Properties of the Hypothesis Space and their Effect on Machine Learning

Ruben Dulek

March 26, 2013

Abstract

The best method to use for machine learning depends on the problem. This thesis considers one aspect of machine learning problems: How do the properties of the hypothesis space affect machine learning? It collects academic advances on how dimensionality and representational capacity of the space and the presence of local optima affect machine learning. Useful additions to generic machine learning methods are listed that deal with these properties. The result is a collective overview on how to design a machine learning process that uses these properties of the hypothesis space.

1 Introduction

Machine learning is used everywhere. From driving cars to playing Stratego, machine learning is applied in a huge variety of settings. These settings have vastly different problems. As follows from the *No-Free-Lunch* theorem, no solution works for all problems [59]. Many solutions have emerged that improve the learning efficiency for problems with specific properties.

In a machine learning setting, the problem to solve can usually be generalised to searching a space of hypothetical solutions for the best solution [46] [47]. This space is called the *hypothesis space.* This space defines many properties of the machine learning problem. Are the best solutions close together? How many dimensions does a hypothesis have? Is the best solution even in this space? These properties immensely affect the performance of machine learning techniques. A hill climber, for instance, will get stuck in the first local optimum if the global optimum is not near, and a neural network requires a large number of nodes for high-dimensional problems. A great amount of research has been done on their effects, and on adapting machine learning procedures to work with these properties, such as the works on feature extraction [16] [33], global optimisation [41], and ensemble techniques [18]. This research is fragmented, however, into topics that cover only one of these properties at a time.

It is the purpose of this thesis to give an overview of the important properties of the hypothesis space and the work that has been done to exploit them. Thus is the scope of this document defined:

> In what ways do the properties of a hypothesis space affect machine learning procedures, and how does one work with these properties?

Three properties of the hypothesis space are discussed: dimensionality, local optima and representational capacity. There are other relevant properties, such as a possible hierarchic structure [24] or lattice structure [38]. The three discussed here however are relevant to nearly all machine learning problems.

General improvements are discussed that can be made to work with any machine learning procedure. This leaves the choice of which procedure(s) to use to the designers of their application. After all, the choice of procedure should fit the problem as well as its hypothesis space. To illustrate the topics in this document, only continuous hypothesis spaces are discussed. While the discrete case is a more common scope for research, a continuous space lends itself better for visualisation, and is in general easier to understand. For further illustration, hill climbers and neural networks are used as examples.

1.1 Structure of this Document

Before discussing recent work on how to work with the properties of the hypothesis space, it is important to describe the properties themselves in detail. This allows for an understanding of what effects on the hypothesis space are addressed by the other sections. Section 2 describes the three properties: dimensionality, local optima and representational capacity. The effect of the properties on machine learning is described and some relevant solutions are introduced that will be discussed later in the document.

Sections 3 and 4 follow a structure that reflects the actual process of a machine learner.

In Section 3, a few steps are discussed that could be taken to prepare the machine learner before the actual process. Transforming with the covariance matrix (Section 3.1.1) and feature extraction (Section 3.1.2) are two ways to deal with the dimensionality of a hypothesis space. Choosing starting point(s) (Section 3.2) is a third step that deals with the problem of local optima.

In Section 4, improvements to the machine learning process itself are discussed. These may require more work to implement. Firstly, in Section 4.1, the problem of local optima is discussed in the setting of active learning. Next, two extrema of representational capacity are discussed: in Section 4.2

the low end with an improvement to the speed of convergence, and in Section 4.3 the high end where the optimal hypothesis may not be within the hypothesis space.

In Section 5, some topics of research are enumerated that deal with the properties of a hypothesis space, but have yet seen little attention in science. Finally, in Section 6, a brief summary of the discussed work is given that can be considered a reasonably comprehensive answer to the research question.

2 Properties of the Hypothesis Space

In this section, three important features of a hypothesis space are described that affect machine learning, and a brief description of their effects. Some common terms and problems need to be introduced before delving deeper into solutions for those problems in Sections 3 and 4.

2.1 Dimensionality

Every hypothesis has a number of dimensions that bring it together. These are the variables the machine learner adjusts to search the best hypothesis in the space, or the parameters of the function. Searching in hypothesis spaces with lots of dimensions is hard. This is due to the curse of dimensionality: The effective size of the hypothesis space increases exponentially with the number of dimensions [7].

In continuous hypothesis spaces, this leads to the *Hughes Phenomenon*: The accuracy of the resulting hypothesis is substantially reduced as the dimensionality increases and the number of training examples is kept constant [27]. As an example, it was also shown that the number of training examples required to train a PAC-learner (Probably Approximately Correct learners are a category of machine learners) equally well scales exponentially with the number of dimensions [39]. The cause seems to be that training examples have a local effect on the resulting objective function; machine learners are more certain of their guess of the objective function close to the training examples. As the dimensionality of the objective function increases, and thereby also that of the required hypothesis space to represent it, the Euclidian distance between the training examples also increases.

To work with a high-dimensional hypothesis space, more training examples have to be acquired, or some dimensions must be removed. The first option is beyond the scope of this work. It is often fairly hard or expensive, and moreover, the number of training examples required scales exponentially with the dimensionality. The second option, removing dimensions, is discussed in Section 3.1.2. First though a third option is discussed: One can also reduce the effect of the Hughes Phenomenon directly by scaling the hypothesis space such that the training data is better distributed across the dimensions. This is discussed in Section 3.1.1.

2.2 Local Optima and Basin of Attraction

It is impossible to search through all possible hypotheses in a continuous hypothesis space, and often infeasible to do so even if the hypothesis space is discrete. Instead, all machine learning algorithms introduce a *bias*: A heuristic to search by [47]. This bias compels them, for instance, to search for hypotheses that are similar to the best ones found so far. A common choice is to search in their local Euclidian area. The hill climber, for instance, takes one hypothesis and searches in its neighbourhood for one that is better until no better hypothesis can be found. This is based on the assumption that the best hypothesis is in the neighbourhood of other good hypotheses.

In many nonlinear optimisation problems, there may be a large number of local optima. These are hypotheses that are good, but not near to the globally best solution. A machine learning procedure may find one of these local optima, find no better solutions nearby, and return it as the answer, even though a better answer exists. Which local optimum is found sometimes depends on the starting state of the machine learning procedure, since that is where it starts searching. The set of starting states that ultimately results in finding a local optimum is called the *basin of attraction* of that optimum. The goal is to start in the basin of attraction of the global optimum.

When faced with a hypothesis space that has many local optima, there are three major categories of approaches to find the global optimum: Deterministic approaches, stochastic sampling approaches and stochastic escaping approaches [41].

Deterministic approaches are typically *branch-and-bound* algorithms. They cut the hypothesis space into pieces (also known as *branches*) and explore those pieces in further detail, exploring the most promising pieces first. One such algorithm is DIRECT, discussed in Section 4.1. Deterministic approaches are guaranteed to converge towards the global optimum, given enough time.

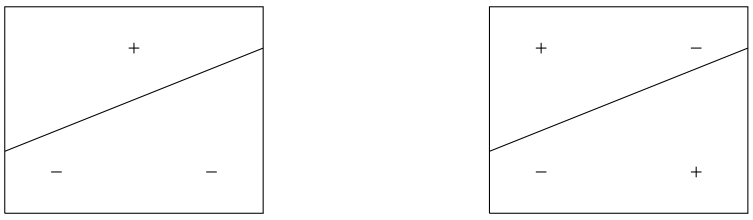
In stochastic sampling approaches, a number of random starting states is sampled, in the hope that one of them is in the basin of attraction. LeGO has such an approach, discussed in Section 3.2. Stochastic sampling approaches are Monte Carlo algorithms, increasing the chance of reaching the global optimum if given more time.

The escaping approach is to provide the option to escape a local optimum. This usually involves trying hypotheses nearby the local optimum and locally optimising those to see if they would converge to a better option, or allowing larger steps through the hypothesis space. Examples are tabu search, stochastic tunneling or simulated annealing.

2.3 Representational Capacity

The hypothesis space contains only hypotheses that can be found by the machine learner. In some cases, the problem is so complex that the model of the machine learner cannot represent all possible solutions. The best solution may not be in this hypothesis space. The capacity of a hypothesis space to represent solutions affects machine learning this way.

To measure the representational capacity of the model, the *Vapnik-Chervonenkis dimension* or *VC dimension* of a model can be used [10]. This metric is defined as the largest number of points that can be shattered by at least one objective function that can be generated by the model. Shattering means here that it can always be separated correctly into two classes. For a simple example, in Figure 1, the model can represent all linear functions. Regardless of how the points are arranged or classified, three points can always be separated by a linear function into two classes. Four points, however, can be arranged and classified in a fashion that cannot be separated by linear functions (the XOR arrangement). There is no straight line possible that can place all positive points of Figure 1b on one side and all negative points on the other. Therefore, a model that represents only two-dimensional linear functions has a VC dimension of 3.



(a) Three points can always be shattered by a linear function.

(b) Four points can not always be shattered by a linear function.

Figure 1: Linear functions have a VC dimension of 3, since that is the largest number of points they can shatter.

Two extremes are discussed in this thesis. When the VC dimension of the model of a machine learning procedure is too low for a complex objective function, there is no hypothesis in the hypothesis space that represents the objective function. One solution to this is to combine multiple models in an *ensemble*, expanding the hypothesis space. This is discussed in Section 4.3. When the hypothesis space is extremely simple, and its representational capacity small, the machine learning procedure may be enhanced with a *least squares* method. This may increase the speed of convergence and reduce computational cost. This is discussed in Section 4.2.

3 Pre-processing Steps

Among the solutions to the problems posed to machine learning by the hypothesis space, many do not actually apply to the machine learning process itself but rather to the preparation for it. These steps can be taken independently, since they are applied only to the data (Section 3.1) or to the hyper-parameters of a machine learning process (Section 3.2).

3.1 Scaling of Dimensions

This section is concerned with the dimensionality property of hypothesis spaces.

The most common bias in machine learning is to give similar hypotheses a similar output. However, ‘similar’ is a rather vague term. Measuring similarity is usually done by measuring the similarity of the individual features [47]. However, not all dimensions of the hypothesis space are of equal importance to the result. Better indications of similarity can be achieved by scaling these dimensions appropriately first, so that important dimensions are emphasised [4].

Two disjunct options are discussed: Scaling the hypothesis space with the covariance matrix of the data, and removing irrelevant dimensions altogether. The techniques used for these options are not wholly different; both use correlations between the dimensions to determine which dimensions are important.

3.1.1 Transforming with the Covariance Matrix

With high-dimensional problems, irrelevant features will often influence the machine learner in two ways. Firstly, random variability will influence the machine learner, because it may see a correlation or a pattern in it. This compromises its accuracy. It is known as the problem of *irrelevant variability* [55]. Secondly, not all dimensions may be in the same order of magnitude. For example, if one parameter ranges from 0 to 1000 and another from 0 to 1, a small variation of 20 units in the first parameter will drown any variation in the second parameter, if both are weighed equally in the similarity measurement.

To remedy these problems, a linear transformation can be performed on the data. This transformation should emphasize the relevant dimensions and bring them into the same order of magnitude. One such transformation is called the *Whitening Transformation* [3]. The whitening transformation will not only scale the features to a comparable range [21], but also amplify strong correlations, suppress noise [57], and maximise entropy along the axes [4]. It can be achieved by using the covariance factors of every pair of dimensions. These covariance factors are a measure of the correlation between the two dimensions.

A simple and widely used implementation is *Relevant Component Analysis* (RCA) [3]. It involves four steps:

1. Split the training data into classes K using an unsupervised classifier. This allows RCA to devise new dimensions and discard old dimensions.

2. Find the average $\vec{\mu}_k$ of each class $k \in K$, by taking the average of all training instances per dimension.

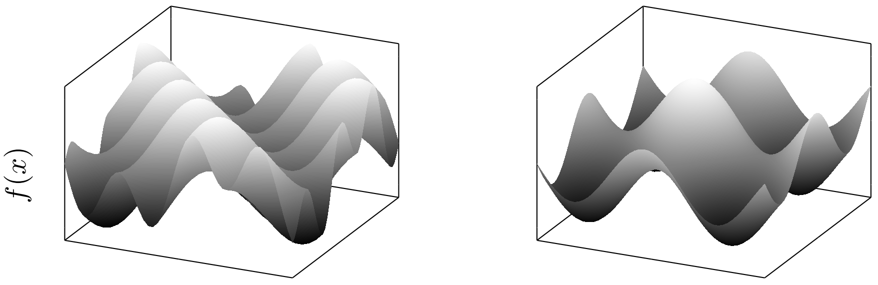
3. Compute the covariance matrix $\hat{C}$:

$$\hat{C} = \frac{1}{|X|} \sum_{j=1}^{n} \sum_{i=1}^{|K_j|} (\vec{x}_{ji} - \vec{\mu}_j)(\vec{x}_{ji} - \vec{\mu}_j)^T \qquad (1)$$

   Here X is the set of all training data, n is the number of classes returned by the classifier and $\vec{x}_{ji}$ is the ith training example in class j.

4. Apply the whitening transformation on the training data: $W = \hat{C}^{-\frac{1}{2}}$ Taking the full inverse, $\hat{C}^{-1}$, would turn the data into white noise, removing all correlations. $\hat{C}^{-\frac{1}{2}}$ retains some of the original correlations while still transforming the data in the right direction.

The result of the whitening transformation is visualised in Figure 2. Note that the domain of the objective function has changed due to the transformation and that all future uses of the original feature set have to be transformed with the whitening transformation.



(a) Before scaling (b) After scaling

Figure 2: The effect of the whitening transformation.

3.1.2 Feature Extraction

Instead of scaling irrelevant features such that their effect is small, one could also just remove them altogether. Removing features reduces the time and memory complexity of the problem, makes a model more robust, and allows knowledge extraction by humans [1]. *Feature Extraction* is a class of algorithms that creates a new set of dimensions by combining the original dimensions. Two major generic Feature Extraction algorithms are called *Principal Components Analysis (PCA)* and *Factor Analysis (FA)*. This section will compare the two.

Principal Component Analysis extracts the *principal components* from the data and chooses a subset of them to keep [33]. These principal components are a set of independent vectors. The first of them is the vector along which the data has most variance. The second is the vector perpendicular to the first, along which the data has most variance. The third is the vector perpendicular to the first and second, along which the data has most variance, and so on until all variance is accounted for. If the data has been scaled with the covariance matrix as in Section 3.1.1, applying PCA is extremely simple. These principal components are the eigenvectors of the covariance matrix, and can be computed by first transforming the covariance matrix with the whitening transformation, and then performing eigenvalue decomposition on it. Choosing a subset of the principal components to keep is not trivial however. Common strategies include:

- Retain only a certain percentage of deviation [30]. The smallest eigenvectors are removed until the deviation drops below a certain fraction of the original deviation.
- Retain only components with higher variation than a certain constant [31] [32]. Components with lower variation are dropped.
- Manually select an eigenvalue treshold using a scree graph [29]. Eigenvalues that are smaller are dropped. The scree graph sorts the eigenvectors by their size and allows a human to make an educated guess where to put the treshold.
- Automatically select an eigenvalue treshold using random data [25]. A simulation is made and a treshold is placed where the statistical deviation can no longer be attributed to random variation.
- Automatically select an eigenvalue treshold using cross-validation [45]. Numerous tresholds are tried with different samples of the data and compared against each other.

The data is then projected onto the remaining principal components, effectively scaling the discarded components to zero. This removes the noise caused by them and allows for a new set of dimensions that includes only the principal components.

Factor Analysis assumes that the observed features can be expressed, except for an error term, as weighted sums of unobserved features or *latent* *factors.* It assumes the following simple model for every training example i [16]:

$$\vec{x}_i = \vec{\mu} + \vec{\varepsilon}_i + \sum_{n=1}^{k} \vec{l}^n F_i^n \tag{2}$$

Where:

- $\vec{x}_i$ is a vector of the features of the ith training example.
- $\vec{\mu}$ is a vector of the average values of all features.
- $\vec{l}^p$ is a vector of unknown weights for the pth latent factor.
- F_i^p is the value of the pth latent factor for the ith training example.
- $\vec{\varepsilon}_i$ is a vector of the errors for the ith training example.
- k is the number of latent factors to model.

The problem is then to find the matrix of weights L that minimises the sum of the errors $\vec{\varepsilon}$. This is done by using the covariance matrix $\hat{C}$ and the following equality [16]:

$$LL^T = \hat{C} \tag{3}$$

Once the covariance matrix is known, L can be computed algebraically.

A key factor to success, as with PCA, is to select the appropriate value for k, the number of latent factors to model [40]. This can be done using similar methods as with PCA [19], such as:

- Set k to the number of eigenvalues in the data greater than a certain constant [35].
- Manually select an eigenvalue treshold using a scree graph [14].
- Automatically select an eigenvalue treshold using random data [25].

It is also shown that FA is most effective when the amount of latent factors is less than a third of the amount of features [42].

Both PCA and FA attempt to reduce the dimensionality of the problem by creating new, useful dimensions while keeping the loss of important information to a minimum. Indeed, they have strong similarities. Both use the covariance matrix to extract the most useful correlations. However, PCA concentrates on the diagonal elements of the covariance matrix, while FA concentrates on the off-diagonal elements [33]. This causes them to select different correlations to keep. In fact, they can even be seen as opposites, since PCA computes dimensions that are a linear combinations of the original features, while FA computes new dimensions such that the original features are linear combinations of those. This leads to PCA being generally more restrictive (and easier to compute) than FA [5].

3.2 Choosing Starting Point(s)

This section is concerned with global optimisation: Finding the global optimum in a hypothesis space that contains many local optima. There are many ways to address this problem, but perhaps the most common one is to try numerous starting states in the hope that one of them falls in the basin of attraction of the global optimum. Selecting these initial states randomly is a simple option, but other methods have been invented that might produce better results.

One of these is LeGO [13]. LeGO tries to increase the chance of finding the global optimum by marking certain areas of the hypothesis space as promising, if they are promising, and searching only in those areas. First, LeGO tries numerous starting states, and records the fitness of the resulting optimal hypotheses. Then, a classifier is trained on those starting states. It classifies good starting states as positive and the rest as negative. Then, in the second stage, more initial states are generated but before they are used by the main machine learning algorithm, they are first checked against the classifier to see whether they have promise, so that only the promising starting points have to be sampled. Figure 3 gives a synthetic example.

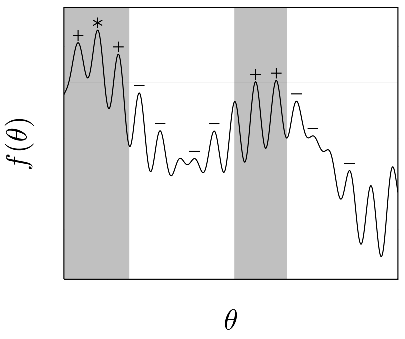


Figure 3: Running LeGO over initial state vector θ. Promising initial states are marked positive (+), others negative (−), resulting in a classifier that marks the shaded area as promising.

The classifier used by Cassioli et al. in their presentation of LeGO is a Support Vector Machine, but different machine learning techniques may be used. In particular, training this classifier is an active learning process, for which the field of information theory provides some useful enhancements (see Section 4.1). The Support Vector Machine constructs a class of positive instances and a class of negative instances. Positive instances result in better outcomes, and the assumption is made that the area around them also results in better outcomes. Therein lies the inductive bias: It is assumed that the global optimum is near to the better subset of local optima. If this assumption is true, then LeGO will trim the search space for initial states correctly and the process convergest faster. If this assumption is false, the process will no longer converge towards the global optimum as with the random choice of starting points.

The idea of using data from previous iterations of the main algorithm wasn't new, even in applying it to choosing starting positions [2]. More often though has it been applied to finding other hyper-parameters than just the initial state [8]. Instead of (or in addition to) classifying starting points, they classify combinations of the parameters of the machine learning algorithm.

4 Improvements

When the data is carefully formatted and selected, and the hyper-parameters of the machine learning algorithm are chosen carefully, it is time to commence the machine learning process itself. In this phase, a different approach must be taken to select improvements to the process. Which improvements to use depends not only on the features of the hypothesis space, but also on the algorithm to embed it in. Some of these improvements may be less effective when used with methods that already incorporate the same bias.

4.1 Exploration vs. Exploitation

Active learning problems are supervised machine learning problems that allow the learner to query an information source to get the desired output of a hypothesis [54]. These problems have an extra element to them: Which hypotheses should the learner query to get the most useful information? Among others, a major consideration is whether to explore the hypothesis space, or exploit. In other words, how often should the machine learner query hypotheses similar to the current best hypotheses (exploiting), and how often should it explore new areas? This ties closely with the property of local optima in a hypothesis space. When a problem has many local optima, more exploration needs to be done to find the basin of attraction of the global optimum. Balancing exploration and exploitation becomes extremely critical.

To guide the decision to explore or exploit, many classical algorithms make use of the so-called *Lipschitz constant*. This constant is a bound on the rate of change of the objective function [48]. If an algorithm assumes the Lipschitz constant K, it means that it assumes the slope of the objective function never exceeds K, or more formally, that $|\frac{\partial f(x)}{\partial x}| \leq K$ for every $x \in X$. The real Lipschitz constant of a function can be estimated [60], but it is often very costly to do so [34] and has little value.

In practice, the Lipschitz constant is not just used as a property of the objective function. It is used as an artificial constraint, to balance exploration and exploitation. It means that hypotheses that are too far ahead of any known hypotheses cannot be exploited yet. It is easiest to visualise in the setting of a hill climber, as in Figure 4. The hill climber cannot exceed the slope that is assumed by the Lipschitz constant. It is not allowed to climb into the steepest direction because its slope is too steep. Instead, it is forced to circle around the hill, and thus it is forced to explore before exploiting. The amount of exploration is tuned by adjusting the Lipschitz constant. Low K will force the machine learner to explore more, and high K will allow the machine learner to exploit more [56].

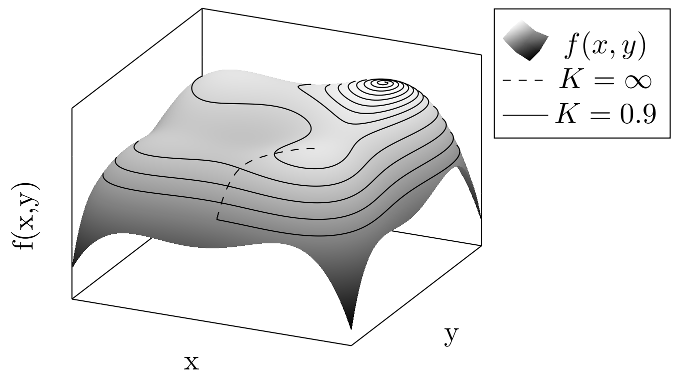


Figure 4: The finite Lipschitz Constant prevents the hill climber from going into the nearest local optimum because it is too steep. It has to circle around and explore.

More recent developments have attempted to do without a pre-defined or calculated constant. This has the advantage that there is one less hyper-parameter to optimise and the algorithm is generally less restricted. Direct is built with that purpose [20]. Direct recursively splits the hypothesis space into pieces. These pieces are rectangular shapes in all n-dimensions called hyper-rectangles. The hyper-rectangles that have the greatest potential are then explored further, by splitting it up into three parts and querying their fitness [34]. The fitness of every hyper-rectangle is measured by querying a sample in its geometrical centre. This is why the hyper-rectangles are split into three: The middle rectangle will have the same centre and does not need to be sampled again. A visualisation of this process is made in Figure 5.

Selecting which hyper-rectangles to explore further embodies the choice between exploration and exploitation. Direct chooses all rectangles that would be selected with any value as Lipschitz constant. Formally, a hyper-rectangle i is chosen if there exists some Lipschitz constant K such that

$$\begin{aligned}(\forall i \in 1, ..., m) f(c_j) - K d_j \leq f(c_i) - K d_i \\ \text{and } f(c_j) - K d_j \leq f_{min} - \varepsilon |f_{min}|\end{aligned} \tag{4}$$

where c_x is the centre of rectangle x, d_x is the Euclidian distance from the centre point of hyper-rectangle x to its corners, m is the number of hyper-

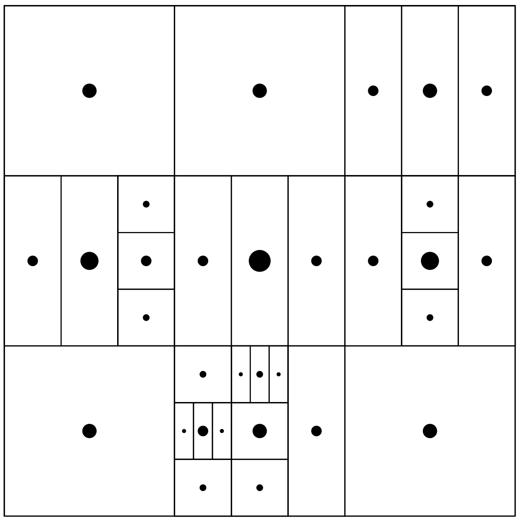


Figure 5: Direct recursively splits the hypothesis space into 3 hyper-rectangles. The size of the dots indicates the order in which their hyper-rectangles are created.

rectangles, f_{min} is the most optimal output found so far, and ε is a positive constant that forces new samples to exceed the most optimal output by a certain factor.

The ε variable is meant as a stopping criterium. For instance, if $\varepsilon = 0.01$, a hyper-rectangle is only exploited if it exceeds the best solution by at least 1%. If no such rectangle can be found, the algorithm stops. Since the distance between the hyper-rectangles keeps getting smaller, direct will converge to the global optimum, as long as a finite global optimum exists and ε is small enough.

A possible extension of this algorithm is *Multilevel Coordinate Search*. MCS imposes a lower limit on how far the hypothesis space is split up, or how many iterations are performed [28]. The resulting fractures of the hypothesis space are individually less complex than the original space, and can be solved by a conventional local search. This idea solves an important problem with direct where the algorithm converges slowly if the global optimum happens to be at the edge of one of the rectangles. Additionally, instead of always splitting a section into three hyper-rectangles, a machine learner can also determine how to split the space into parts. MCS takes the good side of direct [37] and leaves the choice to the designer which local search algorithm would work best for splitting the hypothesis space and exploiting the sections.

4.2 Least Squares

This subsection is concerned with the case where the hypothesis space has little representational capacity. In this subsection, machine learning problems are approached from another perspective. In the field of machine learning, many procedures look for an optimum by an iterative process. Starting from some arbitrary state, the algorithm incrementally changes its parameters towards better solutions until a desirable final state is reached. If the hypothesis space is simple, this process is an overkill.

The field of statistical regression provides a different approach. *Least squares* is a classical statistical technique for regression [51]. Indeed, it is often used as a basic machine learning tool as well. Instead of using least squares techniques to approximate the objective function however, this section will discuss how least squares is used to approximate the hypothesis space, if the space is simple.

Least squares techniques minimise the squared error by making a projection of the data onto a pre-defined function space, effectively fitting a function to the data. To explain this, it is best to describe the process of the most basic variant: linear least squares. Linear least squares computes a linear function. Its function space is the space of all n-dimensional linear functions. The process goes as follows [9]:

1. Let X be a matrix containing the training data, with a row for each training instance and a column for each feature. An extra feature is added, x_0, that is 1 for every training instance.
2. Let y be a vector containing the output for every training instance.
3. Calculate $\beta = (X^T X)^{-1} X^T y$.
4. The resulting regression is the linear function $f(\vec{x}) = \beta_0 x_0 + \beta_1 x_1 + ... + \beta_n x_n$.

The important step is the third, calculating $\beta = (X^T X)^{-1} X^T y$. This is what constructs a projection of the data onto the function space.The resulting β is the vector along which the sum of the squared differences to the training data is minimised. The extra feature x_0 is added as a constant factor, allowing the regressor to add a constant to the resulting linear function, so that the result can be $y = c + \beta_1 x_1 + ... + \beta_n x_n$ instead of $y = \beta_1 x_1 + ... + \beta_n x_n$. For the matrix inversion to take place, the determinant of the matrix cannot be zero. For this, the assumption is made that the data is linearly independent. If Feature Extraction has been applied to the data, as in Section 3.1.2, this is already the case. In other cases, the pseudoinverse is also sufficient [9].

Linear least squares only constructs linear functions, but nonlinear functions can be constructed as well as long as a basis function is given [52]. For instance, a quadratic function (with basis function $\{1, x, x^2\}$) can be approximated by using the terms 1 (the constant factor), x and x^2 as features. This way, any continuous function can be approximated as long as its basis function is known and finite. Moreover, if a specific artificial basis is desired, such as cubic functions, the nonlinear least squares regressor can use that basis as well and will output a cubic function that approximates the data.

In addition to using least squares as a regression technique to approximate the objective function, it can also be used in system identification [15]. Here, it is used to approximate the function that maps the hypothesis space to the fitness of the hypotheses, essentially fitting a function to the hypothesis space. Next, the extrema of that function can be found and thus the optimal hypothesis is computed. Using least squares in this way retains the original model of the machine learning algorithm it is applied to. It replaces the iterative part of a machine learning algorithm. This can greatly increase the speed of convergence in some cases, but that is highly dependent on the problem [44]. In general, least squares as system identification works best when the hypothesis space is simple and a basis function is known that approximates it well.

That brings up some important weaknesses of least squares methods:

- Least squares methods require a matrix inversion, of which the execution time grows cubically with the complexity of the base function [9]. This problem may be remedied by sacrificing accuracy: The (pseudo)inverse may be approximated with heuristics. In general though, least squares is inappropriate to use for complex or high-dimensional hypothesis spaces (high VC-dimension), such as for huge neural networks or Bayesian networks.

- Nonlinear least squares methods require a basis function, that approaches the basis function of the hypothesis space. This basis can be extremely hard to find. Often, a simple quadratic basis function is used, but that is not able to represent a complex hypothesis space.

- Least squares methods are generally fairly sensitive to error [17]. This results from minimising the square of the error, which grows very fast for far outliers. This can be remedied by filtering out the significant outliers or weighting them, but least squares may be the inappropriate technique to use when the input is measured with imprecision.

4.3 Combining Learners

The other effect of the representational capacity of hypothesis spaces occurs when the problem is so complex that the hypothesis space does not represent the solutions well [18]. Even if the machine learner were to find the optimal hypothesis in its hypothesis space, there could be a much better solution that is not represented by one of the hypotheses and the optimal hypothesis may be inadequate. There are ways to increase the representational capacity of a hypothesis space so that it may represent more and better solutions. One of these are *ensemble methods*: systems of machine learners, combined to work together.

Ensemble methods combine multiple hypothesis spaces into a new hypothesis space. This new hypothesis space can be much larger, and can have a much greater representational capacity. Not all learners are limited in the functions it can represent [26], but even for those learners ensembles can also improve the speed of convergence and accuracy greatly [43]. Ensemble methods tend to improve results with a small data set, and increase the basin of attraction of the global optimum [18]. The major drawback lies within its computational cost [1]. This however becomes less and less of a problem due to the rapid advances in hardware technology. Modern professional machine learning systems are often huge ensembles of hundreds of learners combined.

Many types of ensembles exist, but *bagging* and *boosting* are the most common and both fairly easy to understand. Those are the two that will be discussed in this section. Bagging and boosting are both classified as voting methods, since they give learners a (possibly weighted) vote on what the output should be.

Firstly, *bagging*, short for *bootstrap aggregating*, takes multiple learners and trains each of them on a sample of the data. These samples should be taken with replacement and be roughly the same size as the original data set [50]. Random sampling is a good option; some training instances may be missing from a sample and some may be double, but this doesn't matter. To get a result, all learners in the ensemble are then queried and the average of their answers is returned [11]. Because of this averaging, bagging especially improves unstable learners: those that are apt to produce wildly different predictions with slightly different priors [58] [49]. When creating an ensemble of unstable learners (even if they are the same algorithm), or of different algorithms with different bias, averaging the results reduces the variance.

Boosting also combines multiple learners, but works more sophisticatedly. Boosting is less constrained than bagging: It is defined as bagging while changing the distribution of the training set [6]. The idea behind it is to combine multiple weak rules into one strong rule [36]. A weak rule is a rule that is correct at least more often than when guessing. Examples of such rules are that the price of a house increases with its floor area, or that houses with slanted roofs sell for more than those with flat roofs. With boosting, many such weak rules are combined into one stronger rule: a rule that is (nearly) always correct.

AdaBoost is the most influential implementation that brings this into practice [61]. AdaBoost creates a sequence of hypotheses using weak learners, and combines them with weights [22]. Its classical implementation goes as follows:

1. First, scale all labels of the training data to the range $[0, 1]$. This is required for error calculation. If there is no known limit to the output, take the lowest and highest known outputs.
2. Initialise the weight of every learner to 0.
3. At each iteration, until there are no more learners or the error is acceptable:
   (a) Find the yet unused learner with the least error over the training data.
   (b) Set the weight of that learner to $\frac{1}{2}\ln(\frac{1-\varepsilon}{\varepsilon})$ where ε is the error of the learner. Greater error will reduce the influence of this learner.
   (c) Update the distribution of the training instances to emphasize the instances where the weighted average of all learners so far have the greatest error. The next learner will then have to get a better estimation on those instances if it is to be selected for the next round.

Voting methods such as bagging and boosting increase the representational capacity of the hypothesis space, by introducing the hypotheses of multiple machine learners. Remarkably, neither bagging nor boosting is prone to overfit, a phenomenon that normally accompanies large, complex hypothesis spaces due to Occam's Razor [6] [12]. This is because voting mechanisms attempt to maximise their confidence, and to that end they tend to select rules that are similar to each other [53]. Occam's Razor still applies, but it is not the simplest rule that is the best, but one that is close to a simple rule.

Ensemble methods by nature introduce new hyper-parameters: which base learners to use from a library of available learners, how many of each to use, what part of the data to train them on, and of course the hyper-parameters of every base learner itself and how to vary them between learners. These hyper-parameters can be tuned by another learner (or ensemble). This allows ensembles to be thought of as active learning problems themselves [23]. Instead of searching through a space of hypotheses for what could be the best approximation to the objective function, this learner searches through a space of possible ensembles for what could be the best ensemble for this problem. This learner has its own separate hypothesis space. The new hypothesis space may be much easier to learn than the original.

5 Future Research

It is nearly impossible to list all proposed solutions dealing with dimensionality, local optima and representational capacity. The ones discussed in this thesis have been either remarkably influential or have proven to be really effective in common situations. Moreover, some aspects were deliberately left out of this work to keep it simple.

A more comprehensive treatment could include research on the *bias* of the machine learning methods. Each of these methods introduces a new bias [47]. Some hypotheses are chosen over other ones based on this bias, such as hypotheses close to the current best, or those classified positive by LeGO. These methods only work if the bias holds true in the case of that specific problem. Are the best hypotheses really close to the current best, or to the ones classified positive? The bias of these methods is often poorly defined, but it would help to develop better machine learners.

Another topic not included here is how to deduce the properties of a hypothesis space from the given data. Dimensionality is trivial, but when does a problem have enough regions of attraction to apply an algorithm like DIRECT effectively? How does one know that the hypothesis space is not able to represent a sufficient solution?

6 Conclusion

This work has discussed three important properties of a hypothesis space: dimensionality, local optima and representational capacity. This section summarises the topics that have been discussed in the thesis.

Higher dimensionality was found to reduce the effect of training data at an exponential rate. To counter this effect, features could be scaled to maximise relative meaningfulness of each dimension and remove irrelevant variability using Relevant Component Analysis. Additionally, the total dimensionality of the problem could be reduced using Feature Extraction techniques such as Principal Component Analysis and Factor Analysis.

A hypothesis space with many local optima made it more difficult to find the global optimum among them. This problem could be countered using sampling techniques such as LeGO, or by splitting the hypothesis space into segments using DIRECT.

Representational capacity could present a problem in the case that the desired output is not a hypothesis in the hypothesis space of the model used by the machine learning algorithm. Combining multiple machine learners in an ensemble could vastly increase the capacity of the hypothesis space as well as improving accuracy. When a specific, simple hypothesis space is desired, the performance of a machine learning process could be improved by incorporating least squares techniques into it in stead of iterating over the hypothesis space.

The goal of this work was to describe the effects of these properties on machine learning and to describe how to work with these properties. It is written in such a way as to be helpful for any developer in the design phase of their machine learning procedure. I hope it will help them to improve their machine learning algorithms.

References

[1] Alpaydin, E. *Introduction to Machine Learning.* MIT Press, 2004.

[2] Ampatzis, C., and Izzo, D. Machine learning techniques for approximation of objective functions in trajectory optimisation. In *Proceedings of the IJCAI-09 Workshop on Artificial Intelligence in Space* (2009), Springer, pp. 1–6.

[3] Bar-Hillel, A., Hertz, T., Shental, N., and Weinshall, D. Learning distance functions using equivalence relations. In *Proceedings of the Twentieth International Conference on Machine Learning* (2003), vol. 20, p. 11.

[4] Bar-Hillel, A., Hertz, T., Shental, N., and Weinshall, D. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research 6*, 1 (2006), 937.

[5] Bartholomew, D., Knott, M., and Moustaki, I. *Latent Variable Models and Factor Analysis: A Unified Approach*, 3 ed. John Wiley & Sons, Ltd, 2011.

[6] Bauer, E., and Kohavi, R. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning 36*, 1 (1999), 105–139.

[7] Bellman, R. E. *Dynamic Programming.* Courier Dover Publications, 2003.

[8] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *25th Annual Conference on Neural Information Processing Systems* (2011).

[9] Björck, Å. *Numerical Methods for Least Squares Problems.* SIAM: Society for Industrial and Applied Mathematics, 1996.

[10] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery 36*, 4 (1989), 929–965.

[11] Breiman, L. Bagging predictors. *Machine Learning 24*, 2 (1996), 123–140.

[12] Breiman, L. Arcing classifier (with discussion). *The Annals of Statistics 26*, 3 (1998), 801–849.

[13] Cassioli, A., Di Lorenzo, D., Locatelli, M., Schoen, F., and Sciandrone, M. Machine learning for global optimization. *Computational Optimization and Applications 51*, 1 (2012), 279–303.

[14] Cattell, R. B. The scree test for the number of factors. *Multivariate Behavioral Research 1*, 2 (1966), 245–276.

[15] Chen, S., Billings, S. A., and Luo, W. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control 50*, 5 (1989), 1873–1896.

[16] Comrey, A. L., and Lee, H. B. *A First Course in Factor Analysis*, 2 ed. Lawrence Erlbaum, 1992.

[17] Cornbleet, P. J., and Gochman, N. Incorrect least-squares regression coefficients in method-comparison analysis. *Clinical Chemistry 25*, 3 (1979), 432–438.

[18] Dietterich, T. G. Ensemble methods in machine learning. *Multiple Classifier Systems* (2000), 1–15.

[19] Fabrigar, L. R., Wegener, D. T., MacCallum, R. C., and Strahan, E. J. Evaluating the use of exploratory factor analysis in psychological research. *Psychological Methods 4*, 3 (1999), 272.

[20] Floudas, C. A., and Parlados, P. M. *Encyclopedia of Optimization*, 2 ed. Springer, 2009.

[21] Fodor, I. K. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory 9* (2002), 1–18.

[22] Freund, Y., and Schapire, R. E. A short introduction to boosting. *Japanese Society for Artificial Intelligence 14*, 771–780 (1999), 1612.

[23] Friedman, J., Hastie, T., and Tibshirani, R. *The Elements of Statistical Learning*, 2 ed., vol. 1. Springer Series in Statistics, 2001.

[24] Gordon, J., and Shortliffe, E. H. A method for managing evidential reasoning in a hierarchical hypothesis space. *Artificial Intelligence 26*, 3 (1985), 323–357.

[25] Horn, J. L., and Engstrom, R. Cattell's scree test in relation to bartlett's chi-square test and other observations on the number of factors problem. *Multivariate Behavioral Research 14*, 3 (1979), 283–300.

[26] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[27] Hughes, G. F. On the mean accuracy of statistical pattern recognizers. *Information Theory, IEEE Transactions on 14*, 1 (1968), 55–63.

[28] Huyer, W., and Neumaier, A. Global optimization by multilevel coordinate search. *Journal of Global Optimization 14*, 4 (1999), 331–355.

[29] Jackson, D. A. Stopping rules in principal components analysis: A comparison of heuristical and statistical approaches. *Ecology* (1993), 2204–2214.

[30] Jackson, J. E. *A User's Guide to Principal Components*, vol. 244. Wiley-Interscience, 1991.

[31] Jolliffe, I. T. Discarding variables in a principal component analysis; i: Artificial data. *Applied Statistics* (1972), 160–173.

[32] Jolliffe, I. T. Discarding variables in a principal component analysis; ii: Real data. *Applied Statistics* (1973), 21–31.

[33] Jolliffe, I. T. *Principal Component Analysis*, 2 ed. Springer, New York, NY, USA, 2002.

[34] Jones, D., Perttunen, C., and Stuckman, B. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application 79*, 1 (1993), 157–181.

[35] Kaiser, H. F. The application of electronic computers to factor analysis. *Educational and Psychological Measurement* (1960).

[36] Kearns, M. Thoughts on hypothesis boosting. *Unpublished Manuscript* (1988).

[37] Kelley, C. T. *Iterative Methods for Optimization.* SIAM, 1999.

[38] Kobayashi, S. Approximate identification, finite elasticity and lattice structure of hypothesis space. Tech. rep., Department of Computational Science and Informatical Mathematics, University of Electro-Communications, 1996.

[39] Langley, P., and Iba, W. Average-case analysis of a nearest neighbor algorithm. In *International Joint Conference on Artificial Intelligence* (1993), vol. 13, Citeseer, pp. 889–889.

[40] Ledesma, R. D., and Valero-Mora, P. Determining the number of factors to retain in efa: An easy-to-use computer program for carrying out parallel analysis. *Practical Assessment, Research & Evaluation 12*, 2 (2007), 1–11.

[41] Liberti, L. Introduction to global optimization, 2008.

[42] MacCallum, R. C., Widaman, K. F., Zhang, S., and Hong, S. Sample size in factor analysis. *Psychological Methods 4*, 1 (1999), 84–89.

[43] Maclin, R., and Optiz, D. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research 11* (1999), 169–198.

[44] Marcet, A., and Sargent, T. J. Speed of convergence of recursive least squares learning with arma perceptions. Economics Working Papers 15, Department of Economics and Business, Universitat Pompeu Fabra, May 1992.

[45] Mertens, B., Fearn, T., and Thompson, M. The efficient cross-validation of principal components applied to principal component regression. *Statistics and Computing 5*, 3 (1995), 227–235.

[46] Mitchell, T. M. Generalization as search. *Artificial Intelligence 18*, 2 (1982), 203–226.

[47] Mitchell, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[48] Ó Searcóid, M. *Metric Spaces*. Springer, 2006.

[49] Poggio, T., Rifkin, R., Mukherjee, S., and Rakhlin, A. Bagging regularizes. Computer Sciences Technical Report 2139, Massachusetts Institute of Technology, 2002.

[50] Quinlan, J. R. Bagging, boosting, and c4.5. In *Proceedings of the National Conference on Artificial Intelligence* (1996), pp. 725–730.

[51] Rao, C. R., and Toutenburg, H. *Linear Models: Least Squares and Alternatives*. Springer, 1999.

[52] Rawlings, J. O., Pantula, S. G., and Dickey, D. A. *Applied Regression Analysis: A Research Tool*, 2 ed. Springer, 1989.

[53] Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics 26*, 5 (1998), 1651–1686.

[54] Settles, B. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[55] Shental, N., Hertz, T., Weinshall, D., and Pavel, M. Adjustment learning and relevant component analysis. *Computer Vision - ECCV 2002* (2006), 181–185.

[56] Shubert, B. O. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis 9*, 3 (1972), 379–388.

[57] Tenenbaum, J. B., and Freeman, W. T. Separating style and content with bilinear models. *Neural Computation 12*, 6 (2000), 1247–1283.

[58] Turney, P. Technical note: Bias and the quantification of stability. *Journal of Machine Learning 20* (1995).

[59] Wolpert, D. H., and Macready, W. G. No free lunch theorems for search. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[60] Wood, G. R., and Zhang, B. P. Estimation of the lipschitz constant of a function. *Journal of Global Optimization 8*, 1 (1996), 91–103.

[61] Wu, X., and Kumar, V. *The Top Ten Algorithms in Data Mining*, vol. 9. Chapman & Hall/CRC Press, 2009.