Metaheuristics Applied to the Feature Selection Problem

Master thesis

Tom Fredrik Blenning Klaussen

12th June 2006
Preface

This work has been conducted at the Department of Mathematics at the University of Oslo.

Mentors have been Anne Solberg, Associate Professor at the Department of Informatics at the University of Oslo and Geir Hasle, Research Director at SINTEF Applied Mathematics. I would like to thank them both for their valuable support during the work with this thesis.

I would also like to thank Rolv Bredesen and Thomas Elboth for their contributions during the final days of this work.
Summary

This work is my master thesis. It is concentrated around the problem of feature selection, which is an important field in pattern analysis and computer vision.

The thesis consists of 9 chapters. In the first chapter I give an introduction to the problem and the theory involved. In chapter 2 I introduce the feature selection problem. In chapter 3 I look at discrete optimization problems. In chapter 4 I look at what others have done before. In chapter 5 I analyze the problem. In chapter 6 I present some new methods for the feature selection problem, alongside pseudocode for the implementation of previously known methods. In chapter 7 I discuss the numerical difficulties that arise. In chapter 8 I look at some experimental results. In chapter 9 I draw some conclusions and sum up future work.

The novelties I present are foremost the Guided Roaming SA, HuffPuff, Roaming Search and Roaming SA algorithms. There are also the adaptations ILS, VNS, Optimal Path Relinking and Inverse Path Relinking. The problem formulation in (7.13) is quite obvious, although to the best of my knowledge it has never been shown before. The visualization algorithm is also new.

The thesis was originally meant to be about metaheuristics, and I originally intended to develop a small program to do the classification work for me. During the course of the work with the thesis, however, the library grew, and ended up as almost 16000 lines of c++ code. The numerical difficulties necessitated the study of several numerical techniques, including regularization which was done as a stabilizing technique for numerics, rather than in its own right for classification. The library implements among other things function caching, which has been imperative to the speed of many of the local search algorithms, as it precludes reevaluation of most recently evaluated functions.

Since the feature selection problem is a problem which involves several professions, I have added a glossary in the end which aims at resolving some nomenclature issues. The words that I explain in the glossary are marked by an underline in the text.
Contents

Preface
Summary

1 Introduction
  1.1 Motivation .............................................. 1
  1.2 A play example ......................................... 1
    1.2.1 Separability ....................................... 2
    1.2.2 Classifier performance .......................... 5
  1.3 Various approaches to pattern recognition ........... 6
    1.3.1 Statistical pattern recognition ............... 6
    1.3.2 Neural networks .................................. 6
    1.3.3 Syntactic pattern recognition ................. 8
    1.3.4 Recognition as graph matching ............... 8
    1.3.5 Fuzzy systems ................................... 8
  1.4 Real life problems .................................... 9
    1.4.1 Textural classification .......................... 10
    1.4.2 Hyper-spectral sensors .......................... 10

2 Feature selection - A problem presentation ........... 12
  2.1 The problem ........................................... 12
  2.2 Mathematical description ............................. 13
  2.3 Separability ........................................... 15
    2.3.1 Minimum error classification ................. 15
    2.3.2 Separability measures ............................ 15
  2.4 Non-determinism and the curse of dimensionality .... 16
    2.4.1 Parameter estimation errors .................... 16
    2.4.2 Separation of stages ............................. 17

3 Discrete optimization .................................. 18
  3.1 What is a DOP? ........................................ 19
    3.1.1 Ex: The knapsack problem ....................... 19
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2 Complexity theory</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Exact methods</td>
<td>20</td>
</tr>
<tr>
<td>3.2.1 Full enumeration</td>
<td>20</td>
</tr>
<tr>
<td>3.2.2 Branch and Bound</td>
<td>21</td>
</tr>
<tr>
<td>3.3 Heuristic techniques</td>
<td>21</td>
</tr>
<tr>
<td>3.3.1 Local search</td>
<td>21</td>
</tr>
<tr>
<td>3.3.2 Metaheuristics</td>
<td>22</td>
</tr>
<tr>
<td>3.3.3 Traditional applications of metaheuristics to the Feature Selection Problem</td>
<td>22</td>
</tr>
<tr>
<td>3.3.4 More metaheuristics</td>
<td>24</td>
</tr>
<tr>
<td>3.4 Relaxation</td>
<td>26</td>
</tr>
<tr>
<td>3.4.1 LP problem</td>
<td>26</td>
</tr>
<tr>
<td>3.4.2 LP-relaxation</td>
<td>27</td>
</tr>
<tr>
<td>3.4.3 Lagrangian relaxation</td>
<td>27</td>
</tr>
<tr>
<td>3.5 Fast evaluation</td>
<td>28</td>
</tr>
<tr>
<td>3.5.1 Delta-encoding</td>
<td>28</td>
</tr>
<tr>
<td>4 Previous work</td>
<td>29</td>
</tr>
<tr>
<td>5 Analysis of the Feature Selection Problem</td>
<td>32</td>
</tr>
<tr>
<td>5.1 Relaxation</td>
<td>32</td>
</tr>
<tr>
<td>5.1.1 The knapsack problem relaxed</td>
<td>32</td>
</tr>
<tr>
<td>5.1.2 Feature Selection Problem for Hyper-spectral imaging</td>
<td>33</td>
</tr>
<tr>
<td>5.1.3 Yet another restatement of the feature selection problem</td>
<td>33</td>
</tr>
<tr>
<td>5.1.4 Relaxed feature selection problem</td>
<td>34</td>
</tr>
<tr>
<td>5.2 Evaluation of object functions for the feature selection problem</td>
<td>34</td>
</tr>
<tr>
<td>5.2.1 Precalculation of relative constants</td>
<td>34</td>
</tr>
<tr>
<td>5.2.2 Prediction of performance</td>
<td>34</td>
</tr>
<tr>
<td>6 Methods for the Feature Selection Problem</td>
<td>36</td>
</tr>
<tr>
<td>6.1 General techniques</td>
<td>36</td>
</tr>
<tr>
<td>6.2 Moves</td>
<td>36</td>
</tr>
<tr>
<td>6.2.1 Basic moves</td>
<td>37</td>
</tr>
<tr>
<td>6.2.2 Probabilistic moves</td>
<td>37</td>
</tr>
<tr>
<td>6.3 Metaheuristics</td>
<td>38</td>
</tr>
<tr>
<td>6.3.1 Traditional algorithms</td>
<td>38</td>
</tr>
<tr>
<td>6.3.2 Iterated local search</td>
<td>38</td>
</tr>
<tr>
<td>6.3.3 Variable Neighborhood Search</td>
<td>41</td>
</tr>
<tr>
<td>6.3.4 Roaming Search</td>
<td>41</td>
</tr>
<tr>
<td>6.3.5 RoamingSA</td>
<td>43</td>
</tr>
<tr>
<td>6.3.6 Optimal Path Relinking</td>
<td>44</td>
</tr>
<tr>
<td>6.3.7 Inverse Path Relinking</td>
<td>44</td>
</tr>
<tr>
<td>6.4 Hyperheuristics</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Classification of image regions is one of the most important problems in image analysis today. The basic problem we are faced with, is to distinguish between classes of data. The problems takes many forms: You can try to separate objects from background, or you can try to distinguish water from forest, or even harder, you can try to distinguish pine forest from oak forest. This problem gets harder and harder when the classes you want to distinguish become more “similar” and the number of classes higher. The natural solution to this problem is to try to get more data to distinguish better. But we are faced with the problem that throwing in more information often does not increase accuracy, and sometimes assign even decreases accuracy. So we will try to settle for a smaller subset of the data, and use this to classify. But how do you choose this subset?

The problem we just encountered is called the “Curse of Dimensionality”, and will be discussed in greater detail later.

1.2 A play example

How do you recognize objects? In order to describe the different methods, we will look at a model problem. Consider the task of separating different fruits. This is quite easy to do for a human, but given a ton of this stuff mixed up, there would be an expensive task to have humans do the sorting. It would probably be easier, just to let everything slide along an assembly chain, and having a camera, connected to a control system, deciding which class each object belongs to.

One of the many problems we are faced with is how to do the actual discrimination between classes. We will take a closer look at 5 major ways of recognizing objects, described by Sonka et al.[1].

First we will consider a play example.
1.2. A PLAY EXAMPLE

If we have 10 apples, 10 oranges, and 5 tomatoes, we can choose a set of two features to distinguish them. The two we are going to choose are eccentricity/elongatedness of an circumscribing ellipse, and “yellowness” in a CMY space. We might end up with a situation more or like the situation in fig. 1.1

As the reader might notice with the use of only color as a feature, we are able to separate the oranges from the apples, but there is no possibility of separating tomatoes from the apples. The lines in the picture represent a discriminant function, and assign different parts of the feature space to specific classes.

If we expand our feature space to include shape as well (fig. 1.2), the situation improves, and as we can see we are now able to choose a discriminant function that provides a perfectly divided space.

It is not always possible to separate two classes by the features available. Let us use the same example, but this time let the two classes be lemons and oranges. As you can see in fig. 1.3 it is impossible to separate the two classes. In fact it is difficult to improve the results over the expected results of picking classes at random.

A situation might also arise in which objects cannot be separated by simple linear discriminant functions, but they can be separated by higher order functions (see fig. 1.4).

1.2.1 Separability

As one can see from the above examples, there is a problem of separability, it can be trivial, ranging to impossible to perfectly separate two (or more)
1.2. A PLAY EXAMPLE

Figure 1.2: Fruit example, discriminating on two features

Figure 1.3: Fruit example, no perfect discrimination exist
classes of objects. It all depends on choosing the right features so that each of the classes can be contained within a simple circumscribing shape, ensuring that the section between these shapes is empty. The shape of these discrimination functions should preferably, but not necessarily be convex.

The problem of separability is not only a problem of whether it is possible at all to separate the classes, but it is desirable to discriminate between the two classes with “simple” boundaries. Even if it were possible to perfectly separate two classes by adapting a 20th degree polynomial, it would be extremely over fitted to the problem at hand, and would probably not perform that well if the distribution would have to be estimated. Usually we would limit the degree of the polynomial to second degree or lower.

Another problem is measuring distances between “point-clouds” of data. If we were to approximate some points in a space, we would choose from a few well known metrics, usually the $L_1$, $L_2$ or $L_\infty$ norm, and the approximation problem would be straightforward. Here the problem is a bit different, we are trying to find the line which have the minimum amount of points of a different class on the wrong side. This informal requirement could easily be made a formal one, but it would make up a very difficult approximation problem, in a non-metric space, and over-adaptation could arise again. It could for instance be interesting to measure the distance from the nearest object to the line and try to maximize the distance to that point, or one could try to maximize the distance to the class mean. Temporarily we could conclude that there is no longer trivial way to do this, since there is no single reference point.
1.2. A PLAY EXAMPLE

<table>
<thead>
<tr>
<th>True class/Classified class</th>
<th>Cancer</th>
<th>Non-cancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>99%</td>
<td>1%</td>
</tr>
<tr>
<td>Non-cancer</td>
<td>2%</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 1.1: Classifier, this time in the form of a medical test

<table>
<thead>
<tr>
<th>True class/Classified class</th>
<th>Cancer</th>
<th>Non-cancer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>99</td>
<td>1</td>
</tr>
<tr>
<td>Non-cancer</td>
<td>20000</td>
<td>980,000</td>
</tr>
</tbody>
</table>

Table 1.2: Approximate absolute values for (mis-)classification in a hypothetical cancer screening

1.2.2 Classifier performance

There is also a problem connected to measuring how well a particular classifier is performing. The performance of a classifier can be well summarized by a so called confusion matrix. This confusion matrix tells us the relationship between an objects true class, and the class which a certain classifier is assigning to it (Table 1.2). In a perfect system, the matrix would be diagonal.

However since we are going to solve an optimization problem it is highly preferable to have a scalar measure of performance (objective/criterion function). In the feature selection problem this will normally be an objective ranging within the interval [0, 100%], and would measure the degree of diagonality of the matrix. The choice of the criterion function will depend on which kind of errors we would most try to avoid.

We will look at another example. Let us pretend we have a military weapons system. It should be able to distinguish between two classes “friend” or “foe”. If the class is “foe” it should aim a weapon and kill the foe, if on the other hand the class is “friend”, it should do nothing. It would be reasonable to argue that shooting a friend is much worse than letting a foe past. Therefore the criterion function should clearly represent this notion.

Another example is a classical example from medical statistics. If we want to screen a population for cancer, we could have a test with the following properties.

If we assume that 1 person in 10000 has cancer the confusion matrix will look approximately like Table 1.2 if we test 1 million people:

As we can read from Table 1.2, even though the percentages looked very promising, the large difference in class sizes makes for a false detection rate of 200 : 1 which should be considered as a big problem.
1.3 Various approaches to pattern recognition

1.3.1 Statistical pattern recognition

Statistical pattern recognition uses elementary quantitative descriptions of objects called features. The combination of these features into a vector, forms what we will call a pattern, or a feature vector. Given N features, this vector can be said to populate an N-dimensional space. And our task becomes the problem of dividing this space into K disjunct regions\(^1\), where K is the number of classes.

The normal way of doing this \([2]\) is to adapt a low-degree polynomial, usually of degree 1 or 2, or if this does not give a good fit, one could use a piecewise polynomial. However no guarantees can be given for the separability of a class (See \[2.3\] on page 15). Separability may also be possible, but only with functions that require a high number of parameters to be estimated, in which case this will affect the robustness of the classifier. This is somewhat related problem to the “curse of dimensionality”-problem discussed later. The resulting function will be a hyper-surface.

Another important issue to remark is what the classifier wants to accomplish. The discrimination hyper-surface can be adapted to several aims. As discussed in 1.2.2, there could for instance be worse to classify one class to another, than the other to the first. This should reflect the placement of the hyper-surface. Or one could choose to have a proportionally higher or lower weight on outliers. This is the goal of classifier design, and is an important problem, which we leave to rest. \([3]\)

1.3.2 Neural networks

Neural nets try to simulate the human learning process, by coupling systems of small “processors”, called neurons. These systems are typically designed by laying a few layers of neurons on top of each other, and assigning random weights to the input functions.

The beauty of these nets are their ability to take almost any input, and transform to almost any output. In nature they work as a nonlinear approximation. However training time might be a problem, and remedies has been taken to both speed, and to slow convergence.

The most common way to use them are to train them using the back-propagation algorithm, by updating their weights when they do not perform correctly.

---

\(^1\)A very difficult subset of this problem arises when you do not know the true class belongings of your objects, and you need to look for clusters in your data. The problem becomes even worse when the number of classes is not known.
1.3. VARIOUS APPROACHES TO PATTERN RECOGNITION

Figure 1.5: A single neuron

Figure 1.6: A multilayer perception net

\[ z = \text{XOR}(x, y) \]
1.3.3 Syntactic pattern recognition

An object need not be specified by individual features, but quite often powerful descriptions can only be accomplished by defining relations. E.g. if we were asked to describe a banana, we would say that it’s a bent tube, with color yellow, and a black point in each end. One sort of apple, might be a spherical object with color yellow, while a tomato is a spherical object with color red.

The point in this example is that it is not the “elongatedness” or the color alone which describes the object. But the combination of these. If we were to find an object with the properties bent tube, with color red. There would be no sense classifying this to some class, we would probably be better of leaving it unclassified.

A syntactic pattern recognizer is in fact a grammar, which is used to classify different object. A problem (or a strength, depending on the view) of syntactic pattern recognizers are that if we are not careful in the design of the substitution rules, we might end up with an inconsistent grammar[4].

1.3.4 Recognition as graph matching

If structures are the essential to match, we might be better off trying to find a structural match. As for our model problem, this might not be an appropriate approach, since color seems to be a very important issue. But we could rather take a look on the problem of matching a face, see figure (1.7). An essential part of a face is to have a a “roundish” contour with eyes and nose forming a rectangle, and a mouth below.

If we’ve got a template, we could construct a graph, consisting of nodes, with attached springs. We could so represent the problem as the problem of stretching the template to become the new figure. For this method to work, it is essential to know the location of a few fixated points, which must be detected by some mean.

This method however is extremely computationally demanding. In its nature its an optimization problem, and if the problem is badly defined, there might not be a unique optimum.

1.3.5 Fuzzy systems

In reality, we often describe the world in vague terms. Back to our model problem, we can describe the banana about in the same way as for a in our semantic model. A banana is a bent, yellow object, with black points in the ends. However, the word bent is not precise. A banana and a cucumber is bent, but so is a horseshoe. And for a machine, the words yellow and black are just the RGB vectors (255,255,0) and (0,0,0). But we would just as easy describe an object with color (200,190,0) as yellow. So if we could find a
1.4 Real life problems

As our play examples have consisted of very few features, there would be no point in trying to reduce their feature set. However if we turn to some real life problems the problem becomes more pressing.

If we find a good set of descriptors, we could hope to get good matches.

\[ f(r) = \begin{cases} 
0 & r \leq 150 \\
(r - 150)/50 & 150 \leq r \leq 200 \\
1 & r \geq 200 
\end{cases} \]

in an RGB system, and the hedge “very” as \[ g(x) = x^2 \]

then very red becomes \[ g(f(r)) = \begin{cases} 
0 & r \leq (150 \\
(r - 150)/50)^2 & 150 \leq r \leq 200 \\
1 & r \geq 200 
\end{cases} \]

Here adverbs means words like “very”, “a little”, “strongly”. You could for instance say that something is “light blue”, or “very red”. At least one software package I’ve seen implements this as functions of the initial qualifier. If we for instance define red as \[ f(r) = \begin{cases} 
0 & r \leq 150 \\
(r - 150)/50 & 150 \leq r \leq 200 \\
1 & r \geq 200 
\end{cases} \]
1.4. REAL LIFE PROBLEMS

1.4.1 Textural classification

In many applications textures are at the heart of the classification problem. Examples include medical imaging, defect and disease classification, ground classification from for instance aerial and satellite imagery.

Without venturing into details there is easy to generate an arbitrarily big set of texture features. For instance a single class of texture features are formed by Gray Level Cooccurrence Matrices (GLCM) and different statistical features derived from them. When calculating these features you are free to choose 3 parameters, namely distance, angle and feature. If you totally disregard the texture size, you could easily formulate $500 \cdot 3 \cdot 20 = 30,000$ features for a $[500 \times 500]$ pixels picture. And this is for one single class of features. Within this class there would be a large subset of irrelevant features, and most likely a large set of highly correlated ones.

This is a clear candidate for Feature Selection, although any reasonable application would probably exclude most of these features before even considering an automated algorithm.

1.4.2 Hyper-spectral sensors

Continuing with satellite imagery, modern satellites supports a wide array of sensors, among these are the Hyper-spectral sensors, at the time of writing these support up to 220 spectral bands (HYPERION). These satellites have a wide range of applications for instance [5] lists 61 applications for Landsat 7. These include “Discriminating vegetative, crop and timber types”, ”Classifying land uses”, ”Determining water boundaries and surface water areas”.

The datasets delivered by a hyper-spectral sensor consists of several
bands of intensity measurements of electromagnetic radiation, defined on a geometry. A measurement is thus an intensity function $I(b, m, n)$, from the band $b$, and the pixel $(m, n)$. Each of the bands are spectral bands, commonly less than $10nm$ wide. Each pixel represents a high resolution discretization of a emmitance spectrum, integrated over the continuous geometry.

Many of the measurements in the sensor bands of these satellites have a high degree of correlation, especially neighboring bands. This lack of independence means that reduction of the number of bands is a natural solution for getting more robust results.
Chapter 2

Feature selection - A problem presentation

2.1 The problem

The problems in 1.4 are only some of many problems that lead to the desire for some magical tool that reduces the amount of features.

In general the problems that warrants reduction of their feature set can be subdivided into two groups, which I choose to call training limited and bandwidth limited.

Training limited problems are problems where one can simply not acquire enough data to reasonably estimate the parameters within the prescribed confidence boundaries. This could be caused by several factors. These are a few, with an associated example of a field of application:

- Obtaining more samples costs too much money (geology)
- More samples require the death of an affected individual (medical)
- More samples require an occurrence beyond our control (astronomy)

Bandwidth limited problems are problems which for a variety of reasons cannot use the entire feature set. These reasons might include:

Real time requirements Usage of the full feature set will require the processing time to supersede the prescribed limits.

Monetary considerations Eg. for a hardware implemented system, usage of the full feature set will require too complex hardware.

Transmission limits If the features are to be exchanged over some medium, this medium might not have the capacity to transmit the entire set.
So what we want to do is to somehow alter the set so that the problems above become smaller or ultimately overcome. One of the tools at our disposal is a “feature selection algorithm”. It is actually quite easy to device such an algorithm, all one has to choose is some kind of function that assesses the “goodness” of the subset, and then test every combination of subsets.

If we have established that this is an easy problem, why all this fuzz? - The answer lays in the computational complexity of the straightforward solution.

We can then do some quick calculations that show that the size of the solution space is:

\[
\sum_{k=1}^{d} \binom{d}{k} = 2^d
\]

where \(d\) and \(k\) are as defined in section 2.2. What we have just established is that the number of features that must be evaluated in explicit enumeration grows as \(O(2^d)\), and that is catastrophic, as we can see from table 2.1.

It is accordingly more than clear that the easy straightforward approach of enumerating every single solution (3.2.1 on page 20), although conceptually easy and straightforward to program, is practically infeasible in terms of its computational complexity.

### 2.2 Mathematical description

**Definition 1** \(\Xi\) is a set of features, with dimension \(d\), and their corresponding known class labels (ground truths).

**Definition 2** \(\phi \in \{0, 1\}^d\) is a selection vector or an incidence vector. It is 1 if a certain feature is selected, or 0 if it is unselected.

**Definition 3** \(k\) is the number of selected features of \(\phi\) or the number of 1s in the vector \(\phi\).

**Definition 4** If \(\Xi\) has \(C^\#\) different classes. We define the set \(C = \{1, \ldots, C^\#\}\) as the class labeling of this particular feature selection problem. And \(c \in C\) as the class of a particular feature.

<table>
<thead>
<tr>
<th>(d)</th>
<th>(2^d)</th>
<th>Order(Short form)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1024</td>
<td>Thousand</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>Million</td>
</tr>
<tr>
<td>30</td>
<td>1073741824</td>
<td>Billion</td>
</tr>
<tr>
<td>100</td>
<td>1267650600228229401496703205376</td>
<td>Nonillion</td>
</tr>
</tbody>
</table>

Table 2.1: Problem size as function of \(N\)
2.2. MATHEMATICAL DESCRIPTION

Definition 5 \( S \) is a set of training samples for a given \( \Xi \).

Definition 6 \( T \) is a set of test samples for a given \( \Xi \).

Definition 7 If \( S \) contains the samples of a training set belonging to a \( \Xi \), then \( S^* \) is the true class labeling of \( S \). \( S \) may be represented as a matrix, if \( S \) contains \( N_S \) feature vectors as column vectors, then the class labeling is an \( N_S \) row vector.

\( S^* \) is a mapping which is considered to be be perfect. This is in most instances a manual review of every single observation, possibly using information unavailable to the system\(^1\).

Definition 8 If \( T \) contains the samples of a test set belonging to a feature selection problem, then \( T^* \), and \( N_T \) may be defined analogous to definition 7.

Definition 9 \( J \) is a performance measure of a classifier, with \( \phi \) as variable, and \( S, S^*, T, T^* \) as parameters.

Then the feature selection problem consists of determining:

Definition 10

\[
\max \xi = J(S, S^*, T, T^*; \phi) \tag{2.2}
\]

\[
\phi_i \in \{0, 1\} \quad \forall i
\]

The below definition is an alternative, but equivalent specification.

Definition 11

\[
\max \xi = J(S, S^*, T, T^*; \phi) \tag{2.3}
\]

\[
\phi \in \{0, 1\}^d = \Phi
\]

For brevity, we will mostly use \( J(\phi) \) as an abbreviation of \( J(S, S^*, T, T^*; \phi) \).

---

\(^1\)A problem that often arises is the fact that it is exceedingly difficult to pinpoint which factors we use to determine that something is belonging to a certain class. This is closely related to Platons ideas. Stated in a bit grandiose way, what we are trying to is to quantitatively describe an idea. As a thought experiment the reader might try to quantitatively describe a happy face, and do so in a way which clearly distinguishes it from any other mood one may think of.
2.3 Separability

2.3.1 Minimum error classification

If given a two class problem, we could have one single feature, (eg gray level). A very typical problem would be to distinguish foreground from background, eg. in OCR. The goal is then to minimize the classification error, such that as few foreground pixels as possible are classified as background pixels, and vice versa. Each class (foreground and background) then have a corresponding probability distribution \((f)\) and \((b)\). And the problem can then be described as follows:

\[
\inf_t g(t) = \int_{-\infty}^t b(x)dx + \int_t^\infty f(x)dx
\]

assuming of course that the mean density of \(f\) is greater than \(b\). Yet another play example can be seen in 2.1.

This is of course just a problem of thresholds, but never the less this describes a discrimination problem, where the threshold is the discrimination function in a 1-dimensional space.

2.3.2 Separability measures

As a sidetrack it is important to realize that the problem of FS is not only the problem of finding a best feature set, it is also important to find a good separability measure with respect to separating the classes.

The separation of classes is not the same as the distance between classes. In everyday speech one could ask a farmer the following question: “How far from the forest do you your sheep grass?” - The answer would probably be the distance between the rim of the forest and their pasture. In mathematics
2.4. NON-DETERMINISM AND THE CURSE OF DIMENSIONALITY

this would correspond to a minimum-measure. However what we are looking for is how well they separate. If the sheep were grassing on the rim of the forest, and some were in the forest and some outside, we could still claim to have separation, at least partially, while the distance measure would totally fail. And even if the sheep were totally contained within the forest we could say they were near the rim, so even though we could not geographically separate the sheep from the forest, we could still separate large parts of the forest from the sheep.

The design criteria for a separability measure are mainly to keep distances between classes at a maximum (Between class variance), while ensuring that the internal variance (Within class variance) is kept at a minimum. There is no exact solution to this problem, but various criteria have been suggested.

See e.g. [6] for a review of these criteria.

2.4 Non-determinism and the curse of dimensionality

As mentioned earlier we are interested in reducing the dimensionality due to the limited availability of training samples. We will however run into some problems that makes this a bit difficult.

We introduce some new variables:

Definition 12 \( x \) is a single sample of \( \Xi \).

Definition 13 \( \mu \) is the mean of the samples in \( S \).

Definition 14 \( \Sigma \) is the covariance matrix of the samples in \( S \).

For ease of notation, we should also define

Definition 15 \( \exp(y) = e^y \).

2.4.1 Parameter estimation errors

If we are doing feature selection on a problem which has perfectly defined parameters, the problem can be said to be deterministic. However few real world problems are. Parameters have to be estimated, and uncertainty is introduced. What this means for us is that if we measured a set of features once we would get one set of values, and an optimum could be found. If we went out and did our measurements again they would most likely yield slightly different values, which might have affected the optimum. This is characteristic of a non-deterministic problem, where the problem is to estimate the parameters correctly.
We very often assume a multi-normal distribution, if the distribution is $d$-dimensional, it is given by:

$$p(x) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

if we are to estimate the parameter of the distribution, we have to estimate the class means($\mu$), which is a vector of $d$ elements, and the covariance matrix $\Sigma$ which is a symmetric $d \times d$ matrix. The fact that it is symmetric means that we have to estimate only $\sum_{i=1}^{d} i = \frac{d^2 + d}{2}$ elements, of the $d^2$ elements in total. However this is still quite catastrophic. If we assume that we need 10 samples to reasonably estimate one parameter this means that we would need $n = 10\frac{d^2 + 3d}{2}$ of each class to do the estimation. For $d = 30$ this means that we need 9000 samples. Under the assumption of normality of the feature distribution, we could however still do arbitrarily bad (but with low probability). This is the “Curse of dimensionality” [7].

2.4.2 Separation of stages

It is not enough for the feature set to perform well for one set of data. For a set to be good it needs to perform consistently well for several sample-sets. This introduces the need to separate the training and testing stages of the procedure. So what we would like to do is to separate the samples we use to train the classifier from where we test it’s performance. This has to be done in such a way that we can assume that the samples are independent in distribution.

The usual way to do this is to either use two separately collected datasets, if such are available or to subdivide one dataset into two, by means of random drawing.

Further on, the direct adaptation to a certain problem can in every case be done arbitrarily well, provided no two points from different classes exist at the exact same location. But by restricting ourselves to reasonable classifiers\(^2\) we can almost always make very good adaptations which performs very well on the case at hand, but significantly worse on a random sampling of new data.

As a solution we can once again make several subdivisions of the training set into a crossvalidation-training set(CVTS) and a validation set. If this is done in a clever way, we will probably once again over-adapt, but hopefully not so much that the performance degradation is significant on new data.

\(^2\)It is reasonable to expect a classifier to be fairly smooth. As in containing no abrupt changes in curvature nor high variation [8]. Since such changes would most likely mean an over-adaptation.
Chapter 3

Discrete optimization

Consider the following problems:

1. A home delivery service has a certain number of cars, with a certain capacity. How should they distribute their packages among the cars, in order to minimize the distance traveled?

2. A boss at a company has a certain number of hands capable of doing different tasks on a project. Some of the tasks depend on others to be completed. How should he assign the tasks, in order to get the entire project done as fast as possible?

3. A carpenter needs different lengths of planks, he has at his disposal a number of planks but they may not necessarily be of the same length as he needs. How does he cut the planks to minimize waste?

4. A dating service has a number of applicants. Some boys and some girls. They have managed to find a number for the desirability of all the girls for the boys, and vice versa. How should they assign dating partners in order to maximize the total happiness of all their applicants?

All of the above problems are examples of Discrete Optimization Problems. The three first problems are very hard\footnote{These problems are very hard in the sense that there are no exact non-exponential time algorithms known for them. In fact we could show that they are NP-complete.}, although special cases may be solved to optimality or to a prescribed accuracy. The last is easily solvable by well known algorithms.

The three first above examples are all areas of active research, and with enormous economic benefits, given an effective algorithm. Numerous other examples exist and many are so essential to some industries they could be a matter of huge margins, or bankruptcy in a pressed market.
3.1 What is a DOP?

A Discrete Optimization Problem deals with objects that can assume only discrete values in the search space. Discrete objects can be characterized by integer values, whereas continuous objects require real numbers[9].

In the delivery problem above, it would not be a sensible solution to assign half a package to one car, and another half to another. This constraint seems quite obvious, and at first glance should not cause any big problems. But as it turns out, this makes all the difference between a problem that is “trivial” and one that is very hard.

3.1.1 Ex: The knapsack problem

In optimization theory a well known problem is the problem of filling a knapsack with a number of items, of different size($s$) and value($v$). The goal is to maximize the total value, while staying within the allowed boundaries provided by the capacities($c$). The knapsack problem can be given as follows:

$$\max \xi = v^T \phi$$

$$s^T \phi < c$$

$$\phi_i \in \{0, 1\} \quad \forall i$$

3.1.2 Complexity theory

In computer science, there is a lot of talk about efficiency. One want an effective algorithm for doing a task. But what is really an efficient algorithm?

The best measure of efficiency is running time. However due to differences in machine architecture running times differ in a non-linear way. So this is not a very good measure.

Another way to look at this is to count the number of primitive operations used, before terminating, and express this as a function of some variable(s) say $n$. So we might have an algorithm with the running time $\frac{1}{4}n^3+4n^2-n+4$. As $n$ grows, we can easily see that $n^3$ becomes the dominant term. We therefore say that the algorithm in question has a running time of $O(n^3)$ called Big-$O$ notation. See [10] for a nice summary of algorithm efficiency.

Many of the $DOP$ problems that occur are what is called $NP$-complete [11] and are only known to be solvable by full enumeration. (See 3.2.1) Full enumeration has a running time of at least $O(2^n)$ also known as exponential running time. Meaning that increasing $n$ by only 1 means doubling the running time.

It is important to notice that even though some algorithm has a better running time in Big-$O$ it is not always the best choice for small $n$. For instance Intel has shown that the best known implementation for inversion
3.2. Exact methods

Exact methods are as the name suggests methods that guarantee to find the exact optimal solution. They are unfortunately however generally plagued with problems with computation time.

3.2.1 Full enumeration

Full enumeration means looking at every possible point/permutation in the search space. The beauty of this algorithm is that it is relatively easy to implement, but for only moderately big $n$ most problems become practically unsolvable due to the immense computational time required.

Figure 3.2.1 shows the running time for evaluating every possible point of the $\{0, 1\}^d$ space for different $d$ doing absolutely nothing in each evaluation. About the first 30 measurements are very prone to measurement errors.
3.3. HEURISTIC TECHNIQUES

3.2.2 Branch and Bound

Branch and Bound also known as implicit enumeration due to the fact that every point in the search space is implicitly checked for a maximum. This algorithm is one of the few algorithms which comes with a guarantee. The algorithm finds the optimum. The problem is that the algorithm is worst case exponential in time\[14\]. However the most significant impediment of this algorithm is its requirement of the feature selection criterion to be monotone, and that the criterion values must be correctly estimated.

If we first define the union

**Definition 16** If \( \phi_1, \phi_2 \in \Phi \) we define \( \phi = \phi_1 \cup \phi_2 \) to be 0 at element \( i \) if both \( \phi_1 \) and \( \phi_2 \) has a 0 at position \( i \), else \( \phi \) is 1 at position \( i \)

we may specify the monotonicity property as follows

**Definition 17**

\[
J(\phi_1 \cup \phi_2) \geq J(\phi_1) \quad \forall \phi_1, \phi_2 \subseteq \Phi
\]

3.3 Heuristic techniques

Heuristic techniques are, contrary to exact methods, methods that come without any guarantee of finding an optimal solution. They are however designed in a hopefully clever way to do their best to approximate the optimum.

3.3.1 Local search

The idea of a local search is to travel through the search space of a DOP, using only local operations, or neighborhood operations. This is possible if one or more neighborhood relations are defined on a problem. In the feature selection problem such a neighborhood may be something as easy as two selections, only differing by the addition of one feature to one of the vectors compared to the other. Or we may define two selections as being neighbors if exactly one feature is removed from one, and exactly one other is added. Wider neighborhoods may also be chosen.

When a local search starts at an arbitrary point, it utilizes the information about its neighbors to make improving moves within the neighborhood as long as it is possible. The move may not necessarily be the best improving move, but must be an improvement. When no further improving moves may be found the local search has converged, and we are at a local optimum.
3.3.2 Metaheuristics

The Free Online Dictionary of Computing [15] states that a meta heuristic is:

A top-level general strategy which guides other {heuristics} to search for feasible solutions in domains where the task is hard.

From the preface of Handbook of Metaheuristics [16] we can read:

*Metaheuristics*, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build and destroy solutions in constructive and destructive processes.

I will also make my own attempt:

*Metaheuristics are a class of algorithms that uses knowledge of the problem topology in order to move from one place in the search-space to another place in hopefully an intelligent manner. Common to them all are the fact that they use more or less local information to decide where to go next. This use of local information, while maybe carefully devised, and often effective, can in general not guarantee that we find the best solution to our problem. In fact we can only hope that we are in the vicinity of a good solution.*

In contrast to the straightforward solution we devised earlier, these algorithms can easily become quite involved.

It is a goal for a good metaheuristic to employ a good mixture of intensification and diversification strategies in order for the search to gain a balance of seeking an optimum, and trying to avoid getting stuck in a local optima. One could argue that they are quite different, but personally I think the similarities of these algorithms are more striking than their differences.

3.3.3 Traditional applications of metaheuristics to the Feature Selection Problem

Jain and Zongker [17] has made an extensive taxonomy of the feature selection methods currently in use. I will mention some of them here.
3.3. HEURISTIC TECHNIQUES

Best single performing feature

This is the simplest approach one could take, and might not even be called a heuristic. It is a greedy construction heuristic. Simply pick a list of the single best performing features, independently of the others. There is nothing that hinders this solution in giving optimal performance, though in general it under-performs greatly.

Forward and backward selection

Probably the best known feature selection algorithm, is the forward selection algorithm (FSA). It works by starting with an empty solution, and adding \( l \) features at the time, where \( l \) is a parameter that can be adjusted.

This is essentially a greedy algorithm, and while increasing the size of the add step \(^2 l\), will help us in not climbing the flattest local maximum. Once we have embarked on a hill climb, we can never escape.

The backward selection algorithm (BSA) is essentially the same algorithm as the FSA, with the difference that we start with a full vector, and remove the least useful items.

If we want to keep \( k \) features out of \( d \), it is easy to show that FSA beats BSA when \( k < d/2 \), and BSA beats FSA when \( k > d/2 \) in terms of convergence speed.

Adaptive floating search

Somol et al [18] summarizes the different floating search methods. The basic approach of floating search methods is to use some combination of forward and backward selection. One example is to add \( l \) features at a time, but when you have added \( l \) features, you check if there exists a better solution than your previous by removing another set of \( r \) features.

Simulated Annealing

The quite well known method of Simulated Annealing uses the analogy of cooling down an alloy. When you cool it down too fast, it settles at a high energy level, with quite big internal strains, while cooling it down slowly hopes to ensure that the alloy has much less internal strain.

On a defined neighborhood, it explores the neighborhood in a random order. If a move deteriorates the objective, it may accept it, by using an ever decreasing probability of acceptance. If it improves the objective, it will always accept it. If the probability of accepting a deteriorating move reaches 0, the SA process degenerates into a local search.

\(^2\text{Add step} \) should be read as the number of features we are looking at every time we increase the inclusion vector.
3.3. HEURISTIC TECHNIQUES

The way the temperature decreases is known as the annealing schedule. How this is set up is very important for the convergence speed, and the quality of the solution. In fact it has been proven that certain annealing schedules, the probability of a globally optimal solution approaches 100%. This schedule unfortunately has terrible convergence properties, and is in fact worse than full enumeration.

Genetic Algorithms

A genetic algorithm is a population based evolutionary algorithm. It is based on the Darwinian evolutionary theory. By cross breeding “good” parents, they most probably have a set of good genes, if these genes are combined, the children hopefully has even more favorable genes than their parents. Unfortunately, or fortunately depending on the results, genes are subject to mutation, which means that a child has most probably not a strict combination of their parents genes, but rather a subtly different set.

This principle can be traded upon also in the realm of Discrete Optimization. We do much the same, we seed a population with random individuals. The individuals are then evaluated, and only the fittest survive. They get to breed and create offspring, subject to mutation, when the breeding process is finished, a new generation has formed. And the process may be repeated. Often multiple populations are used, and ever so often some kind of cross-breeding process takes place.

3.3.4 More metaheuristics

There is also a selection of algorithms known from other optimization problems, that as far as I know have not been applied to the feature selection problem before. See [16] for an exhaustive list. I will list the one that forms a basis for my own work.

Scatter search

A scatter search is designed to work on a set of points called reference points. It operates on these points by generating combinations of these points in an intelligent manner. It may work on an altered objective function that seeks out other criteria than merely a best as possible objective. Such other criteria may include diversity, etc...

The basis for the algorithm are these five steps:

1. Diversification generation method. A method that seeks to visit very different parts of a search space, and generate so-called seed solutions.

2. Improvement method, a method that seeks to improve an already existing seed.
3. Reference set update method, seeks to maintain a set of the “best” solutions.

4. Subset generation method is a method that produces subsets as basis for creating new solutions.

5. Solution combination method transforms a given subset from a subset generation method into a new set of combined solutions.

**Path relinking**

Given two good solutions, often referred to as elite solutions. One could imagine that there is some common attribute that is contributing to the goodness of such a solution. If one could find a path between these two, it’s plausible that one would also encounter good solutions along the way. This is the basis of the Path relinking strategy.

The path relinking strategy also fits very well within the framework mentioned as scatter search as a solution combination method.

The strength of a path relinking strategy is that it is not guided by the boundaries of local optima. If the candidates for relinking is sufficiently spread out in the search space, new optimas may be reached which are closed off to simple neighborhood operators.

**Tabu search**

A tabu search is an umbrella strategy more than a specific strategy which we classify algorithms. It’s inspired by mathematical optimization, and the main aspect is it’s use of a memory called a tabu list in order to avoid cycling into earlier explored areas for a period of time. Even though aspiration criteria are used to allow us to move into the tabu areas if something is tempting enough.

It is based around a local search, and just as local search requires a neighborhood structure to be defined. The quality of local optima avoidance is largely determined by the size of the memory as well as what the memory remembers. The memory may for instance contain entire solutions, or only part of them.

**Guided local search**

Guided local search is a strategy that sits on top of some local search technique. The basic idea is that when a local search gets stuck, we will punish some part of the solution, by assigning penalties to this part of the search space. This means defining an augmented objective function,
The most important feature of GLS, is however how the penalization is performed, this is done in a very ordered way, that ensures diversity in the search space.

Iterated local search
Iterated local search is perhaps one of the simplest true metaheuristics.

It is based on the idea that we will perform a local search until no further improvement is possible. When such is the case, we will attempt to perturb the solution, and restart the local search from the perturbed solution. This local search will continue and hopefully we will reach a new and if we are lucky better “basin of attraction”.

Variable neighborhood descent
An optimum is relative to the definition of a neighborhood, if we were to systematically redefine our notion of neighborhood. We might be in the situation where an optimal solution under one neighborhood is not optimal under another. However a globally optimal solution should be optimal under all neighborhoods.

By varying the notion of neighborhood, we may therefore using only local-search be able to move closer to the global optimum, when we would else get stuck in a local optimum.

3.4 Relaxation
A relaxation is a method that relaxes a strict requirement, and substitutes it for another requirement more easily handled. Alternatively drops it completely.

3.4.1 LP problem
A linear programming(LP) problem is a problem that can be formulated using only linear constraints and with a linear objective function. In [19] Vanderbei defines the following to be a standard form of an LP problem

\[
\begin{align*}
\max \quad & \xi = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \\
\text{subject to} \quad & a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \leq b_1 \\
& a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \leq b_2 \\
& \quad \vdots \\
& a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \leq b_m
\end{align*}
\]
or equivalently on matrix form

$$\max \quad \xi = c^T x$$
subject to $$Ax \leq b$$

Every LP problem is reducible to this standard form.

### 3.4.2 LP-relaxation

The LP-relaxation of a MIP is what we have if we drop the requirement that a variable should take on a discrete set of values.

Once we drop the requirement, we are in a situation where we may solve the problem, using one of the many known solvers for LP-problems. We must however later check that the discrete requirements are fulfilled, which is generally not true, except for some special cases. (Eg. problems with totally unimodular matrix specifications.)

If this is not true, we may start a Branch and Bound type process, where we fixate a single non set variable to a variable within the set.

### 3.4.3 Lagrangian relaxation

Given an LP problem $$x \in \mathbb{R}^n$$ and $$A \in \mathbb{R}^{m,n}$$ on the following form:

$$\max \quad c^T x$$
$$s.t.$$
$$A x \leq b$$

If we split the constraints in $$A$$ such that $$A_1 \in \mathbb{R}^{m_1,n}, A_2 \in \mathbb{R}^{m_2,n}$$ and $$m_1 + m_2 = m$$ we may write the system:

$$\max \quad c^T x$$
$$s.t.$$
$$A_1 x \leq b_1$$
$$A_2 x \leq b_2$$

We may introduce the constraint (2) into the objective:

$$\max \quad c^T x + \lambda^T (b_2 - A_2 x)$$
$$s.t.$$
$$A_1 x \leq b_1$$
If we let $\lambda = (\lambda_1, \ldots, \lambda_m)$ be nonnegative weights, we get penalized if we violate the constraint (2) on the other hand if we want to maintain a linear objective, we may see that we are rewarded for satisfying the objective strictly. The above system is called the Lagrangian Relaxation of our original problem.

3.5 Fast evaluation

3.5.1 Delta-encoding

When two solutions to a problem are quite similar, and the time requirements for computing them is quite long. There might be a possibility to somehow compute the difference between them. A very simple example is the computation of the estimated mean:

$$\hat{\mu}_1 = \frac{1}{N} \sum_{i=1}^{N} x_i$$  \hspace{1cm} (3.1)

Obviously if we obtain one new sample it can be computed using

$$\hat{\mu} = \frac{1}{N + 1} \sum_{i=1}^{N+1} x_i$$  \hspace{1cm} (3.2)

but by noting that

$$\hat{\mu}_2 = \frac{1}{N + 1} \sum_{i=1}^{N+1} x_i = \frac{N\hat{\mu}_1 + x_{N+1}}{N + 1} = \frac{N}{N + 1} \hat{\mu}_1 + \frac{1}{N + 1} x_{N+1}$$  \hspace{1cm} (3.3)

We can see that (3.1) requires $O(N)$ operations to compute and similarly (3.2) requires $O(N + 1)$ operations to compute.

But if we have already invested the time required to compute $\hat{\mu}_1$ we may compute $\hat{\mu}_2$ using only $O(1)$ operations using equation (3.3). A huge saving, if $N$ is big enough.

The same principle may often be used when evaluating very similar objective functions in DOP. This principle may for instance be used as described in A.2.3
Chapter 4

Previous work

Pattern analysis is a relatively new field of research. The *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE* began publishing in January 1979, and according to [20] there has been approximately 350 papers on pattern recognition in the publication between 1979 and 2000, where about 300 of these where on the statistic approach, out of these about 50 were on dimensionality reduction and 15 on the *Curse of Dimensionality*. These are of course not all the articles in this field, but it gives us an idea of the size of this research field.

One of the earliest and most cited article in the feature selection literature is Hughes article “On the Mean Accuracy of Statistical Pattern Recognizers” [7] where he analyzes one of the most important reasons for dimensionality reduction namely the “Curse of Dimensionality” [21]. Which treats the problem of over-adaptation to a particular sampling.

Although the selection of articles is relatively low several strategies for feature selection has been proposed. In one end of the specter lay strategies such as exhaustive search. This strategy has exponential running time [6],[17].

For classifiers($J$) satisfying the monotonicity property (which is equivalent to Definition 17 on page 21)

$$\phi_1 \subseteq \phi_2 \Rightarrow J(\phi_1) \leq J(\phi_2)$$

however the Branch and Bound algorithm proposed by Narendra and Fukunga [14], can be a better choice, due to its better average running time\(^1\). Several strategies have been proposed in order to improve upon the Branch and Bound algorithm [22],[23],[24],[25],[26]. The predictor used by Somol et al [27] shows promising results which can be exploited further. Their algorithm shows promising results for feature sets of 40 or less, but it can not be expected to deliver results within reasonable time limits for very much

\(^1\)It should be remarked though that the worst case is slightly worse, but is seldom seen.
larger feature sets. In their own article they need about 10,000 evaluations for selecting 28 out of 40 features. Which is a moderately big problem, but modest compared to problems with several thousand features.

At the other end of the specter lays the heuristics. One of the first proposed algorithms was the sequential forward search and the sequential backward search. \cite{28}. These methods are some of the simplest methods available. They are essentially greedy heuristics, and they suffer from the general problem with greedy heuristics. Once a local mode is entered, a feature is added or removed, the choice may never be reverted. A more complex approach is the plus-l-minus-r method \cite{29}. Which allows for every $l$ features added that $r$ may be removed. There is however no good criteria available for the choice of $l$ and $r$.

Some of the problems with SFS and SBS are treated in a smart way by Mao \cite{30}, with his Orthogonal Forward selection and Backward Elimination Algorithms. His algorithm performs well in comparison to the SFS and SBS, but the performance is not easily evaluated since he does not evaluate it against any other algorithms than the SFS,SBS and corresponding (l-r) algorithms.

Pudil et al. \cite{31},\cite{18},\cite{32}, have also improved upon the (l-r) algorithm with their floating and oscillating search techniques.

A similar but never the less different approach is attempted by Serpico et al. \cite{33} where they use “n-opt” neighborhoods.

The well known Genetic Algorithms have also been explored \cite{34} and Simulated Annealing \cite{35}.

Perhaps one of the most extensive surveys made are \cite{17}, \cite{20} Jain et al summarizes the state of feature selection as of 2000 (Table 4.1). A few articles have been released later, but they are in general just improvements on already known methods.
Table 4.1: Feature Selection Methods (adapted from [20], a closer discussion of these methods are available in Section 3.3.3 on page 22)

<table>
<thead>
<tr>
<th>Method</th>
<th>Property</th>
<th>Comments</th>
<th>Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustive search</td>
<td>Evaluate all ( \binom{d}{k} ) points</td>
<td>Guarantees to find the optimal subset. Not feasible even for moderate values of ( d ) and ( k )</td>
<td>Explicit enumeration</td>
</tr>
<tr>
<td>Branch and Bound search</td>
<td>Uses the well-known branch and bound search method; only a fraction of all possible feature subsets need to be enumerated to find the optimal subset.</td>
<td>Guaranteed to find the optimal subset provided the criterion function satisfies the monotonicity property; the worst case complexity of this algorithm is exponential.</td>
<td>Implicit enumeration</td>
</tr>
<tr>
<td>Best Individual Feature</td>
<td>Evaluate all the ( d ) features individually; select the best ( k ) individual features.</td>
<td>Computationally simple; not likely to lead to an optimal subset.</td>
<td>Greedy construction heuristic</td>
</tr>
<tr>
<td>Sequential Forward Selection (SFS)</td>
<td>Select the best single feature and then add one feature at a time which in combination with the selected features maximizes the criterion function.</td>
<td>Once a feature is retained, it cannot be discarded; computationally attractive since to select a subset of size 2, it examines only ( d - 1 ) possible subsets.</td>
<td>Greedy construction heuristic</td>
</tr>
<tr>
<td>Sequential Backward Search (SBS)</td>
<td>Start with all the ( d ) features and successively delete one feature at a time.</td>
<td>Once a feature is deleted, it cannot be brought back into the optimal subset; requires more computation than sequential forward selection.</td>
<td>Greedy construction heuristic</td>
</tr>
<tr>
<td>“Plus l-take away r” selection</td>
<td>First enlarge the feature subset and successively delete one feature at a time.</td>
<td>Avoids the problem of feature subset “nesting” encountered in SFS and SBS methods; need to select values of ( l ) and ( r(l &gt; r) )</td>
<td>Greedy construction heuristic</td>
</tr>
<tr>
<td>Sequential Forward Floating Search and Sequential Backward Floating Search</td>
<td>A generalization of “plus l-take away r” methods; the values of ( l ) and ( r ) are determined automatically and update dynamically.</td>
<td>Provides close to optimal solution at an affordable computational cost.</td>
<td>Local search</td>
</tr>
<tr>
<td>Genetic Algorithms</td>
<td>Population based method; starts with a random selection of “chromosomes” and “breeds” them, in order to attain an optimal “breed”</td>
<td>Highly parallelizable, often attains good solutions, but may converge very slowly. GA may also converge prematurely.</td>
<td>Population based meta-heuristic</td>
</tr>
</tbody>
</table>
Chapter 5

Analysis of the Feature Selection Problem

As we recall from chapter 2.2 on page 13 we may write the Feature Selection Problem as follows:

$$\max \xi = J(S, S^*, T, T^*; \phi)$$

$$x_i \in \{0, 1\} \quad \forall i$$

with an equivalent vector form formulation:

$$\max \xi = J(S, S^*, T, T^*; \phi)$$

$$x \in \{0, 1\}^d$$

5.1 Relaxation

The process known as relaxation (see 3.4.1 on page 27) is often applied to the knapsack problem, mentioned above. We will use a variation of Lagrangian relaxation in such a way that if we violate the restrictions it will severely impair the “goodness” of the objective function.

5.1.1 The knapsack problem relaxed

A relaxed formulation of the knapsack problem is given below:

$$\max \xi = v^T \phi - \gamma(c - s^T \phi)$$

$$\phi \in \{0, 1\}^d$$

where $\gamma(y)$ is a function which is zero when $y$ is positive, and sharply increasing for $y$ negative.
5.1.2 Feature Selection Problem for Hyper-spectral imaging

In the introduction we discussed Hyper-spectral imaging, and the fact that many of their bands are highly corellated. This fact leads us into great difficulties when we are trying to find an optimal solution. Since our features are so strongly correlated, it’s difficult to find any single path to a well performing set. On the other hand the fact that they are so closely related lead us to believe that an optimal set of solutions, if certain features where forced to be included, should be close in cardinality to another set where a slightly different set was forced to be included.

Remembering Definition 3 on page 13 we may define $k^*$

**Definition 18** Let $\phi^*$ denote an optimal $\phi$, then $k^*$ is the corresponding $k$ to such a $\phi^*$

If we use the facts above, we can state a new formulation of the feature selection problem:

$$
\max \xi = J(S, S^*, T, T^*; \phi) \\
|k| = k^* \\
\phi \in \Phi
$$

(5.1)

It is worth to take note that this statement of the feature selection problem is identical to the first if all optimal solutions maintain the same cardinality. If this is not the case, this statement will have a set of solutions with identical value $\xi$, but the solution set will be a subset of the original solution set.

This is really not a problem, since what we search is a single optimal solution, and not the set of all optimal solutions.

5.1.3 Yet another restatement of the feature selection problem

We can easily now produce another restatement of the feature selection problem

$$
\max \xi = J(S, S^*, T, T^*; \phi) \\
\vec{1}^T \phi = k^* \\
\phi \in \Phi
$$

where $\vec{1} = (1, 1, \ldots, 1)^T$ is a vector which contains only ones. Please note the close resemblance to the original statement of the Knapsack problem.
5.2. EVALUATION OF OBJECT FUNCTIONS FOR THE FEATURE SELECTION PROBLEM

5.1.4 Relaxed feature selection problem
If we once again apply the trick of relaxation, we can reformulate the problem as follows:

\[
\max_{\phi \in \Phi} \xi = J(\phi) - \gamma(k^* - \phi^T \bar{1})
\]

This problem is identical to the relaxed knapsack problem, with exception that \( J \) is clearly not a linear function, and \( k^* \) is not a known parameter. However the problem should keep much of the characteristics of the knapsack problem, in the sense that the function should be mostly concave when removing one feature at the time.

5.2 Evaluation of object functions for the feature selection problem
One of the most difficult challenges when it comes to the feature selection problem is the fact that evaluation of the object function \( \xi \) is very time consuming. Tests I’ve performed suggests that direct enumeration is tractable\(^1\) for problems up to around 17 elements. Of course this is highly dependent on the number of samples(\(N\)) and the dimensionality(\(d\)), but it once again indicates that this is a very hard problem.

However problem sizes as small as 20 seems to become very hard, when no optimizations are taken with regard to the object function.

This suggests that certain optimizations should be considered with respect to the calculation of the object function.

5.2.1 Precalculation of relative constants
Many of the possible object functions are at least partially based on the covariance matrix \( \Sigma \), or some sub-matrix. One optimization that saves us huge amounts of time is to precalculate this, and choose only the required sub-matrix, when needed.

Close consideration should also be taken to understand if there are other constants that could be precalculated.

5.2.2 Prediction of performance
Another place to try to optimize is through the use of predictors. If one could easily determine some approximate performance for the removal or addition

\(^1\)Tractable is here understood as having running time of about one hour.
5.2. EVALUATION OF OBJECT FUNCTIONS FOR THE FEATURE SELECTION PROBLEM

of a specific feature, the algorithm should use this as a guideline, before deciding where to proceed in the search space. When the move is actually performed the algorithm must evaluate the real value for the specific case. This has the advantage of speeding up the process, but could easily lead to convergence along the lines of the “Single Best Feature” strategy.
Chapter 6

Methods for the Feature Selection Problem

In this chapter we will discuss actual implementations of the methods discussed earlier, and some new methods will be introduced.

6.1 General techniques

We will first look at a few techniques that can successfully be applied to improve many of the techniques below.

The technique mentioned earlier in 3.3.4 on page 25 as tabu search utilizes tabu lists, they are meant to preclude cycling over a set of solutions. This technique has been used effectively in many of the algorithms below, as it seems only to have positive effects on the performance of an algorithm.

Another technique is elite lists. We simply keep a list of the best solutions encountered up to date, they may later be used for other purposes.

It is always implicitly understood that the best solution ever encountered will be kept, regardless of whether it also is stored in an elite list.

6.2 Moves

Most of the algorithms that will be presented later are based on the notion of a neighborhood, although not necessarily a very local neighborhood. These moves define the basic operations used to move from one solution to the next, for the algorithms presented later.

Definition 19 A move is a specific choice of a relocation within a neighborhood or the choice not to relocate.

Definition 20 A move referred to as a neighborhood should be though of as the neighborhood over which it is defined.
6.2. MOVES

6.2.1 Basic moves

We define 5 basic moves.

Nomove does nothing.

Add\_l adds exactly \( l \) features, picks the best move, unless prohibited by a tabu-list.

Rem\_l removes exactly \( l \) features, picks the best move, unless prohibited by a tabu-list.

Opt\_l exchanges exactly \( l \) features, for \( l \) other features, picks the best move, unless prohibited by a tabu-list.

Flip\_l flips the selection status of exactly \( l \) features, picks the best move, unless prohibited by a tabu-list.

The 4 last moves are obvious, the Nomove move simply hides an evaluation of the current position.

For the last four moves we define their corresponding full moves, as a union of all the moves of lesser \( l \):

\( f\text{Add}_\_l \) picks the best move among all \( \text{Add}_\_l \) moves less than or equal to \( l \).

\( f\text{Rem}_\_l \) picks the best move among all \( \text{Rem}_\_l \) moves less than or equal to \( l \).

\( f\text{Opt}_\_l \) picks the best move among all \( \text{Opt}_\_l \) moves less than or equal to \( l \).

\( f\text{Flip}_\_l \) picks the best move among all \( \text{Flip}_\_l \) moves less than or equal to \( l \).

6.2.2 Probabilistic moves

Exploration of full neighborhoods are often very expensive, or simply not demanded by an algorithm. We define the following simple probabilistic moves:

\( p\text{Add}_\_l \) picks the best move among \( n \) randomly selected elements in an \( \text{add}_\_l \) neighborhood.

\( p\text{Add}_\_l \) picks the best move among \( n \) randomly selected elements in an \( \text{rem}_\_l \) neighborhood.

\( p\text{Add}_\_l \) picks the best move among \( n \) randomly selected elements in an \( \text{opt}_\_l \) neighborhood.

\( p\text{Flip}_\_l \) picks the best move among \( n \) randomly flipped elements in a \( \text{Flip}_\_l \) neighborhood.
pFlip tries to flip 1 randomly selected feature of the original position, max \( n \) times. Always pick an improvement over the current position, but with probability \( p \) accepts a deterioration.

We also define some moves over less local neighborhoods:

- randomSelect picks an arbitrary position in the search space.
- guidedRandomSelect picks the best move among \( n \) randomly selected elements of size \( k \) for all possible \( k \).
- relink picks a random shortest path in add_1 and rem_1 neighborhoods. Between current position and a destination.
- pMutateMove flips each feature of the original position with probability \( m \), \( m \) max \( n \) times. Always pick an improvement over the current position, but with probability \( p \) accepts a deterioration.

6.3 Metaheuristics

There are a large number of existing metaheuristics, and many of them carry with them good ideas that can be incorporated with great success in many other metaheuristics. Within the heuristics we may also vary parameters and moves. Unfortunately all of these ideas cannot be tested in combination with one another. An interesting thing though is that the testing of combination of ideas is in itself might be formulated as a combinatorial optimization problem.

Nevertheless, we will look at a few algorithms here. We will once again start with the traditional approaches.

6.3.1 Traditional algorithms

Looking at listing 6.1 on the next page we see that the algorithm is very much traditional GA, except for the fact that we nest populations in a very non-standard way. This makes the algorithm scale well.

The Add-l-remove approach is also a standard approach.

The SA algorithm can be seen in 6.3 on page 40 it is defined with a flip_1 move. Finally while not a proper heuristic, we do a simple randomSearch to establish a minimum level of performance see listing 6.4 on page 41.

6.3.2 Iterated local search

To the best knowledge of the author, no specific article has been published on the subject of iterated local search with respect to the FS problem. It is however surprising if similar ideas have not been explored before. The
6.3. METAHEURISTICS

Listing 6.1: Genetic Algorithm

//Declaration of variables
CF:= Classifier
testSet:= Testset
stopCond:= Stop condition
mRate:= Mutation chance for breeding stage
mRate:= Decimation rate
tpSize:= Total population size
gSize:= Group size
ipl:= Iterations per level

function GA(CF, testSet, stopCond, mRate, dRate, tpSize, gSize, ipl)

generate randomly global population with subpopulations of size gSize, so that total population size equals tpSize

for each level do ipl nested iterations if not stopCond:
for each lowest level group:
  evaluate fitness
  decimate the population to dRate of it's original size, keeping only the fittest until population reaches gSize:
    choose randomly among the survivors, couples that get to breed a child
    mutate that child
  end
end

if a nested iteration has reached it's end
  let the best element in this level be introduced on all sub levels
end if
end for

return best

end function
6.3. METAHEURISTICS

Listing 6.2: Add-l-remove-r search

//Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>
l:=<max degree of add neighborhood>
r:=<max degree of rem neighborhood>

function l_add_r_remove(CF,testSet,stopCond,l,r)
    keep track of best at each level
    move to Nomove
    while not stopCond:
        move to f_add
        if no move performed: //e.g., we have d features selected
            return best
        end if
        do loop:
            movePerformed:=false
            for i=1:r
                if Rem_i is better than best at level k-i:
                    move to Rem_i
                    movePerformed:=true
                end if
            end for
            while movePerformed loop again
        end while
    end while
end function

Listing 6.3: Simulated annealing 2

//Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>
temp:=<initial temperature>
coolDownRate:=<temperature multiplier at each time step>
max_iter:=<maximum number of iterations without any move>

function SA2(CF,testSet,stopCond,temp,coolDownRate,max_iter)
    move to randomSelect
    while not stopCondition and move performed:
        move to p_flip_move(p=temp,n=max_iter);
    end while
end function
Listing 6.4: Random search

```c
//Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>

function randomSearch(CF, testSet,stopCond)
    while not stopCond:
        move to randomSelect
    end while
    return best
end function
```

algorithm in listing 6.5 on the following page is based on the ideas found in [16]. We keep a list of the local minima visited. All the time it is possible to use local search for moving us to a better solution, we will do so. When a local minima is reached, we want to relocate, in order to hopefully reach a better optimum. However we want to make the relocation move as small as possible in order to keep the attractive features of the current solution, while not so small that we reacquire the same local basin of attraction. We therefore start by making quite small moves, and if we require a minima already visited, we first recognize that this might be acceptable if it happens a few times. The random flip might after all just have been a very unfortunate one with respect to the goal of relocating us within the search space. If this happens more than a few times (acceptMax) we are probably in a situation where the relocation move is simply not big enough, so we increase the size of the movement.

6.3.3 Variable Neighborhood Search

A simple approach is to try to increase the size of a neighborhood, as we encounter local minima. The approach in listing 6.6 on the next page does exactly that. A fundamental problem with combinatorial optimization problems are that the size of the neighborhoods blow up exponentially with the size of the problem. For this very reason, a choice of neighborhoods may be something like \{flip_1,flip_2,pFlip_3(n=100),pFlip_4(n=100)\}, where the last two are partially explored neighborhoods, due to their potentially enormous size.

6.3.4 Roaming Search

The drawbacks of using the VNS approach is that big neighborhoods may not be explored due to their sheer size. We therefore introduce the roaming search in listing 6.7 on page 43
Listing 6.5: Iterated local search

// Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>
acceptMax:=<Number of revisits allowed at a local minima, before increasing flip order>

function ILS(CF, testSet, stopCond, acceptMax):
    keep tabu−list with some tenure
    k:=4 //Rather arbitrary
    c:=0

    while not stopCond:
        perform move
        if no move:
            if currentPosition in tabu−list
                c:=c+1
                if c>|acceptMax|
                    c:=0
                    k:=k+1
                end if
            end if
        insert currentPosition in tabu−list
        move to pFlip_l(n=1,l=k)
    end if
    end while

    return best
end function

Listing 6.6: Variable Neighborhood Search

// Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>
moves:=<The moves to use>

function VNS(CF, testSet, stopCond, moves)
    while stopCondition is not fulfilled:
        Randomly select a starting point
        for i=1:<number of moves>
            do
                perform move[i]
            while move improving loop again
        end for
    end while
end function
Listing 6.7: Roaming Search

```plaintext
// Declaration of variables
Classifier CF;
Testset testSet;
Stop condition stopCond;
Vector of neighborhood probabilities w;
The moves to use moves;

function Roaming(CF, testSet, stopCond, w, moves)
    Randomly select a starting point
    while stopCondition is not fulfilled:
        select a move from moves with probability given in w
        perform move
    end while
end function
```

The roaming search may of course be performed with exactly the same moves as suggested for the VNS, but a set more suitable to the algorithm design is

\{add_1, add_2, add_3, rem_1, rem_2, rem_3, opt_1, opt_2\}

Each of the moves are given weights as to how probable the use of a move should be. The Roaming Search seeks much more diversity than the corresponding VNS, but a major shortcoming is that no local optima is searched out, instead it indiscriminately chooses non improving moves, even if improving moves were available. This leads to a simple improvement, any solution reached by the Roaming Search should at least be locally searched further in order to improve it.

### 6.3.5 RoamingSA

Once again addressing the shortcomings of a previous method, we remark that the Roaming Search does not have a properly defined convergence criteria.

The Roaming SA is an attempt to improve this. The SA letters are of course from Simulated Annealing, but the method also draws upon the knowledge of guided local search and Lagrangian relaxation in punishing unwanted features. (See problem formulation 5.1.)

We first decide whether we want to make an opt move, or an add/remove move. The selection of add/remove move will depend on a guidance function.

We will first need to create a guidance function. The guidance function \( g \) is defined for each \( k \) in the range \([0,1]\) as in figure 6.1 on page 45. The purpose is to use information about the search space in order to guide it to the correct number of features.
6.3. METAHEURISTICS

We then define the probability for making a remove as follows

**Definition 21** Define the projection

\[ P(k) = \begin{cases} 
0 & k \leq 0 \\
 k & 0 \leq k \leq N \\
 N & k \geq N 
\end{cases} \]

we may specify a move probability with respect to the guidance function \( g(k) \)

\[ p(\theta|t, k, r) = \frac{1 + \text{sgn}(\nabla g(k)) \left( \frac{g(P(k+1)) + g(P(k-1))}{2} \right)^{r/t}}{2} \]

where \( \text{sgn} \) is the signum function

\[ \text{sgn}(x) = \begin{cases} 
-1 & x < 0 \\
 0 & x = 0 \\
 1 & x > 0 
\end{cases} \]

What we can read from this is that whenever the guidance function is close to 1, we will have a very high probability for moving towards smaller function values in the guidance function. If the guidance function is close to zero, there will be no special on the direction of the search preference. Note also that if the tangent ever vanished, there would be absolutely no preference. As the time(\( t \)) increases, there will be higher probability for moving in the guidance function minimizing direction.

As mentioned earlier, we use the tabu-list technique in order to avoid getting stuck in local maxima.

6.3.6 Optimal Path Relinking

Optimal Path Relinking is based on the idea as GA, if two solutions are good, it is very likely that a solution in between them may be a good starting point for finding new solutions. The strategy serves as an intensifying strategy as will be discussed in 6.11 But also generates diversity as long as the candidate points are sufficiently spread out.

6.3.7 Inverse Path Relinking

Inverse Path Relinking in contrast to Optimal Path Relinking is a pure diversification strategy. It obtains diversity by linking a set of good solutions with their “inverse”. When a heuristic has converged, this is a way to send it “out fishing” again.
// Declaration of variables
CF:=< Classifier >
testSet: =< Testset >
stopCond: =< Stop condition >
w:=< Vector of move probabilities >
g: =< vector of guidance function values >
t: =< current time step >
coolDownRate: =< rate of cooling >
moveOptBias: =< bias for choosing a nonOpt move in favor of an opt move >

function RoamingSA(CF, testSet , stopCond , w, g, t, coolDownRate, moveOptBias)

  coolDownRate:=1/coolDownRate
  Randomly select a starting point

  while stopCondition is not fulfilled:
    decide opt or non opt move with moveOptBias
    if opt move
      choose degree l with weights from w
      move to opt of degree l
    else
      calculate moveBias from probability function
      decide add or remove with respect to moveBias
      choose degree l with weights from w
      move to add/remove with degree l
    end
  end while

end function
6.3. METAHEURISTICS

Listing 6.9: Optimal Path Relinking

//Declaration of variables
CF:=< Classifier >
testSet:=< Testset >
stopCond:=< Stop condition >
candidates:=< List of candidates for relinking >

function path_relink_optimal(CF, testSet, stopCond, candidates)

    keep list of elements to return
    while not stopCond:
        for candidate1 in all candidates
            move to candidate1
            for candidate2 in all remaining candidates
                relink(candidate2)
                add best relink to return list
            end for
        end for
    end while

    return return list

end function

Listing 6.10: Inverse Path Relinking

//Declaration of variables
CF:=< Classifier >
testSet:=< Testset >
stopCond:=< Stop condition >
candidates:=< List of candidates for relinking >

function path_relink_inverse(CF, testSet, stopCond, candidates)

    keep list of elements to return
    while not stopCond:
        for candidate in all candidates
            icandidate=inverse(candidate)
            relink(icandidate)
            add best relink to return list
        end for
    end while

    return return list

end function
6.4 Hyperheuristics

In this section we discuss hyperheuristics. In this context, the hyperheuristics are labeled hyperheuristics for the mere reason that they employ at least one metaheuristic as a subalgorithms as opposed to the metaheuristics who work with moves as their sub-algorithms. Two hyperheuristics will be presented.

6.4.1 Huff Puff

HuffPuff builds upon the metaheuristics “Inverse Path Relinking”(IPR) and “Optimal Path Relinking”(OPR).

The HuffPuff starts by doing an IPR from its origin. Each IPR will result in a best element. When enough elements are gathered, do optimal relinking in between them. Continue this until no significant improvement can be seen.

From the elite-lists, get the best elements, do OPR between them. While OPR improves the result, reinitialize OPR with new bestElements.

This strategy may be seen as an improvement on the GA algorithm. In the sense that when the GA has converged, it really has converged. The only way to get GA to diverge again is to do so much mutation that the GA almost degenerates to a Random Search.

6.4.2 Guided Roaming SA(GRSA)

If we look back at the Roaming SA algorithm in section 6.3.5 on page 43 we see that as an input parameter we need the guidance function. Guided Roaming SA is designed to provide exactly this function.

We start by systematically sampling the entire search space, and make a move to the best value based on the random sampling.

We are now in a position to make an educated guess as to the number of features we will include. We decide that it is best to include the number of features of the best solution so far. All other feature subset sizes are weighted with the square of their distance to the current best solution.
Listing 6.11: Huff Puff

// Declaration of variables
CF::<Classifier>
testSet::<Testset>
stopCond::<Stop condition>
iterMax::<number of non improving optimal relinks before moving on>

function HuffPuff(CF,testSet,stopCond,iterMax)
    start at current point // Typically non or all features selected

    while not stopCond:
        candList:=path_relink_inverse(CF,testSet,stopCond,candidates)
        c:=0
        while c<iterMax and not stopCond
            path_relink_optimal(CF,testSet,stopCond,candList)
            if no improvement
                c:=c+1
            end if
        end while
        c:=0
        while c<iterMax and not stopCond
            candList:=get best from elite-list
            path_relink_optimal(CF,testSet,stopCond,candList)
            if no improvement
                c:=c+1
            end if
        end while
    end while

    return best
end function
Listing 6.12: RoamingSAGuider

```plaintext
//Declaration of variables
CF:=<Classifier>
testSet:=<Testset>
stopCond:=<Stop condition>

function RoamingSAGuider(CF, testSet, stopCond)
    move to guidedRandomSelect(n=20)
    w:=<move probabilities>
    B:=best element from elite-list
    while not stopCond:
        for k=0:N
            a[i]:=1-<bestElement of size k from elite-list>/B.value)^2
        end for
        stopCond2:=stopCond and <some short time interval>
        RoamingSA(CF, testSet, stopCond2, w, a)
    while stopCondition is not fulfilled:
    end while
end function
```
6.5 Search topology and visualization

6.5.1 Visualization of the search topology

We could try to visualize the structure of the data we are trying to optimize over. I suggest the following algorithm to create a topology.

- Define \( \Phi_k \) as an ordered subset of \( \Phi \), such that if \( \phi_i \in \Phi_k \) then \( \phi_i \) has \( k \) features (The ordering is intentionally omitted).

- Let \( \phi_{n,i} \) be the nodes of a graph.

- Define a function \( f(\phi_{n,i}) \), where \( n \) is the number of features included, and \( i \) is the number of the element in the ordered set \( \Phi_k \), such that \( f(\phi_{n,i}) = J(S,S^*,T,T^*;\phi_{k,i}) \) where \( \phi_{k,i} \) is the \( i \)’th element of the ordered set \( \Phi_k \).

- Add a “1-adding”-edge from each \( \phi_{k,i} \) to \( \phi_{k+1,j} \) where \( \phi_{k+1,j} \) can be obtain from \( \phi_{k,i} \) by adding one element. Each “1-subtracting”-edge will equal a “1-adding” edge with opposite direction.

- Add “1-opt”-edges between each pair \( \phi_{k,i},\phi_{k,j} \), where \( \phi_{k,j} \) is separated from \( \phi_{k,i} \) by only one “1-adding” and “1-subtracting” edge.

- Let \( x, y, z \) be the respective initial coordinates of \( \mathbb{R}^3 \), then:
  \[
  x = \frac{2i - \binom{k}{d}}{2^d}, \quad y = \frac{k}{d+1}, \quad z = f(\phi_{k,i})
  \]

- (Optional) Determine the ordering of each set by applying the FDM (Force Direction Method)\([36]\) to the graph given above.

It is of course never easy to understand an algorithm directly, and it may be helpful to see a picture 6.2

6.5.2 Visualization of the search progression

Using the topology defined above it is in fact easy to visualize the search progression. Using the following process:

- Make a colormapping, which assigns a color to every value in the range \([0,1]\)

- For each iteration in the search
  - When evaluating a selection, update the color of the node using the colormapping.
Figure 6.2: Visualization of initial topology for projection of 7 dimensional feature search space.
When moving from one selection to another in the searchspace, draw a line between them.

For all lines in existence, dim them by a factor $a < 1$ by multiplying their current color with $a$.

If more than $c$ lines exist, delete the least recently made line.

It is difficult to show the process on paper, but it works reasonably well for dimension up to $d = 13$, and reasonable for $d = 14$ on a 19" screen. If more display space is available, it should be possible to visualize dimensions up to about $d = 20$. The visualization is quite crude, but gives very valuable visual feedback during debugging and development of heuristic algorithms, and may also serve as an instructional tool. It is not limited to the feature selection problem, but may be used for any $\{0, 1\}^d$ problem.
Figure 6.4: Beginning to show progress.
Figure 6.5: Some progress has been done, notice the dimming tail of the search.
Chapter 7

Numerical Issues

7.1 Regularization of Covariance matrices

Under certain conditions, e.g. lack of independent samples, the covariance matrix may end up as being very badly conditioned. We may attempt to solve this problem by performing a procedure called regularization. As it turns out regularization may not only help our attempt to solve a matrix equation, but may also improve classification accuracy on its own merit. The tradeoff however is that our classification procedure obtains a bias.

The particular solution chosen is the combination of (7.1) and (7.2) [37]

\[
\hat{\Sigma}_c(\alpha) = \alpha \hat{\Sigma}_c + (1 - \alpha)\hat{\Sigma}
\]

(7.1)

\[
\hat{\Sigma}(\lambda) = \lambda \hat{\Sigma} + (1 - \lambda)\hat{\sigma}^2 I
\]

(7.2)

which combined yields:

\[
\hat{\Sigma}_c(\alpha, \lambda) = \alpha \hat{\Sigma}_c + (1 - \alpha)\hat{\Sigma}(\lambda)
\]

(7.3)

where \( \hat{\Sigma}_c \) is the \( c \)-class specific estimated covariance matrix, \( \hat{\Sigma} \) is the estimated total covariance matrix, and \( \hat{\sigma}^2 \) is a vector of variances for all samples/features.

7.1.1 Optimal parameter selection

When using the regularization (7.3) we are left with the choice of parameters \( \alpha \) and \( \lambda \). This can be seen as an optimization problem of \( f : [0, 1] \times [0, 1] \to [0, 1] \) when the number of samples approaches infinity. Where \( f \) is the accuracy function. However for any finite sample size(\( N \)) evaluated on the accuracy function, the accuracy function has a finite range \( f_N : [0, 1] \times [0, 1] \to [0, 1] \) this causes certain challenges when doing optimization.
One approach to this optimization problem is to grid the entire domain, and evaluate each grid point.

Experimentation shows that this problem has a convex structure which suggests that a convex optimization solver may be used. In fact this has been done for the similar Ridge Parameter problem. [38]

Experimentation has shown that the gradient projection method with the constant step size rule\(^1\) yields good results on the interior of the domain within very few iterations.

Further improvement may be found using the Armijo rule/Wolfe conditions might be possible, if the problem could be proven to be a convex optimization problem, since these conditions provides guarantees for convergence[39].

### 7.2 Cross Validation

As the optimization problem at hand is a statistical optimization problem, the results obtained on a single realization for a parameter are not necessarily representative for the underlying “truth”. The method of cross validation is a way of compensating for this fact.

Cross validation is done in several ways. [41] lists the following:

- **Holdout cross-validation** is the simplest kind of cross-validation. Observations are chosen randomly from the initial sample set to form the validation data, and the remaining observations are retained as the training data. Normally, less than a third of the initial samples are used for validation data.

- **K-fold cross-validation** partitions the original sample set into \(K\) subsample sets. Of the \(K\) subsamples, a single subsample is retained as the validation data for testing the model, and the remaining \(K - 1\) subsamples are used as training data. The cross-validation process is then repeated \(K\) times (the folds), with each of the \(K\) subsamples used exactly once as the validation data. The \(K\) results from the folds then can be averaged (or otherwise combined) to produce a single estimation.

- **Leave-one-out cross-validation** uses a single observation from the original sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. This is the same as K-fold cross-validation where \(K\) is equal to the number of observations in the original sample set.

\(^1\)An overview of step-size rules may be found in [39] the wolfe conditions are explained in [40].
7.3 Cholesky factorization

In Bayesian classification we are trying to optimize the function

\[
\max_c p(x|\omega_c) = \frac{1}{(2\pi)^d/2|\Sigma_c|^{1/2}} \exp\left(-\frac{1}{2}(x - \hat{\mu}_c)^T \Sigma_c^{-1}(x - \hat{\mu}_c)\right)
\]  

(7.4)

with respect to \(C\).

If we take a look at the expression in the exponent

\[
(x - \hat{\mu}_c)^T \Sigma_c^{-1}(x - \hat{\mu}_c)
\]

(7.5)

it may be simplified. \[37\] suggests the following simplification.

\[
(x - \hat{\mu}_c)^T \Sigma_c^{-1}(x - \hat{\mu}_c) = [U_c^T(x - \hat{\mu}_c)]^T D_c^{-1} [U_c^T(x - \hat{\mu}_c)]
\]

(7.6)

This is justified by noting that \(\Sigma_c = U_c D_c U_c^T\) is the eigendecomposition of \(\Sigma_c\).

**Definition 22** If \(L_c\) fulfills the equation

\[
L_c^T L_c = \Sigma_c
\]

then \(L_c^T L_c\) is the Cholesky decomposition of \(\Sigma_c\).

This may be improved further by noting that \(\Sigma_c\) is a positive definite symmetric matrix, and therefore has a Cholesky decomposition.

\[
(x - \hat{\mu}_c)^T \Sigma_c^{-1}(x - \hat{\mu}_c) = (L_c^{-1}(x - \hat{\mu}_c))^T L_c^{-1}(x - \hat{\mu}_c) = \|L_c^{-1}(x - \hat{\mu}_c)\|_2^2
\]

(7.7)

Further we note that

\[
|\Sigma_c|^{-1/2} = |L_c^{-1}|
\]

(7.8)

We also note that we have\(^2\)

\[
|L_c^{-1}| = \prod_{i=1}^{N} (L_c^{-1})_{i,i}
\]

(7.9)

Since Cholesky factorization is known as one of the most numerically stable methods for inverting matrices, in addition to being one of the fastest, \[42\] this is a win-win situation.\(^3\)\(^4\)

\(^2\)The determinant of \(L_c^{-1}\) is equal to the product of the diagonal elements of \(L_c^{-1}\).

\(^3\)Cholesky factorization and Eigenvector decomposition are actually of the same order in big-O notation. However, Cholesky factorization may be implemented with fewer vector touches, using a Gaxpy Cholesky method. Fewer vector touches means fewer cache misses on a modern computer, and this is important for the practical speed. Eigenvector decomposition is treated in \[42\] under the name of \(LDM^T\) decomposition for symmetric matrices.

\(^4\)Eigenvector decomposition is also a very computationally stable method. See \[42\]
7.4. SIMPLIFICATION OF BAYESIAN OPTIMIZATION

7.3.1 Solving without inverting

If we once again look to (7.5) we can see that this may be seen as a series of linear systems.

We first define \( y \) according to:

\[
y = \Sigma^{-1}_c (x - \hat{\mu}_c)
\]

this can be rewritten as:

\[
\Sigma_c y = (x - \hat{\mu}_c) = b
\]

The above expression may be solved by first calculating \( b \) and then use a fast iterative solver for \( y \). This is especially true if \( \Sigma_c \) is sparse.

Now we may insert \( y \) and simply solve

\[
\alpha = (x - \hat{\mu}_c)^T y \quad (7.10)
\]

7.4 Simplification of Bayesian optimization

Again we look to (7.4) we see that for all \( s \) this optimization problem is equal, so we may remove the constant multiplier.

\[
\max_c p_1(x|\omega_c) = \frac{1}{|\Sigma_c|^{1/2}} \exp\left(-\frac{1}{2} (x - \hat{\mu}_c)^T \Sigma^{-1}_c (x - \hat{\mu}_c)\right) \quad (7.11)
\]

as shown in 7.3 we may substitute for \( \Sigma_c \)

\[
\max_c p_1(x|\omega_c) = \frac{1}{|L_c^{-1}|} \exp\left(-\frac{1}{2} ||L_c^{-1}(x - \hat{\mu}_c)||_2\right) \quad (7.12)
\]

since the logarithm function is a strictly increasing function, we may apply it:

\[
\max_c p_2(x|\omega_c) = \log \left[ \frac{1}{|L_c^{-1}|} \exp\left(-\frac{1}{2} ||L_c^{-1}(x - \hat{\mu}_c)||_2\right) \right]
\]

\[
= -\frac{1}{2} ||L_c^{-1}(x - \hat{\mu}_c)||_2 - \log(|L_c^{-1}|)
\]

\[
= -\frac{1}{2} ||L_c^{-1}(x - \hat{\mu}_c)||_2 - \log(\prod_{i=1}^{N}(L_c^{-1})_{i,i}) \quad (7.13)
\]

\[
= -\frac{1}{2} ||L_c^{-1}(x - \hat{\mu}_c)||_2 - \sum_{i=1}^{N} \log((L_c^{-1})_{i,i}) \quad (7.14)
\]

We may now substitute our original problem (7.4) for either (7.13) or (7.14). As we will see in Appendix A.2.1 the first is actually preferable when it comes to calculation.
Chapter 8

Experimental study

The Feature Selection problem is as we have seen by now a non-deterministic optimization problem in the sense that it involves stochastic variables. This means that care has to be taken in order to make a sound experimental setup.

A standard method for evaluating methods is to separate available ground-truthed data into roughly equal sized groups for training and testing. Performance is then reported with respect to the test data.

In order to get a hands on experience with the curse of dimensionality, the training data is again separated to equally sized subsets, with an equal number of samples for each class. These regions are then once again separated into a CVTS (See Section 2.4.2) and a validation set.

Classical methods have been tested and compared with the new methods, we discussed in Chapter 6. The classical methods have mostly been aiming to get maximal performance where the training set is sought to adapt itself as best as possible for a given number of features included ($k$). This however completely ignores the stochastic nature of our parameter estimation problem. And leaves the user to do an educated guess about the number of parameters to include. For this reason the tests have been performed in a way that breaks this design. We will see how they perform on cross-validation experiments.

The case studies are taken from [43]

8.1 Case study A - Forest type classification

The first dataset we study was captured by an airborne sensor(ROSIS) during the European Multisensors Airborne Campaign(EMAC-94) at May 10th 1994. The dataset has 81 spectral bands with sampling bandwidth from 12 nm in the lower part of the spectrum(430-550nm) to 4 nm in the upper part (554-830 nm), and a pixel size of 5.6$m \times 5.6$m. The ground truth is divided into three classes, corresponding to tree type, $C_1$ oak, $C_2$ beech and $C_3$ pine.
8.1. CASE STUDY A - FOREST TYPE CLASSIFICATION

<table>
<thead>
<tr>
<th>CLASS</th>
<th>TRAINING</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$ oak</td>
<td>5195</td>
<td>5518</td>
</tr>
<tr>
<td>$C_2$ beech</td>
<td>2083</td>
<td>2309</td>
</tr>
<tr>
<td>$C_3$ pine</td>
<td>807</td>
<td>950</td>
</tr>
</tbody>
</table>

Table 8.1: Ground truth for Fontainebleu dataset (in number of pixels). 700 samples were used for training for each experiment.

The dataset is summarized in Table 8.1. The total number of ground truthed samples in the dataset was 16862. Five experiments were performed where 700 samples from each class were used from the training data. In these 700 samples 10-fold cross-validation was performed to select features.

The forest classes have very similar spectra, especially the beech and oak classes, and this makes it a very difficult classification problem.\[43\]

Regularization proved to be very important for cross validation performance, the simple optimizer from Section 7.1.1 was used to tune parameters for the regularizer (7.3), but as this procedure is a very computationally expensive one, it was dropped when all experiments showed very little variance around a central point of $(\alpha = 0.85, \lambda = 0.75)$. This Regularization parameter has therefore been used for all subsequent experiments on this dataset. Results have also been generated for unregularized classifiers.

All methods except the deterministic add-l-remove-r have been run for a period of approximately 8 hours. The experiments have been running on hardware in the range between Pentium4 2.4 GHz and Pentium4 3.0 GHz there is therefore a small discrepancy in the amount of computation power each experiment has been allocated, but since the computations are repeated several times on different machines, there is no reason to suspect that this has had any impact on the results presented. For the non-deterministic algorithms, experiments have been repeated 30 times, and the results have been averaged. Some of the experiments have failed, due to networked file system failure, they have not been repeated and the averages are therefore reported over the experiments that have completed. The standard deviation is also computed on basis of the different experiments.

The add-l-remove-r method has been allowed to run until completion, no repeat experiments are necessary, due to the deterministic nature of the algorithm. For this reason no standard deviation has been computed.

Before running the final experiments, the different algorithms were run with different parameters, and the best were selected. The selection criteria was optimal performance in 8 hours. For the GA algorithm the parameters chosen was population size 40, 1 population group, and 100 iterations.

For HuffPuff itermax was chosen as 10, and for ILS acceptMax was chosen as 10 both are defaults. RandomSearch takes no parameters. Roam-
Add-l-rem-r was run with the moves were chosen as \{flip_{1}, flip_{2}, pFlip_{3}(n=100), pFlip_{4}(n=100)\} turn out the same results run with parameters ingSA(GRSA) was run with the same set of moves. The SA algorithm was found with Add-l-remove-r, GRSA, ILS and VNS algorithms. GRSA and SA heuristics, while best performance on the cross validation.

8.4 performance is found for ILS, GRSA and Add-l-remove-r. If we move on to the testset for GA, GRSA and Add-l-remove-r. For the cross validation that for the unregularized classifier, we find the best performance on the number of selected features. This is natural since the increase of the number of selected features for the regularized classifiers do not improve accuracy very much compared to the non regularized classifiers.

Looking at the performance of the individual heuristics in 8.1, we see that for the unregularized classifier, we find the best performance on the testset for GA, GRSA and Add-l-remove-r. For the cross validation the best performance is found for ILS, GRSA and Add-l-remove-r. If we move on to the testset for GA, GRSA and Add-l-remove-r. For the cross validation the best performance is found with the HuffPuff, GRSA and SA heuristics, while best performance on the cross validation is found with Add-l-remove-r, GRSA, ILS and VNS algorithms.

1Only 16 out of 30 runs completed, but it is still very strange that all 16 runs should turn out the same results.
8.1. CASE STUDY A - FOREST TYPE CLASSIFICATION

<table>
<thead>
<tr>
<th>Method</th>
<th>Average number of features selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>26.30% ± 0.60%</td>
</tr>
<tr>
<td>HuffPuff</td>
<td>31.57% ± 1.95%</td>
</tr>
<tr>
<td>ILS</td>
<td>22.93% ± 3.26%</td>
</tr>
<tr>
<td>RandomSearch</td>
<td>31.00% ± 0.66%</td>
</tr>
<tr>
<td>Roaming</td>
<td>27.03% ± 5.91%</td>
</tr>
<tr>
<td>GRSA</td>
<td>18.40% ± 3.60%</td>
</tr>
<tr>
<td>SA</td>
<td>22.57% ± 0.43%</td>
</tr>
<tr>
<td>VNS</td>
<td>44.93% ± 6.91%</td>
</tr>
<tr>
<td>Add-l-Remove-r</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 8.3: The average number of features selected and one standard deviation for the Fontainebleu dataset. Using no Regularization

<table>
<thead>
<tr>
<th>Method</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>Total</th>
<th>Total CV</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>81.39% ± 0.00%</td>
<td>79.90% ± 0.00%</td>
<td>93.16% ± 0.00%</td>
<td>82.27% ± 0.00%</td>
<td>86.91% ± 0.00%</td>
</tr>
<tr>
<td>HuffPuff</td>
<td>82.01% ± 0.48%</td>
<td>80.02% ± 0.88%</td>
<td>92.95% ± 0.42%</td>
<td>82.67% ± 0.58%</td>
<td>85.92% ± 0.04%</td>
</tr>
<tr>
<td>ILS</td>
<td>80.62% ± 0.21%</td>
<td>80.28% ± 0.66%</td>
<td>94.07% ± 0.07%</td>
<td>81.99% ± 0.12%</td>
<td>87.03% ± 0.16%</td>
</tr>
<tr>
<td>RandomSearch</td>
<td>82.40% ± 0.79%</td>
<td>79.17% ± 1.28%</td>
<td>92.84% ± 1.49%</td>
<td>81.68% ± 1.03%</td>
<td>85.81% ± 1.77%</td>
</tr>
<tr>
<td>Roaming</td>
<td>81.38% ± 0.80%</td>
<td>79.64% ± 2.08%</td>
<td>93.21% ± 1.11%</td>
<td>82.21% ± 0.93%</td>
<td>86.26% ± 0.56%</td>
</tr>
<tr>
<td>GRSA</td>
<td>81.44% ± 1.07%</td>
<td>81.08% ± 1.57%</td>
<td>93.24% ± 0.02%</td>
<td>82.63% ± 0.26%</td>
<td>87.09% ± 0.10%</td>
</tr>
<tr>
<td>SA</td>
<td>81.59% ± 1.25%</td>
<td>80.44% ± 0.81%</td>
<td>92.79% ± 0.16%</td>
<td>82.50% ± 0.56%</td>
<td>86.97% ± 0.23%</td>
</tr>
<tr>
<td>VNS</td>
<td>81.44% ± 1.82%</td>
<td>79.66% ± 1.31%</td>
<td>93.89% ± 0.53%</td>
<td>82.32% ± 0.74%</td>
<td>87.08% ± 0.27%</td>
</tr>
<tr>
<td>Add-l-Remove-r</td>
<td>81.53%</td>
<td>80.03%</td>
<td>94.00%</td>
<td>82.49%</td>
<td>87.17%</td>
</tr>
</tbody>
</table>

Table 8.4: The results for the Fontainebleu dataset on 30 repeated experiments. Average correct classification and one standard deviation of the experiments in percent. Using Regularization parameter $\alpha = 0.85, \lambda = 0.75$

<table>
<thead>
<tr>
<th>Method</th>
<th>Average number of features selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>45 ± 0.00</td>
</tr>
<tr>
<td>HuffPuff</td>
<td>44.43 ± 0.57</td>
</tr>
<tr>
<td>ILS</td>
<td>43.83 ± 1.83</td>
</tr>
<tr>
<td>RandomSearch</td>
<td>43.00 ± 2.34</td>
</tr>
<tr>
<td>Roaming</td>
<td>42.50 ± 14.89</td>
</tr>
<tr>
<td>GRSA</td>
<td>49.36 ± 2.85</td>
</tr>
<tr>
<td>SA</td>
<td>43.03 ± 3.23</td>
</tr>
<tr>
<td>VNS</td>
<td>44.66 ± 5.61</td>
</tr>
<tr>
<td>Add-l-Remove-r</td>
<td>42</td>
</tr>
</tbody>
</table>

Table 8.5: The average number of features selected and one standard deviation for the Fontainebleu dataset. Using Regularization parameter $\alpha = 0.85, \lambda = 0.75$
8.2 Case study B - Urban landcover classification

To further investigate the methods presented, we compare the results from the first dataset with data from an urban scene over Pavia, Italy, captured by an airborne sensor (DAIS) under the HySens project at June 8th 2002. This dataset consists of 80 bands, however the last 8 bands are thermal infrared bands and since the first dataset only consisted of reflective measurements, these 8 were excluded from this study. Furthermore, one band at 1.9580\,nm was extremely noisy. The number of bands used is 71.

The ground truth consisted of a total of 14585 samples describing common urban landcover classes chosen by an analyst. The classes were $C_1$ water, $C_2$ trees, $C_3$ asphalt, $C_4$ parking lot, $C_5$ bitumen, $C_6$ roofs, $C_7$ meadow, $C_8$ soil and $C_9$ shadow.

Regularization has a big influence also on this problem, however for this experiment there was no obvious choice for regularization parameters. As mentioned earlier optimization of the regularization parameters is extremely expensive. To optimize regularization during feature selection would mean prohibitively long computations, so no regularization is performed for this case study.

The parameters for the algorithms are chosen equivalently to the previous case study.

From table 8.2 We may see that SA and ILS performs somewhat better on the test set, but ILS, Roaming,GRSA and SA performs best for the cross validation.

Except for GA and Add-l-Remove-r most solutions have an approximately equal number of features included, as we may read from table tbl:nuPavia

We may note that in this case the performance of the best classifiers are better than what we may find in [43], but the classifiers presented here have approximately double the amount of training samples available compared to [43]

8.3 Summary of the experiments

It is tempting to rank the algorithms with respect to their performance on the test set, which would be natural for most classifiers, but we should remember that the task of any optimization algorithm is to optimize the function at hand. The effects we observe here indicates that the crossvalidation setup is not good enough, and that we slightly overadapt. Some of the results indicate that we should seek out good, but not too good results, as they are more likely not to be overadaptations. We may note that there are quite small deviances around many of the tests, which indicates that the effects are not totally random.

The effects can on the other hand not be dismissed as entirely random
8.3. SUMMARY OF THE EXPERIMENTS

<table>
<thead>
<tr>
<th>Method</th>
<th>GA</th>
<th>HullPuff</th>
<th>ILS</th>
<th>RandomSearch</th>
<th>Roaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>93.16% ± 0.03%</td>
<td>92.95% ± 1.44%</td>
<td>94.37% ± 1.32%</td>
<td>91.61% ± 3.84%</td>
<td>93.55% ± 0.77%</td>
</tr>
<tr>
<td>$C_2$</td>
<td>86.46% ± 0.16%</td>
<td>87.06% ± 0.98%</td>
<td>90.18% ± 0.43%</td>
<td>87.65% ± 1.15%</td>
<td>88.77% ± 0.06%</td>
</tr>
<tr>
<td>$C_3$</td>
<td>100.00% ± 0.00%</td>
<td>100.00% ± 0.00%</td>
<td>100.00% ± 0.00%</td>
<td>100.00% ± 0.00%</td>
<td>100.00% ± 0.00%</td>
</tr>
<tr>
<td>$C_4$</td>
<td>98.16% ± 0.31%</td>
<td>97.75% ± 0.87%</td>
<td>98.14% ± 0.17%</td>
<td>97.80% ± 0.33%</td>
<td>98.02% ± 0.40%</td>
</tr>
<tr>
<td>$C_5$</td>
<td>70.20% ± 1.61%</td>
<td>76.77% ± 5.56%</td>
<td>70.08% ± 3.41%</td>
<td>74.73% ± 3.26%</td>
<td>67.42% ± 4.55%</td>
</tr>
<tr>
<td>$C_6$</td>
<td>50.10% ± 1.47%</td>
<td>45.49% ± 1.05%</td>
<td>44.97% ± 0.94%</td>
<td>47.80% ± 0.63%</td>
<td>47.01% ± 0.16%</td>
</tr>
<tr>
<td>$C_7$</td>
<td>94.84% ± 1.29%</td>
<td>94.43% ± 0.27%</td>
<td>95.01% ± 2.14%</td>
<td>94.37% ± 2.10%</td>
<td>93.89% ± 1.02%</td>
</tr>
<tr>
<td>$C_8$</td>
<td>98.31% ± 0.00%</td>
<td>97.81% ± 0.25%</td>
<td>97.91% ± 0.04%</td>
<td>97.78% ± 0.26%</td>
<td>98.13% ± 0.26%</td>
</tr>
<tr>
<td>$C_9$</td>
<td>99.71% ± 0.05%</td>
<td>99.89% ± 0.07%</td>
<td>99.82% ± 0.11%</td>
<td>99.67% ± 0.06%</td>
<td>99.85% ± 0.13%</td>
</tr>
<tr>
<td>Total</td>
<td>96.16% ± 0.05%</td>
<td>96.06% ± 0.27%</td>
<td>96.57% ± 0.30%</td>
<td>95.98% ± 0.49%</td>
<td>96.32% ± 0.07%</td>
</tr>
<tr>
<td>Total CV</td>
<td>99.76% ± 0.02%</td>
<td>99.74% ± 0.01%</td>
<td>99.85% ± 0.04%</td>
<td>99.47% ± 0.13%</td>
<td>99.88% ± 0.01%</td>
</tr>
</tbody>
</table>

Table 8.6: Ground truth for the Pavia dataset (in number of pixels). 100 samples were used for training for each experiment.

<table>
<thead>
<tr>
<th>Method</th>
<th>RSA</th>
<th>SA</th>
<th>VNS</th>
<th>Add-l-Remove-r</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>93.05% ± 2.46%</td>
<td>93.46% ± 2.97%</td>
<td>91.00% ± 0.47%</td>
<td>91.64%</td>
</tr>
<tr>
<td>$C_2$</td>
<td>87.02% ± 0.40%</td>
<td>89.80% ± 1.06%</td>
<td>88.40% ± 0.30%</td>
<td>86.69%</td>
</tr>
<tr>
<td>$C_3$</td>
<td>100.00% ± 0.00%</td>
<td>100.00% ± 0.00%</td>
<td>99.99% ± 0.01%</td>
<td>100.00%</td>
</tr>
<tr>
<td>$C_4$</td>
<td>98.30% ± 0.14%</td>
<td>98.21% ± 0.13%</td>
<td>97.88% ± 0.11%</td>
<td>99.00%</td>
</tr>
<tr>
<td>$C_5$</td>
<td>70.35% ± 0.11%</td>
<td>67.29% ± 2.08%</td>
<td>71.54% ± 0.32%</td>
<td>68.94%</td>
</tr>
<tr>
<td>$C_6$</td>
<td>44.92% ± 3.41%</td>
<td>44.65% ± 2.52%</td>
<td>44.20% ± 1.44%</td>
<td>50.31%</td>
</tr>
<tr>
<td>$C_7$</td>
<td>95.11% ± 1.63%</td>
<td>93.58% ± 1.12%</td>
<td>94.06% ± 0.23%</td>
<td>94.09%</td>
</tr>
<tr>
<td>$C_8$</td>
<td>98.24% ± 0.15%</td>
<td>98.12% ± 0.27%</td>
<td>98.31% ± 0.04%</td>
<td>98.13%</td>
</tr>
<tr>
<td>$C_9$</td>
<td>99.83% ± 0.10%</td>
<td>99.93% ± 0.05%</td>
<td>99.71% ± 0.05%</td>
<td>99.63%</td>
</tr>
<tr>
<td>Total</td>
<td>96.15% ± 0.28%</td>
<td>96.44% ± 0.25%</td>
<td>96.04% ± 0.06%</td>
<td>95.99%</td>
</tr>
<tr>
<td>Total CV</td>
<td>99.91% ± 0.04%</td>
<td>99.84% ± 0.01%</td>
<td>99.27% ± 0.05%</td>
<td>99.40%</td>
</tr>
</tbody>
</table>

Table 8.7: The results for the Pavia dataset on 15 repeated experiments. Average correct classification and one standard deviation of the experiments in percent. Using no regularization.
8.3. Summary of the Experiments

<table>
<thead>
<tr>
<th>Method</th>
<th>Average number of features selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>24.00 ± 0.00</td>
</tr>
<tr>
<td>HuffPuff</td>
<td>35.00 ± 2.53</td>
</tr>
<tr>
<td>ILS</td>
<td>31.40 ± 1.40</td>
</tr>
<tr>
<td>RandomSearch</td>
<td>32.87 ± 4.56</td>
</tr>
<tr>
<td>Roaming</td>
<td>35.46 ± 4.27</td>
</tr>
<tr>
<td>GRSA</td>
<td>31.46 ± 5.09</td>
</tr>
<tr>
<td>SA</td>
<td>32.67 ± 2.09</td>
</tr>
<tr>
<td>VNS</td>
<td>36.60 ± 3.18</td>
</tr>
<tr>
<td>Add-l-Remove-r</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 8.8: The average number of features selected and one standard deviation for the Pavia dataset. Using no Regularization effects, given the small deviances around the testsets. A possible resolve to the problems we encounter is to increase the number of cross validations, or possibly finding a smarter way of setting up cross validations.

Looking at the experiments we see that the best performance may be repeatedly be found for GA, GRSA, ILS and Add-l-Remove-r, and possibly SA. We may look to figure 8.1 for a comparison of the convergence speed of the first three. As we see there is a notable difference in the convergence speed. While the GRSA converges quite fast, the ILS, climbs behind, and eventually overcomes the GRSA after a long running time. The Add-l-Remove-r algorithm is comparatively more expensive to run.

The VNS may have been a bit unfairly treated. During the initial experiments it showed some very good runs, but given a bad starting point, it uses most of its time searching for improvements in “bad” neighborhoods. It could however be a very promising method for longer running experiments.

To sum it all up, we see that for the Fontainebleu dataset, the feature reductions techniques does not work very well, compared to [43]. For the Pavia dataset, the results look very promising.
Figure 8.1: Selected runs of 3 algorithms, to demonstrate different convergence functions. The runs are for the Rosis set, without regularization.
Chapter 9

Conclusions/Future work

9.1 Conclusion

In this thesis I have presented a number of metaheuristics, designed to improve the classification accuracy of hyperspectral images. I have adapted 2 known algorithms to the feature selection problem. I have also presented 3 new metaheuristics. I’ve found that 2 of these new algorithms offers great promise compared to traditional methods. The ILS shows good results quite rapidly, and also improves over time. The Guided Roaming SA reaches good solutions in a very short time. It would be very interesting to explore them in an even broader perspective and with better cross validation.

I’ve also developed a method of visualizing the search space, which fits not only the feature selection problem, but also the more general class of optimization problems over the \( \{0, 1\}^d \) space. Although being crude, it offers visual help during debugging and may also be helpful as an instructional tool.

9.2 Future work

There are several issues that should be explored.

Foremost the development of better crossvalidation setup, and the testing of the methods on many more problems.

The software library now contains a lot of optimizations, but is far from perfect, it would be very interesting to implement some other techniques, like predictors etc. There are also some design choices that should be rethought.

The heuristics ILS, and GRSA shows very promising results, but the combination of ideas may lead to an even better heuristic.

It would be interesting to pursue the idea of updating the Cholesky factorization. As it turns out the matrix vector product uses 70% of the processing time, if an update formula for the Cholesky factorization could be found,
maybe an update formula could be shown for the repeated classifications.
Appendix A

Implementation issues

Investigation of libraries has suggested that many existing libraries do not perform very well with respect to speed when it comes to feature selection. The author want to argue that this is due to the fact that most libraries are not written with feature selection in mind, but rather with the intention of performing classification after feature selection is already performed.

I have implemented a new library, where I’ve used most of the “tricks in the book”. I make no claim that this is the fastest library existing, but comparison with for example matlab suggests that it performs quite well.

Many of the tricks employed are extremely basic, and many of the subjects touched in this appendix are in my opinion rather obvious.

A.1 Object orientation

The library is built in an object orientated way. The reason behind this choice of implementation, beside being the prescribed way of doing implementation today, is that every module may be easily interchanged for another. The cost of this is a relatively minor performance hit of about 10–20% computation time.

It consists of a set of main objects:

Classifier performs the actual classification. This particular implementation does Bayesian classification, but several other classification methods could be implemented.

Confusion contains the confusion matrix generated by a classification. It also provides several derived metrics from this matrix.

FeatureSet is a container class. It contains every single sample, in addition to all the true labels of a class, if provided. In addition it also contains the currently selected subset.
A.2. ALGORITHM EFFICIENCY

Point  Contains low dimensional points, convenience class.

Random  contains several functions for generating random draws, with different statistical properties.

Selection  Selection contains the actual selection. This is in some-ways a “Big Integer” class, with several possible iterations and logical operations.

There are also a set of “procedures” that does the actual computation on these objects:

aux_matrix  contains all matrix functionality that does not fit elsewhere.

common_functions/common  contains a set of convenience functions, in addition to some “switchable” debug functionality. Meaning that debug can efficiently be switched on and off at compile time.

heuristics  contains all the local operators and simple heuristics that are implemented for this problem\(^1\).

invert_matrix  has different operations for doing matrix inversion.

optimize  contains some functionality for doing optimization of sub problems.

A.2  Algorithm efficiency

Some work has been invested in making most parts of the software as fast as possible. However this has not been a major concern, and some parts have known speed issues, that could be easily resolved if one invested the time in doing this.

A.2.1  Pre calculation of relative constants

Several of the calculations in this problem can be done before hand.

Probably the most important Pre-calculation is the calculation of the “global” covariance matrix, or the covariance matrix with all features selected. All selected covariance matrices, can be derived from this by only selecting the rows and columns of this matrix which corresponds to the current selection.

If we take a look on (7.13) we see that this has to be done for each \(s\). The obvious thing to notice about this expression is that \(L_s^{-1}\) is constant for each \(x\), and typically the number of \(x\)'s(\(M\)) are large. This means that we should precompute \(L_s^{-1}\) before attempting to solve this. If \(M\) is a very

\(^1\)All in this respect refers to all that has been implemented in this context.
A.2. ALGORITHM EFFICIENCY

small number, it might be better to try an iterative method as suggested with (7.10)\(^2\).

If we once again take a look at (7.13) we may also see that the last term in the sum is also the same for each \(x\).

\[
\max_c p_2(x|\omega_c) = -\frac{1}{2}||L_c^{-1}(x - \hat{\mu}_c)||_2 - \log(\prod_{i=1}^{N}(L_{c}^{-1})_{i,i})
\]

This means that we should rather calculate:

\[
\min_c p_3(x|\omega_c) = ||L_c^{-1}(x - \hat{\mu}_c)||_2 + b_c \tag{A.1}
\]

\[
b_c = 2 \log(\prod_{i=1}^{N}(L_{c}^{-1})_{i,i}) \tag{A.2}
\]

and Pre-calculate \(b_c\) as we did with \(L_c^{-1}\).

Should we choose to use the alternative iterative scheme suggested in 7.3.1 we would replace the first term in (A.1) with \(\alpha\) from (7.10) such that

\[
\min_c p_3(x|\omega_c) = \alpha + b_c \tag{A.3}
\]

A.2.2 Random selection

The random selection is encapsulated into a separate class, it provides a quite efficient implementations of the random selection process. Initialization of the draw with replacement randomization is \(O(M_c)\) where \(M_c\) is the pool size. Typically the number of features in a single class, and retrieval is \(O(1)\).

This is important, because it supports the possibility of selecting classes with equal sample sizes. Which again is important for generating the covariance matrices. Early experiments have shown that without uniform sample sizes, small sample sizes in one class occurred so frequently that is posed a great problem with respect to invertibility of the corresponding covariance matrix.

For the draw without replacement, all operations are \(O(1)\)

A.2.3 Updating the Cholesky factorization

As mentioned in A.2.1 we may derive a selected submatrix from the “global” matrix. If we choose two selections \(a\) and \(b\) where

\(^2\)This has not been implemented, since the number \(M\) must either be very small, or \(\Sigma_s\) has to have an exploitable structure. Either way these are special cases that should only be solved if such a structure is ubiquitous.
A.3. UNIT TESTING

\[ a = \{a_1, a_2\}, b = \{b_1, b_2\} \]

and \(a_1, a_2, b_1, b_2\) are smaller selections.

We can now find the two matrices \(\Sigma_a\) and \(\Sigma_b\), with their respective Cholesky factorizations \(L_a\) and \(L_b\).

Suppose

\[ a_1 = b_1, \quad n = \#(a_1) = \#(a_2) \]

is the number of selected features of respectively \(a_1\) and \(a_2\).

Then the \(n \times n\) principal sub-matrices of \(L_a\) and \(L_b\) are equal.

While this has not been exploited in this implementation, a speed improvement might be very possible by exploiting this structure.

A.2.4 Selection

This is probably one of the least optimized classes. It consists of an STL vector of \texttt{bools}. This is a very inefficient way of implementing such a class, and is only done since it requires extremely simple logic.

There are two problems with this implementation. Firstly it has a very large memory footprint, resulting in inefficient memory copying, when copying objects. Secondly, it is inefficient because many of the logic operations that have to be done for each \texttt{bool} could easily be done on a machine word length group as a single operation.

The right way of doing this would have been to implement it as a series of machine word length unsigned integers and do the necessary bit operations on them.

A.3 Unit testing

Following the object orientation principle from before, all classes have been fitted with two methods.

\texttt{isValid} provides basic and easily performed tests that secures a basic level of consistency on a class.

\texttt{integrityCheck} provides extensive checking and may even redo larger calculations in order to secure internal consistency within a class. Passing integrityCheck implies passing isValid.

These methods are provided within the framework, and makes extending the framework less error prone. If debugging is turned on, these methods makes it very hard to make programs that uses internal data-structures in such a way that they break.

In summary we can say: \textit{This does not mean that errors can not occur, but the error has to be consistent.}
A.4  Compile time optimization

As this library has gotten more functionality, the compile time has soared. While this is not a big issue during use, it is during development. Object orientation is also known to be less efficient than optimal “spaghetti code”, and some steps have been done to resolve these issues.

Separation of code

Perhaps the easiest optimization is the separation of code into units that naturally cooccur. Effort has been put into making mutual inclusion sets as small as possible.

Compiler settings

Object orientation principles such that protection of variables dictates that access to private variables should be encapsulated in function calls. Attempting to do this without unrolling these function calls, yields a substantial overhead.

In addition mathematical functions have a lot of added functionality for checking boundary cases.

Loops may also be expressed more easily if unrolled.

Using aggressive compiler optimization is essential for getting the most out of a library. By default this is done.

A.5  Final notes on implementation

In summary the implementation is quite efficient compared to other implementations known to us. Additional work may be done especially with respect to the selection class, and the mathematical aspects of updating the Cholesky factorization. Other than this there are no known deficiencies in the implementation.
## Appendix B

### List of symbols and their definition

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Defined where</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ξ</td>
<td>Definition 1 on page 13</td>
<td>Feature Set with ground truths</td>
</tr>
<tr>
<td>d</td>
<td>Definition 1 on page 13</td>
<td>Dimensionality/Number of features in feature vector</td>
</tr>
<tr>
<td>φ</td>
<td>Definition 2 on page 13</td>
<td>Selection vector</td>
</tr>
<tr>
<td>k</td>
<td>Definition 3 on page 13</td>
<td>The $L_1$ norm of $\phi$</td>
</tr>
<tr>
<td>c</td>
<td>Definition 4 on page 13</td>
<td>The class of a feature</td>
</tr>
<tr>
<td>C</td>
<td>Definition 4 on page 13</td>
<td>The set of allowed classes</td>
</tr>
<tr>
<td>C#</td>
<td>Definition 4 on page 13</td>
<td>The number of allowed classes</td>
</tr>
<tr>
<td>S</td>
<td>Definition 5 on page 14</td>
<td>Trainingset</td>
</tr>
<tr>
<td>T</td>
<td>Definition 6 on page 14</td>
<td>Testset</td>
</tr>
<tr>
<td>N</td>
<td>Definition 7 on page 14</td>
<td>The number of samples in a set</td>
</tr>
<tr>
<td>S*</td>
<td>Definition 7 on page 14</td>
<td>Ground truth of the training-set</td>
</tr>
<tr>
<td>T*</td>
<td>Definition 8 on page 14</td>
<td>Ground truth of the test-set</td>
</tr>
<tr>
<td>J J(φ)</td>
<td>Definition 9 on page 14</td>
<td>The objective function</td>
</tr>
<tr>
<td>i</td>
<td>Definition 10 on page 14</td>
<td>Iteration variable, assumes values in ${1, \ldots, d}$</td>
</tr>
<tr>
<td>Φ</td>
<td>Definition 11 on page 14</td>
<td>Search space</td>
</tr>
<tr>
<td>x</td>
<td>Definition 12 on page 16</td>
<td>A single sample</td>
</tr>
<tr>
<td>µ</td>
<td>Definition 13 on page 16</td>
<td>Mean of a set of samples</td>
</tr>
<tr>
<td>Σ</td>
<td>Definition 14 on page 16</td>
<td>Covariance matrix of a set of samples</td>
</tr>
<tr>
<td>α</td>
<td>Equation 7.1 on page 55</td>
<td>Regularization parameter 1</td>
</tr>
<tr>
<td>λ</td>
<td>Equation 7.2 on page 55</td>
<td>Regularization parameter 2</td>
</tr>
<tr>
<td>L</td>
<td>Definition 22 on page 57</td>
<td>Cholesky factorization of $\Sigma$</td>
</tr>
</tbody>
</table>
**Glossary**

**Bias**
A Biased estimator is an estimator that for some reason over or underestimates the quantity being estimated., \(^{55}\)

**Boolean**
is a primitive variable type. This variable may take on only the values *true* or *false*, \(^{72}\)

**CMY**
is a mirror space to the RGB space. It is a subtractive model, and is often preferred for instance in color printing where one subtracts colors from the white paper sheet, instead of adding to them. If RGB is described in the coordinates \(c \in [0, \color{\text{max}}]\), then CMY can be described with \(c' = \color{\text{max}} - c\). (For more information see \([?]\))., \(^{2}\)

**confusion matrix**
is a matrix which describes the relation between the true labels of a class, and which label it has been given. “Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e. commonly mislabeling one as an other)” (For more information see \([?]\))., \(^{5}\)

**discrimination-functions**
are functions which separates a (hyper-)space into separate regions. It is used for classification in such a way that all observations which are not separated by such a function will be classified to the same label., \(^{4}\)
feature is a single quantitative description obtained from an object, either directly or after some pre-processing., 2

GLCM Gray Level Cooccurrence Matrix is a matrix of the pairwise transitions between different intensities in a picture. A transition may eg. be the transition between two neighboring pixels., 10

Local mode A local mode refers to an area under a function $f$ surrounded by local minima. In other words for a point $x$, if there is a path $p(\lambda)$ connecting $x$ and an arbitrary point $y$ for which the path fulfills the condition $f(p(\lambda)) \geq f(x)$ then $y$ belongs to the local mode., 30

Machine Word is the “base” unit of a machine. It is generally the largest unit on which effective calculations may be performed on a specific machine architecture., 72

MIP A Mixed Integer Problem is an optimization problem, that takes on both discrete and continuous constraints. 27

OCR Optical character recognition is the process of translating digitized images of text, into machine-editable text. With or without preservation of typesetting. (For more information see [?])., 15

point-clouds If you visualize a set of objects, and their corresponding in some coordinate system. You will hopefully see some kind of clustering. These clusters corresponds to the classes you are observing., 4
RGB is a color-model. Human color vision is normally comprised of 3 sensors (cones) which are most sensitive at “red”, “green” and “blue”. Thus the color-model can be fully described in 3D-coordinates, each ranging within \([0, \text{color\_max}]\) (For more information see \([\text{?}]) \), 8, 9

Ridge Parameter Problem is the problem of finding an optimal parameter for Ridge Regression, 56

STL is an abbreviation for Standard Template Library. This is a c++ library, providing template classes, 72

Unsigned integer is the computer primitive correspondent to Z. It may only contain positive numbers, although capped to be less than a certain number. This is a very efficient way of representing series of bools, 72
Bibliography


## Index

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandwidth limited</td>
<td>12</td>
</tr>
<tr>
<td>Branch and Bound</td>
<td>21</td>
</tr>
<tr>
<td>Cholesky</td>
<td></td>
</tr>
<tr>
<td>updating</td>
<td>71</td>
</tr>
<tr>
<td>Cholesky factorization</td>
<td>57</td>
</tr>
<tr>
<td>Crossvalidation</td>
<td>17</td>
</tr>
<tr>
<td>Curse of dimensionality</td>
<td>16</td>
</tr>
<tr>
<td>CVTS</td>
<td>17</td>
</tr>
<tr>
<td>deterministic</td>
<td>16</td>
</tr>
<tr>
<td>DOP</td>
<td>19</td>
</tr>
<tr>
<td>enumeration</td>
<td></td>
</tr>
<tr>
<td>full</td>
<td>20</td>
</tr>
<tr>
<td>implicit</td>
<td>21</td>
</tr>
<tr>
<td>error</td>
<td></td>
</tr>
<tr>
<td>parameter estimation</td>
<td>16</td>
</tr>
<tr>
<td>feature selection</td>
<td></td>
</tr>
<tr>
<td>applications</td>
<td>9, 10</td>
</tr>
<tr>
<td>complexity</td>
<td>13</td>
</tr>
<tr>
<td>example</td>
<td>1</td>
</tr>
<tr>
<td>Mathematical description</td>
<td>14</td>
</tr>
<tr>
<td>feature selection</td>
<td></td>
</tr>
<tr>
<td>problem description</td>
<td>12</td>
</tr>
<tr>
<td>fuzzy logic</td>
<td>8</td>
</tr>
<tr>
<td>gradient projection method</td>
<td>56</td>
</tr>
<tr>
<td>graph matching</td>
<td>8</td>
</tr>
<tr>
<td>LP</td>
<td></td>
</tr>
<tr>
<td>problem</td>
<td>26</td>
</tr>
<tr>
<td>relaxation</td>
<td>27</td>
</tr>
<tr>
<td>minimum error classification</td>
<td>15</td>
</tr>
<tr>
<td>Neural networks</td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td>6</td>
</tr>
<tr>
<td>non-deterministic</td>
<td>16</td>
</tr>
<tr>
<td>parameter estimation error</td>
<td>16</td>
</tr>
<tr>
<td>Path relinking</td>
<td>25</td>
</tr>
<tr>
<td>performance</td>
<td>5</td>
</tr>
<tr>
<td>relaxation</td>
<td>27</td>
</tr>
<tr>
<td>Lagrangian</td>
<td>27</td>
</tr>
<tr>
<td>LP</td>
<td>27</td>
</tr>
<tr>
<td>separability</td>
<td>2, 15</td>
</tr>
<tr>
<td>statistical pattern recognition</td>
<td>6</td>
</tr>
<tr>
<td>syntactic pattern recognition</td>
<td>8</td>
</tr>
<tr>
<td>Tabu search</td>
<td>25</td>
</tr>
<tr>
<td>testset</td>
<td>17</td>
</tr>
<tr>
<td>training limited</td>
<td>12</td>
</tr>
<tr>
<td>trainingset</td>
<td>17</td>
</tr>
<tr>
<td>crossvalidation</td>
<td></td>
</tr>
<tr>
<td>validationset</td>
<td>17</td>
</tr>
</tbody>
</table>