

MASTERARBEIT

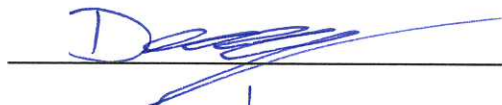
AN EFFICIENT PROBABILISTIC ONLINE CLASSIFICATION APPROACH FOR OBJECT RECOGNITION WITH RANDOM FORESTS

Freigabe:

Der Bearbeiter:

Unterschriften

Maximilian Denninger



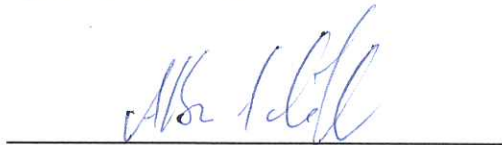
Betreuer:

Dr. Rudolph Triebel



Der Institutsdirektor

Dr. Alin Albu-Schäffer



Dieser Bericht enthält 93 Seiten, 23 Abbildungen und 9 Tabellen



Robotics, Cognition, Intelligence
(Int. Master's Program)

Technische Universität München

Master's Thesis

An efficient probabilistic online classification approach
for object recognition with random forests

Ein effizienter probabilistischer online
Klassifizierungsansatz für die Objekterkennung mittels
zufälliger unkorrelierter Entscheidungsbäume

Author:	Maximilian Denninger
1 st examiner:	Prof. Dr. Rudolph Triebel
Thesis handed in on:	April 12, 2017

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, April 12, 2017, 2016

Maximilian Denninger

Acknowledgments

I would like to thank my supervisor and professor Dr. Rudolph Triebel, who helped me through my whole master thesis. Furthermore for the possibility to research the topics Random Forest and Gaussian Processes at the German Aerospace Center (DLR) at the Institute of Robotics and Mechatronics in the department of Perception and Cognition. With his help throughout my thesis, we were able to make a valuable contribution for online learning on big data sets.

Furthermore I want to thank my colleagues at German Aerospace Center for their support, the pleasant atmosphere and the interesting discussions we had.

Abstract

Online learning on big data sets is still an open problem in the classification of images. Many problems in the real world don't have all data available in the beginning of the training. Therefore it is necessary that the approach is able to integrate new incoming data points. Random Forest have been proven to be good in online learning [SLS⁺09]. However the existing approaches do only generate very few trees, which only have a height of five. To overcome this shortcoming this thesis presents several methods to improve the generation of Decision trees, which leads to an algorithm, which can train thousands of tree with a sufficient height. Furthermore the Random Forest were then used in combination with an online sparse Gaussian Process to classify the outliers. These falsely classified points weren't classified correctly by the Random Forest in the first place. This whole approach was then optimized and tested on different datasets. The far most important result was that the presented online approach always yields better results than the offline approach, which is a remarkable result for an online learning approach. Furthermore we outperformed the result from Saffari *et al.* on the USPS dataset [Hul94, SLS⁺09].

Contents

Acknowledgements	v
Abstract	vii
1 Introduction	1
1.1 Classification	1
1.2 Related Work	2
1.3 Motivation and goal	6
2 Decision Trees	7
2.1 Introduction	7
2.2 Training	8
2.3 Metrics	8
2.3.1 Misclassification rate	8
2.3.2 Gini impurity	9
2.3.3 Information gain	9
3 Random Forest	11
3.1 Fast implementation of binary trees	11
3.1.1 Memory efficient	12
3.2 Advantages and disadvantages of decision trees	14
3.3 Deep decision trees	17
3.4 Online learning	20
4 Gaussian Processes	23
4.1 Bayesian Linear Regression	23
4.2 Gaussian Processes for Regression	25
4.3 Gaussian Processes for Classification	27
4.4 Kernels for Gaussian Processes	30
4.4.1 Gaussian kernel	30
4.4.2 Gaussian kernel with expanded length	31
4.4.3 Random Forest kernel	32
4.5 Informative Vector Machine	35
4.6 Assumed-density filtering	36
4.7 Expectation Propagation	38

4.8	Informative Vector Machine as Extension of EP	41
4.9	Hyperparameter optimization for the Gaussian Kernel	44
4.9.1	CMA-ES	46
4.9.2	CMA-ES for hyperparameter optimization	47
4.10	Enhanced error measurement	48
4.11	Improved selection of the active set	51
4.12	ADF, EP and the active set	53
4.13	Multiclass scenario	55
5	Online Random Forest with Informative Vector Machines	57
5.1	Online learning	59
5.2	Biggest challenge	59
6	Implementation Details	61
6.1	Online learning	61
6.2	Thread Master	61
6.2.1	Multithreading	63
7	Examples	65
7.1	MNIST	65
7.1.1	Results for the Random Forest	66
7.1.2	Results of the Informative Vector Machine	68
7.1.3	Results for the Random Forest with the Informative Vector Machine	70
7.2	USPS	71
7.2.1	Random Forest Results	72
7.2.2	Results of the Informative Vector Machine	73
7.3	Washington	73
7.3.1	Result for the Random Forest and the Informative Vector Machine .	74
8	Conclusion and Future Work	77
8.1	Conclusion	77
8.2	Future Work	78
	Bibliography	79

1 Introduction

In the last decades machine learning got more and more attention through the growing computational power. Many different approaches were developed and presented to the world. One of the main tasks in modern machine learning is classification, in which the class of a new data point is estimated. Several solutions have been proposed for that, which all use different angles to solve this problem. One of them is called Gaussian Processes, it is a powerful tool for classification and regression problems. The biggest advantages of them are their non-linear and non-parametric behavior and their explicit probabilistic formulation of the model, which compares to the learning of an estimation, like in the most other approaches. However their disadvantage is that the computational time scales cubically with the amount of data points [RW06, Bis06, Mur12].

This thesis mainly builds up on the work of Fröhlich *et al.*, which combined a Gaussian Process a the Random Forest [FRKD12]. By doing this the main disadvantage of Gaussian Processes is reduced, which makes them usable for big data sets. However a lot of problems in the area of mobile robotic do not have all data points in the beginning and an online approach is therefore necessary. In the most cases just a fraction of the whole data set is available at the start and the algorithm should already be able to predict unknown data points. One further advantage of Gaussian Process is their expressiveness on small data sets, which is important in this thesis.

So the core idea is to generate an online approach, which first splits a given data set into sub splits with the help of a Random Forest, which are then further used by an online Gaussian Process. The main challenge here is a fast and online approach, which combines the Random Forest with the Gaussian Processes. Random Forest on their own have great properties, like their fast training and prediction time. Furthermore they are without any extension already useable for multiclass scenarios.

1.1 Classification

The main task of this thesis is classification. In classification the main goal is to determine to which class a data point belongs to. In order to that trainings samples are given, which provide the underlying algorithm, with enough knowledge to make an informed decision. In figure 1.1 two classes are given, one is plotted in blue the other one in yellow. This simple data set will be later used to explain the different approaches tested in this thesis. The goal now is identifying the class of a new unseen data point, which does not belong to the trainings set. Important to note is that all approaches uses some sort of similarity

measurement to determine the class of an unseen data point. For example an unclassified point in the upper right corner of figure 1.1, would be classified as yellow by a human. The reason for that is that the human classification in this case would work by proximity, which is similar to the so called nearest neighbor algorithm, which is not further elaborated. In this thesis Random Forest and Gaussian Processes are used to classify unknown data points.

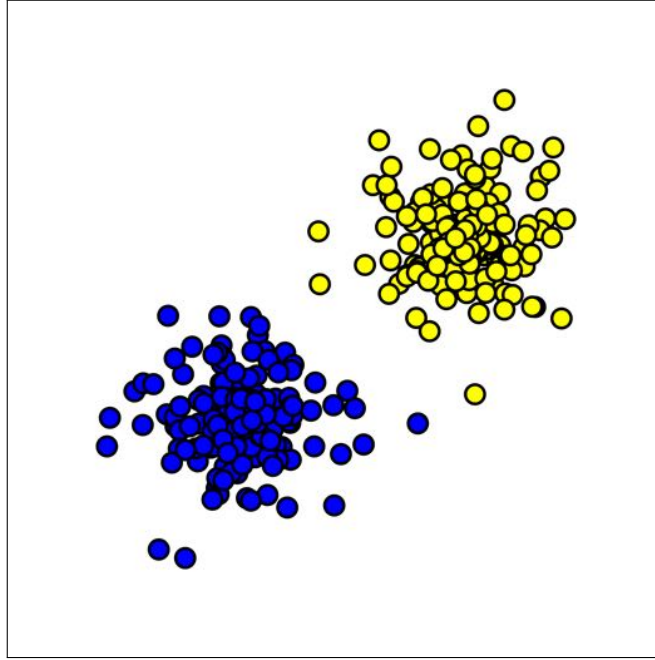


Figure 1.1: A simple dataset with two classes, which are plotted with little dots. Each dot represents a 2D data point and the color determines to which class the point belongs.

1.2 Related Work

As mentioned before the idea of combining Random Forest with Gaussian Process was presented in the paper "Large-scale Gaussian process classification using random decision forests" by Fröhlich *et al.* [FRKD12]. This idea is close to the approach of Chang *et al.*, which uses Support Vector Machines instead of Gaussian Processes in their paper "Tree Decomposition for Large-Scale SVM Problems" [CGLL10]. The combination of two methods lead to two different related work parts, which are covered here. First the Random Forest is covered and afterwards the Gaussian Process.

Random Forest

One of the first comprehensive books about Random Forest was "Classification and regression trees" by Breiman [BFSO84]. It introduced the concept of using splits to separate data sets and let the group of trees vote for the best result. Breiman also introduced the concept of bagging in the article "Bagging predictors", where he described how randomly selecting a subset of the data improves the predictive result [Bre96]. In his article "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization" Dietterich combined the Breiman's idea with a random selection of the split criteria [Die00]. This improves the decision in each node and therefore the whole tree, which is one of the easiest ways of improving Decision trees. In 2001 Breiman introduced in his paper "Random Forest", the concept behind modern Random Forests, where he combined several approaches like the random split criteria and the bagging concept [Bre01]. These core ideas are used in this thesis, too. Based on that Geurts *et al.* have developed the "Extremely randomized trees", which are published in the paper with the same name [GEW06]. These trees select the splits partially or totally random, so there is not necessarily a connection between the input data and the resulting tree. This idea was routed in the problem of high variance in the splits, which solely depended on the training points. This was compared and evaluate in "Investigation and reduction of discretization variance in decision tree induction" by Geurts and Wehenkel [GW00]. The work "An empirical evaluation of supervised learning in high dimensions" from Caruana *et al.* showed that Random Forest compared to other approaches like neuronal nets, boosted trees and support vector machines are in better in handling high dimensional problems [CKY08]. The high dimensionality is one of the main goals of this thesis and therefore this approach could not be used.

In the work "On-line Random Forest" from Saffari *et al.* an online Random Forest approach is presented, which tried to unlearn bad selections in the trained trees [SLS⁺09]. This differs to this thesis in the point that a Decision tree is discarded if a tree does not represent the data well enough. This is possible through the fast training times of our approach. Lakshminarayanan *et al.* presented in their paper "Mondrian Forests: Efficient On-line Random Forests" a method of using Random Forest with Mondrian trees. Mondrian trees are in general more expressive than simple Decision trees, however their handling of high dimensional data is worse, because the trainings time depends on the amount of dimensions. These trees were not used here, because of that [LRT14].

Gaussian Process

The standard Gaussian Process evolves naturally from the Bayesian Linear Regression and is therefore best suited for a regression setting. This was shown in the paper "Gaussian Process for Regression" from Williams and Rasmussen [WR96]. Nonetheless with a few changes the Gaussian Process can be used for the classification. However these changes have the disadvantage of removing the analytical solution for the problem, which means

an approximation has to be made [RW06]. This leads to an algorithm, which is quadratic in space complexity and cubic in time complexity so that the training for big data sets is not possible with the standard approach.

In order to overcome this several methods were designed, which are able to find a solution for classification on big data sets with Gaussian Processes. The paper "Sparse Online Gaussian Processes" by Csató and Opper presents an algorithm, including basis vectors from the data set, if the change on the sample averaged posterior mean due to the sparse approximation is maximal. This can be done in one iteration over the data and furthermore the amount of points can be controlled by removing the points again, which had the smallest change [CO02]. However the paper does not over an optimization of the hyperparameters nor is the resulting approximation good enough for high dimensional problems. For regression Tipping introduced the Relevance Vector Machine it is a probabilistic model on the Support Vector Machine, which forms therefore a Gaussian Process. This was shown in the paper "Sparse Bayesian Learning and the Relevance Vector Machine" [Tip01]. But the predictive distribution of their approach suffers from a unreasonable predictive distribution, which can be healed with the approach presented in "Healing the relevance vector machine by augmentation" from Rasmussen and Quiñero-Candela [RQC05]. A different approach is the "Sparse greedy Gaussian Process Regression", which finds a subset in several iteration over the data. This procedure is more expensive than the Relevance Vector Machine, but yields better results [SB⁺01]. Lawrence *et al.* presented the Informative Vector Machine in the paper "Fast sparse Gaussian process methods: The informative vector machine" [LSH⁺03]. In it an algorithm is described, which is able to find a subset of the data, which represents the data best and on it a Gaussian Process is calculated. This can be done iteratively in combination with the approximation of the posterior, which makes it faster than the most other approaches. This thesis uses an Informative Vector Machine, with some changes to perform better with fewer points in the selected subset. The main reason for selecting this approach is the fast calculation of the active set and the approximation for the posterior at the same time. In 2006 Snelson and Gahramani showed in their paper "Sparse Gaussian Process using Pseudo-inputs" an approach, where new points were estimated, which carry the Gaussian Process for a regression scenario. But the pseudo-inputs always have an initial sampling area in which they most likely occur, which can be a problem in multi dimensional problems, where the area of interest is unknown [SG06]. To make it work on higher dimensional data sets they had to use a Principal Component Analyse to find a good representation in a lower dimension. An alternative to the PCA is Automatic Relevance Detection. Naish-Guzman and Holden used the approach from Snelson and Gahramani in a classification scenario in their paper "The Generalized FITC Approximation" . They draw the points randomly from the approximated posterior [NGH07]. This lead to a very sparse approximation with only few points, however the computational effort is high and in combination with the estimation of the hyperparameters of the kernel too unstable for higher dimensional problems. In 2009 the article "Variational Learning of Inducing Variables in Sparse Gaussian Processes" from Titsias

was published. His approach minimizes the KL-divergence between the true posterior and an approximated one. This makes it possible to optimize the hyperparameters and the inducing points at the same time [Tit09]. But this whole approach only works for regression and not for classification. Vanhatalo and Vehtari presented a speed up for the general Gaussian Process. Their paper "Speeding up the binary Gaussian process classification" showed that by using a covariance function instead of the full covariance matrix the inference and the memory requirements could be reduced. However the algorithm still scales $O(n^3)$, which does not fully solve the problem for big data sets [VV12].

The paper "Gaussian Process for Big Data" by Hensman *et al.* uses a sparse representation of the input points. They improve the selected points in comparison to the Informative Vector Machine by blurring them with a Gaussian, using a full covariance matrix for the points around that basis point. That makes it possible to summarize local data manifold information with fewer data points [HFL13]. However the computational effort is much higher and the big advantage is the usage on bigger data sets, which is not necessary here, because the Random Forest already splits the data set beforehand. Seeger *et al.* presented in their paper "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression" a method to quickly find an active set for representing a given data set. Their approach only needs $O(n)$ steps to find the next point to include in the data. So they use an approximation for finding the best active set [SWL03]. Nonetheless this method can not be used here, because it is not applicable to a classification setting.

The paper "Gaussian process training with input noise" from McHutchon and Rasmussen introduces the NIGP, this approach can work on noise input data with noise free output data, which is the opposite to the usual assumption. Furthermore their approach can estimate the noise variance alongside the hyperparameters [MR11]. Based on that Bijl *et al.* showed in their paper "Online Sparse Gaussian Process Training with Input Noise" a method for generating a Gaussian Process online on stochastic noisy input data. Their approach can include additional measurements very efficiently, where each incorporation only takes constant runtime [BSvWV16]. Nevertheless there is no obvious way of transforming their approach to the problem of classification. The presented methods have all strong and weak spots. However the main theme is if the approach is fast and stable in regression, there is no clear way of transforming it to classification. On the hand the classification approaches all suffer a stable hyperparameter optimization or can not deal with high dimensional problems. Furthermore using them in an online fashion is in the most case not possible. Therefore in this thesis the Informative Vector Machine from Lawrence *et al.* was used. It offers an online approach, which can be trained fast on different problems and uses the data points as subset for the active set. This reduces the complexity of the problem. However a bigger amount of induced points has to be used. The big advantage of that is that there is no dependency on the amount of dimensions.

1.3 Motivation and goal

Several approaches have been presented in the past, which already offer different solutions for classification of images. However these approaches are in general offline and the computation time is not an important factor. Furthermore the most approaches only build up a model, which estimates the underlying dataset without any probabilistic reasoning. Gaussian Processes can calculate a probabilistic model for a data set and make their prediction on it.

Therefore this thesis tries to build up a framework for fast online learning for classification. In order to do that an Online Random Forest should be designed, which is able to work in an online setting. So a new library has to be designed, which supports online learning on big data sets and can further on use a probabilistic model for the reasoning. So these Online Random Forest are combined with the Informative Vector Machine, which is a sparse Gaussian Process. Additionally the learning should use all system resources as best as possible. The overall goal is that the Random Forest and the Informative Vector Machine can be trained in parallel to improve the global result. The implementation should always favor the worst classifier, so that the overall solution is improved as fast as possible.

2 Decision Trees

2.1 Introduction

Decision Trees are simple classifier, which deduces the class of an unlabeled data point by assigning the point to a part of the split training set. This part should be as pure as possible, that means that all elements in this part should be from the same class. The point will then get the class with the majority of points in the split training set. This can be achieved by splitting the training data at each node along of the axis in a binary tree until the purity of each node is high enough or the maximal amount of layers is reached. Figure 2.1 shows the result for data set from the section 1.1. The Decision tree used in this figure has overfitted and all the test and trainings data points are classified correctly. However the yellow stripe on the right side of the picture only fulfills the purpose of covering only one point and the area between the point and the main mass of yellow points is blue again. This shows that the algorithm just tried to fit the points and does not model the data correctly. Still all points were classified correctly, but the generalization is not optimal.

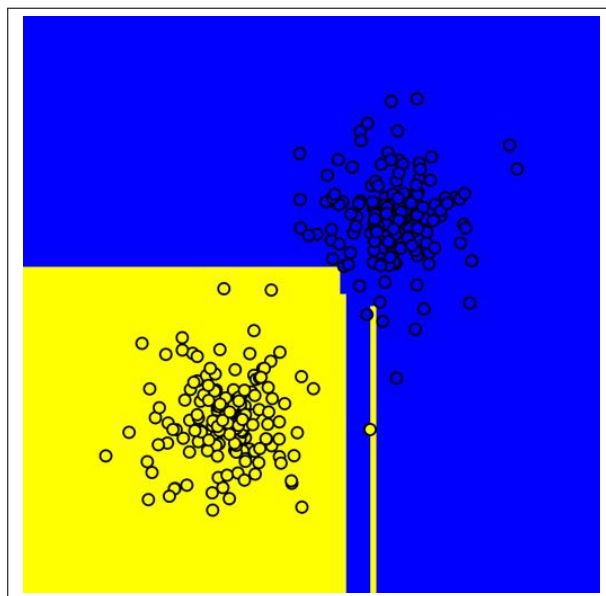


Figure 2.1: This figure shows the result on the data set from section 1.1. The background in this picture indicates the membership of new points at this location. This shows that both of the datasets were correctly classified.

2.2 Training

Labeled Data is usually used to train decision trees and they are therefore considered as a supervised learning approach. The training follows a greedy approach so in each node always the best split is performed. These splits depends on the selected metric. A metric calculates the quality of a split on a training set. Before such a metric is evaluated, a split dimension and a split value has to be determined, these values are usually randomly drawn. First a random dimension is selected out of the possible dimension of the input data. After that a random split value is selected depending on the minimum and the maximum value provided in this specific dimension of the data in this node. This reduction of the minimum and maximum values for each node leads to better performing trees, because a split has a higher chance of splitting the data points. If a dimension minimum and maximum values have gotten equal than this dimension will be omitted and a new one will be drawn, because that means in this dimension no information can be gained. A decision tree consisting out of decision nodes is depicted in figure 2.2. In this figure each decision node has a split value and a split dimension. Each leaf in the end represents a region of the input space. These are applied on an input point. If the expression is true the left child is selected else the right child will be the way to go down the tree. This is performed until a leaf is reached, in which the input point gets the class probabilities based on the trainings data in this leaf. Equation 2.1 contains the formula to calculate the probability for an input value being of a certain class z , which is based on the amount of training points $\mathcal{D}_{\mathcal{R}}$ in a specified region \mathcal{R} . Each leaf of a tree has its on region \mathcal{R} in the data \mathcal{D} . The sum of the equation results in the amount of points in \mathcal{R} , which belong to the class z . The resulting class label y is the one with the highest probability $y = \operatorname{argmax}_c p(z = c \mid \mathcal{R})$.

$$\pi_{c,\mathcal{R}} = p(z = c \mid \mathcal{R}) = \frac{1}{\|\mathcal{D}_{\mathcal{R}}\|} \sum_{i \in \mathcal{D}_{\mathcal{R}}} \mathbb{1}(z_i = c) \quad (2.1)$$

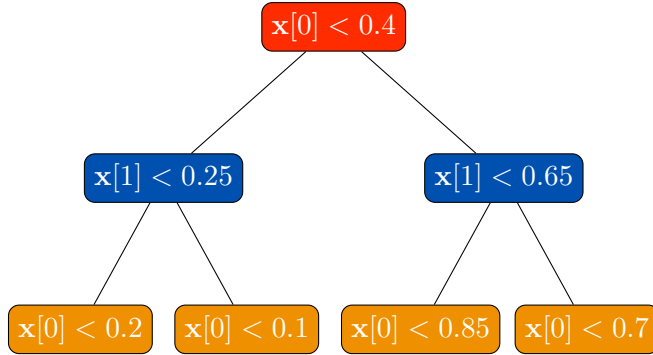
2.3 Metrics

There are different metrics to evaluate a split in a decision node. A metric should always has its highest value if all classes are distributed equally and the lowest value if only one class is left. These metrics are then minimized so that in the end only one class is left. Three metrics are explained in this thesis and all of them are compared in a binary case scenario in figure 2.3.

2.3.1 Misclassification rate

The simplest of all metrics is the misclassification rate. It gives the amount of misclassified labels in a decision node. Equation 2.2 calculates the error for each data subset $\mathcal{D}_{\mathcal{R}}$. It is the sum of misclassified points divided by the total amount of points in $\mathcal{D}_{\mathcal{R}}$. The misclassification rate is depicted as a red line in figure 2.3.

Decision tree:



Divided input space:

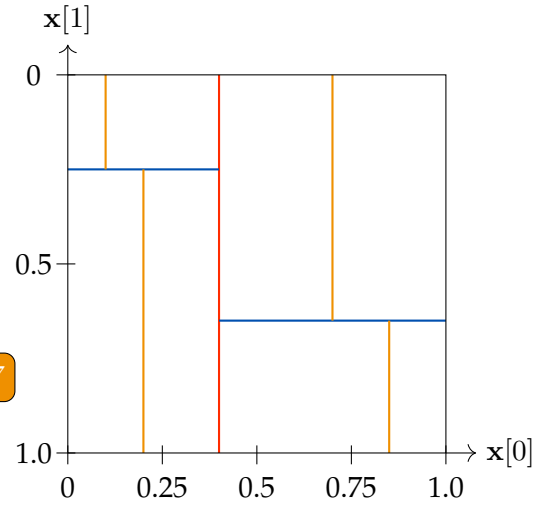


Figure 2.2: On the left is a pretrained decision tree with a height of three depicted. In each node the data point is checked against the split value in a certain split dimension. In the root node the input value is checked against 0.4 in the first dimension, depending on the value of the input one of the children is chosen. If the value in the first dimension is smaller the left child is selected. This process is repeated for all nodes. The right side of the figure shows the input space split by the decision tree, each nodes adds a new split line and the colors of the nodes correspond to the colors of the splits lines.

$$err_{\mathcal{D}_{\mathcal{R}}} = \frac{1}{\|\mathcal{D}_{\mathcal{R}}\|} \sum_{i \in \mathcal{D}_{\mathcal{R}}} \mathbb{1}(y_i \neq z_i) \quad (2.2)$$

2.3.2 Gini impurity

An alternative to the misclassification rate is the gini impurity, which is the sum of the probability for a class times the counter probability for the same class, see equation 2.3. It has its maximum if all classes are distributed equally and it is the blue curve in figure 2.3. Therefore minimizing it leads to a good separation.

$$Gini(\pi) = \sum_{c=1}^C \pi_c \cdot (1 - \pi_c) \quad (2.3)$$

2.3.3 Information gain

A third metric can be derived from the theory of information gain. The main idea behind this is that an unlikely datapoint gives more information than a more likely one. Based on that the entropy is defined as the average amount of information needed to specify the

state of a random variable. That means a uniform distribution would have the highest entropy, because the average amount of information is maximized. On the other hand a pure dataset would have no entropy, because all elements are very likely and therefore have no information at all. Figure 2.3 shows the described behavior for a binary case, the green curve depicts the entropy. The equation 2.4 contains the entropy defined by Shannon [Sha48]. Important to note is that $\mathcal{H}(\pi_c)$ is zero for $\pi_c = 0$. So entropy can be used to make an informed decision for picking a good split value [Bis06]. Using the $\log \log_C$ with a base of C , would give the normalized entropy, which would have at 0.5 a value of 1. However the entropy would be divided by a constant factor in the binary case this would be $\log(2)$, which would not change the result of the optimization. Therefore it is omitted in the implementation of the thesis.

$$\mathcal{H}(\pi) = - \sum_{c=1}^C \pi_c \cdot \log \pi_c \quad (2.4)$$

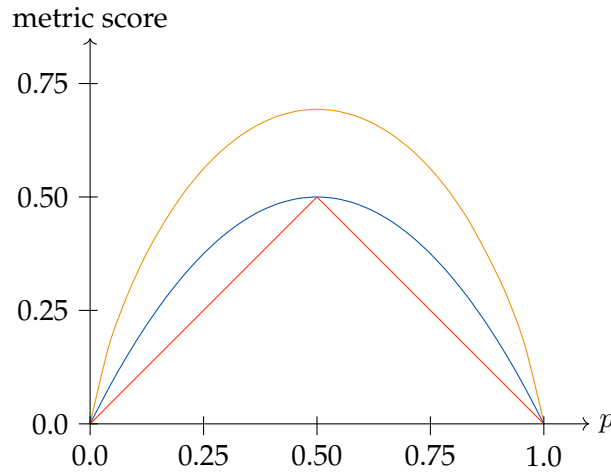


Figure 2.3: The red line represents the misclassification rate, the blue curve the gini impurity and the green curve the entropy calculated based on the information gain theory. All of them are computed in a binary example, where the probability for the first class is p and for the second class is $(1.0 - p)$.

3 Random Forest

A Random Forest is the combination of several Decision Trees. This has several advantages in comparison to one single tree. In order to understand the advantages of a Random Forest, the disadvantages of a Decision Tree have to be elaborated. The four most important disadvantages are:

1. Decision Trees tend to overfit, which requires expensive pruning operations
2. Dealing with uneven sized datasets is problematic
3. Instability of the result, which means that minor noise in the data changes the result
4. Classification plateaus, because of the splitting along the axis

All these disadvantages can be eradicated by using Random Forest. Through the combination of more than one Decision Tree the most of these flaws can be reduced or eradicated. Figure 3.1 shows the result for a Random Forest trained on the data set presented in section 1.1. Both data sets are well presented by the Random Forest and all points are classified correctly. The grey areas in the left upper and right lower corner show that there are places, where the algorithm can not decide to which class a point belongs to.

A Random Forest combines the results of the Decision Trees in a democratic fashion, that means each tree can vote for a class. In the end the class with the most votes wins. Furthermore the amount of votes can be used as a certainty measurement. This whole procedure is explained in more detail in the end of this chapter.

Through the combination of trees pruning is not longer necessary, because the overfitting of the trees get eradicated through the different selections in the trees. The same holds for the problem with the uneven data set, with the aid of many overfit trees, even small parts of the data can be represented in the model. In a not overfitted tree, this data might not get represented at all. So all of the mentioned disadvantages get eradicated by using a Random Forest. However instead of training just one tree a lot of trees have to be trained, which is more computational effort and furthermore needs more memory space. The next sections will provide some solutions for this.

3.1 Fast implementation of binary trees

In order to improve the training time of the Random Forest the single Decision trees are improved in this section. The reason for this is that the trainings and prediction time of the Random Forest mainly depends on the amount of Decision trees and their own training and prediction time.

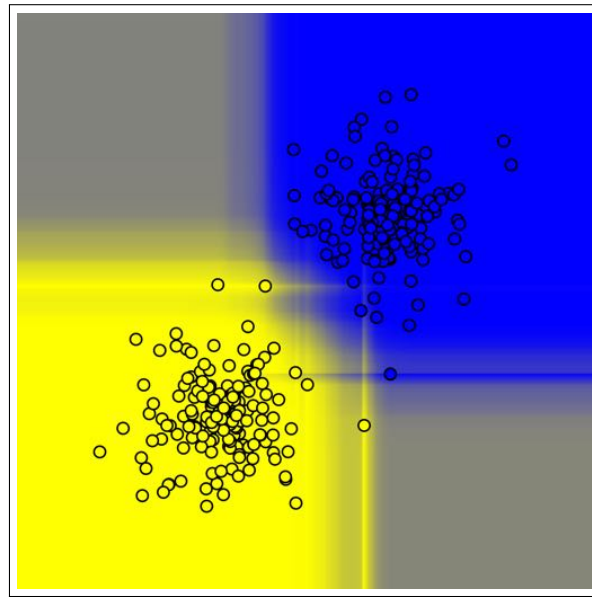


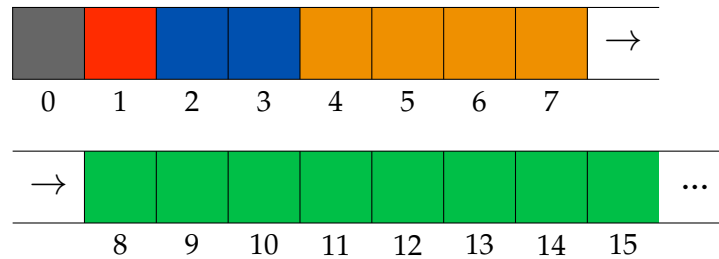
Figure 3.1: The result of a Random Forest on the data set from section 1.1 is shown in this picture. The background indicates the membership of a new point at this position. The grey areas show that there are areas, which do not belong to any of the classes.

3.1.1 Memory efficient

Trees are usually implemented with pointers. Each node of the tree gets a pointer to their two children and to its root. Therefore in each node of the tree three pointers are needed. This approach has the advantage that the resulting trees do not have to be balanced. However the goal of decision trees is that they are as much balanced as possible, because that means that the data is well distributed over the leaves. For this reason using pointers only increases the amount of needed space in the memory.

There is an alternative for storing binary trees in the memory. Instead of storing each element as a single object the information of the decision node can be stored in an array. Each node in the tree can be accessed by its index. In this arrays the root is saved in the element with the index 1. The index for the left child can be calculated by multiplying the index of the root with two, which can be implemented by a bitshift to the left. The right child is always the next element in the memory ergo multiplying by two and adding one gives the index of the right child. This is especially useful, because the used decision nodes only consist out of the split value and the dimension in which the split will be performed. So only two arrays per decision tree have to be saved one for the split values, which are saved as doubles. The other one is used for the split dimension, which are saved as unsigned integers. Each element in the array represents a node in the tree. Furthermore an array with the amount of leaves saves the winner in each leaf, to speed up the prediction step. It has the same length than the other arrays and is filled with unsigned integers.

Array:



Tree:

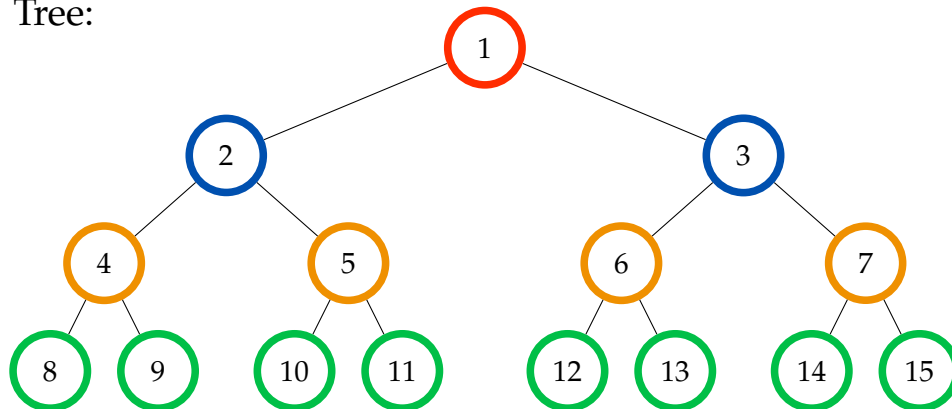


Figure 3.2: The array at the top represents how a binary tree is saved in the memory. The element with index one contains the root and multiplying the index with two leads to the left child adding one to the result gives the right child. These rules hold for every node in the tree. In addition it is the same for finding the father, but in this case the index is divided by two. The upper array was only split for presentation purposes.

This is illustrated in figure 3.2. The lower half of the image contains a binary Decision tree, where each node gets a number to identify it in the upper array. For both, the split dimension and the split values, an array is generated and the indices correspond to the numbers of the nodes in the tree shown in the picture.

Therefore a decision trees needs four bytes for the dimension value, eight bytes for the split values and four bytes for the winning class in each leaf, eg. a tree with a depth of eight needs $2^8 \cdot 16$ bytes, which are roughly 4 kB. In comparison a tree consisting of nodes connected by pointers would need 40 bytes per node and the same array to save the winning classes per leaf. So a node would then have twelve bytes for the data and additionally 24 bytes for the pointer on a 64 bit computer. A tree with a height of eight would then need 11.26 kB. In general using pointers in decision trees uses 2.75 times more

memory than using the array approach, compare therefore equation 3.1.

$$\underbrace{(2^{\text{height}} \cdot 44 + 40 \text{ bytes})}_{\text{pointer approach}} \div \underbrace{(2^{\text{height}} \cdot 16 + 40 \text{ bytes})}_{\text{array approach}} \approx 2.75 \quad (3.1)$$

The second approach was used throughout this thesis to save the amount of used memory and to speed up the accessing of nodes in the tree, because the position of each node in the tree can easily be calculated by bitshifts, which is much fast than following nodes through the tree. The reason for this is, how objects are stored in the memory, a vector always lies as one sequential object in the memory. This does not hold for individual node objects, which are just connected by pointer, which makes pages refreshes much more likely and therefore the speed of the algorithm is improved by this approach too. Furthermore the training in this array approach can be performed very easily by iterating over the arrays and filling the split values and split dimensions in. This form of training correspond to a breath first parsing of the tree, which does not need a recursive function call and therefore prevents a stack overflow. So each level of the tree can easily be processed by iterating over the array.

3.2 Advantages and disadvantages of decision trees

A advantage of decision trees is there fast training. The approach presented in this thesis is able to train many trees, without any problems and the limits are only set by the memory space available on the system. The amount of trained trees depend on two things first on the height of a tree and second on the amount of used data points, because both have a significant influence on the trainings process. At first the influence of the height is described. Figure 3.3 shows on the left side the dependency of the amount of trained trees

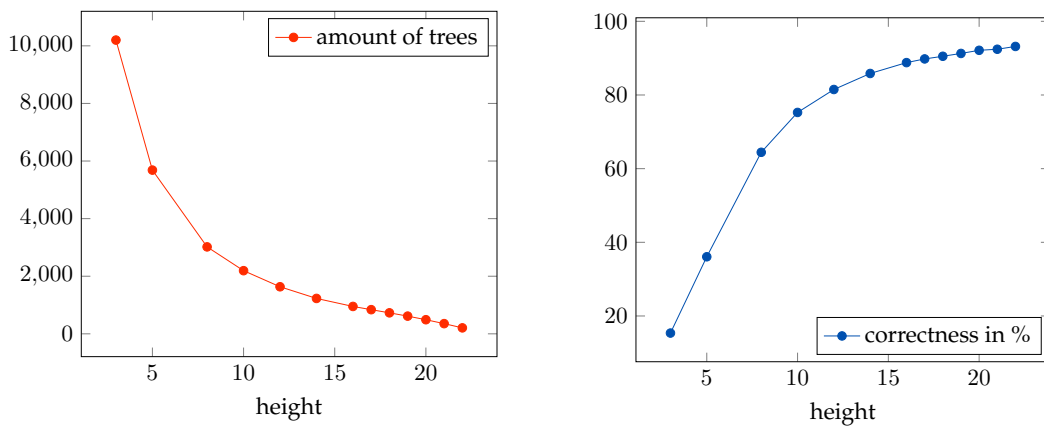


Figure 3.3: On the left the amount of trees generated in 30 seconds with different heights for roughly 30,000 points per tree. The right shows the correctness for the trained trees on the MNIST test set, which will be explained in section 7.1.

on the height of all trees.

These trees were trained in 30 seconds on the MNIST data set, which is described in section 7.1. It has 60,000 training data points, 10,000 test points and consists out of ten classes. Each tree was trained with roughly 30,000 randomly drawn points. The amount of trained trees reduces quadratically by increasing the height of the trees. This is caused, by the increase in amount of nodes, which have to be processed. The right term of equation 3.1 shows the memory spaces needed for a tree with a certain *height*. From that a tree with height of three needs 168 bytes compared to a tree with a height of 22, which needs 67.1 MB. That means by adding 11 layers, the memory consumption rises by a factor of 399,457.8. However the amount of trained trees is only reduced by the factor of 49.28, due to the optimized implementation of the trees and the fact that not all leaves of the tree will be processed in the end. Furthermore in figure 3.3 the amount of correctly classified points on the 10,000 points of the test set, is illustrated. From this plot the demand for deeper trees can be derived, because a deeper tree can more easily map the data points to classes. In particular as the amount of trees used for the prediction is even lower.

Secondly the influence of the amount of trainings point per Decision tree is evaluated. Raising the amount of training points decreases the amount of trained trees, which can be trained in 30 seconds. The effect of the amount of trainings point is depicted on the left in figure 3.4. Instead of changing the amount of points, the step size over the data is changed. A step size of 25 means that a random number between 1 and 25 is drawn and added to the last used element index. This procedure is repeated until the whole data set is processed. The test were performed on the MNIST data set explained in section 7.1. The left of figure 3.4 shows that the amount of used points does only marginally influence the number of

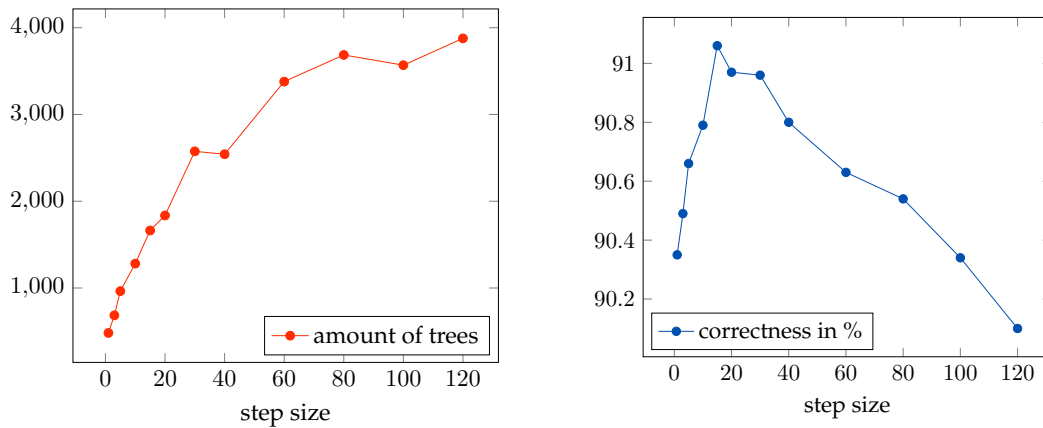


Figure 3.4: The left figure shows the amount of trees, which could be generated with different step sizes. On the right the correctness on the 10,000 points of the MNIST test data are plotted. This shows that the amount of trees, rises when the step size is higher. Furthermore the correctness isn't effected much by the change in the step size, because the prediction loss in one tree is covered by simply more trees.

correctly classified test points. Even with only 482 trees and a step size of one, the result is still similar to 3877 trees and a step size of 120, which decreases the amount of points from 30,000 to roughly 992 per tree. The reason for this is that the loss of information in one tree is covered by simply more trees. One major disadvantage of a decision tree is the problem of overfitting. That means that a tree will always try to perfectly represent the trainings data, which implies that the generalization is bad. There are several possible solutions to overcome this problem. Instead of always performing a split if possible, a split can only be performed if some extra criteria are met. One of this criteria could be that the amount of points in the resulting leaves must be bigger than a predefined threshold. An other option is pruning the tree, that means remove splits, which do not perform well on a validation set, which does not belong to the training set. However pruning is a very expensive operation and therefore not used in Random Forest. Furthermore the overfitting of one tree can be ignored, because of the amount of other available trees. This means a lot of overfitted tree have in combination a good generalization. That can be explained with the Condorcet's jury theorem [dC85]. The formula is stated in equation 3.2, where T is the amount of trees in the forest and p the probability that a tree is right. The sum starts at $t = 0.5T + 1$, because that is the minimal amount of trees necessary for getting the majority in a binary setting.

$$\mu = \sum_{t=0.5T+1}^T \left(\frac{T!}{(T-t)!t!} \right) (p)^t (1-p)^{N-t} \quad (3.2)$$

If all trees have a probability higher than 0.5, then the result of all trees μ is higher than the probability of the individual [dC85]. This means that for $T \rightarrow \infty$ the probability of μ gets to infinity. So increasing the amount of trees gives better results, except the probability of a tree being right is below 0.5 then the overall probability gets down. However getting in a binary example below 0.5 means the performance is worse than random guessing, which should already tell that the classifier does not work. This means in general the probability of a classifier being right must be higher than the one for a random guesser, which is defined by $\frac{1}{\text{classAmount}}$.

The approach in this thesis had a different focus than the existing approaches, like the one from Saffari *et al.*, where they only used a 100 trees with only a depth of five, which makes in total only 3200 Decision nodes. In comparison on the same problem this thesis used 23,600 Decision trees with a possible depth of 35, which makes in theory 810 trillion decision nodes [SLS⁺09]. So instead of getting the best possible tree this thesis tried to find as many good trees as possible. The reason for this is mentioned above, because more trees have a higher probability of getting a better result, even if they are in general worse than a few better trees.

3.3 Deep decision trees

The section 3.1.1 describes a efficient way of storing decision trees in the memory. However there is one problem with this approach. It is not well suited for trees which are very deep, because *e.g.* a deep tree with height of 23 already needs 134 MB. This is unpractically, because big portions of the tree are not used and therefore memory is allocated without any benefit. A tree with height 23 has 2^{23} leaves, which are roughly eight million leaves. in a scenario with 60,000 different training samples only 1 of 133 leaves would be filled, which is very inefficient. This can be solved by omitting a subtree, which is not used in a particular tree. Such a subtree can only be omitted if the layout of a tree is not arranged in an array.

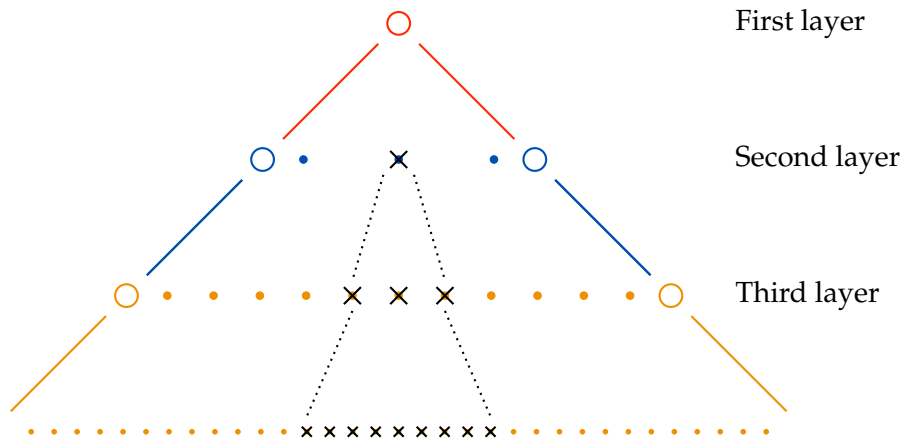


Figure 3.5: The layer layout used for the deep decision trees is depicted in this figure. The first layer has only one tree and splits the trainings data in subdata sets. The amount of subsets depends on the amount of leaves in the tree. For each subset, which can be split again a new tree is generated and used. For the middle tree in the second layer the splitting criteria wasn't fulfilled and no new tree was generated on the dataset. Furthermore all children from this tree were not used, which dramatically reduces the amount of used memory space.

In order to still get the advantages of the memory efficient implementation. A combination was performed in this thesis, by splitting a tree in different layers. Each layer then consists out of a bunch of trees, which are connected with the leaves of the upper trees. In the root only one tree is placed and this tree is trained first. After the training, the training set is split according to this Decision tree. For each split, which contains at least two different classes a new tree is generated, which gets as an input this split. This procedure is repeated for all resulting trees until the amount of needed layers is met or no data points are left. This separation can be seen in figure 3.5. Each layer has its own color and consists out of a bunch of trees, except the first layer, which only consist out of one Decision tree. The big advantage is that not all subtrees will be trained. The crossed out tree in the middle

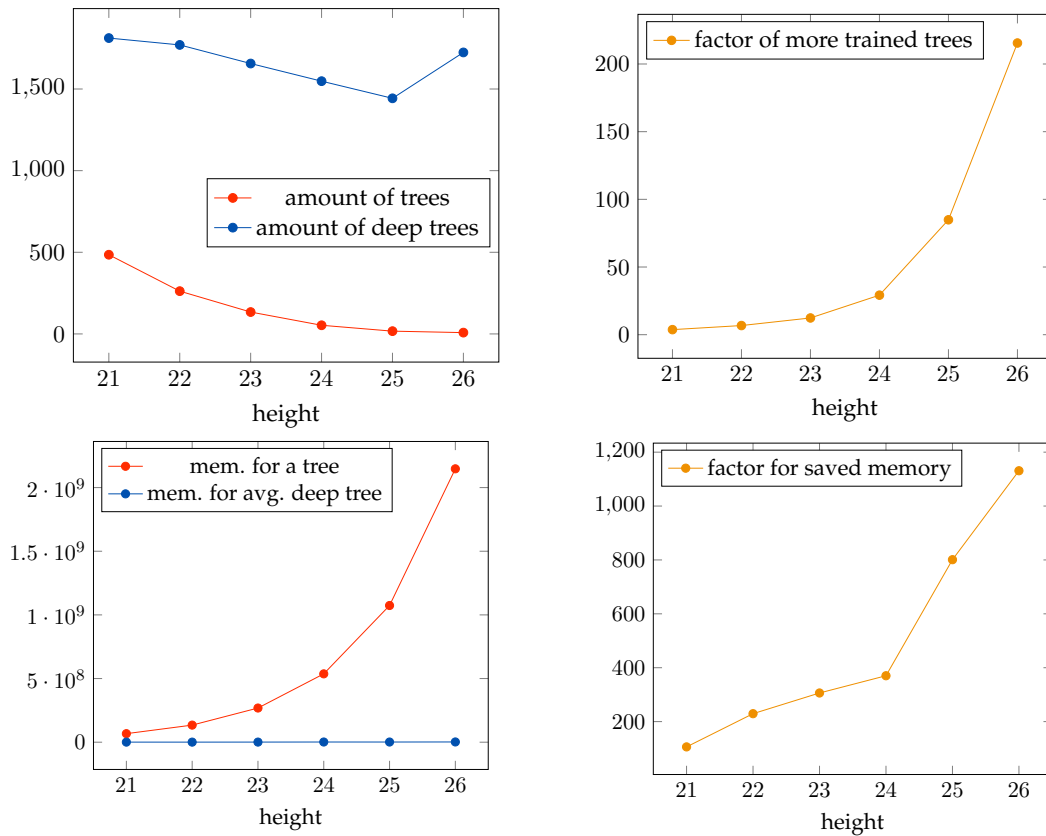


Figure 3.6: This figures show the behavior of the Deep Decision trees compared to the memory efficient approach. The left upper plots illustrates how many trees can be generated in 30 seconds on the MNIST data set. It shows that the amount, which can be generated in 30 seconds is much higher with the deep approach, because of the better usage of the memory. In the right upper corner the factor between the amounts is plotted. It shows that the amount grows nearly quadratically, because the memory requirements in the old approach increases quadratically. This can be seen in the lower plots, which show the memory requirements for a tree for this problem. The deep trees only have an average given, because the size depends on the used branches, which depends on the amount of splits and therefore directly on the data.

of the second layer in figure 3.5 wasn't generated, because there is only one class left. The important aspect here is that also all the children of the not used tree are not computed, which then reduces the amount of needed memory space.

In figure 3.6 four different plots show the advantages of the deep Decision trees compared with the approach from section 3.1.1. The used deep Decision trees uses three deep layers for all tests between 21 and 26. The left upper plot shows that the amount of trees, which can be generated on the MNIST dataset in 30 seconds, reduces if the height is increased. This was shown before in figure 3.3, which had the same overall conditions.

However in figure 3.6 the height is further increased from 21 to 26, which means 67,108,864 leaves are available in a tree with a height of 26. On the right side of this plot it is illustrated how much more trees could be trained in 30 seconds with the new approach. The only reason the comparison stops at the height 26 is that the creation of 8 trees in parallel already takes 38 seconds with a height of 26 in the old approach. To increase it further would increase the amount of time and memory. Furthermore the eight trees with a height of 26 need 8.5 GB of RAM, so generating more than 32 trees is a problem for modern computers.

The line for the amount of deep trees continues till a height of 45. In this settings a Random Forest with deep trees with a 45 height would be able to train 1626 trees. Where each tree has 2^{45} leaves, which are roughly 35.1 trillion leaves. In the non-deep approach the computer would need 562 TB of RAM for just one tree. But with the deep technique trees can be saved in just 504.2 MB and the training time was just 30 seconds. The graphic in the lower left corner shows how the memory for a tree of the non-deep technique compares to an average tree of the deep method. The blue line in the plot has nearly no slop in comparison, where as the red curve has a very steep slope, caused by the right part of equation 3.1. For the deep method only an average size can be given, because the size depends on the performed splits, which change the size of each tree. In the right lower corner the plot shows how much memory per tree can be saved for the deep decision trees. This shows that the deep method is useful and necessary if an appropriate height for a problem is chosen. A higher height results in better results, which was shown in section 3.2.

One problem, which now arises is how the amount of layers is chosen. This can be solved by training all possible options for 10 seconds and take the configuration which generates the most trees. This works, because the training of the trees depends on how much sub trees in each tree have to be generated. If the splitting of the data works well and the demand for new subtrees is low more trees can be generated, which reduces the overall memory usage and therefore grants a speed up. The current described approach saves all subtrees for a layer in an array, to guarantee a fast access. The problem with this is that if the tree gets sparse the amount of not used trees in this array get high, like depicted in the third layer in figure 3.5. This can be solved by replacing the arrays in the lower layers with maps, which only save the existing trees, that can reduces the amount of needed memory drastically, especially if the trees are very deep. For instance a tree with a height of 45 and a desired amount of layers of 9, would have a height of 5 per layer. This means that the last layer could have 2^{40} trees, saving them in an array would need roughly 8.80 TB for the last layer, which is not possible on a computer at the moment and furthermore a lot of the trees will not be used in the end. Therefore it is much better to use a map for the lower layers. The amount of layers and the amount of so called fast access layers must be defined before hand or can be determined like the described above. In this scenario each pair of configuration is compared to the others and the one, which was able to train the most trees in ten seconds is selected.

3.4 Online learning

Instead of using an offline approach, which already needs all the data at the beginning of the algorithm an online approach is presented here, which uses the data in a batch fashion. This means at the beginning of the training an offline step is performed like described in the sections before. After that an online update step is performed with a new unknown labeled data set. At first the new points get classified and all of them which are not correctly classified form the new update data set. This update set is then added to the existing training set. Afterwards an online update is performed, where all existing trees are first evaluated on the old and new update points. This evaluation is then used to sort all trees of the Random Forest after their performance. Then until the performance on the combined sets is good enough a new Decision tree is trained and tested on the combined set. If this new tree is better than the worst performing tree of the Random Forest it is added and the worst one is removed. If the new tree is not as good as the worst performing tree it is forgotten. This behavior is defined in algorithm 1.

Algorithm 1 The Online Random Forest update

Require: $\mathcal{D}, R =$ pretrained Random Forest, \mathcal{D}_{new}

- 1: $\mathcal{D}_{\text{all}} = \mathcal{D} + \mathcal{D}_{\text{new}}$
 - 2: $R_{\text{sorted}} = R.\text{sortTreesAfterPerformance}(\mathcal{D}_{\text{all}})$
 - 3: **while** not converged **do**
 - 4: $t_{\text{new}} = R.\text{trainNewTree}(\mathcal{D}_{\text{all}})$ ▷ Train a new tree t_{new} with \mathcal{D}_{all}
 - 5: $c = t_{\text{new}}.\text{predict}(\mathcal{D}_{\text{all}})$ ▷ Get performance of t_{new} on all points
 - ▷ If a new tree t_{new} is better than the worst tree of the sorted Random Forest then
 - 6: **if** $c > R_{\text{sorted}}.\text{getWorstPerformingTree}().\text{getCorrectness}()$ **then**
 - 7: $R_{\text{sorted}}.\text{removeWorstPerformingTree}()$ ▷ Replace worst tree with new tree
 - 8: $R_{\text{sorted}}.\text{addNewTree}(t_{\text{new}}, c)$ ▷ Add with insertion sort
-

By only adding the tree, which performs best on the combined sets. It is assured that the old and the new data set are presented in a good way. The online step is performed in this scenario after 200 points are added. However it could be done iteratively, this means each new point is evaluated by all trees. If the point is classified correctly and the certainty for the classification is high enough it is not added. But if one of the criteria is not full filled then the worst performing tree is selected and a new one is trained. This is possible, because the training for a single Decision tree is very fast.

Selecting the right approach depends on the problem, if there is more than a few milliseconds time, between the different data points then adding them one by one is possible. If the frequency is higher, the batch system will work better. With this the training can be adjusted to the frequency of the incoming points. Some results for this are shown in

section 7. The results there show that the online learning approach leads to better results than the offline approach, because only the good performing trees are kept and the bad ones are removed from the Random Forest.

4 Gaussian Processes

Modeling a probabilistic correlation between the data and the labels is the goal of Gaussian processes. In order to achieve this the Gaussian process posterior has to be calculated. It models the relation ship between the input value and the output values. The assumption is that similar inputs lead to similar results.

4.1 Bayesian Linear Regression

Before using Gaussian Processes for classification, the regression case has to be defined and this can be done over the bayesian linear regression. The equation 4.1 defines the regression for the estimated output values y . In this equation $\phi(\mathbf{x})$ is a vector of non linear basis function and \mathbf{w} is a vector of parameters, which specifies the strength of each non linear basis function. In this scenario a basis function is a Gaussian. The so called model complexity is the amount of parameters and basis functions M and states how many Gaussian functions are used to describe the underlying function.

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}) \quad (4.1)$$

To get a probabilistic model each parameter needs a probabilistic distribution, describing the spreading of the values. The Gaussian prior is here defined over the parameter \mathbf{w} as $p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{m}_0, \mathbf{S}_0)$, where \mathbf{m}_0 is the mean and \mathbf{S}_0 is the covariance. This can now be extended to equation 4.2 to make the treatment simpler. Here a zero-mean isotropic Gaussian with a single precision parameter α is used.

$$p(\mathbf{w} | \alpha) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I}) \quad (4.2)$$

The corresponding posterior distribution $p(\mathbf{w} | \mathbf{t}, \alpha, \beta)$ is then given by equation 4.5. Here the observed data value \mathbf{t} is defined by equation 4.3 and therefore β is the precision of the data noise.

$$\mathbf{t} = \mathbf{y} + \mathcal{N}(\mathbf{0}, \beta^{-1} \mathbf{I}) \quad (4.3)$$

From that the probability for getting the observed data from the real data is derived in equation 4.4.

$$p(\mathbf{t} | \mathbf{y}) = \mathcal{N}(\mathbf{t} | \mathbf{y}, \beta^{-1} \mathbf{I}) \quad (4.4)$$

The following equations 4.6 and 4.7 define the update rule for the mean and the covariance. These equations contain Φ , which is the design matrix, the elements are given by $\Phi_{nm} =$

$\phi_m(\mathbf{x}_n)$, that means Φ is a $N \times M$ matrix and each column consists out of the results from one specify basis function on each value of the data set.

$$p(\mathbf{w} \mid \mathbf{t}, \alpha, \beta) = \mathcal{N}(\mathbf{w} \mid \mathbf{m}_N, \mathbf{S}_N) \quad (4.5)$$

$$\mathbf{m}_N = \beta \mathbf{S}_N \Phi^T \mathbf{t} \quad (4.6)$$

$$\mathbf{S}_N^{-1} = \alpha \mathbf{I} + \beta \Phi^T \Phi \quad (4.7)$$

The posterior can now be calculated step-by-step. At first the prior is multiplied with the likelihood for the first data point $p(t \mid x, \mathbf{w})$, which describes how likely a regression value t for the first point x and the parameters \mathbf{w} is. After that a new data point is used and for it the likelihood is calculated and multiplied with the posterior from the last step, which is now the prior. This procedure is repeated until all points are processed.

In the end the posterior $p(\mathbf{w} \mid \mathbf{t}, \alpha, \beta)$ has a sharp variance, if the fitting was successful and the mean coincide with the mode and represents the optimal parameter set. However in practice it is more interesting to compute new predictions \mathbf{t}^* for new values \mathbf{x}^* then calculating the parameters \mathbf{w} .

$$p(\mathbf{t}^* \mid \mathbf{x}^*, \mathbf{t}, \alpha, \beta) = \int p(\mathbf{t}^* \mid \mathbf{w}, \mathbf{x}^*, \beta) p(\mathbf{w} \mid \mathbf{t}, \alpha, \beta) d\mathbf{w} \quad (4.8)$$

$$p(\mathbf{t}^* \mid \mathbf{x}^*, \mathbf{t}, \alpha, \beta) = \mathcal{N}(\mathbf{t}^* \mid \mathbf{m}_N^T \Phi(\mathbf{x}), \sigma_N^2(\mathbf{x})) \quad (4.9)$$

$$\sigma_N^2(\mathbf{x}) = \frac{1}{\beta} + \Phi(\mathbf{x})^T \mathbf{S}_N \Phi(\mathbf{x}) \quad (4.10)$$

In order to do that the predictive distribution has to be evaluated, which is defined in equation 4.8. The left part in equation 4.8 is the conditional distribution of the target variable and the right part is posterior weight distribution defined in equation 4.5. From that the two Gaussians can be convoluted, which results in equation 4.9 [Abr72]. In equation 4.10 the variance of the predictive distribution is depicted. The left term represents the noise on the data and the right term is the uncertainty of the parameters \mathbf{w} , because both Gaussians are independent the variance is additive. That means after adding $N \rightarrow \infty$ points the variance \mathbf{S}_N should go to zero and therefore the whole left term. So the whole noise will solely be generated by β^{-1} .

One problem this approach has is that for test points further away from the training points the confidence can be as low as β^{-1} . This is counterintuitive, because further away from the data the confidence should fall and not stay on the same level. This is caused by the used localized basis functions such as Gaussians. Further away from them reduces their influence and therefore only β^{-1} stays as variance. This can avoided if Gaussian Processes are used. In equation 4.11 the parameters \mathbf{w} , where replaced with the means calculated in equation 4.6, which leads to equation 4.12. Breaking that down for the indi-

viduals data points lead to equation 4.13.

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{m}_N^T \phi(\mathbf{x}) \quad (4.11)$$

$$= (\beta \mathbf{S}_N \Phi^T \mathbf{t})^T \phi(\mathbf{x}) = \beta \phi(\mathbf{x})^T \mathbf{S}_N \Phi^T \mathbf{t} \quad (4.12)$$

$$= \sum_{n=1}^N \beta \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}_n) t_n \quad (4.13)$$

That shows that at a point \mathbf{x} can be calculated by a linear combination of the training set variables t_n . This means it can be replaced by a function defined in equation 4.15, which is used in equation 4.14. This function is commonly known as kernel and can then replace the used basis functions, which is the core idea behind Gaussian Processes.

$$y(\mathbf{x}, \mathbf{m}_N) = \sum_{n=1}^N k(\mathbf{x}, \mathbf{x}_n) t_n \quad (4.14)$$

$$k(\mathbf{x}, \mathbf{x}') = \beta \phi(\mathbf{x})^T \mathbf{S}_N \phi(\mathbf{x}') \quad (4.15)$$

4.2 Gaussian Processes for Regression

Instead of using a parametric model like before in the bayesian linear regression, a prior probability distribution over functions is used. This is in theory an approach, which does work in infinite dimension, but due to the constrainend amount of data points the calculation can be done in a finite space. This means in general a gaussian process is defined as probability distribution over functions $y(\mathbf{x})$, so that the evaluation of an arbitrary set of data points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ has a joint probability distribution. Important to note is that the joint distribution is completely specified over the second-order statistic, which are the mean and the covariance of the joint probability distribution of y_1, y_2, \dots, y_N . In the most cases, there is no prior information for the mean given, which means that is set to zero. So a Gaussian process is only defined by the covariance of $y(\mathbf{x})$, which was defined in the last part of section 4.1, as the kernel of two input vectors, see equation 4.16.

$$\mathbb{E}[y(\mathbf{x}_i), y(\mathbf{x}_j)] = k(\mathbf{x}_i, \mathbf{x}_j) \quad (4.16)$$

This marginal distribution $p(\mathbf{y})$ is defined in equation 4.17, where \mathbf{K} is the Gram Matrix with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. This matrix contains the combination of each element with each other element in the selected kernel space.

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid \mathbf{0}, \mathbf{K}) \quad (4.17)$$

There are possible kernels, which are defined in section 4.4. The most commonly used kernel is the 'Gaussian' kernel, which is also known under the names Radial Basis Function (RBF) and squared exponential kernel [DG14, Bis06]. It is defined in equation 4.18, where

σ_f^2 is the signal variance, which is the noise of the process, l^2 is the length scale, σ_n^2 is the noise variance, which is the noise of the data and δ_{ij} is the same as $\mathbb{1}(i = j)$.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2} \|\mathbf{x}_i - \mathbf{x}_j\|^2\right) + \delta_{ij} \sigma_n^2 \quad (4.18)$$

Usually a selected kernel express the property that two points \mathbf{x}_i and \mathbf{x}_j , which are similar have corresponding values $y(\mathbf{x}_i)$ and $y(\mathbf{x}_j)$ so that these are more strongly correlated than for dissimilar points. The marginal distribution $p(\mathbf{t})$, conditioned on the trainings data set $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ can be calculated by integrating over \mathbf{y} . In order to keep it simple the dependency on the trainings data is not depicted here. This leads to equation 4.19.

$$p(\mathbf{t}) = \int p(\mathbf{t} | \mathbf{y}) p(\mathbf{y}) d\mathbf{y} = \mathcal{N}(\mathbf{t} | \mathbf{0}, K + \beta^{-1} \mathbf{I}_N) \quad (4.19)$$

The simple addition of the covariance is possible, because the sources of randomness in the two Gaussians, namely in $y(\mathbf{x})$ and in the noise of equation 4.3 are independent. So far only a model of the Gaussian Process viewpoint was build, however the goal is to make a prediction. This means to calculate the value t^* for a new unseen data point \mathbf{x}^* . To find the predictive distribution $p(t^* | \mathbf{t})$, the joint distribution $p(\mathbf{t}^*)$ has to be defined, where \mathbf{t}^* is $(t_1, t_2, \dots, t_N, t^*)^T$. The joint distribution is again a Gaussian and is defined in equation 4.20.

$$p(\mathbf{t}^*) = \mathcal{N}(\mathbf{t}^* | \mathbf{0}, \mathbf{C}^*) \quad (4.20)$$

$$\mathbf{C}^* = \begin{pmatrix} (\mathbf{K} + \beta^{-1} \mathbf{I}_N) & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix} \quad (4.21)$$

$$c = k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1} \quad (4.22)$$

The covariance \mathbf{C}^* of this Gaussian is the combination of the covariance of the trainings data with the new unseen date point. The matrix is depicted in equation 4.21, where c is defined in equation 4.22. Here \mathbf{k} denotes the vector $(k(\mathbf{x}_1, \mathbf{x}^*), k(\mathbf{x}_2, \mathbf{x}^*), \dots, k(\mathbf{x}_N, \mathbf{x}^*))^T$, which represented the kernel combination of each trainings point with the unseen data point.

Using all this results leads to the conditional distribution $p(t^* | \mathbf{t})$, which is also again a Gaussian with the mean and covariance given by equation 4.23 and equation 4.24, where c was defined in equation 4.22.

$$m(\mathbf{x}^*) = \mathbf{k}^T (\mathbf{K} + \beta^{-1} \mathbf{I}_n)^{-1} \mathbf{t} \quad (4.23)$$

$$\sigma^2(\mathbf{x}^*) = c - \mathbf{k}^T (\mathbf{K} + \beta^{-1} \mathbf{I}_n)^{-1} \mathbf{k} \quad (4.24)$$

The biggest computation effort is the inversion of the matrix $\mathbf{K} + \beta^{-1} \mathbf{I}_n$ in the middle of 4.23, which has $O(N^3)$ computationally complexity. This limits the amount of used trainings data drastically, there is an approach to avoid this problem, which is discussed in

section 4.8.

4.3 Gaussian Processes for Classification

As described before the main topic of this thesis is classification. This can be done with Gaussian processes. In the last section the regression case naturally evolved from the bayesian linear regression. The same holds for the classification from the regression. In figure 4.1 the dataset from section 1.1 is used and a Gaussian process is applied to it. The dots in the picture are like before the data points, the background indicates the result of the algorithm. So the blue area illustrates, where the algorithm is sure about assigning the class label blue. The same holds for the yellow area and furthermore there is a separated grey area in which the Gaussian process can say that there is no further information available and it is neither of the both classes. This is especially usefully to detect outliers and unknown classes. The size of the areas depend on the hyperparameters of the used Gaussian kernel.

In order to do that the posterior probabilities of the target value have to lie in the interval $(0, 1)$, where as before the target values where on the entire real axis. This can be achieved by applying a logistic function $\sigma(x)$ on the resulting function $f(x)$ from the Gaussian process regression. This results in $\sigma(f)$, which is limited to $(0, 1)$. A logistic function can be anything, which takes an input value and limits it to the interval $(0, 1)$. A typical logistic function is the sigmoid function, which is defined in equation 4.25 and depicted in red in figure 4.2.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (4.25)$$

The sigmoid function has the advantage that it is smooth and differentiable. This enables the existence of borders between two classes, in which the Gaussian processes then can say that there is no further information available, based on a probabilistic model. That is something, which is very unique for a machine learning approach.

An other logistic function is the cumulative Gaussian $\Phi(x)$, which is defined in equation 4.26. It is the cumulative distribution function over a Gaussian normal distribution. In this case the standard Gaussian is used with mean zero and variance of one. The function is plotted in blue in figure 4.2. The cumulative Gaussian has some advantages over the sigmoid function, which are elaborated in section 4.8.

$$\Phi(x) = \int_{-\infty}^x \mathcal{N}(z | 0, 1) dz \quad (4.26)$$

The goal is again finding the predictive distribution $p(t^* | \mathbf{t})$, where t^* correspond to a unseen input value x^* and each element of \mathbf{t} to one input data point $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$. This can be achieved by introducing a Gaussian process prior over the vector \mathbf{f}^* , where the elements are $f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_N), f(\mathbf{x}^*)$. The relation between one output value t and the corresponding output of the activation function $f(x)$ leads to the non-Gaussian process over

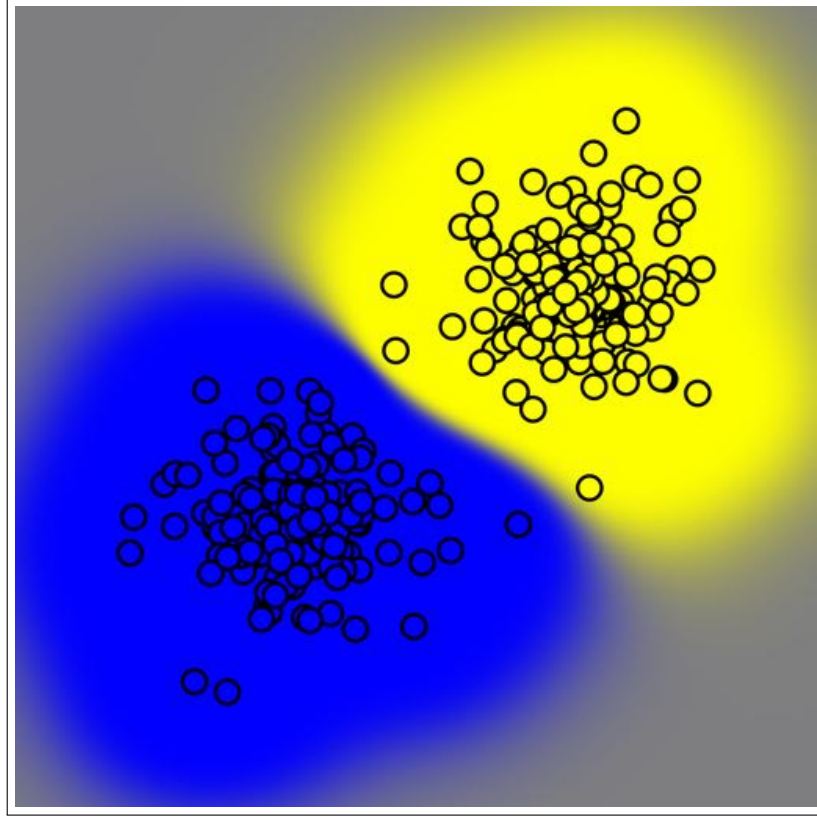


Figure 4.1: The dataset from section 1.1 is used in combination with the Gaussian process approach, where a standard Gaussian kernel is used. As before the data points are plotted as dots and behind it the color of the background indicates the class membership. Around each class center the probability for this class is high, further away it is low. The grey area around both classes shows that there is no further information available and both classes are equally probable. The size of the areas depend on the hyperparameters in the Gaussian kernel.

\mathbf{t}^* by conditioning on \mathbf{t} , where \mathbf{t}^* is $(t_1, t_2, \dots, t_N, t^*)^T$. This leads to the Gaussian process prior for \mathbf{f}^* defined in equation 4.27, where \mathbf{C} is the covariance matrix. The elements of \mathbf{C} are like in the gram matrix the combination of two input vectors in the kernel space. This is defined in equation 4.28, here any valid kernel is possible. Adding a small value to the diagonal of \mathbf{C} can improve the numerical stability, because the labels in the classification case do not contain any error, they belong either to one class or an other.

$$p(\mathbf{f}^*) = \mathcal{N}(\mathbf{f}^* | \mathbf{0}, \mathbf{C}) \quad (4.27)$$

$$C(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j) \quad (4.28)$$

For the binary case scenario it is enough to predict $p(t^* = 1 | \mathbf{t})$ and calculating $p(t^* = 0 | \mathbf{t})$ by $1 - p(t^* = 1 | \mathbf{t})$. The predictive distribution is again the marginalization over the prior and the likelihood. It is defined in equation 4.29 and in equation 4.30 the likelihood was

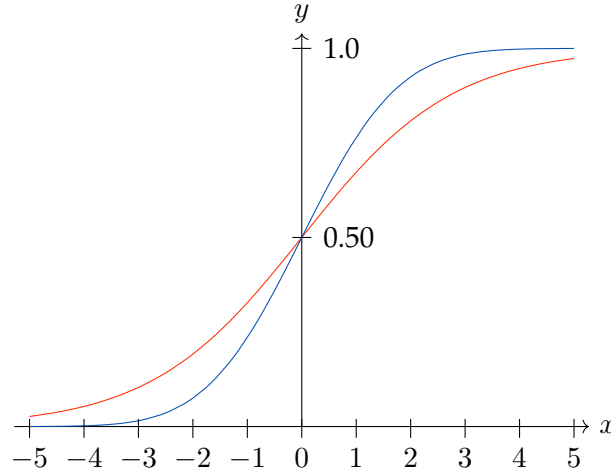


Figure 4.2: Plots of the used sigmoids functions. The magenta line represents the logistic function $\sigma(\mathbf{x})$ defined in equation 4.25. The blue line is the cumulative Gaussian $\Phi(x)$ in equation 4.26. For the standard configuration of the cumulative Gaussian with mean of zero and variance of one, the curve is steeper than the one of the logistic function.

replaced by the logistic function $\sigma(f^*)$. This shows that the predictive distribution is no longer a Gaussian and is therefore not analytically tractable.

$$p(t^* = 1 \mid \mathbf{t}) = \int p(t^* = 1 \mid f^*) p(f^* \mid \mathbf{t}) df^* \quad (4.29)$$

$$p(t^* = 1 \mid \mathbf{t}) = \int \sigma(f^*) p(f^* \mid \mathbf{t}) df^* \quad (4.30)$$

There are different approaches to overcome the obstacle of intractability. Different methods have been used in the past to approximate the integral. A good comparison is done in "Approximations for Binary Gaussian Process Classification", which compared Laplace Approximation, Variational Bounding and Expectation Propagation with each other. Nickisch and Rasmussen showed that the Expectation Propagation algorithm produces the best results, however the computation time is ten times higher than in the Laplace Approximation case [KR06, NR08]. So in this thesis the Expectation Propagation was used, because its iterative behavior can be combined with the training of the Gaussian Process, so that the result is a fast and precise algorithm. This is elaborated more in detail in the following sections.

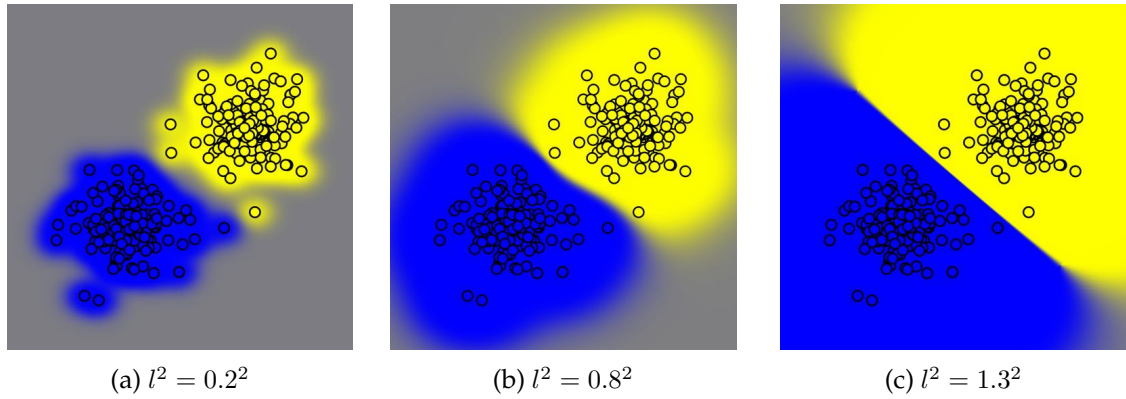


Figure 4.3: These figures show the influence of the length hyperparameter on the result of the Gaussian process. The left picture (a) has a low length scale of 0.2^2 , which is in this case too low to fully grasp the data. There are holes between some data points, which could be better filled like in the middle picture (b), where the length scale is higher with 0.8^2 and therefore a broader area is covered. However in the right picture (c) the length is too big with 1.3^2 and the covered area makes assumptions about the data, which are not necessarily true.

4.4 Kernels for Gaussian Processes

4.4.1 Gaussian kernel

There are many different possible kernels for Gaussian processes. The standard kernel is the Gaussian kernel previously defined in equation 4.18. It has three different hyperparameters, which have to be determined. The hyperparameters determine the strength between the similarity of two data points. Here similarity is measured as the inverse of the kernel result. A high similarity corresponds to a low value of the kernel result. In equation 4.18 the length parameter is inverted, which means that increasing it, increases the similarity between the points by decreasing the kernel result. So increasing the length scale enlarges the areas of the assigned classification. This can be seen in figure 4.14, where the length scale is varying and the noise parameters are fixed. On the left side in the figure 4.14 the length is too low and therefore the similarity between the data points is too low. However on the right side in the figure the length parameter is too high and the similarity between the points is so high, that the areas expand far beyond the knowledge of the data. Good in this scenario is a length of $l^2 = 0.8^2$.

The second important parameter, which needed to be selected is the signal noise σ_f^2 . It influences the gradient at the border of the areas, where a low noise value also lowers the gradient at the borders and makes a smoother blending. A higher signal variance σ_f^2 sharpens the edges except there is no other area at the boundary then it just increases the area in this direction. In figure 4.4 the influence of the signal variance on the data set from section 1.1 is pictured. The left picture has a low variance, which blurs the edges of the areas of both classes and in this case the data is therefore not well presented. In

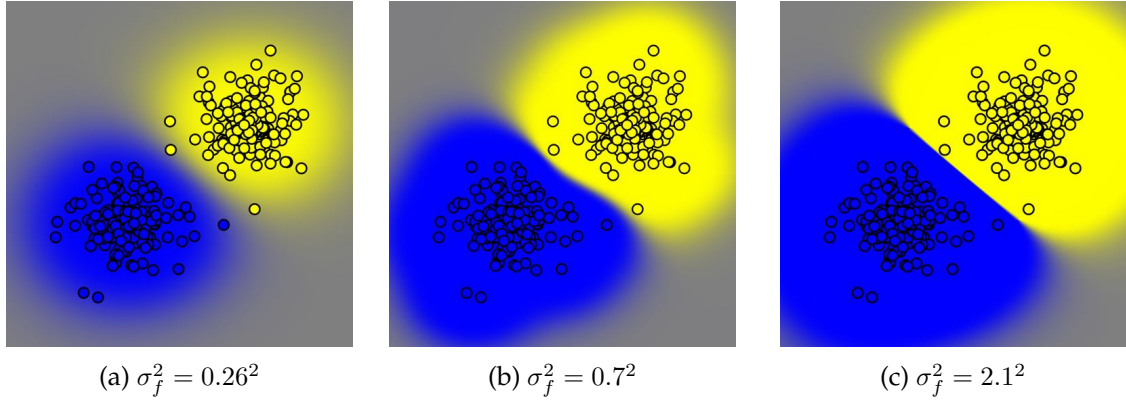


Figure 4.4: Different signal noise hyperparameters are plotted in this figure. In the left picture (a) a low signal noise value of 0.26^2 is used, which makes the edges in this case to be fuzzy and the outer points are not clearly covered. The middle is the optimal case from before. But in the right picture (c) the signal noise is a bit too high and the border between the classes is too sharp. Furthermore the high value increases the area, which is not a desired behavior.

a good representation the training data would have a high certainty, because without further knowledge the data points are correctly classified. The middle picture in figure 4.4 is a good representation, the data points are well classified and further away from the data the Gaussian process detects that there is no knowledge available. However in the right picture the signal noise is too high and the edge between both areas is just a sharp line without any blending and this representation does most likely not represent the real world. Furthermore through the high noise the area increases and makes assumptions about the spaces, which are not necessarily true.

4.4.2 Gaussian kernel with expanded length

In equation 4.18 the Gaussian kernel is defined, but in a high dimensionally space the standard kernel does not perform best, because the length parameter scales all dimension equally. In order to overcome this flaw the length parameter can be replaced with a matrix, this can be seen in equation 4.31.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)\right) + \delta_{ij} \sigma_n^2 \quad (4.31)$$

Here \mathbf{M} can take different forms, in the standard one it is $\mathbf{M} = l^{-2} \mathbf{I}$. But now each dimension can have its own length scale parameter with $\mathbf{M} = \text{diag}(l_1, l_2, \dots, l_D)^{-2}$, this makes it possible to adjust the Gaussian Process more to the data. This can be seen in figure 4.5, where the upper Gaussian process has only one length scale parameter. This leads for this data set to a bad representation of the points, which can be seen in the right and left corner, where the blue points are printed on a yellow background. In the lower picture the Gaus-

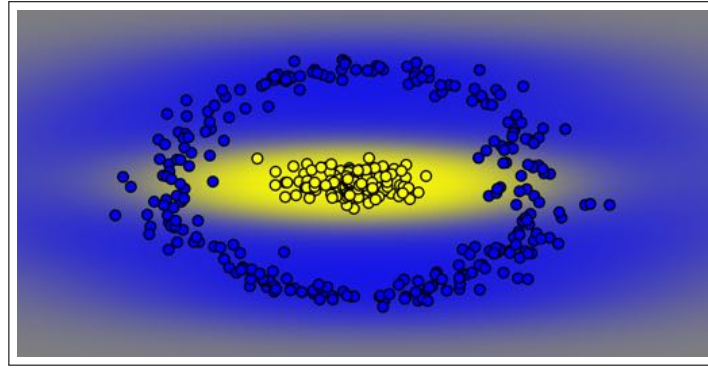
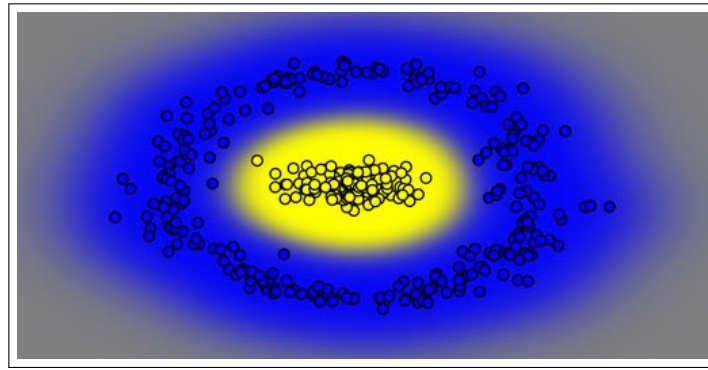
(a) With one length parameter $l^{-2}\mathbf{I}$ (b) With D lengths parameters $\text{diag}(l_1, l_2, \dots, l_D)^{-2}$

Figure 4.5: The influence of two different lengths parameters is shown here. The Gaussian process in the upper figure has the same length parameter for both dimension and in this scenario this is not enough for representing the data well. However in the lower picture each dimension has its own length scale and so the vertical dimension can have a lower length scale, which represents the data better.

sian process uses two length parameter for each dimension one, which leads to a better representation. The background of the data points contains the right color, which means that the classification of this data set would succeed. If the amount of dimension would increase this approach would get even more useful, because in a higher dimensional space the abstraction with just one parameter is more difficult and then using more than one could lead to a better representation. However increasing the amount of hyperparameters means also that the training of these gets more complex.

4.4.3 Random Forest kernel

One other kernel is the Random Forest kernel, it has some advantages and disadvantages over the standard Gaussian kernel [DG14]. This approach is based on the idea of the random partition kernel, which defines the kernel function as described in equation 4.32. The kernel function compares if two input vectors share the same cluster, which is given

by the partition function $\varrho(\mathbf{x})$.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E} [\mathbb{1}(\varrho(\mathbf{x}_i) = \varrho(\mathbf{x}_j))] \quad (4.32)$$

To prove that this is a valid kernel, the kernel from equation 4.32 can be rewritten to equation 4.33. Here the expected value was resolved into an unbounded sum.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{\varrho}^n \mathbb{1}(\varrho(\mathbf{x}_i) = \varrho(\mathbf{x}_j)) \quad (4.33)$$

If now only the inner part of the sum is used to generate a kernel, the kernel values can only be zero or one. The resulting kernel matrix \mathbf{K}_{ϱ} for any dataset with N elements can now be generated and can be reordered with the permutation matrices \mathbf{P} to the matrix shown in equation 4.34. It is arranged as a block diagonal matrix, that means that each 1 represents a block of ones, which all belong to the same cluster defined before.

$$\mathbf{P}\mathbf{K}_{\varrho}\mathbf{P}^T = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.34)$$

This block diagonal matrix $\mathbf{P}\mathbf{K}_{\varrho}\mathbf{P}^T$ is a positive-definite kernel, because each block matrix has only eigenvalues which are non negative and the eigenvalues of the whole matrix are just the combination of all of the block matrices eigenvalues. That means all eigenvalues are non negative and so the block diagonal matrix is a positive-definite kernel. Furthermore a permutation matrix does not change the eigenvalues of a matrix and therefore \mathbf{K}_{ϱ} is a valid positive-definite kernel. This means that also the inner part of the sum is a valid kernel. From that it is possible to conclude that the kernel defined in equation 4.33 is a valid kernel, because the linear combination of valid kernels is also a valid kernel. This proves that the kernel defined in equation 4.32 is a valid kernel.

TODO: nice citation for the math rules

However the computation of this kernel is not possible in practice, to overcome this an approximation has to be made, which is then called the m -approximate Random Partition Kernel. To do that instead of calculating the expectation value, the cluster assignment is evaluated m -times and averaged, this can be seen in equation 4.35. This corresponds to a maximum likelihood estimator and is also again a valid kernel, because as before the combination of valid kernels is also valid kernel.

$$k(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{m} \sum_{\varrho}^m \mathbb{1}(\varrho(\mathbf{x}_i) = \varrho(\mathbf{x}_j)) \quad (4.35)$$

So in order to construct the Random Forest Kernel a Random Forest has to be trained, as described in section 3. Then the m -approximate Random Partition Kernel from before can be used to define a valid kernel on a random forest. The pseudo algorithm for that is

defined in algorithm 2.

Algorithm 2 Random Forest Kernel function

Require: $\mathcal{D}, \mathbf{y}, \text{maxHeight}, m$

```

1:  $F \leftarrow \text{RandomForest}(\mathcal{D}, \mathbf{y}, \text{maxHeight}, m)$ 
2: procedure KERNELFUNC( $F, \mathbf{x}_1, \mathbf{x}_2$ )
3:    $sum \leftarrow 0$ 
4:   for  $i$  in  $F.\text{amountOfTrees}$  do
5:      $h \leftarrow \text{UniformDiscrete}(1, \text{maxHeight})$ 
6:     if  $F[i].\text{haveSameNodeAt}(\mathbf{x}_1, \mathbf{x}_2, h)$  then
7:        $sum \leftarrow sum + 1$ 
   return  $sum/m$ 

```

The required inputs for this algorithm are the data points \mathcal{D} and the corresponding labels \mathbf{y} for each point. Furthermore the maximal amount of layers of a tree has to be defined in *maxHeight* and also the amount of samples m per kernel evaluation. In Line 1 of the algorithm the RandomForest is generated and also trained, the amount of samples define how many trees have to be trained. So that for each sample step a single tree is available. The next line defines the kernel function $k(\mathbf{x}_1, \mathbf{x}_2)$ for two data points, it uses the trained Random Forest from before. Now for each tree $F[i]$ in the Random Forest the function *haveSameNodeAt()* is called. It returns true if the two data points \mathbf{x}_1 and \mathbf{x}_2 share the same node at the height h in the tree. This height is sampled from a uniform distribution between 1 and the height of the trees in the Random Forest. If the function *haveSameNodeAt()* returns true the sum is incremented until all trees are processed. In the end the sum is divided through the amount of trees, which is then the kernel function result. That means if the two input values are the same it will return 1, because all decision trees will decide identically. But if they are different the result of the decision trees will not be similar and then a value between zero and one is returned, describing the similarity of this two input values. That this algorithm is a valid kernel can be deduced from the fact that it describes a m -approximate Random Partition Kernel. The indicator function in equation 4.35 is represents the function call *haveSameNodeAt()*, which increments the sum if the result is true. The last line of the algorithm resembles the normalization factor $\frac{1}{m}$ in the equation 4.35. This proves that this algorithm is a valid kernel and can be used in any Gaussian Process. In figure 4.6 the Random Forest Kernel is used on the data set defined in section 1.1. It demonstrates the properties of a Random Forest Kernel. One of them is that it is non-stationary, which means that it does not only depend on the distance between two data points. This means also that it is a piece-wise constant function, which can be seen in figure 4.6. The lower left corner is classified with a high certainty to the blue class, even if there is no given evidence for that and this behavior is valid for the whole area in the lower left area. An other property is that the Random Forest Kernel has no hyperparameters, which removes the need for an expensive search of them. This hyperparameter optimization is in general the most expensive part of the training. The last property is that the kernel is supervised, which means that the trainings data is used

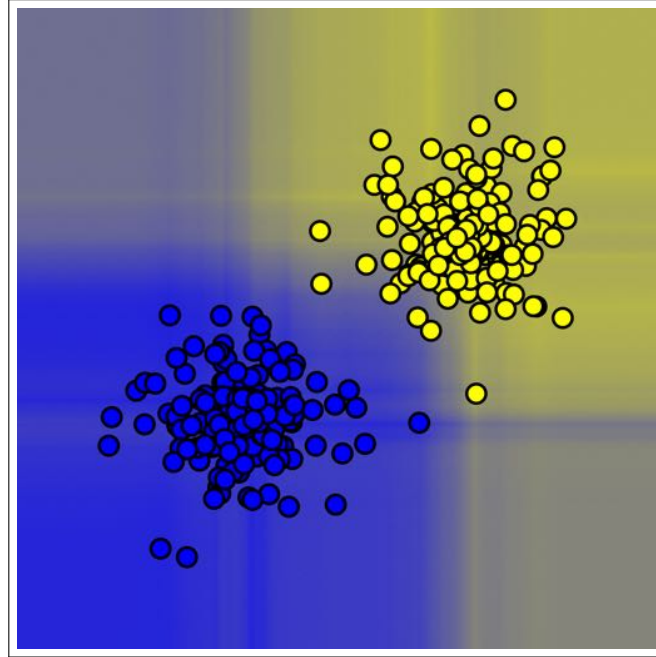


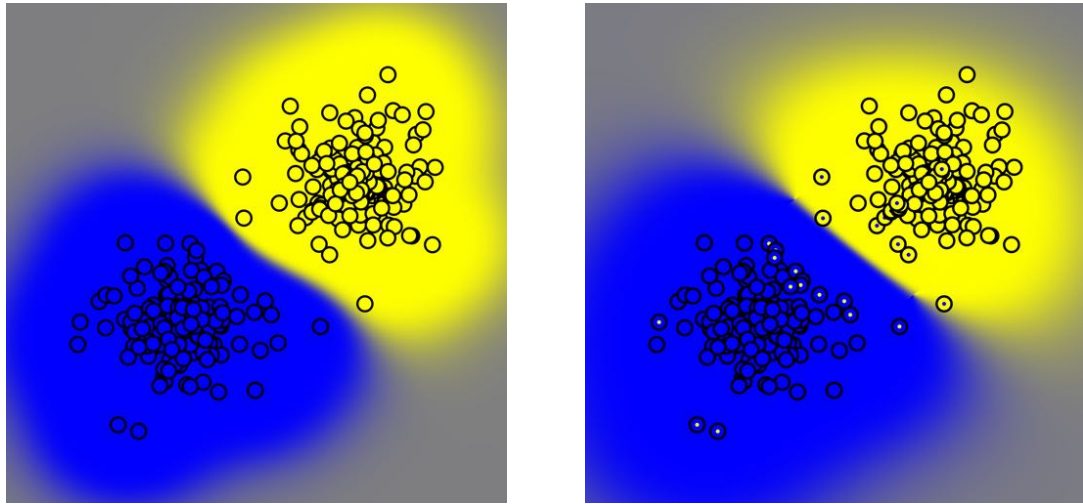
Figure 4.6: A Gaussian Process with the Random Forest Kernel is used here on the dataset from section 1.1. Like before the data points are plotted as dots and behind it the color of the background indicates the class membership. In contrast to the Gaussian Kernel the area is not smoothly defined, because the splits from the used Random Forest are always applied to an axis. Furthermore the certainty around the points is slightly lower than in the Gaussian Kernel case, where around the centers the certainty of the classification was one. Important to note is the non-stationary behavior of the kernel, which means the lower left corner and the upper right corner belong to the class even if there is no evidence for that.

to generate the kernel. The most standard kernels are unsupervised and only adjust to the data over the hyperparameters, where as this kernel is supervised and is trained on the data. This mean that the kernel can overfit. But the tests run by Davies and Ghahramani does not show that [DG14]. In summary the Random Forest Kernel is a good alternative to the Gaussian Kernel and has some unique properties, especially the capability not to train the hyperparameters, can be very useful in the setting of this thesis. Furthermore it can be as an online kernel, like described in section 3.4.

4.5 Informative Vector Machine

The section 4.3 describes how Gaussian Process can be used to classify points. One big withdrawal is that the complexity of the standard algorithms rises cubically with the number of points. To solve this there are several approaches one of them is called Informative Vector Machine (IVM), which is a Sparse Gaussian Process. That means that not all points

have to be used. Figure 4.7 compares the full Gaussian Process to the sparse Informative Vector Machine. In the right picture all points were used and the computation time therefore was comparatively high. The left picture however only used 28 of the data points as inducing points, which dramatically reduces the computational effort. In order to understand how the Informative Vector Machine works the Expectation Propagation algorithm is explained first. This way the IVMs naturally expands from the knowledge about the Expectation Propagation [LSH⁺03].



(a) Gaussian Process approach with all points (b) Informative Vector Machine with only 28 induced points

Figure 4.7: The left figure shows the Gaussian Process result for the simple dataset introduced in section 1.1. The Gaussian posterior was calculated on all data points, which takes more time than only using a subset of points. In the right picture the Informative Vector Machine was used, it only uses 28 points as active set, which reduces the trainings and prediction time dramatically.

4.6 Assumed-density filtering

In order to understand the Expectation Propagation algorithm the assumed-density filtering (ADF) is elaborated first. The predictive distribution defined in equation 4.30 is intractable, therefore an algorithm like ADF has to be used to make an approximation of the integral [REM11].

There are different approaches to approximate the area of an integral. They can be divided into deterministic and non-deterministic approaches [Min01]. The deterministic approaches try to calculate the area of the integral, by using for example a couple of function values over the integral and interpolate from there the true area. The problem with that is that in the given case the posterior is sparse and furthermore has high a dimensionality, both mean that this simple approach is not able to capture the area. More complex ap-

proaches try to use more than just the function values as information. The other side of the medal are the non-deterministic approaches, these sample points in the integral and evaluate them. Good approximators are the Metropolis sampling and the Gibbs sampling [Bis06]. However they are not used in this setting, because it is known that the integrals of Gaussian Processes are sparse, which makes it easier just to look at the area where the action is happening [REM11]. This can be done with a deterministic approach like ADF or the Laplace's method. However several paper have already proven that the Expectation Propagation works better than the Laplace's method, therefore it is not elaborated here [Min01, Bis06].

The assumed-density filtering (ADF) is a fast sequential method for approximating the posterior $p(\mathbf{f} \mid \mathcal{D})$. Where \mathcal{D} is again the combination of the input points \mathbf{x} with the corresponding labels \mathbf{y} and \mathbf{f} are the hidden values. The prior $p(\mathbf{f})$ was defined in equation 4.27 and the joint distribution of \mathbf{f} and \mathcal{D} for n independent observations can then be written as can be seen in equation 4.36.

$$p(\mathbf{f}, \mathcal{D}) = p(\mathbf{f}) \prod_i^n p(\mathbf{x}_i \mid \mathbf{f}) \quad (4.36)$$

To apply the assumed-density filtering the equation 4.36 has to be split into simple factors. This is done in equation 4.37.

$$p(\mathbf{f}, \mathcal{D}) = \prod_i^n t_i(\mathbf{f}) \quad (4.37)$$

These splitting can be done in many different ways. In general it is better to use lesser terms, since it entails fewer approximations. But each term has to be simple enough that the expectation can be propagated through. From equation 4.36 and equation 4.37 it is possible to infer equation 4.38 and equation 4.39.

$$t_0(\mathbf{f}) = p(\mathbf{f}) \quad (4.38)$$

$$t_i(\mathbf{f}) = p(\mathbf{x}_i \mid \mathbf{f}) \quad (4.39)$$

The next step is to choose an appropriate approximation to explain the assumed-density filtering. A spherical Gaussian is used, which is defined in equation 4.40.

$$q(\mathbf{x}) \sim \mathcal{N}(\mathbf{m}_x, v_x \mathbf{I}_d) \quad (4.40)$$

After that the terms $t_i(\mathbf{f})$ have to be incorporated into the approximate posterior. In each iteration a new $q(\mathbf{f})$ is generated from an old $q^i(\mathbf{f})$. The series of $q(\mathbf{f})$ is initialized with one and taking the prior term into account is trivial, because no approximation is needed. However incorporating the other terms $t_i(\mathbf{f})$ is more difficult, this can be done by taking

the exact posterior $\hat{p}(\mathbf{f})$ defined in equation 4.41.

$$\hat{p}(\mathbf{f}) = \frac{t_i(\mathbf{f}) q^{\setminus i}(\mathbf{f})}{\int_{\mathbf{f}} t_i(\mathbf{f}) q^{\setminus i}(\mathbf{f}) d\mathbf{f}} \quad (4.41)$$

The KL-divergence $\text{KL}(\hat{p}(\mathbf{f}) \| q^{\text{new}}(\mathbf{f}))$ from the exact posterior $\hat{p}(\mathbf{f})$ subject to the $q(\mathbf{f})$ is now minimized, where as $q(\mathbf{f})$ is part of the exponential family. This minimization is performed to integrated the terms $t_i(\mathbf{f})$ in $q(\mathbf{f})$. For any exponential family this is just the propagation of the Expectations, which is just the matching of the moments of the exponential family. In each step a normalizing factor Z_i is computed, it is defined in equation 4.42.

$$Z_i = \int_{\mathbf{f}} t(\mathbf{f}) q(\mathbf{f}) d\mathbf{f} \quad (4.42)$$

As an extension the assumed-density filtering also estimates the $p(\mathcal{D})$, which can be calculated by taking the product of all Z_i .

The final assumed-density filtering algorithm is than for the spherical Gaussian defined in equation 4.40:

1. Init $\mathbf{m}_x = \mathbf{0}$, $v_x = \text{prior}$, $s = 1$ (as the scale factor)
2. Each data point \mathbf{x}_i , updates now the parameters (\mathbf{m}_x, v_x, s) based on the old parameters $(\mathbf{m}_x^{\setminus i}, v_x^{\setminus i}, s^{\setminus i})$ without the new data point \mathbf{x}_i with:

$$\begin{aligned} \mathbf{m}_x &= \mathbf{m}_x^{\setminus i} + v_x^{\setminus i} r_i \frac{\mathbf{x}_i - \mathbf{m}_x^{\setminus i}}{v_x^{\setminus i} + 1} \\ v_x &= v_x^{\setminus i} - r_i \frac{(v_x^{\setminus i})^2}{v_x^{\setminus i} + 1} + r_i (1 - r_i) \frac{(v_x^{\setminus i})^2 (\mathbf{x}_i - \mathbf{m}_x^{\setminus i})^T (\mathbf{x}_i - \mathbf{m}_x^{\setminus i})}{d(v_x^{\setminus i} + 1)^2} \\ s &= s^{\setminus i} \times Z_i(\mathbf{m}_x^{\setminus i}, v_x^{\setminus i}) \end{aligned}$$

In words this algorithm describes that for each point \mathbf{x}_i the probability r_i of not being part of the approximation is used to make a soft update to our estimate of $\mathbf{f}(\mathbf{m}_x)$ and change our confidence in the estimate v_x . From this procedure it is possible to conclude that the order in which the points are processed defines the end result. This can be explained by the probability r_i , which always depends on the current estimate of \mathbf{f} and therefore has a big influence on the order of the points. So the error of the assumed-density filtering depends mainly in which order the points are presented to the algorithm. This problem is tackled later in the next section.

4.7 Expectation Propagation

In the last section the assumed-density filtering algorithm was explained, which suffers under the problem that the ordering of the points is essential to the approximation of the

posterior. To overcome this problem the Expectation Propagation algorithm was designed, which approximation does not depend on the order of the points.

In the ADF the observation term $t_i(\mathbf{f})$ is exactly used and then with that the posterior is approximated. However it is also possible first to approximate $t_i(\mathbf{f})$ with some $\tilde{t}_i(\mathbf{f})$ and then using an exact posterior with $\tilde{t}_i(\mathbf{f})$. That means the observation term $t_i(\mathbf{f})$ is just an intermediate state to an higher term. In order to do that the approximate term $\tilde{t}_i(\mathbf{f})$ is the ratio of the new posterior to the old posterior times a constant, which can be seen in equation 4.43. This ensures that the approximation is always possible.

$$\tilde{t}_i(\mathbf{f}) = Z_i \frac{q(\mathbf{f})}{q^{\setminus i}(\mathbf{f})} \quad (4.43)$$

The approximation term $\tilde{t}_i(\mathbf{f})$ multiplied by $q^{\setminus i}(\mathbf{f})$ gives the $q(\mathbf{f})$, which is as desired. The important property here is that if the approximated posterior is in an exponential family than the term approximations will be in the same family.

The assumed-density filtering sequentially computing a Gaussian approximation $\tilde{t}_i(\mathbf{f})$ to every observation term $t_i(\mathbf{f})$ and combines all approximations $\tilde{t}_i(\mathbf{f})$ analytically to a Gaussian posterior on \mathbf{f} . This is illustrated in figure 4.8, where the upper line illustrates the old already transferred observation terms $t_i(\mathbf{f})$ to the approximation term $\tilde{t}_i(\mathbf{f})$ for the first and second element. The third element is the actual element and is the next to be integrated into the posterior $q(\mathbf{f})$. This is done as mentioned before in a sequential manor for all points.

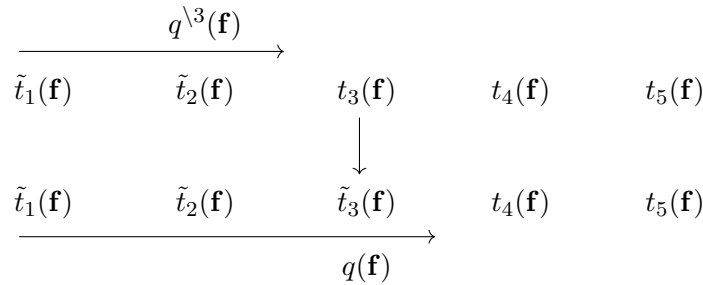


Figure 4.8: The behavior of the assumed-density filtering is illustrated here. For each new observation term $t_i(\mathbf{f})$ an approximation term $\tilde{t}_i(\mathbf{f})$ is made and then these are analytically combined to get the posterior $q(\mathbf{f})$ [Min01]

In the Expectation Propagation however the ordering is not important, because one term is selected and excluded from the posterior. Then the posterior is newly approximated, which takes into account the whole context. This behavior is illustrated in figure 4.9, where first all approximation terms have to be determined, that means a complete ADF algorithm is done, before the EP can really start. After that a term is selected and excluded as can be seen in figure 4.9. The new posterior $q(\mathbf{f})$ can be calculated without it and after that the approximation term is the fraction between the newly calculated posterior and the old

posterior multiplied with a constant.

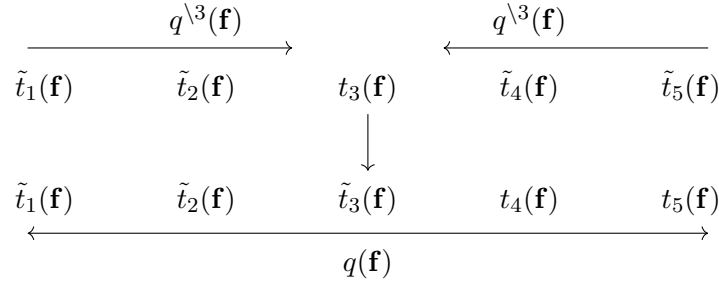


Figure 4.9: Instead of the sequential behavior of the assumed-density filtering the Expectation Propagation uses all points by using the posterior $q^{3}(\mathbf{f})$ without the actual observation term $t_3(\mathbf{f})$. By integrating over all the others terms and then approximating the new term by the difference between the old and new posterior [Min01].

So from that the general EP algorithm can be defined as:

1. Init the term approximations $\tilde{t}_i(\mathbf{f})$
2. Compute the posterior $q(\mathbf{f})$ from the product of $\tilde{t}_i(\mathbf{f})$:

$$q(\mathbf{f}) = \frac{\prod_i \tilde{t}_i(\mathbf{f})}{\int \prod_i \tilde{t}_i(\mathbf{f}) d\mathbf{f}}$$

3. Till all $\tilde{t}_i(\mathbf{f})$ converge (until the change is smaller than some epsilon):
 - a) Select an approximation term $\tilde{t}_i(\mathbf{f})$ to refine
 - b) Remove this one from the posterior to get an 'old' posterior $q^{old}(\mathbf{f})$, this can be done by dividing and normalizing again:

$$q^{old}(\mathbf{f}) \propto \frac{q(\mathbf{f})}{\tilde{t}_i(\mathbf{f})}$$

- c) The normalizing factor Z_i and the new posterior $q^{new}(\mathbf{f})$ can be calculated like before in the assumed-density filtering.
 - d) Update the approximation term:

$$\tilde{t}_i(\mathbf{f}) = Z_i \frac{q^{new}(\mathbf{f})}{q^{old}(\mathbf{f})}$$

4. At last $p(\mathcal{D})$ can be computed by:

$$p(\mathcal{D}) \approx \int \prod_i \tilde{t}_i(\mathbf{f}) d\mathbf{f}$$

A side note here is that step 3b can also be performed by accumulating all other terms except term i . This is defined in equation 4.44, however the division is usually more efficient, than multiplying all terms.

$$q(\mathbf{f}) \propto \prod_{j \neq i} \tilde{t}_j(\mathbf{f}) \quad (4.44)$$

4.8 Informative Vector Machine as Extension of EP

As mentioned in the beginning of this section the Informative Vector Machine evolves naturally from the Expectation Propagation Algorithm. In order to understand this the problem of the Gaussian Process for classification has to be revisited. In equation 4.30 the posterior $p(\mathbf{f} | \mathcal{D})$ for a new data point was defined. This posterior contains the non-Gaussian part $\sigma(\mathbf{f})$, which makes analytically intractable. Therefore an approximation algorithm has to be used, this approximation algorithm should ideally preserve the mean and covariance function of the former. For all functions from the exponential family this can be done by moment matching, where the parameters of the exponential function have to be equal. The resulting Gaussian approximation $q(\mathbf{f})$, which solely depends on the posterior $p(\mathbf{f} | \mathcal{D})$, so that the conditional posterior $p(f^* | \mathbf{f})$ for a new point \mathbf{x}^* is identical to the conditional prior $p(f^* | \mathbf{f}, \mathcal{D})$ [LSH⁺03].

This Gaussian approximation $q(\mathbf{f})$ can not be computed analytically. However a parametric representation can be found in equation 4.45.

$$q(\mathbf{f}) \propto p(\mathbf{f}) \prod_i^n \exp\left(-\frac{p_i}{2}(u_i - m_i)^2\right) \quad (4.45)$$

This representation is close to the definition in the Expectation Propagation in equation 4.36. That means that the approximation $q(\mathbf{f})$ can be obtained from the true posterior $p(f^* | \mathbf{f})$ by a likelihood approximation. The factors (p_i, m_i) in equation 4.45 are called sites and are initialized with zero. This means that in the beginning the approximation equals the prior $q(\mathbf{f}) = p(\mathbf{f})$. In order now to update these site parameters the assumed-density filtering algorithm is applied. So a new data point \mathbf{x}_i is added, by replacing the approximate site parameters by the true parameters given from $p(\mathbf{x}_i | \mathbf{f})$. This results in a non-Gaussian distribution, whose mean and covariance can still be computed. So it is possible to approximate a new posterior $q^{new}(\mathbf{f})$ by using the properties of the exponential family and just match the moments. This update of the site parameters is called the inclusion of i into the active set \mathbf{I} , which can also be seen as the minimization of the Kullback Leibler divergence defined in equation 4.46 [LSH⁺03, LPJ05].

$$\text{KL}(q(\mathbf{f}) || q^{new}(\mathbf{f})) = - \int q(\mathbf{f}) \log\left(\frac{q^{new}(\mathbf{f})}{q(\mathbf{f})}\right) d\mathbf{f} \quad (4.46)$$

The connection to the assumed-density filtering can be best seen through the local property, which only changes one term at a time by adjusting the posterior to it, this correspond

to the inclusion step in the Informative Vector Machine.

From that the algorithm can be inferred it starts with setting all the site parameters p_i, m_i to zero for all i for which $i \notin \mathbf{I}$, where $\mathbf{I} \subset \{1, 2, \dots, n\}$ is the active set and $\|\mathbf{I}\| = d$, with $d \leq n$. A big influence on the result of the IVM is the selection of the active set \mathbf{I} , because a bad selection leads to a bad representation of the feature space and therefore to a poor performing Informative Vector Machine. One of the simplest methods is to follow a greedy select approach, this can be done by calculating a score function over all remaining points $\mathbf{J} \subset \{1, 2, \dots, n\} \setminus \mathbf{I}$ and selecting the one, which has the highest score. The heuristic suggested by Lawrence *et al.* in "Fast Sparse Gaussian Process Methods: The Informative Vector Machine" measures the decrease in entropy of $q(\mathbf{f})$ upon its inclusion [LSH⁺03]. This heuristic called differential entropy score, has the advantage that the posterior variance is reduced. This can be proven over the reduction of entropy, which is derived in equation 4.47 to equation 4.50. The difference in the entropy is always non positive, because the new entropy should always be smaller or equal than the old one, which implies that $\sigma_{new}/\sigma_{old}$ is always smaller than one, so that it is possible to state $\sigma_{new} \leq \sigma_{old}$. This can also be seen as selecting the point, which slices the largest possible section away from the feature space, which is similar to the understanding of variances.

$$D = \mathcal{H}[q^{new}(\mathbf{f})] - \mathcal{H}[q(\mathbf{f})] \quad (4.47)$$

$$D = \left(\frac{1}{2} \ln(2\pi e \sigma_{new}^2)\right) - \left(\frac{1}{2} \ln(2\pi e \sigma_{old}^2)\right) \quad (4.48)$$

$$D = \ln\left(\sqrt{\frac{2\pi e \sigma_{new}^2}{2\pi e \sigma_{old}^2}}\right) \quad (4.49)$$

$$D = \ln\left(\frac{\sigma_{new}}{\sigma_{old}}\right) \quad (4.50)$$

Thus, this heuristic favors points, whose inclusion leads to a large reduction in posterior variance. The biggest advantage of this heuristic is the fast evaluation of the entropy. The algorithm 3 defines how the Informative Vector Machine works.

The inputs of the algorithm are the size of the active set d , the labels \mathbf{y} from the data points, which have to be either 1 or -1 , the Gram matrix defined section 4.2 and the variance parameter λ of the cumulative Gaussian. In comparison to the sigmoid function defined in equation 4.25, the cumulative Gaussian from equation 4.26 is simpler to evaluate in the prediction case, because there is an analytical solution for it. For the sigmoid function a sampling would be necessary.

In the initialization of the algorithm 3 the active set \mathbf{I} is initialized empty and the inactive set \mathbf{J} gets the indices of all data points. The vector zeta ζ is set with the diagonal of the gram matrix, which is for the Gaussian Kernel just the noise variance σ_n^2 . Line four defines the bias it depends on the fraction of points belonging to the first class. In the lines six to eleven in the algorithm the entropy reduction for each point of \mathbf{J} is calculated. Important to note is that in comparison to active learning in line five the label \mathbf{y}_j of the j point is used. This is done for each inducing point k in d , from that the highest entropy reduction can be

Algorithm 3 The Informative Vector Machine with a simple point selection**Require:** $d = \text{nr. of inducing points}, \mathbf{K}, \mathbf{y}, \lambda$

- 1: $\mathbf{J} = \{1, 2, \dots, n\}, \mathbf{I} = \{\}$
- 2: $\boldsymbol{\mu} = \mathbf{0}, \mathbf{m} = \mathbf{0}, \beta = 0$
- 3: $\zeta_0 = \text{diag}(\mathbf{K})$
- 4: $b = 1 - \Phi\left(\frac{\text{amountOfOnes}(\mathbf{y})}{n}\right)$ $\triangleright \text{amountOfOnes}(\mathbf{y})$ returns the nr. of ones in \mathbf{y}
- 5: **for** $k = 1$ to d **do**
- 6: **for all** j in \mathbf{J} **do**
- 7: $c_{k-1,j} = \frac{\mathbf{y}_j}{\sqrt{\lambda^{-2} + \zeta_{k-1,j}}}$
- 8: $u_{k-1,j} = c_{k-1,j}(\mu_{k-1,j} + b)$
- 9: $g_{kj} = \frac{c_{k-1,j}}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}u_{k-1,j}^2\right) \Phi^{-1}(u_{k-1,j})$ $\triangleright \Phi(x)$ is defined in equation 4.26
- 10: $v_{kj} = g_{kj}(g_{kj} + u_{k-1,j}c_{k-1,j})$
- 11: $\Delta\mathcal{H}_{kj} = -\frac{1}{2} \log(1 - v_{kj}\zeta_{k-1,j})$
- 12: $n_k = \underset{j \in \mathbf{J}}{\text{argmax}} \Delta\mathcal{H}_{kj}$ \triangleright Index from the point with highest \mathcal{H} reduction
- 13: $g_{n_k} = g_{k, n_k}$ and $v_{n_k} = v_{k, n_k}$
- 14: $m_{n_k} = \frac{g_{n_k}}{v_{n_k}} + \boldsymbol{\mu}_{k-1, n_k}$
- 15: $\beta_{n_k} = \frac{v_{n_k}}{1 - v_{n_k}\zeta_{k-1, n_k}}$
- 16: $\mathbf{s}_{k-1, n_k} = \mathbf{K}.\text{col}(n_k) - \mathbf{M}_{k-1}.\text{col}(n_k)^T \mathbf{M}_{k-1}^{-1}$ $\triangleright \mathbf{A}.\text{col}(i)$ returns the i th column of \mathbf{A}
- 17: $\boldsymbol{\mu}_k = \boldsymbol{\mu}_{k-1} + g_{n_k} \mathbf{s}_{k-1, n_k}$
- 18: $\boldsymbol{\zeta}_k = \boldsymbol{\zeta}_{k-1} - v_{n_k} \text{diag}(\mathbf{s}_{k-1, n_k} \mathbf{s}_{k-1, n_k}^T)$
- 19: Append $\sqrt{v_{n_k}} \mathbf{s}_{k-1, n_k}^T$ as new row to \mathbf{M}_{k-1} to get \mathbf{M}_k
- 20: Add $\mathbf{M}_{k-1}.\text{col}(n_k)^T$ as a row to \mathbf{L}_{k-1} and add the column vector $\left(\mathbf{0}^T, v_{n_k}^{-\frac{1}{2}}\right)^T$
- 21: Add n_k to \mathbf{I} and remove it from \mathbf{J}

determined, which is done in line twelve. The following lines update the site parameters and the matrices \mathbf{M} and \mathbf{L} , where the \mathbf{L} matrix is the lower triangle matrix used later in combination with vector \mathbf{m} to make an prediction. For the whole derivation of the algorithm 3 see "Extensions of the Informative Vector Machine" by Lawrence *et al.* [LPJ05].

The algorithm 4 shows the prediction procedure of an IVM. It only needs the active set \mathbf{I} , the lower triangular matrix \mathbf{L} , the vector \mathbf{m} and the bias b from algorithm 3. Furthermore the parameters d and λ are predefined and used again in here. In the first line of the algorithm 4 the new unseen point \mathbf{x}^* is presented. Between the third and the sixth line the kernel values for the unseen point in combination with all the inducing points are calculated and stored in \mathbf{k}^* . After that the μ^* and σ^* for the unseen point are calculated

and in the end the probability for belonging to the first class is returned. That means that the prediction only requires calculating the kernel between all points of the active set with the new point and evaluating it with the learned lower triangular matrix, which only needs $O(d)$ time steps. So the prediction only depends on the amount of selected inducing points.

Algorithm 4 The Prediction for the Informative Vector Machine

Require: $d, \mathbf{m}, \mathbf{L}, \mathbf{I}, b, \lambda$
 \triangleright Calculated in algorithm 3

```

1: procedure PREDICT( $\mathbf{x}^*$ )
2:    $\mathbf{k}^* = \mathbf{0}_d$  and  $\tilde{\mathbf{m}} = \mathbf{0}_d$ 
3:   for  $i = 1$  to  $d$  do
4:      $\tilde{\mathbf{m}}_i = \mathbf{m}_i + b$ 
5:      $\mathbf{x}_i = \mathcal{D}[\mathbf{I}_i]$   $\triangleright \mathbf{x}_i$  is the  $i$ th inducing point
6:      $\mathbf{k}_i^* = k(\mathbf{x}_i, \mathbf{x}^*)$ 
7:      $\mathbf{v} = \mathbf{L}.\text{solve}(\mathbf{k}^*)$   $\triangleright$  Solve correspond to a forward and backward substitution
8:      $\mu^* = \langle \tilde{\mathbf{m}}, \mathbf{v} \rangle$ 
9:      $\sigma^* = k(\mathbf{0}, \mathbf{0}) - \langle \mathbf{k}^*, \mathbf{v} \rangle$ 
10:    return  $0.5 - \frac{1}{2} \Phi\left(-\frac{\mu^*}{\sqrt{2\lambda^{-2} + 2\sigma^*}}\right)$ 

```

4.9 Hyperparameter optimization for the Gaussian Kernel

In section 4.4 different kernels for the IVM were explained, as mentioned there the Random Forest Kernel does not need any hyperparameters. However the standard Gaussian kernel does and the estimation of these parameters, defined in equation 4.18, is explained here. The first approach estimating hyperparameters on Gaussian Processes is always a Gradient Descent approach. To do that the marginal likelihood is calculating and derivate in respect to the parameters. Since the derivation of a long term of multiplications is harder than the derivation of terms of a sum the log of it is used. So the log marginal likelihood is defined in equation 4.51, where \mathbf{w} contains the parameters of the Gaussian kernel.

$$\ln(p(\mathbf{t} | \mathbf{w})) = -\frac{1}{2} \ln(\|\mathbf{K}\|) - \frac{1}{2} \mathbf{t}^T \mathbf{K}^{-1} \mathbf{t} - \frac{n}{2} \ln(2\pi) \quad (4.51)$$

This approach works fine for the toy example defined in section 1.1, by using the derivative of the Gram matrix in respect to the selected hyperparameter, as defined in equation 4.52.

$$\frac{\partial}{\partial \mathbf{w}_i} \ln(p(\mathbf{t} | \mathbf{w})) = -\frac{1}{2} \text{Tr}\left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \mathbf{w}_i}\right) + \frac{1}{2} \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \mathbf{w}_i} \mathbf{K}^{-1} \quad (4.52)$$

But in a more complex setting, where the hyperparameters influence the decision bound-

aries and even more the selection of the active set, this approach fails. So that means a simple gradient descent approach does not work any more. This can be understood by an example, in which the log marginal likelihood for a given problem like the MNIST-dataset is tried to optimize. Since the log marginal likelihood $\ln(p(\mathbf{t} | \mathbf{w}))$ is non convex there is more than one maximum, so the algorithm is started with a random sampling in the three dimensional search space. And after that a gradient descent approach is used. In this thesis the Adagrad approach was used, because there is no need of tuning the learning rate [DHS11]. It is defined in equation 4.53, the learning rate η can usually be set to 0.01 and the matrix $\mathbf{G} = \text{diag}(\mathbf{g})$ is just a diagonal matrix of the sum of all old squared gradients $\mathbf{g} = \sum \partial \mathbf{w}^2$. The epsilon only ensures that the root is always bigger than zero.

$$\mathbf{w}^{\text{new}} = \mathbf{w} - \frac{\eta}{\sqrt{\mathbf{G} + \epsilon}} \partial \mathbf{w} \quad (4.53)$$

The problem of the gradient descent approach in combination with the informative vector machine is illustrated in figure 4.10. It shows how the log marginal likelihood behaves for the hyperparameter optimization of the third number in the MNIST-dataset.

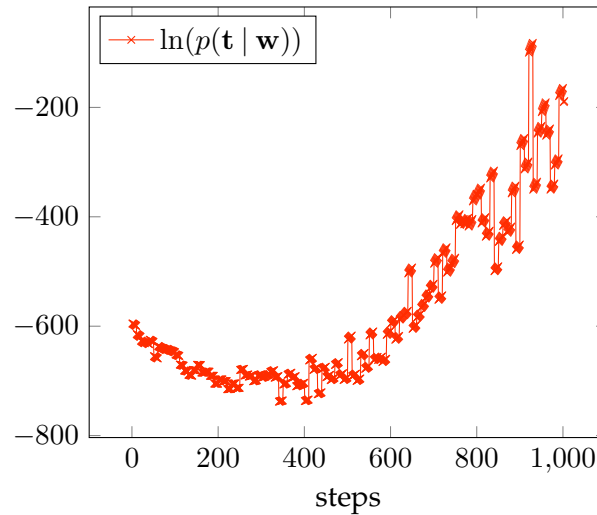


Figure 4.10: Optimization of the log marginal likelihood for the number three of the MNIST-Dataset in a one vs. all fashion. Following the gradient always maximizes the log marginal likelihood. However it changes the parameters slightly and after a few steps the active set is calculated again, which then produces a jump in the log marginal likelihood. In the first 200 steps the objective is even dropping, because of the changing active set. Important to note is that the biggest jumps are generated by the changing of the active set, not of the actual gradient descent approach.

The Adagrad approach is used and the gradients are subtracted from the randomly sampled best starting position. For ten steps the gradient is followed without changing the active set and in this ten steps the marginal likelihood always increases. However after

ten steps the active set is calculated again and in the most cases that changes the used induced points, which then changes the log marginal likelihood. In figure 4.10 it can be seen that these jumps are not always going up in the first 200 steps the direction is more down than up. Important to note is that the big changes in the objective are not generated by following the gradient, but more of the changing active set. It can be seen that only very small changes are performed on the log marginal likelihood, since the points in the ten steps only slightly move upwards. This behavior is more clearly visible at the end of the optimization. The problem is, moving with bigger steps along the gradient would also generated bigger jumps in the log marginal likelihood. So the gradient descent approach for the informative vector machine on more complex problems does not always yield a good result. This was also shown in the paper "Fast Forward Selection to Speed Up Sparse Gaussian Process Regression" from Seeger *et al.* in 2003 [SWL03]. However this was just the regression case, where the calculation of the posterior and therefore the log marginal likelihood is much easier.

Therefore an other approach was used in this thesis, instead of following the unreliable gradient a sampling approach was used. At first a simple sampling approach over the parameter space was tried. However finding good parameters by just sampling randomly takes a lot of computational power. Furthermore the result of the log marginal likelihood is not used at all. There are many different black box sampler, which utilize the function value. The one chosen for this thesis is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), because it is simple, fast and does not need many evaluations of the objective function [HO96].

4.9.1 CMA-ES

The standard CMA-ES consists of three steps. First new points are sampled, these points are then sorted by their fitness, in this case by the log marginal likelihood and after that the internal state variables are updated. This corresponds to the normal behavior of evolution strategies (ES), where new candidate solutions are sampled from a multivariate normal distribution. A new mean of this distribution is selected by recombining the samples and the generation of new samples is done by slightly moving existing samples with a zero mean Gaussian. The pairwise dependencies between two points are then used to generate a covariance matrix. The covariance matrix adaption (CMA) is then used to update the covariance matrix of the distribution. This approach would only need the similarity between two points is needed. In this thesis the euclidean distance between two solutions is used, to express this relation.

In algorithm 5 the CMA-ES is defined. It starts with the initialization of the used parameters. Important here is the parameter λ , which is the population size, describing the amount of samples generated in each iteration. It is the only parameter, which has to be chosen in the CMA-ES sampling. Except for the σ , which defines how big the search space is. The parameter n is again the amount of data points used in this example. The mean and the covariance are initialized and the isotropic and anisotropic path are set to zero. The

Algorithm 5 CMA-ES

Require: λ, n, σ

- 1: Init $\mathbf{m}, \mathbf{C} = \mathbf{I}, \mathbf{p}_\sigma = \mathbf{0}, \mathbf{p}_c = \mathbf{0}, \mu = \lambda/2$
 - 2: **for** $i = 1$ in μ **do**
 - 3: $w_i = \ln(\mu + \frac{1}{2}) - \ln(i)$
 - 4: **while** Not terminated **do** ▷ Need a terminate condition
 - 5: **for** $i = 1$ in λ **do**
 - 6: $\mathbf{x}_i = \mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})$ ▷ Sample a new point from $\mathcal{N}(\mathbf{m}, \sigma^2 \mathbf{C})$
 - 7: $f_i = \text{fitness}(x_i)$ ▷ Evaluate new point
 - 8: Sort \mathbf{x} to $\tilde{\mathbf{x}}$, so that the first has the highest f_i
 - 9: $\mathbf{m} = \sum_i^\mu w_i \tilde{\mathbf{x}}_i$ ▷ Use only the μ best of the sampling
 - 10: updatePs($\mathbf{p}_\sigma, \sigma^{-1} \mathbf{C}^{-\frac{1}{2}} (\mathbf{m} - \mathbf{m}_{\text{old}})$) ▷ Update the isotropic evolution path
 - 11: updatePs($\mathbf{p}_c, \sigma^{-1} (\mathbf{m} - \mathbf{m}_{\text{old}}), \|\mathbf{p}_\sigma\|$) ▷ Update the anisotropic evolution path
 - 12: updateC($\mathbf{C}, \mathbf{p}_c, \{\sigma^{-1} (\mathbf{x}_1 - \mathbf{m}_{\text{old}}), \sigma^{-1} (\mathbf{x}_2 - \mathbf{m}_{\text{old}}), \dots, \sigma^{-1} (\mathbf{x}_\lambda - \mathbf{m}_{\text{old}})\}$)
 - 13: updateSigma($\sigma, \|\mathbf{p}_\sigma\|$) ▷ Update the step size
 - 14: **return** \mathbf{m}
-

lines two and three describe how the weights for the mean are calculated. After that the real optimization starts, at first some samples are drawn from the multivariate Gaussian and evaluated in the fitness function. After the sampling the samples \mathbf{x} would be sorted in descending order after the corresponding fitness values. And then the first half of them would be combined with the weights from the beginning to get the new mean \mathbf{m} . After that the two path are refined. There are two paths to get a better grasp of the underlying function, they contain information about how to consecutive steps are related to each other. The first path is used for the covariance matrix adaptation step and the other path is used as an additional step-size control. This control tries to make sequential movements of the multivariate Gaussian orthogonal in expectation. This prevents effectively premature convergence and still it allows fast convergence to an optimum. At last the covariance matrix is updated. These update steps are done in the lines 10 to 13, the complete update steps are covered in "A Restart CMA Evolution Strategy With Increasing Population Size" by Auger and Hansen [AH05].

4.9.2 CMA-ES for hyperparameter optimization

The CMA-ES is used to estimate the hyperparameters of the Gaussian kernel. The estimation of three parameters can be very efficiently done with the CMA-ES approach, for that the external library "c-maes" is used. It provides the algorithm defined above in a numerical stable way [Han14].

In figure 4.11 the population development for the CMA-ES on an IVM used on the data

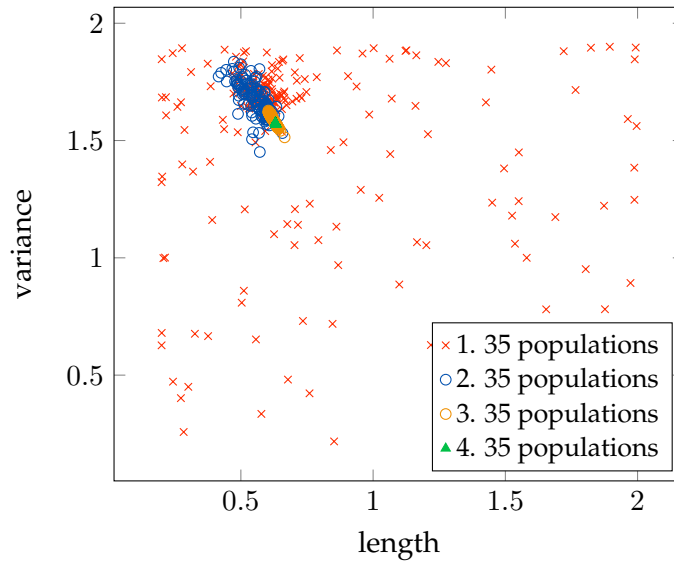


Figure 4.11: The population development of the CMA-ES for the example from section 1.1 is shown in this figure. The red circles depict the first 35 population samples, in blue the second 35 populations are illustrated. Each population consists out of five samples. The area of the blue samples is already much smaller than the initial samples. This behavior is the same for the circles in green and orange, which are the last two groups of populations.

set defined in section 1.1 is shown. The first 35 populations are sampled in red, these cover the whole search space. Each of the populations consist out of five samples and an update step as described in algorithm 5. In the next 35 population steps the area covered by them is already much smaller, the samples of it are depicted in blue.

Figure 4.12 shows the development of the objective function. The points are again samples of the CMA-ES and the color indicates the objective function with a linear scaling, where as the yellow dots have the worst objective value and the blue ones the best. The background is calculated by a k-nearest neighbor search. In figure 4.12 a k of seven was used and it illustrates, which values the background could have at this positions. The figures 4.11 and 4.12 indicate the fast convergence of the CMA-ES for the hyperparameter optimization for the Informative Vector Machine.

4.10 Enhanced error measurement

The CMA-ES for the hyperparameter optimization relies on the quality of the log marginal likelihood, which was defined in equation 4.51. However the log marginal likelihood does not always yield to the best result. This can have several reason. One of them could be that the best solution lies inside of a field of unattractive samples. An other could be that the log marginal likelihood is low, but the representation of the data is not the

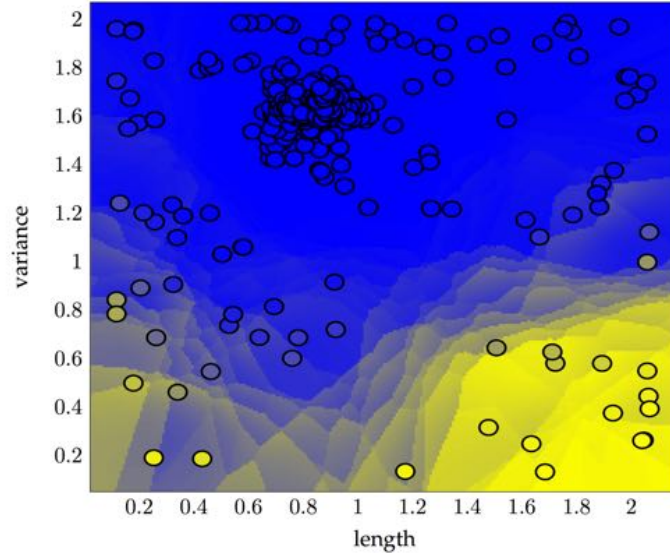


Figure 4.12: The error term for the samples of the CMA-ES is plotted here. A linear scaling is used, to show how the search space is processed, where yellow is the worst value and blue is the best. The background is the combined value of the k -nearest neighbors, where k is seven in this image.

best possible. This can be seen in figure 4.13, which shows the result of the optimization on the basic log marginal likelihood, all values are classified correctly. But the average probability belonging to the blue class is 0.6034 and the average probability for the yellow class is 0.5604 in this image. In a perfect trained model it would return for both an average probability close to 1.0.

This means the log marginal likelihood, can be further improved. In order to do that this thesis combined the log marginal likelihood with the misclassification error on a test set. This is defined in equation 4.54. To compare this values it is necessary to divide the log marginal likelihood by the number of inducing points, so that there is no correlation between the amount of points and the result. That means in the beginning of the optimization the log marginal likelihood is optimized, because in the beginning the average error for the prediction of the test set is random and will be around 0.5. The average error is calculated by predicting for each data point the probability, that it corresponds to the right class. At the end the term is multiplied by 100 to make it more comparable to the error. The value 100 and the division by d are values, which were defined by experimenting.

$$e_{new} = \underbrace{\frac{1}{d} \ln(p(\mathbf{t} | \mathbf{w}))}_{\text{log marginal likelihood}} + \underbrace{\frac{100}{m} \sum_{i=1}^m 1 - p(y_i^* = y_i | \mathbf{x}_i, \mathbf{t}, \mathbf{w})}_{\text{misclassification error}} \quad (4.54)$$

One further step is to expand the misclassification error in a binary setting to equation 4.55.

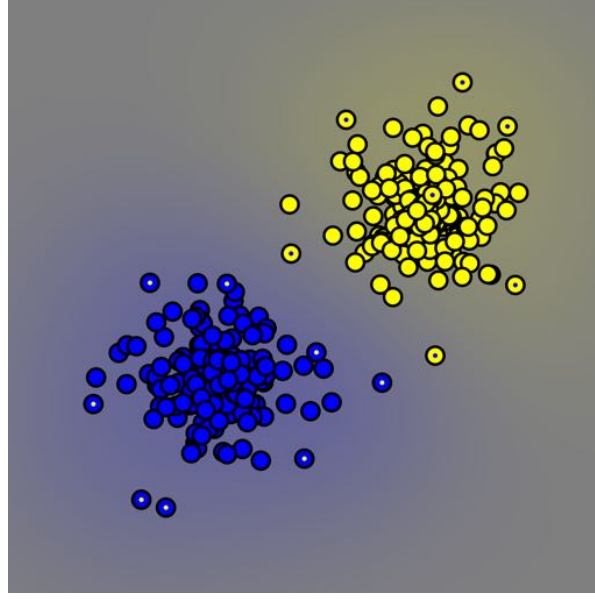


Figure 4.13: One possible result for the informative vector machine if the log marginal likelihood is solely maximized. All of the points are classified correctly. However the over all certainty is low.

In this the misclassification rate is calculated for both classes $(-1, 1)$ and then the average is computed. Here m_1 is the amount of points of the class 1 and m_{-1} is the amount of points for the other class, with $m_1 + m_{-1} = m$. The average avoids that a unequal share of the classes leads to a wrong result, for example the case $m_1 \ll m_{-1}$ would favor the correct classification of the -1 class, but would ignore the classification for the 1 class. This can be expanded even more, by adding a weight value θ to the formula, resulting in equation 4.56. Instead of just combining both rates a weighting between them can be chosen. In order to avoid the favoring of a result, in which the predictive distribution always returns a high probability for elements which belong to the 1 class and a low probability for the elements of the -1 class. In this case the weighting could favor the -1 class more so that the training optimizes both at the same, without disregarding one of the terms.

$$e_{avg} = \frac{1}{2m_1} \sum_{i=1}^{m_1} (1 - p(y_i^* = 1 | \mathbf{x}_i, \mathbf{t}, \mathbf{w})) + \frac{1}{2m_{-1}} \sum_{i=1}^{m_{-1}} (1 - p(y_i^* = -1 | \mathbf{x}_i, \mathbf{t}, \mathbf{w})) \quad (4.55)$$

$$e_{avg}(\theta) = \frac{1}{\theta m_1} \sum_{i=1}^{m_1} (1 - p(y_i^* = 1 | \mathbf{x}_i, \mathbf{t}, \mathbf{w})) + \frac{1}{(1 - \theta) m_{-1}} \sum_{i=1}^{m_{-1}} (1 - p(y_i^* = -1 | \mathbf{x}_i, \mathbf{t}, \mathbf{w})) \quad (4.56)$$

This improved error measurements furthermore can be used later in the online evaluation, because already an evaluation of the Informative Vector Machine was performed and should be used at as many points as possible. By doing so the amount of reevaluation can

be reduced.

4.11 Improved selection of the active set

In figure 4.14a the predictive distribution of the Informative Vector Machine was shown. The selection criteria for the IVM was defined in section 4.8. A drawback of this simple approach can be described with the help of the figure 4.7b. The right upper corner of the image does not contain any inducing point, which means this area is not that well represented. But at the border a lot of inducing points are used, which means that the selection criteria does not always find the best representation. Furthermore the representation does focus too much on finding a separation to other data points and not a separation to non-data. Figure 4.14b contains the Informative Vector Machine with an improved approach

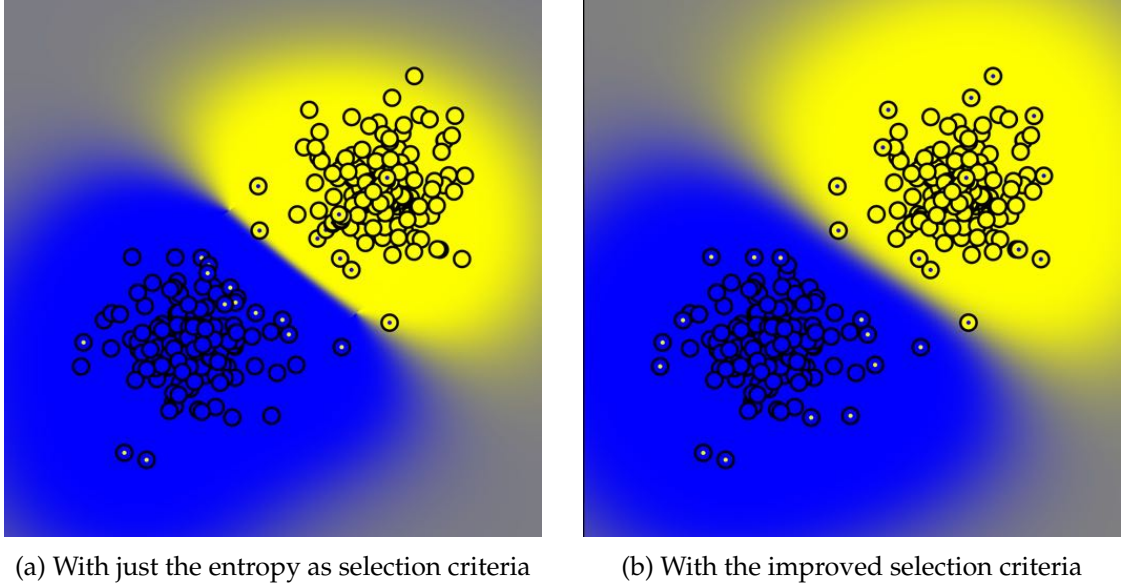


Figure 4.14: Both pictures contain an IVM with 28 inducing points. However in the left picture the original selection criteria from Lawrence et. al was used [LSH⁺03]. In the right picture the result of an improved version is shown. This approach suppress points which are too similar to points from the active set, which can then later be used to reduce the number of points.

for selecting the next point. It results in a far more spread representation of the data points, which does not only rely on representing the border well. This can be achieved by suppressing points, which are similar to points from the active set. The improved selection criteria is defined in equation 4.57.

$$\mathcal{H}^{\text{new}}(\mathbf{x}) = \underbrace{\mathcal{H}(\mathbf{x})}_{\text{original information gain}} - \sum_{i \in \mathbf{I}} \underbrace{k(\mathbf{x}, \mathbf{x}_i)}_{\text{weight}} \mathcal{H}^{\text{new}}(\mathbf{x}_i) \quad (4.57)$$

The new selection criteria consists of maximizing the new information gain $\mathcal{H}^{\text{new}}(\mathbf{x})$ for points, which are not in the active set \mathbf{J} . The improved information gain is the old information gain minus the weighted sum of the information gain of the active set. The weighting decides how much a point of the active set \mathbf{I} influences the information gain of the current point \mathbf{x} . Using the kernel is the natural choice, because it already provides an excellent measurement of the similarity between a new point and the points of the active set. It is also important to note, that the results of the kernels do not have to sum up to one, because only the maximum for $\mathcal{H}^{\text{new}}(\mathbf{x})$ is important. The overall maximum stays the same if all points of \mathbf{J} are punished in the same way. This can be assumed, because the used kernels are valid Mercer kernels [Mer09].

In order to compare the old with the new selection criteria a binary classification was selected. The MNIST dataset was used, it is described in section 7.1. In this example the handwritten zero was trained against the rest. Both approaches were trained on all 60,000 points of the MNIST training set and were tested on the 10,000 test data points. The active set \mathbf{I} has a size of 800 in both cases. Table 4.1 shows the results for the new and the old approach. The two rows show the results for the old and the improved approach. The amount of correctly classified points rise dramatically with the improved selection criteria, because with the original one the amount of inducing points is not enough. This can be seen in the first column of the table. However the second column shows the average time an optimization step for calculating one IVM for a new hyperparameter configuration needs. By using the improved approach the time needed for an fast IVM step goes up. Furthermore the times for the original approach vary far more than for the improved approach. With the original method wrong parameters need around 50 seconds, better parameters in comparison only need around 30 seconds. But the amount of good parameters is very low in the original method, so more sampling steps are needed and furthermore the best result isn't as good as the one from the improved approach. Both IVMs only check randomly around 200 points for choosing the next inducing point and not all of \mathbf{J} . This can be achieved by choosing a random number between 1 and 100 and add this to the current index. This means that in general around 200 points are checked. By doing that the speed can dramatically improved otherwise the training would need around 20 minutes per optimization step. Checking only 200 points is not as good as checking $60,000 - \|\mathbf{I}\|$, but the influence is not as strong and the training can be done in manageable time. After the optimization of the hyperparameter the whole dataset is used, to choose the best possible active set \mathbf{I} . The last two columns show the recall for the original and new selection criteria. The original method does not capture all zero numbers and misclassify some of them, where as in the new method the recall for both is equally high and therefore a better splitting was found.

	correct classified	time for IVM optimization	Recall for 0	Recall for rest
Old approach	91.27 %	45.21 sec	90.3750 %	99.3212 %
New approach	98.39 %	53.80 sec	98.4694 %	98.3814 %

Table 4.1: This table compares the original proposed approach from Lawrence et. al with the improved approach presented in this thesis [LSH⁺03]. It shows the advantages of the new method in comparison to the original method on a binary classification task on the MNIST dataset. Here the handwritten zero was trained against the rest.

4.12 ADF, EP and the active set

The Expectation Propagation has one major advantage over the Assumed Density Filtering. The approximation of the EP does not depend on the order of the points, which is a drawback in the ADF. However through a good selection approach this drawback can be reduced to a point where the difference to the Expectation Propagation algorithm gets small. This could be achieved by different factors the first one is the improved selection criteria explained in section 4.11.

Additionally instead of selecting always the point with the highest reduction in entropy independent of the class. The point with the highest information gain, which also balance out the classes in the active set \mathbf{I} can be used. This means the next used class c_{next} can be calculated with equation 4.58. For the first inducing point ($\|\mathbf{I}\| = 0$) a random class is chosen. For all points after that the amount of inducing points belonging to a certain class is compared to the desired fraction v for the 1 class. It is defined in equation 4.59. The desired fraction for the 1 class in the active set \mathbf{I} is the fraction of the data set, which belongs to the 1 class. In order to avoid that the fraction gets too small it is averaged with 0.5. This helps in situation where not many data points are used for the class 1. The norm of a set $\|\mathcal{D}\|$ returns the number of elements and the amount of points in a set of the 1 class is $\|\mathcal{D}\|_1$.

$$c_{\text{next}} = \begin{cases} 1 \text{ or } -1 & \text{if } \|\mathbf{I}\| = 0 \\ 1 & \text{else if } \frac{\|\mathbf{I}\|_1}{\|\mathbf{I}\|} > v \\ -1 & \text{else if } \frac{\|\mathbf{I}\|_{-1}}{\|\mathbf{I}\|} > 1 - v \end{cases} \quad (4.58)$$

$$v = \frac{1}{2} \left(\frac{\|\mathcal{D}\|_1}{\|\mathcal{D}\|} + \frac{1}{2} \right) \quad (4.59)$$

Instead of using the Expectation Propagation algorithm after the Assumed Density Filtering. It is also possible to use another ADF, where the active set \mathbf{I} is fixed and the points from the active set (\mathbf{x}_i with $i \in \mathbf{I}$) are processed in a flipped order. Without the search for

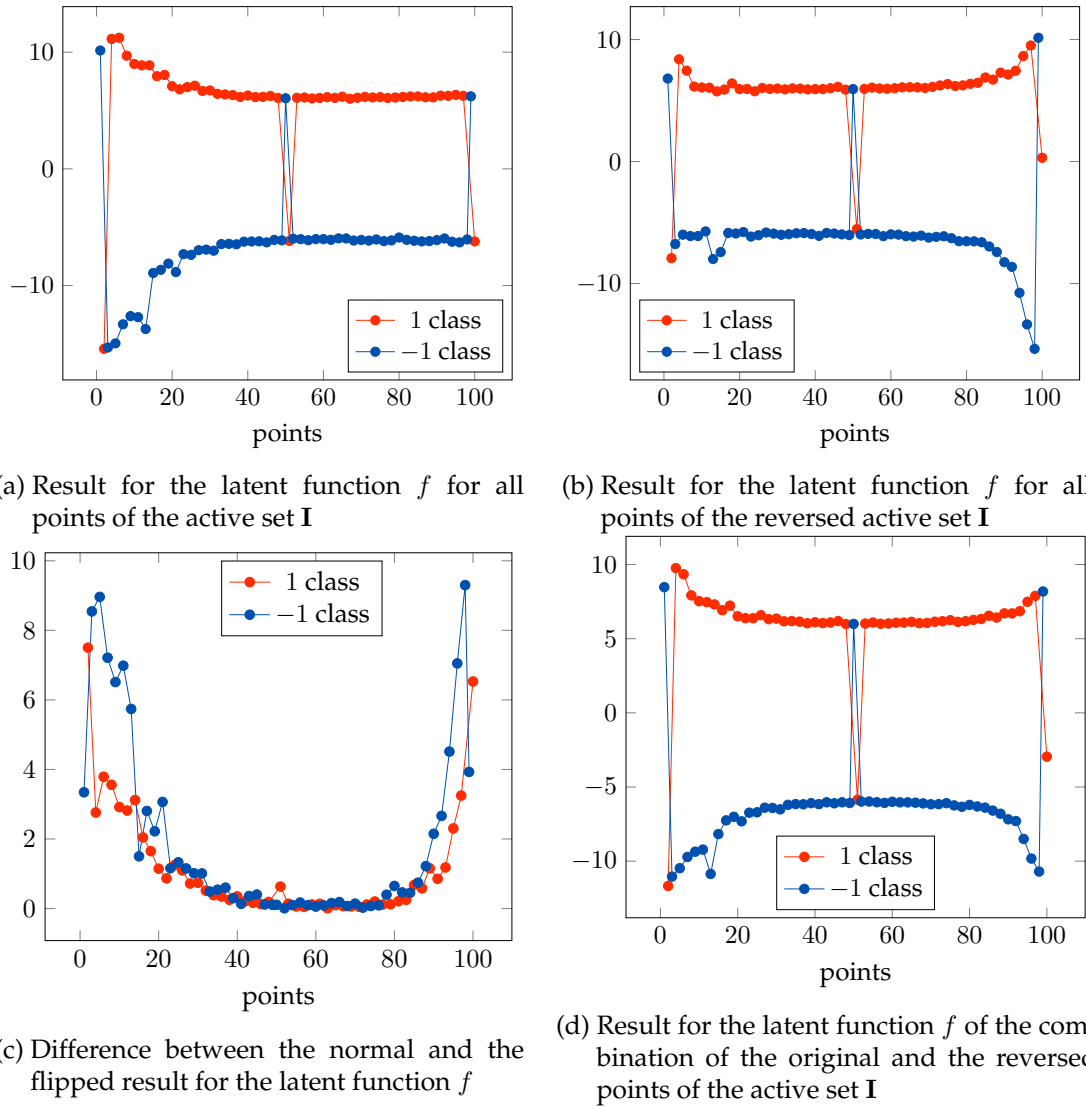


Figure 4.15: This four plots show the different results for the latent function f after the training. They have different colors for the different classes. The 1 class is red and the -1 class is blue. The left upper figure shows the result for the initial training of the ADF. The picture in the upper right shows the result for the reversed active set \mathbf{I} . In the lower left the absolute difference between both is plotted and to the right of it the combination of both is illustrated.

the active set the calculation of the Informative Vector Machine is ten times as fast, which means this does not influence the trainings time. However the calculation of the flipped active set is not always possible. If it is possible the latent functions are combined with the values from the other order. This is visualized in figure 4.15. The upper left figure 4.15a shows the latent functions result from the ADF on the example from section 1.1 with 100 inducing points. It shows that the latent functions for the first values varies the most. This

is caused by the unstable posterior estimate from the ADF and the bad selection of the first values, because the selection criteria depends on the already seen points. The right upper figure 4.15b contains the result of the latent functions for the reversed active set. In order to compare both the order was flipped for presentation purposes. Again the last points in this figure, which were selected in the first iteration of the IVM nearly randomly, produce the most noise. The difference between the normal and the reversed is plotted in the lower left figure 4.15c. At the beginning and at the end the values are the most different, which can now be smooth through the combination of both results. The combination is shown in figure 4.15d, it reduces the overall noise.

With this approach there is no need for the expensive Expectation Propagation, which then makes the training of the Informative Vector Machine by a factor of seven to ten faster.

4.13 Multiclass scenario

In the sections so far the binary case was covered. This means only two classes were used, however the most problems in machine learning consists out of more than two classes. In order to achieve a multiclass algorithm between two approaches can be chosen.

The first approach is to expand the Informative Vector Machine to a multiclass approach, where there are more than two classes $(-1, 1)$. One advantage is that the computation can be done in a block matrix fashion. However the disadvantage is that the inducing points have to be split through the existing classes, which decreases the amount of information per class. This can be solved by increasing the amount of inducing points nonetheless the computation time grows quadratically with the amount of inducing points. Furthermore the kernel can now only carry one set of hyperparameter, which decreases the effectiveness of the kernel. Therefore this approach wasn't used in this thesis.

An alternative to this is using instead of one giant Informative Vector Machine, several small binary trained IVMs. These are trained in a one vs all fashion, this means in a scenario with C classes, C IVMs are trained. Each of them gets its own class and is trained against the other $C - 1$ classes. These IVMs then have through the training process their own best hyperparameter set and can therefore better express the difference between this class and the rest of the classes. The biggest disadvantage of this approach is that if one of the IVMs can not find good hyperparameter and always returns a probability of one for a point belonging to its class. Then the whole result is bad. The reason for this is that each point gets the class from the IVM with the wrong hyperparameter.

So in order to get good results, the parameters of all sub IVMs have to be trained carefully as shown in section 4.9.

5 Online Random Forest with Informative Vector Machines

The last two chapters described two different approaches to get a solution for a classification problem. The Online Random Forest is a fast and adaptive learning procedure. The Informative Vector Machine builds up a full probabilistic model instead of estimating one, but it can't handle big data sets. In order to get a better algorithm both algorithms are combined here. The resulting algorithm can handle big streams of data and still builds up a probabilistic model on the data.

The combination of both can be achieved by using the Random Forest as a partition approach of the input data set. That means that the Random Forest splits the data set in sub splits. Then a multiclass IVM is used on the resulting splits of data, which are smaller than the complete data set. This reduces the points used in the IVM approach and furthermore improves the results of the splits of the Random Forest. This idea is a basic one in computer science and is known as divide and conquer. The reason for that is that the IVM can not handle big amounts of data, because the calculation of the kernel on so many elements takes too long. However after the problem is split in some smaller subproblems IVMs can be used to find a good solution for it. In theory the Random Forest should be able to classify all training points, as long as the information is given in the data. However through limitations like the memory space or trainings time the optimum can not always be reached.

The Random Forest with the Informative Vector Machine is used in figure 5.1. In the upper part of the picture are four datasets illustrated. There are two yellow classes, which are plotted as stars and two blue classes, which are represented as circles. The assumption is that similar shape and color means that the classes are harder to separate and that the classes it self represents something similar. This could be for example in a classification setting for pictures, that the yellow ones are animals like cats and dogs and the blue ones are tools like a power drill or a screwdriver. Both classes have similar features like the fur for the animals and plastic on the tools. The Random Forest is now used to separate the color and shape structure of the problem. In figure 5.1 this is shown by a black line, which resembles a Decision tree with just one Decision node. Both of the split datasets are then further processed by the Informative Vector Machine. The result of that is shown in the lower part of the picture. As before the background of the image shows the membership of belonging to one of the classes. Furthermore for a better separation the lines for the different probabilities were drawn too, where as the line in the middle shows the points, which have a probability of 0.5 belonging to one of the classes.

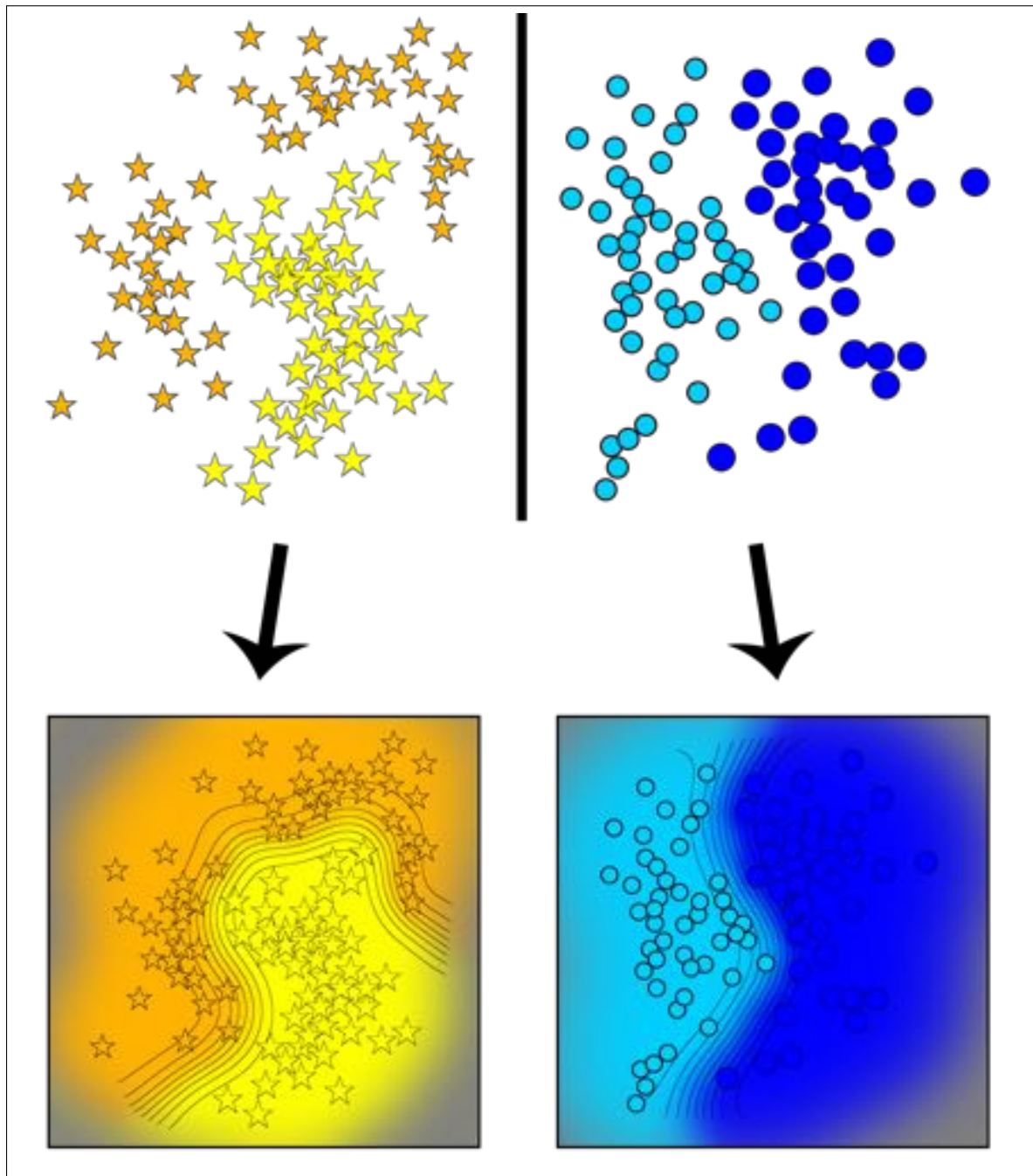


Figure 5.1: The upper part of the figure shows a data sets with four classes. Two blue ones and two yellow ones, each pair is kind of similar. In this example only one decision tree with one node was used. The separation generated by this tree is represented by the black line between the four sets. On the left and the right set an Informative Vector Machine is now trained and the result is shown in the lower part of the figure. The decision boundaries for different probabilities are drawn in too. Both approaches combined makes it possible to separate these four data sets.

5.1 Online learning

In order to make this whole approach online, first an offline step has to be performed. In which the first Random Forest is trained like described in section 3. After that the points are separated after the prediction of the Random Forest. If now all points are classified correctly then the algorithm has finished. However this is only the case if the dataset is small or too simple for this online approach. In an usual application not all points are classified correctly and the points, which belong to the same predicted class are combined and given to a multiclass Informative Vector Machine. This combined points consist of many samples of the true class and the points, which were classified by the Random Forest as this class label. These wrong classified points are in general outliers. So the trained Informative Vector Machine is mainly trained with the true class and some outliers to make the borders of the class clear to the algorithm. So that if a new point comes in the Random Forest can first decide, which Informative Vector Machine should be used. Afterwards the right IVM tells then if the points belongs to this class or not.

After this offline step has be done the approach get a new batch of data, which is first classified by the Random Forest and then the points are tested on the Informative Vector Machine. If the amount of correct values fall under a certain threshold, the Informative Vector Machine is trained again with the same hyperparameter. If the final result is then above the threshold the next Informative Vector Machine is evaluated until all of them are processed. However if one of them does not get above the threshold after the retraining the hyperparameter are trained again, which takes far for my time and can be considered as an offline step in the online learning. If the approach is used in an online fashion, where the prediction shouldn't be stopped, then the IVM could optimize the hyperparameter in a separate thread or could do that at later point.

However training of this approach is difficult, because of the unstable behavior of the Informative Vector Machines on the predictive class data sets.

5.2 Biggest challenge

From the explanation of the Online Random Forest with Informative Vector Machine the biggest challenge can be deduced. A Random Forest separates the data set into different subset, which are most likely not pure. This means each predicted class contains some values, which are not from the true class. These few points, which classified incorrectly are mostly outliers, which are already in some way similar to the true labels. Separating these points is therefore especially difficult. That also means that the selection of the hyperparameter is more challenging than for a usual problem, because these outliers could miss the necessary information in their feature vectors to classify them correctly.

6 Implementation Details

6.1 Online learning

In order to get an easy to use and simple designed online learning approach. The underlying classes have to be designed in way that from the beginning the data is a container which supports online updates.

In order to do that a class was designed, which implements the observer pattern [Gam95]. In which the online storage is the subject and each learning approach is an observer, which gets notified if any change on the storage is made. These notifications in general contain an add event. With this information an online approach like the Online Random Forest can then reevaluate their result on the new set. From that the Random Forest can consider if a new training is necessary. The great thing about the observer pattern is that an append to the existing array is enough to trigger the whole background update of the model. This is implemented in all existing models and combinations in this thesis. That means the same Online Storage can be used in an IVM and a Random Forest and both of them would be start their training if necessary, after the append function was called. This procedure makes the algorithm easy to use in an online scenario, because the algorithm has the necessary structure to update its own during the arrival of new data. Otherwise the user of this approach would always have to trigger the training by himself and have to plan in advance when an online step is performed or has to decide in which intervals an offline training step is necessary.

6.2 Thread Master

In order to guarantee that the learning is done online and as fast as possible, some control mechanisms have to be in place to ensure that the right thing is learned at the right time. For that each training and prediction step is performed in a single thread. The execution of this threads are controlled by the so called Thread Master. It decides, which training step is more important than an other.

This problem arises through the fact that in this thesis several approaches were combined. The combination of the Online Random Forest with the Informative Vector Machine leads to a splitting in the dataset, which changes the requirements for the IVMs during the training. In this section the focus is more on the processing of the problem than on the original splitting. As mentioned before the splitting of the input data set is done by the initial training of the random forest, where as each available core on the machine is used

for the training of the trees. This can training process can be easily separated into multiple threads without any problems. It would be even possible to train that on different machines and just combine the resulting trees in a new forest. This is possible, because the training of one tree is independent of the rest of the trees. In the case that only an Online Random Forest is used, there would be no need for a Thread Master.

But in combination with the Informative Vector Machines the simplicity vanishes. If the amount of points in a split of the trainings data generated by the ORF is high enough and has furthermore enough samples, which were classified wrong. A multibinary IVM is generated with this dataset, which adds a new thread to the Thread Master. This multibinary IVM then itself generates binary IVMs if they are needed. In this thesis the necessity for such a new binary IVM, was determined by the amount of available points in the data set of the corresponding class. In a multiclass scenario with several classes the amount of threads can grow very fast. Therefore it is important to manage the order of execution of these trainings threads. At first they are sorted by priority, the initial training of the Online Random Forest for example has a higher priority than an following IVM training. The reason for this is simple the first offline training has to be completed before any IVM training can be started.

After the offline training of the ORF is finished the training of the Decision Trees is done in parallel to the IVM. In order to decide, which thread runs next the attraction for a certain thread is calculated with the formula given in equation 6.1.

$$attr(c, p) = \begin{cases} \frac{p - p_{min}}{p_{max} - p_{min}} + c & \text{if } p_{max} > p_{min} \\ c & \text{else} \end{cases} \quad (6.1)$$

The attraction consists of the amount of correctly classified values c in percentage and the amount of used points p in this thread. Without further knowledge the assumption is that the amount of correctly classified points is as important as the share of the used points. So this means p_{min} is the minimum amount of points of any running thread at the moment and p_{max} is the maximum of amount of points of any running thread. If all running threads have the same amount of points just the correctly classified share is used. The Thread Master checks ten times per second if the running configuration of threads is the best to go. If one of the waiting threads has a lower attraction value than one of the running threads it is replaced. To avoid that the threads are changed to often a certain amount of time is waited until a thread can be put on the waiting list again. In this thesis each thread was always guaranteed to train for at least two seconds. For a smooth and controlled running of the Thread Master each new thread got an Information Package, which needed to get live updates from the runnings threads. So that the Thread Master can make an informed decision on the actual state of the system.

6.2.1 Multithreading

In order to provide a fast and dynamic algorithm the whole training and update process has to be parallelized. In this thesis a computer, which supports multithreading and which can run up to eight threads in parallel, is used.

Online Random Forest Then in the Random Forest approach this means that eight trees can be trained at the same time. This is possible, because the training of a single Decision trees does not depend on the training of an other tree. The biggest problem for training decision trees in parallel is to guarantee that the used random number generators are deterministic. So that each execution with the same random seed leads to the same result. This can be solved by using a different random generator for each thread. So in the case that the same amount of trees is generated in each thread then the resulting trees should all be the same. However if the a few thousands to million trees are trained and in the end the training is ended. It can happen that the amount of trees generated by a single thread vary and therefore the result changes slightly. To avoid this a fixed number of trees in each thread can be calculated, to achieve a complete deterministic learning approach.

For the online update this process gets more complicated, because all threads work on the shared list of Decision trees, which is sorted after their performance. So adding and removing have to be synchronized, this was implemented by an Insertion Sort. This means a tree is added by walking over the list of trees and adding it in front of the first tree, which performance is better than the one newly generated. This adding operation takes in the worst case $O(n)$, where n the amount of trees in the list is. To speed up the whole process even more the initial sorting process of the trees on the new combined set can be done in parallel too. At first a eight threads are started, which first split the existing trees evenly under one each other. After that each thread evaluates the given trees for the combined data points and then the list are combined with a merge sort algorithm.

Informative Vector Machine The steps of a binary IVM can not be easily parallelized, because the assumed-density filtering is used and each approximation directly depends on the result of the last iteration. However for a new batch of data important points could be calculated and these are then added to the approximation. This is not done in this thesis, because the recalculation time of only a binary single Informative Vector Machine with known hyperparameters is fast.

7 Examples

In order to verify the implemented algorithms different commonly known datasets were tested.

7.1 MNIST

One of the most commonly known datasets is the MNIST dataset [LBBH98]. It contains handwritten digits, which can be seen in figure 7.1. There are 60,000 training images and 10,000 test images. Each class has a similar amount of pictures in the training and test set. A picture consists of 28×28 pixels, where each pixel is a grey value from 0 to 255. These pictures are then transformed into a column vector and this vector with 784 dimensions is used as an input vector. However it is important to note that through the vectorization some local information of the picture is lost. Nonetheless the presented approaches achieve good results on this dataset.



Figure 7.1: This picture shows some random handwritten digits from the MNIST dataset [Eic14]. All of them have 28×28 pixels

To reduce the amount of processed data, these vector are filtered beforehand and all the dimension which don't carry any information are removed. Figure 7.2 shows in black all the pixels of all images, which don't change their value in any of the 60,000 training images and are therefore removed from all the training and test vectors. In grey the used pixels are illustrated. These procedure reduces the amount of dimension from 784 to 717, which reduces the memory usage from 439 MB to 401 MB. Each MB which can be saved reduces the trainings time. The data is clamped to the interval from 0.0 to 1.0, so that the kernel evaluation in the Informative Vector Machine case is easier.

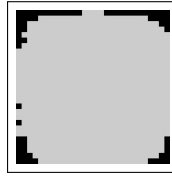


Figure 7.2: The black pixels shows the values of the MNIST images, which never changed. These 67 black pixels are therefore removed from the vectors. The grey areas illustrates which part of the images is later used in the algorithm.

7.1.1 Results for the Random Forest

The Random Forest has short training times. Table 7.1 shows the result for the test set in a one vs. all scenario. The amount of trees were limited by the trainings time. The trainings time was set to 30 seconds and the height was set to 35, which means the Deep Dynamic Decision Tree was used. In general around 800 trees were trained. The first row

Class	0	1	2	3	4	5
Error:	0.004	0.0027	0.0107	0.0123	0.0089	0.0122
OC:	0.0541	0.084	0.0483	0.0297	0.0259	0.0173
UC:	0.3805	0.2300	0.4985	0.5730	0.5320	0.5522
Class	6	7	8	9	Avg	Var
Error:	0.0061	0.0099	0.0154	0.0143	0.00966	0.00001
OC:	0.0613	0.0621	0.0210	0.0570	0.04607	0.00042
UC:	0.3843	0.4515	0.6336	0.5588	0.47944	0.01291

Table 7.1: The results for the binary case, one vs. all are shown here. The error is shown in the first row and the second and third contain the over- and underconfidence defined in equation 7.1 and equation 7.2.

of both tables contains the number of the handwritten digit, which was trained against all other classes. In the second row the overconfidence for the correct classified is shown. The overconfidence is defined as one minus the average entropy over the wrong classified ones [MTC15]. A value close to one would mean the approach is sure about the wrong classified ones. In this case the value is close to zero, which means the wrong classified ones can not be ordered to one of the classes. The last row shows the underconfidence, which is defined as the average entropy over the correctly classified points [MTC15]. A value close to one would mean that the prediction is correct, however the predictor is uncertain about the result. In this scenario the approach is not that sure for correctly classified points. The over- and underconfidence o and u are defined in equation 7.1 and equation 7.2 [MTC15]. The average of the whole table 7.1 is written in the lower table in the second last column,

on the right from it the variance for this data is printed.

$$o = \frac{\sum_i^n \mathbb{1}(y_i^* \neq y_i)(1 - \mathcal{H}(\mathbf{x}_i))}{\sum_i^n \mathbb{1}(y_i^* \neq y_i)} \quad (7.1)$$

$$u = \frac{\sum_i^n \mathbb{1}(y_i^* = y_i) \mathcal{H}(\mathbf{x}_i)}{\sum_i^n \mathbb{1}(y_i^* = y_i)} \quad (7.2)$$

The multiclass performance was measured too. In order to avoid overfitting the data was trained in an online fashion, which was described in section 3.4. The 60,000 data points were split into 10 sets of data and each of them were presented to the Online Random Forest. For the first data set an offline approach was chosen and it generated 2,002 trees with a depth of 35, which lead to a perfect fitting on the trainings data. That means each sample of the 10,000 points could be matched with the corresponding class. However on the test data only 91.84 % were classified correctly. This can be seen in figure 7.3. It shows the percentage of correctly classified points on the 10,000 points of the test data set. It was evaluated after adding a new batch to the already used training set and retrain the decision, which performed poorly on the new data points. By only adding the wrong classified points to the trainings set the result on the test set tends to go upwards after each iteration. However this must not always be the case, in this example each training steps increases the accuracy on the test data set. So a trend is clearly visible and therefore using an online approach to train the trees, improves the overall result to an overall error of 0.0382. The average trainings time for a new batch is 90 seconds.

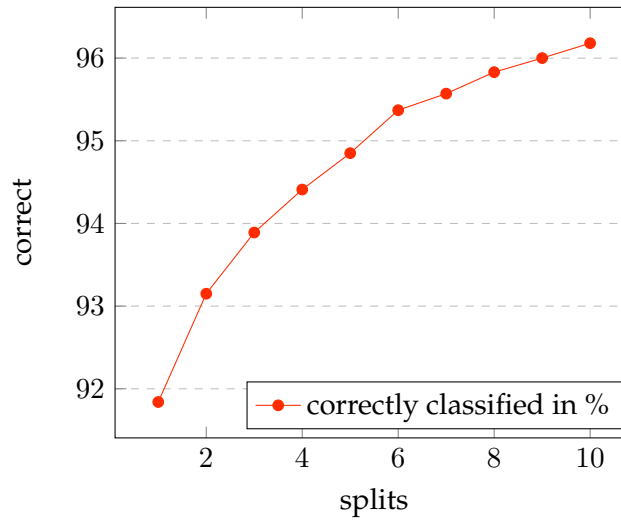


Figure 7.3: The result for the correct classified points on the 10,000 points of the test data set is illustrated in this figure. It is visible that by adding the wrong classified points of the new batch to the trainings set the amount of correctly classified points on test rises. However that must not always be the case.

The Random Forest is not as good as the best state of the art approaches. However

the Random Forest didn't use any local features on the images and only had the feature vector as an input, which decreases the amount of information presented to the algorithm. Furthermore it is an adaptive online learning approach, which can adapt more easily to new data points in comparison to existing algorithms. Therefore comparing just the final results is not justified.

Outliner detection

One important thing of a machine learning approach is the ability to recognize if a certain point does not belong to any of the learned classes. In order to test that all pictures of the handwritten digit five were removed from the training set and later on were tested on the algorithm. The goal is now that the algorithm is not confident on the result and that not always the same class for the unknown is returned, because that would mean that the classifier has overfitted. Table 7.2 shows the result of a Random Forest, which has 104 trees with a depth of 35. These Random Forest has never seen a picture of a number five and therefore this label was not used once. However all the other numbers are there at least once, the result often contains the three and the eight, which are already close to a five. The overconfidence after equation 7.1 is for this test data set 0.16356, which is close to zero which means that the Random Forest is not certain on the classification of the unknown points. If the Random Forest would be used in active learning setting, a human supervisor could be asked to classify these data points, because the algorithm is uncertain about them.

Class	0	1	2	3	4	5	6	7	8	9
#	115	61	7	311	55	0	48	34	217	44

Table 7.2: The results for the test data set are plotted here. All of them have the same true label, which is in this example five. The reason because no value of five is recognized is, because the Random Forest has never seen a five before. Important is that none of the classes is completely favored, which means that the algorithm has not overfitted.

7.1.2 Results of the Informative Vector Machine

At first the Informative Vector Machine is evaluated in a binary one vs. all fashion. The result for this is shown in table 7.3. As before for the Random Forest case each class is trained against the other ones. The error for this is in the first row of the table 7.3, the second and third row contain the over- and underconfidence defined in equation 7.1 and equation 7.2. The results are flipped in comparison to the Random Forest, where the overconfidence was low and the underconfidence comparatively high. This is in general a better result, because this means that the algorithm was sure on the correctly classified ones and just a little bit to sure for the very few falsely classified points. The error for example for the first class is 0.004, which means only 40 pictures were classified incorrectly. On these points the

algorithm was too sure, but compared with the certainty on the correct ones is this not so bad.

Class	0	1	2	3	4	5
Error:	0.0048	0.004	0.0107	0.0123	0.0089	0.0122
OC:	0.8685	0.5022	0.0483	0.0297	0.0259	0.0173
UC:	0.0011	0.0090	0.4985	0.5730	0.5320	0.5522
Class	6	7	8	9	Avg	Var
Error:	0.0061	0.0099	0.0154	0.0143	0.00966	0.00001
OC:	0.0613	0.0621	0.0210	0.0570	0.04607	0.00042
UC:	0.3843	0.4515	0.6336	0.5588	0.47944	0.01291

Table 7.3: The results for the binary one vs. all on the MNIST data set are shown here. The first row contains the error on the 10,000 points of the test data set. The second and third row display the over- and underconfident for the result.

The Informative Vector Machine is used in a multi binary setting here, which means that several binary classifications are trained and in the end all of them are evaluated to get the final result. The biggest problem with a multi binary approach is that if one of the binary Informative Vector Machine is too confident the good results of the other ones are overshadowed. This can happen if the selection of the hyperparameters fails. To reduce this effect the amount of classes in such a multi binary setting shouldn't be too high to avoid generating a bad trained IVM. In table 7.4 the confusion matrix for the multiclass

TODO: Nur Klasse 0 und 1 sind korrekt, der Rest ist noch falsch!

		predicted label									
		0	1	2	3	4	5	6	7	8	9
true label	0	946	0	1	2	0	8	16	0	6	1
	1	0	1049	3	2	0	0	40	20	18	3
	2	5	1	936	22	12	4	12	3	33	4
	3	0	4	2	887	0	6	6	4	69	32
	4	0	3	2	8	671	14	24	0	42	218
	5	3	1	0	26	1	698	36	0	96	31
	6	3	1	0	0	6	8	938	0	2	0
	7	2	13	18	22	3	20	0	685	37	228
	8	0	2	0	7	2	1	13	1	927	21
	9	1	6	0	7	94	13	1	7	22	858

Table 7.4: This table shows the confusion matrix for ten binary IVMs trained in a one vs. all fashion on the MNIST data set.

binary Informative Vector Machine is shown. From it is easy to see, which class are more easy to separate and which ones are harder. For example the handwritten number zero, can be separated with a high accuracy. However the numbers four, six and seven are more difficult. The optimized algorithm used in this thesis deals with it. Here the training heavily relies on the Thread Master. As described before based on their performance on

the test set the Informative Vector Machines get differently much trainings time. So after the zero class has good parameters more computational power is spend on the numbers, which perform worse. The reason for this is that the overall result can be more improved by always training the worst classifier most.

7.1.3 Results for the Random Forest with the Informative Vector Machine

In order to use this approach on this special data set some things have to be changed. The online approach is not applicable here, because if the 60,000 points are split in ten subsets then each subset would only contain 6,000 points. From these 6,000 points only 3 pictures would be classified incorrectly, because all the trees where apple to perfectly fit them and the classification error is therefore only 0.0005. These three pictures spread over ten classes, which is not enough to train a Informative Vector Machine. Even by adding a new batch, without changing the Random Forest and only forwarding them through the Random Forest, the error would be 0.0822, which would lead to more points. However these points are spread over all different combinations of true label with predicted labels. In this example the highest error can be measured between the true label four and the Random Forest returning a nine, but these are only 30 points, which are not enough to train an IVM.

This means the offline approach has to be chosen. At first the Random Forest is trained to have enough wrong classified points. To achieve that only 320 trees with a depth of 14 are used. These trees produce the following result on the 60,000 points of the training set given in table 7.5. The rows of this confusion matrix correspond to the true values of the data points, the column however is the predicted label of the Random Forest for this point. The sum of a row corresponds therefore to the total amount of points of this class. Some remarks to the result are, that nearly all the ones were classified correctly. However a lot of other numbers where falsely classified as one even though they weren't. Remarkable here is that the amount of eights classified as one is the highest amount of wrong classified points in the whole matrix.

Each column represents one of the predictive class mentioned above. These columns are then given to a multiclass Informative Vector Machine, which generates for each class, which has more than 100 points a binary IVM in a one vs. all fashion. These are then trained and the hyperparameter for this special case are then found, which then results in the confusion matrix shown in table 7.6.

This confusion matrix shows the result for the combination of the Random Forest with the IVMs. The bold numbers show the results, where an Informative Vector Machine was used, in nearly all cases the result could be improved with an IVM. However the hyperparameter optimization on such outliers is hard and not in all cases an improvement could be reached. There are two examples in this confusion matrix for that. The first one is the true label nine, which was confused with an seven, the IVM used for this example couldn't find stable parameters for the problem and made the result worse than the original prediction of the Random Forest. The same holds for the IVM with the true label five and the

		predicted label									
		0	1	2	3	4	5	6	7	8	9
true label	0	5647	32	1	18	5	4	185	1	20	10
	1	0	6708	17	4	0	0	5	5	1	2
	2	47	339	5351	29	41	0	35	60	53	3
	3	37	410	71	5413	3	10	23	66	55	43
	4	5	417	15	2	4805	0	106	46	4	442
	5	117	454	11	589	29	3931	162	18	32	78
	6	56	252	2	1	3	16	5586	0	2	0
	7	18	445	24	2	27	0	2	5612	3	132
	8	92	784	28	366	32	21	48	52	4344	84
	9	55	436	5	85	135	2	7	287	14	4923

Table 7.5: This table shows the confusion matrix for a Random Forest on the MNIST data set.

		predicted label									
		0	1	2	3	4	5	6	7	8	9
true label	0	4831	10	22	17	0	1007	0	8	20	8
	1	3	6708	17	2	2	3	0	4	1	2
	2	16	0	5690	9	11	74	2	13	55	88
	3	14	1	77	5816	11	44	1	63	55	49
	4	0	0	26	1	5746	6	0	11	6	46
	5	0	6	15	0	7	5242	0	67	33	51
	6	10	0	2	1	27	262	5612	0	2	2
	7	2	3	39	0	0	20	0	6161	3	37
	8	33	8	32	0	19	111	4	87	5482	75
	9	21	0	7	43	3	71	1	1361	33	4409

Table 7.6: The confusion matrix on the MNIST data set for a Random Forest with an Informative Vector Machine is plotted here.

confused result of zero, which overshoot the result by far. It reduced the amount of falsely classified fives as zeros however it increased the amount of zero recognized as fives. However with these two IVMs the overall result still improved from a correctness of 87.200% to 92.828%, ignoring these two bad IVMs would improve the result further to 93.815%.

This result show the possibilities of the Random Forest with the Informative Vector Machines. However it also shows the disadvantages and the limitations of this method.

7.2 USPS

An other well known data set consisting of handwritten digits is the USPS dataset [Hu194]. It is consider to be much harder than the MNIST data set, because not all classes occur with the same amount of samples and the test set contains some images, which are not as clear

as in the MNIST dataset. There are only 7291 images for the training and 2007 for testing. The share for the trainings data is shown in table 7.7. Each image has a size of 16×16 , which results into a 256 element vector. However in this data set each pixels changes at least once and therefore no reduction is performed. In comparison to the MNIST dataset the amount of data is lower by a factor of around 25.20. The test data set is only lower by a factor of 15.26, which shows that in this example there is in comparison to MNIST less data and more test data.

Class	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

Table 7.7: This table shows the amount of samples per class in the USPS data set [Hul94].

7.2.1 Random Forest Results

Like for the MNIST data set in section 7.2.1 decision trees were trained. For this dataset just the multiclass performance was evaluated. In this case 23,664 trees with a height of 35 were used, they allocated 15 GB of the RAM. The data was split again in 10 splits and they were presented to the algorithm. As result of the increased amount of trees the algorithm needed 59 seconds per trainings step. The performance was evaluated on the 2007 points of the test set. After each split the whole test set was measured, this is plotted in figure 7.4. In the end the algorithm could classify 1863 points correctly and only missed 144 points.

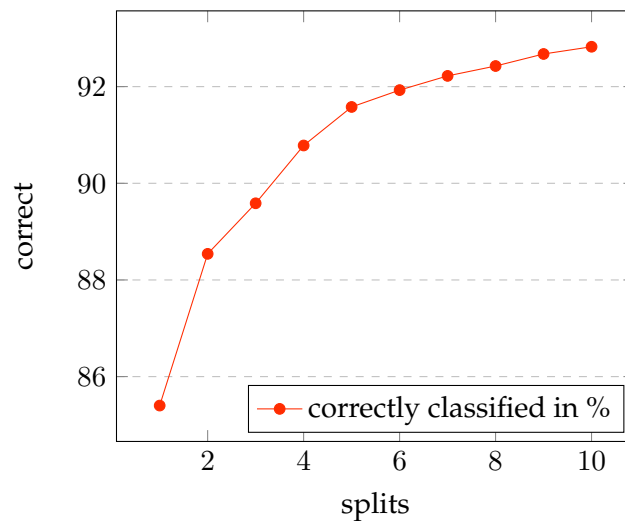


Figure 7.4: The amount of correctly classified points in percent on the 2007 points of the test set is illustrated in this figure. Each adding of a new batch improves the overall result of the Random Forest.

For comparison choosing an offline approach here would go faster but it would only

achieve 91.47 % correctly classified points on the test data set. However the the online method achieves 92.83 %, which is slightly better than the results from Saffari *et al.* [SLS+09]. They presented an online Random Forest algorithm too, which only used 100 trees with a height of five and an final online result for the USPS data set of 91.4 %. That means the approach of this thesis was able to classify 29 pictures more correctly than theirs. Furthermore the approach presented in this thesis is able to train more trees and even deeper trees than their approach, which explains the result.

7.2.2 Results of the Informative Vector Machine

The multiclass Informative Vector Machine was used in this section. The confusion matrix for the result is shown in table 7.8. It has an overall error of 14.5491%, which means that in this example 292 pictures were classified incorrectly. Furthermore similar handwritten digits like before in the MNIST scenario have difficulties here. Nonetheless was the multiclass Informative Vector Machine approach able to find for each class hyperparameter, which give good results and don't destroy the overall performance.

		predicted label									
		0	1	2	3	4	5	6	7	8	9
true label	0	340	0	2	1	1	0	13	0	1	1
	1	0	206	4	0	48	0	4	0	0	2
	2	5	0	182	4	2	1	1	1	2	0
	3	5	0	3	149	0	5	0	0	3	1
	4	1	1	6	2	160	1	4	0	0	25
	5	3	0	0	12	1	131	4	0	5	4
	6	5	0	3	0	3	2	154	0	3	0
	7	1	0	3	1	5	1	0	100	1	35
	8	4	0	2	7	0	3	2	0	145	3
	9	3	0	3	1	17	2	0	1	2	148

Table 7.8: The confusion matrix for a Informative Vector Machine on the USPS data set is plotted here.

7.3 Washington

In this section the publicly available University of Washington RGB-D Object dataset was used [LBRF11]. It is a widely used multi-view dataset, which already was used by many others and is therefore great for benchmarking. The dataset contains 300 different household objects, which can be categorized into 51 classes. Because of the structure of this work we focused on the object category recognition on this dataset. The possible instance recognition was neglected. All of the images were taken on a rotatable table with changing views and angles for all the objects. This results in 200,000 RGB images, depth pictures and binary masks for each configuration of the camera position and instance.



Figure 7.5: In this figure some of the used classes are illustrated. All of them are separated from the background and display different objects [LBRF16].

These pictures could be now transformed like before into vectors, which are then used in the selected approach. However there is a better approach for generating features on pictures than using their pixel information. The best known approach to extracted features from a picture is called convolutional neural networks. These are able to extract local information and combine them in several steps to reliable features, which are even better than human crafted features. This was shown by different authors like Bluche *et al.* and Antipov *et al.* [BNK13, ABRD15]. They even outperformed existing feature extractors like SIFT [FDB14].

As a result of this, this thesis uses calculated features from a CNN and classifies them. The training of these features was done by Monika Ullrich in her work "Combined Deep and Active Learning for Online 3D Object Recognition" [Ull16]. We used the combined features from the embedded depth and the RGB features, which were reduced from 8192 dimensions to 1000 by using MaxRel [Ull16]. These 1000 long vectors were presented to the Random Forest with the Informative Vector Machine and the results are shown in the next section. This data set was not tested on the pure Informative Vector Machine, because training 51 stable binary Informative Vector Machine, with so less data points is not possible. The chance that one of them would become an overconfident classifier is too high.

7.3.1 Result for the Random Forest and the Informative Vector Machine

Before the results are described here, the problem has to be formulated more carefully. Each class consists out of several instances and the training was performed with all but one instances and the not used instance was used for the testing. This means that the approach has seen for example five different bananas and has then to decide for an unseen instance

of a banana, if it is a banana or something else.

The results of the Random Forest with the Informative Vector Machine show the big challenge behind this formulation. The training is done like before in an online fashion on the training set, where 135 Decision trees with a height of 35 were used. The test was performed on the third fold generated by Ullrich, because the results for all folds generated in her thesis are similar only one was used here. This training set was split in five subsets. Each of these splits only contains out of 6898 data points. The results for the test data set and the next training split are shown in figure 7.6. The blue line shows the result after the training of the split number shown on the x-axis. This means the first point in the line corresponds to the first online update. The results are measured before the next trainings split had been integrated in the Random Forest. This means all of these points were never been seen before by the algorithm and still the result is above 98.35% for all the training splits. This shows that the training set consists out of instances, which can easily be recognized and the error is generated by the difference between the trainings instances and the test instance. This means this information is probably not covered in the selected features provided by Ullrich and can therefore not be separated here[Ull16]. This can be seen in the line for the test points, which is plotted in red. In the end it reaches 87.722% correctly classified points on the test data set.

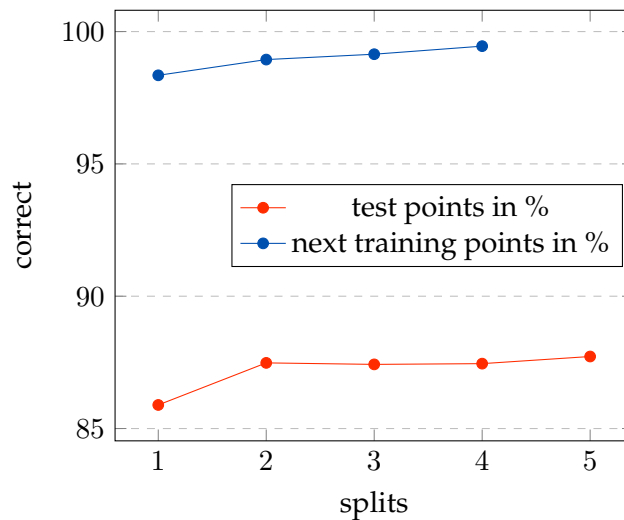


Figure 7.6: The classification result for the test and the next training split are shown here. The red line shows the result for the test result after each online step, the first step corresponds to the offline step with the first split. In blue the amount of correct classified points for the next trainings data split is given. All the results are calculated after the Random Forest where updated. That's why there is no point for the fifth iteration, because no new trainings split is available.

The classification result for the next training split shows a challenge here. The multiclass Informative Vector Machine is build on the wrong classified points, which is a problem in this scenario. The reason for this is that each split only consists out of 6898 points and only

112 were classified incorrectly, this can be seen by the blue line in figure 7.6. These 112 wrong classified elements are spread over the existing combinations of the 51 classes. So that each binary Informative Vector Machine only would get at most 10 falsely classified points, which is not enough to train reliable hyperparameter for a 1000 dimensional problem. In the end the Informative Vector Machines couldn't be used at all. The problem here is that the whole trainings data set from Ullrich only consist out of 34,488 points and even if this whole set is presented to the Random Forest all trainings points are then classified correctly. So that no Informative Vector Machine could be used here and furthermore the overall results on the test is only 86.8417%.

8 Conclusion and Future Work

8.1 Conclusion

This thesis presented an online learning approach for big data sets. It used Random Forest as dataset splitters, which are able to classify big portions of the dataset. The falsely classified points in these sub splits are then processed by an Informative Vector Machine. This whole approach was trained in an online fashion, where not all points were available in the beginning. By improving the Decision trees used in this thesis, the overall trainings time and memory consumption for a Decision tree could be dramatically reduced. This reduction made it possible to train the Decision trees in an online fashion during the execution of the algorithm. Furthermore the result of the online trained Random Forest are better than the offline results, which was shown on the MNIST, USPS and the Washington data set. The results of our Online Random Forest for the USPS data set were better than the online approach presented by Saffari *et al.* [SLS⁺09]. Furthermore our approach nearly always leads better results than our own offline approach, which is remarkable for an online training.

However the results for the combination of the Random Forest with the Informative Vector Machine have shown the advantages of this approach. The combination of both approaches lead to a improved result than the Random Forest on its own. However this was only possible because the trained Random Forest was limited in trainings time and height, which was necessary, because of the simplicity of the MNIST dataset. Even for the dataset from the University of Washington the problem still arose that a Random Forest could be trained, which was able to capture the data better than a bad trained Random Forest with a combined Informative Vector Machine.

The Informative Vector Machine used in this thesis was improved with several ideas, like the better selection criteria for the active set or the approximations for the flipped active set instead of an full Expectation Propagation approach. Several other improvements were performed to get a fast and online learning approach, which is able to train stable hyperparameter even on high dimensional problems like the MNIST dataset. However the flaws of the binary one vs. all method were discussed and could be reduced through a thorough use of the Thread Master. It always trained the worst Informative Vector Machine first and so improved the one, which was most likely to give a bad result. Nonetheless is the hyperparameter optimization of the Gaussian kernel still the hardest part of the Informative Vector Machine.

So this thesis presented two different approaches, which were changed to fit the online requirements of this thesis and were further improved to give the best possible results in

as little time as possible.

8.2 Future Work

In a future work the newly designed online approach of this thesis could be used on a real world online problem, where the data is only available in a stream fashion. This would use the advantages of this method best and could lead to state of the art results, because of the fast and adaptive learning approach presented in this thesis. The reason for that is the fast trainings times of the Random Forest and the fast online update of it.

The Decision trees it self could be improved by a better decision criteria, which would improve the quality of each splits. One of the big with draws of the entropy and gini index are that the influence of the size of the split is not represented well enough. This means that a split, which only splits a few points away, could have a lower cost than splitting of more points in a correct fashion. So improving the splitting criteria could lead to a even better performing tree and therefore Random Forest. Additionally the splitting in the different layers in a deep decision tree, could be improved by a more dynamic approach, where the amount of necessary layers can be estimated by the current split. This means that after the root tree was calculated the depth for the following trees depends on the amount of points in the leaf, which would increase the flexibility of this approach and further improve the trainings time and the memory demands.

On the other hand the Informative Vector Machine could be further improved. A better hyperparameter optimization could be searched, which would decrease the trainings time and would make it possible to increase the amount of used inducing points even further. Because in the moment the trainings time solely depends on the amount of evaluations necessary to find stable and good hyperparameter for the Gaussian kernel. If the Random Forest Kernel is used the kernel evaluation time should be optimized, because evaluating several hundred Decision trees takes more time than just the inner product between two vectors. Furthermore the selection of the active set could be improved by designing an online approach, which would be able to select induced points, which aren't part of the trainings set. However there is currently no approach to do that for high dimensional problems, because finding the right spot to place such a point is still an unsolved problem.

Bibliography

- [Abr72] MILTON Abramowitz. 1. a. stegun, 1972: Handbook of mathematical functions. *National Bureau of Standards Applied Mathematics Series*, 55:589–626, 72.
- [ABRD15] Grigory Antipov, Sid-Ahmed Berrani, Natacha Ruchaud, and Jean-Luc Duge-
lay. Learned vs. hand-crafted features for pedestrian gender recognition. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1263–
1266. ACM, 2015.
- [AH05] Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with
increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE
Congress on*, volume 2, pages 1769–1776. IEEE, 2005.
- [BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Clas-
sification and regression trees*. CRC press, 1984.
- [Bis06] Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer,
2006.
- [BNK13] Théodore Bluche, Hermann Ney, and Christopher Kermorvant. Feature ex-
traction with convolutional neural networks for handwritten word recogni-
tion. In *Document Analysis and Recognition (ICDAR), 2013 12th International
Conference on*, pages 285–289. IEEE, 2013.
- [Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- [BSvWV16] Hildo Bijl, Thomas B Schön, Jan-Willem van Wingerden, and Michel Verhae-
gen. Online sparse gaussian process training with input noise. *arXiv preprint
arXiv:1601.08068*, 2016.
- [CGLL10] Fu Chang, Chien-Yang Guo, Xiao-Rong Lin, and Chi-Jen Lu. Tree decomposi-
tion for large-scale SVM problems. *J. Mach. Learn. Res.*, 11:2935–2972, Decem-
ber 2010.
- [CKY08] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical
evaluation of supervised learning in high dimensions. In *Proceedings of the
25th international conference on Machine learning*, pages 96–103. ACM, 2008.

- [CO02] Lehel Csató and Manfred Opper. Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668, 2002.
- [dC85] Marie-Jean Antoine Nicolas de Caritat. Condorcet. essais sur l’application de l’analyse la probabilité des decisions rendues a la pluralité des voix, 1785.
- [DG14] Alex Davies and Zoubin Ghahramani. The random forest kernel and other kernels for big data from random partitions. *arXiv preprint arXiv:1402.4293*, 2014.
- [DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [Die00] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.
- [Eic14] Hubert Eichner. Neural net for handwritten digit recognition in javascript. <http://myselfph.de/neuralNet.html>, 2014.
- [FDB14] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.
- [FRKD12] B Fröhlich, E Rodner, M Kemmler, and J Denzler. Large-scale gaussian process classification using random decision forests. *Pattern Recognition and Image Analysis*, 22(1):113–120, 2012.
- [Gam95] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.
- [GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.
- [GW00] Pierre Geurts and Louis Wehenkel. Investigation and reduction of discretization variance in decision tree induction. In *European Conference on Machine Learning*, pages 162–170. Springer, 2000.
- [Han14] Nikolaus Hansen. Cma-es. <https://github.com/CMA-ES/c-cmaes>, 2014.
- [HFL13] James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.
- [HO96] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation.

-
- In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 312–317. IEEE, 1996.
- [Hul94] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.
- [KR06] Malte Kuss and Carl Edward Rasmussen. Assessing approximations for gaussian process classification. *Advances in Neural Information Processing Systems*, 18:699, 2006.
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [LBRF11] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.
- [LBRF16] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Washington homepage. <https://rgbd-dataset.cs.washington.edu/index.html>, 2016.
- [LPJ05] Neil D Lawrence, John C Platt, and Michael I Jordan. Extensions of the informative vector machine. In *Deterministic and Statistical Methods in Machine Learning*, pages 56–87. Springer, 2005.
- [LRT14] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. Mondrian forests: Efficient online random forests. In *Advances in neural information processing systems*, pages 3140–3148, 2014.
- [LSH⁺03] Neil Lawrence, Matthias Seeger, Ralf Herbrich, et al. Fast sparse gaussian process methods: The informative vector machine. *Advances in neural information processing systems*, pages 625–632, 2003.
- [Mer09] James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209:415–446, 1909.
- [Min01] Thomas P Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.
- [MR11] Andrew McHutchon and Carl E Rasmussen. Gaussian process training with input noise. In *Advances in Neural Information Processing Systems*, pages 1341–1349, 2011.
-

- [MTC15] Dennis Mund, Rudolph Triebel, and Daniel Cremers. Active online confidence boosting for efficient object classification. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1367–1373. IEEE, 2015.
- [Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.
- [NGH07] Andrew Naish-Guzman and Sean B Holden. The generalized fitc approximation. In *NIPS*, pages 1057–1064, 2007.
- [NR08] Hannes Nickisch and Carl Edward Rasmussen. Approximations for binary gaussian process classification. *Journal of Machine Learning Research*, 9(Oct):2035–2078, 2008.
- [REM11] Naveen Ramakrishnan, Emre Ertin, and Randolph L Moses. Assumed density filtering for learning gaussian process models. In *Statistical Signal Processing Workshop (SSP), 2011 IEEE*, pages 257–260. IEEE, 2011.
- [RQC05] Carl Edward Rasmussen and Joaquin Quinonero-Candela. Healing the relevance vector machine through augmentation. In *Proceedings of the 22nd international conference on Machine learning*, pages 689–696. ACM, 2005.
- [RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
- [SB⁺01] Alexander J Smola, Peter Bartlett, et al. Sparse greedy gaussian process regression. *Advances in neural information processing systems*, 13:619–625, 2001.
- [SG06] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.
- [Sha48] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(4):623–656, October 1948.
- [SLS⁺09] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1393–1400. IEEE, 2009.
- [SWL03] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318, 2003.
- [Tip01] Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.
- [Tit09] Michalis K Titsias. Variational learning of inducing variables in sparse gaussian processes. In *AISTATS*, volume 5, pages 567–574, 2009.

- [Ull16] Monika Ullrich. Combined deep and active learning for online 3d object recognition. Master's thesis, Technical University of Munich, November 2016.
- [VV12] Jarno Vanhatalo and Aki Vehtari. Speeding up the binary gaussian process classification. *arXiv preprint arXiv:1203.3524*, 2012.
- [WR96] Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for regression. *Advances in neural information processing systems*, pages 514–520, 1996.