Model Selection for Boosting and Deep Learning Binary Classification Methods

By Mario Alberto Triviño Munévar

a work submitted as a requirement for the degree of Mathematician

Advisor:
Adolfo Quiroz, PhD.

Departamento de Matemáticas
Facultad de Ciencias

2017

Acknowledgements

First of all I would like to thank my advisor, Professor Adolfo Quiroz for all his help. Without his patient encouragement and insight, I could not have done this job. I owe him many more Bolívares than he cares to admit. I also would like to thank Professor Mauricio Velasco for his patience and for taking the time to read this work.

I am blessed to have the family I have. My parents and my sister are always there for me. Their love and support help me in uncountable many ways.

I am grateful to my friends, specially to Diego, Juan Camilo, JC, Giefried, Adrián, Andrea and Yulieth, for constantly reminding me that there is much more to life than books and computers.

I thank God for all the opportunities, and all the silver linings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What is Machine Learning

Nowadays we hear very often stories about how *Big Data* and *Artificial Intelligence (AI)* are going to change the way we do things. They are supposed to be the ultimate tools that will give us better control and understanding of many of the tasks we do in our daily lives. While the former term is used to refer to large data sets and the ability to analyze them, identifying and exploiting patterns, the latter is the study of how to create intelligent agents, that is, it looks for ways to make a computer behave and perform a task as a human would.

But how exactly are they going to do that? They certainly pose great technical challenges since we want them to work with as little human intervention as possible. In the kind of applications we are interested in, we generally do not know what data our system is going to be presented with or what exact situation will our agent face. Because of this, the set of all possible decisions given all possible inputs is normally too complex to describe. We clearly have to devise a way for our system to adapt given its experience.

Machine Learning is the answer to such challenges. It is a scientific discipline that addresses the question of how to program systems to automatically learn and to improve with experience. This is very different from programming a computer to do a specific task (other than learning), for instance, two systems with the same algorithm may perform differently if they are presented with different sequences of

data. However, we also want our systems to act more alike when they are given enough data, i.e., we want them to be able to apply what they learn to new, yet to be seen data in a consistent way.[1] This is in fact the main goal in Machine Learning: generalization.

A formal definition of **Machine learning** is given by Mitchell[21]:

Definition 1.1. *A computer program is said to learn from experience* ***E*** *with respect to some class of tasks* ***T*** *and performance measure* ***P****, if its performance at tasks in* ***T****, as measured by* ***P****, improves with experience* ***E****.*

Machine Learning problems are generally classified into two big groups. The first one is called *Supervised Learning* and deals with problems where given an input vector $\mathbf{x}$ the task is to learn a function $f(\mathbf{x})$. This function can take arbitrary values in $\mathbb{R}^n$ where $n \geq 1$ (regression) or in a discrete and finite set (classification). The other group of problems is known as *Unsupervised Learning* and in this case rather than finding a function from the input data we are interested in studying the relation between data points themselves, e.g. existence of clusters. Note that in the former group we have a "teacher" who shows the right labels and the algorithms we develop will aim to learn a rule to predict those correctly, hence the name.

There are a wide variety of machine learning tasks and successful applications. The developed methods are at the basis of many applications, ranging from computer vision to language processing, pattern recognition, games, data mining, ranking systems and robotics.

In this work we will focus on binary classification, a task that belongs to Supervised Learning, however, many of the concepts and discussions presented here are easily extended to regression problems[5, 13].

1.2 Binary Classification

Definition 1.2. *A* ***classifier*** *is a function $f : \mathbb{R}^d \to A$, where A is a finite set.*

Definition 1.3. *A* ***two-class classifier*** *is a function $f : \mathbb{R}^d \to A$, where $|A| = 2$.*[2]

[1] Consistency in this context means that when both systems have enough experience, they will make very similar decisions. A formal definition will be given shortly.

[2] In this work we will consider $A = \{-1, 1\}$ unless stated otherwise.

In binary or two-class classification, given a set $S_m = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_m, y_m)\}$ where $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ for $d \geq 1$ and $i = 1, ..., m$, we assume S_m was constructed by taking a sequence of m independent identically distributed (i.i.d.) random pairs from an unknown probability distribution $\mathcal{D}$ on $\mathbb{R}^d \times \{-1, 1\}$, i.e., $\mathbf{z}_i = (\mathbf{x}_i, y_i) \sim \mathcal{D}$.

Our goal is to find a classifier f_m such that given a new sample $\mathbf{z}_s \sim \mathcal{D}$, $f_m(\mathbf{x}_s) = y_s$.

Note that f_m depends on S_m, as this classifier is found by some sort of optimization procedure that takes into account performance on the given data set. This process is called **learning**.

In practice we need a way to measure how good a classifier f is at predicting y given $\mathbf{x}$, to this end, the notion of an optimal classifier will be a great help.

Definition 1.4. *Let $(\mathbf{X}, \mathbf{Y}) \sim \mathcal{D}$. The best possible classifier f^* is defined by*

$$f^* = \underset{f:\mathbb{R}^d \to \{-1,1\}}{\arg\min} P_{\mathcal{D}}(f(\mathbf{x}) \neq y) \tag{1.1}$$

*and is called the **Bayes classifier** (or the **Bayes rule**).*

Note that f^* depends on $\mathcal{D}$. If this distribution is known, f^* may be computed. Note also that this classifier need not be unique, since several classifiers may achieve the same minimal error.

Definition 1.5. *For $(\mathbf{x}, y) \sim \mathcal{D}$, the **risk** L of a classifier f is defined as*

$$L(f) = P_{\mathcal{D}}(f(\mathbf{x}) \neq y). \tag{1.2}$$

*$L^* = L(f^*)$ is called the **Bayes error** and is the minimal possibility of error.*

Ideally we would like to calculate $L_m = L(f_m)$. However, this is not possible because we do not know $\mathcal{D}$ and we can only hope to find an estimate that gives us a tight bound with a high confidence. For now, we can say that in finding f_m we want it to be as close as possible to f^*, in a way yet to be defined.

1.3 Empirical Error and Generalization Error

Up to now we have only considered a specific classifier, f_m, but it is actually more important to think about a sequence of classifiers:

Definition 1.6. *A sequence of classifiers $\{f_m\}_{m\geq 1}$ is called a **(discrimination) rule**.*

Since our main goal in finding classifiers is generalization, a desirable feature is that our rule approaches the Bayes classifier as $m \to \infty$.

Definition 1.7. *Let $S_m = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_m, y_m)\}$ a set of m independently drawn random samples from $(\mathbf{X}, \mathbf{Y}) \sim \mathcal{D}$. A rule is **consistent** if*

$$\lim_{m\to\infty} \mathrm{E}[L_m] = L^* \tag{1.3}$$

or equivalently, that $L_m \to L^*$ in probability as $m \to \infty$ [9].

Since we cannot calculate L_m and this is a natural measure of performance, we will have to explore alternatives in order to estimate its value.

Definition 1.8. *Let S_m be a set of m independently taken samples from $(\mathbf{X}, \mathbf{Y}) \sim \mathcal{D}$ and let f be a classifier. The **empirical risk** or **empirical error** is*

$$\hat{L}_m(f) = \frac{1}{m}\sum_{i=1}^{m} I(f(\mathbf{x}_i) \neq y_i), \tag{1.4}$$

where I is the indicator function, namely

$$I(Q) = \begin{cases} 1 & Q \text{ is true} \\ 0 & Q \text{ is false} \end{cases} \tag{1.5}$$

Now in addition to the consistency requirement described above, we will be interested in classifiers which satisfy $\hat{L}_m \to L_m$ as $m \to \infty$.

How do we go about finding our rule? To minimize the empirical error seems like a good starting point. The general idea is to choose a family of hypotheses[3], say

[3]Classifier and hypothesis are terms that we will use interchangeably in this work.

for instance $\mathcal{H} = \{h_\theta \colon \theta \in \Theta \subseteq \mathbb{R}^n\}$ where Θ represents a parameter space, and to choose a h_θ that predicts y in a way that minimizes empirical error for S_m. Notice that there could be more than one function fulfilling this condition.

In these terms, our problem becomes:

$$\min_{h_\theta \in \mathcal{H}} \hat{L}_m(h_\theta) \tag{1.6}$$

If $\hat{L}_m \to L_m$, we expect this procedure will find a classifier near f^*. However, this need not be the case for small sample set sizes. We could define a classifier

$$g(x) = \begin{cases} y & \text{if } x \in S_m \\ 1 & \text{otherwise} \end{cases} \tag{1.7}$$

This would make $\hat{L}_m = 0$, but clearly we expect it to perform badly in the general case. The problem $g(x)$ has is that it completely neglects the importance of novel data. A way to improve the results of the learning procedure is to divide S_m into a training set S_T and a validation set S_V such that $S_m = S_T \cup S_V$ and $\{i \colon \mathbf{z}_i \in S_T\} \cap \{j \colon \mathbf{z}_j \in S_V\} = \emptyset$.[4] This will allow us to learn a hypothesis based on the training set and to test it on the validation set. Now our goal will not be to minimize empirical error, but an estimate of the generalization error obtained from the validation set.

1.4 Overfitting

The problem $g(x)$ has is that it relies too much on training data. Let us know delve a little deeper into the causes of this problem. The following reasoning follows closely that of Devroye et al. [9]. Suppose we are considering classifiers from a class $\mathcal{H}$ and

[4] Notice that we admit repeated samples, namely it is possible to have $\mathbf{z}_i = \mathbf{z}_j$ for $i \neq j$. It is also possible to have $\mathbf{x}_i = \mathbf{x}_j$ and yet $y_i = y_j$. In the latter case it is said that the problem has mislabeled data. However, the same treatment is still possible, but the Bayes Classifier will no longer have $L^* = 0$.

let us denote the empirically optimal classifier by h_m^*:

$$h_m^* = \arg\min_{h \in \mathcal{H}} \hat{L}_m(h) \tag{1.8}$$

We would like to know how far $L(h_m^*)$ is from L^*. This difference can be written as

$$L(h_m^*) - L^* = \left(L(h_m^*) - \inf_{h \in \mathcal{H}} L(h) \right) + \left(\inf_{h \in \mathcal{H}} L(h) - L^* \right) \tag{1.9}$$

For a family of classifiers $\mathcal{H}$, the size of $\mathcal{H}$ presents us with a trade-off: If it is large, we have a better chance of $\inf_{h \in \mathcal{H}} L(h)$ being close to L^*, but at the same time it will make it harder to find $L(h_m^*)$ close to $\inf_{h \in \mathcal{H}} L(h)$. How does this relate to the problem with $g(x)$? In a large hypothesis class there is a larger probability that there will be distinct classifiers that minimize empirical risk, even though they might be overly complicated and far from the best achievable within the class. In a simpler class we would expect a classifier like $g(x)$ to be less likely to exist because it becomes an ever increasingly complicated function as more data is taken into account.

This situation is known as the **bias-variance trade-off**: if we ask for a large precision on the training data set, we will have a high variance on the training set, whereas if we allow for a larger error on the training set (i.e. a higher bias), we expect our classifier will be characterized by a smoother function (with a lower variance).

1.5 Complexity and Regularization

In order to build a classifier like $g(x)$ we would need to have a lookup table listing the right output y for every input x, this means that for m data points we would have to store m parameters, namely the corresponding response for every x. Based on this observation, we see that we will have to somehow penalize complex hypotheses in favor of simpler ones.

The common approach is to include an additional term in eq. 1.6:

$$h = \arg\min_{h_\theta \in \mathcal{H}} \left(\mathcal{L}(y, h_\theta(\mathbf{x})) + \mathcal{C}(h_\theta) \right) \tag{1.10}$$

where $\mathcal{L}(y, h_\theta(\mathbf{x}))$ is a loss function, usually $\hat{L}_m(h_\theta)$. $\mathcal{C}(h)$ is a measure of the

complexity of the hypothesis h. This is called a ***regularization term***.

1.6 The Big Picture

Now that we have defined the problem of binary classification and outlined a very general procedure for learning, we can list some of the fundamental challenges that are faced for realization of such classifiers[19]:

- Is it possible to solve the problem using any model? This is equivalent to asking if we are dealing with a well-posed problem.
- Is it possible to solve the problem using a given $\mathcal{H}$?
- Is it possible to determine the right classifier from a data set?
- Is it possible to determine the classifier efficiently?

We do not have general answers for this questions. We see that finding a classifier is no trivial matter. That is why having tools for comparing classifiers plays a fundamental role at the moment of choosing a final classifier from a group of candidates.

Chapter 2

Model Selection

We have already seen that a classifier h_θ with parameters θ may fit well the training set, but it is very likely that the training error will be lower than the actual generalization error. Therefore, we need to develop methods for choosing a classifier from a set of viable candidates (those with small $\hat{L}_{S_T}$) in order to reduce the expected generalization error.

The general solving process for a two-class classification problem can be outlined as follows:

1. Given a binary classification problem (in the form of a set of observations S_m, containing each a feature vector $\mathbf{x}_i$ and the corresponding label y_i), check if it seems well-posed and verify if the problem has a deterministic nature and the relationship between variables and labels is clearly understood. If it is so, this might not be a problem for machine learning, as there are better tools (computationally speaking) available to us and we do not need to learn from data. Otherwise, proceed to step 2.

2. Randomly divide the data into **at least** two sets: one for training and another one for testing.

3. Choose a family of hypotheses $\mathcal{H}$. Look for classifiers within this family by minimizing a function $\mathcal{F}$ that depends on the empirical error and possibly on the complexity of the classifiers. This will yield hypotheses described by a set of parameters θ for each h_θ.

4. Assess the performance of classifiers h_θ with an acceptable empirical error on novel data.

5. Pick the best classifier based on performance on the test set and other specified criteria.

6. Estimate generalization error for the final model.

When we talk about model selection we generally consider steps 4 to 6. Step 6 is usually called **model assessment**.

There is an important remark about step 2. In chapter 1 we said that the data could be split into two different sets. However, if we are in a data-rich situation, it is actually desirable to make three groups instead of two. One of them will be the training set, used to fit the model (step 3), the second one will be a validation set, used to estimate prediction error for model selection (step 5), and the last set is used to estimate the generalization error of the final model (step 6).

Ideally we would not mix our validation and test sets, since the use of the same samples for both model selection and model assessment will lead us to underestimate the true prediction error.

How many samples do we need for every set? This is a difficult question to answer, but generally it is preferred to assign a larger fraction of samples to the training set (say 50%), as it is the one the hypothesis is learned from, and the rest of the data is split evenly into two groups, one for validation and one for testing.

In this chapter we describe some methods for estimating the generalization error for a model, specially when the data is insufficient for partitioning our samples into three sets. A method approximates the validation step analytically (SRM) while the other takes advantage of efficient sample re-use (CV).

2.1 Cross Validation (CV)

Consider the overfitting problem we discussed in section 1.4. How can we overcome it? One way to avoid overfitting is to use a lot of data. The main reason overfitting happens is because the data set is small and we try to learn from it. The learning procedure for finding h has then great control over the data and therefore tries to

find the right label for every datapoint. If the data set is large, say a million samples, then the algorithm is forced to generalize and to come up with a model that fits all points reasonably well.

Sadly, in most situations we do not have such large data sets. That is when this technique comes into play. Cross validation is a distribution-free method for measuring the predictive performance of a statistical model. The general idea is to split the data training set into two sets and to use only one of them for training, while the other set is used for validation. Once this has been done, the process can be repeated with different partitions of the training set.

2.1.1 Holdout Cross Validation

This the simplest kind of cross validation. The data set is separated into two sets, the training set and the test set. Note that this is what we said we should always do: learn in a set and test in another. The problem with this approach is that the learning and validation results may depend heavily on which data points end up in the training set and which end up in the test set, and thus the classification results may be significantly different depending on how the division is made. In a data-rich situation this problem can be somewhat alleviated by taking the test set to be larger than the training set (it is desirable that the test set size be at least an order of magnitude bigger). Intuitively, this gives a higher confidence in the result.

In holdout cross validation the model with the smallest test error is chosen.

2.1.2 K-fold Cross Validation

This is a way to improve the holdout method. In this case the data is divided into k subsets, and the holdout method is repeated k times. Each time one of the k sets is used for validation while the remaining $k-1$ sets are combined to form the training set. For every round the test error is computed and in the end all of them are averaged. If we denote by $h^{-k}(\mathbf{x})$ the classifier learned from the holdout leaving subset k out, we can write the estimate of the prediction error from $k-$fold cross validation as

$$CV(\hat{h}) = \frac{1}{m}\sum_{i=1}^{m} I(y_i \neq h^{-k(i)}(\mathbf{x})) \tag{2.1}$$

The model with the smallest mean error is then selected. The advantage of this method is that it matters less how the data gets divided. Every sample gets to be in a test set exactly once, and gets to be in a training set $k-1$ times. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation. We expect the variance to decrease as k becomes large. However, this might not be the case as we will see in the next kind of cross validation.

2.1.3 Leave-one-out Cross Validation

Leave-one-out cross validation is $k-$fold cross validation taken to its limits, namely $k = m$, where m is the number of samples in the data set. This means that a classifier is trained on all data but one point, and a prediction is made for the remaining point. Because of this, the learning algorithm has to be run m times on data sets of size $m - 1$, which can be computationally intensive. This is actually not worth it. Consider the case of exact duplicate observations, then leaving one observation out will not be effective. Besides, we would have only one test sample in each run and the results should therefore have a high variance.

In fact, Shao proved that this kind of cross validation does not lead to a consistent estimate of the model[25]. This means that if there is a true model between the ones being considered, leave-one-out cross validation will not find it, even with very large data set sizes.

2.2 Structural Risk Minimization (SRM)

Let us consider once again the problem of finding an optimal classifier that was stated in eq. 1.6:

$$h = \arg\min_{h_\theta \in \mathcal{H}} \left(\hat{L}_m(h_\theta) + \mathcal{C}(h_\theta) \right)$$

where we now have taken $\mathcal{L}(y, h_\theta(\mathbf{x})) = \hat{L}_m(h_\theta)$. Recall that the second term on the right, $\mathcal{C}(h_\theta)$, was introduced as a way to prevent overfitting. The idea is that since empirical error alone is not good enough to predict generalization error, a complexity term helps the algorithm to avoid being overly optimistic about its performance on the training set as a measure of L. We will now motivate and explain a procedure that follows this approach and that is guaranteed to be consistent.

Let us start by stating **Hoeffding's inequality**, a very powerful result[9].

Theorem 2.1. *Let $X_1, \ldots, X_m$ be independent bounded random variables such that X_i falls in the interval $[a_i, b_i]$ with probability one. Let $S_n = \sum_{i=1}^m X_i$. Then, for any $\varepsilon > 0$*

$$P(S_m - \mathrm{E}[S_m] \geq \varepsilon) \leq e^{-2\varepsilon^2 / \sum_{i=1}^m (b_i - a_i)^2} \tag{2.2}$$

and

$$P(S_m - \mathrm{E}[S_m] \leq -\varepsilon) \leq e^{-2\varepsilon^2 / \sum_{i=1}^m (b_i - a_i)^2} \tag{2.3}$$

Notice that the empirical risk can be seen as the weighted sum of the indicator function on the training set, therefore $S_m = m\hat{L}_m(h)$, where $X_i = I(h(\mathbf{x_i}) \neq y_i)$. This leads to the following result.

Corollary 2.1. *For a binary classification problem, given a fixed classifier h, m random training samples and any $\delta > 0$,*

$$P(L_m \geq \hat{L}_m + \varepsilon) \leq e^{-2m\varepsilon^2} \tag{2.4}$$

Moreover, with probability $1 - \delta$

$$L_m \leq \hat{L}_m + \sqrt{\frac{\ln(1/\delta)}{2m}} \tag{2.5}$$

Proof.

$$P(\hat{L}_m - L_m \leq -\varepsilon) = P(m(\hat{L}_m - L_m) \leq -m\varepsilon) \tag{2.6}$$

Note that $I(h(\mathbf{x_i}) \neq y_i) \in [0, 1]$. Now using Hoeffding's inequality (eq. 2.3),

$$P(\hat{L}_m - L_m \leq -\varepsilon) \leq e^{-2(m\varepsilon)^2/\sum_{i=1}^{m}(b_i-a_i)^2} \tag{2.7}$$

$$P(\hat{L}_m + \varepsilon \leq L_m) \leq e^{-2(m\varepsilon)^2/\sum_{i=1}^{m}(1)^2} \tag{2.8}$$

$$P(\hat{L}_m + \varepsilon \leq L_m) \leq e^{-2m\varepsilon^2} \tag{2.9}$$

If we take $\delta = e^{-2m\varepsilon^2}$, then

$$P(\hat{L}_m + \varepsilon \leq L_m) \leq \delta \tag{2.10}$$

and hence

$$P(\hat{L}_m + \varepsilon \geq L_m) \leq 1 - \delta. \tag{2.11}$$

Eq. 2.5 is obtained by writing ε in terms of δ. □

We have now found an upper bound for the generalization error of a **fixed classifier** in terms of the number of samples and an arbitrary confidence parameter. Note also that a lower bound could be obtained in exactly the same way.

It is extremely important to remember that this result is not valid for a learning algorithm. In this case, the problem is that the training set is used twice, once for selecting the best classifier h and then for estimating the generalization error; therefore the independence hypothesis of theorem 2.1 no longer holds. However, corollary 2.1 gives us some insight about the kind of bound we should expect to find: we want the estimate to improve as the number of samples used for training increases or the empirical error decreases. The only thing that we expect the generalization error to depend on that is missing is the complexity of the hypothesis, that is not taken into account in these bounds.

Since we want to find a similar bound for a learning algorithm, the general idea is to choose a family of classifiers $\mathcal{H}$ in a way that guarantees that $|\hat{L}_m(h) - L_m(h)| \leq \varepsilon$ with a high probability and then to find a bound that is applicable to every hypothesis in $\mathcal{H}$. Finally the bound is extended to the whole family using the union bound of probabilities or some sort of extension of it.

The simplest case to consider is when $\mathcal{H}$ is a finite family[23].

Theorem 2.2. *Let $\mathcal{H}$ be a finite space of hypotheses and assume that a random training set $S_m = \{Z_i\}$ of size m is chosen. Then for any $\varepsilon > 0$,*

$$P(\exists h \in \mathcal{H} \colon L(h) \geq \hat{L}_m(h) + \varepsilon) \leq |\mathcal{H}| e^{-2m\epsilon^2} \tag{2.12}$$

Thus, with probability $1 - \delta$,

$$L_m(h) \leq \hat{L}_m(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln(1/\delta)}{2m}} \tag{2.13}$$

Proof. Since the hypothesis space $\mathcal{H}$ is fixed before training, we can think of it like having $|\mathcal{H}|$ hypotheses fixed, and hence for each one of them we can apply corollary 2.1:

$$P(\exists h \in \mathcal{H} \colon L(h) \geq \hat{L}_m(h) + \varepsilon) \leq \underbrace{e^{-2m\epsilon^2} + \cdots + e^{-2m\epsilon^2}}_{|\mathcal{H}|\text{-times}} \tag{2.14}$$

$$P(\exists h \in \mathcal{H} \colon L(h) \geq \hat{L}_m(h) + \varepsilon) \leq |\mathcal{H}| e^{-2m\epsilon^2} \tag{2.15}$$

$$\tag{2.16}$$

Setting $\delta = |\mathcal{H}| e^{-2m\varepsilon^2}$ and solving for ε we obtain

$$\varepsilon = \sqrt{\frac{\ln |\mathcal{H}| + \ln(1/\delta)}{2m}} \tag{2.17}$$

and hence with probability $1 - \delta$

$$L_m(h) \leq \hat{L}_m(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln(1/\delta)}{2m}} \tag{2.18}$$

for all $h \in \mathcal{H}$. □

Eq. 2.18 has an additional $\ln |\mathcal{H}|$ when compared to eq. 2.5. Intuitively, we expect the complexity to grow with the size of the family, and hence we can see this term as a measure of the family's complexity. Now the upper bound also increases with the size of the hypothesis space, in agreement with the discussion from section 1.4.

What happens when there is a large number of hypotheses in $\mathcal{H}$? If $|\mathcal{H}|$ is very large, the bound in eq. 2.18 becomes too loose or completely ineffective. Yet, there are cases where even though the space has an infinite number of members, there seems to be only a small number of essentially different hypotheses. Consider for instance, $\mathcal{H}$ to be the set of linear separators in $\mathbb{R}^2$. Despite the fact that there is an infinite number of them, they cannot be used to separate all possible 2-colorings of any 4-point configuration.

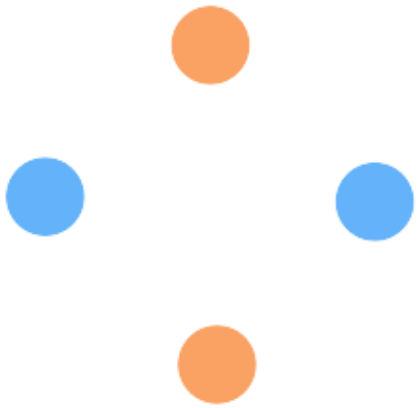

Figure 2.1: Example of the limitations of linear separators. For this configuration, no line in $\mathbb{R}^2$ correctly separates the two colors.

Because of this, it is reasonable to think that we could replace the cardinality of the set for some sort of "effective size" or "capacity". The concept of a growth function realizes this notion in the context of binary classification.

2.2.1 Growth function and VC-dimension

We are now going to explore a concept that will allow us to find a nontrivial bound like the one from theorem 2.2 for large families of classifiers.

Definition 2.1. *A **dichotomy** of a set S is a partition of S into two disjoint subsets. If $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is a finite sample, $\Pi_{\mathcal{H}}(S) = \{\langle h(\mathbf{x}_1), \ldots, h(\mathbf{x}_m)\rangle : h \in \mathcal{H}\}$ is the set of all possible dichotomies, i.e., all possible labelings of S by the binary functions in $\mathcal{H}$.*

Definition 2.2. *The* ***growth function*** *of $\mathcal{H}$, $\Pi_{\mathcal{H}}(m)$, is the maximum number of dichotomies for any sample S of size m, namely*

$$\max_{S \in X^m} |\Pi_{\mathcal{H}}(S)|. \tag{2.19}$$

It is possible to state an analog result to that of theorem 2.2 in terms of $\Pi_{\mathcal{H}}(m)$[9, 23].

Theorem 2.3. *Let $\mathcal{H}$ be a measurable space with respect to an appropriate probability space, and assume that m random training samples are chosen. Then for any $\varepsilon > 0$*

$$P(\exists h \in \mathcal{H} \colon L_m(h) \geq \hat{L}_m(h) + \varepsilon) \leq 8\Pi_{\mathcal{H}}(m) e^{-m\varepsilon^2/32} \tag{2.20}$$

Hence, with probability $1 - \delta$

$$L_m \leq \hat{L}_m + \sqrt{\frac{32[\ln(8/\delta) + \ln \Pi_{\mathcal{H}}(m)]}{m}} \tag{2.21}$$

Note that if $\mathcal{H}$ is capable of realizing any labeling, then $\Pi_{\mathcal{H}}(m) = 2^m$ and hence $\ln \Pi_{\mathcal{H}}(m) = m \ln 2$. In such case the bound given in eq. 2.21 is useless. On the other hand, if $\Pi_{\mathcal{H}}(m) = O(m^d)$, we have a much favorable $\sqrt{d \ln m/m}$ rate of convergence and d turns out to be a good measure of complexity. Let us now try to determine the value of d.

The VC-dimension is a measure of capacity or complexity for families of binary classifiers. It is the maximum number of points that can be classified by $\mathcal{H}$ if they are given any spatial configuration and labeling. Let us know state this in a formal way.

Definition 2.3. *A set of instances S is* ***shattered*** *by the hypothesis space $\mathcal{H}$ if and only if for every dichotomy of S there exists some hypothesis in $\mathcal{H}$ consistent with this dichotomy, i.e., $|\Pi_{\mathcal{H}}(S)| = 2^m$.*

Definition 2.4. *The* ***Vapnik-Chervonenkis dimension,*** *$V_{\mathcal{H}}$ of a hypothesis space $\mathcal{H}$ defined over an instance space X is the largest finite subset of X than can be shattered by $\mathcal{H}$. If arbitrarily large finite sets of X can be shattered by $\mathcal{H}$, then $V_{\mathcal{H}} = \infty$.*

The importance of $V_{\mathcal{H}}$ comes from **Sauer's lemma**[23]:

Lemma 2.1. *If $\mathcal{H}$ is a hypothesis space of $V_{\mathcal{H}} = d < \infty$, then for all m*

$$\Pi_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \tag{2.22}$$

Moreover, if $m \geq d \geq 1$

$$\sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^{d} \tag{2.23}$$

Plugging eq. 2.23 back into theorem 2.3 we get:

Theorem 2.4. *Let $\mathcal{H}$ be a measurable class with respect to an appropriate probability space with $V_{\mathcal{H}} = d < \infty$. Assume a set of m random training samples is given and $m \geq d \geq 1$, then*

$$P(\exists h \in \mathcal{H} \colon L_m(h) \geq \hat{L}_m + \varepsilon) \leq 8 \left(\frac{em}{d}\right)^{d} e^{-m\varepsilon^2/32} \tag{2.24}$$

Thus, with probability $1 - \delta$

$$L_m \leq \hat{L}_m + \sqrt{\frac{32[\ln(8/\delta) + d \ln\left(\frac{em}{d}\right)]}{m}} \tag{2.25}$$

The use of the VC-dimension allows us to find bounds like those from the previous section for a larger group of binary classifiers. Nevertheless, due to the combinatorial nature of this concept, it is usually hard to calculate the exact $V_{\mathcal{H}}$ and we usually have only bounds on its size, making the constraints on the generalization error looser. Moreover, not every $\mathcal{H}$ has a finite VC-dimension.

2.2.2 SRM procedure

Since $V_{\mathcal{H}}$ is an adequate measure of complexity, we can use it to define a hierarchy of hypothesis spaces **before** any data is observed. If the spaces of hypotheses considered have a finite VC-dimension, then it is possible to carry out the following model selection procedure:

1. Choose a class of hypothesis, e.g. polynomials of degree n, neural networks with n hidden layer neurons, or AdaBoost classifiers with n weak learners.

2. Divide the class of functions into a hierarchy of nested subsets in order of increasing complexity.

3. Perform empirical risk minimization on each subset.

4. Select the model in the sequence with smallest $\hat{L}_m + \sqrt{\frac{32[\ln(8/\delta)+V_{\mathcal{H}}\ln\left(\frac{em}{V_{\mathcal{H}}}\right)]}{m}}$.

The next theorem states that this method avoids overfitting of the data[9].

Theorem 2.5. *Let $\mathcal{H}^{(1)}, \mathcal{H}^{(2)}, \ldots$ be a sequence of families of classifiers such that for any distribution of (X, Y)*

$$\lim_{i\to\infty} \inf_{h\in\mathcal{H}^{(i)}} L(h) = L^* \tag{2.26}$$

Assume also that the VC-dimensions $V_{\mathcal{H}^{(1)}}, V_{\mathcal{H}^{(2)}}, \ldots$ are all finite and satisfy

$$\Delta = \sum_{i=1}^{\infty} e^{-V_{\mathcal{H}^{(i)}}} < \infty \tag{2.27}$$

Then the classifier h_m^* based on the procedure described above is strongly universally consistent.

2.3 Information Criteria

Information Criteria for model selection take a different approach based on the Maximum Likelihood Principle: they are based on minimizing an estimate for the **Kullback-Leibler divergence (KL)**.

For two probability distributions P and Q of a continuous random variable, the

KL-divergence is defined as

$$KL(P||Q) = \int_{-\infty}^{\infty} \ln\left(\frac{p(x)}{q(x)}\right) p(x)dx \tag{2.28}$$

$$KL(P||Q) = \underbrace{\int_{-\infty}^{\infty} p(x)\ln p(x)dx}_{\text{entropy}} \underbrace{- \int_{-\infty}^{\infty} p(x)\ln q(x)dx}_{cross-entropy} \tag{2.29}$$

This integral takes positive values unless $p(x) = q(x)$ almost everywhere. We can think of the $KL-$divergence as a measure of how close two probability distributions are from each other. This gives us a criterion for model selection, just consider $p(x)$ and $q(x)$ to be the density functions associated with the true model and the hypothesis h_θ being tested and minimize the $KL-$divergence. The problem with this approach is that for this we require the real $p(x)$. Both Akaike's and Bayes Criterion rely on finding an estimate for $KL(P||Q)$ without having $p(x)$[17].

2.3.1 Akaike's Information Criterion (AIC)

Akaike's Information Criterion is defined as

$$AIC = -2\ln\mathcal{L} + 2k \tag{2.30}$$

where $\mathcal{L}$ is the maximized likelihood using all available data for estimation and k is the number of free parameters in the model. Stone proved that asymptotically, for any model, minimizing the AIC is equivalent to minimizing the cross validation value[26].

2.3.2 Bayesian Information Criterion (BIC)

Schwarz's Bayesian Information Criterion is defined as

$$BIC = -2\ln\mathcal{L} + k\ln(n) \tag{2.31}$$

where n is the number of observations used for estimation. Note that BIC has a heavier penalty than AIC, because of this, the model chosen by BIC is either the

same as that chosen by AIC, or one with fewer terms.

BIC is a consistent method for model selection, i.e. given enough data BIC will select the true model.[24]

Chapter 3

Boosting

3.1 Boosting Algorithms

The idea behind boosting algorithms is to combine a set of "dumb" individuals to form a committee that can make better decisions than the individuals can on their own. We are going to call those individuals **weak learners** or **base classifiers** interchangeably.

An important assumption is that all base classifiers h belong to the same family $\mathcal{H}$. We do not, however, care much about the internal structure of these weak learners or intend to modify them in any way. What we want is to have a voting committee rather than a single classifier, namely

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{3.1}$$

where α_t is the weight assigned to h_t and $f(\mathbf{x})$ is the final classifier.

Note that a boosting algorithm's only means of learning is through calls to the base classifier learning procedure. If we simply call this routine every time we want to add a new classifier, we would expect to have identical or very similar results, therefore we come to the conclusion that the data we feed to the weak learner algorithm must be altered in some way. In fact, that is the key of boosting: to force the weak learner algorithm to output a base classifier that **infers something new** every time it is called.

This leads us to two important issues:

- What samples should be used for choosing the weak learners that will be selected?
- Once the classifiers have been selected, how are they to be combined?

3.2 AdaBoost

This is the most popular boosting algorithm and is due to Freund and Schapire [11]. Given m training samples, AdaBoost initializes all weights to $\frac{1}{m}$ and at every iteration t, the samples that were misclassified in iteration $t-1$ have their weights increased, while those that were correctly classified have their weights decreased. In this manner, observations that are difficult to classify become more important with every round and therefore the new classifier is forced to concentrate on samples that were misclassified by previous ones in the sequence. Algorithm 1 describes in detail the procedure.

Since AdaBoost final classifier outputs labels rather than continuous values, it is sometimes called "discrete AdaBoost"[13].

3.2.1 Weak learners

In section 3.1 we mentioned that there are not many constraints on the nature of base classifiers, however, we must now state formally the only requirement we impose on them.

Definition 3.1 (Weak learning assumption)**.** *The weak learner produces a weak hypothesis h that is at least slightly better than random guessing on the samples on which it was trained.*

What does this mean? It means that we do not want $\hat{L}_m(h) = \frac{1}{2}$[1]. In practice it is very hard to find this extreme case of bad performance, and therefore this condition is often disregarded.

[1] If the empirical error of the base classifier $h(x)$ is greater than 0.5 we could get a better hypothesis in a trivial way by choosing $\bar{h}(x)$ to assign opposite labels to those of $h(x)$

Algorithm 1 The boosting algorithm AdaBoost

Inputs: $T \in \mathbb{N}$ and $S_m = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$ where $\mathbf{x}_i \in X$ and $y_i \in \{-1, 1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, ..., m$.
for t=1 to T **do**
Train weak learner using distribution D_t.
Get weak hypothesis $h_t : \mathbf{X} \to \{-1, 1\}$ that minimizes the weighted error:

$$\epsilon_t = P_{D_t}(h_t(x_i) \neq y_i) = \sum_{i:\, h_t(x_i) \neq y_i} D_t(i)$$

Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.
for i=1 to m **do**

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where Z_t is a normalization factor such that $\sum_{i=1}^{m} D_{t+1}(i) = 1$.
end for
end for
Return final hypothesis:

$$H(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

The good news is that if the weak learning assumption is fulfilled, the training error drops very quickly. The following theorem[23] states this precisely.

Theorem 3.1. *Given the notation of algorithm 1, let $\gamma_t = \frac{1}{2} - \epsilon_t$ and let D_1 be an arbitrary initial distribution over the training set. Then the weighted training error of the combined classifier H with respect to D_1 is bounded as*

$$P_{i \sim D_1}(H(x_i) \neq y_i) \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq \exp \left(-2 \sum_{t=1}^{T} \gamma_t^2 \right). \tag{3.2}$$

Since there are no hard conditions on weak learners, how should we choose the right family? There are many possibilities and it somewhat depends on the specific application, but the general consensus is that base classifiers should not be too complicated as a means of avoiding overfitting and to keep computational complexity

low.

3.2.2 Decision Stumps

The kind of weak classifier that we now define is by far the most widely used in current applications. It can be seen as a simple rule or yes/no question on one of the input features.

Definition 3.2. *A **decision stump** is a classifier $h_{i,a,b}$ in $\mathbb{R}^d$ such that for $\mathbf{x} = (x_1, x_2, ..., x_d)$*

$$h_{i,a,b}(\mathbf{x}) = \begin{cases} 1 & \text{if } ax_i \leq b \\ -1 & \text{otherwise.} \end{cases} \tag{3.3}$$

Without loss of generality, $a \in \{-1, 1\}$.

The appeal of this classifier comes from its simplicity, which allow us to do the weak learner training step in a short time.

VC dimension of Decision Stumps

We will need the VC dimension of the family of base classifiers in order to use the empirical risk bound on generalization error for AdaBoost. Therefore, we now calculate some bounds for the VC dimension of decision stumps.

First of all, note that in $\mathbb{R}$, we can use a family of intervals $\mathcal{A} = \{(-\infty, a] \colon a \in \mathbb{R}\}$ as two-class classifiers by defining $f_a(x_i) = 1$ if $x_i < a$ and -1 otherwise. Given n points, $\mathcal{A}$ is able to provide $n + 1$ different labelings. By symmetry, the classifiers f_b induced by $\mathcal{B} = \{[b, \infty) \colon b \in \mathbb{R}\}$ also yield $n + 1$ different configurations. If we take $\mathcal{C} = \mathcal{A} \cup \mathcal{B}$, we realize the classifiers produced by $\mathcal{C}$ are able to provide at most $2n + 2$ labelings. A closer inspection will reveal that there are actually at most $2n$ configurations since both $\{f_a\}$ and $\{f_b\}$ classify the set of all points and the empty set as 1. Now, how can we define $\{f_a\} \cup \{f_b\}$? This union are the decision stumps in one dimension. Since the same analysis is applicable if we consider every dimension in $\mathbb{R}^d$, decision stumps in this space can produce up to $2nd$ different labelings.

Suppose the VC dimension of stumps is k, this means there is a set of k points

that the stumps can shatter, therefore:

$$2^k \leq 2dk \tag{3.4}$$

$$k \leq \log_2(2dk) \tag{3.5}$$

$$k \leq 1 + \log_2 d + \log_2 k \tag{3.6}$$

This does not look too good, but we can actually get a bound for k is we consider high dimensional spaces.

Proposition 3.1. *Let $d \geq 5$, then the VC dimension of decision stumps in $\mathbb{R}^d$ is at most $2(1 + log_2 d)$.*

Proof. Since $2^k > k^2$ for $k \geq 5$ (a fact that can be proved by induction), we also have that $2\log_2 k < k$. Using this in eq. 3.6, we obtain:

$$k < 1 + log_2 d + \frac{k}{2} \tag{3.7}$$

$$\frac{k}{2} < 1 + log_2 d \tag{3.8}$$

$$k < 2(1 + log_2 d) \tag{3.9}$$

□

Now we have a bound on the $VC-$dimension of this family of base classifiers that could be used to calculate the SRM bound on generalization error for adaBoost. However, we note that $1.3(\log_2 d + 3)$ was empirically discovered to be a tighter bound. Besides, for low dimensional spaces we can get an even better bound using eq. 3.4 and computing k by exhaustion.

3.3 SRM Bounds for AdaBoost

The bounds for empirical error from section 2.2 can be specified further in terms of the parameters of the AdaBoost algorithm, namely, the number of training samples, the number of rounds and a complexity measure. The following theorems state such results. The details can be found in the work of Schapire and Freund [23].

Theorem 3.2. *Suppose AdaBoost is run for T iterations on $m \geq T$ random samples using weak classifiers from a finite space $\mathcal{H}$. Then, with probability at least $1-\delta$ (over the choice of the random sample), the combined classifier H satisfies*

$$L(H) \leq \hat{L}_m(H) + \sqrt{\frac{32[T\ln(em|\mathcal{H}|/T) + \ln(8/\delta)]}{m}} \tag{3.10}$$

Theorem 3.3. *Suppose $\mathcal{H}$ is a family of weak learners with $VC-$dimension $V_{\mathcal{H}} \geq 1$ and AdaBoost is run for T iterations on $m \geq \max\{T, V_{\mathcal{H}}\}$ random samples using base classifiers from $\mathcal{H}$. Then, with probability at least $1-\delta$ (over the choice of the random sample), the combined classifier H satisfies*

$$L(H) \leq \hat{L}_m(H) + \sqrt{\frac{32[T(\ln(em/T) + V_{\mathcal{H}}\ln(em/V_{\mathcal{H}})) + \ln(8/\delta)]}{m}} \tag{3.11}$$

3.4 Margin Bounds for AdaBoost

A closer look at eq. 3.11 (see table 5.1) will reveal that overfitting is predicted for a small number of rounds. Therefore, the number of weak learners should be increased up until the empirical error becomes zero or before. However, in most practical applications it has been shown that allowing to run the algorithm for many more rounds after $\hat{L}_m = 0$ actually improves the generalization error. This is due to the fact that the adaBoost algorithm does not tries minimize empirical error, but to maximize normalized margins[23].

Definition 3.3. *If $f(\mathbf{x}) = \sum_{t=1}^{t} \alpha_t h_t(\mathbf{x})$, then*

$$M(y, f(\mathbf{x})) = \frac{yf(\mathbf{x})}{\sum_{t=1}^{T} \alpha_t} \tag{3.12}$$

is called the normalized margin of y and $f(\mathbf{x})$.

Note that $M(y, f(\mathbf{x})) \in [-1, 1]$ and that it is negative only if $\mathbf{x}$ is misclassified by H, therefore pushing the margins to the right implies reducing error. Note also that $|M(y, f(\mathbf{x}))|$ is large if the classifier has a high confidence on its prediction. This

intuitively explains why generalization error keeps improving after zero empirical error has been achieved: the algorithm is simply try to improve the confidence it has on each prediction.

This also lead us to believe that the number of rounds is not really an accurate complexity criterion and the bounds from last section should be modified to take the margin information into account. This leads to the following modifications[23]: For a finite family of weak classifiers,

Theorem 3.4. *Let $S_m = \{z_i\}$ a set of m samples where $Z_i \sim \mathcal{D}$. Assume that the base classifier space $\mathcal{H}$ is finite and let $\delta > 0$. Then for all $\theta > \sqrt{(\ln|\mathcal{H}|)/4m}$, with probability at least $1 - \delta$,*

$$P_{\mathcal{D}}(M(y, f(\mathbf{x})) \leq 0) \leq P_S(M(y, f(\mathbf{x})) \leq \theta) + O\left(\sqrt{\frac{\ln|\mathcal{H}|}{m\theta^2}\ln\left(\frac{m\theta^2}{\ln|\mathcal{H}|}\right) + \frac{\ln(1/\delta)}{m}}\right) \tag{3.13}$$

For $|\mathcal{H}| = \infty$ we use the VC-dimension of the base classifiers instead:

Theorem 3.5. *Let $S_m = \{z_i\}$ a set of m samples where $Z_i \sim \mathcal{D}$. Suppose that the base classifier space $\mathcal{H}$ has VC-dimension $V_{\mathcal{H}}$. Assume $m \geq V_{\mathcal{H}} \geq 1$ and let $\delta > 0$. Then with probability at least $1 - \delta$ over the choice of the random set S_m,*

$$\begin{aligned} P_{\mathcal{D}}(M(y, f(\mathbf{x})) \leq 0) \leq & P_S(M(y, f(\mathbf{x})) \leq \theta) + 4e^{-n\theta^2/8} \\ & + \sqrt{\frac{32[\ln(n(n+1)^2) + nV_{\mathcal{H}}\ln(em/V_{\mathcal{H}}) + \ln(8/\delta)]}{m}} \end{aligned}$$

for all $\theta > \sqrt{8V_{\mathcal{H}}\ln(em/V_{\mathcal{H}})}$ and $n = \left\lceil \frac{4}{\theta^2}\ln\left(\frac{m\theta^2}{8V_{\mathcal{H}}\ln(em/V_{\mathcal{H}})}\right)\right\rceil$.

The proofs of these theorems can be found in the work of Schapire and Freund[23].

Chapter 4

Deep Learning

Deep learning refers to the use of deep architectures rather than shallow ones[1]. A shallow architecture is one where we can think of the system as containing at most one or two layers of nonlinear feature transformations, usually followed by some sort of linear classifier. On the other hand, a deep architecture has hierarchical levels, which can be thought of as the classifier being some sort of composition of functions.

Even though the first machine learning systems featured some deep architectures, shallow ones quickly became mainstream because of the problems associated with the learning algorithms for classifiers where knowledge had to be propagated through several layers.

4.1 Motivation for Deep Architectures

The desire for deep architectures is largely inspired by nature. The study of the brain and the nervous systems suggests that our reasoning comes from the hierarchical way our cells are organized. Besides, observation leads us to believe that we are programmed with a "one learning algorithm" that we use for every possible task we are able to perform. This means that the mechanics and the architecture of the system do not change from task to task, but only the inputs and desired outputs. We also have the capability to transfer knowledge gained from experience in one task

[1]The architecture specifies what variables are involved in the process and their topological relationships [20].

to another.

The problem with shallow architectures is that they do not look at all like this ideal. For instance, distinct families of classifiers have been devised for different kinds of tasks, learning cannot usually be transferred from a task to another in an obvious way, and features are generally preprocessed and hand-picked in order to improve performance. This has led some people to say that most of today's practical applications of machine learning use merely "glorified linear classifiers" or "glorified template matching"[18].

In order to overcome these difficulties, deep learning strives for building layered architectures that also have the ability of learning features in an unsupervised way.

4.2 Neural Networks

Neural networks were first developed as a mathematical model of information processing in the biological systems[5]. They can be considered an extension of linear classification and regression models.

In this work, the neural networks we are interested in are called **feed-forward networks**. In a linear classification model, the label prediction $\hat{y}$ is given by

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i \phi_i(x)\right) \tag{4.1}$$

where $w_i \in \mathbb{R}$, every $\phi_i(x)$ is a fixed, not necessarily linear function, and f is some sort of nonlinear activation function. The idea now is to allow the basis functions to depend on some parameters and then to adjust those along with the weights during training. Now the question is, how can we select a family of $\Phi = \{\phi_i\}$? Certainly there are many choices. In feed-forward networks the answer is to use functions of the form eq. 4.1. For instance, consider the input vector to the network, $\mathbf{x}$. We then can define a new unit

$$z_j = f_j\left(\sum_{i=1}^{m} \alpha_{ij} x_i\right). \tag{4.2}$$

Note that z_j can be seen as an expansion in terms of the basis functions x_i. In this way we can produce a new vector $\mathbf{z} = (z_1, \ldots, z_n)$, and this one can be used in

time as input to another function $g_k(\mathbf{z})$:

$$y_a = g_a(\mathbf{z}) = g_a\left(\sum_{j=1}^{n} \beta_{jk} z_j\right) = g_a\left(\sum_{j=1}^{n} \beta_{jk} f_j\left(\sum_{i=1}^{m} \alpha_{ij} x_i\right)\right). \tag{4.3}$$

This leads to the schematic representation of Figure 4.1. Notice that we could go on and add more layers[2] on top of previous ones using the sort of composition we just described. This is sometimes called a multilayer perceptron (MLP). The units in the middle of the network, i.e., the ones between inputs and outputs, are called hidden units because their values are not directly observed.

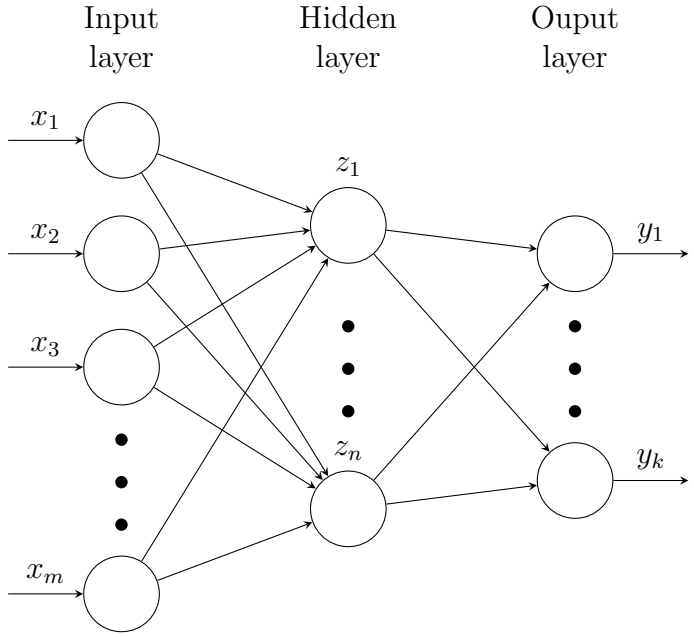

Figure 4.1: Schematic view of a 2-layer neural network.

Once the architecture of the network is set up, the parameters are determined via a maximum likelihood framework. This requires the evaluation of derivatives of the log likelihood function with respect to the network parameters. The good news is that there is a set of procedures that allow us to calculate the derivatives (and

[2]There are several uses for the word "layer" in the neural networks literature. In this work, a 2-layer network will mean that 2 sets of weights have to be calculated, i.e. there are two levels of units on top of the input.

hence the weight updates) in a modular way. The general idea is to feed data to the network to get predictions and then to **backpropagate** the error, layer by layer. The advantage of this approach is that a unit shares information only with those other units it has a connection with, hence making the process considerably simpler and parallel computing techniques suitable[5, 13].

4.3 Neural network complexity and representational power

The approximation capabilities of feedforward networks have been widely studied and found to be very general[7, 12]. Because of this, they are said to be **universal approximators**[3][5]. However, such representational power comes at a price: the high non-linearity of the system makes the objective function non-convex, therefore difficult to solve. Besides, the more intricate the architecture is, the higher risk of reaching a poor local minimum. The problem is even worse when more than two layers are considered because of poor weight updates in the deeper layers. This is called the "vanishing gradient problem" and we can think of it like this: as soon as a neural network becomes reasonably good at a task, the error propagated back to the deeper layers is so small that there is no clear way to help do the task better.

Regarding classification problems, another aspect we need to consider is that if neural networks are such good approximators, we also expect them to have a large $VC-$dimension. Indeed this is the case. Even though, the $VC-$dimension is finite, it is for instance, far larger than that of decision stumps.

The following theorem from Karpinski and Macintyre is the fundamental tool in providing an upper bound for feed forward networks with sigmoidal activation functions[1, 3].

Theorem 4.1. *Consider the parametrized class*

$$\mathcal{F} = \{\mathbf{x} \mapsto f(\theta, x) \colon \theta \in \mathbb{R}^d\}, \tag{4.4}$$

[3]This basically means that for every function f and $\epsilon > 0$, there is an architecture yielding a function approximator $\hat{f}$ such that f, such that $||f - \hat{f}|| \leq \epsilon$

for some $\{\pm 1\}$-valued function f. Suppose that, for each input $\mathbf{x} \in \mathbb{R}^n$, there is an algorithm that computes $f(\theta, \mathbf{x})$ and this computation takes no more than t operations of the following types:

- *the arithmetic operations $+, -, \times$, and $/$ on real numbers,*
- *jumps conditioned on $>$, $\geq$, $<$, $\leq$, 0, and $\neq$ comparisons of the real numbers, and*
- *the exponential function $\alpha \mapsto e^{\alpha}$.*

Let $V_{\mathcal{F}}$ be the VC-dimension of $\mathcal{F}$. Then $V_{\mathcal{F}} \leq t^2 d(d + 19 \log_2(9d))$.

Furthermore, if the t steps include no more than q in which the exponential function is evaluated, then

$$V_{\mathcal{F}} \leq (d(q+1))^2 + 11d(q+1)(t + log_2(9d(q+1))) \tag{4.5}$$

This immediately leads to estimate the upper bound for neural networks with sigmoidal activation in $O(w^4)$, where w is the number of weights. Unfortunately, this VC bound is particularly bad for large networks, just like the ones deep learning requires. There is another theorem for 2-layer networks that gives a tighter bound in the case of a discrete and restricted input domain[1]:

Theorem 4.2. *Consider a 2-layer feedforward network with input domain $X = \{-k, -k+1, \ldots, k\}^m$ (for $k \in \mathbb{N}$) and first-layer computation units, each with the standard sigmoid activation function (the output unit being a linear threshold unit). Let W be the total number of parameters in the network, and suppose that the number of inputs of each first-layer unit is no more than N. Then the class $\mathcal{F}$ of functions computed by this network has $V_{\mathcal{F}} \leq 2W \log_2(60Nk)$.*

4.4 Unsupervised pre-training

The difficulties mentioned at the end of section 4.2 steered the machine learning community away from neural networks for many years. Since multi-layered networks were hard to train and fine-tune, and single layer ones were performing at the same

level or worse than shallow architectures, there was no good reason to pursue this path.

That changed in 2006, when the optimization problems were empirically alleviated by the introduction of two new kinds of unsupervised learning algorithms[4, 14, 16, 22]. The first kind is based on deep generative models[4] composed of a stack of restricted Boltzmann machines (RBMs), while the other uses autoencoders as building blocks (both concepts are explained below). The general idea is to train a deep belief network (DBN) or a Stacked Auto Encoder (SAE) with the same topology as the desired neural network for an unsupervised pre-training stage (i.e. without using labels). Once the procedure is done, the resulting weights are used to initialize the multilayer perceptron. Surprisingly this leads to much better performance than if the weights were randomly initialized.

The feedforward networks that are initialized with weights obtained from a pre-training step using DBNs or SAEs are usually called **Deep Neural Networks** in the recent literature.

How and why this kind of pre-training helps with classification problems is analyzed from two perspectives:

- The optimization point of view: "generative models trained in an unsupervised fashion can provide excellent initialization points in highly nonlinear parameter estimation problems"[8].

- The regularization perspective: unsupervised learning provides a prior on the set of functions representable by the model.

For an excellent discussion of these perspectives see the work of Erhan et al. [10].

4.4.1 Restricted Boltzmann Machines (RBM)

A Restricted Boltzmann Machine is a bipartite graph of visible and hidden units where every node is connected to each node in the other class.

[4] A generative model strives to learn the distribution of data whereas discriminative models learn the boundaries between classes.

Inspired by statistical physics, the idea is to model the probability distribution $p(\mathbf{v}, \mathbf{h})$ in terms of an energy function $E(\mathbf{v}, \mathbf{h})$. We will consider now such function to be of the form

$$E(\mathbf{v}, \mathbf{h}) = \sum_{i=1}^{V} v_i b_i^v + \sum_{h=1}^{H} h_j b_j^h + \sum_{i,j} v_i h_j w_{ij} \tag{4.6}$$

where $\mathbf{v} = (v_1, \ldots, v_V)$ and $\mathbf{h} = (h_1, \ldots, h_V)$ are binary state vectors for the visible and hidden units, respectively. w_{ij} is the real-valued weight between units v_i and h_j. The terms b_i^v and b_j^h correspond to biases into units i and j.

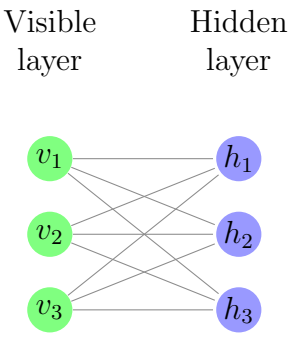

Figure 4.2: Pictorial view of a Restricted Boltzmann machine. Biases are not included to avoid cluttering. Note it is an undirected graph, and that there are no in-layer connections.

The probability associated with this distribution is given by:

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{4.7}$$

where Z is the partition function, namely,

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{4.8}$$

Looking at eq. 4.6 it is intuitively clear that low energy configurations will have a high probability while those with high energy are assigned low probabilities.

This joint probability distribution also makes it easy to find marginal probabilities:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) \tag{4.9}$$

$$= \frac{\sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})}{Z}. \tag{4.10}$$

Similarly,

$$p(\mathbf{h}) = \sum_{\mathbf{v}} p(\mathbf{v}, \mathbf{h}) \tag{4.11}$$

$$= \frac{\sum_{\mathbf{v}} p(\mathbf{v}, \mathbf{h})}{Z}. \tag{4.12}$$

Conditional probabilities have also simple expressions:

$$p(\mathbf{v}|\mathbf{h}) = \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{h})} \tag{4.13}$$

$$= \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u}} e^{-E(\mathbf{u},\mathbf{h})}} \tag{4.14}$$

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{v})} \tag{4.15}$$

$$= \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{g}} e^{-E(\mathbf{v},\mathbf{g})}} \tag{4.16}$$

Lengthy calculations also provide us with two additional formulas:

$$p(v_k = 1|\mathbf{h}) = \frac{1}{1 + e^{-(\sum_{j=1}^{H} h_j w_{kj} + b_k^v)}} \tag{4.17}$$

$$p(h_k = 1|\mathbf{v}) = \frac{1}{1 + e^{-(\sum_{i=1}^{V} v_i w_{ki} + b_k^h)}} \tag{4.18}$$

Notice the symmetry of the probability equations. There is no distinction in the roles played by visible and hidden units, and therefore the difference lies in interpretation.

Training RBMs

Given a set of observations $S = \{\mathbf{v}^1, \ldots, \mathbf{v}^n\}$, the goal is to maximize the log probability using the model's distribution given in eq. 4.7:

$$\sum_{i=1}^{n} \ln p(\mathbf{v}^i) = \sum_{i=1}^{n} \ln \left(\frac{e^{-E(\mathbf{v}^i, \mathbf{h})}}{Z} \right) \tag{4.19}$$

Differentiating with respect to an arbitrary weight w_{ij} yields

$$\frac{\partial}{\partial w_{ij}} \sum_{k=1}^{n} \ln p(\mathbf{v}^k) = -\sum_{k=1}^{n} \frac{\sum_h e^{-E(\mathbf{v}^k, \mathbf{h})} v_i^k h_j}{\sum_{\mathbf{h}} e^{-E(\mathbf{v}^k, \mathbf{h})}} + \sum_{k=1}^{n} \frac{\sum_{\mathbf{u},\mathbf{h}} e^{-E(\mathbf{u}, \mathbf{h})} u_i h_j}{Z} \tag{4.20}$$

The first term on the right of eq. 4.20 is the expected value of $v_i^k h_j$ fixing $\mathbf{v} = \mathbf{v}^k$. The second term, on the other hand, is the expected value of $u_i h_j$ under the model's distribution. This is not easy to calculate and the usual way to proceed is to use Gibbs sampling. However, it has been shown that a procedure known as Contrastive Divergence is a good approximation[15].

Similar equations can be derived for the bias terms and using contrastive divergence update rules for gradient descent are obtained [15]:

$$\Delta w_{ij} = \epsilon_w (\mathrm{E}_{data}[v_i h_j] - \mathrm{E}_{model}[v_i h_j]) \tag{4.21}$$

$$\Delta b_i^v = \epsilon_b^v (\mathrm{E}_{data}[v_i] - \mathrm{E}_{model}[v_i]) \tag{4.22}$$

$$\Delta b_j^h = \epsilon_b^h (\mathrm{E}_{data}[h_i] - \mathrm{E}_{model}[h_j]) \tag{4.23}$$

The RBM model presented here was restricted to binary input values. It can be modified to allow for other types of variables by considering a different energy function.

4.4.2 Auto-encoders

A closer analysis of RBMs will reveal that the algorithms they need can become computationally intensive when the number of units is large. Take for instance, the

partition function Z in eq. 4.8. Since it is a sum over all possible visible and hidden configurations, it grows exponentially in the number of units.

A cheaper alternative at the expense of giving up the probabilistic interpretation is to use auto-encoders. The idea behind them is to train a 2-layer feedforward network to learn a function $f(\mathbf{x})$ such that $f(\mathbf{x}) = \mathbf{x}$. This means that we want to minimize a loss function of the form

$$Loss(f) = \sum_{i=1}^{m} ||\mathbf{x}_i - f(\mathbf{x}_i)||^2 \tag{4.24}$$

Now, we do not want the network to learn an identity function since that would give us no additional information to take advantage of. We want the system **to learn something new** about the data. Therefore, usually three types of modifications are applied to enforce constraints that avoid trivial learning:

1. **Reduced number of hidden units:** The number of units in the hidden layer is smaller than the number of units in the visible layer. From the interpretation of hidden units as new features, we can think of this restriction as forcing the system to code the information in a more efficient way (i.e. using less variables).

2. **Denoising encoding:** The idea here is to introduce some random noise, usually turning off some input signal, but still requiring the system to learn the original vector. Note that this does not need the number of inputs to be greater than the number of hidden units, for the noise alone should be enough to guarantee that the identity function is not a good fit.

3. **Increased number of hidden units:** In this case the number of hidden units is larger than the number of input units in an effort to produce sparse coding, that is to say that a small groups of neurons are responsible almost entirely (via significantly larger weight norms compared to the rest) for every output. Since this kind of network has such a big hidden layer, it could just choose the identity function and set all other weights to zero. To prevent this, the method is usually used in conjunction with denoising encoding.

Why to choose auto-encoders over RBMs? Since it is only a two layer architecture, it can be done efficiently using backpropagation techniques. Note that unlike

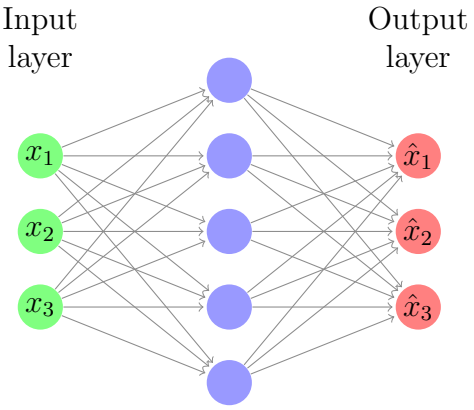

Figure 4.3: Denoising autoencoder

RBMs, auto-encoder are deterministic machines: they give us a value $\hat{x}$ and not $p(\hat{x})$. Because of this they cannot be used to form deep generative models. However, they can still be stacked and trained in a greedy way layer by layer.

Chapter 5

Model Selection Experiments

5.1 General Procedure

The goal of our experiments is somewhat different to the typical situation where one wants to choose a model. What we want is to test the performance of some model selection methods on two very different architectures of classifiers: AdaBoost classifiers using decision stumps as weak learners and Deep Neural Networks (DNN) using auto-enconders. Furthermore, we are mainly interested in the case of limited sample data; something that matches most real life applications. Because of this, we want to present the learning algorithms with a relatively small data set[1] from a **known** distribution and then we are going to apply the different criteria we considered.

Performance is judged by two aspects:

1. The ability of the method to choose the right architecture, that is to find the Bayes classifier, which is always included among the models being tested.

2. The ability to produce a correct and tight bound for the generalization error.

The general procedure for testing, which is repeated many times, includes the following steps:

[1] In general, the size of a data set alone is not a good indicator of learnability. It is actually better to consider the ratio of the number of samples, m, to the input space dimension, d. The larger m/d is, the better chance of effective learning.

1. For each family of learners $\mathcal{H} = \{h_\theta\}$ considered, fix a classifier by choosing a set of parameters given by $\theta \in \mathbb{R}^n$.

2. Get a sample set $X = (\mathbf{x_1}, \ldots, \mathbf{x}_m)$ and calculate $y_i = h_\theta(\mathbf{x}_i)$. Let $S_m = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.

3. Split the set S_m into two disjoint sets $S_T = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_k}, y_k)\}$ and $S_V = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_l}, y_l)\}$ with $m = k + l$.

4. Feed the data set S_T to the corresponding learning algorithm. Run the calculations necessary for all criteria.

5. Select the best classifier according to every criterion.

6. Estimate generalization error by calculating classification error on the set S_V.

7. For the best classifiers according to every criterion, calculate the relative error with respect to the best classifier based on generalization error estimations.

5.1.1 Building an oracle: model and sample generation

Since we require to know the right model before testing, real life data sets are not a good option here. We have to come up with a way of creating data, to have an oracle, so to speak.

The preparation of the data the algorithms will learn from is no trivial matter. When choosing model parameters θ and the input samples X, it is important to realize that merely picking numbers at random could lead to "concepts" that can be very easy to learn (and therefore not suitable for benchmarking) or very intricate, even pathological cases where the algorithm fails, but that are not statistically significant in practice. First of all, the input data, namely the elements of set X must follow the same distribution as the elements we expect the learner will find in the future. This is a fundamental hypothesis underlying learning theory. Therefore it is recommended to choose X samples generated by some sort of pattern or distribution. Regarding labels, the easiest way to avoid difficulties is to produce them through a function having the same form as the classifier. Note that at the learning stage if

the right architecture is chosen, the learning procedure goal will be to recover the parameters that were used to generate such distribution of data.

An oracle for AdaBoost

For the AdaBoost family of classifiers, X was obtained in the following way: First, $11,000$ random vectors in $\mathbf{R}^{12}$ were generated such that every coordinate $x_i \sim \mathcal{N}(0,1)$. Then, the vectors were separated in 4 different groups. To every group, a different constant vector where every coordinate was $\pm 1/2$ was added. This forms 4 clouds of data in $\mathbf{R}^{12}$, it is therefore reasonable to believe a two-class classification rule can be found.

According to eq. 3.1, to completely define an AdaBoost classifier the number of weak classifiers T, and the parameters and weights for each one of them have to be specified. In our case, the number of weak classifiers was chosen first at random from a uniform distribution on the integers from 10 to 100. After that, a positive weight was assigned randomly to every classifier and they were then normalized. Finally, since the base classifiers were chosen to be decision stumps, the parameters for each one of them (coordinate, threshold, and direction) were taken randomly from uniform distributions.

Once the parameters were set, labels for each entry vector were obtained using eq. 3.1. At this stage, it is necessary to ask ourselves what a good result is. If there is a considerably larger proportion (more than 80%) of data points assigned to one label l, then there is a trivial classification rule ($f_t(\mathbf{x}) = l$) with a good performance. In this case, even though there still some learning to do, the classifier might be able to improve a lot its predictions by improving a little on those misclassified samples. This suggests that a small number of base classifiers may be enough to boost classification. This means that this distribution favors models with a small number of weak learners.

It is therefore desirable to have models yielding both labels in roughly the same proportions. Indeed, experimentally it was seen that this was the case for the proposed generation scheme.

An oracle for Deep Neural Networks

A different procedure was used for generating the samples of the neural network models. Since the premise of deep learning is to learn different levels of representations, e.g., going from pixels to curves, and from curves to figures; it is not so straightforward to create synthetic data from scratch in a way that actually matches the strengths of hierarchical models. Because of this, a real life data set (MNIST) was used as a base for building a model. MNIST is a data set of handwritten numbers. Each number is represented by a 28×28 pixel frame, therefore every digit can be considered a 784-feature input vector. The database consists of 60,000 training samples and 10,000 test samples[2]. Different data sets were drawn from the training database by selecting all 45 possible combinations of two digits. Then, for every set, a number of hidden layers ($l \in \{1, 2\}$) was chosen, and a number of neurons for the first ($n_1 \in \{80, 100, 120\}$) and second ($n_2 \in \{8, 10, 12\}$) layers were also randomly selected.

It was experimentally verified that the errors on the corresponding test sets were generally below 5%. This suggests that the networks actually learned a concept close to the numbers given, and hence had undergone considerable learning. Because of this, the weights found, along with the number of layers and units per layer were taken to be the real model, and a new label y_i' was assigned to every input vector $\mathbf{x}_i$ using the network to produce it. New labels were also generated for the test set. The new sets were then used in the model selection experiments.

5.1.2 Considered models

For AdaBoost one must select the weak learners and how many of them are to be included in the final classifier. For all tests decision stumps were selected as the family of weak classifiers and the number of rounds tested was chosen to be in the range of 1/4 of the real model to 2 times, namely, $T \in \{\frac{k}{4}T^* : k = 1, 2, \ldots, 8\}$ where T^* is the number of weak learners in the Bayes classifier. Every problem had a data set of $11,000$ points, and 600 of them were used for training, while rest was reserved

[2]There is a distinction between training and test samples because the original data was augmented by applying different transformations to some samples, and it is considered that the data points in the test sample are actually harder to classify.

for validation purposes.

In the case of deep neural networks the twelve possible combinations of one and two layers using the same number of neurons that the oracle could chose from were tested. This ensures that the architecture of the Bayes classifier was included. Since there is roughly the same amount of samples for each digit in the original MNIST database, when the data corresponding to two different digits was drawn from it, the data set obtained would have a size of about $12,000$ samples. These were used for the training stage. The test samples were taken from the original test set and they were about $2,000$ in each experiment.

5.2 Adjusted SRM

Even though the bounds from section 2.2 have important theoretical value, they are not very good in practical situations where the number of training samples available is small. Tables 5.1 and 5.2 show the calculations of the corresponding structural risk bounds for AdaBoost and Deep Neural Networks.

For AdaBoost, the bound was calculated using eq. 3.4 to estimate the VC-dimension of the base classifiers by exhaustion. Note that this is the tightest bound on the complexity of the decision stumps we had, therefore, better results were expected. Since the generalization error should be in the $0-100\%$ range, any number above 1 is useless. We see that the bound increases with complexity (i.e. number of weak learners or rounds) and decreases with the number of samples used for training as expected. The problem is that for the range considered, the complexity term (the only one included in table 5.1 since training error was taken to be zero), will always outweigh the error term and hence SRM would always pick the least complex model, regardless of the empirical error.

The large penalty on the complexity has rendered this criterion useless. One could argue that the number of samples is insufficient to draw reasonable conclusions, but the problem is results are already very good for AdaBoost ($|S_T| = 600$, $|S_V| = 10400$) and SAE ($|S_T| \approx 12000$, $|S_V| \approx 2000$), having mean prediction errors of % and % over 100 and 40 different models, respectively.

Let us take a step back for a moment. In what follows we tacitly assume that

Table 5.1: Calculations of the SRM bound for AdaBoost from eq. 3.11 when the empirical error is zero and stumps are the weak learners. There are 12 features and $\delta = 0.05$.

T \ m	600	6000	60000
10	4.3671	1.6999	0.6223
50	9.4859	3.7297	1.3720
100	13.2664	5.2368	1.9300

Table 5.2: Calculations of the SRM bound for Neural Networks using a modified estimate for the VC dimension from Thm. 4.2. Every vector has 784 features and $\delta = 0.05$.

Neurons \ m	6000	12000	24000	48000
80	1.2723	0.7384	0.4204	0.2358
100	1.5199	0.8893	0.5093	0.2870
120	1.7533	1.0332	0.5949	0.3366

every inequality holds with probability $1 - \delta$. Remember eq. 2.20, we could rewrite it as

$$P(\exists h \in \mathcal{H} \colon L_m(h) \geq \hat{L}_m + \varepsilon) \leq A\Pi_{\mathcal{H}}(m)e^{-m\varepsilon^2/B} \tag{5.1}$$

where $A = 8$ and $B = 32$. Hence we also have,

$$L_m \leq \hat{L}_m + \sqrt{\frac{B[\ln(A/\delta) + \ln \Pi_{\mathcal{H}}(m)]}{m}} \tag{5.2}$$

This makes us wonder if we could find a different combination of A and B such that eq. 5.1 still holds but achieves a tighter bound[3]. It is not straightforward to come up with a new inequality, but we could at least try to adjust the criterion by introducing a new constant C, we then get

$$L_m \leq \hat{L}_m + C\sqrt{\frac{B[\ln(A/\delta) + \ln \Pi_{\mathcal{H}}(m)]}{m}} \tag{5.3}$$

[3]In fact there are another combinations derived analytically that yield slightly better bounds[6, 13, 27], but their deductions are more complicated and hence the author decided to neglect them since an adjustment factor was going to be introduced anyway.

This change amounts to modifying the value of B. Since A is being held fixed, it is unlikely that eq. 5.3 will hold universally, but it is somewhat reasonable to think that it will at least work for some restricted case. Bearing this in mind, it is possible to find a suitable value for C via cross validation.

In this situation holdout cross validation was performed. Independent tests were carried out for both AdaBoost and DNN. In each case, a collection of classifiers was chosen so that all of them had different VC-dimensions. Training and validation sets were fixed. The criterion from eq. 5.3 was calculated using the regular values for A and B and taking $C \in \{1, \frac{1}{2}, \ldots, \frac{1}{2^{10}}\}$. For every possible value of C, the configuration yielding the lowest bound was selected to be part of a new set $\mathcal{F}$. Finally, the validation error was calculated for all of the classifiers in $\mathcal{F}$. The one with the smallest error indicates an adequate value for C. In both cases several values of C chose the same model and this in turn had the best performance. One of the smallest values, $C = \frac{1}{2^9}$ was selected since it produced bounds reasonably close to the validation error and arguing that indeed the complexity penalty was too large and had to be attenuated.

5.3 10-Fold Cross Validation

From the realm of cross validation a 10-fold configuration was chosen. $k = 10$ as is customary. Note that a lower k gives a higher variance on the choice of training and test samples. On the other hand, a larger k means that we have more testing sets and this intuitively gives us a higher confidence in the estimate of the generalization error derived from this criterion. One has to choose a number of cells that gives some balance between these two perspectives.

The great advantage of this method is that it is distribution-free, therefore it was not necessary to incorporate in the calculations any information about the actual structure of the family of classifiers being considered. The problem is that it is also the most computationally intensive method from all considered. At the number of samples considered for both AdaBoost and NN, this procedure took roughly 10 times the amount of time required for training.

5.4 Model-dependent methods

Even though we would like to find a universal method for model selection, it is also interesting to consider methods that are devised for a specific family of classifiers. The use of such methods is justified if they consistently have better performance than more general methods such as CV and SRM.

5.4.1 AdaBoost-specific Model Selection

The AdaBoost-tailored method considered is the one based on the margin approach from section 3.4. It is important to notice that these bounds are derived from the original VC-dimension-based bound, and hence they could still be too big if applied to small size data sets. Indeed this is the case. Using the lower bound on the value of θ from theorem 3.5 for $m = 600$ and $\mathbf{x} \in \mathbb{R}^{12}$, the lowest possible value is $\theta = 0.7133$. Considering that normalized margins range from -1 to 1, this is a rather strong condition and therefore we do not expect to find accurate estimates of the generalization error (because most of the sample margins will be below this threshold), but only a suggested model. Another alternative would be to adjust the criterion in a similar way to that of SRM. However, this alternative was quickly discarded because in this case both terms depend on the margin threshold and the introduction of a multiplying constant does not translate into a tighter bound in a straightforward way.

5.4.2 DNN-specific Model Selection

In 2010 Arnold et al.[2] proposed a model selection method based on deep learning techniques. The idea is to choose the number of neurons layer by layer, and not for all of them at once. This is the usual approach for neural networks. However, traditional techniques focus on improving prediction error, whereas Arnold's method aims to minimize the reconstruction error in a greedy way. This means that for every layer j one should start with a small number of neurons and add more as long as it translates in a reduction in the value of the function

$$\mathcal{R}_j = \sum_{i=1}^{m} ||\mathbf{z} - \Phi_j(\mathbf{z})||^2 \tag{5.4}$$

where Φ_j is the forward propagation through layer j and $\mathbf{z}$ is the input vector to it.

The method was tested by their authors for the case of RBM-based DNN. In this work the same technique for stacked auto-encoders is explored. An important observation is that reconstruction error is precisely the loss function for auto-encoder training, therefore this criterion requires in principle no additional computations. Nevertheless, recall also that the optimization is done by a backpropagation algorithm and results can vary significantly because of random initialization. Because of this, it was decided to run the algorithm for 30 epochs[4], using training batches of size a 100 in an effort to smooth results. Additionally, the procedure was repeated in a 10-fold cross validation scheme.

Because the goal was to compare the performance of this selection criterion to the general methods, instead of adding one neuron at a time, only the number of neurons that were possible choices for the oracle were allowed, namely, $n_1 \in \{80, 100, 120\}$ and $n_2 \in \{8, 10, 12\}$. Besides, two classifiers were selected for each problem considered, one with a 1-layer configuration and another with a 2-layer configuration. This was done because the process suggests a number of neurons by layer but not a number of layers. From the classifiers with smallest overall loss a model is finally chosen by considering the generalization error.

5.5 Results

5.5.1 Classifier selection

Following Hastie et al.[13], the results of the experiments are presented in boxplots showing the distribution of the error in using the chosen model relative to the best model, namely,

[4]In the context of neural networks an epoch is an iteration over the complete training set.

$$100 \times \frac{\mathrm{Err}_{\mathcal{D}}(\hat{\theta}) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)}{\max_\theta \mathrm{Err}_{\mathcal{D}}(\theta) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)} \tag{5.5}$$

where $\mathrm{Err}_{\mathcal{D}}(\theta)$ is the prediction error over an independent test sample:

$$\mathrm{Err}_{\mathcal{D}}(\theta) = \mathrm{E}[L_m(f_\theta)|\mathcal{D}]. \tag{5.6}$$

Figure 5.1 presents the results for the AdaBoost experiments. First of all, notice that the margins method outlined in section 3.4 performs in a extremely poor way. This was not unexpected, though. Because of the large threshold value, most margins fall beneath and hence the method is forced to choose the simplest model (i.e. the one with smallest VC-dimension) and since this is a classifier with only $\lfloor T/4 \rfloor$ weak learners, it is no surprise that it is insufficient to explain the data. The experiments therefore suggest that this method should be discarded for small data set sizes.

The situation is quite different for structural risk minimization. Even though it has larger mean and variance than cross validation, the method is not so far from it, which turned out to be the best selection method (among the ones tested) for AdaBoost. It is important to note that if a correction factor had not been used to adjust SRM, the results would have been very similar to those of the margins method because an extremely large complexity penalty excessively favors simple models. Perhaps a finer search could improve the results even further.

Figure 5.2 shows the results for deep neural networks. For this type of classifiers cross validation also outperformed SRM, but in this case the former behaved in roughly the same way as it did for AdaBoost, while the latter now concentrates more samples in the middle of the distribution. In general we could say that results got worse for both methods, making clear that this was a tougher problem.

The layer-by-layer reconstruction method has achieved the best performance of all three methods considered. This comes as a surprise, since stacked auto-encoders training results vary a lot depending on the initialization. This suggests that the mechanisms put in place to smooth out such variations worked. Such behavior makes this technique a viable candidate for model selection in deep neural network architectures, since both results and computational complexity are roughly the same for CV and layer-by-layer reconstruction.

5.5.2 How big is the selected model?

We should now ask ourselves how big a model must be for optimum performance and how this size relates to the size of the Bayes classifier in a situation with small data sets. Due to the consistency theorem of SRM and the relation of CV to Akaike's criterion, we expect that for large data sets the rule for choosing the classifier converges to f^*, but how much does the architecture of f_m resembles that of f^* when m is small? In an attempt to answer this question, let us see what kind of classifier is each method selecting. Figure 5.3 displays the frequencies of selection of each possible number of weak classifiers for AdaBoost. The bin in the middle always represents T^*, the number of weak learners in the Bayes classifier.

We see that for 600 data points in the training set, the best model has most of the time twice as many weak classifiers. Adjusted SRM and cross validation also prefer a high number of learners.

Now consider the situation for deep neural networks, illustrated in table 5.3. In this case, the selected models are above or below the ideal complexity in roughly the same proportions for all methods. It is interesting to see that Adjusted SRM seems to have a strong preference for smaller models even when the complexity term has been greatly attenuated.

Table 5.3: Classifier distribution by complexity in the DNN experiments.

	$V_{\mathcal{H}} < V_{\mathcal{H}^*}$	$V_{\mathcal{H}} = V_{\mathcal{H}^*}$	$V_{\mathcal{H}} > V_{\mathcal{H}^*}$
Best	19	6	15
Adjusted SRM	16	6	18
CV	21	8	11
Layer-by-layer R.	23	7	10

5.5.3 Conclusion

According to the results, the best off-the-shelf method for model selection in the case of AdaBoost and DNN with a reduced number of samples for learning is cross validation, as it gives robust results and does not require any knowledge of the system. Nonetheless, Adjusted SRM seems to be a great option outperforming the regular

version and achieving similar results to those of CV in a way that is considerably less expensive because it doesn't require as many calculations as CV does.

It is reasonable to believe that the results of SRM could be further improved if the parameter A and B were calibrated simultaneously. The problem with this bounds is that without an analytical justification we cannot expect that they will work in a large variety of situations and therefore it is necessary to recalculate every time they want to be used in a new setting. In this case the constant was the same for both kinds of learning algorithms, but this was just incidental, since we have to take into account that the VC-dimension of the DNN had to be estimated in a way that would allow the criterion to be applied.

To find an alternative, unsupervised method for model selection also gives great promise for further development of such techniques.

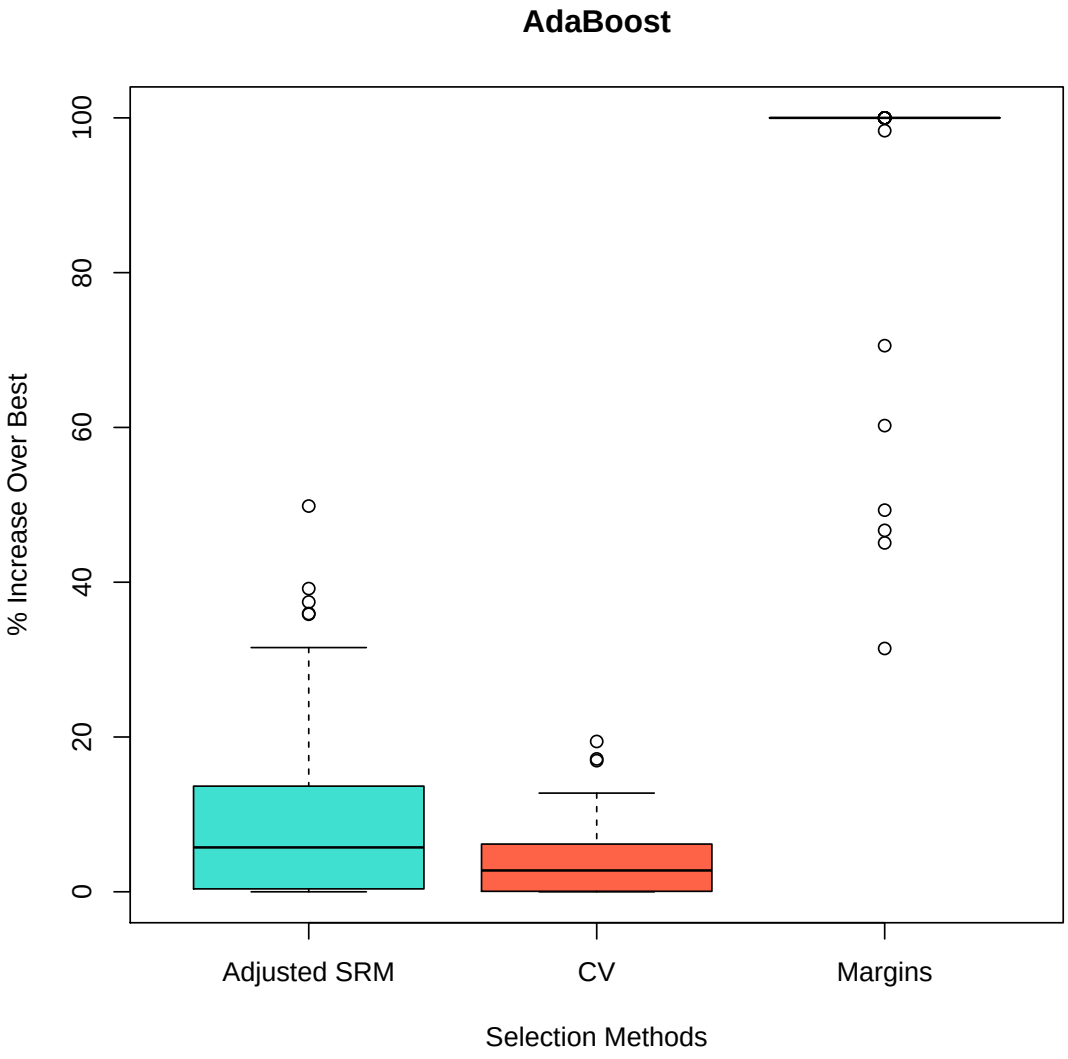

Figure 5.1: Distribution of the relative error in using a chosen model relative to the best model, i.e., $100 \times [\mathrm{Err}_{\mathcal{D}}(\hat{\theta}) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)]/[\max_\theta \mathrm{Err}_{\mathcal{D}}(\theta) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)]$ for AdaBoost. There are 100 training sets, each of size 600 represented in each boxplot, with errors computed on test sets of size $10,400$.

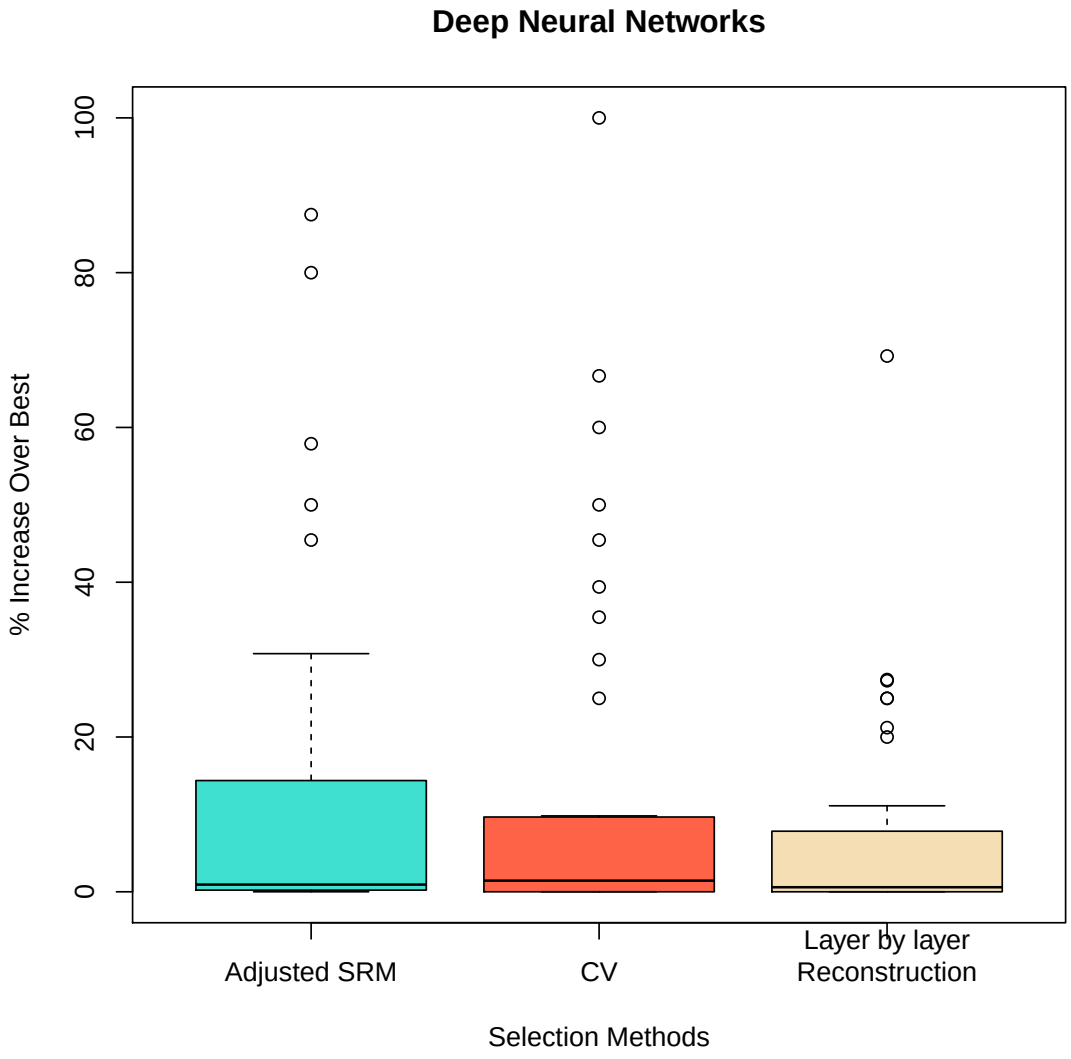

Figure 5.2: Distribution of the relative error in using a chosen model relative to the best model, i.e., $100 \times [\mathrm{Err}_{\mathcal{D}}(\hat{\theta}) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)]/[\max_\theta \mathrm{Err}_{\mathcal{D}}(\theta) - \min_\theta \mathrm{Err}_{\mathcal{D}}(\theta)]$ for DNN. There are 40 training sets, each with a size of approximately $12,000$ represented in each boxplot, with errors computed on test sets of size $2,000$.

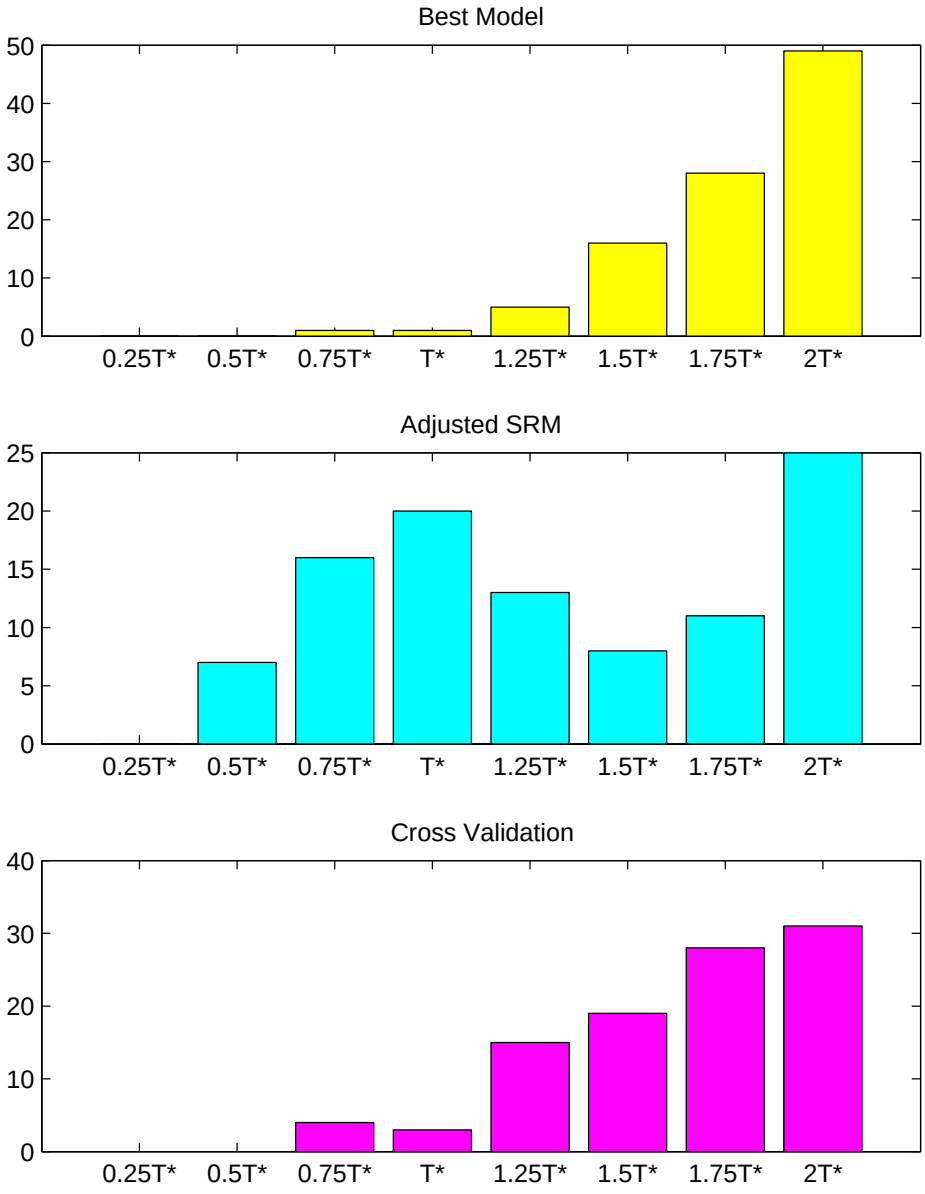

Figure 5.3: Frequency of architectures for the best model, the model selected by SRM, and the one selected by cross validation.

Bibliography

[1] Martin Anthony and Peter L. Bartlett. *Neural Network Learning - Theoretical Foundations*. Cambridge University Press, 2002. ISBN 978-0-521-57353-5.

[2] Ludovic Arnold, Hélène Paugam-Moisy, and Michèle Sebag. Unsupervised layer-wise model selection in deep neural networks. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 915–920. IOS Press, 2010. ISBN 978-1-60750-605-8.

[3] Peter L. Bartlett and Wolfgang Maass. Vapnik-chervonenkis dimension of neural nets.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, Université De Montréal, and Montréal Québec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[6] Vladimir S. Cherkassky and Filip Mulier. *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998. ISBN 0471154938.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[8] Li Deng and Dong Yu. *Deep Learning: Methods and Applications*. NOW Publishers.

[9] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Applications of mathematics : stochastic modelling and applied probability. U.S. Government Printing Office, 1996. ISBN 9780387946184. URL http://books.google.com.co/books?id=uDgXoRkyWqQC.

[10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, March 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1756025.

[11] Yoav Freund and Robert 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. ISSN 0022-0000. doi: http://dx.doi.org/10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X.

[12] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Netw.*, 2(3):183–192, May 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90003-8. URL http://dx.doi.org/10.1016/0893-6080(89)90003-8.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2009. ISBN 9780387848587. URL http://books.google.com.co/books?id=tVIjmNS3Ob8C.

[14] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006. doi: 10.1126/science.1127647. URL http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google.

[15] Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors,

Neural Networks: Tricks of the Trade (2nd ed.), volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012. ISBN 978-3-642-35288-1.

[16] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[17] P. Lahiri. *Model Selection*. Institute of Mathematical Statistics Lecture Notes Monograph. Institute of Mathematical Statistics, 2001. ISBN 9780940600522. URL http://books.google.com.co/books?id=CAeeYjuJv2oC.

[18] Yann LeCun and Marc'Aurelio Ranzato. Deep learning tutorial, June 2013.

[19] Fernando Lozano. Aprendizaje supervisado, January 2012.

[20] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002. ISBN 0521642981.

[21] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[22] Marc'Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 1137–1144. MIT Press, 2006. ISBN 0-262-19568-2.

[23] R.E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. Adaptive computation and machine learning. MIT Press, 2012. ISBN 9780262017183. URL http://books.google.com.co/books?id=blSReLACtToC.

[24] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978. ISSN 00905364. doi: 10.2307/2958889. URL http://dx.doi.org/10.2307/2958889.

[25] Jun Shao. Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association*, 88(422):486–494, 1993. ISSN 01621459. doi: 10.2307/2290328. URL http://dx.doi.org/10.2307/2290328.

[26] M Stone. An asymptotic equivalence of choice of model by cross-validation and akaike's criterion. *Journal of the Royal Statistical Society Series B*, (39):44–47, 1977.

[27] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.