5.4.2
h_{fs}	a feature selection hypothesis - says which n features are selected	5.4.1
$\mathcal{H}_{fs}$	the feature selection hypothesis class	5.4.1
h_s	a mapping from meta-feature space to $\{0,1\}$	5.4.2
$\mathcal{H}_s$	class of mappings from meta-feature space to $\{0,1\}$	5.4.2
d_s	the VC-dimension of $\mathcal{H}_s$	
J	number of iterations *Mufasa* (algorithm 6) makes	5.3, 5.4.1
$\|MF\|$	number of possible different values of meta-features	5.4.1

Chapter 6

Epilog

Learning is one of the most fascinating aspects of intelligence. The amazing learning ability of humans is probably the main thing that distinguishes humans from other animals. Machine learning tries to theoretically analyze learning processes and to mimic human-like learning abilities using computers. Many different methods have been suggested for learning and it is widely agreed that given a "good" representation of the data, many of these can give good results. This observation is consistent with popular saying: “asking the question in the right way is half way to the solution” . However, it is less clear how to build a good representation of the data, i.e., a representation which is both concise and reveals the relevant structure for the problem. One valid way to do so is first to measure (or produce) large number of candidate features which have a high probability of containing "good features" but also contain many weakly relevant or noisy features which mask the better features, and then use feature selection techniques to select only some of them. Thus, feature selection, the task of selecting a small number of features that enable efficient learning, deals with the fundamental question of finding good representations of data.

One thing to keep in mind is that feature selection is also a learning process in itself, and therefore is exposed to over-fit. Thus, care should always been taken in analyzing the results, especially when the number of candidate features is huge and we only have a small number of training instances. In this case the noise of the measurement of feature quality may be

too high and any selection algorithm that uses only the data will fail. Such a situation is common in the field of gene expression, as described by Ein-dor et al. [31]. Therefore for these cases we need to adopt a different approach. One possible way is to use properties of the features, as we suggest in chapter 5. Another option is to use information extracted in other related tasks, as done in Inductive Transfer.

We believe that the selection algorithms we suggested in this thesis can work well on many problems, but it is important to understand that any selection algorithm is based on some assumptions. If these assumptions are violated the algorithm can fail. On the other hand, if a stronger assumption holds, another algorithm that assumes this stronger assumption might outperform the first one. For example, a method that ranks individual features by assigning a score to each feature independently assumes that complex dependency on sets of features does not exist or is negligible. This assumption narrows the selection hypothesis space, and therefore allows for generalization using fewer instances. Thus, if this assumption is true, we would expect such a ranking to work better than methods that do not assume this, i.e. methods that consider subsets and are able to reveal complex dependencies (as these methods look in a larger hypothesis space). However, we cannot expect such rankers to work well when this "independency" assumption is not true. To demonstration this, the performance of SKS and RGS in figure 4.1 in chapter 4 provide a good example. SKS and RGS use the same evaluation function, but SKS considers each feature independently whereas RGS considers all the features at the same time. We can see that SKS outperforms RGS on problem (a), where there are no complex dependencies, but completely fails on problem (d), where complex dependencies exist.

Another example can be seen in chapter 3. Simba and G-flip are based on margins, and therefore implicitly assume that the Euclidean distance is meaningful for the given data. In the data we used for our experiments this assumption was true and they outperformed all the alternatives. However on other kinds of data where this assumption is violated, Infogain, which does not use this assumption, might outperform Simba and G-flip. Thus, there is no one selection algorithm which is the ideal for all problems. When data of a new kind emerge, we need to try many selection algorithms to find which is most suitable. Of course, we can also use some prior knowledge on the data to predict which selection algorithm is most

adequate, if such prior knowledge exists. By analyzing the success of the different kinds of algorithms we can also discover some important properties of the data. An example of such an analysis is given in the chapter 4. In section 4.4 of this chapter we use the fact that RGS works better than SKS on the motor control task to suggest that complex dependencies upon sets of neurons or time bins are an important property of the neural code.

List of Publications

In order to keep this document reasonably sized, only a subset of the work I have done during my studies is presented in this dissertation. Here is a complete list of my publications.

Journal Papers

- R. Gilad-Bachrach, A. Navot, and N. Tishby, *A Study of The Information Bottleneck Method and its Relationship to Classical Problems.* Under review for publication in IEEE transaction on information theory.

- E. Krupka, A. Navot and N. Tishby, *Learning to Select Features using their Properties.* Submitted to Journal of Machine Learning Research (JMLR).

Chapters in Refereed Books

- A. Navot, R. Gilad-Bachrach, Y. Navot and N. Tishby. Is Feature Selection Still Necessary? In C. Saunders, M. Grobelnik, S. Gunn and J. Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection.* Springer-Verlag, 2006.

- R. Gilad-Bachrach, A. Navot and N. Tishby Large margin principles for feature selection. In *Feature extraction, foundations and applications,* I. Guyon, S. Gunn, M. Nikravesh and L. Zadeh (eds.) , Springer-Verlag 2006.

- R. Gilad-Bachrach, A. Navot and N. Tishby Connections with some classic IT problems. In *Information Bottlenecks and Distortions: The emergence or relevant structure from data,* N. Tishby and T. Gideon (eds.) MIT press (in preparation).

Refereed Conferences

- R. Gilad-Bachrach, A. Navot and N. Tishby, *Query By Committee made real*, in Proceedings of the 19$^{\text{th}}$ Conference on Neural Information Processing Systems (NIPS), 2005.
- R. Gilad-Bachrach, A. Navot and N. Tishby, *Bayes and Tukey meet at the center point*, in Proceedings of the 17$^{\text{th}}$ Conference on Learning Theory (COLT), 2004.
- R. Gilad-Bachrach, A. Navot and N.Tishby, *Margin based feature selection - theory and algorithms*, in Proceedings of the 21$^{\text{st}}$ International Conference on Machine Learning (ICML), 2004.
- R. Gilad-Bachrach, A. Navot, and N. Tishby, *An information theoretic tradeoff between complexity and accuracy*, in Proceedings of the 16$^{\text{th}}$ Conference on Learning Theory (COLT), pp. 595-609, 2003.
- K. Crammer, R. Gilad-Bachrach, A. Navot, and N. Tishby, *Margin analysis of the lvq algorithm*, in Proceedings of the 16$^{\text{th}}$ Conference on Neural Information Processing Systems (NIPS), 2002.
- A. Navot, L. Shpigelman, N. Tishby and E. Vaadia, *Nearest Neighbor Based Feature Selection for Regression and its Application to Neural Activity.* Proc. 20th Conference on Neural Information Processing Systems, 2006

Technical Reports

- R. Gilad-Bachrach, A. Navot, and N. Tishby, *Kernel query by committee (KQBC)*, technical report 2003-88, Leibniz Center, the Hebrew University, 2003.

Bibliography

[1] D. Aha and R. Bankert. Feature selection for case-based classification of cloud types: An empirical comparison, 1994.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] R. F. Ahlswede and J. Korner. Source coding with side information and a converse for degraded broadcast channels. *IEEE transaction on information theory*, 21(6):629–637, November 1975.

[4] H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Inte lligence (AAAI-91)*, volume 2, pages 547–552, Anaheim, California, 1991. AAAI Press.

[5] T. W. Anderson. Classification by multivariate analysis. *Psychometria*, 16:31–50, 1951.

[6] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[7] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *AI Review*, 11.

[8] Jerzy Bala, J. Huang, Haleh Vafaie, Kenneth DeJong, and Harry Wechsler. Hybrid learning using genetic algorithms and decision trees for pattern classification. In *IJCAI (1)*, pages 719–724, 1995.

[9] P. Bartlett. The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[10] Jonathan Baxter. A bayesian/information theoretic model of learning to learn viamultiple task sampling. *Mach. Learn.*, 28(1):7–39, 1997.

[11] A. Bell and T. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7:1129–1159, 1995.

[12] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[13] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, 100:157–184, 1989.

[14] B. Boser, I. Guyon, and V. Vapnik. Optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

[15] C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proceedings of the* 17^{th} *International Conference on Machine Learning (ICML)*, 2000.

[16] Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, 1997.

[17] Rich Caruana and Dayne Freitag. Greedy attribute selection. In *International Conference on Machine Learning*, pages 28–36, 1994.

[18] G. C. Cawley. MATLAB support vector machine toolbox (v0.55β) [`http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox`]. University of East Anglia, School of Information Systems, Norwich, Norfolk, U.K. NR4 7TJ, 2000.

[19] Kevin J. Cherkauer and Jude W. Shavlik. Growing simpler decision trees to facilitate knowledge discovery. In *Knowledge Discovery and Data Mining*, pages 315–318, 1996.

[20] Shay Cohen, Eytan Ruppin, and Gideon Dror. Feature selection based on the shapley value. In *IJCAI*, pages 665–670, 2005.

[21] D. Cohn, L. Atlas, and R. Ladner. Training connectionist networks with queries and selective sampling. *Advanced in Neural Information Processing Systems 2*, 1990.

[22] T.M. Cover and P.E. Hart. Nearest neighbor pattern classifier. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[23] K. Crammer. Mcsvm_1.0: C code for multiclass svm, 2003. http://www.cis.upenn.edu/∼crammer.

[24] K. Crammer, R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin analysis of the lvq algorithm. In *Proc. 17'th Conference on Neural Information Processing Systems (NIPS)*, 2002.

[25] Ofer Dekel and Yoram Singer. Data-driven online to batch conversions. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

[26] A.P. Dempster, N.M. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, B, 39:1–38, 1977.

[27] L. Devroye. The uniform convergence of nearest neighbor regression function estimators and their application in optimization. *IEEE transactions in information theory*, 24(2), 1978.

[28] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

[29] R.O. Duda, P.E. Hart, and Stork D.G. *Pattern Classification 2'nd edition*. Wiley-Interscience Publication, 2000.

[30] L. Kasatkina E. Kussul, T. Baidyk and V. Lukovich. Rosenblatt perceptrons for handwritten digit recognition. In *Int'l Joint Conference on Neural Networks*, pages 1516–20, 2001.

[31] L. Ein-Dor, O. Zuk, and E. Domany. Thousands of samples are needed to generate a robust gene list for predicting outcome in cancer. *Proc Natl Acad Sci U S A*, 103(15):5923–5928, April 2006.

[32] E. Fix and j. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. Technical Report 4, USAF school of Aviation Medicine, 1951.

[33] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[34] Y. Freund, H. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.

[35] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Trans. of Computers*, 23(9):881–890, 1974.

[36] Jerome H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, 1987.

[37] J. H. Gennari, P. Langley, and D. Fisher. Models of incfremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.

[38] R. Gilad-Bachrach. *To PAC and Beyond*. PhD thesis, Hebrew University, 2006.

[39] R. Gilad-Bachrach, A. Navot, and N. Tishby. An information theoretic tradeoff between complexity and accuracy. In *Proc. 16'th Conference on Computational Theory (COLT)*, pages 595–609, 2003.

[40] R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin based feature selection - theory and algorithms. In *Proc.* 21^{st} *(ICML)*, pages 337–344, 2004.

[41] R. Gilad-Bachrach, A. Navot, and N. Tishby. Large margin principles for feature selection. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature extraction, foundations and applications*. Springer, 2006.

[42] R. Greiner. Using value of information to learn and classify under hard budgets. NIPS 2005 Workshop on Value of Information in Inference, Learning and Decision-Making, 2005.

[43] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learnig Research*, pages 1157–1182, Mar 2003.

[44] I. Guyon and S. Gunn. Nips feature selection challenge. `http://www.nipsfsc.ecs.soton.ac.uk/`, 2003.

[45] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh. *Feature extraction, foundations and applications*. Springer, 2006.

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

[47] P. M. Hofman, J. G. A. Van Riswick, and A. Van Opstal. Relearning sound localization with new ears. *Nature Neuroscience*, 1:417–421, 1998.

[48] j. Holland. *Adaption in Neural and Artificial Systems*. University of Michigen Press, 1975.

[49] J. Hua, Z. Xiong, and E. R. Dougherty. Determination of the optimal number of features for quadratic discriminant analysis via normal approximation to the discriminant distribution. *Pattern Recognition*, 38(3):403–421, March 2005.

[50] P.J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[51] J.P. Ignizio. *Introduction to Expert Systems*. McGraw-Hill, Inc., USA, 1991.

[52] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 604–613, 1998.

[53] A. K. Jain and W. G. Waller. On the monotonicity of the performance of bayesian classifiers. *IEEE transactions on Information Theory*, 24(3):392–394, May 1978.

[54] A. K. Jain and W. G. Waller. On the optimal number of features in the classification of multivariate gaussian data. *Pattern Recognition*, 10:365–374, 1978.

[55] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *International Conference on Machine Learning*, pages 121–129, 1994. Journal version in AIJ, available at http://citeseer.nj.nec.com/13663.html.

[56] I.T. Jolliffee. *Principal Component Analysis*. Springer Varlag, 1986.

[57] M.C. Jones and R. Sibson. What is projection pursuit ? *J. of the Royal Statistical Society, ser. A*, 150:1–36, 1987.

[58] M. W. Kadous and C. Sammut. Classification of multivariate time series and structured data using constructive induction. *Mach. Learn.*, 58:179–216, 2005.

[59] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[60] Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994.

[61] K. Kira and L. Rendell. A practical approach to feature selection. In *Proc. 9th International Workshop on Machine Learning*, pages 249–256, 1992.

[62] R. Kohavi and G.H. John. Wrapper for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[63] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, 1995.

[64] Daphne Koller and Mehran Sahami. Toward optimal feature selection. In *International Conference on Machine Learning*, pages 284–292, 1996.

[65] I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *Proc. ECML*, pages 171–182, 1994.

[66] E. Krupka and N. Tishby. Generalization in clustering with unobserved features. In *NIPS*, 2006.

[67] J. B. Kruskal. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition edition, 2002.

[68] K. J. Lang and E. B. Baum. Query learning can work poorly when a human oracle is used. In *Proceedings of the International Joint Conference on Neural Networks*, pages 335–340, 1992.

[69] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[70] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[71] K. Levi, M. Fink, and Y. Weiss. Learning from a small number of training examples by exploiting object categories. LCVPR04 workshop on Learning in Computer Vision, 2004.

[72] N. Littlestone. *Mistake Bounds and Logarithmic Linear-threshold Learning Algorithms*. PhD thesis, University of California Santa Cruz, 1989.

[73] H. Liu and R. Sutiono. Feature selection and discretization. *IEEE Trans Knowledge and Data Eng*, 9.

[74] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Min. Knowl. Discov.*, 6(4):393–423, 2002.

[75] D. Lizotte, O. Madani, and R. Greiner. Budgeted learning of naive-bayes classifiers. In *UAI*, 2003.

[76] George F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[77] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, pages 281–297. University of California Press, 1967.

[78] T. Marill and D Green. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9:11–17, 1963.

[79] O. Maron and A. Moore. The racing algorithm: Model selection for lazy learners. In *Artificial Intelligence Review*, volume 11, pages 193–225, April 1997.

[80] A.M. Martinez and R. Benavente. The ar face database. Technical report, CVC Tech. Rep. #24, 1998.

[81] L. Mason, P. Bartlett, and J. Baxter. Direct optimization of margins improves generalization in combined classifier. *Advances in Neural Information Processing Systems*, 11:288–294, 1999.

[82] A.J. Miller. *Subset Selection in Regression*. Chapman and Hall, 1990.

[83] Kensaku Mori and Hiroshi Nagao andYoshihiro Yoshihara. The olfactory bulb: Coding and processing of odor molecule information. *Science*, 286(5440):711–715, 1999.

[84] A. Navot, R. Gilad-Bachrach, Y. Navot, and N. Tishby. Is feature selection still necessary? In C. Saunders, M. Grobelnik, S. Gunn, and J. Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection*. Springer-Verlag, 2006.

[85] A. Navot, L. Shpigelman, N. Tishby, and E. Vaadia. Nearest neighbor based feature selection for regression and its application to neural activity. In *Proc.* 20^{th} *Conference on Neural Information Processing Systems (NIPS)*, forthcoming 2006.

[86] A. Ng. Feature selection, l_1 vs l_2 regularization and rotational invariance. In *Proc.* 21^{st} *International Conference on Machine Learning (ICML)*, pages 615–622, 2004.

[87] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[88] R. Paz, T. Boraud, C. Natan, H. Bergman, and E. Vaadia. Preparatory activity in motor cortex reflects learning of local visuomotor skills. *Nature Neuroscience*, 6(8):882–890, August 2003.

[89] J. Pearl. *Probablistic Reasonong in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA, 1988.

[90] A. Perera, T. Yamanaka, Gutierrez-Galvez A., B. Raman, and R. Gutierrez-Osuna. A dimensionality-reduction technique inspired by receptor convergence in the olfactory system. *Sensors and Actuators B*, 116(1-2):17–22, 2006.

[91] Bernhard Pfahringer. Compression-based feature subset selection, 1995.

[92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, 1988.

[93] J. R. Quinlan. Induction of decision trees. *Journal of Machine Learning*, 1:81–106, 1986.

[94] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[95] R. Raina, A.Y. Ng, and D. Koller. Constructing informative priors using transfer learning. In *Proc. Twenty-Third International Conference on Machine Learning*, 2006.

[96] S. Raudys and V. Pikelis. On dimensionality, sample size, classification error and complexity of classification algorithm in pattern recognition. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-2(3):242–252, May 1980.

[97] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[98] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[99] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 12 2000.

[100] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. Boosting the margin : A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 1998.

[101] Thomas Serre, Maximilian Riesenhuber, Jennifer Louie, and Tomaso Poggio. On the role of object-specific features for real world object recognition in biological vision. In *BMCV '02: Proceedings of the Second International Workshop on Biologically Motivated Computer Vision*, pages 387–397, London, UK, 2002. Springer-Verlag.

[102] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, July and October 1948.

[103] J. Shawe-Taylor, P.L. Bartlett, R.C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE transactions on Information Theory*, 44(5):1926–1940, 1998.

[104] P. Simard, Y. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*, pages 239–27, 1996.

[105] P. Y. Simard, Y. A. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 50–58. Morgan Kaufmann, 1993.

[106] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *International Conference on Machine Learning*, pages 293–301, 1994.

[107] G. W. Snedecorand and W. G. Cochran. *Statistical Methods*. Iowa State University Press, Berlin, Heidelberg, New York, 1989.

[108] B. Taskar, M. F. Wong, and D. Koller. Learning on the test data: Leveraging unseen features. In *ICML*, 2003.

[109] D. M. Taylor, S. I. Tillery, and A. B. Schwartz. Direct cortical control of 3d neuroprosthetic devices. *Science*, 296(7):1829–1832, 2002.

[110] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Proc. 15th Conference on Neural Information Processing Systems (NIPS) 8*, pages 640–646. MIT Press, 1996.

[111] N. Tishby, F.C. Pereira, and W. Bialek. The information bottleneck method. In *Proc. 37th Annual Allerton Conf. on Communication, Control and Computing*, pages 368–377, 1999.

[112] Godfried Toussaint. Proximity graphs for nearest neighbor decision rules: Recent progress.

[113] G. V. Trunk. A problem of dimensionality: a simple example. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-1(3):306–307, July 1979.

[114] S. Ullman, M. Vidal-Naquet, and E. Sali. Visual features of intermediate complexity and their use in classification. *Nature Neuroscience*, 5(7):682–687, 2002.

[115] H. Vafaie and K. DeJong. Robust feature selection algorithms. *Proc. 5th Intl. Conf. on Tools with Artifical Intelligence*, pages 356–363, 1993.

[116] V. Vapnik. *The Nature Of Statistical Learning Theory*. Springer-Verlag, 1995.

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

[118] V. Vapnik and A. Y. Chervonenkis. On the uniform covergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[119] J. Weston, A. Elisseeff, G. BakIr, and F. Sinz. The spider, 2004. http://www.kyb.tuebingen.mpg.de/bs/people/spider/.

[120] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In *Proc. 15th NIPS*, pages 668–674, 2000.

[121] A. D. Wyner. On source coding with side information at the decoder. *IEEE transaction on information theory*, 21(3):294–300, May 1975.

על בחירת תכוניות בלמידה חישובית

חיבור לשם קבלת תואר דוקטור לפילוסופיה
מאת

אמיר נבות

הוגש לסנט האוניברסיטה העברית בירושלים בשנת תשס'ו

עבודה זו נעשתה בהנחייתו של פרופסור נפתלי תישבי

תקציר

תזה זו דנה באספקטים שונים של של בחירת תכוניות בלמידה חישובית, ובפרט, בבחירת תכוניות עבור למידה מונחה. לכן בתקציר זה נתחיל בהסבר קצר על מה היא למידה חישובית ובמבוא קצר לתחום בחירת התכונית. בהמשך נציג בקצרה את התוצאות החדשות המרכזיות המוצגות בתזה.

למידה חישובית

למידה חישובית עוסקת בהיבטים תיאורטיים, אלגוריתמיים ויישומיים של למידה והסקת כללים מדוגמאות. על קצה המזלג, באומרנו למידה מדוגמאות הכוונה היא שאנו מנסים לבנות מכונה (תוכנת מחשב בד"כ) המסוגלת ללמוד לבצע משימה ע"י צפייה בדוגמאות. בדרך כלל הלומד (כלומר המכונה) משתמש בקבוצת אימון של דוגמאות בכדי לבנות מודל של העולם המאפשר לתת תח-זיות אמינות על העולם. זה בניגוד לתוכנה המסוגלת לתת תחזיות ע"י שימוש באוסף חוקים שהו-גדרו מראש (הגישה הקלאסית של בינה מלאכותית). לכן בלמידה חישובית המכונה צריכה ללמוד "מנסיונה" היא, ובמובן זה הגישה של למידה חישובית דבקה באמירה הישנה של חז"ל:

אין חכם כבעל ניסיון

ישנם מספר מודלי למידה שונים הנחקרים תחת התחום של למידה חישובית, אנו כאמור מתמ-קדים בעבודה זו בלמידה מונחה. בלמידה מונחה הלומד מקבל כקבוצת אימון מדגם מתוייג בו כל דוגמא היא זוג מהצורה (עצם, תווית). מטרתו של הלומד היא להצליח לחזות במדויק ככל האפשר את התווית של עצם חדש אותו הוא לא ראה קודם. באופן כללי העצם יכול להיות ישות אבסטרקטית כגון אדם, תמונה או טקסט, אך בד"כ בלמידה חישובית מניחים שהעצם מיוצג ע"י וקטור של מספר-ים ממשיים. כל קואורדינאטה בווקטור זה נקראת תכונית (feature), ולכן העצם מיוצג ע"י וקטור של תכוניות. התווית יכולה להיות קטגורית, ואז נאמר שיש לנו בעיית תיוג (classification) או רציפה, ואז נאמר שיש לנו בעיית רגרסיה (regression). בחיבור זה נתמקד רוב הזמן בבעיות תיוג (מלבד

בפרק 4). ההנחה בד"כ היא שהתיוג נעשה על פי כלל קבוע (הנקרא מושג המטרה) שאינו ידוע לנו ומטרתנו היא להשתמש בקבוצת האימון בכדי למצוא כלל תיוג (הנקרא השערה) שיקרב כלל זה היטב.

המידה המקובלת לאיכותו של לומד היא יכולת ההכללה של הכלל הנלמד, כלומר כמה טוב הוא תוזה את ערכו של מושג המטרה על עצם חדש שהלומד לא ראה בזמן הלימוד. אנו מתמקדים בלמידה אינדוקטיבית ובפרט במודל Probably Approximately Correct [31, 811] בו מניחים שדוגמאות האימון נדגמו באופן בלתי תלוי מתוך התפלגות קבועה (iid). כאן יכולת ההכללה של כלל תיוג נמדדת ע"י שגיאת ההכללה שלו שהיא הסיכוי שהוא יטעה על דוגמא חדשה שנדגמה באופן בלתי תלוי מאותה התפלגות. עם זאת בד"כ לא נוכל למדוד ישירות גודל זה ולכן נסתכל בשגיאת האימון המוגדרת כחלקן היחסי של דוגמאות האימון עליהן הכלל הנלמד טועה מתוך כלל דוגמאות האימון.

בצד התיאורטי למידה חישובית מנסה לאפיין מה למיד (ניתן ללמידה), כלומר למצוא תחת אילו תנאים שגיאת אימון קטנה מבטיחה שגיאת הכללה קטנה. אפיון כזה עשוי לתת תובנות על המגבלות להן כל מערכת לומדת צריכה לציית. מובן שאם לא נטיל שום מגבלות, כלומר ההשערה שלנו יכולה להיות כל דבר, מסובך ככל שיהיה, לא ניתן לחסום את הפער בין שגיאת האימון לשגיאת ההכללה ולכן חייבים להניח הנחות מגבילות כלשהן. ההנחה המקובלת היא שההשערה שלנו נבחרה מעולם השערות (מחלקת השערות) נתון ו"פשוט יחסית". במקרה זה ניתן לחסום את הפער כפונקציה של סיבוכיות מחלקת ההשערות ושל מספר דוגמאות האימון, כך שככל שמחלקת ההשערות פשוטה יותר, נצטרך פחות דוגמאות בכדי להבטיח אותו חסם על הפער. תוצאה זו היא הדגמה של העיקרון של Occam's Razor :

entia non sunt multiplicanda praeter necessitatem

שמיתרגם ל: ישויות לא צריכות להיות משוכפלות מעבר לנדרש

ולעיתים קרובות מנוסח מחדש כ: ההסבר הפשוט ביותר הוא הטוב ביותר

בצד האלגוריתמי למידה חישובית מנסה למצוא אלגוריתמים המסוגלים למצוא השערה טובה (המקרבת טוב את מושג המטרה) בהינתן קבוצת האימון. באופן אידיאלי נעדיף אלגוריתמים עבורם ניתן להוכיח חסמים על זמן הריצה ועל דיוק התוצאה. עם זאת גם אלגוריתמים יורסטיים ש"פשוט עובדים" נפוצים למדי בתחום ולעיתים בפועל הם עובדים טוב יותר מהחלופה עבורה ניתן להוכיח תכונות שונות. בצד היישומי למידה חישובית מנסה להתאים אלגוריתמים למשימות ספציפיות. אלגוריתמי למידה משמשים למשימות רבות הן באקדמיה והן בתעשייה. כמה דוגמאות לישומים הן: זיהוי פרצופים, מיון טקסטים, אבחנה רפואית, זיהוי תרמיות בכרטיסי אשראי.

בעיית הייצוג

שאלה מרכזית בלמידה חישובית היא איך לייצג את העצמים. ברוב מודלי הלמידה מניחים שעצמים נתונים כוקטורים ב $\mathbb{R}^N$ (כאשר N הוא מימד סופי כלשהו), והניתוח של תהליך הלמידה מתחיל מנקודה זו. ישנה הסכמה רחבה על כך שבהינתן ייצוג טוב, רוב אלגוריתמי הלמידה הסבירים ישיגו ביצועים סבירים לאחר כוונון מתאים. מצד שני אם נבחר ייצוג גרוע אין תקווה להשיג ביצועים טובים. לכן נשאלת השאלה איך בונים ייצוג טוב של העצמים (למשל תמונות) על ידי וקטור של מספרים? ייצוג טוב צריך להיות תמציתי מחד ובעל משמעות לבעיה המונחת לפנינו מאידך. בחירת ייצוג משמועתה לבחור אוסף של תכוניות שימדדו עבור כל עצם. בבעיות אמיתיות, אוסף תכוניות זה בד"כ נבחר על ידי מומחה בתחום הרלוונטי באופן ידני. אך נשאלת השאלה האם ניתן למצוא ייצוג טוב באופן אוטומטי תוך שימוש בקבוצת האימון.

אחת הדרכים לתקוף בעיה זו היא בעזרת בחירת תכוניות. במקרה זה נתחיל בלמדוד (או לייצר) הרבה מאוד תכוניות, כך שסביר להניח שביניהן מתחבאות גם תכוניות טובות, ואז נשתמש באלגוריתם לבחירת תכוניות בכדי למצוא תת קבוצה קטנה המאפשרת ביצועים טובים, כלומר שתהווה ייצוג טוב. היתרונות העיקריים של שימוש בבחירת תכוניות על פני שימוש באלגוריתמי הורדת מימד כללים הם יכולת פענוח נוחה יותר של התוצאה וחיסכון (בכך שנכסך הצורך למדוד בעתיד תכוניות שלא נבחרו).

בחירת תכונית

בלמידה מונחה, בהרבה מהמקרים, העצמים מיוצגים על ידי מספרת גדול מאוד של תכוניות, כאשר חלק גדול מהן לא נחוצות לצורך חיזוי של התגיות, ואף פוגעים ביכולת ליצר חיזוי מדויק. בחירת תכוניות היא המשימה של בחירת מספר (בד"כ קטן) של תכוניות מתוך כלל התכוניות המאפשר חיזוי טוב של תגיות המטרה. ארבע המניעים העיקריים לביצוע בחירת תכוניות הם:

1. **שיפור הסיבוכיות החישובית.** אלגוריתמי למידה רבים פשוט לא ניתנים להרצה על נתונים המיוצגים על יד מספר עצום של תכוניות, בגלל הסיבוכיות החישובית הגבוהה שלהם. שלב מקדים של בחירת תכוניות יכול לפתור בעיה זו.

2. **חסכון כלכלי.** בחירת תכוניות מונעת את הצורך למדוד בעתיד (בשלב החיזוי) את כל התכוניות שלא נבחרו ובכך נחסכת עלות מדידתם. חסכון זה יכול להיות עצום ובמקרים מסוימים יכול להפוך את כל העניין מבלתי ישים לישים.

3. **שיפור הדיוק.** בהרבה מקרים בחירת תכוניות יכולה גם לשפר את יכולת החיזוי של הכלל הנלמד על ידי שיפור יחס האות לרעש. גם האלגוריתמי הלמידה הטובים ביותר הידועים לנו כיום לא יכולים להתגבר על נוכחותם של מספר עצום של תכוניות לא רלוונטיות או כמעט לא רלוונטיות. לכן, במקרים אלו צעד מקדים של בחירת תכוניות יכול לשפר את הדיוק באופן דרמטי.

4. **הבנת הבעיה.** זהות התכוניות שנבחרו יכולה לעזור בהבנת הבעיה שאנו מנסים לפתור. בהרבה מקרים היכולת להצביע על התכוניות החשובות חשובה יותר מדיוק התיוג עצמו. למשל נחשוב על אבחון רפואי (חולה/לא חולה) על פי רמות ביטוי של גנים שונים. בעוד שישנן עוד דרכים לדעת אם אדם חולה או לא, זהות הגנים שחשובים לחיזוי יכולה לעזור בהבנת מנגנון המחלה ובפיתוח תרופות.

לכן בחירת תכוניות היא מרכיב חשוב בלמידה יעילה של נתונים מרובי תכוניות. לגבי מניעים 2 ו 4 לעיל, לבחירת תכוניות יש יתרון על פני שיטות הורדת מימד כלליות.

בהרבה מחקרים על בחירת תכוניות מגדירים את המשימה כזיהוי אלו תכוניות רלוונטיות ואילו אינן רלוונטיות. הגדרה זו מעלה את הבעיה של הגדרת רלוונטיות. מכיוון שהתרומה של תכונית תלויה באילו תכוניות נוספת אנו משתמשים, כלל לא פשוט להגדיר מושג זה, והגדרות שונות ניתנו על ידי חוקרים שונים. אך לפי כל הגדרות אלו רלוונטיות לא תמיד גוררת חברות בקבוצת התכונות האופטימלית ולהיפך. לכן אנו מעדיפים להגדיר את משימת בחירת התכוניות כ "בחירת תת-קבוצה קטנה של תכוניות המאפשרת חיזוי טוב של תגיות המטרה". עם זאת, כשהמטרה היא הבנת הבעיה (מניע 4 לעיל), ידיעת מידת הרלוונטיות של כל תכונית היא חשובה. אך בבעיות אמיתיות כמעט אף פעם תכונית אינה לחלוטין לא רלוונטית, ולכן עדיף לדבר על מידת התרומה של כל תכונית. כהן וחבריו [20] הציעו את ערך שפלי (Shapley) של תכונית כמדד לחשיבותה. ערך שפלי הוא גודל הלקוח מתורת המשחקים, שם הוא משמש למדידת תרומו של שחקן במשחק קבוצתי.

אלגוריתמים רבים הוצעו במשך השנים לבחירת תכוניות גם בתחום הלמידה החישובית וגם במחקר בתחום הסטטיסטיקה. חלק מהאלגוריתמים שונים מאוד ביניהם, אך רבים מהם חולקים את המבנה הכללי שנתאר להלן. במבנה זה ישנה פונקצית הערכה הנותנת ציון לקבוצות של תכוניות וישנו אלגוריתם חיפוש המשמש לחיפוש קבוצת תכוניות עם ציון גבוה. ראה איור 1.1 להמחשה של מבנה זה. פונקצית ההערכה יכולה להיות מבוססת על אלגוריתם למידה מסוים (מודל המעטפת [26]), או על איזושהי מידה כללית לחיזוי תרומתה של כל תכונית לחיזוי (מודל הסינון [26]). ברוב האלגוריתמים העוטפים, איכותה של קבוצת תכוניות משוערכת על ידי בדיקת הכלל הנלמד על

קבוצת הערכה (validation set). היתרון העיקרי של שיטות מסוג זה הוא בכך שהן פועלות על הגודל שבאמת מעניין אותנו - איכות החיזוי של האלגוריתם בו נשתמש ללמידה. החיסרון הוא בזמן ריצה ארוך, מכיוון שבכל צעד יש להריץ מחדש את אלגוריתם הלמידה. רבים מאלגוריתמי הסינון משתמשים בגדלים כגון שונות מותנה (של התכונית בהינתן התיוג), מקדם קורלציה או אינפורמציה משותפת כמדדים כללים לאיכות.

בכל מקרה, חיפוש ממצא על פני כל תתי הקבוצות העיקריות של תכוניות אינו אפשרי ולכן משתמשים בשיטות חיפוש, המשתמשות ביוריסטיקות שונות. נזכיר כרגע רק שתי שיטות קלאסיות. בחירה קדימה (forward selection) והסרה לאחור (backward elimination). בבחירה קדימה מתחילים עם קבוצת תכוניות ריקה ובכל צעד מוסיפים תכונית אחת באופן חמדני. בכל צעד מוסיפים את התכונית המביאה לגידול מרבי בערך פונקצית ההערכה. בהסרה לאחור מתחילם בקבוצת כל התכוניות ובכל צעד מסירים את התכונית המביאה לגידול מרבי (או להקטנה מזערית) בערכה של פונקצית ההערכה. חסרונה העיקרי של גישה זו הוא זמן ריצה ארוך כאשר מספר התכוניות גדול. שילוב בין השתיים אפשרי גם כן כמובן. כפי שכבר הזכרנו, אלגוריתמים רבים הוצעו לצורך בחירת תכוניות השונים ביניהם הן בבחירה של פונקצית ההערכה והן באופן החיפוש. חלקם פשוט בודקים את איכותה של כל תכונית לבדה ובוחרים את התכוניות בעלות האיכות האישית הגבוהה ביותר וחלקם מתייחסים גם לקבוצות של תכוניות. אחרים בכלל חורגים מהפרדיגמה שהוצגה לעיל ופוע-לים בגישה אחרת, למשל השתלבות אינטגרלית בתהליך הלימוד. קצרה היריעה מלסקור בתקציר זה אפילו מדגם מבין האלגוריתמים הרבים שהוצעו בעבר, אך חשוב להבין שכל אלגוריתם מתבסס על הנחות מסויימות. אם הנחות אלו לא מתקיימות, האלגוריתם עלול להיכשל. לעומת זאת עם הנחות חזקות יותר מתייקמות, אלגוריתם אחר המניח אותם צפוי לעבוד טוב יותר, כלומר להזדקק לפחות דוגמאות בכדי להגיע לאותם ביצועים, זאת מכיוון שהוא מחפש בתוך מרחב השערות קטן יותר. לכן לא ניתן לצפות לקיום אלגוריתם בחירה אחד שהוא תמיד "הטוב ביותר". לבעיות שונות אלגוריתמים שונים יפעלו טוב יותר ואין מנוס מניסוי וטעיה. כאן גם המקום להזכיר כי בחירת תכוניות היא תהליך למידה בפני עצמו ולכן חשוף להתאמת יתר (overfitting), בעיקר כאשר ישנן תכוניות רבות מאוד אך רק מעט דוגמאות אימון.

תקציר התוצאות

בפרק 2 אנו דנים בנחיצותה של בחירת תכוניות. אנו מעלים את השאלה האם שלב נפרד של בחירת תכוניות אכן נדרשת, או אולי אלגוריתמי למידה מודרניים יכולים להתגבר על נוכחותן של

מספר עצום של תכוניות בעצמם. בכדי לענות על שאלה זו אנו מציגים ניתוח חדש של הבעיה הפשוטה של תיוג נקודות שהוגרלו משתי מחלקות בעלות התפלגות גאוסית. בשלב ראשון אנו מציגים ניתוח עבור מפריד הנראות המרבית. אנו מנתחים את שגיאתו כפונקציה של מספר התכוניות ושל מספר דוגמאות האימון ומראים שבעוד שהשגיאה עלולה להיות גרועה כמו השגיאה של הכרעה אקראית כאשר משתמשים ביותר מידי תכוניות, היא שואפת לשגיאה האופטימאלית האפשרית כאשר בוחרים את מספר התכוניות בחכמה. בנוסף, אנו מוצאים באופן מפורש את מספר התכוניות המיטבי כפונקציה של מספר דוגמאות האימון עבור מספר דוגמאות ספציפיות. בשלב שני, אנו בוחנים את SVM [41] , שהוא אלגוריתם התיוג הבולט ביותר הלמידה חישובית כיום, באופן אמפירי באותו מטווה. התוצאות מראות שלמרות ש SVM נחשב לאלגוריתם המתמודד היטב עם מימד גבוה, ביצועיו תואמים במדויק את התחזיות של הניתוח שלנו. תוצאות אלו מרמזות שבחירת תכוניות עדיין מהווה מרכיב קריטי בתכנון של מערכות ללמידת מפרידים מדויקים, גם כאשר משתמשים באלגוריתמי למידה מודרניים, וגם כאשר אילוצים חישוביים ועלויות מדידת תכוניות לא מהווים שיקול.

בפרק 3 אנו מציעים שיטות חדשות לבחירת תכוניות בבעיות תיוג המבוססות על עקרון השוליים המרביים. שוליים [41, 001] הם מידה גיאומטרית להערכת מידת ביטחונו של מפריד ביחס להחלטתו. שוליים כבר משחקים תפקיד חשוב בלמידה חישובית עכשווית. לדוגמא, SVM הוא אלגוריתם בולט המבוסס על שוליים מרביים. החידוש בתוצאות המוצגות בפרק זה נעוץ בשימוש בעיקרון השוליים המרביים לשם בחירת תכוניות. השימוש בשוליים מאפשר לנו לפתח אלגוריתמים חדשים לבחירת תכוניות וגם להוכיח חסמי הכללה. חסמי ההכללה הם על שגיאת ההכללה של 1-Nearest-Neighbor כאשר נעשה שימוש בקבוצת תכוניות נבחרת. החסמים אינם ספציפיים לאלגוריתם בחירת תכוניות מסויים, אלא מבטיחים ביצועים טובים עבור כל אלגוריתם בחירת תכוניות הבוחר קבוצת תכוניות קטנה תוך שמירה על שוליים גדולים. בצד האלגוריתמי אנו משתמשים בקריטריון מבוסס שוליים בכדי למדוד את איכותם של קבוצות תכוניות. אנו מציגים שני אלגוריתמי בחירת תכוניות חדשים, *Simba* ו*G-flip* המבוססים על קריטריון זה. יכולתם של אלגוריתמים אלו מודגמת על מספר סוגי נתונים.

בפרק 4 אנו דנים בבחירת תכוניות בלמידת פונקציה רציפה (regression). אנו עושים שימוש באלגוריתם 1-Nearest-Neighbor פעם נוספת ומשתמשים בפונקצית הערכה הדומה בטבעה לזו בה השתמשנו בפרק 3. אנו מפתחים אלגוריתם לבחירת תכוניות בבעיית רגרסיה שהוא לא ליניארי אך עם זאת פשוט ויעיל. האלגוריתמים מסוגל לתפוס תלות מסובכת של פונקצית המטרה בקלטים ומשתמש בשגיאת ה leave-one-out כרגולריזמיה טבעית. אנו מסבירים את טבעו של האלגוריתם

שפיתחנו בעזרת נתונים מלאכותיים ומדגמים את יתרונותיו ביחס לאלגוריתמים ידועים. בהמשך, אנו משתמשים באלגוריתם שפיתחנו בהקשר של חיזוי מהירות תנועת יד מספייקים שהוקלטו בקורטקס של קוף מתנהג. על ידי שימוש בבחירת תכוניות אנו מצליחים לשפר את איכות החיזוי ומציעים דרך חדשה לניתוח נתונים נוירונליים. התוצאות מרמזות שתלות מורכבת בתכוניות השונות (באופן גס, כל תכונית היא פעילות של נוירון מסויים בפלח זמן מסויים) אכן קיימת בקוד עצבי זה.

בפרק 5 אנו מרחיבים את המטווה הסטנדרטי של בחירת תכוניות על ידי כך שאנו לוקחים בחשבון גם הכללה במימד התכוניות. המטרה של בחירת תכוניות במטווה הסטנדרטי היא לבחור תת-קבוצה של תכוניות מתוך קבוצה נתונה של תכוניות, והכללה נידונה רק בהקשר של דוגמה חדשה, ולא בהקשר של תכונית חדשה. בפרק זה, במקום לנסות ולמצוא ישירות אילו תכוניות טובות יותר, אנו מנסים ללמוד מהן המאפיינים של תכוניות טובות. לשם כך אנו מניחים שכל תכונית מיוצגת על ידי אוסף מאפיינים, להם אנו קוראים מטה-תכוניות. כעת ניתן להשתמש הקבוצת האימון על מנת ללמוד מיפוי ממטה-תכוניות של תכונית לאיכותה של התכונית, ואז להשתמש במיפוי זה בכדי לחזות את איכותה של תכונית חדשה שלא הופיעה בקבוצת האימון. ישנם שלושה יתרונות מרכזיים לגישה זו. דבר ראשון, היא מאפשרת להציג את בעיית בחירת התכוניות כבעיית למידה סטנדרטית. הסתכלות זו מאפשרת לתת חסמי הכללה טובים יותר לתהליך הכולל של בחירת תכוניות ותיוג. דבר שני, גישה זו מאפשרת לנו לפתח אלגוריתמי בחירת תכוניות המסוגלים לחפש תכוניות טובות ביעילות גם כאשר מספר התכוניות שעומדות על הפרק הוא עצום. לבסוף, גישה זו תורמת להבנת הבעיה. אנו מראים גם כיצד ניתן ליישם את גישתנו החדשה בהקשר של העברת ידע ממשימה למשימה (*inductive transfer*). אנו מראים שהעברת המאפיינים של תכוניות טובות עשויה להביא לתוצאות טובות יותר מאשר העברת הידע מיהן התכוניות הטובות עצמן. אנו מדגימים את השימוש במטה-תכוניות לשימושים השונים על בעיית זיהוי ספרות הכתובות בכתב יד.