# Noisy Active Learning from a Bayesian Perspective

**Master Thesis**

**Author(s):**
Rupprechter, Benjamin

**Publication date:**
2012

**Permanent link:**
https://doi.org/10.3929/ethz-a-007305105

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Noisy Active Learning from a Bayesian Perspective

Master Thesis

Benjamin Rupprechter

April 17, 2012

Supervisor: Prof. Dr. Andreas Krause

Department of Computer Science
Learning & Adaptive Systems Group
ETH Zurich

**Abstract**

We approach the problem of active learning from a Bayesian perspective, working with a probability distribution over the solution space. In addition to the classical active selection of data points, we formulate the construction of minimum decision trees from noisy datasets as an active learning task.

Building on the OASIS algorithm, we compare active learning score functions based on the $EC^2$ criterion with uncertainty sampling, two GBS approaches, and random selection by performing experiments on several standard datasets. While constituting a unique approach in its original online setting, according to our findings, the OASIS algorithm does not generally offer performance benefits in a classical offline setting.

Furthermore, for active learning from noisy data samples, we introduce a new intrinsic $EC^2$-based stopping criterion and show that in many cases, it outperforms a standard information gain based method paired with the $\chi^2$-test on the decision tree learning problem. In particular, our criterion enables the effective construction of small decision trees, and may provide the first effective method to learn a close-to-minimal tree classifier with bounded expected error rates from noisy data samples.

**Acknowledgments**

I want to thank Andreas Krause for his caring supervision, in particular for all the time, effort and energy spent on providing guidance and explanations. Furthermore, I am indebted to Victor Candia for his support and helpful advice, and to Wito Traub for his comments.

My parents, Josef and Madeleine Rupprechter, deserve special thanks along with the rest of my family for their ongoing support, without which the writing of this thesis would have been impossible. Last but not least, a big thank you to the two loves of my life, Maya Minwary and Jesus Christ.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

Chapter 1

# Introduction

In the world of today, digital information dominates many aspects of business and everyday life. As storage and sensors become cheaper by the year, collecting data has never been easier, and with the increase of affordable processing power, the past decade has seen a rise of machine learning and pattern recognition methods.

As an illustrative example, consider digital images as data which are tagged with labels from two distinct classes. Let us assume that one class of images depicts trees while the other one shows houses. The goal for a machine learning method is to learn from a given set of labeled training images in such a way that the essential pattern, which separates images showing houses from images depicting trees, is extracted. This information can then be used to build an algorithm that classifies any image – not just the ones from the training set – as depicting a tree or a house.

The setting described so far is also known as *supervised learning*, because it corresponds to the algorithm having access to some higher instance – a teacher, so to speak – which knows the correct labels for all training instances and provides them without hesitation. In *unsupervised learning*, on the other hand, no labels are provided. Nonetheless, we can still try to find structure in the data, for example by grouping images according to their similarity. We will also look at a *semi-supervised* setting, where some labels are available and some are not, with the algorithm learning something in both cases. For the interested reader, subtler differences between these fields are discussed by Bishop ([2]).

After extracting pattern information from data, a learning algorithm is able to make predictions for new data samples. However, most of the time the algorithm will not be perfectly sure about the decision it makes, leading to the use of probability theory as a tool to formulate this incertitude. As explained by Bland and Altman ([3]), two schools of thought have emerged over the

years with different ways of thinking about probabilities. The *frequentist* perspective sees a probability as the expected frequency of an event to occur after making observations for a long time. For *Bayesians*, on the other hand, a probability specifies how plausible an event is given the currently available knowledge, and is therefore in some way related to a degree of belief.

When dealing with large datasets, labeling all the data examples that are available for training often is an enormous or even impossible task. To stay in our previous example, digital images are now widely available on the Internet, and we could easily give our algorithm access to millions of training examples without being able to provide the human attention and manpower required to label them all.

Fortunately, in order to extract the relevant structural information, often not that many data samples are necessary;[1] relatively few indicative data points often suffice to train the machine learning methods that are available today. While we may thus only need to manually label a manageable number of training examples, we are additionally faced with the task of selecting the best data points, i.e. the ones that are most indicative and provide important structural information.

The subfield of machine learning that deals with these concerns is called *active learning*,[2] and assumes that an algorithm can actively query the labels of unlabeled training examples that it is given. Of course, the more labels an algorithm queries, the better we expect its performance to be in terms of the classification mistakes made; therefore, an algorithm should ask for as few labels as possible. In practice, active learning, as we discuss it, is driven by *score functions*, which are greedy strategies choosing the label to query next based on the observations made so far.[3]

Instead of actively querying the labels of data samples, we could also imagine that in order to classify a new example, we want to only look at as few of its features as possible. Given a dataset together with its labels, this corresponds to learning a *decision tree* for classification.[4] Decision trees, in essence, specify which features to look at to classify a data sample, and if we ask for a decision tree that is as short as possible, we can interpret this setting as an active learning task.

Furthermore, an important aspect to consider is *noise* and how it influences active learning algorithms. We think of noise as a random process that might

[1] See Tong ([42]) for some examples and a discussion.

[2] See Settles ([37]) for a comprehensive introduction.

[3] We base the term *score function* on Goldberg et al. ([14]), who talk about a *score criterion*. Other authors use different terminology for the same concept, Settles ([37]), for example, denotes it *query strategy*.

[4] Mitchell ([29]) as well as Bishop ([2]) explain decision trees, which are useful for many tasks in the field of computer science.

change the label that an algorithm queries, i.e. instead of the true one it would receive a different, wrong class label. Our notion of noise is independent of time, meaning that even if an algorithm queries the label of a data point multiple times, it will always get the same result. In the presence of noise, knowing when to stop an active learning algorithm becomes a particular challenge. Learning from too many noisy labels easily leads to *overfitting*, a situation where the learner is well adjusted only to the particular noisy dataset it sees, but fails in its ability to generalize and apply the learned structure to new data instances.[5]

In this thesis, we compare and evaluate several active learning score functions in different settings, always taking on a Bayesian perspective.[6] Additionally, we propose a new stopping criterion that helps an active learning algorithm deal with noise, and show its merit for the decision tree construction problem.

[5]Hawkins ([18]) provides a good introduction to the problem of overfitting and reasons for its occurrence.

[6]Our main reason for operating within a Bayesian framework is that the methods we used as a starting point for our research adopt this perspective.

Chapter 2

# Literature Review

As a foundation, we assume the reader is aware of basic machine learning terminology and techniques as they are put forth in the standard works of Bishop ([2]), Mitchell ([29]) or Devroye et al. ([9]). Furthermore, many machine learning approaches, especially Bayesian models[1] as considered in this thesis, make heavy use of probability theory and statistics – fields into which an overview is provided by e.g. Wassermann ([48]). For the subdomain of active learning, Settles ([37]) provides the most recent and complete survey of the current literature and variety of techniques available.

Evaluating the results of a recent challenge for active learning methods, Guyon et al. ([16]), among other things, conclude that "semi-supervised learning is needed to achieve good performance in the first part of the learning curve". In this line of thought, a recent work by Goldberg et al. ([14]) introduces a particle filter based algorithm providing an integrated framework for active and semi-supervised learning in an online setting. We investigate this algorithm in some detail in its original form, and then adapt a noise model from Golovin et al. ([15]) to remove the online setting, facilitating a comparison with other active learning methods.

One important aspect we look at in our comparisons is how strategies for actively querying labels differ in terms of their classification mistakes. A strategy is usually represented by a score function that, given the current observations, maps each data point to a numerical value that describes its informativeness. Many different score functions have been proposed in the literature, but while there exists a wide variety of alternatives, most strategies are based on one of a few basic ideas.

Although usually not counted as active learning, the simplest strategy is to choose a sequence of data points at random. This can be done for all

[1] Consult e.g. Chaloner and Verdinelli ([6]) for a discussion of the Bayesian approach in experimental design.

kinds of learning algorithms, as no assumptions on the model or setting are made; as long as an algorithm can be phrased to learn iteratively from single examples, random selection is applicable. For example, Bottou ([5]) provides an overview of learning algorithms based on *stochastic gradient descent*, which iteratively optimize an objective function and randomize the input sequence of data points. While Bottou notes that stochastic gradient descent is a poor optimization algorithm, he also argues that it has proven itself as a very effective learning method.

Moving on to more informative approaches, a very common idea behind the creation of score functions is to rank data points according to some level of uncertainty that they possess. Intuitively, it makes sense to query data points that the current model is very uncertain about, because we expect to learn more from the label of such a data point rather than one whose outcome is almost certain. While Lewis and Gale ([23]) first used the term *uncertainty sampling* to describe this general method, we can see the same idea in the definition of the *entropy* by Shannon ([40]). In particular, Settles ([37]) notes that for binary classification, classical uncertainty sampling and usage of the entropy coincide.

While uncertainty-based methods are straight-forward to implement for a probabilistic model, variants have also been used in other settings. For example, a recent work by Vijayanarasimhan and Grauman ([44]) applies uncertainty sampling to *support vector machines*[2] (SVM), enabling them to perform large-scale image detection tasks. Instead of describing incertitude with probabilities, in the case of SVMs, the distance of a data point to the separating hyperplane is used as a measure of uncertainty. Similarly, as Lindenbaum et al. ([24]) demonstrate, uncertainty-based selection can be used for active nearest-neighbor classifiers.

Apart from uncertainty sampling, many active learning approaches are based on the notion of a *version space* that should be reduced as quickly as possible. Mitchell ([28]) defines a version space as the set of all hypotheses, among which we want to distinguish in a learning task, that are consistent with the observed training instances. When learning iteratively, this search space changes with every observation, and an active selection strategy can be defined based on how much candidate data points reduce the version space. Tong and Koller ([43]) apply this notion to SVMs, and discuss the *MaxMin* as well as *Ratio Margin* criteria to calculate the version space reduction. Many other ways of using the same basic idea have been proposed in the literature, e.g. He et al. ([19]) use a *mean version space*, and Dasgupta et al. ([8]) put forth a method to calculate the version space approximately in what they call *agnostic active learning*.

[2]SVMs are one of the standard classification tools available today. For an introduction, see Bishop ([2]).

Furthermore, Nowak ([30, 31]) describes *generalized binary search* (GBS), which extends the proven ideas of binary search to the search for the best hypothesis in an arbitrary space. In this case, strong theoretical guarantees that exists for classical binary search apply to GBS-based score functions. However, as Golovin et al. ([15]) note, these guarantees do not hold anymore when we consider noisy datasets as usually encountered in real-world applications.

Golovin et al. ([15]), therefore, propose the use of the EC² criterion, for which they prove strong theoretical results that hold even in the presence of correlated noise. In particular, the authors show that the EC² algorithm is guaranteed to choose the correct[3] hypothesis with close-to-optimal cost, which does not hold for common version space reduction approaches.[4]

While additional active learning strategies exist,[5] in this thesis' investigation we limit ourselves to some of the popular approaches. Our score functions include GBS and uncertainty sampling strategies, as well as an entropy-based information gain criterion. In addition, we work with the EC² criterion and evaluate how it performs in several experimental settings compared to the other score functions. Existing comparisons of active learning methods in the literature – e.g. the ones by Schein and Ungar ([34]) or Settles and Craven ([38]) – provide mixed results, i.e. which score function performs best depends on the concrete experimental setting. However, no experimental comparison that takes the EC² criterion into account has been conducted until now.

Especially in the case of noise, Settles ([37]) notes that a current active learning challenge is to find good stopping criteria that tell a learner when to stop to avoid overfitting the training data. Contributing to the ongoing research, we develop an intrinsic stopping criterion for the EC² score function and test it on the problem of constructing minimum decision trees, which corresponds to an active selection of features. Our approach can potentially benefit from the theoretical guarantees of the EC² score function, and it displays a very stable behavior even in the presence of a lot of noise.

In contrast with existing stopping criteria, we differ from work by Olsson and Tomanek ([32]) by not needing multiple learner instances, and from Vlachos ([45]) by not using uncertainty sampling, which in our experiments often does not yield a good performance. Bloodgood and Vijay-Shanker ([4]) base their criterion on the general notion of stability, which makes a consideration of the information that is present in the score function itself

[3] In this case, we define the correct hypothesis as the one which would have been chosen if all class labels had been observed.

[4] For example, Shi ([41]) notes that an entropy-based strategy struggles with the parity test problem.

[5] For example, there are *committee*-based query strategies, first proposed by Seung et al. ([39]), and *variance* reducing methods as Cohn ([7]) describes.

impossible. We further show in our simulations that our criterion in many instances outperforms the standard $\chi^2$-test[6] paired with the information gain criterion.

[6]See Walpole et al. ([46]) for a motivation and discussion of this statistical test.

Chapter 3

# Formal Foundation

In this chapter, we lay a formal foundation, which we use and need throughout the rest of this thesis. After spending a few words on the mathematical notation used, we introduce the general formal setting within which we work, and provide a framework for modeling noise, therefore unifying Chapters 4 and 5 by a common perspective.

## 3.1 Mathematical Notation

In general, we try to follow common mathematical notation as used in computer science. We use upper case calligraphic letters like $\mathcal{A}$ and $\mathcal{X}$ to denote sets, with the exception of standard sets of numbers such as $\mathbb{N}$ and $\mathbb{R}$. Variables are represented by lower case characters like $a$, $b$, $x$, and $y$, and upper case letters like $X$ and $Y$ are used for random variables.

We further write column vectors in bold lower case letters – for example $\mathbf{x}$ and $\mathbf{y}$ – and use the transpose sign $^\intercal$ to denote row vectors, e.g. $\mathbf{x}^\intercal$ and $\mathbf{y}^\intercal$. If we want to address the $i$-th element of some vector $\mathbf{x}$, we write $\mathbf{x}_{[i]}$. Other subscripts are usually used to index a set of variables, for example, we might use $\mathcal{X} := \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ to denote a set containing three column vectors as elements.

While we try to remain as consistent as possible, the meaning of some symbols changes according to context,[1] and thus we kindly ask the reader to pay attention to the setting within which mathematical formulas appear. Nevertheless, as a reference overview, Table 3.1 lists symbols that are used consistently to represent the same mathematical object. However, we must note that the specific interpretation of these objects might also change, i.e. the semantics depend on the context to some extent.

[1] In particular, this can hardly be avoided with variables used for indexing.

| Symbol | Description |
|---|---|
| $\mathbb{N}$ | Set of natural numbers (without zero) |
| $\mathbb{R}$ | Set of real numbers |
| $\mathcal{H}$ | Set of hypotheses |
| $\mathcal{T}$ | Set of tests |
| $\mathcal{Z}$ | Set of test outcomes |
| $\mathcal{X}$ | Data set |
| $\mathbf{x}$ | Data sample as column vector |
| $y$ | Class label of a data sample |

**Table 3.1:** Some mathematical symbols and their description as used in this thesis. Note that some general descriptions might be given different semantic meaning depending on the context.

## 3.2 General Setting

While investigating different problems, we assume an underlying general formal setting adapted from Golovin et al. ([15]), upon which the specific instances build.

Suppose that we are given a set $\mathcal{H} = \{h_1, \ldots, h_m\}$ of hypotheses from which we want to select a single hypothesis after performing a series of tests. These hypotheses can represent different objects, whose concrete interpretation will become clear when we present the specific settings later on. For now, we can think of them spaning a solution space from which we want to select the true solution (i.e. the true hypothesis), which we depict by the random variable $H$.

In order to be able to choose the correct hypothesis from the set $\mathcal{H}$, we can pick tests from a set $\mathcal{T} = \{t_1, \cdots, t_n\}$, where each test $t \in \mathcal{T}$ has a certain cost $c(t)$ associated with it and results in an outcome $z_t \in \mathcal{Z}$. With $\mathcal{Z}$, we denote the finite set of possible outcomes and suppose $|\mathcal{Z}| = \ell \in \mathbb{N}$. Again, this set of tests is an abstract entity that will take on different interpretations. depending on the context. We use random variables $Z_1, \ldots, Z_n$ to describe the outcomes of tests $\{t_1, \cdots, t_n\}$. After performing a test $t$, we denote its observed outcome with $z_t$. Note that we assume this outcome is always fixed, i.e. repeatedly performing a test or performing a test again at a later time will always result in the same outcome.

In our setting, an algorithm selects tests in an iterative and adaptive way, where the challenging part is to choose an order such that we need only a small total number of tests to decide upon the correct hypothesis. In iteration $i$, we use $\mathcal{A}_i \subseteq \mathcal{T}$ to denote the tests that have already been performed, and $\mathcal{S}_i = \{\bigcup_{t \in \mathcal{A}_i}(t, z_t)\}$ for the tests and the corresponding outcomes seen so far.

We use a probabilistic model to describe the knowledge that we have at any point in time. In particular, we follow a Bayesian perspective and

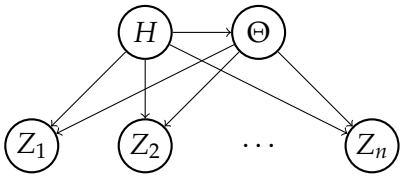


**Figure 3.1:** The graphical model which we use to describe the influence of noise. Each node stands for a random variable and the directed edges depict their dependences. $H$ denotes the true hypothesis, $\Theta$ models the noise, and $Z_1, \ldots, Z_n$ are the outcomes of active learning tests.

assume that in every iteration $i$, we have a way of computing the posterior distribution $\mathrm{p}_{\mathrm{post}}(H \,|\mathcal{S}_i)$, which is the probability over all hypotheses given the observations we have made so far. The set of all hypotheses that are consistent with the current observations at iteration $i$ is called the *version space* and defined as $\mathcal{V}(\mathcal{S}_i) := \{h \in \mathcal{H} : \mathrm{p}_{\mathrm{post}}(h \,|\mathcal{S}_i) > 0\}$.

Our ultimate goal is to pick a set of tests to perform such that $|\mathcal{V}(\mathcal{S}_i)| = 1$ and the total cost is as low as possible. For the rest of this thesis, for simplicity, we assume without loss of generality that the cost of all tests is uniform,[2] which results in a setting where we want to pick as few tests as possible. Also note that in the noiseless case, we further assume that the true hypothesis can always uniquely be determined by performing all tests. In practice, however, we expect noise to enter the picture, which leads us to the description of the formal noise model that we use.

## 3.3 Noise Model

To describe noise and how it interacts with the observations an algorithms makes, we continue in the footsteps of Golovin et al. ([15]). We think of noise as influencing the outcomes of tests, which are observed by an algorithm, in such a way that repeatedly performing the same test will not help in detecting it.

While we still suppose that there exists a noise-free version of the data instances that the algorithm sees, with noisy observations, we loose the guarantee that the true hypothesis can always be determined if all tests are performed. The observed outcomes of tests might lead us to decide on a wrong hypothesis, or there might still be several hypotheses with non-zero probability after performing all tests such that we cannot determine the true one for sure.

Figure 3.1 depicts the graphical model of our noise. Formally, we model the noise with a random variable $\Theta$ that influences the test outcomes and

[2] In the experiments we perform, the cost of all test is indeed uniform. However, there are many scenarios where this is does not hold. Nevertheless, all methods we use can easily be extended to non-uniform test costs by e.g. weighting tests according to their costs.

depends on $H$. While this is a fairly general model, in the following chapters we investigate two different settings that can be viewed as its instances.

### 3.3.1 Active Selection of Data Points

In Chapter 4, we look at a classical active learning question: Given a set of data points in some high-dimensional space, for which ones should we select labels and in what order to train a linear classifier most efficiently? We restrict ourselves to the case of two classes, such that $\mathcal{Z} := \{-1, 1\}$ and the classifier is represented by a hyperplane that separates the two classes.

The set of hypotheses $\mathcal{H}$ in this case is a set of normal vectors of hyperplanes, where we assume that one of them represents the true classifier. The tests $\mathcal{T}$ are all the data points that are given, and a data point $\mathbf{x}_i$ upon being queried reveals its label $y_i \in \{-1, 1\}$ as test outcome. We imagine the random variable $\Theta$ depicting the noise to influence the labels that data points reveal upon selection.

### 3.3.2 Active Selection of Features

Another instance of the general setting is investigated in Chapter 5. Again, we assume that we have access to a set of data points $\mathcal{X}$ with corresponding class labels. However, now we are interested in inspecting as few of a data point's features as possible in order to classify it.

Thus, in this setting, $\mathcal{H}$ equals the set of possible classes, and the tests that we can choose are given by the features, i.e. the number of dimensions, a data point has. Outcomes reveal the value of a certain feature, where we assume for simplicity that every feature may only take on a finite set of values.[3] Here $\Theta$ models the set $\mathcal{X}$ that we are given, i.e. the noisy representation of data points.

A common way to represent this active selection of features are decision trees. We discuss the issue of when to stop selecting additional tests in a noisy setting, which is a crucial issue in any active learning setting.

## 3.4 Methodology

In each chapter, we start with a more detailed description of the setting and introduce the specific algorithms that we investigated. As machine learning techniques in the end prove their merit in practice, we present the results of extensive simulations on standard machine learning datasets.[4]

[3] However, we also discuss ways to deal with real-valued features.

[4] The datasets that we use are described in Appendix A in more detail, and for more information on the experimental setting, please refer to Appendix B.

The experimental results are presented after the theoretical sections, and are followed by a discussion of the lessons learned, as well as suggestions for promising future avenues of research.

Chapter 4

# Active Selection of Data Points

A common task in machine learning is to train a classifier from a provided set of labeled data points. We look at the setting where an algorithm learns from one data point at a time in an iterative way while being able to actively query corresponding class labels. The sequence of data points from which the algorithm learns is either outside of its control, leading to an *online*, stream-based setting,[1] or can be chosen by the algorithm itself, which we term *offline* learning in contrast to the first case.

Throughout this chapter, we assume that every dataset we deal with is standardized such that it has a mean of 0 and a standard deviation of 0.5 in every dimension, and that the data points the algorithm receives are independent and identically distributed.

## 4.1 Online, Active, and Semi-supervised Learning

Goldberg et al. ([14]) introduce the so-called OASIS algorithm, which combines online, active and semi-supervised learning in a Bayesian framework.

In each iteration $i$, the algorithm receives a data point $\mathbf{x}_i \in \mathbb{R}^d$ in a sequential fashion from a potentially endless data stream. We assume that each $\mathbf{x}_i$ has a corresponding label $y_i \in \{-1, 1\}$ that indicates which of two classes the data point belongs to.[2] While this label might or might not be revealed to the OASIS algorithm, it learns from both labeled and unlabeled data points. It can also decide to actively query the label of an unlabeled data point. However, we assume that querying labels is an expensive operation and should be kept to the necessary minimum. For example, querying a label

[1] We also assume that the data points cannot be saved and used again at a later point in time.

[2] While we restrict ourselves to binary classification, various ways to build an extension to multiple classes, which can be adopted to our case, are discussed in the literature. For example, Hsu and Lin ([21]) as well as Wang et al. ([47]) investigate multi-class approaches.

might result in the need for human interaction or performing an expensive experiment.

At its core, the OASIS algorithm assumes a dataset to be linearly separable[3] by a hyperplane, which means that there exists some $\mathbf{w} \in \mathbb{R}^d$ such that

$$\operatorname{sgn}(\mathbf{w}^\intercal \mathbf{x}_i) = \begin{cases} -1 & \text{if } y_i = -1 \\ 1 & \text{if } y_i = 1 \end{cases}$$

for all $i$.

A dataset may not be linearly separable in practice due to its nature or due to noise. However, we may still be able to rank hyperplanes by how likely they are, i.e. by how well they separate the dataset into two classes. Therefore, the algorithm uses a likelihood function $\mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}, \mathbf{w})$ that returns a high value if a point is more than a certain distance on the correct side of the hyperplane, and a low value if it is on the wrong side of the hyperplane.

In addition, we want to make use of unlabeled data points in the learning process, and we do so by what is called the gap assumption. This assumption states that there is a gap in the hyperspace between data points with the label $y = 1$ and data points belonging to the class $y = -1$, describing the intuitive notion that data points form clusters according to their classes.

As noted above, the OASIS algorithm at iteration $i$ may not receive the label $y_i$ along with the data point $\mathbf{x}_i$. In such a case, we write $y_i = \emptyset$. The likelihood function is defined as

$$\mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}, \mathbf{w}) := \begin{cases} \min\{1-\gamma, \max\{\gamma\alpha, c\exp(-yc'\mathbf{w}^\intercal \mathbf{x})\}\} & \text{if } y \in \{-1, 1\} \\ 1 - \sum_{y \in \{-1,1\}} \mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}, \mathbf{w}) & \text{if } y = \emptyset \end{cases}$$

where $\gamma = 0.2$, $\alpha = 0.5$, $c = \sqrt{(1-\gamma)\gamma\alpha}$, and $c' = \log\left(\frac{\gamma\alpha}{c}\right)$. Figure 4.1 shows a plot of the likelihood function and its components.

Let us have a closer look at this function. If the product $\mathbf{w}^\intercal \mathbf{x}$ has a value less than $-1$ or greater than 1, meaning the data point $\mathbf{x}$ has a certain distance to the hyperplane, the function is flat and returns a high or low value, depending on whether $\mathbf{x}$ is on the right side of the hyperplane or not. In between, we observe an exponential descent respectively ascent.

Note also that the sum $\sum_{y \in \{-1,1\}} \mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}, \mathbf{w})$ always evaluates to less than 1, which leaves some probability mass for the third case when $y = \emptyset$, i.e. when the class label of $\mathbf{x}$ is unknown. In such a case, we prefer hyperplanes with a certain distance to $\mathbf{x}$, because according to our gap assumption, the hyperplane defined by $\mathbf{w}$ should not lie very close to any data point. This is achieved by returning a higher value when $\mathbf{w}^\intercal \mathbf{x}$ is greater than 1 or smaller than $-1$.

[3] Goldberg et al. mention that the algorithm can also be kernelized, removing the linearity restriction.

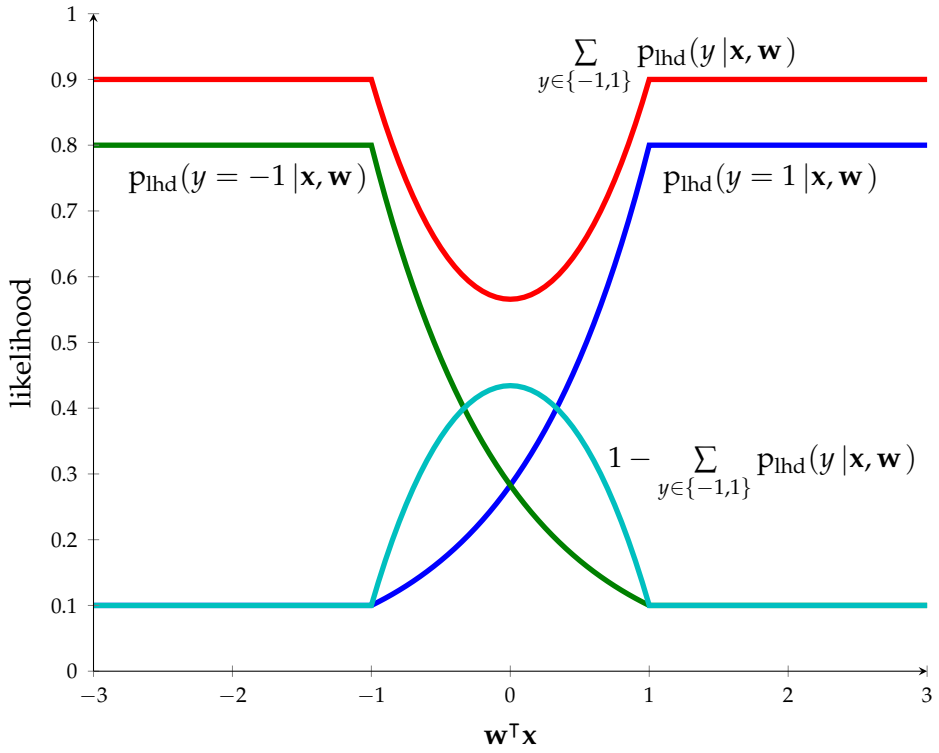


**Figure 4.1:** Plot of the OASIS likelihood function $p_{\text{lhd}}(y\,|\mathbf{x},\mathbf{w})$ and its components. If in some iteration the algorithm receives the label $y$, the likelihood function is given by $p_{\text{lhd}}(y=1\,|\mathbf{x},\mathbf{w})$ or $p_{\text{lhd}}(y=-1\,|\mathbf{x},\mathbf{w})$, depending on the class label. If $y=\emptyset$, i.e. the class label has not been provided, the likelihood function becomes $1-\sum_{y\in\{-1,1\}} p_{\text{lhd}}(y\,|\mathbf{x},\mathbf{w})$. Directly used by the OASIS algorithm, however, is its complement $\sum_{y\in\{-1,1\}} p_{\text{lhd}}(y\,|\mathbf{x},\mathbf{w})$, which intuitively assigns a higher likelihood to data points that have a certain distance from the decision boundary, thus modeling the gap assumption.

### 4.1.1 Bayesian Model

So far, we have seen how a data point interacts with a single hyperplane vector $\mathbf{w}$. However, the OASIS algorithm looks at the problem from a Bayesian perspective and works with a probability distribution over the space of all hyperplanes.

To arrive at a Bayesian model, we first need to define a prior distribution $p_{\text{prior}}(\mathbf{w})$ over hyperplane vectors $\mathbf{w}$. While there are many possibilities, we follow Goldberg et al. in using independent centered Cauchy distributions[4]

[4] A Cauchy distribution is a special form of Student's t-distribution with one degree of freedom. See Bishop ([2, pp. 691–692]) for details.

in every dimension. Thus, we define

$$\mathrm{p}_{\mathrm{prior}}(\mathbf{w}) := \prod_{i=1}^{d} \mathrm{Cauchy}\Big(\mathbf{w}_{[i]}; 0, \lambda\Big)$$

where

$$\mathrm{Cauchy}(x; 0, \lambda) = \frac{\lambda}{\pi\,(x^2 + \lambda^2)}$$

with $\lambda$ denoting the scale parameter of the distribution.

Let $\mathcal{D}_i = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_i, y_i)\}$ be the data points and labels the algorithm has seen up to iteration $i$. Given the prior distribution and our likelihood function, we arrive at the posterior distribution $\mathrm{p}_{\mathrm{post}}(\mathbf{w}\,|\mathcal{D}_i\,)$ by applying Bayes' rule, which yields

$$\mathrm{p}_{\mathrm{post}}(\mathbf{w}\,|\mathcal{D}_i\,) = \frac{\prod_{j=1}^{i} \mathrm{p}_{\mathrm{lhd}}\big(y_j\,\big|\mathbf{x}_j, \mathbf{w}\big)\,\mathrm{p}_{\mathrm{prior}}(\mathbf{w})}{\int \prod_{j=1}^{i} \mathrm{p}_{\mathrm{lhd}}\big(y_j\,\big|\mathbf{x}_j, \mathbf{w}'\big)\,\mathrm{p}_{\mathrm{prior}}(\mathbf{w}')\;\mathrm{d}\mathbf{w}'}$$

since we assume the data points that the algorithm sees are independent and identically distributed.

Furthermore, we are interested in predicting the label $y_{i+1}$ of the next data point $\mathbf{x}_{i+1}$, and can use the predictive distribution

$$\mathrm{p}_{\mathrm{pred}}(y\,|\mathbf{x}_{i+1}, \mathcal{D}_i\,) = \int \mathrm{p}_{\mathrm{lhd}}(y\,|\mathbf{x}_{i+1}, \mathbf{w}\,)\,\mathrm{p}_{\mathrm{post}}(\mathbf{w}\,|\mathcal{D}_i\,)\;\mathrm{d}\mathbf{w}.$$

for this task. However, we do not want to include the case where $y = \emptyset$, because we assume that a data point does have the label 1 or $-1$. Conditioning on $y \in \{-1, 1\}$ yields

$$\mathrm{p}_{\mathrm{pred}}(y\,|\mathbf{x}_{i+1}, \mathcal{D}_i, y \in \{-1, 1\}\,) = \frac{\mathrm{p}_{\mathrm{pred}}(y\,|\mathbf{x}_{i+1}, \mathcal{D}_i\,)}{\sum\limits_{y' \in \{-1,1\}} \mathrm{p}_{\mathrm{pred}}(y'\,|\mathbf{x}_{i+1}, \mathcal{D}_i\,)}$$

which we can use to predict labels of new data points. In particular, $\mathrm{p}_{\mathrm{pred}}(y\,|\mathbf{x}_{t+1}, \mathcal{D}_i, y \in \{-1, 1\}\,)$ can be used as a classifier by simply returning the label with the highest probability, i.e. by setting

$$y_{i+1} = \operatorname*{arg\,max}_{y' \in \{-1,1\}} \mathrm{p}_{\mathrm{pred}}\big(y'\,|\mathbf{x}_{i+1}, \mathcal{D}_i, y \in \{-1, 1\}\big)\,.$$

### 4.1.2 Particle Filter Approximation

We have discussed probability distributions without specifying how they are actually represented. The OASIS algorithm uses a particle filter[5] with

[5]Please refer to Doucet and Johansen ([11]) for a comprehensive introduction and discussion of particle filtering methods.

$m$ particles to model, approximate and keep track of these distributions. A single particle $\mathbf{w}_j$ describes one hyperplane; its corresponding weight $w_j$ defines the probability of this particle representing the true decision boundary.

Formally, we attain an empirical approximation $\hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_i)$ of the posterior distribution by summing over $m$ weighted particles. As we assume that the particle weights $w_j$ are normalized and sum up to one, we have

$$\mathrm{p}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_i) \approx \hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_i) = \sum_{j=1}^{m} w_j \delta(\mathbf{w} - \mathbf{w}_j)$$

where $\delta(\mathbf{w})$ is the Dirac delta function. The general predictive distribution becomes

$$\begin{aligned}
\hat{\mathrm{p}}_{\mathrm{pred}}(y \,|\mathbf{x}_{i+1}, \mathcal{D}_i) &= \int \mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}_{i+1}, \mathbf{w}) \, \hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_i) \, \mathrm{d}\mathbf{w} \\
&= \int \mathrm{p}_{\mathrm{lhd}}(y \,|\mathbf{x}_{i+1}, \mathbf{w}) \sum_{j=1}^{m} w_j \delta(\mathbf{w} - \mathbf{w}_j) \, \mathrm{d}\mathbf{w} \\
&= \sum_{j=1}^{m} w_j \, \mathrm{p}_{\mathrm{lhd}}\big(y \,|\mathbf{x}_{i+1}, \mathbf{w}_j\big)
\end{aligned}$$

and we use

$$\begin{aligned}
\hat{\mathrm{p}}_{\mathrm{pred}}(y \,|\mathbf{x}_{i+1}, \mathcal{D}_i, y \in \{-1, 1\}) &= \frac{\hat{\mathrm{p}}_{\mathrm{pred}}(y \,|\mathbf{x}_{i+1}, \mathcal{D}_i)}{\sum_{y' \in \{-1,1\}} \hat{\mathrm{p}}_{\mathrm{pred}}(y' \,|\mathbf{x}_{i+1}, \mathcal{D}_i)} \\
&= \sum_{j=1}^{m} w_j \frac{\mathrm{p}_{\mathrm{lhd}}\big(y \,|\mathbf{x}_{i+1}, \mathbf{w}_j\big)}{\sum_{y' \in \{-1,1\}} \mathrm{p}_{\mathrm{lhd}}\big(y' \,|\mathbf{x}_{i+1}, \mathbf{w}_j\big)}
\end{aligned}$$

to predict labels of new data points.

### 4.1.3 Online Learning

The OASIS algorithm was designed for an online setting, where it sees data points sequentially while keeping track of and updating the posterior distribution $\hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_i)$. Before any data points arrive, the algorithm starts with sampling $m$ particles from the prior distribution[6] $\mathrm{p}_{\mathrm{prior}}(\mathbf{w})$ and assigning each particle the uniform weight $1/m$.

In step $i+1$ we receive a pair $(\mathbf{x}_{i+1}, y_{i+1})$, which we use to update the current model that is described by the posterior distribution. If $y_{i+1} = \emptyset$, meaning

[6] In our case, one simple way to sample from a Cauchy distribution with zero mean, as noted by Marsaglia ([26]), is to calculate the ratio $X/Y$ of two random variables $X$ and $Y$ that have both been sampled from the standard normal distribution and then multiplying the result with the scale factor $\lambda$.

the algorithm did not receive a label for the data point $\mathbf{x}_{i+1}$, we calculate a score for $\mathbf{x}_{i+1}$ and request the label $y_{i+1} \in \{-1, 1\}$ if this score is lower than a given threshold $s_0$.

The new posterior distribution $\hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_{i+1})$ after step $i$ is calculated based on the previous posterior, which now acts as a prior, and the likelihood. The two-step process involves first adjusting the current particle weights, yielding intermediate weights $w'_j = w_j \,\mathrm{p}_{\mathrm{lhd}}\big(y_{i+1} \,|\mathbf{x}_{i+1}, \mathbf{w}_j\big)$. In the second step, these are then normalized to compute the new weights $w''_j = w'_j / (\sum_k w'_k)$, which gives us

$$\begin{aligned}\sum_{j=1}^{m} w''_j \delta(\mathbf{w} - \mathbf{w}_j) &= \frac{1}{s_w} \sum_{j=1}^{m} w_j \,\mathrm{p}_{\mathrm{lhd}}\big(y_{i+1} \,|\mathbf{x}_{i+1}, \mathbf{w}_j\big)\, \delta(\mathbf{w} - \mathbf{w}_j) \\ &= \hat{\mathrm{p}}_{\mathrm{post}}(\mathbf{w} \,|\mathcal{D}_{i+1})\end{aligned}$$

where $s_w := \sum_k w'_k$ is the normalization factor.

### 4.1.4 Resample-Move Step

Using the methods described, the algorithm is able to adjust the weights of particles according to the data stream it sees. However, as the initial particles are sampled randomly from the uninformative prior distribution, we expect that after some time, many particles will have small weights, whilst only a few will influence the posterior distribution significantly with big weights. In other words, we expect many particles to become uninformative and therefore useless after some time has passed.

To overcome this issue and use the fixed number of particles effectively, Goldberg et al. utilize the so-called resample-move algorithm as introduced by Gilks and Berzuini ([13]), which shifts the particle mass to the informative regions of the posterior distribution.

First, we resample new particles from the set of our current particles according to their weights, where Algorithm 4.1 depicts the *systematic resampling* method that we use. After obtaining the resampled particles, we reset all particle weights to $\frac{1}{m}$.

Resampling ensures that the weight of any particle does not become so small as to make that particle insignificant. However, repeatedly performing resampling over time will cause the algorithm to work with fewer and fewer unique particles, because with high probability particles with big weights will be selected multiple times by systematic resampling.

Therefore, a subsequent *move* step shifts particles by applying one step of the Metropolis-Hastings algorithm[7] as depicted in Algorithm 4.2. Given a

[7]The Metropolis algorithm was proposed by Metropolis et al. ([27]) and later extended by

**Algorithm 4.1** The resampling step of the resample-move algorithm, which takes as input a set of particles and corresponding particle weights. We use unbiased systematic resampling based on the description by Doucet and Johansen ([11]).

**procedure** RESAMPLE($\{\mathbf{w}_i : 1 \leq i \leq m\}$, $\{w_i : 1 \leq i \leq m\}$)
  $u_1 \sim \mathcal{U}\left(0, \frac{1}{m}\right)$ ▷ sample $u_1$ from uniform distribution
  **for** $i = 2, \ldots, m$ **do**
    $u_i \leftarrow u_1 + \frac{i-1}{m}$
  **end for**
  **for** $i = 1, \ldots, m$ **do**
    $p_i \leftarrow \left|\left\{1 \leq j \leq m : \sum_{k=1}^{j} w_k < u_i\right\}\right| + 1$
    $\mathbf{w}'_i \leftarrow \mathbf{w}_{p_i}$
  **end for**
  **return** new particles $\{\mathbf{w}'_i : 1 \leq i \leq m\}$
**end procedure**

**Algorithm 4.2** Move step of the resample-move algorithm. It requires the set of current particles $\{\mathbf{w}_i \mid 1 \leq i \leq m\}$, the data points received and observations made so far $\mathcal{D}_i = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_i, y_i)\}$, a symmetric proposal distribution $\mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w})$, and the buffer length $\tau$ as input arguments.

**procedure** MOVE($\{\mathbf{w}_i : 1 \leq i \leq m\}$, $\mathcal{D}_i$, $\mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w})$, $\tau$)
  **for** $i = 1, \ldots, m$ **do**
    $\hat{\mathbf{w}}_i \sim \mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w}_i)$ ▷ sample new particle from proposal distribution
    $p_{\mathbf{w}_i} \leftarrow \mathrm{p}_{\text{prior}}(\mathbf{w}_i) \prod_{k=t-\tau+1}^{t} \mathrm{p}_{\text{lhd}}(y_k \,|\mathbf{x}_k, \mathbf{w}_i)$
    $p_{\hat{\mathbf{w}}_i} \leftarrow \mathrm{p}_{\text{prior}}(\hat{\mathbf{w}}_i) \prod_{k=t-\tau+1}^{t} \mathrm{p}_{\text{lhd}}(y_k \,|\mathbf{x}_k, \hat{\mathbf{w}}_i)$
    $\alpha_i \leftarrow \min\left(1, p_{\hat{\mathbf{w}}_i} / p_{\mathbf{w}_i}\right)$ ▷ $\alpha_i$ denotes the acceptance probability
    $\beta_i \sim \mathcal{U}(0, 1)$
    **if** $\beta_i \leq \alpha_i$ **then**
      $\mathbf{w}'_i \leftarrow \hat{\mathbf{w}}_i$
    **else**
      $\mathbf{w}'_i \leftarrow \mathbf{w}_i$
    **end if**
  **end for**
  **return** new particles $\{\mathbf{w}'_i : 1 \leq i \leq m\}$
**end procedure**

**Algorithm 4.3** The OASIS algorithm as described by Goldberg et al. ([14]). We assume that it has access to an unlimited stream of data points $\mathbf{x}_1, \mathbf{x}_2, \ldots$ and corresponding labels $y_1, y_2, \ldots$. If $y_i = \emptyset$, i.e. a label has not been revealed, the algorithm can query the label via the function $\text{label}(\mathbf{x}_i)$. In addition, the algorithm requires as parameters the number of particles $m$, a prior distribution $\mathrm{p}_{\text{prior}}(\mathbf{w})$, a proposal distribution $\mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w})$, the score threshold $s_0$, and the buffer size $\tau$.

**procedure** OasisAlgorithm($m$, $\mathrm{p}_{\text{prior}}(\mathbf{w})$, $\mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w})$, $s_0$, $\tau$)
  **for** $j = 1, \ldots, m$ **do** ▷ sample initial particles
    $\mathbf{w}_j \sim \mathrm{p}_{\text{prior}}(\mathbf{w})$
    $w_j \leftarrow \frac{1}{m}$
  **end for**
  $\mathcal{D}_0 \leftarrow \emptyset$
  **for** $i = 0, \ldots$ **do**
    Receive $\mathbf{x}_{i+1}$ and $y_{i+1}$
    **if** $y_{i+1} = \emptyset \wedge \text{score}(\mathbf{x}_{i+1} \,|\mathcal{D}_i) < s_0$ **then**
      $y_{i+1} \leftarrow \text{label}(\mathbf{x}_{i+1})$
    **end if**
    **for** $j = 1, \ldots, m$ **do** ▷ update particle weights
      **if** $y_{i+1} \neq \emptyset$ **then**
        $w_j \leftarrow w_j \, \mathrm{p}_{\text{lhd}}\big(y = y_{i+1} \,\big|\mathbf{x}_{i+1}, \mathbf{w}_j\big)$
      **else**
        $w_j \leftarrow w_j \, \mathrm{p}_{\text{lhd}}\big(y \in \{-1, 1\} \,\big|\mathbf{x}_{i+1}, \mathbf{w}_j\big)$
      **end if**
    **end for**
    **for** $j = 1, \ldots, m$ **do** ▷ normalize particle weights
      $w_j \leftarrow w_j / \left(\sum_{k=1}^{m} w_k\right)$
    **end for**
    **if** $1 / \sum_{k=1}^{m} w_k^2 < \frac{m}{2}$ **then** ▷ resample-move step
      $\{\mathbf{w}_j : 1 \leq j \leq m\} \leftarrow \text{Resample}(\{\mathbf{w}_j\}, \{w_j\})$
      $\{\mathbf{w}_j : 1 \leq j \leq m\} \leftarrow \text{Move}(\{\mathbf{w}_j\}, \mathcal{D}_i, \mathrm{q}(\hat{\mathbf{w}} \,|\mathbf{w}), \tau)$
      **for** $j = 1, \ldots, m$ **do**
        $w_j \leftarrow \frac{1}{m}$
      **end for**
    **end if**
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup \{(\mathbf{x}_{i+1}, y_{i+1})\}$
  **end for**
**end procedure**

symmetric proposal distribution $q(\hat{\mathbf{w}} \mid \mathbf{w})$, for each particle $\mathbf{w}_i$ we generate an alternative $\hat{\mathbf{w}}_i$ and use the unnormalized posterior to compute two values $p_{\mathbf{w}_i}$ and $p_{\hat{\mathbf{w}}_i}$ that describe how well $\mathbf{w}_i$ and $\hat{\mathbf{w}}_i$ fit the observed data. If the proposed particle fits the data better, it is always accepted, and otherwise it is accepted randomly according to the acceptance probability, which is given by the ratio $p_{\hat{\mathbf{w}}_i} / p_{\mathbf{w}_i}$.

To speed up these calculations and arrive at a constant time complexity, Goldberg et al. suggest to consider only the past $\tau$ data points to compute the acceptance probability, which results in an approximation to the Metropolis-Hastings algorithm if $\tau$ is smaller than the number of data points the algorithm has received so far.[8]

Computing the Metropolis-Hastings step involves a lot of computations, even when considering only the past $\tau$ particles. Therefore, the OASIS algorithm does not apply the resample-move algorithm in every loop iteration but only when the *effective sample size*[9] given by $\left(\sum_{i=1}^{m} w_i\right)^2 / \sum_{i=1}^{m} w_i^2$ becomes less than $m/2$. Note that as we normalize the particle weights beforehand such that they sum up to one, the calculation simplifies to $1 / \sum_{i=1}^{m} w_i^2$.

We now have all the pieces together that make up the OASIS algorithm. It is shown in its final form in Algorithm 4.3.

### 4.1.5 Score Functions

To decide whether or not to query the label of an unlabeled data point $\mathbf{x}_{i+1}$, the OASIS algorithm uses a function to compute a score and actively asks for the label $y_{i+1}$ if this score is lower than a certain threshold $s_0$. Thus, we interpret a score as a measure of certainty about the label of a data point, with high scores denoting that the algorithm is very sure about a data point's label, whereas low scores correspond to high uncertainty.

In this section, we present different score functions that can be used for this task, and will later evaluate how they perform in practice on several datasets. Table 4.1 gives an overview of these score functions along with the abbreviations we use to refer to them.

#### OASIS Score

Goldberg et al. use a score function that was proposed by Nowak ([31]) for noisy generalized binary search. We call it the OASIS score function and note

Hastings ([17]). It knows wide applications and features in various lists of the most important algorithms of the 20th century, for example in one provided by Dongarra and Sullivan ([10]).

[8] Goldberg et al. also note that considering only a fixed window of the past could enable to algorithm to handle concept drift in situations where this is desired.

[9] See Doucet and Johansen ([11]) as well as Ridgeway and Madigan ([33]) for details.

| Score Function Name | Abbreviation |
|---|---|
| OASIS Score | OASIS |
| Generalized Binary Search Score | GBS |
| Uncertainty Score | UC |

**Table 4.1:** Overview of the score functions presented in this section along with their corresponding abbreviations.

that it can be simplified to

$$\begin{aligned}
\text{score}_{\text{OASIS}}(\mathbf{x} \,|\mathcal{D}_i) &:= \left| \sum_{j=1}^{m} w_j \operatorname*{arg\,max}_{y \in \{-1,1\}} \mathrm{p}_{\text{lhd}}\big(y \,|\mathbf{x}, \mathbf{w}_j\big) \right| \\
&= \left| \sum_{j=1}^{m} w_j \operatorname*{arg\,max}_{y \in \{-1,1\}} \{\min\{1-\gamma, \max\{\gamma\alpha, c\exp\big(-yc'\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}\big)\}\}\} \right| \\
&= \left| \sum_{j=1}^{m} w_j \operatorname*{arg\,max}_{y \in \{-1,1\}} -y\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x} \right| \\
&= \left| \sum_{\{j|\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}>0\}} w_j - \sum_{\{j|\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}<0\}} w_j \right|.
\end{aligned}$$

This holds because whether $y = -1$ or $y = 1$ is selected by the argument of the maximum function depends solely on the result $\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}$, since all other variables are constant.[10]

### Generalized Binary Search Score

In an earlier work, Nowak ([30]) discusses a slightly different version of *generalized binary search* (GBS). From the definition

$$\text{score}_{\text{GBS}}(\mathbf{x} \,|\mathcal{D}_i) := \min\left\{ \sum_{\{j|\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}>0\}} w_j, \sum_{\{j|\mathbf{w}_j{}^{\mathsf{T}}\mathbf{x}<0\}} w_j \right\}$$

of its corresponding score function, we see that it is very close in spirit to the OASIS score function. Note, however, that in the case of GBS, the more certain the algorithm is about the label of a data point, the lower its score will be, in contrast to the other score functions. Therefore, when using it, we adjust the OASIS algorithm so that it tests whether the score is greater than the threshold $s_0$.

[10] This can be easily verified by looking at the likelihood function form as presented in Figure 4.1.

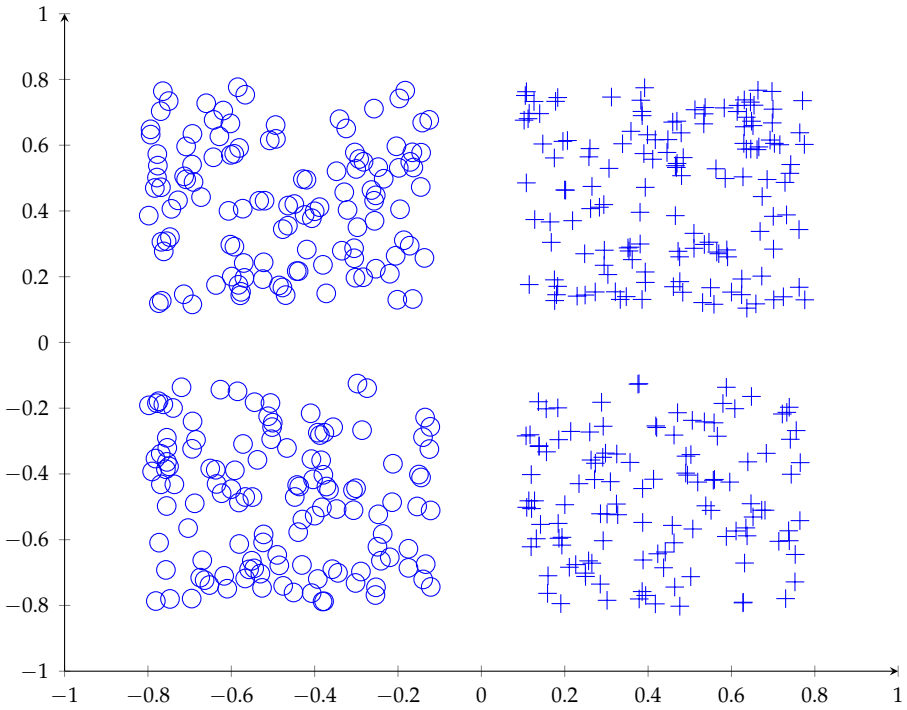


**Figure 4.2:** An instance of the synthetic DICED dataset in two dimensions with 500 data points. The two different classes are distinguished by small circles versus plus signs. Note that it is shown normalized, i.e. with a mean of 0 and a standard deviation of 0.5.

### Uncertainty Score

Another common heuristic is to choose labels for data points that we are unsure about in our current model, an idea formulated by Lewis and Gale ([23]). The uncertainty score

$$\text{score}_{\text{UC}}(\mathbf{x} \,|\mathcal{D}_i) := \left|\hat{p}_{\text{pred}}(y = 1 \,|\mathbf{x}, \mathcal{D}_i, y \in \{-1, 1\}) - 0.5\right|$$

achieves this by predicting the label for the data point $\mathbf{x}$ with the conditional predictive distribution.

## 4.1.6 Example

Having described the OASIS algorithm, let us now look at a simple example that shows it in operation.[11] We use the synthetic DICED dataset in two dimensions and with 500 samples as shown in Figure 4.2. The correct classification hyperplane is a vertical line through the origin that separates the left and the right halves.

[11] We base this example on the motivational example given by Goldberg et al.

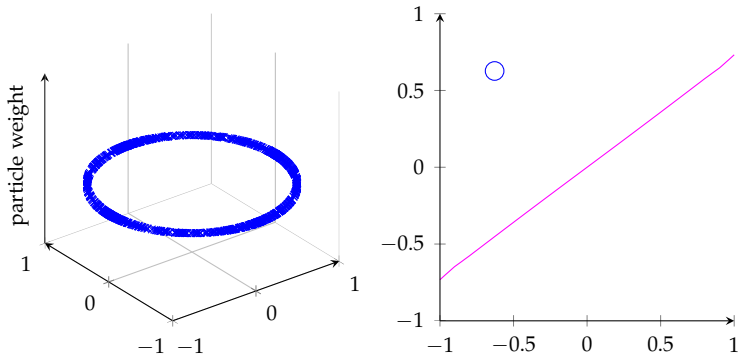


**Figure 4.3:** The initial particle filter approximation over the space of all hyperplanes together with the first data point the OASIS algorithm receives. In the beginning, all particles have the same weights as shown on the left, resulting in a random decision boundary, which is indicated by a purple line in the feature space on the right.

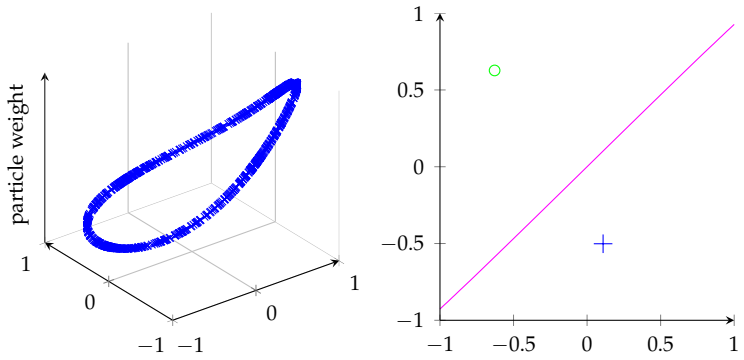


**Figure 4.4:** In iteration two, the particle distribution has adjusted to the first data point and now prefers some hyperplanes with bigger weights. The next data point that is given together with its label to the OASIS algorithm is shown in blue on the right.

We only provide the first two data point labels to the algorithm and use the OASIS score function with a threshold of $s_0 := 0.1$. Furthermore, we set the number of particles $m := 1000$ and the buffer size for the resample-move algorithm $\tau := 100$. For the purpose of easily understandable plots, we deviate in this example from the OASIS algorithm and keep all particle vectors normalized to a length of one.[12]

Figure 4.3 depicts the space of hyperplane vectors, i.e. particles, at the first iteration of the OASIS algorithm together with the current decision boundary and the first data point that is received. After updating the

[12]The OASIS algorithm does not normalize the particle vectors, i.e. they can have arbitrary lengths.

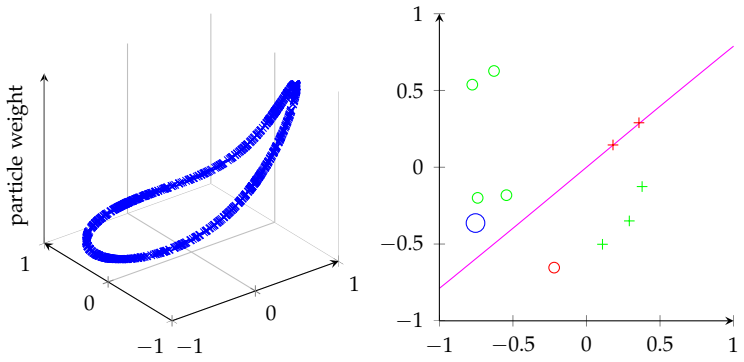


**Figure 4.5:** After processing 10 data points, the distribution over the space of all hyperplanes has become highly peaked. On the right, we can see that the current decision boundary wrongly classifies three of the data points seen so far.

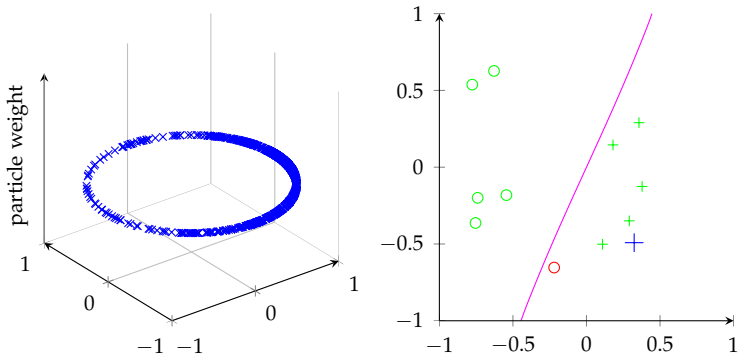


**Figure 4.6:** After the resample-move step of iteration 11, more particles are concentrated in regions that held particles with big weights beforehand, as can be seen on the left. The decision boundary starts to resemble the true boundary more and more, as depicted on the right.

particle filter model, the represented posterior distribution over the space of particles changes is shown in Figure 4.4. Note that in contrast to the two axes corresponding to the feature dimensions, the third axis depicting the particle weights in the space of particles does not show any quantitative information, since we are mainly interested in a qualitative, relative comparison between the weights of particles. Absolute values would not tell us something useful per se, as they heavily depend on the total number of particles due to normalization.

While the first two class labels are provided to the algorithm, the scores that are computed for the subsequent data points are all higher than the score threshold. The algorithm, therefore, learns from unlabeled data points for some time. This changes again in iteration 11, which Figure 4.5 shows. At this

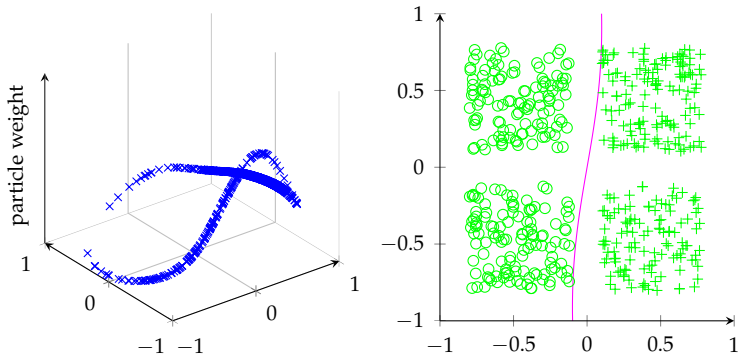


**Figure 4.7:** Having processed all 500 data points, the decision boundary shown on the right classifies all data points correctly. Its curved nature is due to the exponential ascent respectively descent in the likelihood function used. In the space of hyperplanes on the left, there is a peak around the correct classification hyperplane, even though many particles still reside (with smaller weights) in other regions.

point, the computed score is lower than the threshold for the first time and the OASIS algorithm actively requests a label. In addition, after this update, the effective sample size drops below the limit $m/2$ and the resample-move algorithm is invoked, which resamples the particles and resets their weights. The resulting situation is pictured in Figure 4.6.

After running the OASIS algorithm on all 500 data samples, the particle model, as shown in Figure 4.7, comes close the correct hyperplane. At this point, the algorithm queried 10 class labels in addition to the first two that were provided, and would have made six mistakes when classifying data points as they were received. If the final model is used to classify the full dataset, zero mistakes are made.

## 4.2 Offline Active Learning

The OASIS algorithm provides an integrated framework for online, active and semi-supervised learning. However, its active selection of data points is not particularly sophisticated, since it uses a simple threshold to decide whether or not to query the label of an unlabeled data point. While we introduced alternative active learning score functions, it is difficult to compare their performance because the results depend on the sequence of data points that the algorithm receives. Therefore, we now remove the online and semi-supervised aspects from the previous setting and focus on the active learning part.

Let us assume that we are given a dataset $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. First, we run a modified version of the OASIS algorithm on this dataset that does neither

**Algorithm 4.4** Schematic overview of the offline active learning algorithm. It returns a classification model and takes a prior distribution $\mathrm{p}_{\text{prior}}(\mathbf{w})$, some score function, and the maximal number of labels to query $n_{\max}$ as parameters. The particle model used is the same as in the OASIS algorithm depicted in Algorithm 4.3, except that no resample-move step is applied. As prior distribution, we use the resulting posterior from a run of the OASIS algorithm without any labels provided.

**procedure** OFFLINEACTIVELEARNING($\mathrm{p}_{\text{prior}}(\mathbf{w})$, score$(\cdot)$, $n_{\max}$)
  initialize particle model based on $\mathrm{p}_{\text{prior}}(\mathbf{w})$
  $\mathcal{S}_0 \leftarrow \emptyset$
  **for** $i = 1, \ldots, n_{\max}$ **do**
    $\mathbf{x}_i^\star \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} \text{score}(\mathbf{x} \,|\mathcal{S}_{i-1})$
    $y_i^\star \leftarrow \text{label}(\mathbf{x}_i^\star)$
    update particle model
    $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup \{(\mathbf{x}_i^\star, y_i^\star)\}$
  **end for**
  **return** particle model
**end procedure**

query nor receive any labels. This results in a particle filter representation of the data distribution, which we regard as a prior distribution for the subsequent procedure. By calculating a score for every data point, we then select the most informative one and request its label, which we subsequently use to update the particle weights of the posterior distribution. Iterating this procedure thus allows us to become more and more sure about the right hyperplane, whose weight more and more dominates all others. Algorithm 4.4 depicts the procedure in more detail.

Stating the problem in terms of the general noise model introduced in Section 3.3, here the set of hypotheses $\mathcal{H}$ corresponds to the set of particles $\mathcal{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$, i.e. the set of normal vectors of hyperplanes among which we want to select the correct one. The set of tests $\mathcal{T}$ equals the set of data points $\mathcal{X}$, and the outcomes of a test $\mathcal{Z} = \{-1, 1\}$ are the class labels. Recall that we write $z_t$ to denote the observed outcome of test $t$, and $\mathcal{S}_i = \{\bigcup_{t \in \mathcal{A}_i}(t, z_t)\}$ are the tests $\mathcal{A}_i$ with corresponding observed outcomes up to iteration $i$. The noise variable $\Theta$ cannot be directly observed, but influences the outcomes that we see, meaning that heavy noise influence could potentially cause the algorithm to receive a wrong class label $y_i$ that it queries for some data point $\mathbf{x}_i$.

Within this setting, we again use all score functions that we defined in Section 4.1.5. In addition, we consider a random (RD) score that simply returns a random number from the interval $[0; 1]$ for some data point $\mathbf{x}$, and now introduce the EC² score function.

### 4.2.1 EC² Score Function

The EC² criterion proposed by Golovin et al. ([15]) builds on the general noise model we introduced in Section 3.3, and splits the set of hypotheses $\mathcal{H}$ into $k$ disjunct hypothesis classes such that $\mathcal{H} = \biguplus_{i=1}^{k} \mathcal{H}_i$. We can think of some $\mathcal{H}_i$ as a set of hypotheses that are very similar, but slightly different because of noise, such that we ultimately want to decide upon one hypothesis class instead of a single hypothesis. Formally, we therefore require $\mathcal{V}(\mathcal{S}_i) \subseteq \mathcal{H}_i$ for some $i$ to make our decision instead of $|\mathcal{V}(\mathcal{S}_i)| = 1$ in the noiseless case.

We define a graph whose nodes are the hypotheses $\mathcal{H}$ and whose edges $\mathcal{E} = \bigcup_{1 \leq i < j \leq k} \{\{h, h'\} : h \in \mathcal{H}_i, h' \in \mathcal{H}_j\}$ connect all hypotheses in different hypothesis classes. We rule out hypotheses that are inconsistent with the observations made by *cutting* edges, i.e. by eliminating them from the graph. The meaning of an edge $\{h, h'\}$ is that both hypotheses $h$ and $h'$ can be used to interpret the observed data equally well. As edges only exists between different hypothesis classes, among which we ultimately want to distinguish, our goal in terms of the EC² criterion is to cut all edges in the graph.

Let $\mathcal{E}(t, h) := \{\{h', h''\} \in \mathcal{E} : h'(t) \neq h(t) \vee h''(t) \neq h(t)\}$ be the set of edges that are cut given a test result $h(t) \in \mathcal{Z}$, which is obtained by evaluating the test $t$ on hypothesis $h$.[13] In our case, we can compute $h(t)$ for some hypothesis $h$, given by a hyperplane vector $\mathbf{w}$, and a test $t$, which corresponds to a data point $\mathbf{x}$, by determining on which side of the hyperplane the data point lies, i.e. $h(t) = \operatorname{sgn}(\mathbf{w}^\intercal \mathbf{x})$.

Using this graph, the EC² score function calculates the weighted sum of edges that would be cut in expectation over all hypotheses by selecting a data point $\mathbf{x}$, and is defined as

$$\operatorname{score}_{\text{EC}^2}(\mathbf{x} \,|\mathcal{S}_i) := \mathbf{E}_H[f_{EC^2}(\mathcal{A}_i \cup \{\mathbf{x}\}, H) - f_{EC^2}(\mathcal{A}_i, H) \mid \mathcal{S}_i]$$

where the random variable $H$ in our case models hyperplane vectors,

$$f_{EC^2}(\mathcal{A}, h) := f_{\text{w}}\left(\bigcup_{\mathbf{x} \in \mathcal{A}} \mathcal{E}(\mathbf{x}, h)\right)$$

is a (adaptively) submodular function, and

$$f_{\text{w}}(\mathcal{E}) := \sum_{\{h, h'\} \in \mathcal{E}} \text{p}(h)\,\text{p}(h')$$

is a weight function based on the probability of hypothesis. Golovin et al. suggest to use uniform weights for every edge in the graph, i.e. to set $\text{p}(h) := 1/|\mathcal{H}|$.

[13] Note that if an edge connected to some hypothesis $h$ is cut, all other edges connected to $h$ are cut as well, and we eliminate $h$ from the version space.

**Algorithm 4.5** Algorithm for the efficient calculation of the EC² score of a data point $\mathbf{x}$ given the current observations $\mathcal{S}_i$. This method is based on the description by Golovin et al. ([15]).

**procedure** ScoreEC²($\mathbf{x}$, $\mathcal{S}_i$)
  **for** $i = 1, \ldots, k$ and $z \in \mathcal{Z}$ **do**
    $\alpha_{i,z} \leftarrow \mathrm{p}(\mathcal{H}_i \cap \mathcal{V}(\mathcal{S}_i \cup \{\mathbf{x}, z\}))$
  **end for**
  **for** $i = 1, \ldots, k$ **do**
    $\beta_i \leftarrow \sum_{z \in \mathcal{Z}} \alpha_{i,z}$
  **end for**
  **for** $z \in \mathcal{Z}$ **do**
    $\gamma_z \leftarrow \sum_{i=1}^{k} \alpha_{i,z} / \sum_{i=1}^{k} \beta_i$
  **end for**
  **return** $\frac{1}{2} \sum_{z \in \mathcal{Z}} \gamma_z \cdot \left( \left( \sum_{i=1}^{k} \beta_i \right)^2 - \sum_{i=1}^{k} \beta_i^2 + \left( \sum_{i=1}^{k} \alpha_{i,z} \right)^2 - \sum_{i=1}^{k} \alpha_{i,z}^2 \right)$
**end procedure**

### Efficient Calculation

Golovin et al. describe an efficient way of calculating the criterion by using the alternative definition

$$\mathrm{score}_{\mathrm{EC}^2}(\mathbf{x} \,|\mathcal{S}_i) := \frac{1}{2} \mathbf{E}_{z \sim Z|\mathcal{S}_i} \left[ f_{\mathcal{V}}(i,j) - f'_{\mathcal{V}}(i,j,\mathbf{x},z) \right]$$

where we define

$$f_{\mathcal{V}}(i,j) := \mathrm{p}(\mathcal{H}_i \cap \mathcal{V}(\mathcal{S}_i)) \, \mathrm{p}\big(\mathcal{H}_j \cap \mathcal{V}(\mathcal{S}_i)\big)$$

and

$$f'_{\mathcal{V}}(i,j,\mathbf{x},z) := \mathrm{p}(\mathcal{H}_i \cap \mathcal{V}(\mathcal{S}_i \cup \{\mathbf{x}, z\})) \, \mathrm{p}\big(\mathcal{H}_j \cap \mathcal{V}(\mathcal{S}_i \cup \{\mathbf{x}, z\})\big)$$

for clarity of notation. Furthermore, Golovin et al. note that the EC² score calculation according to this definition can be implemented as shown in Algorithm 4.5.[14]

### Submodularity Property

As noted earlier, the EC² score function is based on a helper function $f_{EC}$ that has the interesting submodularity property, which is formally defined as follows.[15]

[14] In addition, Golovin et al. introduce a fast approximation to the EC² criterion, which they call EffECXtive. However, this approximation cannot be applied within the particle filter setting, and therefore we do not consider it in this thesis.

[15] We base our definition of submodularity on Schrijver ([35, pp. 766–767]).

**Definition 4.1** *Let $\mathcal{P}(\mathcal{S})$ be the power set of some set $\mathcal{S}$. A set function $f : \mathcal{P}(\mathcal{S}) \to \mathbb{R}$ is called* submodular *if it satisfies one of the following equivalent criteria:*

1. $f(\mathcal{T}) + f(\mathcal{U}) \geq f(\mathcal{T} \cap \mathcal{U}) + f(\mathcal{T} \cup \mathcal{U})$ *for all* $\mathcal{T}, \mathcal{U} \subseteq \mathcal{S}$.
2. $f(\mathcal{T} \cup \{s\}) + f(\mathcal{T} \cup \{t\}) \geq f(\mathcal{T}) + f(\mathcal{T} \cup \{s,t\})$ *for all* $\mathcal{T} \subseteq \mathcal{S}$ *and* $s, t \in \mathcal{S} \setminus \mathcal{T}$ *with* $s \neq t$.
3. $f(\mathcal{T} \cup \{s\}) - f(\mathcal{T}) \geq f(\mathcal{U} \cup \{s\}) - f(\mathcal{U})$ *for all* $\mathcal{T}, \mathcal{U} \subseteq \mathcal{S}$ *and* $s \in \mathcal{S} \setminus \mathcal{U}$ *with* $\mathcal{T} \subseteq \mathcal{U}$.

To gain some intuition of the meaning of submodularity, consider the third criteria in Definition 4.1. It essentially states that for a submodular set function, the gain of adding a new element to a set is less than the gain of adding this element to one of its subsets.

Thus, in the case of the EC² score function, when considering marginal benefits between $f_{EC}(\mathcal{A}_i)$ and $f_{EC}(\mathcal{A}_{i+1})$, the submodularity property implies $\text{score}_{\text{EC}^2}(\mathbf{x} \,|\mathcal{S}_i) \geq \text{score}_{\text{EC}^2}(\mathbf{x} \,|\mathcal{S}_{i+1})$ for a fixed data point $\mathbf{x}$, which by straightforward induction yields

$$\text{score}_{\text{EC}^2}(\mathbf{x} \,|\mathcal{S}_i) \geq \text{score}_{\text{EC}^2}\left(\mathbf{x} \,\middle|\mathcal{S}_j\right) \text{ if } i \leq j$$

for any data vector $\mathbf{x}$.

We can exploit this property in order to gain significant speed increases by not having to reevaluate the score for each data point in every iteration. Let our algorithm keep a sorted list of the scores together with their corresponding data points of the past iteration. In the current iteration, we then reevaluate the score function only for the highest scoring data point. If the resulting score is still the highest, we can safely select this data point and know that none other could have resulted in a higher score. Otherwise, we move this data point and insert it in the list according to its new score and reiterate the procedure with the second highest scoring data point from the past iteration.

While in the worst case we might still have to compute the score for each data point in every iteration, in practice often a few evaluations per iteration suffice. This results in significant time savings, especially for big datasets.

#### Choosing Equivalence Classes

The EC² algorithm builds upon a partitioning of the set of all hypotheses $\mathcal{H}$ into disjunct hypothesis classes $\mathcal{H}_i$ for $1 \leq i \leq k$. We therefore need to think about how to translate the particles with corresponding weights into hypothesis classes that we can use for the EC² criterion.

First, let us deal with the particle weights. We opt for a simple approach by reusing the systematic resampling step as described in Algorithm 4.1, which

returns a set of particles with equal weights. We use these particles as the hypotheses among which the EC² algorithm distinguishes.

Second, we are now faced with the problem of partitioning this set of particles into equivalence classes. Intuitively, an equivalence class describes a set of particles that are similar but exhibit small fluctuations due to noise. The simplest of options is to use one equivalence class for every unique[16] particle. Alternatively, we can first cluster particles in their space and then use these clusters as equivalence classes. To achieve this, we use a hierarchical clustering approach based on squared euclidean distances. Even though this choice makes us less dependent on on a fixed parameter denoting the number of clusters than for example the $k$-means algorithm[17] would, we still need to specify a cut-off parameter $k_{\max}$ that defines the maximal number of clusters. We choose $k_{\max} := \sqrt{\frac{m}{2}}$, a simple heuristic proposed by Mardia et al. ([25]).

When running the active learning algorithm, we propose two different ways to implement the discussed steps, which translate the particles and particle weights into EC² conforming hypothesis classes. Either the conversion happens once at the very beginning, and the algorithm from then on keeps track separately of the OASIS and the EC² model, using both for their respective purposes. Or the algorithm translates the current particles into hypothesis classes in every step to calculate the EC² scores, and thus stays within the overall OASIS framework.

The first approach has the benefit of providing a "purer" way to calculate the EC² scores. In particular, we can take full advantage of the EC² submodularity property, which will speed up calculations considerably. However, in essence, we are working with two different model representations of the current situation and have no guarantee that these two models will still correspond to each other after some iterations.

The second version does not experience this model drift and is in some sense more integrated. However, we essentially change the EC² equivalence classes in every iteration, which are fundamental to the design of the algorithm. Therefore, we loose the guarantees that EC² provides – in particular the submodularity property – and can only hope for an approximation at best.

However, even in this approximative setting, we can still try to exploit the submodularity property. To make up for the recreation of EC² hypotheses in every step by resampling, we downweight the scores of the previous iteration by a certain factor $f_{\mathrm{dw}}$. We choose this factor as the ratio of the number of particles from the previous iteration that were coherent with the actual observation made.

[16] Recall that the systematic resampling step most likely returns duplicate particles such that $k \leq m$.

[17] For a description of the $k$-means algorithm refer to e.g. Bishop ([2, pp. 424–430]).

| Name | Fast | Clustering | Pure |
|---|---|---|---|
| $EC^2$ | No | No | No |
| $EC^2_f$ | Yes | No | No |
| $EC^2_c$ | No | Yes | No |
| $EC^2_{f,c}$ | Yes | Yes | No |
| $EC^2_p$ | No | No | Yes |
| $EC^2_{f,p}$ | Yes | No | Yes |
| $EC^2_{c,p}$ | No | Yes | Yes |
| $EC^2_{f,c,p}$ | Yes | Yes | Yes |

**Table 4.2:** Family of $EC^2$ score functions. *Fast* determines if the submodularity property is exploited or not, a yes for *clustering* denotes that hierarchical clustering is used to choose the $EC^2$ equivalence classes, and *pure* variations translate the OASIS model to $EC^2$ hypothesis classes only once instead of in every iteration.

#### Family of $EC^2$ Score Functions

All the discussed issues with the $EC^2$ score function lead us to define a family of $EC^2$ score functions depending on which of the various resolutions we choose. Table 4.2 gives an overview of the resulting eight different variations.

### 4.2.2 Example

Let us look at an example that shows the offline learning algorithm in operation on the synthetic Diced dataset also used in the example of Section 4.1.6. We adopt as well the other choices from that example, in particular the OASIS score function and a normalization of all particle vectors for ease of presentation. Again, note that with respects to particle weights, we are mainly interested in qualitative differences, and thus do not quantify them in the graphs.

As a first step, we run the OASIS algorithm without its active component, i.e. without requesting any labels, on the whole dataset. The resulting particle filter representation of the probability distribution over the hyperplane vectors, depicted in Figure 4.8, has four distinct modes that essentially correspond to the four principal ways that a hyperplane can separate the two-dimensional Diced dataset – two vertical hyperplanes distinguished by opposite directions of their defining normal vectors and likewise two horizontal hyperplanes.

The algorithm iteratively scores all data points and queries the label of the most informative one, which it subsequently uses to update all particle weights. Figure 4.9 shows the first three iterations of this offline algorithm. As can be seen, after only three data points, two of the modes dominate the others, and the current decision boundary very coarsely approximates the true one.

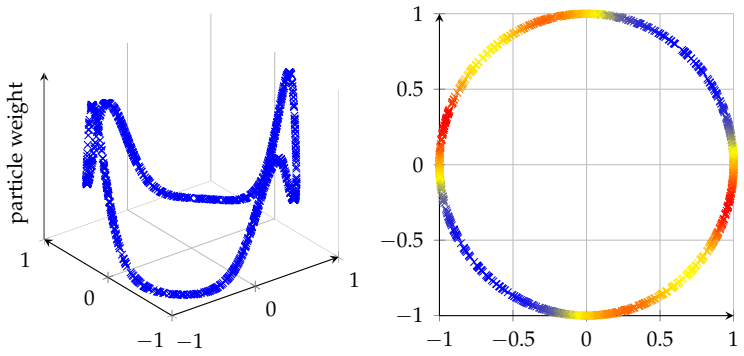


**Figure 4.8:** Particle representation of the probability distribution over the hyperplanes after running the OASIS algorithm without any label requests on an instance of the DICED dataset. On the left, the particle weights are drawn in a third dimension, while on the right, big weights have a red coloring. In both cases, four main modes, which correspond to the possible separating hyperplanes in feature space, are clearly visible.

After 10 iterations – a situation depicted in Figure 4.10 – the picture looks similar, but now two modes have almost completely vanished. As the algorithm progresses, it is able to approximate the true decision boundary better and better. Figure 4.11 shows the model after 25 iterations, where the correct solution clearly dominates and the right decision boundary has almost perfectly been found.[18]

## 4.3 Experimental Results

One of the goals of this thesis is to compare and evaluate different score functions on real datasets. Building on the theory introduced so far, we present experimental results of the OASIS as well as the modified offline active learning algorithm. Each value reported is the mean over a number of random trials, with error bars indicating the double standard error. For more information on the datasets used, consult Appendix A.

### 4.3.1 OASIS Algorithm

We conducted various experiments to compare the different score functions in the OASIS setting. Every result shown was obtained by 50 random trials over a dataset by randomly permuting the number in which the OASIS algorithm sees the data points. Except for setting $m = 10000$ as the number of particles, we chose all parameters as proposed by Goldberg et al. ([14]).

[18] If we are interested in a single best hypothesis instead of the whole model, we choose the hyperplane with the biggest weight.

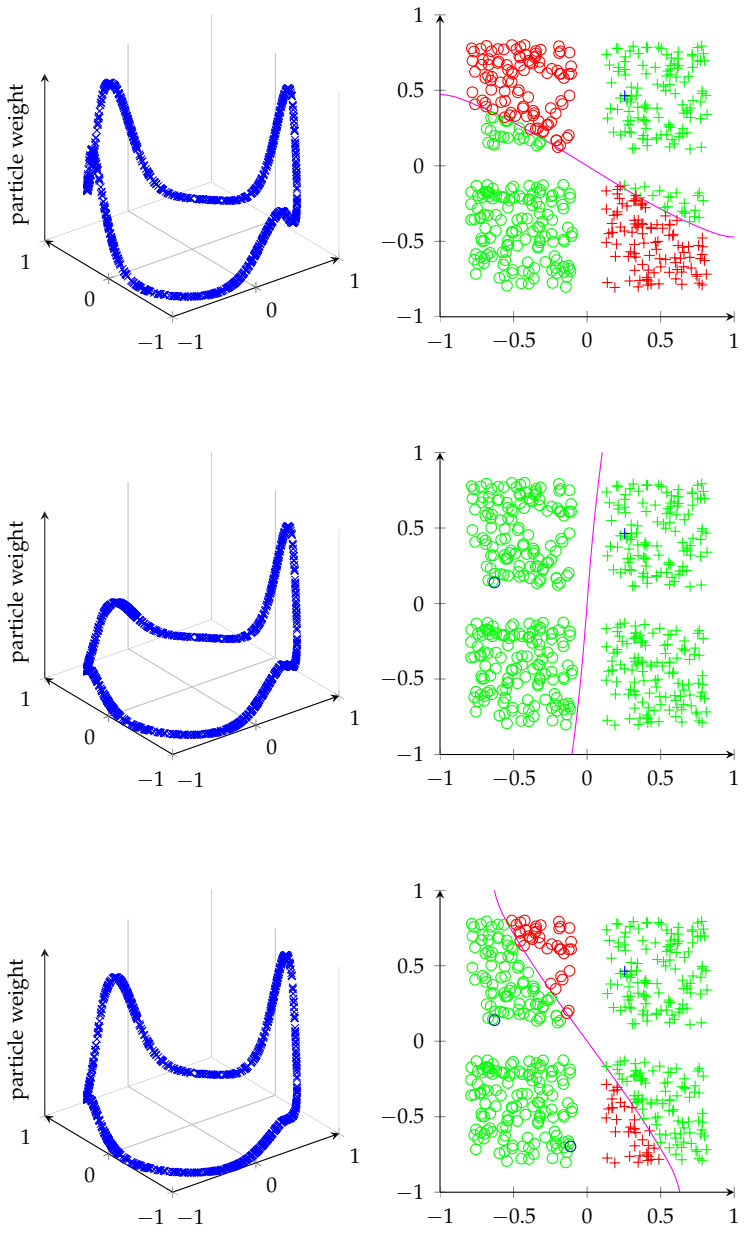


**Figure 4.9:** The first three iterations of the offline learning algorithm on the Diced dataset. On the right, the current decision boundary is indicated by a purple line, and the data points for which labels have been requested are shown in blue. On the left, we can see that after making three observations, two modes of the initial particle distribution have almost vanished.

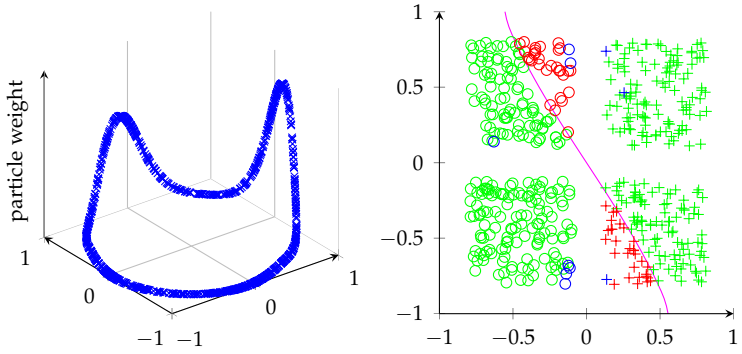


**Figure 4.10:** At iteration 10, only two possible modes in the distribution over all hyperplanes remain. Although the model has been refined, the resulting decision boundary has not changed much compared to iteration three.

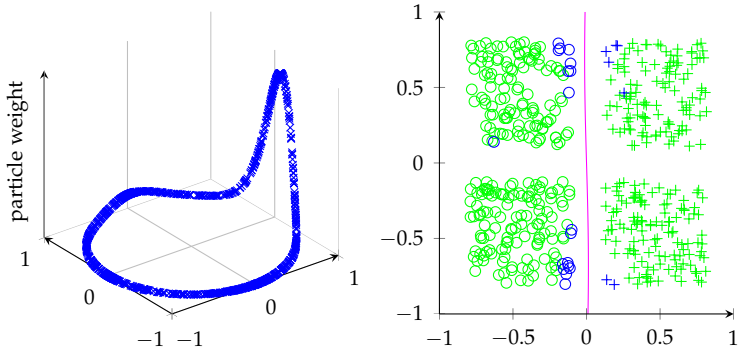


**Figure 4.11:** After making 25 observations, the offline active learning algorithm has found the correct hyperplane. This is evident from the probability distribution over all hyperplanes on the left as well as the resulting decision boundary on the right.

As we expect the threshold value $s_0$ to be one of the most influential parameters on the final outcome, we decided to test the performance of each score function with seven different threshold values as shown in Table 4.3. To arrive at these threshold values, we started with the two thresholds given for the OASIS score function by Goldberg et al.[19] and chose the other threshold values around them. For the remaining score functions, we tried to find two initial values that resulted in similar results to the OASIS score function, and again chose the other five threshold values around these points.

To keep things simple and transparent, we first evaluated the influence of the score threshold with only the OASIS score function on the Letter dataset. Figure 4.12 shows the results for four of the seven score thresholds,

[19]They use $s_0 = 0.01$ and $s_0 = 0.1$.

| Score function | | |
|---|---|---|
| OASIS | GBS | UC |
| 0.001 | −0.49 | 0.001 |
| 0.005 | −0.45 | 0.005 |
| 0.01 | −0.4 | 0.01 |
| 0.05 | −0.35 | 0.02 |
| 0.1 | −0.3 | 0.05 |
| 0.2 | −0.25 | 0.1 |
| 0.5 | −0.2 | 0.25 |

**Table 4.3:** Threshold values used for the different score functions.

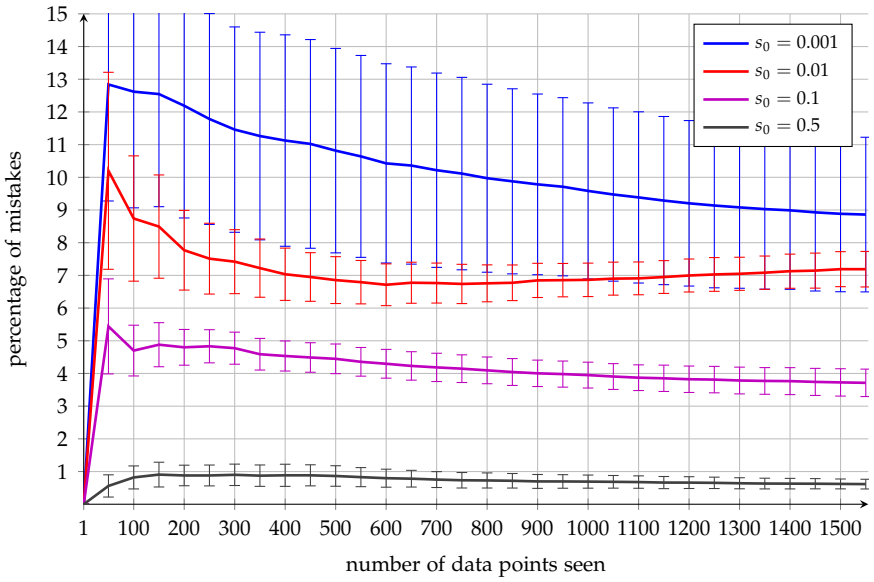


**Figure 4.12:** Results of running the OASIS algorithm with its original score function on the Letter dataset while choosing four different values for the threshold parameter $s_0$. The percentage of mistakes reported is computed relative to the data points the algorithm has seen so far. For $s_0 = 0.001$, the results fluctuate heavily, because almost no additional labels are queried, and therefore the performance of the model largely depends on the random placement of the first two labeled data points.

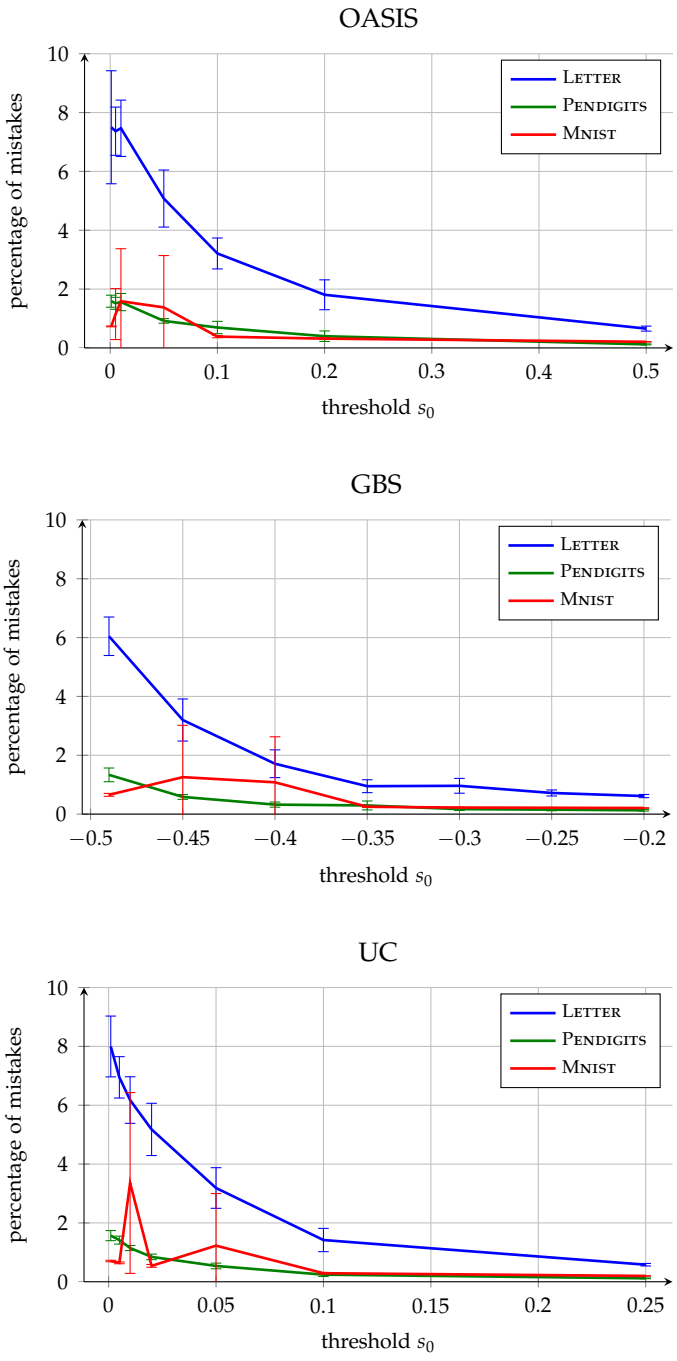


**Figure 4.13:** Overview of the influence of the threshold value on the percentage of mistakes for different score functions and datasets. All score functions are heavily dependent on the choice of $s_0$, and while there are some small individual fluctuations, the overall results look similar.

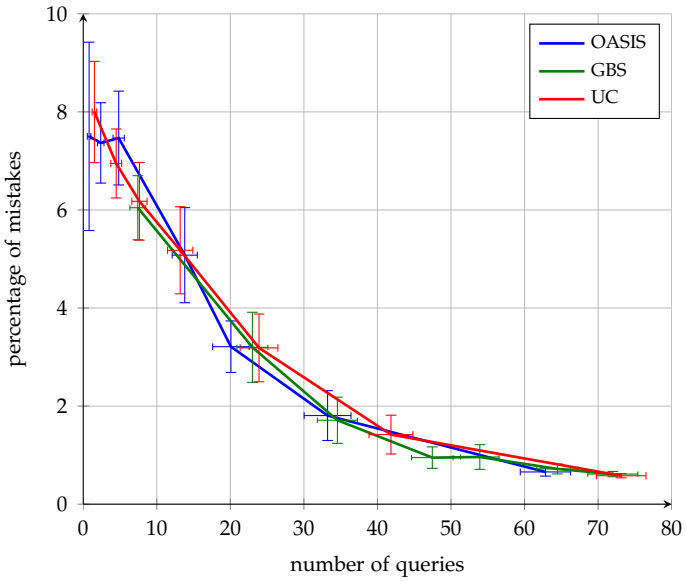


**Figure 4.14:** Percentage of mistakes the OASIS algorithm makes depending on the number of queries for different score functions on the LETTER dataset. There are no significant differences apparent between the three score functions.

and strongly suggests that the score threshold parameter influences the performance of the algorithm in a major way. As expected, if higher values are chosen for the threshold $s_0$, less mistakes are made, because the algorithm will also query more labels of data points and have more information available.

Furthermore, as common with machine learning techniques, this plot shows that most of the learning time is done in the beginning, i.e. when the algorithm receives the first few data points.[20]

Figure 4.13 depicts the influence different thresholds have on the final percentage of mistakes – i.e. the error rate of the algorithm after going through all data points once – for the score functions we discussed, evaluated on three datasets. While from this perspective we cannot say anything about how quickly the error rate reaches or comes close to the final percentage of mistakes, we remark that the curves look very similar for all score functions, i.e. we can achieve similar error rates with every score function by choosing the right score thresholds.

However, in active learning it is not only the performance in terms of the mistakes an algorithm makes that interests us. While we are not neglecting

[20]Note that the sequence of data points the OASIS algorithm receives has a high importance attached to it; data points received in the beginning influence the model more than data samples that arrive after a lot of data has already been seen.

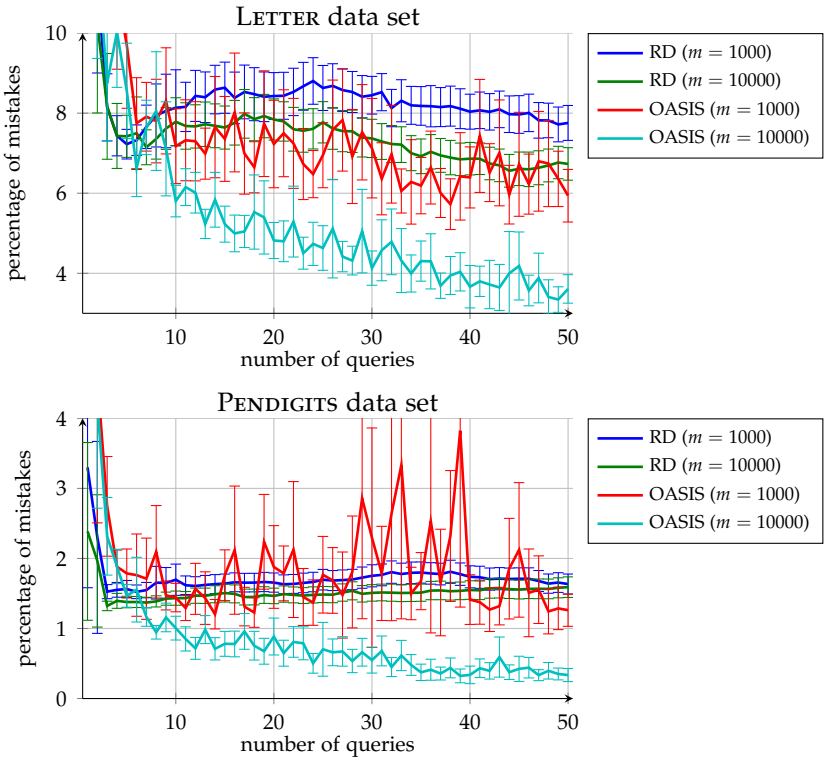


**Figure 4.15:** The percentage of mistakes made depending on the number of data points queried for 1000 versus 10000 particles. Both a random selection as well as the OASIS score function exhibit significant improvements on the Letter and Pendigits datasets when using 10000 particles. Note that for visibility, the scaling of the vertical axis differs between the upper and lower plots.

this aspect, we are primarily interested in how many label requests an algorithm makes. Figure 4.14 shows how the number of labels that the OASIS algorithm queries influences the mistakes made.

From our observations so far, we conclude that while some small differences are noticeable, overall the choice of the score function alone does not seem to be the deciding factor for good or bad performance in the OASIS setting.

### 4.3.2 Offline Learning

We now turn to offline active learning, which deviates from the online setting in which the OASIS algorithm operates, with the offline algorithm having access to all data samples without their labels, iteratively picking the next best data point and requesting its label. This allows us to better investigate the influence of single parameters, because the performance is not dependent on the sequence in which data points arrive. In addition, we can easily

| | Likelihood form | | |
|---|---|---|---|
| Parameter | normal | steep | flat |
| $\alpha$ | 0.5 | 0.1 | 0.7 |
| $\gamma$ | 0.2 | 0.1 | 0.3 |

**Table 4.4:** Three different sets of arguments for the parametrized likelihood function $p_{\text{lhd}}(y \,|\mathbf{x}, \mathbf{w})$ defined in Section 4.1. The *normal* version is suggested by Goldberg et al. while the *steep* and *flat* variations make the function more respectively less steep.

compare our score functions with a random selection of data points in order to investigate their merit over a simple baseline approach. All results are reported over 100 independent trials, and we stopped after selecting the first 50 data points.[21]

The first parameter of interest is the number of particles $m$. Goldberg et al. propose the use of 1000 particles for the OASIS algorithm, but as Figure 4.15 shows for the OASIS score function and on two datasets, more particles can lead to significantly better performance, at least in the offline setting. Also, note that using 10000 particles widens the gap between the performance of the OASIS and random score functions compared to a particle filter model with 1000 particles. Therefore, for all other experiments, we opt to use 10000 particles.

A second suggestion by Goldberg et al. is the particular form of the likelihood function $p_{\text{lhd}}(y \,|\mathbf{x}, \mathbf{w})$ defined in Section 4.1. Apart from the parameter values they suggest, we look at two additional versions, one with a flatter and one with a steeper curve. Table 4.4 shows the exact values for the two likelihood function parameters $\alpha$ and $\gamma$ that we used.

Comparing the performance in terms of the percentage of mistakes of these likelihood functions, some differences are visible, even though they are not as striking as the number of particles. In general, however, the *normal* and *flat* likelihood functions, which inhibit weaker assumptions, seem to perform better, as Figure 4.16 shows for the Letter dataset. Nevertheless, we decided to use the suggested likelihood form for the rest of the experiments.[22]

Let us now turn to the question of how different active learning score functions compare to each other. In Figure 4.17, we plot the performance of the various score functions discussed in terms of the percentage of mistakes they

[21] Note that when reporting the percentage of mistakes, we always use the complete dataset to compute the result.

[22] Not choosing the flat function form does not influence the conclusions we draw from our observations. We are primarily interested in how different score functions perform, which is not affected by the likelihood function form – the differences we observe are consistently similar over the score functions for all likelihood function variations.

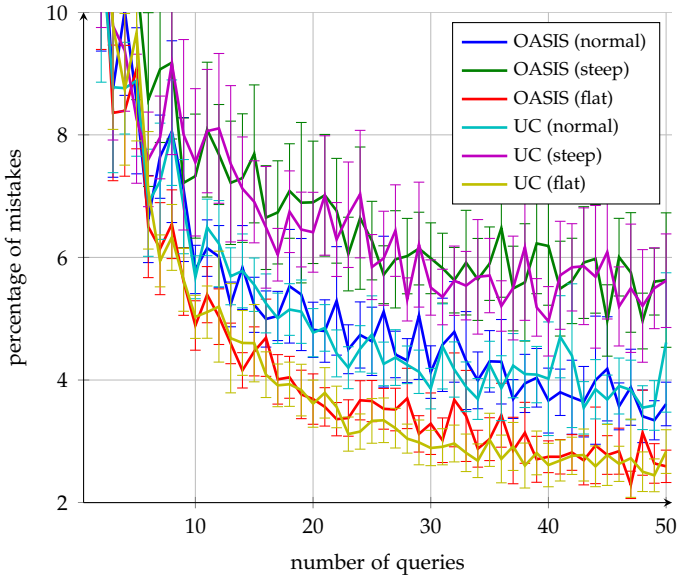


**Figure 4.16:** Influence of different likelihood functions on the performance, measured on the LETTER dataset. We can see that the functions which are less steep, i.e. make weaker assumptions, perform better.

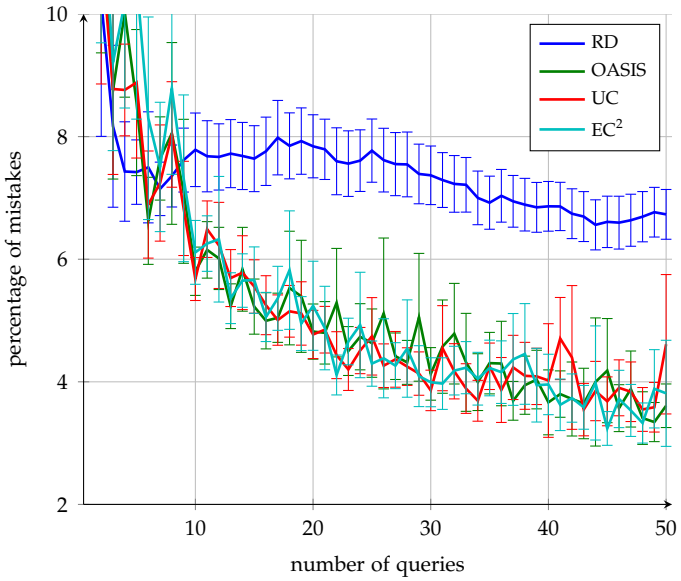


**Figure 4.17:** Comparison in terms of the percentage of mistakes of different score functions on the LETTER dataset. All informative score functions achieve results that are similar to each other and significantly better than simple random selection. The fluctuations are due to outliers that influence the mean, using the median instead would smooth all curves.

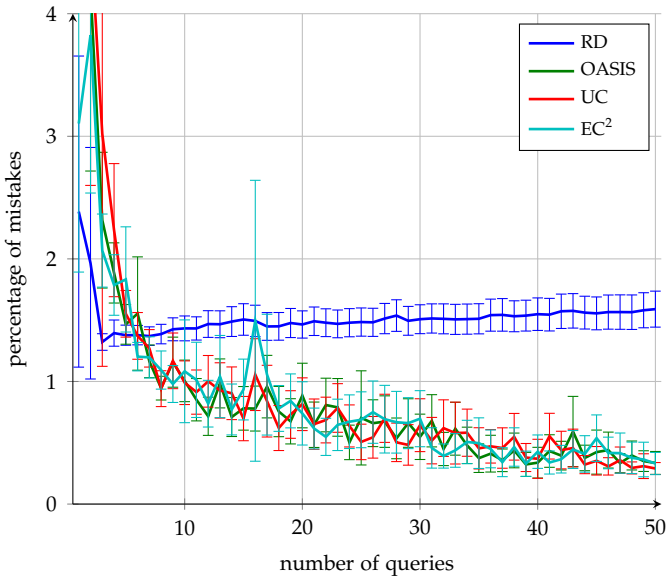


**Figure 4.18:** Performance of various score functions on the Pendigits dataset depending on the number of labels queried. The results are similar to the the ones observed on the Letter dataset. Again note that the sometimes heavy fluctuations are due to individual outliers.

make on the whole dataset as a function of how many data points labels were queried. We make several observations:

1. Except for random selection, all score functions exhibit considerable fluctuations. The reason for this lies in outliers that shift the mean significantly. If we were to draw the medians instead, the curves would become smooth.

2. The three informative score functions perform significantly better than random. However, they also exhibit equal performances and we cannot distinguish between the OASIS, uncertainty, and EC² score functions.

Figures 4.18 and 4.19 show similar results on the Pendigits and Wdbc datasets. This increases our confidence that the observations made have general validity and are not just a special case for the Letter dataset.

### $EC^2$ Score Function Family

We have seen that the $EC^2$ score function performs similar to the OASIS and uncertainty scores. Figure 4.20 compares other score functions from the EC²-family to the standard $EC^2$ version, again on the Letter dataset. Note that in terms of the percentage of mistakes made, because of the submodularity property, we know that $EC^2_p$ returns exactly the same results as $EC^2_{f,p}$ and

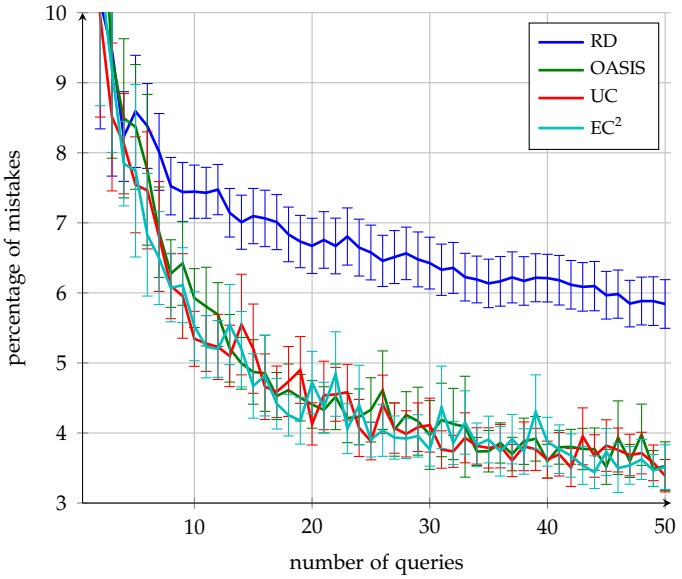


**Figure 4.19:** Percentage of mistakes depending on the number of queries for the WDBC dataset and several score functions. These results are similar in spirit to the observations made on the PENDIGITS and LETTER datasets.

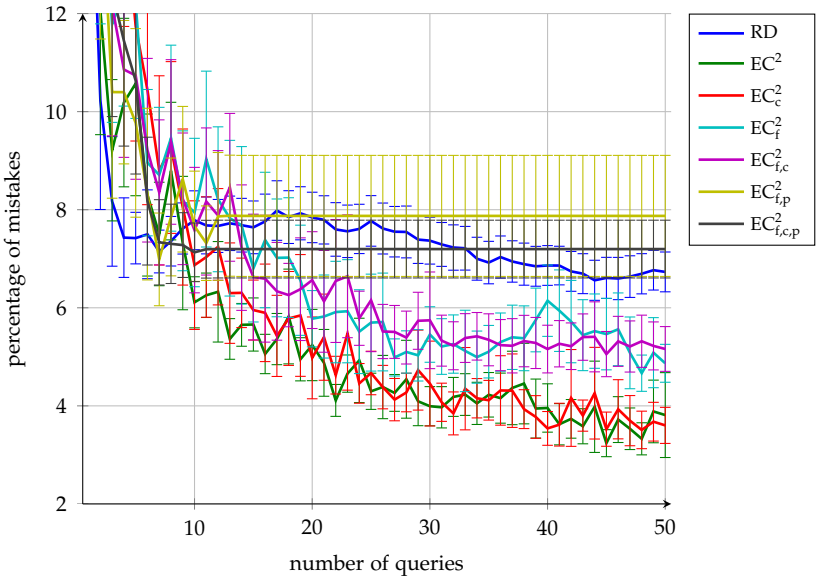


**Figure 4.20:** A comparison of the $EC^2$ family of score functions on the LETTER dataset. The pure versions $EC^2_{f,p}$ and $EC^2_{f,c,p}$ soon hit a plateau due to running out of particles that correspond to the observations made, resulting in poor overall performance. We can also see that our clustering approach generally does not yield superior performance.

$EC^2_{c,p}$ the same as $EC^2_{f,c,p}$. Therefore, these two score functions are not shown in the graph.

From the results, we make the following observations:

1. While the pure EC² versions $EC^2_{f,p}$ and $EC^2_{f,c,p}$ start similar to all other score functions, they soon hit a plateau, because none of the initial sampled particles correspond to the observations anymore, i.e. the EC² model deviates too much from the OASIS model. Hence, these score functions generally perform worse than simple random selection, even though the version with clustering $EC^2_{f,c,p}$ can keep up a little bit longer.
2. All other score functions perform better than random.
3. For the non-pure versions of the EC² family, clustering does not seem to decrease the percentage of mistakes.
4. The fast versions $EC^2_f$ and $EC^2_{f,c}$ make slightly more mistakes than the $EC^2$ and $EC^2_c$ score functions.

#### Runtime

There is a second important factor when choosing a score function, which we have not investigated so far: The speed of the algorithm. While the runtime of a program depends on many factors, we nonetheless tried to make a practical comparison.[23] Figure 4.21 shows the time in seconds that the various algorithms, using different score functions, from the EC² family need.

Evidently, the pure EC² score functions exhibit a constant time behavior as soon as they reach the plateaus we observed earlier. On the other hand, most other score functions result in a steep linear function, with the exception of the fast $EC^2_f$ version. Comparing these runtimes with the mistakes made from Figure 4.20, We can see that there is a trade-off between computational speed and accuracy.[24]

Figure 4.22 adds the non-EC² score functions to the comparison. Whereas a random selection will always be fastest, as no additional computations are required, the OASIS and uncertainty score functions exhibit a linear increase in runtime, with the OASIS algorithm being significantly faster. The only EC² score function that can keep up – apart from the inaccurate pure versions – is $EC^2_f$, which shows a sublinear behavior. Therefore, we can expect it to be faster than the other score functions as more and more labels are queried.

[23] For implementation details, please refer to Appendix B.

[24] In particular, when using the fast $EC^2_f$ score function, the details of this trade-off depend on how the parameter $f_{dw} \in [0;1]$, which determines how much scores are downweighted, is chosen. If $f_{dw} = 0$ then the $EC^2_f$ and $EC^2$ score functions coincide, otherwise the higher its value, the faster and more inaccurate the algorithm becomes.

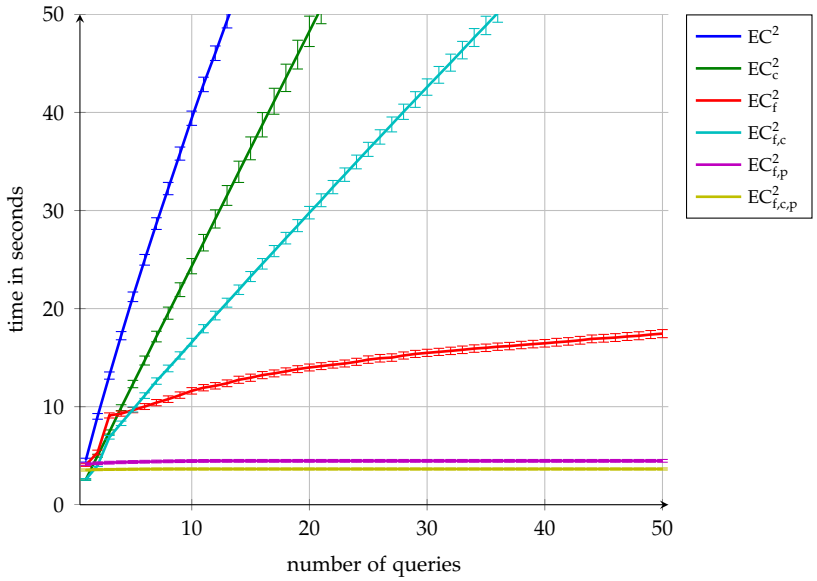


**Figure 4.21:** Runtimes of score functions from the $EC^2$ family, measured on the LETTER dataset. When keeping the percentage of mistakes made – shown in Figure 4.20 – in mind, we can see that the $EC^2_f$ score function provides the best trade-off.

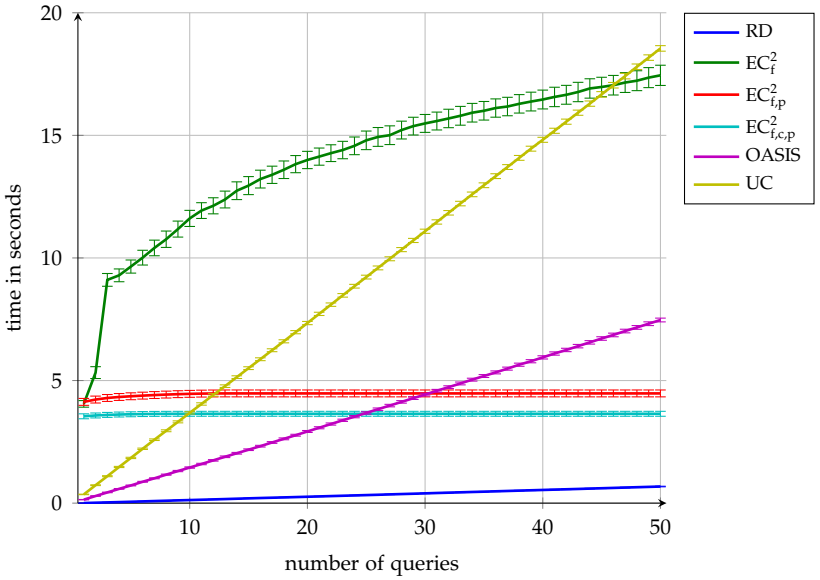


**Figure 4.22:** Comparison in terms of the runtime of different score functions depending on the number of labels queried on the LETTER dataset. The $EC^2$ score functions shown here have a sublinear curve and can be expected to be faster than the other functions when many labels are queried. However, if pure speed is the goal, random selection is the fastest.

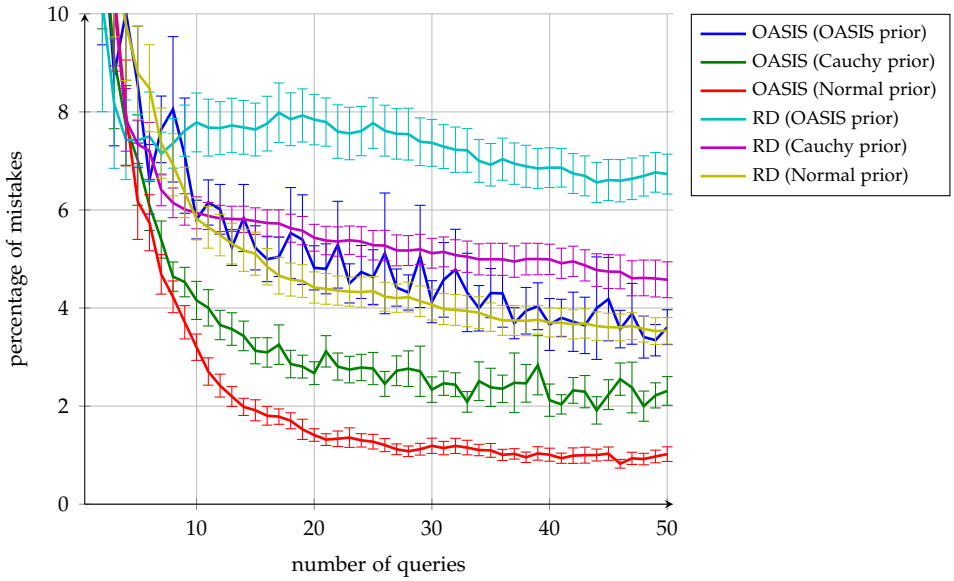


**Figure 4.23:** Results of comparing the offline active learning scenario with an OASIS-based prior to simple Cauchy and Normal priors, here shown on the Letter dataset and depending on the number of queried labels. Surprisingly, the informative OASIS prior performs consistently worse than the Cauchy and especially Normal prior.

### Alternative Prior Distributions

Lastly, we ask what the effect of the OASIS algorithm itself is in our setting. Recall that before using the active learning approach, we once run the OASIS algorithm without the active learning part in the hope that we receive a good representation of the distribution over the space of all hyperplane vectors. Alternatively, we can sample the particles from a simple prior distribution such as the Cauchy[25] or Normal distribution. Figure 4.23 shows a selection of the surprising results, here on the Letter dataset and comparing the OASIS score function with simple random selection. Apparently, the OASIS prior performs significantly worse than the Cauchy and Normal priors, which leads us to question its applicability together in the classical active learning setting that we described.

## 4.4 Discussion

This chapter investigated the OASIS algorithm by Goldberg et al. ([14]) and several active learning score functions in two different ways.

[25]We follow the use of the OASIS algorithm for the Cauchy prior, i.e. by independently sampling in every dimension.

First, we looked at the OASIS algorithm as it has been proposed, and evaluated it using several score functions on various datasets. The performance differences observed due to the different score functions, however, are almost nonexistent. Therefore, all the other factors of the integrated OASIS framework – like its online and semi-supervised aspects, or specific parameter values – seem to have a higher impact. In essence, this means that the active part of the algorithm only plays a relatively small role. Also, note that the algorithms' active selection procedure has not been highly developed, to which the simple score threshold determining whether a label will be actively queried or not testifies.

There is an additional potential issue with the OASIS algorithm that we have not mentioned so far and that has not been addressed yet in the literature. In its current form, the OASIS algorithm assumes that the dataset is centered around the origin. Moreover, it requires that the separating decision hyperplane runs through the origin as well. Although Goldberg et al. mention that in practice, a dummy feature could be added to the dataset to model a bias term – a common method for linear classifiers[26] –, we question the validity of this approach.

In particular, consider how the semi-supervised approach deals with a hyperplane that is far away from all given data points, i.e. which has all data points lying on one side outside its margin. If no label is given, the learning algorithm will always give a relatively big weight to this hyperplane compared to a hyperplane that lies close to data points as a consequence of the gap assumption. However, if the dataset is made up of two equal-sized classes, we know that such a hyperplane will always classify half the data points incorrectly, and should be given a very low weight. If the dataset is of such a nature that its data points are approximately evenly arranged around the origin, as in the example datasets provided by Goldberg et al., this critique does not directly apply. However, in general, one should make sure that this condition is fulfilled when using the OASIS algorithm.

Second, in this chapter's investigation, we deviated from the online setting and used a modified version of the OASIS algorithm to construct a distribution over the space of all hyperplanes. Subsequently, we applied a standard active learning method to iteratively choose the best data points and refine the distributional model. Comparing different options for single parameters showed that many of them influence the end result significantly, in particular the form of the likelihood function and the number of particles used. We were also able to compare informative active learning score functions with a simple random selection procedure, which showed that often care has to be taken to see an actual increase in performance.[27]

[26] See for example Bishop ([2, p. 138]).

[27] For example, we noticed that the number of particles suggested by Goldberg et al. does

We further showed how the EC² score function introduced by Golovin et al. ([15]) can be applied in this setting based on the generic noise model described in Section 3.3. Because of the particle filter approximation, however, the EC² criterion cannot be computed exactly and consistently over a sequence of iterations, so that we looked at different options of computing it approximately. This caused a whole family of EC²-based score functions, many of which unfortunately turned out to be impractical to use in terms of speed or percentage of mistakes.

Some approaches based on EC² did show potential, in particular the score function we termed $EC^2_f$, which can be tuned by an additional parameter $f_{dw}$ that gives a trade-off between accurateness and speed. We could also imagine that other ways of computing the EC² score might perform better, for example a mixture that in principle uses the approach of the pure EC² functions, but resamples the particles like the non-pure versions do when a plateau, as we described it, is reached.

Lastly, however, we provided experimental evidence that questions a combination of the OASIS algorithm with a classical active learning setting. This leads us to conclude that the OASIS algorithm may be a valuable approach in certain situations, which directly correspond to its unique combinatorial framework, i.e. when an integrated solution to deal concurrently with online, active, and semi-supervised learning is needed. But in other cases, a thorough examination should come first to investigate whether the OASIS algorithm with its many parameters and potential complications can actually deliver superior performance.

not suffice for some datasets.

Chapter 5

# Active Selection of Features

In Chapter 4, we presented learning methods that actively request labels for data points and seek to identify a separating hyperplane. Taking on a different perspective, we can also ask which features are the most informative ones, so that if we want to classify an unlabeled data point, we only have to look at a subset of its attributes. This approach corresponds to learning a decision tree[1] for classification with the goal to develop short trees, i.e. to inspect as few features as possible.

We can formalize this setting by using the noise model defined in Section 3.3. Let us assume that we are given a dataset $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} = \{\mathbf{x}_1^\intercal, \ldots, \mathbf{x}_d^\intercal\}$ with $n$ data points in $d$ dimensions, such that $\mathbf{x}_i \subseteq \mathbb{R}^d$ for all $i$. In addition, for each data point $\mathbf{x}_i$, we are given the corresponding label $y_i = \text{label}(\mathbf{x}_i) \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of all possible classes and $|\mathcal{Y}| = k$, i.e. there are $k$ distinct classes to which a data point can belong. We assume that the data points we see, together with their labels, are drawn from some distribution in an independent and identically distributed way.

The set of hypotheses $\mathcal{H}$ then corresponds with the classes a data point can belong to, and the noise variable $\Theta$ models the data points $\mathbf{x}_i$. Note that here we look at the data points we see as noisy instances of the "pure" classes. Furthermore, there are $d$ feature dimensions that are mapped to the set of tests $\mathcal{T} = \{t_1, \ldots, t_d\}$, and performing a test results in one of $\ell$ distinct outcomes. Without loss of generality, for the rest of this chapter we assume that $\ell = 2$ and our set of outcomes is $\mathcal{Z} = \{0, 1\}$, which we interpret as a Boolean result.

Therefore, we construct a binary decision tree where each node within the tree has exactly two children, and leaf nodes are responsible for classification.

[1] We assume that the reader is familiar with the basic concepts of decision trees. For an informative introduction to decision trees, see Mitchell ([29]).

A test in this setting can be viewed as a logical predicate operating on some feature that returns either true or false for every data point.

How exactly we map a single data point feature to one or more tests is a question worthy of consideration. If all possible values of a certain feature are part of a finite set of cardinality $c$, a simple solution is to generate $c$ tests that each indicate if the feature is equal to one of the possible values or not. If features can take on infinitely many values, for example a number from a rational interval, we cannot apply the same strategy anymore. However, we can introduce arbitrarily many one-sided checks to determine whether the feature value is greater or smaller than a certain threshold.[2]

## 5.1 Score Functions

Similar to the setting where we actively query the labels of data points, we have different options on what score function to use. Table 5.1 provides an overview of all the score functions that we consider for the active selection of features.

We interpret the value that a score function calculates for a specific feature $t \in \mathcal{T}$ as a measure of informativeness, where in general higher values translate into a more informative feature.[3] Recall that the set $\mathcal{S}_i$ contains the first $i$ features that were selected along with their observed outcomes, and we write $\mathcal{X}_i := \{\mathbf{x} \in \mathcal{X} : \forall (t, z_t) \in \mathcal{S}_i, \mathbf{x}_{[t]} = z_t\}$ for the data points that agree with them.

### 5.1.1 Random Score

The simplest way to assign a score is, again, to draw a random number from the interval $[0; 1]$. We denote this score function $\text{score}_{\text{RD}}(\cdot)$.

### 5.1.2 Uncertainty Score

An intuitive criterion is to select tests about whose outcomes we are most unsure. In our case, one way to achieve this is to rate features highly that split as evenly as possible between the two outcomes, which yields

$$\text{score}_{\text{UC}}(t \,|\mathcal{S}_i) := \min_{z \in \mathcal{Z}} \frac{|\{\mathbf{x} \in \mathcal{X}_i : \mathbf{x}_{[t]} = z\}|}{|\mathcal{X}_i|}$$

where $\mathcal{Z} = \{0, 1\}$. Note that when selecting features with the uncertainty score, lower values are considered more informative.

[2] Mitchell ([29]) provides a more in-depth discussion of this topic.

[3] The exception is the uncertainty score, for which lower values are considered to be better.

| Score Function Name | Abbreviation |
|---:|:---:|
| Random Score | RD |
| Uncertainty Score | UC |
| Information Gain Score | IG |
| EC² Score | EC² |

**Table 5.1:** Overview of the score functions used for the active selection of features along with their corresponding abbreviations.

### 5.1.3 Information Gain Score

One shortcoming of the uncertainty score as we defined it is that it ignores the class label information. A common way to take the distribution over classes into account is to compute a score that measures the information gain in terms of the Kullback-Leibler divergence[4] $D_{KL}$. In our case, this results in the score function

$$\text{score}_{\text{IG}}(t \,|\mathcal{S}_i) := \mathbf{E}_{z_t \sim Z_t|\mathcal{S}_i}\big[D_{KL}\big(\text{p}_{\text{emp}}(\mathcal{Y} \,|\mathcal{S}_i) \parallel \text{p}_{\text{emp}}(\mathcal{Y} \,|\mathcal{S}_i \cup \{(t, z_t)\})\big)\big]$$

where $\text{p}_{\text{emp}}(\mathcal{Y} \,|\mathcal{S}_i)$ is the empirical distribution over the class labels given the observations $\mathcal{S}_i$.

### 5.1.4 EC² Score

We again want to compare the EC² criterion with existing methods. Based on our earlier definition in Section 4.2.1, the EC² score function becomes

$$\text{score}_{\text{EC}^2}(t \,|\mathcal{S}_i) := \mathbf{E}_H[f_{EC^2}(\mathcal{A}_i \cup \{t\}, H) - f_{EC^2}(\mathcal{A}_i, H) \mid \mathcal{S}_i]$$

for the active selection of features. Recall that

$$f_{EC^2}(\mathcal{A}, h) := f_{\text{w}}\left(\bigcup_{\mathbf{x} \in \mathcal{A}} \mathcal{E}(\mathbf{x}, h)\right)$$

is a (adaptively) submodular function and

$$f_{\text{w}}(\mathcal{E}) := \sum_{\{h,h'\} \in \mathcal{E}} \text{p}(h)\,\text{p}(h')$$

is a weight function based on the probability of hypothesis. We follow Golovin et al. ([15]) in using uniform probabilities $\text{p}(h) := 1/|\mathcal{H}|$ for every hypothesis $h$.

In the current setting, we choose the partitioning of the set of hypotheses $\mathcal{H}$ into equivalence classes such that each equivalence class $\mathcal{H}_i$ corresponds to one of the $k = |\mathcal{Y}|$ distinct classes, amongst which we want to distinguish.

[4]The Kullback-Leibler divergence is also known as the relative entropy and defined as $D_{KL}(\text{p} \parallel \text{q}) := \sum_i \text{p}(i) \log(\text{p}(i)/\text{q}(i))$, where p and q are probability distributions of a discrete random variable.

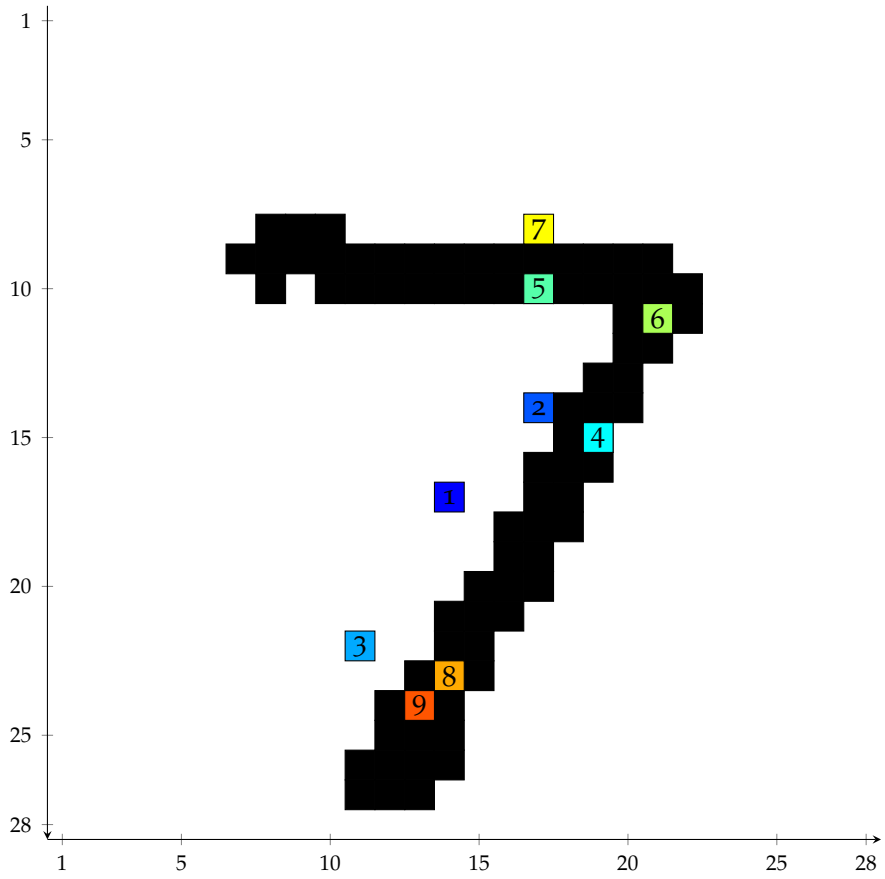


**Figure 5.1:** A handwritten digit seven from the MnistFullTest dataset along with the pixels that are inspected by a simple decision tree, based on the $EC^2$ score function, until a decision is made. The numbers in the colored squares depict the sequence in which the individual pixels were queried.

## 5.2 Example

Consider the MnistFullTrain dataset, which encodes hand written digits as pixels of images. We have a total of ten classes given by the digits $\{0, \ldots, 9\}$ and a training set of 60000 images to learn a decision tree. In order to simplify things, we convert the pixels to zero and one values by using 100 as threshold value.

We now ask which individual pixels are the most informative ones to be inspected. As a practical illustration, we could imagine that with some device we can only inspect one pixel at a time at a high cost; therefore, we want to inspect as few pixels as possible to make a decision.

For now, let us assume that we have a method available to learn a decision

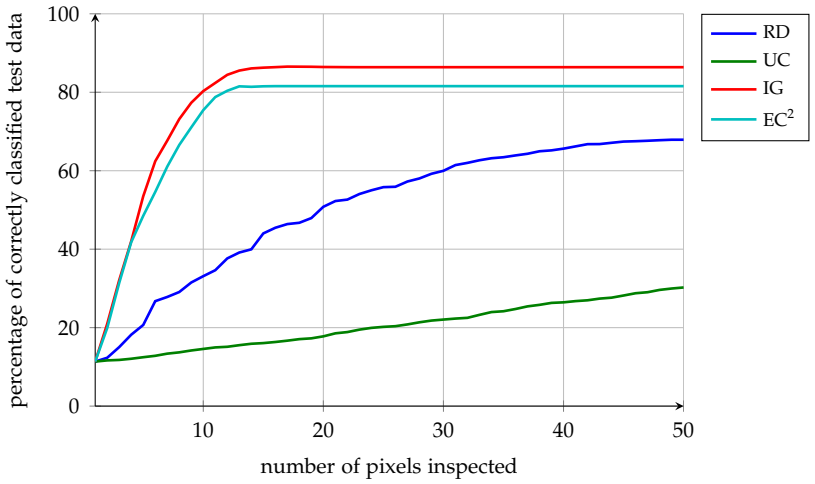


**Figure 5.2:** Percentage of correctly classified examples from the MnistFullTest dataset depending on the number of features that are inspected. As classifiers, we used simple decisions tree based on different score functions and trained on the MnistFullTrain dataset. Note that we do not show any error bars in this plot as all computations – except for the calculation of the random score function – are deterministic.

tree from the MnistFullTrain training set based on the EC² score function. We can use this decision tree to classify new images, for example a digit from the MnistFullTest set. Then Figure 5.1 shows which pixels are inspected in what order until the final and in this case correct classification is made. Note that only nine pixels out of the total 784 are inspected, i.e. only a small subset of all the information available is needed to decide on a class.

However, not all test samples are correctly classified. Figure 5.2 gives an overview of how decision trees, based on different score functions, perform when tested on all examples from the MnistFullTest dataset. We can see that the performances of the four score functions differ significantly. Not surprisingly, the fewer mistakes made, the more pixels are inspected until a certain point is reached – usually the maximum level of the tree. Therefore, we can see from this figure that in this case a decision tree based on the information gain or EC² score function is much shorter than one made with the help of the uncertainty or random score.

## 5.3 Learning Decision Trees

Let us now have a closer look at how to learn decision trees. We assume here that the dataset has already been transformed such that all features take on

**Algorithm 5.1** Simple greedy algorithm to learn a decision tree from a set of data points $\mathcal{X}$ by using a given score function. The main part of the work is done in the call to GrowDecisionTree, a function that is detailed in Algorithm 5.2.

**procedure** LearnDecisionTree($\mathcal{X}$, score$(\cdot)$)
  $T \leftarrow$ GrowDecisionTree($\mathcal{X}$, $\emptyset$, score$(\cdot)$)
  optionally prune decision tree $T$
  **return** $T$
**end procedure**

**Algorithm 5.2** A recursive function to learn a decision tree. The set $\mathcal{S}_i$ holds the observed outcomes at the $i$-th tree depth, $\mathcal{X}_i \subseteq \mathcal{X}$ are the data points consistent with these observations, and a score function specifies the strategy to use in selecting the next feature.

**function** GrowDecisionTree($\mathcal{X}_i$, $\mathcal{S}_i$, score$(\cdot)$)
  **if** stop tree construction **then**
    **return** $T = \emptyset$
  **end if**
  choose feature $t^\star = \arg\max_{t \in \mathcal{T}} \text{score}(t \,|\mathcal{S}_i)$
  $\mathcal{S}_{i+1}^0 \leftarrow \mathcal{S}_i \cup \{t^\star, 0\}$
  $\mathcal{S}_{i+1}^1 \leftarrow \mathcal{S}_i \cup \{t^\star, 1\}$
  $\mathcal{X}_{i+1}^0 \leftarrow \{\mathbf{x} \in \mathcal{X}_i : \mathbf{x}_{[t^\star]} = 0\}$
  $\mathcal{X}_{i+1}^1 \leftarrow \{\mathbf{x} \in \mathcal{X}_i : \mathbf{x}_{[t^\star]} = 1\}$
  subtree $T_0 \leftarrow$ GrowDecisionTree($\mathcal{X}_{i+1}^0$, $\mathcal{S}_{i+1}^0$, score$(\cdot)$)
  subtree $T_1 \leftarrow$ GrowDecisionTree($\mathcal{X}_{i+1}^1$, $\mathcal{S}_{i+1}^1$, score$(\cdot)$)
  create tree $T$ with two children $T_0$ and $T_1$
  **return** $T$
**end function**

binary values only.[5]

The simple greedy algorithm that we use is depicted in Algorithm 5.1 and requires the dataset $\mathcal{X}$ together with a score function as input. It uses the function GrowDecisionTree shown in Algorithm 5.2 to build the tree recursively, in each level selecting the current best feature with the given score function according to the input data and observed test outcomes. Note that the algorithm we present here is very abstract; in practice, we will make sure that at each tree level additional information is stored, such as the class that currently fits the observations best.

[5] Another, more complicated possibility is to do the binary transformation on-the-fly, so that (for non-discrete valued features) the algorithm in each step not only decides which feature to pick, but also what threshold to use.

## 5.4 Stopping Tree Growth and Pruning

In any active learning scenario – no matter if we look at the selection of data points or learning decision trees – a major challenge is to decide when to stop learning and pronounce the current model "good enough". If a learning algorithm stops too soon, it may not have learned a good classifier (yet) and deliver poor performance. If, on the other hands, a learner stops too late, it risks overfitting the observed data and looses some of the speed advantage over classical non-active learning methods, because it requests more labels than necessary.

When learning decision trees, there are two important aspects that relate to this challenge. On one hand, the algorithm needs to decide when to stop constructing the tree; on the other hand, we can prune the tree after it has been constructed to reduce overfitting. Note that if we have a choice, we prefer to stop the tree construction rather than pruning it afterwards, as this makes the algorithm that constructs the tree more efficient.

There are three basic criteria that tell the algorithm to stop adding additional levels to the tree:

1. All features have been inspected.
2. The algorithm runs out of training data that agree with the current observations.
3. All remaining training data are from a single class.

For the case of the EC² score function, we now present a new, more sophisticated stopping criterion that allows us to keep decision trees small while constructing them.

### 5.4.1 EC² Stopping Criterion

Recall from Section 4.2.1 that the EC² criterion conceptually operates on a graph and calculates how many edges are cut in that graph by a certain observation. Our approach is to make a probabilistic statement on how likely it is that in any given step, the edges that are cut are not informative anymore, i.e. that these edges are only due to noise rather than any real pattern in the data.

First, we compute the expected number of edges that have not been cut in the graph after a certain number of observations because they were perturbed by noise.

**Lemma 5.1** *Suppose $\mathcal{S}_\mathcal{A} = \{\bigcup_{t\in\mathcal{A}}(t, z_t)\}$ are the outcomes from tests $\mathcal{A}$ that we have seen so far using the EC² algorithm. Let then $X(\mathcal{S}_\mathcal{A})$ be a random variable denoting the number of edges that have not been cut due to noise after observing*

*outcomes $\mathcal{S}_\mathcal{A}$, i.e. in the noiseless case these edges would have been cut by one of the observations $\mathcal{S}_\mathcal{A}$, but because of noise, they still remain in the graph.*

*Further suppose that a noise parameter $0 \leq \alpha \leq 1$ denotes the probability that each observation we make is wrong, independent from other observations. We assume that if such an erroneous reading occurs, one of the other outcomes is picked with equal probability. Furthermore, let $\ell$ be the number of distinct outcomes a test can result in, let*

$$\mathcal{E} = \bigcup_{1 \leq i < j \leq m} \{\{h, h'\} : h \in \mathcal{H}_i, h' \in \mathcal{H}_j\}$$

*be all edges in the original graph constructed by EC², and let*

$$\mathcal{E}(\mathcal{S}_\mathcal{A}) = \bigcup_{1 \leq i < j \leq m} \{\{h, h'\} : h \in \mathcal{H}_i \cap \mathcal{V}(\mathcal{S}_\mathcal{A}), h' \in \mathcal{H}_j \cap \mathcal{V}(\mathcal{S}_\mathcal{A})\}$$

*be the edges that remain uncut after $\mathcal{A}$ tests and $\mathcal{S}_\mathcal{A}$ corresponding observations. Then it holds that*

$$\mathbf{E}[X(\mathcal{S}_\mathcal{A})] = |\mathcal{E}(\mathcal{S}_\mathcal{A})| \cdot \left(1 - \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell - 1)^2}\right)^{|\mathcal{A}|}\right).$$

**Proof** For any hypothesis $h \in \mathcal{H}$ and any already performed test $a \in \mathcal{A}$, we denote with $h(a) \in \mathcal{Z}$ the true outcome that would have been seen without any noise, and with $\hat{h}(a) \in \mathcal{Z}$ the outcome that the EC² algorithm actually observed, i.e. the outcome influenced by $\Theta$. Note that with probability $\alpha$, we have $\hat{h}(a) \neq h(a)$, and with probability $(1 - \alpha)$, it holds that $\hat{h}(a) = h(a)$.

We have

$$X(\mathcal{S}_\mathcal{A}) = \sum_{\{h,h'\} \in \mathcal{E}} X_{h,h'}(\mathcal{S}_\mathcal{A}) = \sum_{\{h,h'\} \in \mathcal{E}(\mathcal{S}_\mathcal{A})} Y_{h,h'}(\mathcal{S}_\mathcal{A})$$

where

$$X_{h,h'}(\mathcal{S}_\mathcal{A}) = \begin{cases} 1 & \text{if } \left(\forall a \in \mathcal{A} : \hat{h}(a) = \hat{h}'(a)\right) \wedge (\exists a \in \mathcal{A} : h(a) \neq h'(a)) \\ 0 & \text{otherwise} \end{cases}$$

are random variables that indicate if edges $\{h, h'\} \in \mathcal{E}$, connecting $h$ and $h'$ in the original graph constructed by EC², have not been cut after observations $\mathcal{S}_\mathcal{A}$ due to noise. Similarly,

$$Y_{h,h'}(\mathcal{S}_\mathcal{A}) = \begin{cases} 1 & \text{if } \exists a \in \mathcal{A} : \left(\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a)\right) \\ 0 & \text{otherwise} \end{cases}$$

are random indicator variables denoting if edges $\{h, h'\} \in \mathcal{E}(\mathcal{S}_\mathcal{A})$ that remain after observations $\mathcal{S}_\mathcal{A}$ would have already been cut in the noiseless case. Note

that in contrast to the random variables $X_{h,h'}(\mathcal{S}_\mathcal{A})$, which are defined for every edge in the original graph, the variables $Y_{h,h'}(\mathcal{S}_\mathcal{A})$ only operate on the edges in the graph that remain after observations $\mathcal{S}_\mathcal{A}$.

Using linearity of expectation, and noting that for any binary random variable $X$ we have $\mathbf{E}[X] = 1 \cdot \mathbf{Pr}[X = 1] + 0 \cdot \mathbf{Pr}[X = 0] = \mathbf{Pr}[X = 1]$, we calculate

$$\begin{aligned}
\mathbf{E}[X(\mathcal{S}_\mathcal{A})] &= \mathbf{E}\left[\sum_{\{h,h'\}\in\mathcal{E}(\mathcal{S}_\mathcal{A})} Y_{h,h'}(\mathcal{S}_\mathcal{A})\right] \\
&= \sum_{\{h,h'\}\in\mathcal{E}(\mathcal{S}_\mathcal{A})} \mathbf{E}[Y_{h,h'}(\mathcal{S}_\mathcal{A})] \\
&= \sum_{\{h,h'\}\in\mathcal{E}(\mathcal{S}_\mathcal{A})} \mathbf{Pr}[Y_{h,h'}(\mathcal{S}_\mathcal{A}) = 1] \\
&= \sum_{\{h,h'\}\in\mathcal{E}(\mathcal{S}_\mathcal{A})} (1 - \mathbf{Pr}[Y_{h,h'}(\mathcal{S}_\mathcal{A}) = 0])
\end{aligned}$$

and further have

$$\begin{aligned}
\mathbf{Pr}[Y_{h,h'}(\mathcal{S}_\mathcal{A}) = 0] &= \mathbf{Pr}\Big[\forall a \in \mathcal{A} : \Big(\hat{h}(a) \neq \hat{h}'(a) \vee h(a) = h'(a)\Big)\Big] \\
&= \prod_{a\in\mathcal{A}} \mathbf{Pr}\Big[\hat{h}(a) \neq \hat{h}'(a) \vee h(a) = h'(a)\Big] \\
&= \prod_{a\in\mathcal{A}} \Big(1 - \mathbf{Pr}\Big[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a)\Big]\Big)
\end{aligned}$$

because of our independence assumption and basic properties of probability theory.

Next, we calculate the probability that given two hypotheses $h, h' \in \mathcal{H}$ and some test $a \in \mathcal{A}$, the observed outcomes $\hat{h}(a)$ and $\hat{h}'(a)$ are equal while the true outcomes $h(a)$ and $h'(a)$ would not have been equal:

$$\begin{aligned}
&\mathbf{Pr}\Big[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a)\Big] \\
&\quad= \mathbf{Pr}\Big[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a) \wedge h(a) \neq \hat{h}(a) \wedge h'(a) \neq \hat{h}'(a)\Big] && (5.1) \\
&\qquad+ \mathbf{Pr}\Big[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq \hat{h}(a) \wedge h'(a) = \hat{h}'(a)\Big] && (5.2) \\
&\qquad+ \mathbf{Pr}\Big[\hat{h}(a) = \hat{h}'(a) \wedge h(a) = \hat{h}(a) \wedge h'(a) \neq \hat{h}'(a)\Big] && (5.3) \\
&\quad= \alpha^2 \frac{\ell - 2}{(\ell-1)^2} + (1-\alpha)\alpha \frac{1}{\ell-1} + \alpha(1-\alpha)\frac{1}{\ell-1} \\
&\quad= \frac{\alpha^2(\ell-2)}{(\ell-1)^2} + \frac{2\alpha(1-\alpha)}{\ell-1} = \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}
\end{aligned}$$

There are three cases to distinguish, corresponding to the lines numbered (5.1), (5.2), and (5.3):

(5.1) Both outcomes $h(a)$ and $h'(a)$ were influenced by noise and both changed into a single third outcome.

(5.2) Outcome $h(a)$ changed into the outcome of $h'(a)$, which remained the same.

(5.3) The reverse case happened, i.e. while outcome $h(a)$ was not perturbed by noise, $h'(a)$ changed to the same value.

The respective probabilities are calculated by means of the noise parameter $\alpha$ and counting possibilities.

Substituting, we thus arrive at

$$\begin{aligned}\mathbf{Pr}[Y_{h,h'}(\mathcal{S}_\mathcal{A}) = 0] &= \prod_{a\in\mathcal{A}} \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}\right) \\ &= \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}\right)^{|\mathcal{A}|}\end{aligned}$$

and

$$\begin{aligned}\mathbf{E}[X(\mathcal{S}_\mathcal{A})] &= \sum_{\{h,h'\}\in\mathcal{E}(\mathcal{S}_\mathcal{A})} \left(1 - \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}\right)^{|\mathcal{A}|}\right) \\ &= |\mathcal{E}(\mathcal{S}_\mathcal{A})| \cdot \left(1 - \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}\right)^{|\mathcal{A}|}\right)\end{aligned}$$

which is what we wanted to prove. □

Outfitted with a way of calculating the expected number of edges not cut, standard tools of probability theory allow us to make a probabilistic statement on how certain we are that the edges that are cut in a certain step by the EC² criterion are not due to noise.

**Theorem 5.2** *Suppose $\mathcal{S}_\mathcal{A}$ are the outcomes from tests $\mathcal{A}$ that we have seen so far using the EC² algorithm. Let $|\mathcal{E}(\mathcal{S}_\mathcal{A})|$ be the number of edges remaining uncut, let $S(\mathcal{S}_\mathcal{A})$ be the number of edges the EC² algorithm will cut, and let $X(\mathcal{S}_\mathcal{A})$ be a random variable denoting the number of edges that have not been cut due to noise given observations $\mathcal{S}_\mathcal{A}$.*

*For the number of distinct outcomes $\ell$, a noise parameter $0 \leq \alpha \leq 1$, some $\varepsilon \geq 0$ and $S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] \geq \varepsilon$ it holds that*

$$\mathbf{Pr}[S(\mathcal{S}_\mathcal{A}) - X(\mathcal{S}_\mathcal{A}) > \varepsilon] \geq 1 - \exp\left(-\frac{2\left(S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon\right)^2}{|\mathcal{E}(\mathcal{S}_\mathcal{A})|}\right)$$

*where*

$$\mathbf{E}[X(\mathcal{S}_\mathcal{A})] = |\mathcal{E}(\mathcal{S}_\mathcal{A})| \cdot \left(1 - \left(1 - \frac{\alpha(2\ell - \ell\alpha - 2)}{(\ell-1)^2}\right)^{|\mathcal{A}|}\right)$$

*assuming i.i.d. noise.*

**Proof** We can write

$$\begin{aligned}
&\mathbf{Pr}[S(\mathcal{S}_\mathcal{A}) - X(\mathcal{S}_\mathcal{A}) > \varepsilon] \\
&\quad = 1 - \mathbf{Pr}[S(\mathcal{S}_\mathcal{A}) - X(\mathcal{S}_\mathcal{A}) \leq \varepsilon] \\
&\quad = 1 - \mathbf{Pr}[X(\mathcal{S}_\mathcal{A}) \geq S(\mathcal{S}_\mathcal{A}) - \varepsilon] \\
&\quad = 1 - \mathbf{Pr}[X(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] \geq S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon]
\end{aligned}$$

because of basic properties of probabilities. $X(\mathcal{S}_\mathcal{A})$ is a sum of $|\mathcal{E}(\mathcal{S}_\mathcal{A})|$ independent bounded random variables $0 \leq Y_{h,h'}(\mathcal{S}_\mathcal{A}) \leq 1$, as shown during the calculation of $\mathbf{E}[X(\mathcal{S}_\mathcal{A})]$ in Lemma 5.1, allowing us to apply Hoeffding's inequality ([20]), which yields

$$\begin{aligned}
&\mathbf{Pr}[X(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] \geq S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon] \\
&\quad \leq \exp\left(-\frac{2\left(S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon\right)^2}{\sum_{i=1}^{|\mathcal{E}(\mathcal{S}_\mathcal{A})|}(1-0)^2}\right) \\
&\quad = \exp\left(-\frac{2\left(S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon\right)^2}{|\mathcal{E}(\mathcal{S}_\mathcal{A})|}\right)
\end{aligned}$$

for $S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] \geq \varepsilon$.

Thus,

$$\mathbf{Pr}[S(\mathcal{S}_\mathcal{A}) - X(\mathcal{S}_\mathcal{A}) > \varepsilon] \geq 1 - \exp\left(-\frac{2\left(S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon\right)^2}{|\mathcal{E}(\mathcal{S}_\mathcal{A})|}\right)$$

by substitution. □

We can now use this result when constructing decision trees in a straightforward way.

**Corollary 5.3** *We assume that $\mathcal{S}_\mathcal{A}$ are the outcomes from tests $\mathcal{A}$ that we have seen so far in the EC² algorithm. Using the confidence level $\delta$ that we define as*

$$\delta := \begin{cases} 0 & \text{if } S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] < \varepsilon \\ 1 - \exp\left(-\frac{2\left(S(\mathcal{S}_\mathcal{A}) - \mathbf{E}[X(\mathcal{S}_\mathcal{A})] - \varepsilon\right)^2}{|\mathcal{E}(\mathcal{S}_\mathcal{A})|}\right) & \text{otherwise} \end{cases}$$

*we obtain the following stopping criterion that tells us if we should stop at the current level of the decision tree construction. Alternatively, the same criterion can be used to prune an already constructed decision tree.*

*Let $0 \leq \delta_{\min} \leq 1$ be a threshold that describes how high the computed confidence needs to be at least. Then we stop the construction if $\delta < \delta_{\min}$ and continue on with the EC² algorithm if $\delta \geq \delta_{\min}$.*

Note that for the computation of $\mathbf{E}[X(\mathcal{S}_\mathcal{A}]$ in Lemma 5.1, we assume that when an outcome is perturbed by noise, it changes into any other outcome with equal probability. If we instead assume a prior distribution over the outcomes, we arrive at a more general result.

**Lemma 5.4** *Suppose the same setting as described in Lemma 5.1, except that* $\mathrm{p}(z)$ *is a prior distribution over the set of outcomes* $\mathcal{Z}$ *that describes how likely each* $z \in \mathcal{Z}$ *is picked as an outcome in case of an error due to noise.*

*Then it holds that*

$$\mathbf{E}[X(\mathcal{S}_\mathcal{A})] = |\mathcal{E}(\mathcal{S}_\mathcal{A})| \cdot \left(1 - (1-\beta)^{|\mathcal{A}|}\right)$$

*where*

$$\begin{aligned}\beta = \mathbf{E}_{z\sim Z, z'\sim Z'}&\left[\alpha^2 \left(\frac{(1-\mathrm{p}(z)-\mathrm{p}(z'))\left(\sum_{z''\in\mathcal{Z}} \mathrm{p}(z'')^2\right)}{(1-\mathrm{p}(z))(1-\mathrm{p}(z'))}\right)\right] \\ &+ \mathbf{E}_{z\sim Z, z'\sim Z'}\left[\alpha(1-\alpha)\left(\frac{\mathrm{p}(z)}{1-\mathrm{p}(z')} + \frac{\mathrm{p}(z')}{1-\mathrm{p}(z)}\right)\right]\end{aligned}$$

*with* $Z$ *and* $Z'$ *denoting random variables distributed according to* $\mathrm{p}(z)$.

**Proof Sketch** The proof is very similar to the one already presented for Lemma 5.1. The main difference lies in the calculation of the probability $\mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a)\right]$. We write

$$\begin{aligned}&\mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a)\right] \\ &\quad = \mathbf{E}_{z\sim Z, z'\sim Z'}\left[\mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a) \,\middle|\, z, z'\right]\right]\end{aligned}$$

where $z$ and $z'$ are the outcomes $h(a)$ and $h'(a)$ if we fix hypotheses $h$ and $h'$. We can then calculate

$$\begin{aligned}&\mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a) \,\middle|\, z, z'\right] \\ &\quad = \mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq h'(a) \wedge h(a) \neq \hat{h}(a) \wedge h'(a) \neq \hat{h}'(a) \,\middle|\, z, z'\right] \\ &\qquad + \mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) \neq \hat{h}(a) \wedge h'(a) = \hat{h}'(a) \,\middle|\, z, z'\right] \\ &\qquad + \mathbf{Pr}\left[\hat{h}(a) = \hat{h}'(a) \wedge h(a) = \hat{h}(a) \wedge h'(a) \neq \hat{h}'(a) \,\middle|\, z, z'\right] \\ &\quad = \alpha^2 \frac{(1-\mathrm{p}(z)-\mathrm{p}(z'))\left(\sum_{z''\in\mathcal{Z}} \mathrm{p}(z'')^2\right)}{(1-\mathrm{p}(z))(1-\mathrm{p}(z'))} \\ &\qquad + (1-\alpha)\alpha\frac{\mathrm{p}(z')}{1-\mathrm{p}(z)} + \alpha(1-\alpha)\frac{\mathrm{p}(z)}{1-\mathrm{p}(z')}\end{aligned}$$

and arrive at the final result by straight-forward substitution. □

### 5.4.2 Chi-Squared Pruning Method

We have looked at ways of deciding when to stop the construction of a decision tree. Alternatively, we can prune a tree in a second step after it has been built. A statistically motivated and popular way to prune is the so-called chi-squared test[6] ($\chi^2$-test), which allows the removal of nodes that with high probability split the data only due to noise but not any real structure. The $\chi^2$-test extends the information gain criterion since both operate in the same setting, but in principle can also be used together with any other decision tree construction method.

When looking at a node that splits the data according to some feature in our binary decision tree, we calculate the test statistic

$$\chi^2 = \sum_{y \in \mathcal{Y}} \sum_{z \in \mathcal{Z}} \frac{\left(o_{y,z} - e_{y,z}\right)^2}{e_{y,z}}$$

where $\mathcal{Y}$ is the set of all class labels, $z$ selects the child with the value of the feature fixed to 0 or 1, $o_{y,z}$ are the observed frequencies, and $e_{y,z}$ are the expected frequencies for child $z$ and class $y$. With the cumulative distribution function of the chi-squared distribution,[7] we can then calculate how likely it is that the observed outcomes are statistically independent. If the result exceeds a certain $p$-value threshold, we suspect that the feature we selected does not inhibit any statistically significant information gain, and we prune the tree at that point. This procedure is applied recursively on a tree starting at its leaves and repeated until no more nodes are pruned.

However, as Walpole et al. ([46, p. 377]) note, the $\chi^2$-test is only statistically sound for large sample sizes. Therefore, we follow their suggestion in using an adjusted test if any expected frequency is lower than 10.

If $5 \leq \min\{e_{y,z}\} \leq 10$, we apply Yates' correction for continuity, which yields

$$\chi^2 = \sum_{y \in \mathcal{Y}} \sum_{z \in \mathcal{Z}} \frac{\left(|o_{y,z} - e_{y,z}| - 0.5\right)^2}{e_{y,z}}$$

and then proceed in the same way as above.

If $\min\{e_{y,z}\} < 5$, we directly calculate the result to compare against the

[6] Also known as the test for independence or Pearson's chi-squared test; it is described in detail by Walpole et al. ([46]).

[7] In our case of the binary decision tree, the chi-squared distribution has one degree of freedom and the cumulative distribution function can be evaluated for some value $x$ by calculating $P(1/2, x/2)$, where $P(\cdot)$ is the regularized Gamma function.

$p$-value threshold with Fisher's exact test[8]

$$p_{\text{fisher}} = \frac{\prod_z \text{fac}\left(\sum_y o_{y,z}\right) \cdot \prod_y \text{fac}\left(\sum_z o_{y,z}\right)}{\text{fac}\left(\sum_{y,z} o_{y,z}\right) \cdot \prod_{y,z} \text{fac}\left(o_{y,z}\right)}$$

where $\text{fac}(x)$ is the factorial of $x$.

In principle, the $\chi^2$-test can also be used as a stopping criterion while constructing the decision tree. However, due to the nature of the information gain criterion usually paired with it, the algorithm can run into situations where no immediate information gain is visible in any single step, although several steps may indeed produce new information.[9] Therefore, we regard the $\chi^2$-test in this thesis as a post-construction pruning criterion.

## 5.5 Experimental Results

We evaluated the various score functions and stopping criteria on several datasets with a focus on how noise influences the observed performance. All reported results were obtained over 50 iterations, each randomly splitting the dataset into a training set with 90% of the data points and a test set with the remaining 10% of the data points. The decision trees were then trained on the training set and afterwards tested on the test set. We assume that the given datasets[10] did not contain any noise, and created noisy versions of them by randomly changing a certain percentage of their labels.

The results shown in Figure 5.3 help us to get an idea of the overall performance difference of the four score functions we investigate, here evaluated on the Letter dataset. Not using any sophisticated stopping criterion and without performing post-creation pruning, we observe significant differences at low noise levels. It is particularly worthy to note that the uncertainty score function performs worse than random selection, which is something we also observed for other datasets, regardless of pruning. Therefore, we ignore results based on the uncertainty score from now on and focus on the standard information gain versus the $\text{EC}^2$ criterion, with a random feature selection as baseline. For high noise levels, all score functions perform very similar without pruning, which is something we can expect due to overfitting.

Figure 5.4 shows what happens when we prune all decision trees with the $\chi^2$-test using a $p$-value threshold of 0.05. For a random score function,

[8] Also known as the Fisher-Irwin test.

[9] Shi ([41]) mentions the parity test as such an example.

[10] For a description of all datasets used, please refer to Appendix A.

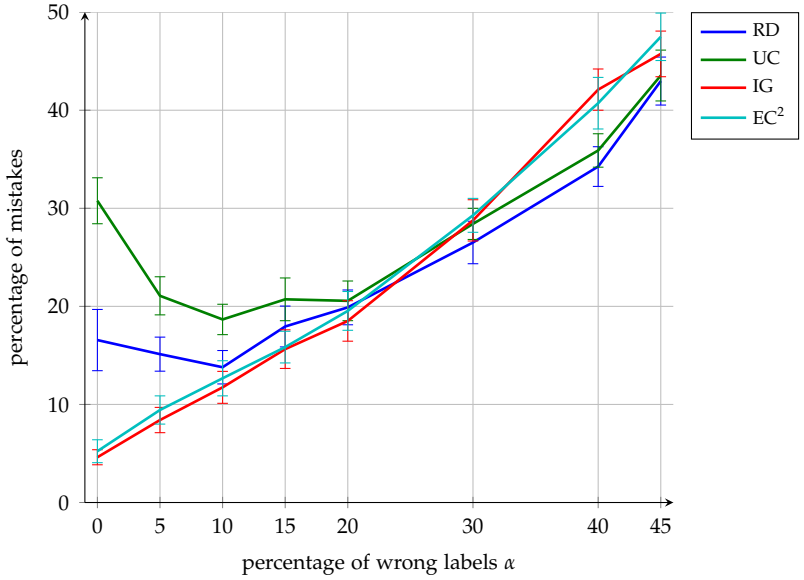


**Figure 5.3:** Percentage of mistakes for various levels of noise created by randomly corrupting a certain percentage of labels for the LETTER dataset. While for low noise levels we can clearly see significant performance differences, due to overfitting, all score functions perform very similar at higher noise levels.

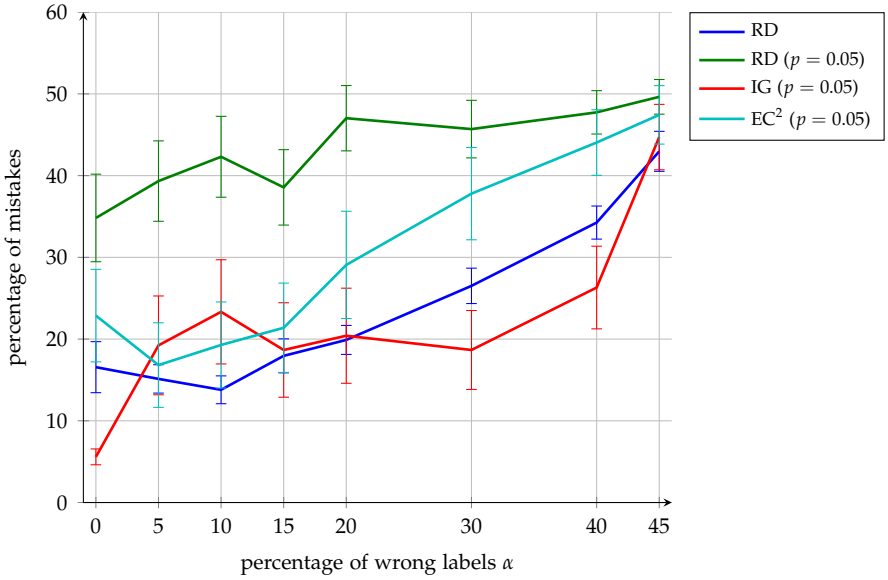


**Figure 5.4:** Percentage of mistakes for various noise levels, where each decision tree has been pruned with the $\chi^2$-test ($p = 0.05$) on the LETTER dataset. Note that with this pruning method, the information gain score function is impacted less by high noise levels.

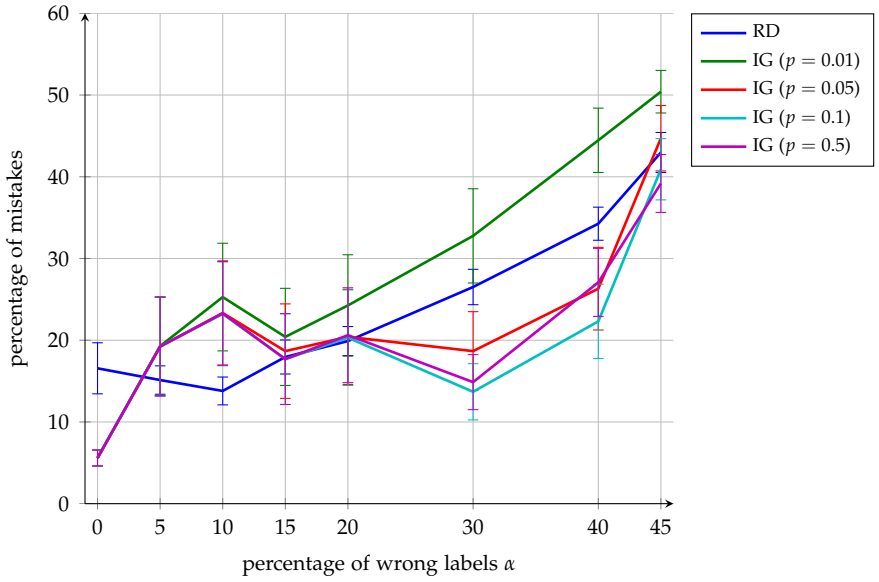


**Figure 5.5:** Percentage of mistakes for various noise levels using the information gain criterion on the Letter dataset. We pruned with the $\chi^2$-test for different levels of statistical significance, and can see that a standard $p$-value threshold of $0.05$ seems to be a reasonable choice.

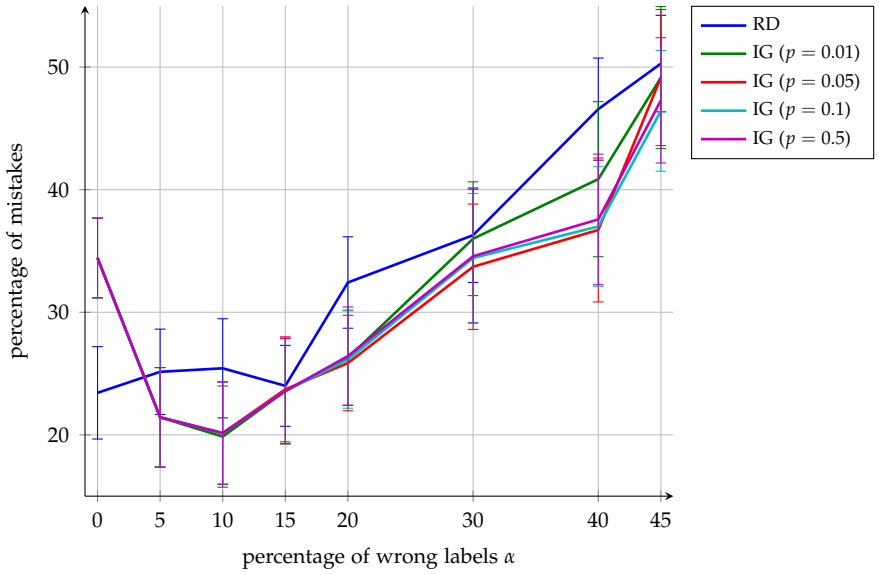


**Figure 5.6:** Percentage of mistakes for various noise levels using the information gain criterion on the Wdbc dataset. Pruning was performed with the $\chi^2$-test for different $p$-value thresholds. No statistically significant differences are observable, therefore a choice of $p = 0.05$ cannot be disputed from these results. For $\alpha = 0$, the poor performance of the information gain criterion paired with the $\chi^2$-test is due to underfitting the data, i.e. too many levels of the tree are pruned.

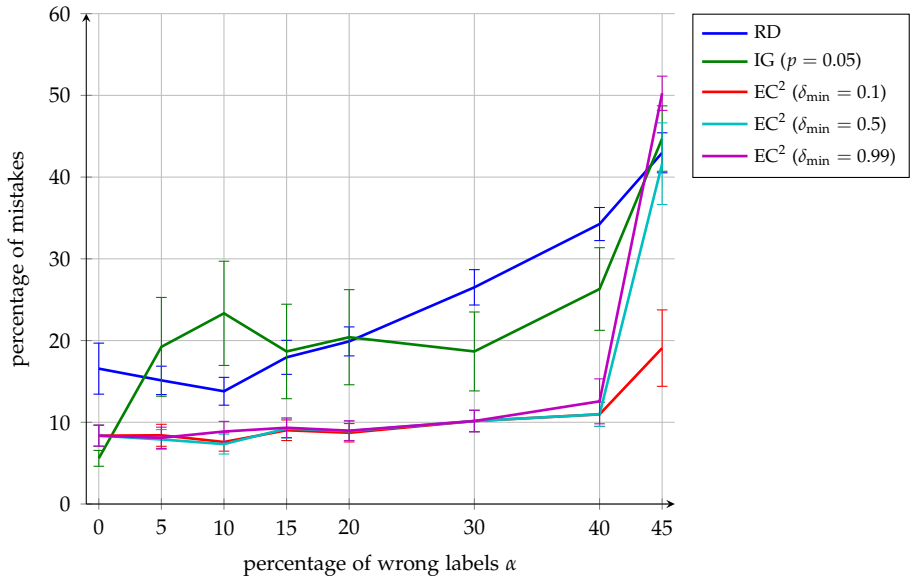


**Figure 5.7:** Percentage of mistakes for various noise levels on the LETTER dataset. The information gain score function was pruned with the $\chi^2$-test ($p = 0.05$), while the decision trees build with the $EC^2$ score utilized the $EC^2$ stopping criterion confidence levels of 0.10, 0.50 and 0.99. The LETTER dataset does not seem to be inherently difficult to classify, as all $EC^2$-based strategies are able to provide relatively good results up to an error rate of 40%.

$\chi^2$-pruning does not help at all, and we will furthermore just report the non-pruned random baseline. Random selection, however, remains very competitive – in particular, it outperforms the pruned EC² criterion. Only the pruned information gain criterion is able to keep up and sometimes makes less mistakes than a simple random score function.

Nevertheless, $p$-values other than 0.05 might give us better results. Figures 5.5 and 5.6 show the influence of different choices on the performance of the information gain score function for two datasets. From the results, we conclude that while other $p$-values may have an impact on performance, a threshold value of 0.05 seems to be a good overall choice.

We have seen that the EC² score function did not perform well when paired with the $\chi^2$-test. But while the $\chi^2$-test was developed for the information gain criterion, we introduced the EC² stopping criterion for the EC² score function. To investigate the influence of the stopping criterion on the EC² score function, we start with comparing it to the random baseline and the pruned information gain score functions, using different choices for the significance level $\delta_{\min}$. Figures 5.7 and 5.8 show that in many cases, the EC² score function used together with its corresponding stopping criterion yields superior performance on the LETTER and WDBC datasets.

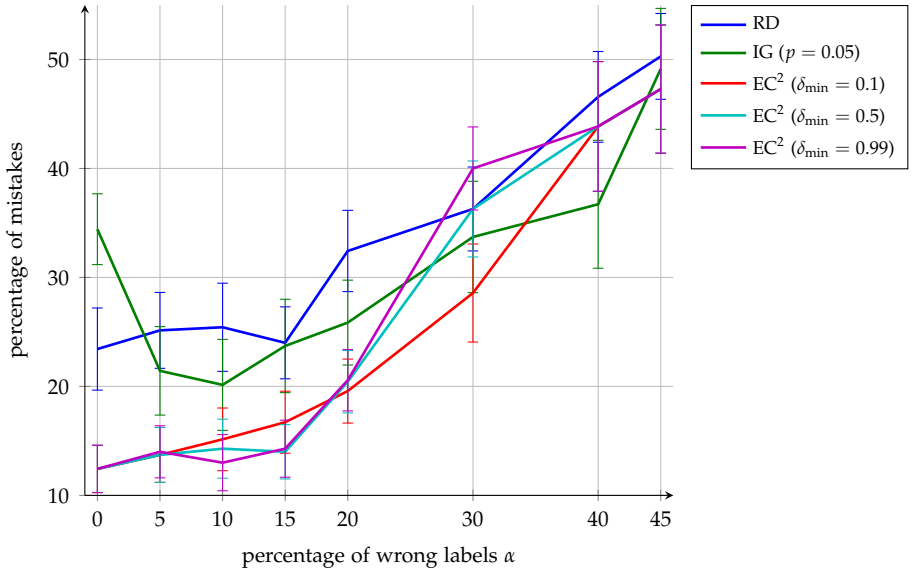


**Figure 5.8:** Percentage of mistakes for various noise levels on the WDBC dataset. The information gain score function was pruned with the $\chi^2$-test ($p = 0.05$) while the decision trees build with the EC$^2$ score utilized the EC$^2$ stopping criterion confidence levels of 0.10, 0.50 and 0.99.

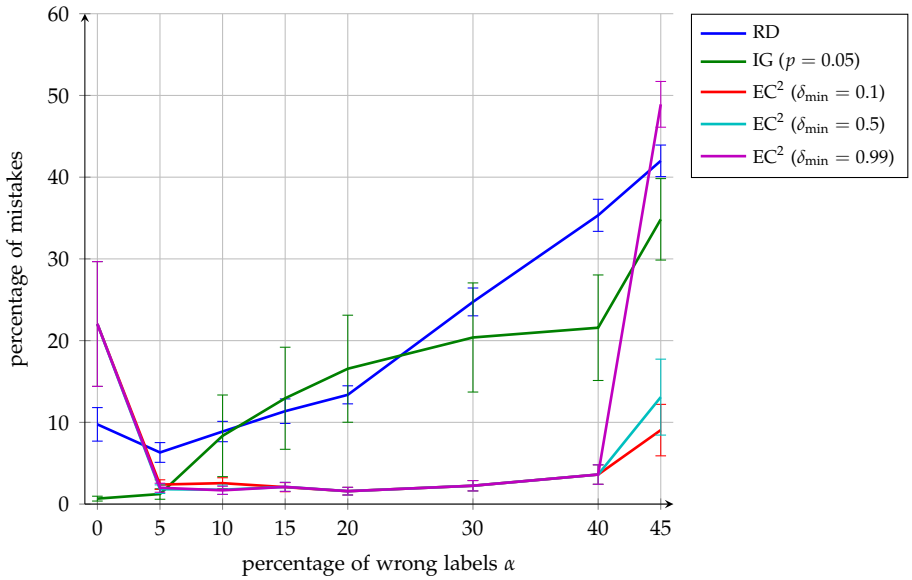


**Figure 5.9:** Percentage of mistakes versus noise levels measured on the PENDIGITS dataset. The EC$^2$-based score functions, regardless of the confidence level used, in general outperform information gain for higher noise levels.

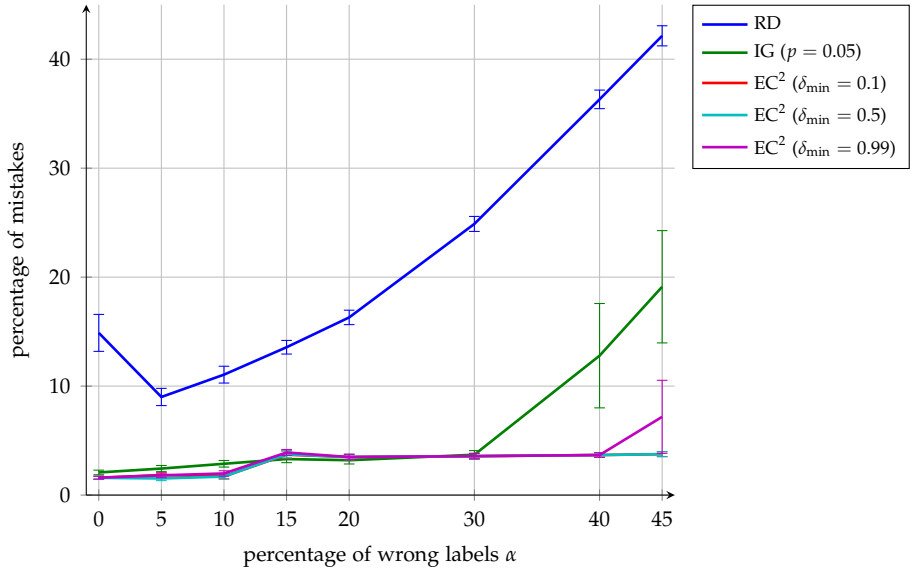


**Figure 5.10:** For the MNIST dataset, all informative score functions show very similar performance for most noise levels $\alpha$, and significantly outperform a random selection procedure. At very high noise levels, the EC$^2$ criterion achieves a lower percentage of mistakes.

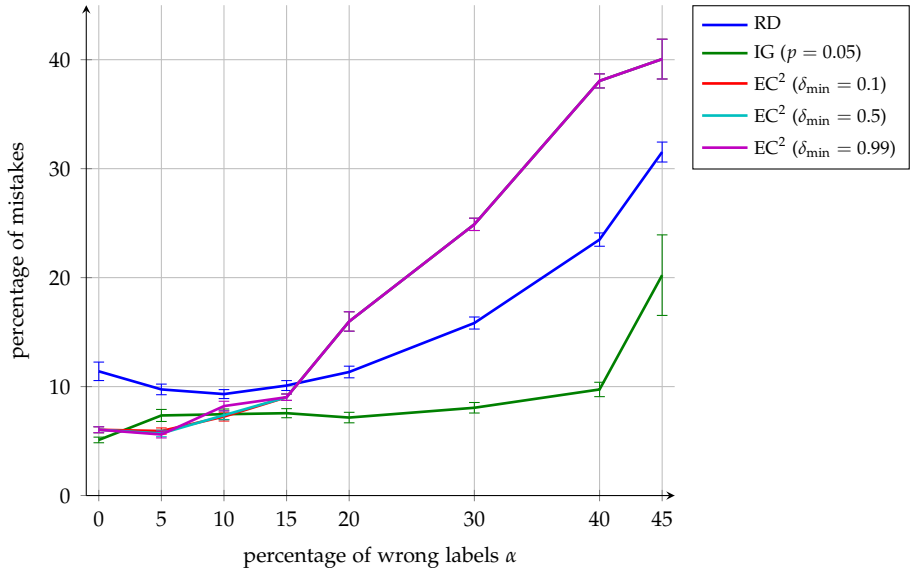


**Figure 5.11:** Using the IBNSINA dataset, the EC$^2$ criterion performs worse than information gain based – and sometimes even random – selection. In addition, the three confidence levels of 0.10, 0.50 and 0.99 result in almost exactly the same outcomes. The poor EC$^2$ performance, especially in regions with a lot of noise, is due to underfitting the data and cutting the decision tree too soon.

To make sure that the EC² stopping criterion does not overfit in these two specific data instances, we extend the investigation to three more datasets. Figures 5.9, 5.10 and 5.11 show results for the Pendigits, Mnist and IbnSina datasets. On the first two datasets, the EC² stopping criterion often achieves better performance, especially for high levels of noise. In particular, note that even for extremely noisy data instances (where almost half of all the training data points given to the algorithm have incorrect labels), the classification results are still acceptable.[11] The results from the IbnSina dataset present different results and show that the EC² stopping criterion is not always the best choice; in some situations, other criteria may outperform it. Nonetheless, even in this situation it remains competitive with the pruned information gain score function for lower levels of noise.

So far, we have focused on the performance in terms of the percentage of wrong labels on the test set. There is another important aspect related to decision tree construction, however, namely the depth of the decision tree, i.e. how many features need to be evaluated before a classification is made. In particular, we now look at the maximum depth of the constructed decision trees, which gives us an upper bound on the number of features that need to be looked at. Additionally, when using the EC² stopping criterion, we get an idea of what the time complexity to build these trees is.

Let us start by looking at the depths of decision trees that were trained on the Letter dataset. As evident from Figure 5.12, all unpruned or unstopped trees quickly reach the maximum tree depth, i.e. they look at all features before making a decision. The challenge, therefore, remains between the information gain criterion that is pruned with the $\chi^2$-test and the EC² score function with its corresponding stopping criterion. There, we note that the information gain based solution shows more variability, and leads to greater tree depths, especially for high noise levels.

Figures 5.13 and 5.14 show the results for the Wdbc and IbnSina datasets, this time without the random baseline. These figures give us some insight into why the EC² stopping criterion sometimes performs poorly. The regions of poor performance observed earlier correspond to very shallow trees, i.e. underfitting occurs. Another observation is that for the IbnSina dataset, the trees pruned with the EC² stopping criterion are significantly smaller than the trees created with the help of the $\chi^2$-test, even though they deliver similar performance in terms of the percentage of mistakes.

[11] However, in this case we know the noise parameter $\alpha$ exactly, which may not be the case in real-world situations.

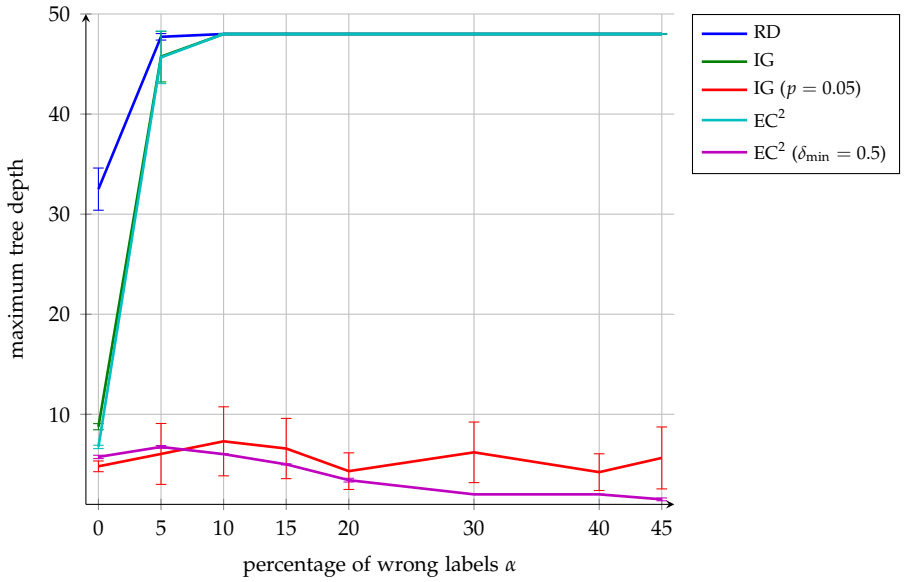


**Figure 5.12:** Maximum depths of decision trees on the LETTER dataset for various levels of noise. Without pruning or using a stopping criterion, all score function reach the maximum tree depth already for low noise levels. Otherwise, for higher noise levels, EC$^2$-based trees exhibit lower depth and less variability than an information gain based scoring.

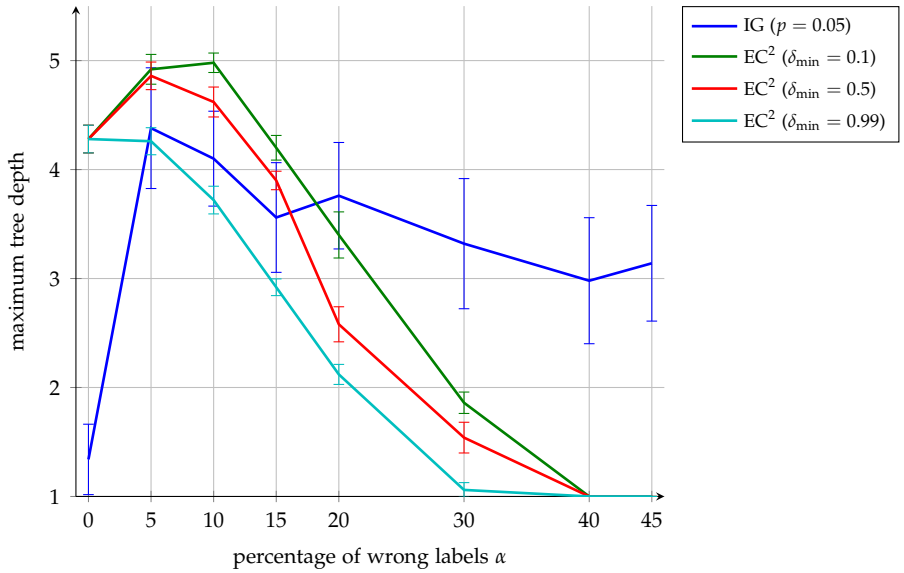


**Figure 5.13:** Maximum tree depth for different noise levels on the WDBC dataset. For high noise levels, the EC$^2$ stopping criterion produces very shallow trees prone to underfitting the data. The size of the trees created by the information gain criterion paired with the $\chi^2$-test, on the other hand, stays relatively stable – although with greater individual fluctuations – for most noise levels.

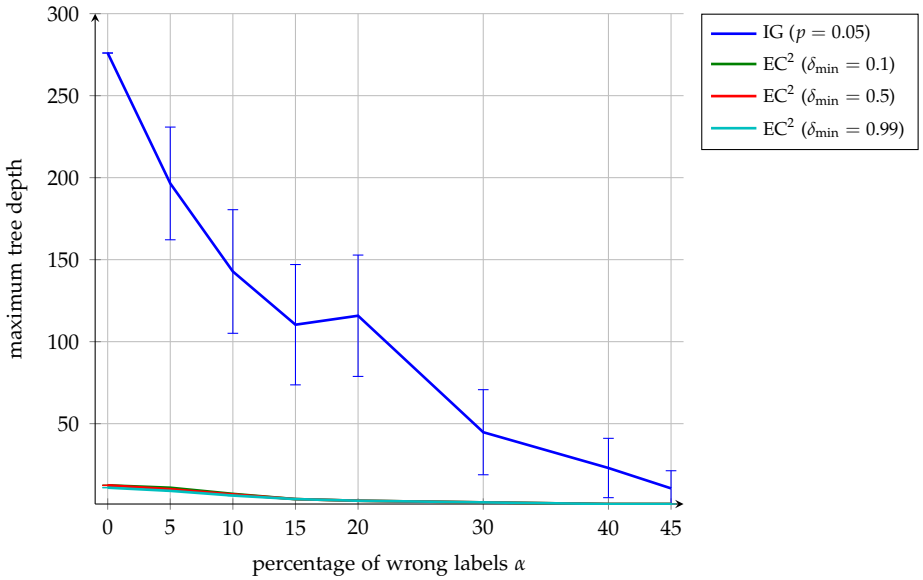


**Figure 5.14:** For the IbnSina dataset, the differences in tree depths are most striking. Note especially that for lower noise levels, the decision trees created with the help of the $EC^2$ stopping criterion perform about the same as an information gain based approach in terms of the classification mistakes made, which can be seen in Figure 5.11, even though they are much shallower and therefore only inspect a fraction of the number of features.

## 5.6 Discussion

Decision trees provide a straight-forward and easily interpretable classification method. By formulating the problem of constructing them – given a dataset – in the right way, we can view it as an active learning task and apply, for example, the EC² score function along with the newly introduced EC² stopping criterion.

How to deal with noisy data is a major open challenge in regards to decision trees. Overfitting is the main issue,[12] which translates back to the questions of when to stop the tree construction and how to best prune a tree.

Comparing the EC² stopping criterion with a combined information gain and $\chi^2$-test solution, we argue that our new criterion exhibits several desirable characteristics:

1. In many cases, the EC² stopping criterion delivers superior performance in terms of the percentage of classification mistakes.

[12] For example, Segal ([36]) notes that classification trees in the form of random forests may overfit for noisy datasets.

2. Especially for high noise levels, the trees created by the EC²-based solution are significantly smaller.
3. The criterion can be applied with exactly the same results either during tree construction or for pruning afterwards.

As a combination of the first two characteristics, sometimes EC²-based trees exhibit a similar performance compared to information gain based trees, but are smaller and thus faster to construct and evaluate.[13] Therefore, in these cases, we expect the EC² score function to select features that are more informative than the ones chosen by the information gain criterion.

One challenge of the EC² stopping criterion, especially for high levels of noise, is underfitting, and so far we have not investigated possible ways of counteracting this issue. In addition, the criterion requires several parameters, such as the (estimated) level of noise in the dataset, in its current form either as fixed value or as a prior distribution over the classes. In our experiments, we introduced noise artificially and were thus aware of the exact[14] noise level. In practice, however, estimating the right value for this parameter might be more challenging.

Last but not least, decision trees are often not used alone, but as a group in constructing random forests. Therefore, different methods of constructing decision trees – for example some based on information gain and the $\chi^2$-test while others by using the EC² score function and its stopping criterion – may provide effective approaches by averaging over different classifiers, with some more prone to overfitting and others tending more towards underfitting.

[13] This is also connected with the third aspect mentioned; as we can use the EC² stopping criterion during tree construction, there is no need to build the full tree anymore, which may heavily boost performance in terms of the speed of the learning algorithm.

[14] This is true at least in regards to the noise that we introduced. As the datasets themselves were based on real-world examples, we expect some additional noise to already have existed beforehand.

Chapter 6

# Conclusion

Active learning offers methods capable to deal with large datasets, as we see them today, and provides efficient and effective algorithms. However, as in other machine learning areas, care must be taken that a particular model is appropriate for the task at hand.

We investigated the OASIS algorithm proposed by Goldberg et al. ([14]) as a one-of-its-kind approach to an integrated online, active and semi-supervised learning framework. From our results, we conclude that since different active learning score functions do not exhibit significant differences in performance, its other aspects – e.g. its likelihood function and particle filter tracking method – have more influence on the observed performance.

When integrating the OASIS algorithm as a prior distribution into a classical active learning framework, our experiments did not produce superior results compared to using uninformative priors. Nonetheless, we observed that informative active learning methods are able to make fewer mistakes than a simple random selection procedure, even though sometimes care has to be taken to choose the right parameter values.

Furthermore, in focusing on the EC² active learning score function introduced by Golovin et al. ([15]), we developed an innovative stopping criterion that can deal well with noisy observations. In an alternative active learning setting – where instead of labels, features are actively queried, which leads to the construction of minimum decision trees –, we compared this new criterion to the standard information gain and $\chi^2$-test combination. Our results show that in general, our EC²-based criterion provides superior performance while building smaller tree classifiers, essentially providing an efficient way to construct such small decision trees.

## 6.1 Future Work

In our opinion, our new EC² stopping criterion provides interesting avenues for future research, both for theoretical as well as practical considerations. As it is based on the EC² score function, which guarantees that the number of requests it makes is close to optimal in the noiseless case, there is hope that our stopping criterion can be used for efficiently constructing close to minimal depth decision trees with bounded expected error rates, even though theoretical work is still needed to provide the necessary formal proofs. At the same time, thinking about the observed weakness of underfitting may lead to a better understanding of its behavior and potential solutions.

From a practical point of view, the datasets we have considered, can be considered small compared to some real world applications. Thus, observing and evaluating the performance of the EC² stopping criterion when learning from millions of data samples might clarify its practical merit. In addition, further experiments may provide insight into how to choose the noise parameter $\alpha$ in real world scenarios.

Apart from these research areas, there are also a number of smaller gaps left open that have not been filled in this thesis:

- We have applied the EC² stopping criterion to the task of actively selecting features only, but it can also be employed for the classical active learning problem of querying labels for data points.
- While we did investigate a number of parameters, there are many other options and variations that could still be looked at, e.g. different particle filter methods for the OASIS algorithm or alternative clustering methods when translating OASIS particles into EC² hypothesis classes.

We look forward to the development in the years to come, and hope that the reader will take part in providing sound methods for the world of tomorrow to handle its increasing demand of knowledge extracted from an abundance of data.

Appendix A

# Datasets

Throughout this thesis, we use various datasets for experiments and examples. In this chapter, we shortly outline each of them and provide references as to where to obtain them. Table A.1 provides an overview of all datasets used, specifying their size, number of features, and the domain they come from.

| Name | Features | Data points | Domain |
|---|---|---|---|
| Diced | $\geq 1$ | $\geq 1$ | Synthetic |
| Letter | 16 | 1555 | Handwritten letters |
| Pendigits | 16 | 2286 | Handwritten digits |
| Wdbc | 30 | 568 | Breast cancer diagnostics |
| Mnist | 10 | 14780 | Handwritten digits |
| MnistFullTrain | 784 | 60000 | Handwritten digits |
| MnistFullTest | 784 | 10000 | Handwritten digits |
| IbnSina | 92 | 20722 | Arabic handwriting |

**Table A.1:** Overview of all datasets used in this thesis. *Features* describes how many dimensions the dataset has, *data points* is the number of samples present, and *domain* mentions the area each dataset was collected from.

## A.1 Diced

The synthetic Diced dataset is used and described by Goldberg et al. ([14]). Let $d$ be the number of dimensions and $n$ the desired size of the dataset. An instance of this dataset is generated by sampling $n$ data points in $d$ dimensions from the uniform distribution with each feature taking on values from the interval $[-0.5; -\varepsilon] \cup [\varepsilon; 0.5]$ where $\varepsilon = \frac{1}{2}(n/10)^{-1/d}$. The data points are then partitioned into two classes according to the value of their first dimension – if it is greater than zero the class label is 1, if less than zero then we use the label 0.

## A.2 Letter Recognition

The *Letter Recognition Data Set* is available at the UCI Machine Learning Repository ([12]), where we view the provided training and test sets as a single unified dataset. It consists of 20000 data points with 16 attributes divided into 26 classes according to the capital letters of the English alphabet. The attributes are measurements of handwritten letters and consist of integer values. Selecting only the letters *A* and *B* yields the Letter dataset with 1555 data points that is used in this thesis.

## A.3 Pen-Based Recognition of Handwritten Digits

The *Pen-Based Recognition of Handwritten Digits Data Set* is also available at the UCI Machine Learning Repository ([12]). The whole dataset contains 10992 instances of hand-written digits described by 16 attributes. For the Pendigits dataset used in this thesis, we use only two digits, namely 0 and 1, which reduces the number of data points to 2286.

## A.4 Breast Cancer Wisconsin (Diagnostic)

The *Breast Cancer Wisconsin (Diagnostic) Data Set* can be obtained from the UCI Machine Learning Repository ([12]) as well, and provides measurements for the diagnosis of cancerous cells. We use it without any modification as the Wdbc dataset, and it provides 568 data points in 30 dimensions.

## A.5 MNIST Database of Handwritten Digits

The *MNIST Database of Handwritten Digits* ([22]) consists of a training set of 60000 examples, which we denote MnistFullTrain, and a test set of 10000 examples, which we call MnistFullTest. Each data point is given by a 784-dimensional vector that encodes the pixel values of an image with a width and height of 28 pixels, and belongs to one of 10 classes denoting the digits zero to nine.

We construct an additional dataset, called Mnist, based on the MNIST database. Following Goldberg et al. ([14]), we combine all training and test instances for the digits 0 and 1, resulting in 14780 data instances. Then, we perform principal component analysis (PCA) on the first 1000 images of the training dataset, and use the first 10 principal components to project the whole dataset to 10 dimensions.

## A.6 Ibn Sina Handwriting Recognition

The *Ibn Sina Handwriting Recognition Data Set* was compiled by the group of Mohamed Chériet with the objective to recognize Arabic words from an ancient manuscript written by Ibn Sina, also known under his Latinized name Avicenna. It consists of 20722 examples, each with 92 feature dimensions, and can be obtained from the Active Learning Challenge 2009 ([1]).

Appendix B

# Experimental Notes

This chapter outlines the software created and hardware used to arrive at the experimental results presented in this thesis.

## B.1 Software Implementation

We used MATLAB R2010a – version 7.10.0.499, 64-bit – to implement and run all experiments. To measure the runtimes of algorithms, we used the function `cputime` instead of the recommended `tic` and `toc` functions because the simulation environment was a shared server that could at any time also be executing other tasks.

## B.2 Hardware

The server on which all calculations were performed for the results presented is a parallel machine with 128 GB RAM and 24 Intel® Xeon® X7542 processors, each clocked at 2.66 GHz and consisting of six cores.

As operating system, Red Hat Enterprise Linux (RHEL) was used with Linux version 2.6.32.[1]

[1] The exact version number is 2.6.32-220.4.1.1.el6.x86_64.

# Bibliography


[1] Active learning challenge, 2009. http://www.causality.inf.ethz.ch/activelearning.php [accessed April 6, 2012].

[2] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] J.M. Bland and D.G. Altman. Bayesians and frequentists. *BMJ*, 317(7166):1151, 10 1998.

[4] M. Bloodgood and K. Vijay-Shanker. A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 39–47. Association for Computational Linguistics, 2009.

[5] L. Bottou. Stochastic learning. In O. Bousquet and U. von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer, 2004.

[6] K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10(3):273–304, 1995.

[7] D.A. Cohn. Neural network exploration using optimal experiment design. *Neural Networks*, 9(6):1071–1083, 1996.

[8] S. Dasgupta, D. Hsu, and C. Monteleoni. A general agnostic active learning algorithm. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 353–360. MIT Press, 2008.

[9] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic theory of pattern recognition*. Springer, 1996.

[10] J. Dongarra and F. Sullivan. Guest editors' introduction: The top 10 algorithms. *Computing in Science and Engineering*, 2:22–23, 2000.

[11] A. Doucet and A.M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later. In *The Oxford Handbook of Nonlinear Filtering*. Oxford University Press, 2011.

[12] A. Frank and A. Asuncion. UCI machine learning repository, 2010. http://archive.ics.uci.edu/ml [accessed April 6, 2012].

[13] W.R. Gilks and C. Berzuini. Following a moving target—Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146, 2001.

[14] A.B. Goldberg, X. Zhu, A. Furger, and J.M. Xu. OASIS: Online active semi-supervised learning. 2011.

[15] D. Golovin, A. Krause, and D. Ray. Near-optimal Bayesian active learning with noisy observations. 2010.

[16] I. Guyon, G. Cawley, G. Dror, and V. Lemaire. Results of the active learning challenge. 2011.

[17] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[18] D.M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004.

[19] J. He, M. Li, H.J. Zhang, H. Tong, and C. Zhang. Mean version space: a new active learning method for content-based image retrieval. In *Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval*, pages 15–22, 2004.

[20] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[21] C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[22] Y. LeCun and C. Cortes. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist [accessed April 6, 2012].

[23] D.D. Lewis and W.A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval*, pages 3–12. Springer, 1994.

[24] M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine learning*, 54(2):125–152, 2004.

[25] K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate analysis*. Academic Press, 1979.

[26] G. Marsaglia. Ratios of normal variables and ratios of sums of uniform variables. *Journal of the American Statistical Association*, 60(309):193–204, 1965.

[27] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087, 1953.

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

[29] T.M. Mitchell. *Machine learning*. WCB/McGraw-Hill, 1997.

[30] R. Nowak. Generalized binary search. In *Proceedings of the 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 568–574, 2008.

[31] R. Nowak. Noisy generalized binary search. *Advances in neural information processing systems*, 22:1366–1374, 2009.

[32] F. Olsson and K. Tomanek. An intrinsic stopping criterion for committee-based active learning. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 138–146. Association for Computational Linguistics, 2009.

[33] G. Ridgeway and D. Madigan. A sequential Monte Carlo method for Bayesian analysis of massive datasets. *Data Mining and Knowledge Discovery*, 7(3):301–319, 2003.

[34] A.I. Schein and L.H. Ungar. Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265, 2007.

[35] A. Schrijver. *Combinatorial optimization*. Springer, 2003.

[36] M.R. Segal. Machine learning benchmarks and random forest regression. 2004.

[37] B. Settles. Active learning literature survey. Technical report, 2010.

[38] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1070–1079. Association for Computational Linguistics, 2008.

[39] H.S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 287–294, 1992.

[40] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[41] H. Shi. Best-first decision tree learning. Master's thesis, 2007.

[42] S. Tong. *Active learning: theory and applications*. PhD thesis, Stanford University, 2001.

[43] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2001.

[44] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning: training object detectors with crawled data and crowds. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 1449–1456, 2011.

[45] A. Vlachos. A stopping criterion for active learning. *Computer Speech & Language*, 22(3):295–312, 2008.

[46] R.E. Walpole, R.H. Myers, S.L. Myers, and K. Ye. *Probability & Statistics for Engineers & Scientists*. Pearson, eighth edition, 2007.

[47] Z. Wang, K. Crammer, and S. Vucetic. Multi-class Pegasos on a budget. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[48] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer, 2004.