Machine Learning Based Water Analysis Using an Underwater Robot

Bachelor Thesis

by

Hendrik Emil Eichhorn

Degree Course: Industrial Engineering and Management B.Sc.

Matriculation Number: 1958970

Institute of Applied Informatics and Formal Description Methods (AIFB)

KIT Department of Economics and Management

Advisor:	Prof. Dr. York Sure-Vetter
Second Advisor:	Prof. Dr. Andreas Oberweis
Supervisor:	Dr. Patrick Philipp
Submitted:	July 21, 2020

Abstract

Water Contamination is an important issue in many cases, for example aquaculture and aquaponics and dealing with it requires to recognize and identify it. While some approaches for water analysis exist, they either are non-spatial or only use 2D data. To solve that issue, this thesis deals with developing an approach that can transform spatial data into features and then use them with standard classifiers to classify different kinds of contamination, as well as introduce requirements for a robotic system, which is capable of gathering the data.

To validate this approach, data was simulated in the form of multiple distributions exhibiting different properties to examine how well the approach handles them. Besides that, different sets of measurements were used with different spatial resolutions alongside a whole set of classifiers, and additional factors such as incomplete data and measurements deviating from their planned position were included.

The classifier "Nearest Neighbors" proves to be the most potent for classification and delivers good results even for low-resolution data. Incompleteness and Deviation show effects with rising degree, and should be avoided, but the approach still delivers robust results, proving that with a robot capable of making and delivering precise measurements, the approach is viable.

Contents

List of Abbreviations

CNN Convolutional Neural Network.

GNN Graph Neural Network.

IMU Inertial Measurement Unit.

IoT Internet of Things.

LBL Long Baseline.

ML Machine Learning.

RL Reinforcement Learning.

ROS Robot Operating System.

USBL Ultra Short Baseline.

List of Figures

List of Tables

1 Introduction

One of the greatest mysteries of the oceans must be Atlantis, the legendary city, which was supposedly drowned by the sea. So far, nobody was successful in finding it, and it might stay this way forever. Something that can be found very well though, not only in the oceans, but almost every aggregation of water is contamination. Sometimes algae, which can be spotted by the naked eye, sometimes micro-particles, which require advanced sensing. Depending on the environment, this contamination can be a problem, so it needs to be detected through environmental monitoring, which includes the collection and analysis of data. Often this is about large spaces of water, like bays, harbors, or the oceans themselves. But there are also cases, in which the environment is much smaller, for example aquafarming [10] or aquaponics [18] sites. Especially for systems like aquaponics, there are so many factors that are important to be monitored, like bacteria or biological substances. To do that, there have to be repetitive measurements at various points within the system. While those measurements can be taken and analyzed manually by a person, this work is laborious and offers great potential for automation. There are two problems to be solved: Gathering data and analyzing it.

This thesis will focus on creating a system, which solves those two problems with low-cost equipment in such a way, that it could be widely employed by anyone interested in using it for their systems. In a first step I will develop a Machine Learning (ML) approach to solve the task of automatically analyzing spatial underwater data and classify it into different groups, which allows better understanding of the respective systems and enable increased autonomy allowing advanced scaling. To allow the approach to run on low-cost hardware, particular emphasis will be put on making the approach simple and lightweight. In the second step, I will discuss the requirements that a robot needs to fulfill, so that it can gather the data as needed. To further aid the robot selection or development, I will also introduce some base technologies which would be suitable to be used with the system and make some suggestions for the robot's design and structure. Finally, I will also introduce a complete setup which will allow running the system, alongside information on the implementation of the approach so it can be used and adapted by others.

Section 3 will focus on the Data Analysis, which includes processing the data to create features from the raw data and training models. The approach will be tested with different sets of data and under different scenarios to examine it's robustness. In Section 4 I will then go on the robotics parts, which covers different topics including localization and propulsion, briefly discussing different options. Afterwards, Section 5 will introduce further technolgy necessary to create a working system, together with a mock-up setup showcasing how it would work. Besides that, this section will also include information on the implementation of the code.

2 Related Work

The are several ways to ways to approach the analysis of water contamination. On a very small scale, this can happen through pictures, such as image classification of contamination in a culture dish, as presented by Gupta and Ruebush [6]. Their approach performs very well with a accuracy of 96% for the given setup. Unfortunately it has some shortcomings: For once, the camera sight under water often becomes limited very quickly. Additionally, many forms of contamination are not even visible to a camera. Other systems aiming to analyze water quality take measurements of more properties, like Mohammed et al [13]. While they include values for pH and bacteria count among others, they omit the spatial features of their measurements, as they are examining a water distribution system, which is space-wise strictly limited. Systems focusing on analyzing data from remote sensing such as the one presented by Hafeez et al [7] can deliver great results for large areas from space, and there different options to analyze the data, for example through a 3D Convolutional Neural Network (CNN) as done by Li et al [12]. It might be possible to create a small-scale remote-sensing setup for smaller environments with a hyper-spectral camera, but this would basically lead back to the problems of a normal camera, but would add the significant higher price for a normal camera.

While the afore mentioned water analysis approaches won't work the way that is needed, using Deep Learning for classification of 3D data is the core of numerous methods, for which Ahmed et al [1] give a good reference. Problems with those models include that they are computationally quite expensive, which is an issue with low-level hardware, as well as that they often need hyperparameter-optimization to work properly. Since the form of the contamination are usually significantly less complex than the objects for which those methods were developed, the required computational cost is not justified.

Taking data in agriculture can happen for example through Internet of Things (IoT) systems as shown by Dutta et al [4], but those are usually restrained to operating at the edge of a water system, and mostly are limited in number. This means that the spartial density of measurements from IoT systems will remain low in general. A very interesting approach, which is combining robotics to gather spatial data and then analyze it in the search of contamination is presented by Hu et al [8]. They designed a swarm of robotic fish, which collectively search for contamination. Since the goal environment is a port, a rather large water body, the fish don't have set paths, but instead swim around in search of traces of contamination. This way they create a map of contamination concentration and adopt their paths to monitor areas with higher concentration with higher frequency. While for their environment, this is a great approach, it doesn't fit too well for an environment, which needs regular monitoring of multiple parameters. In addition, rather than simply finding contamination concentrations, I also want to classify the results.

3 Data Analysis

The work-flow to classify different kinds of contamination consists of numerous steps. First, the data has to be gathered and then processed in such a way that it can be used for classification. Afterwards the classification itself can take place. This pipeline can be found on the left side of Figure 1. This pipeline will always be the same, may it be for experiments, training a model for application or finally putting the model to work.

In this section I will exclusively focus on developing the model and running experiments with it. The data gathering for this usually consists of using an external source, or taking measurements yourself. Since it proved itself difficult to find real-world data, which can be used for my approach, I decided to simulate data instead. This means that the step of data gathering was split into several sub-steps. The first of those is the selection of a proper environment to simulate data inside. To keep this process as simple as possible I decided to use a a cube of edge length 100 cm, which can also easily be used to scale the results to larger environments. With this given, I introduced segmentations of the environment and contamination distributions to fill it with next. The segmentations and distributions are chosen in such a way, that they cover as many options as possible, so that the final result is reliable and comprehensive. Afterwards the processing of data takes places, which in this case is the conversion of raw data to features, which subsequently are used to train and test different classifiers. To increase the realism, I also introduced different scenarios to test the classifiers against. This can be seen on the right side of Figure 1.

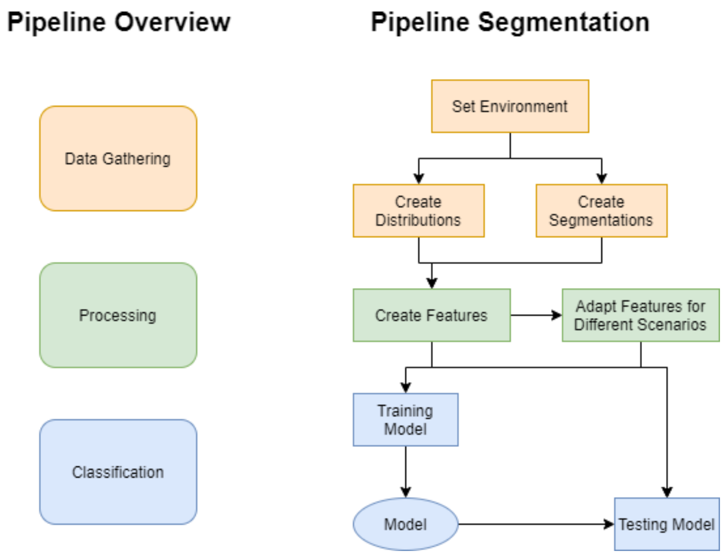


Figure 1: Pipeline Analysis

3.1 Segmentation

Since it is impossible to gather continuously data for the complete environment, it is necessary to find a discrete segmentation, where a measurement is taken in every segment. What has to be considered for the segmentation is that segments should be small enough so that the measurements create a complete and accurate representation of the environment. Generally speaking I expect to see increased accuracy of the classification with increased resolution. While in theory this means that it would be perfect to have an extremely high resolution, the measurement becomes a problem. For once, the tighter the segmentation grid is the, the harder it is for the robot to actually take a measurement exactly at the planned points, as there is very little space for maneuvering. Secondly, a longer path through many points means that the robot takes a bigger influence on the environment, distorting the result. Finally, a longer path also means a longer tour, during which the environment keeps changing, as water is a very volatile medium. All of this leads to a decrease in accuracy, which means that the segments should be big enough so that a measurement in every one of them can be done by a robot in reasonable time and with reasonable accuracy. To do this I created multiple segmentations, which simulate where a robot would take measurements.

Figure 2 shows the segmentation set "Basic", which contains different resolutions, dividing the given environment evenly in X, Y and Z dimension. The points in those segmentation plots are at the center of all segments respectively, and show where a measurement should be taken.

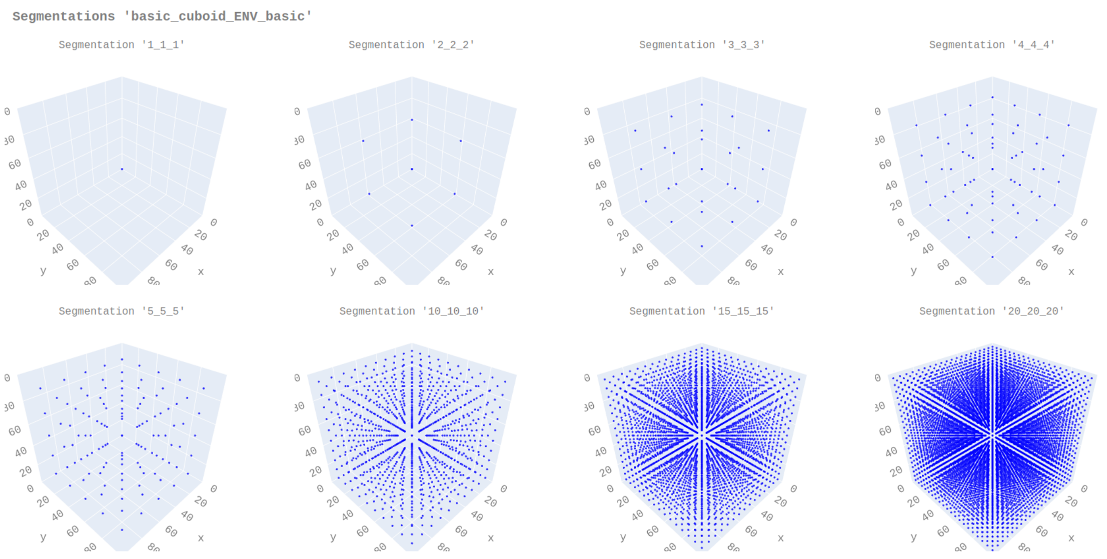


Figure 2: Segmentation Set "Basic"

Apart from the basic approach, another option would be to not use the same resolution in every dimension. This is especially an interesting approach since the data will be selected by a robot. The path of a robot might include long straight path-segments in one dimension. In this case it would be possible to increase the resolution in that dimension without substantially elongating the path or making it more complicated, as it is displayed in Figure 3. To examine this option, I also created the segmentation set "Advanced", which can be seen in Figure 4.

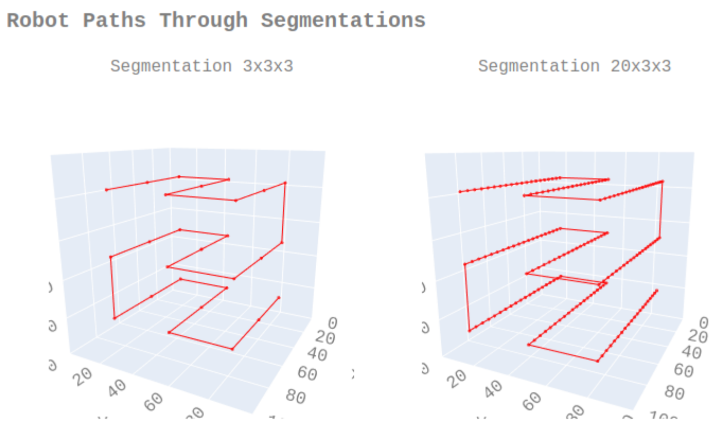


Figure 3: Path Comparison

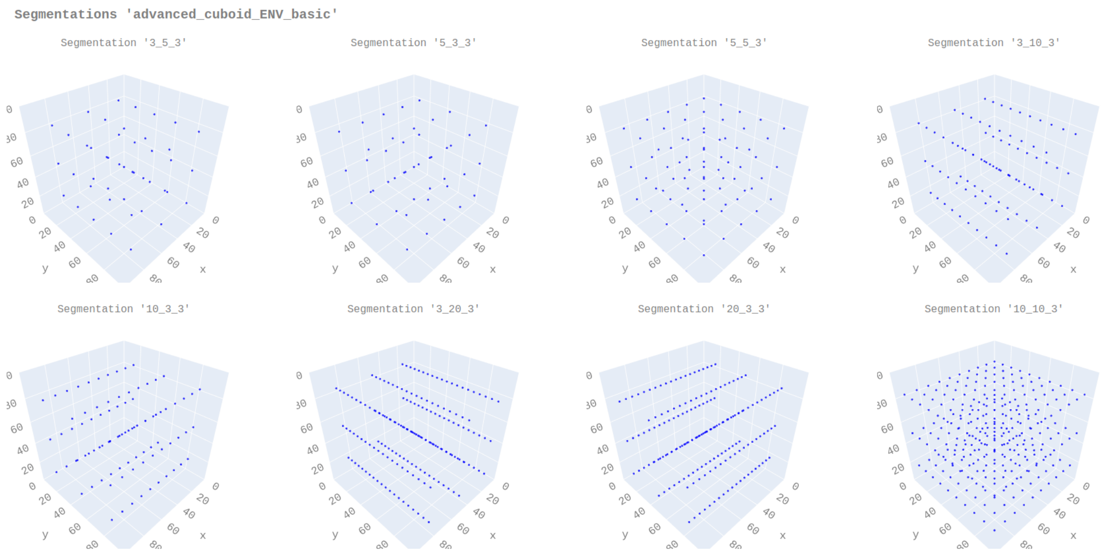


Figure 4: Segmentation Set "Advanced"

The center points for both sets are saved as arrays, where for every point the place in the environment, in the following referred to as the coordinates, as well as the place in the grid, in the following referred to as position, are saved. For the example in the segmentation 2x2x2 the point with the position [1,1,1] would have the coordinates [25,25,25].

3.2 Distributions

The general assumption here is that all contaminations resemble a 3D Gaussian distribution. Those distributions can be distinct in different ways such as through orientation, shape and position. Moreover I assumed, that at all times, only one contamination can be found instead of multiple.

To closely examine the how well each of those characteristics can be used for classification, I created different distribution sets. Each of those sets contains different classes, specifying a certain distribution through the mean and the variance in X, Y and Z dimension respectively. Since members of a class are supposed to be only similar, not the same, instead of a fixed value, a range is given instead. Based on those ranges, the classes can be filled with samples. For each of those samples, six values are drawn from the six ranges. For all figures, which are used to visualize the different sets and their respective classes, the mean value of all six ranges is selected.

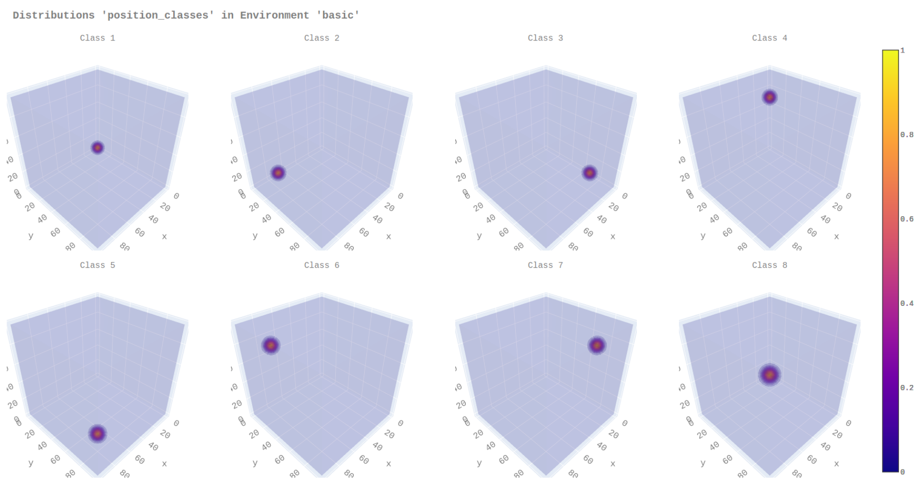


Figure 5: Distribution Set "Position Classes"

Class	X Mean	Y Mean	Z Mean	X Variance	Y Variance	Z Variance
1	[15, 25]	[15, 25]	[15, 25]	[10, 10]	[10, 10]	[10, 10]
2	[75, 85]	[15, 25]	[15, 25]	[10, 10]	[10, 10]	[10, 10]
3	[15, 25]	[75, 85]	[15, 25]	[10, 10]	[10, 10]	[10, 10]
4	[15, 25]	[15, 25]	[75, 85]	[10, 10]	[10, 10]	[10, 10]
5	[75, 85]	[75, 85]	[15, 25]	[10, 10]	[10, 10]	[10, 10]
6	[75, 85]	[15, 25]	[75, 85]	[10, 10]	[10, 10]	[10, 10]
7	[15, 25]	[75, 85]	[75, 85]	[10, 10]	[10, 10]	[10, 10]
8	[75, 85]	[75, 85]	[75, 85]	[10, 10]	[10, 10]	[10, 10]

Table 1: Distribution Set "Position Classes" Ranges

The first distribution set "Position Classes" focuses on the influence of different positions. It contains eight classes, each of which describe a distribution sitting in the eight corners of the environment. To take out other characteristics, the variances are same for all classes and the ranges are eliminated. This is shown in Figure 5 and Table 1.

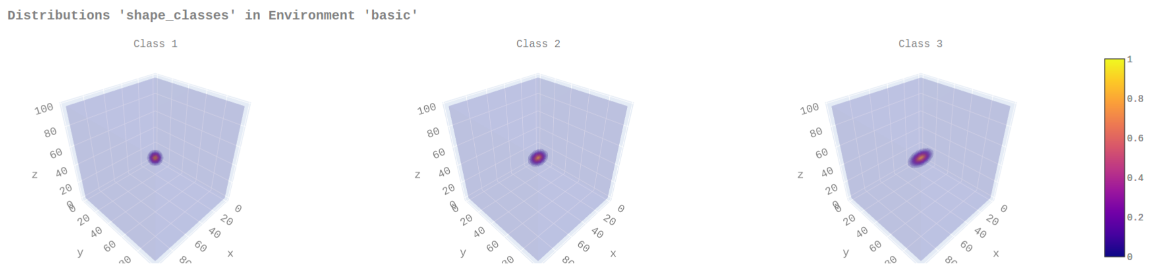


Figure 6: Distribution Set "Shape Classes"

Class	X Mean	Y Mean	Z Mean	X Variance	Y Variance	Z Variance
1	[50, 50]	[50, 50]	[50, 50]	[5, 10]	[5, 10]	[5, 10]
2	[50, 50]	[50, 50]	[50, 50]	[15, 20]	[5, 10]	[5, 10]
3	[50, 50]	[50, 50]	[50, 50]	[30, 40]	[5, 10]	[5, 10]

Table 2: Distribution Set "Shape Classes" Ranges

For the second distribution set "Shape Classes", the shape of the distributions is the important factor. For that reason, all three classes are set exactly to the center of the environment and don't have any variation in position. Additionally, the variation for the Y and Z dimension are constant. Only in X dimension the variance increases from class to class, changing the shape by stretching the distribution. This is shown in Figure 6 and Table 2.

The third distribution set "Orientation Classes" follows a similar pattern as the "Shape Classes" set, as once again three classes can be found and for all of them the position is set exactly to the center. This time two dimensions have the same variance, while the variance of the third dimension is increased, stretching the distribution. The dimension with the increased variance changed for each class, representing a change of orientation for an object keeping its shape. This is shown in Figure 7 and Table 3.

Class	X Mean	Y Mean	Z Mean	X Variance	Y Variance	Z Variance
1	[50, 50]	[50, 50]	[50, 50]	[30, 40]	[5, 10]	[5, 10]
2	[50, 50]	[50, 50]	[50, 50]	[5, 10]	[30, 40]	[5, 10]
3	[50, 50]	[50, 50]	[50, 50]	[5, 10]	[5, 10]	[30, 40]

Table 3: Distribution Set "Orientation Classes" Ranges

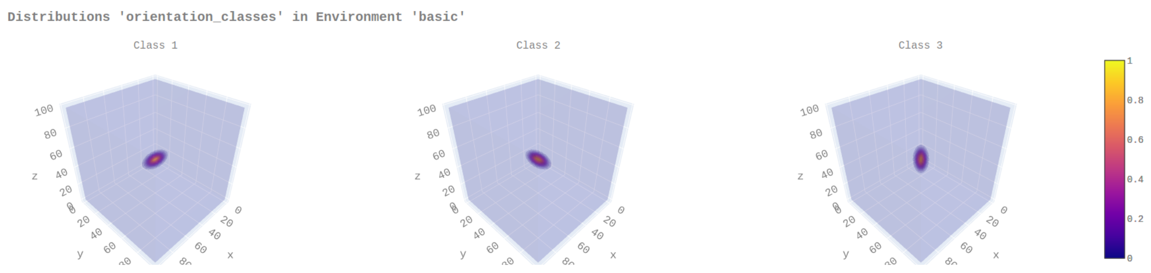


Figure 7: Distribution Set "Orientation Classes"

Finally all those characteristics are brought together for the distribution set "Combined Classes", which includes differences in position, orientation and shape. There is no more clear method in the way the ranges are set, as they are chosen arbitrarily. This set is supposed to be close to what we would expect to see in a real-life application and can be seen in Figure 8 and Table 4.

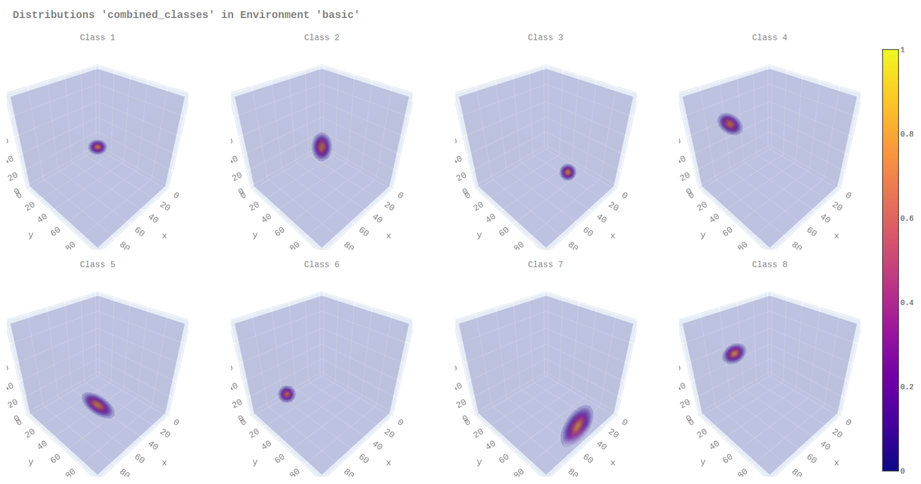


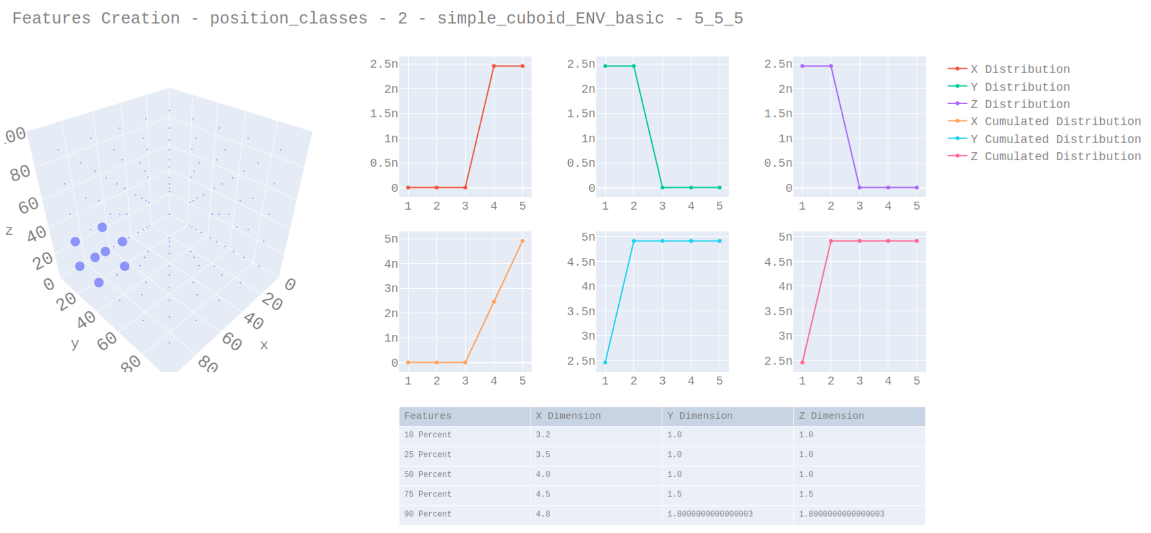
Figure 8: Distribution Set "Combined Classes"

Class	X Mean	Y Mean	Z Mean	X Variance	Y Variance	Z Variance
1	[15, 25]	[15, 25]	[15, 25]	[15, 20]	[15, 20]	[5, 10]
2	[45, 55]	[45, 55]	[45, 55]	[10, 15]	[10, 15]	[30, 40]
3	[30, 40]	[60, 70]	[15, 25]	[10, 10]	[10, 10]	[10, 10]
4	[75, 85]	[30, 40]	[75, 85]	[5, 10]	[20, 40]	[10, 10]
5	[45, 55]	[45, 55]	[10, 15]	[10, 15]	[70, 80]	[10, 15]
6	[85, 95]	[45, 55]	[45, 55]	[5, 10]	[5, 10]	[5, 10]
7	[45, 55]	[85, 95]	[10, 15]	[70, 80]	[5, 10]	[70, 80]
8	[60, 70]	[15, 25]	[60, 70]	[20, 30]	[5, 10]	[10, 15]

Table 4: Distribution Set "Combined Classes" Ranges

3.3 Features

Given a segmentation and a distribution, the next step is to turn the data into features, which can be given directly to the classifiers. Since the data is three dimensional, the number of data points grows exponentially with new increasing resolution. To prevent this, but still preserve the spatial features of the data, I designed the following features as shown for example in Figure 9.

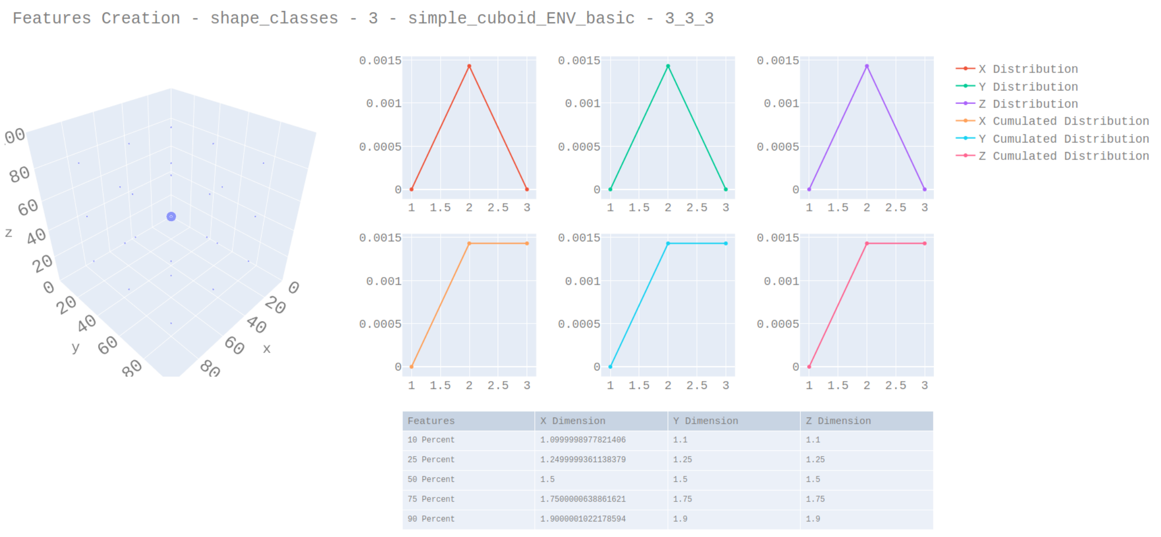


Features	X Dimension	Y Dimension	Z Dimension
10 Percent	3.2	1.0	1.0
25 Percent	3.5	1.0	1.0
50 Percent	4.0	1.0	1.0
75 Percent	4.5	1.5	1.5
90 Percent	4.8	1.8000000000000003	1.8000000000000003

Figure 9: Features Distribution "Orientation Classes - 2" for Segmentation "5x5x5"

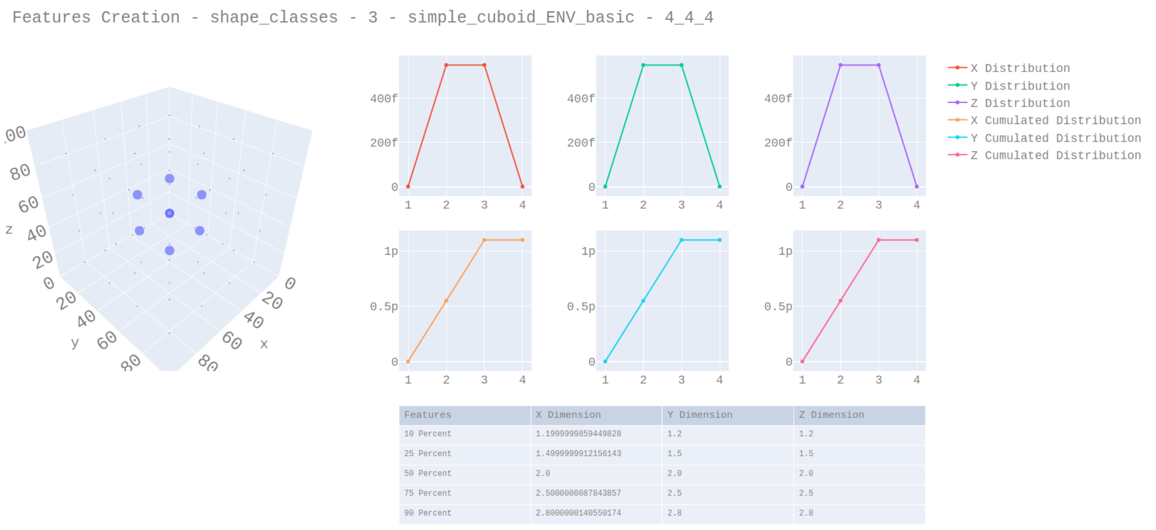
First the segmentation is placed over the distribution and for all points of the segmentation a value is drawn at the respective coordinates. This is shown in the graph on the left, where the size of the points respond to the taken value. Afterwards, all values stemming from points with the same X coordinate are summed up respectively. For a cube of size 2x2x2 this would mean that the values of positions [1,1,1], [1,1,2], [1,2,1] and [1,2,2] would be added up, as well as the values for [2,1,1], [2,1,2], [2,2,1] and [2,2,2], resulting in two values for the X dimension. The same procedure is repeated for the Y and Z dimensions. The resulting values are shown in the three upper graphs on the right as "X Distribution", "Y Distribution" and "Z Distribution". In this case, where we look at distribution "2" of the set "Position Classes", in X dimension there is a peak at the 4 and 5, while for dimensions Y and Z the peak is at 1 and 2, as the center for this class is placed at [80,20,20]. There are different ways to move on from here. For example the features could be defined as the the position of the peak, height of the peak etc. This approach would work as well for distributions I used. But what if in the future the approach should be used with different distributions, which for examples have multiple peaks, or which are not evenly distributed left and right of the peak?

To solve this problem, in the next step, which can be seen in the graphs of the second row on the right side, the values are now added up to create monotone rising cumulated distributions. Based in those graphs I now calculate the positions where 10%, 25%, 50%, 75% and 90% of the maximum value of the cumulated distrbution is reached. In case any of those percentages is reached before 1, meaning the value at 1 is higher than the respective percentage of the maximum value, the position is simply set to 1. These 15 values, 5 for every dimension are then finally used as the features for classification, and can be seen in the table at the bottom.



Features	X Dimension	Y Dimension	Z Dimension
10 Percent	1.0999998977821406	1.1	1.1
25 Percent	1.2499999361138379	1.25	1.25
50 Percent	1.5	1.5	1.5
75 Percent	1.7500000638861621	1.75	1.75
90 Percent	1.9000001022178594	1.9	1.9

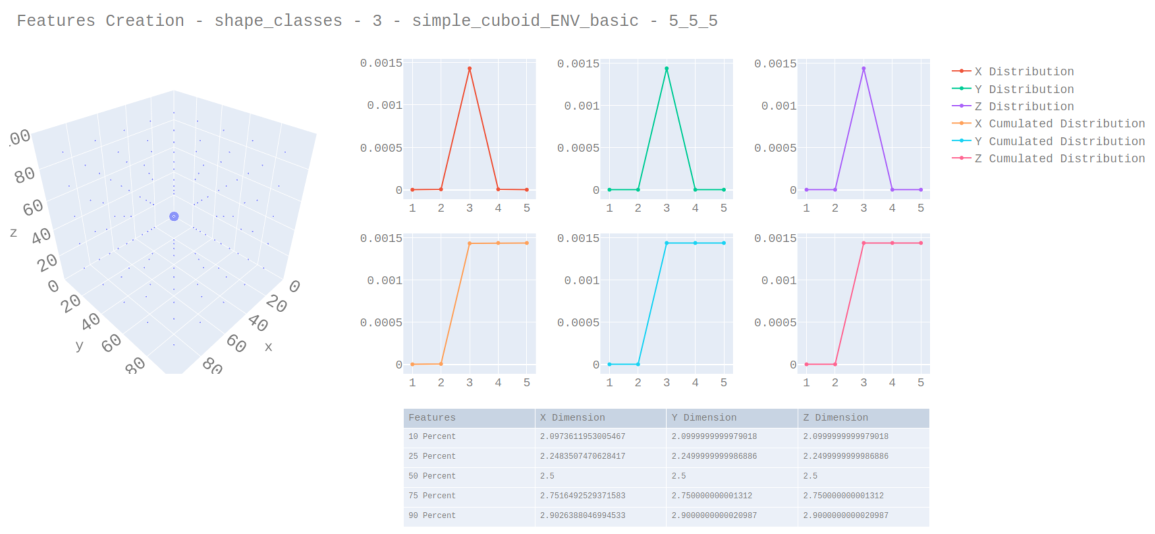
Figure 10: Features Distribution "Shape Classes - 3" for Segmentation "3x3x3"



Features	X Dimension	Y Dimension	Z Dimension
10 Percent	1.1999999859449828	1.2	1.2
25 Percent	1.4999999912156143	1.5	1.5
50 Percent	2.0	2.0	2.0
75 Percent	2.5000000087843857	2.5	2.5
90 Percent	2.8000000140550174	2.8	2.8

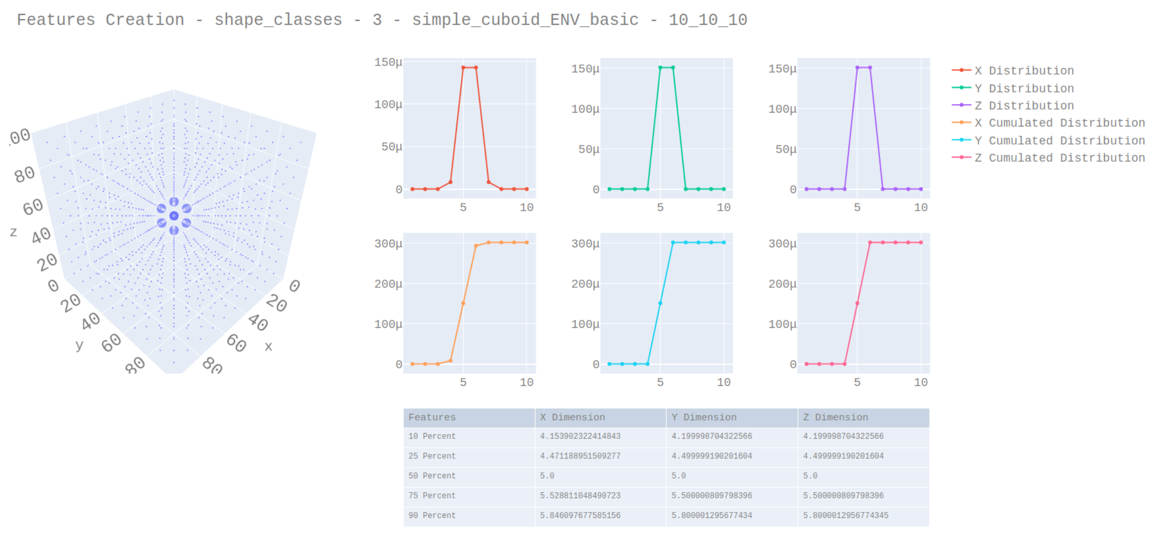
Figure 11: Features Distribution "Shape Classes - 3" for Segmentation "4x4x4"

The features belonging to the "Position Classes" distribution set shown in Figure 9 are very clear in both graph form and the final features values. In comparison, the differences become a lot less obvious for the set "Shape Classes". Figures 10 to 13 show the features for distribution "3" for different segmentations. Neither 3x3x3 nor 4x4x4 and 5x5x5 show a strong difference in the graphs between X and Y and Z dimension, even though the distribution is significantly stretched in X dimension. Only 10x10x10 begins to show obvious differences. When looking at the final features, the differences, while visible, remain small. For 3x3x3 they are as little as 10^{-7}. Interestingly, for 4x4x4 the differences are partly even smaller. A reason for this could be that since this segmentation is odd, there is not measurement directly at the center of the distribution, which is the by far most expressive value. Both 5x5x5 and 10x10x10 deliver significantly more meaningful results.



Features	X Dimension	Y Dimension	Z Dimension
10 Percent	2.0973611953005467	2.0999999999979018	2.0999999999979018
25 Percent	2.2483507470628417	2.2499999999986886	2.2499999999986886
50 Percent	2.5	2.5	2.5
75 Percent	2.7516492529371583	2.750000000001312	2.750000000001312
90 Percent	2.9026388046994533	2.9000000000020987	2.9000000000020987

Figure 12: Features Distribution "Shape Classes - 3" for Segmentation "5x5x5"



Features	X Dimension	Y Dimension	Z Dimension
10 Percent	4.153902322414843	4.199998704322566	4.199998704322566
25 Percent	4.471188951509277	4.499999190201604	4.499999190201604
50 Percent	5.0	5.0	5.0
75 Percent	5.528811048490723	5.500000809798396	5.500000809798396
90 Percent	5.846097677585156	5.800001295677434	5.8000012956774345

Figure 13: Features Distribution "Shape Classes - 3" for Segmentation "10x10x10"

3.4 Classification

The "No Free Lunch Theorem" [20] says, that over all problems, all search algorithms perform the same on average or less technical, no algorithm is the best for all problems. For that reason, instead of selecting a single classifier, I wanted compared a number of them. A great option to do this is the SKLearn Machine Learning package [15] for Python, which provides a large number of classifiers. I decided to use the following classifiers as they were presented in a SKLearn comparison[1], since the comparison already reveals information on the way different classifiers handle different data structures. Additionally, I also adopted the hyperparameters chosen in this comparison. While it is possible to optimize them, with different distribution sets and segmentations, the search space is already extensive, which is why I refrained from including this.

Nearest Neighbors The K-Nearest-Neighbors Algorithm[2] doesn't create a model, but rather stores all training data points. When a new data point gets classified, its k nearest neighbors get calculated. The result will be the class, from which most of the nearest neighbors are.

Linear SVM and RBF SVM The Support Vector Classification[3] works by calculating a hyperplane, which divides the points of two different classes. But sometimes, the data is not split along a simply plane. In that case, a kernel is used to transform the data into another dimension, where it can be split. For the Linear SVM, the linear kernel is used, while for the RBF SVM the the Radial Basis Function (RBF) is used as the kernel instead. For multiple classes, that hyperplane is calculated for each class against all other classes.

Gaussian Process For the Gaussian Process Classifier[4] a probability distribution is calculated for each class, giving the probability that a sample belongs to this class versus that it belongs to any other class. When a new data point is classified, the probability for all classes is calculated and then the class with the highest probability is chosen as a result.

Decision Tree The idea of the Decision Tree[5] is to build a tree based on data by learning certain rules describing the differences between the classes. When building the

[1]SKLearn Comparison
[2]SKLearn Nearest Neighbors
[3]SKLearn Support Vector Classification
[4]SKLearn Gaussian Process Classification
[5]SKLearn Decision Tree

tree, each branch is split by a rule until a leaf is reached. When a new points gets classified, it starts at the top of the tree and then follows the branches. The leaf which is reached in the end is the result.

Random Forest Based on the Decision Tree the Random Forest[6] is, as the name implies, a ensemble of Decision Trees. To determine the best parameters for the trees, different numbers of training data and different numbers of features are used. While single Trees can have significant variances, by assembling them this variance is decreased. The result of a new data point is calculated by averaging the result of all trees.

Neural Net A Neural Net[7] consists of layers with different numbers of nodes. When a feature vector is given to the net the values are passed through the net to end, where each class has one final node. The final values can be influenced through training, where the weights of the edges between the nodes, which the values are multiplied with, are adapted.

AdaBoost AdaBoost[8] follows a similar idea as Random Forest. It samples weaker classifiers, which only have to be better than guessing, by default Decision Trees. Similar to Random Forest, it also creates sub-samples of the training data and which are subsequently paired with weights. Through an iterative process, the weight of the wrongly classified data gets increased so that the weak classifiers are forced to focus on this data, gradually improving.

Naive Bayes The Naive Bayes Classifier[9] is based on Bayes' Theorem. It assumes that the theorem can be applied to classes and features belonging to that class. There are different probability distributions for the probability of a feature to occur given a class. For the Gaussian Naive Bayes, which we are using here, this is the Gaussian distribution. All probability parameters are estimated during training.

QDA Quadratic Discriminant Analysis[10] assumes a distribution for each class and tries to differ between them by determining their respective co-variance matrices, which is then used, similar to the Naive Bayes classifier with the Bayes' Theorem.

[6]SKLearn Random Forest
[7]SKLearn Neural Net
[8]SKLearn AdaBoost
[9]SKLearn Naive Bayes
[10]SKLearn Quadratic Discriminant Analysis

3.5 Experiments

3.5.1 Base Case

To get a general overview on how well the classifiers work for different scenarios, I trained them for the two segmentation sets and the four distribution sets, with 450 samples for training per class and 50 samples for testing. The results can be seen in Figures 14 and 15. Additionally, the precise accuracies can be found in the Appendix A.1.

Nearest Neighbors "Nearest Neighbors" performs extremely well throughout all distribution sets down to segmentation 3x3x3, with the only exception being the 4x4x4 segmentation on the "Shape Classes" set. For "Combined Classes" and "Position Classes", good results are even achieved at 2x2x2. The quality in classification can be explained through the underlying structure of the data. Both the distribution parameters and the resulting features are structured in clusters, which is optimal for "Nearest Neighbors".

Linear SVM and RBF SVM Linear SVM performs well on the "Combined Classes" and "Position Classes" distribution sets, but struggles on the other two, which shows that splitting the values transformed by the linear kernel with one hyperplane is problematic, if the value are too close together. In comparison to the "Linear SVM", the "RBF SVM" performs better or the same in all cases, which shows that the RBF kernel is better suited to transform the class data. This seems to be especially true for cases where orientation is involved, as for the "Orientation Classes" set, the difference is espcially striking with the "RBF SVM" performing well down to the 3x3x3 segmentation.

Gaussian Process Down to the segmentation 5x5x5, "Gaussian Process" delivers an accuracy of 100% for all distribution sets. Below that, the accuracy becomes rather unsteady. Opposed to other classifiers, which almost exclusively see a non-monotone growth of accuracies for growth of segmentation resolution only for the "Shape Classes" set, for "Gaussian Process" this also happens for other sets. A reason for this instability could be the fact that this classifiers calculates probabilities, which can be unstable.

Decision Tree "Decision Tree" delivers an accuracy of 100% for all distribution sets down to a segmentation of 5x5x5 and for sets "Combined Classes" and "Position Classes" down to 3x3x3 an accuracy of 96% and 100% respectively. While the idea of splitting the classes is somewhat similar to the approach of SVM, "Decision Tree" is not fixed to split with only one plane, but allows to find more subtle distinctions, which is especially an advantage for the "Shape classes" set, where the differences between classes can be fine.

Random Forest When comparing the results of "Random Forest" to "Decision Tree", which the "Random Forest" is build on, there are little differences. In general, as the "Decision Tree" already shows a very smooth accuracy curve over the segmentations, there is not a lot of variance, which could be removed by "Random Forest".

Neural Net As other classifiers, the "Neural Net" performs quite well for the "Combined Classes" and "Position Classes". But in contrast to those classifiers, the accuracy drops down to accuracy of guessing for any segmentation below 15x15x15 for "Shape Classes" and "Orientation Classes". The issue of data points of different classes being too similar is especially influential here, as the net has a large number of parameters.

AdaBoost The results for "AdaBoost" are very similar to the ones of "Decision Tree" and "Random Forest", since it uses Decision Trees as well for base classifiers. The only expection here is the "Combined Classes" set, for which the highest accuracy is 75%. This probably happens because of the way "AdaBoost" samples data packages, as the single classes are very different, and if one of them is underrepresented in the data packages with high weights, the classifier has a problem to deal with them.

Naive Bayes The "Naive Bayes" is another high performing classifier, having good accuracies among all distribution sets. With the features modeling Gaussian distributions and the parameters for each class coming from a uniform distribution, they are in most cases quite easy for the "Naive Bayes" to model, which, as for other classifiers, becomes difficult difficult once, the data points of different classes are too similar. This can be seen for segmentations from 4x4x4 in the "Shape Classes" set.

QDA The "QDA" classifier performs generally weak. There is no good result achieved for any distribution set for a segmentation below 5x5x5, and for the "Shape Classes" set, no accuracy goes above 70%. One reason why this approach doesn't work two well might be that it uses variances for classification. With the features, especially for lower resolution segmentations, only being rough representations of the actual data and the data including significant variance as well, this might be a difficult task.

General Results A major observation here is that all classifiers perform best on the "Position Classes" distribution set, with all classifiers except for "QDA" achieving an accuracy of 1.0 for all segmentations except for 1x1x1, since all classes lead to the same measurement. Moreover, most classifiers are performing well on the "Combined Classes", which, despite also drawing from other characteristics, heavily relies on position. The overall best performing classifier is "Nearest Neighbors". Besides that, the highest necessary resolution seems to be 10x10x10, as only "Neural Net" and "QDA" see an increased accuracy with a higher resolution. The lowest resolution where we can still see an acceptable accuracy at least in some classifiers is 3x3x3.

As mentioned before, the "Advanced" segmentation set was created with the goal to examine the effect of having a higher resolution in only one or two dimensions. One interesting result here is that the dimension in which the resolution is increased can have a strong influence. When looking at the "Shape Classes" distribution set and comparing it with the "Basic" segmentation set, the relation $accuracy[20*3*3] \geq accuracy[10*3*3] \geq accuracy[5*3*3] \geq accuracy[3*3*3]$ can be seen, which is shown most clearly for the classifiers "RBF SVM", "Linear SVM" and "Neural Net". This means that increasing the resolution in X dimension does increase the accuracy, which is related to the major differences between classes of the "Shape Classes" distribution set being found along the X-axis. On the other hand, in general the reversed relation $accuracy[3*20*3] \leq accuracy[3*10*3] \leq accuracy[3*5*3] \leq accuracy[3*3*3]$ can be observed with a few exceptions. This shows that a higher resolution in a dimension, which is not carrying significant information just introduces higher noise and complexity. In conclusion, it makes sense in some cases to partially increase the resolution.

Another factor when it comes to choosing a classifier is training time. If a classifier takes too much resources for training it might simply not be viable. The results for how long it took the classifiers to be trained and tested can be seen in Table 5. The by far longest training time was measured for "Gaussian Process", at around 30 minutes for the "Combined Classes" distribution set. "Neural Net" took the place as second slowest classifier, taking slightly more than 6 seconds to train on the "Combined Classes" set. All other classifier stay well below 1 second. Given that "Gaussian Process" only demonstrates a mediocre performance in comparison with other, significantly faster algorithms, it does not look worthwhile pursuing in this case.

Distribution Set	Nearest Neighbors	Linear SVM	RBF SVM	Gaussian Process	Decision Tree	Random Forest	Neural Net	AdaBoost	Naive Bayes	QDA
Combined	0.0038	0.0116	0.0599	1746.9334	0.0216	0.0208	6.1798	0.3602	0.0015	0.0023
	0.0105	0.0037	0.0041	5.0628	0.0003	0.0015	0.0012	0.0102	0.0006	0.0007
Orientation	0.0014	0.0369	0.0853	54.9823	0.0028	0.0294	1.1300	0.1566	0.0007	0.0009
	0.0039	0.0033	0.0062	0.2205	0.0005	0.0027	0.0008	0.0074	0.0003	0.0003
Position	0.0109	0.0123	0.2230	1715.0908	0.0263	0.0227	5.7916	0.3992	0.0014	0.0022
	0.0277	0.0026	0.0079	4.9111	0.0003	0.0015	0.0013	0.0101	0.0006	0.0006
Shape	0.0014	0.0368	0.0839	80.1319	0.0019	0.0302	1.1635	0.0900	0.0009	0.0011
	0.0037	0.0032	0.0057	0.2322	0.0005	0.0027	0.0004	0.0080	0.0004	0.0004

Table 5: Training and Testing Duration [s] for Segmentation 5x5x5

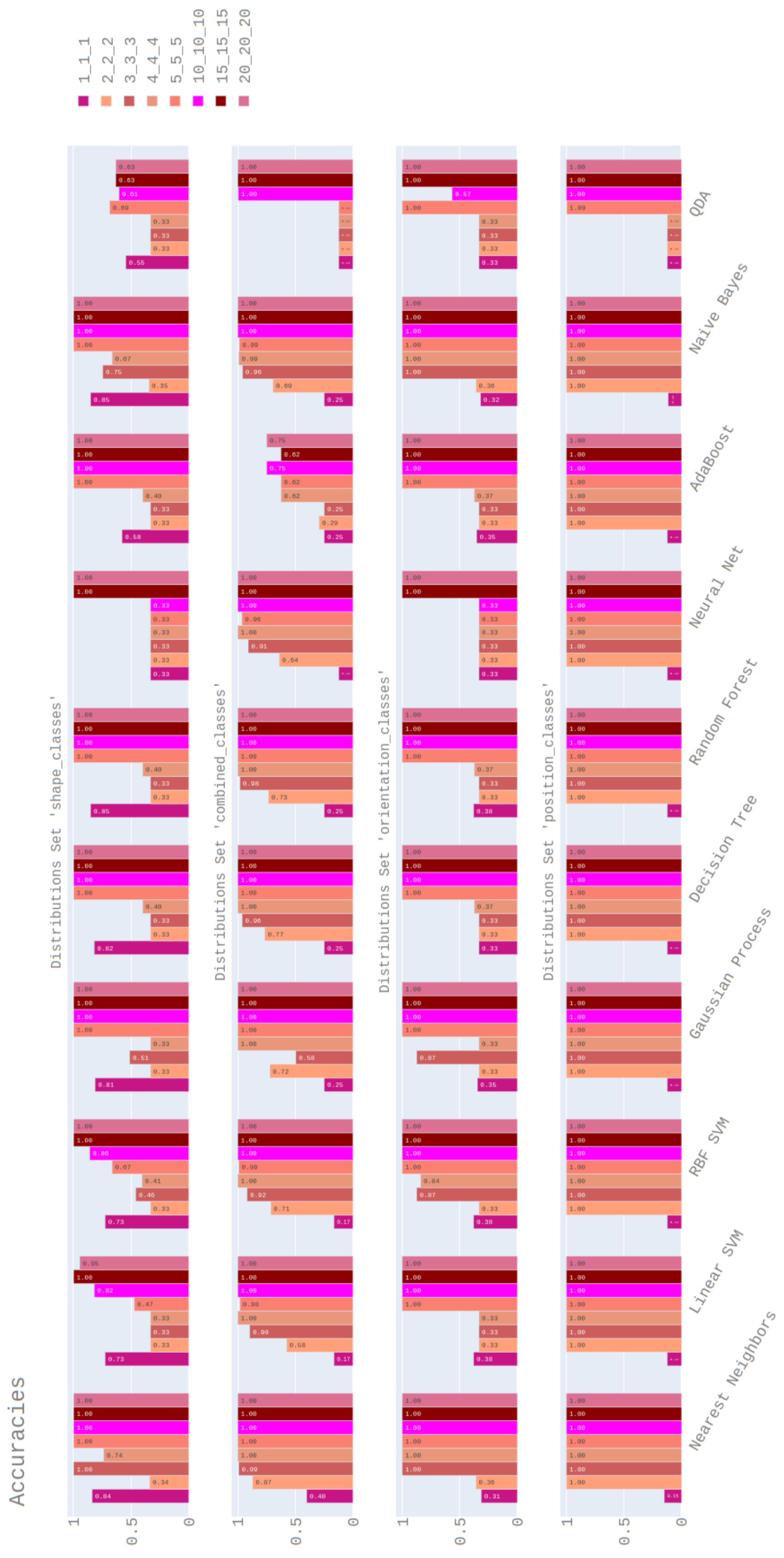


Figure 14: Base Case Results for Segmentation Set "Basic"

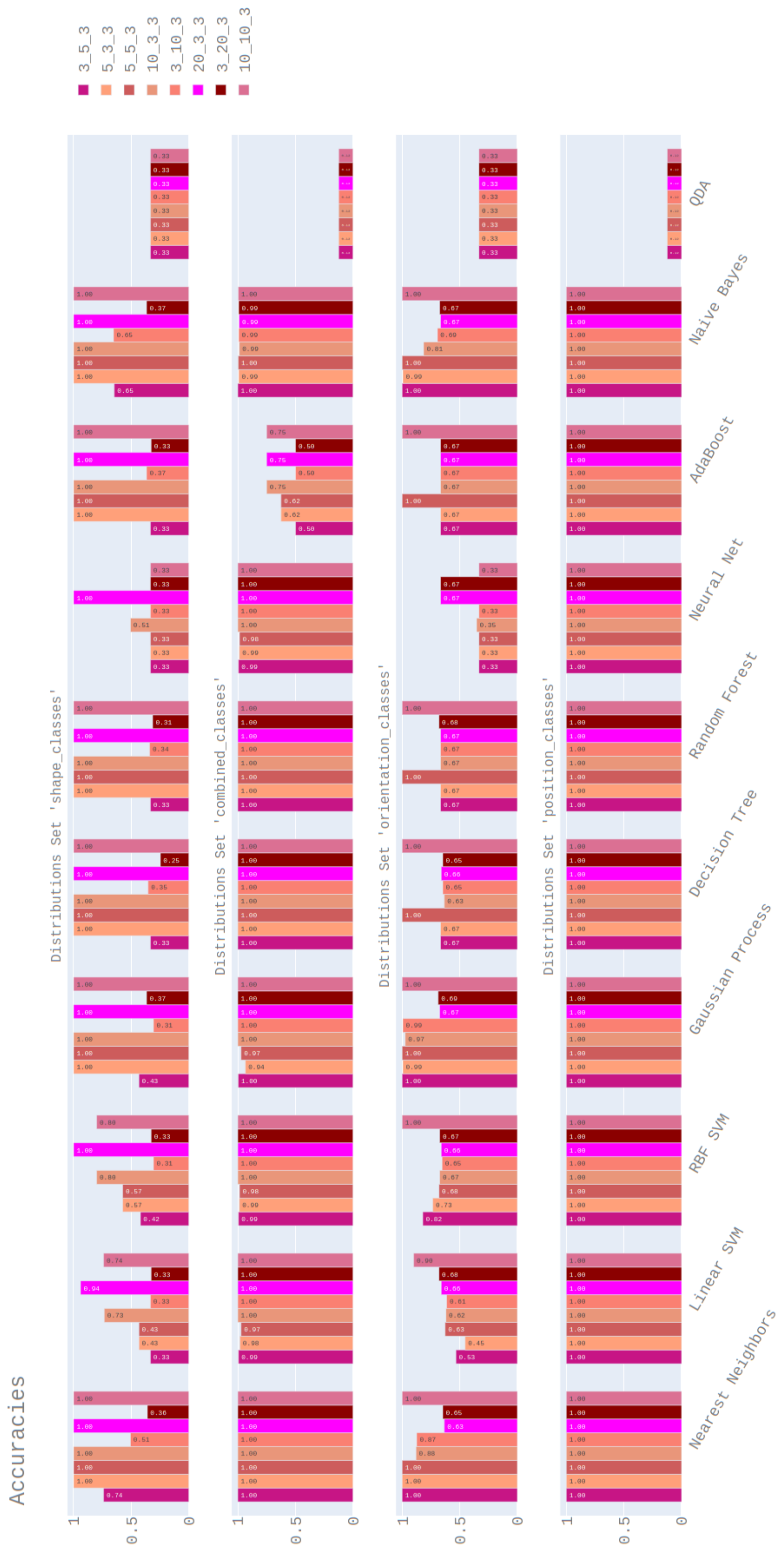


Figure 15: Base Case Results for Segmentation Set "Advanced"

3.5.2 Changing Amount of Training Data

As mentioned before, for the base case I used 450 samples to train the classifiers. While in this case I could create as much data as I want, in real cases that amount of data is usually limited, which is why it is especially important to check how well the classifiers perform with less data. To do this I trained all classifiers with 25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 350, 400 and 450 samples, once again with all four distribution sets. To keep the results clear, I used for any further experiments only the segmentations 3x3x3, 4x4x4, 5x5x5 and 10x10x10 of the "Basic" segmentation set, as the base case hinted them to be the most interesting. The following classifiers were selected to display different possible behavior for a varying amount of training data.

Results The classifier "Nearest Neighbors", which can be seen in Figure 16, shows a steady development with a small drop in accuracy below 75 samples for the "Combined Classes" distribution set. The only striking deviation from this is the development of the 4x4x4 segmentation for the "Shape Classes" distribution set, which seems to be extremely unstable above 100 samples and slowly, but steadily decreases below.

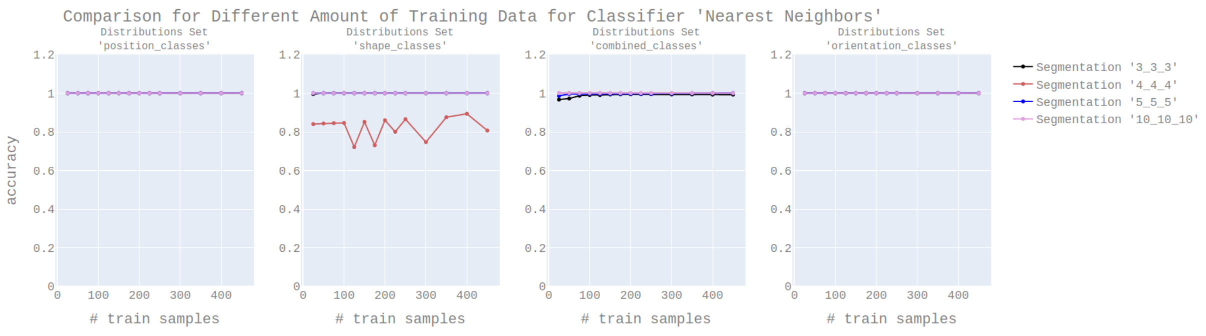


Figure 16: Influence of Amount of Training Data for Nearest Neighbors Classifier

An interesting behavior can also be seen for the "Neural Net" classifier, as displayed in Figure 17. While for the "Position Classes" and "Combined Classes" distribution sets the accuracy is overall steady, for the other two distribution sets it shows some spikes in accuracy for lower amounts of training data, but those are highly inconsistent.

The "RBF SVM" classifier, as shown in Figure 18, partly exhibits the behavior that is generally expected with a decreasing amount of training data. While for the "Position Classes" and "Combined Classes" distribution sets the accuracy is once again rather steady, the other two distribution sets show a trend. For the "Orientation Classes", the two segmentations 10x10x10 and 5x5x5 show rather stable accuracies, while the accuracies for 4x4x4 and 3x3x3 show a negative trend, especially below 75 samples. For the "Shapes classes", this is swapped and segmentations 10x10x10 and 5x5x5 show a decline

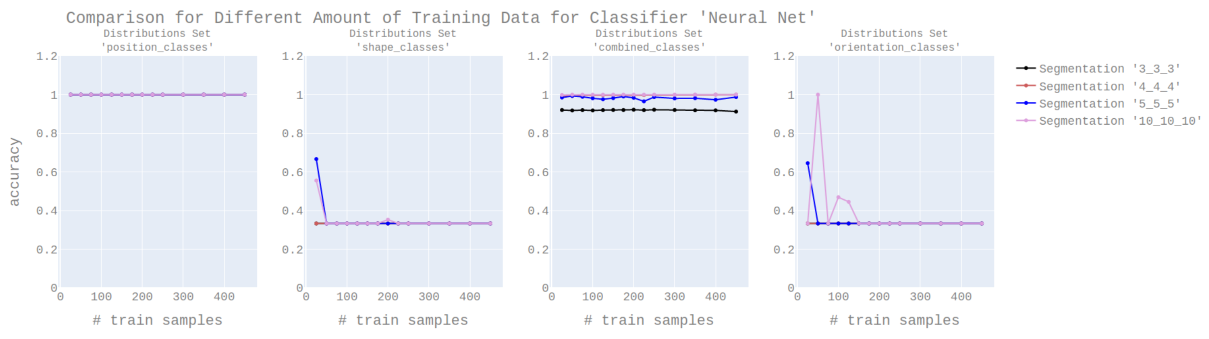


Figure 17: Influence of Amount of Training Data for Neural Net Classifier

in accuracy with a decreasing amount of samples, while the accuracies for 4x4x4 and 3x3x3 are stable and even rising to a certain extent.

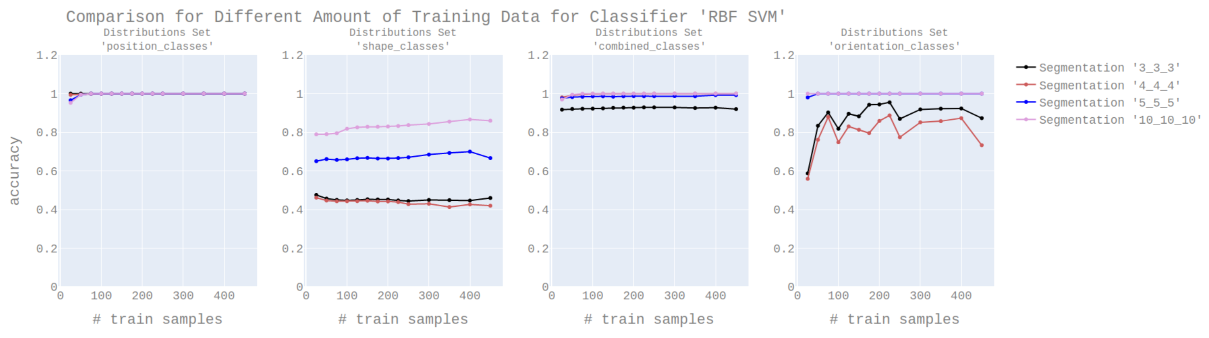


Figure 18: Influence of Amount of Training Data for RBF SVM Classifier

The results for all other classifiers can be found in the Appendix A.2. An important conclusion here is that the approach works fairly well for even for lower amounts of training data for some classifiers, which is quite important for real-world applications. But the turbulences of accuracy for some classifiers shown in the Appendix also demonstrate the selection of data can be quite important, as those strong variations are obviously not simply the result of more of less data, but rather of certain data points being added or taken away.

3.5.3 Incomplete Data

Until now I always assumed that the data we can use for training is perfect. But in realty, there are numerous problems, which might occur while gathering the data. One of those problems is the loss of data, may it be in transmission or processing. Because of this, I want to make sure that the approach is robust with regard to the incompleteness of data. To do so, after drawing values from the distribution, each value is deleted by a certain probability given as the incompleteness probability. Figure 19 shows the effect of an increasing incompleteness probability.

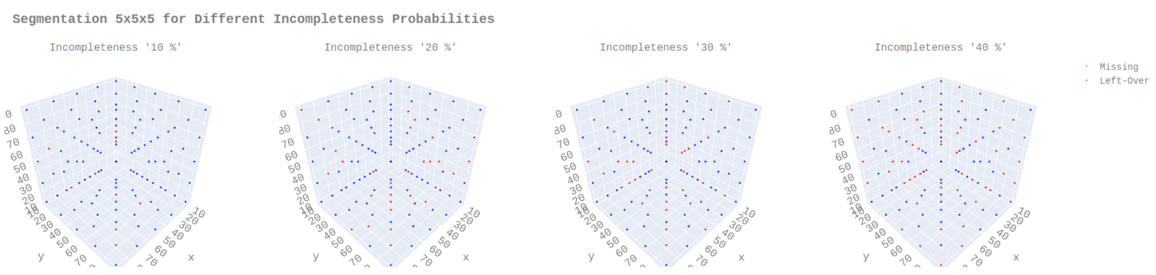


Figure 19: Segmentation with Incomplete Data

There are different ways to deal with missing data. One option would be to simply leave the data point out, but this is not possibble the way my features work. Another way is to set the data point as '0'. But this would distort the result, which is why instead I fill the missing points with a value based on their surrounding points.

The method for this is shown in Figure 20. All data points in a cube around the missing data point are first checked for eligibility. If they are neither outside the grid, nor missing a value themselves, they are added to a list, from which the mean is then calculated. What is shown in the Figure in two dimensions happens in reality with three dimensions. In this examples, the new value for our missing data point would be:

$$\frac{1.5 + 1.5 + 3.0 + 2.0}{4} = 2$$

The analysis is based on the same distribution sets, segmentations and classifiers as the analysis in 3.5.2. To examine to influence of incomplete data, the accuracy was measured for incompleteness probabilities ranging from 0% to 40% in steps of 5%, with the amount of training samples rising from 50 to 450 samples in step of 50 samples. For the analysis I will focus on the "Nearest Neighbors" classifier, as it presented the best results so far, which makes it the most interesting working towards a robust, working system. Plots of the results for other classifiers can be found in the Appendix A.4. For this experiment (as for the following), I refrained from training the classifier "Gaussian Process", as the extensive search space would let the training last an unreasonable amount of time.

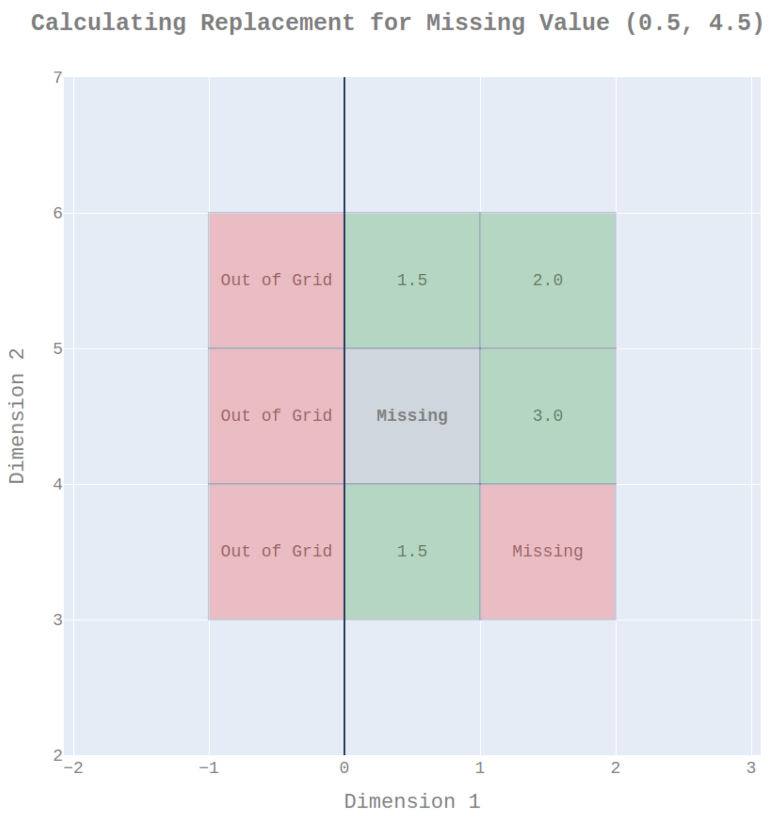


Figure 20: Dealing with Incomplete Data

Results As it can be seen in Figure 21, for the two distribution sets "Position Classes" and "Combined Classes" the results are very good, with only the 3x3x3 segmentation seeing drops in accuracy. As for the base case, results for the "Combined Classes" are slightly worse than for the "Position Classes", showing that position characteristics remain important for high accuracies. For the "Shapes Classes" and "Orientation Classes" distribution sets the accuracy slowly decreases with rising incompleteness, while the amount of training samples plays practically no role. What can be observed though is that the results for for the 4x4x4 segmentation are worse than for 3x3x3 in the case of "Shape Classes" and "Orientation Classes". Apart from that, the trajectories are quite similar.

With the given results in mind, incompleteness seems to be an important factor, and should be kept as low as possible, as a low resolution still performs very well, as long as the incompleteness is kept at a low level.

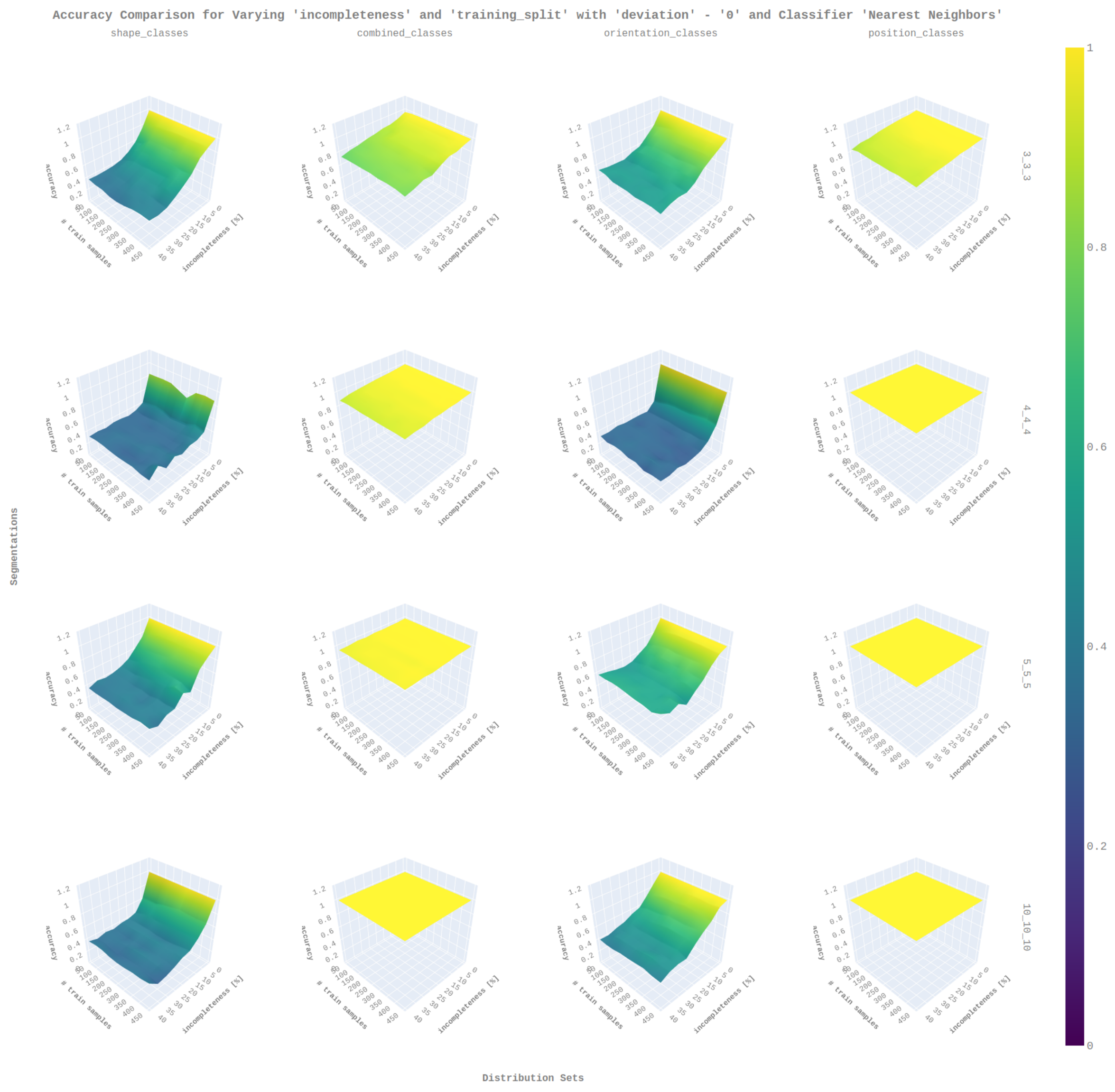


Figure 21: Results with Incomplete Date for Nearest Neighbors Classifier

3.5.4 Deviation in Segmentation

So far, all data points were taken from precise locations, exactly in the center of each segment. But this premise is unrealistic for data taken by a mobile robot, as the goal path will only be approximated. To simulate this behavior, I introduced deviation for segmentations. For each point of a segmentation, a 3D Gaussian distribution is created, with the mean values being the original coordinates of the point, and the variance in each dimension being the deviation value. Every time the segmentation is used to draw values from a contamination distribution, first values are drawn from the many distributions around the segmentation points. How this looks like for different levels of distributions can be seen in Figure 22. While for a deviation of 1 cm the off-set is still fairly low, it is quite significant for a deviation of 4 cm.

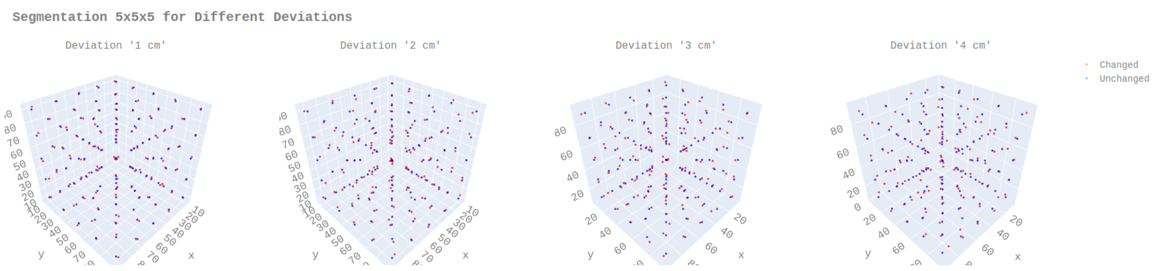


Figure 22: Segmentation with Deviation

Once again, the same distribution sets, segmentations and classifiers were used as in 3.5.2. The deviation grows in steps of 0.5 cm from 0 cm to 4 cm and is matched against the training rate, which ranges from 50 to 450. Again, I focused on the "Nearest Neighbors" classifier, the results for the other classifiers can be seen in the Appendix A.4.

Results The results, as displayed in Figure 23, show that for the distribution sets "Combined Classes" and "Position Classes" the approach still yields an accuracy of essentially 1. Moreover, this time for for distribution sets "Shape Classes" and "Orientation Classes" an accuracy of 1 is achieved for segmentations 3x3x3 and 5x5x5, while 4x4x4 and 10x10x10 show significantly worse accuracies. This alternating pattern is an amplification of what was already shown in 3.5.3. The reason why the segmentations with a even number perform poorly here is likely that neither of them have one segmentation point directly inside the center of the contamination distribution, which is especially for those two specific distribution sets the probably most expressive value as all changes between changes happen with a very small margin. Additionally the amount of training data only has a limited influence. For 4x4x4, where the accuracy drops down to a plateau for all deviations higher than 0.5 cm, an increased amount of samples changes nothing, while for

10x10x10, where the accuracy drops much slower and stays at a higher level, the increase in training samples leads to an increase in accuracy. The take-away here is that for most cases, where the different contaminations are expected to be distinct in a way similar to the "Combined Classes" distribution set, deviation only plays a minor role. Nevertheless, it it advisable to keep the deviation as low as possible and configure the segmentation to fit the contamination.

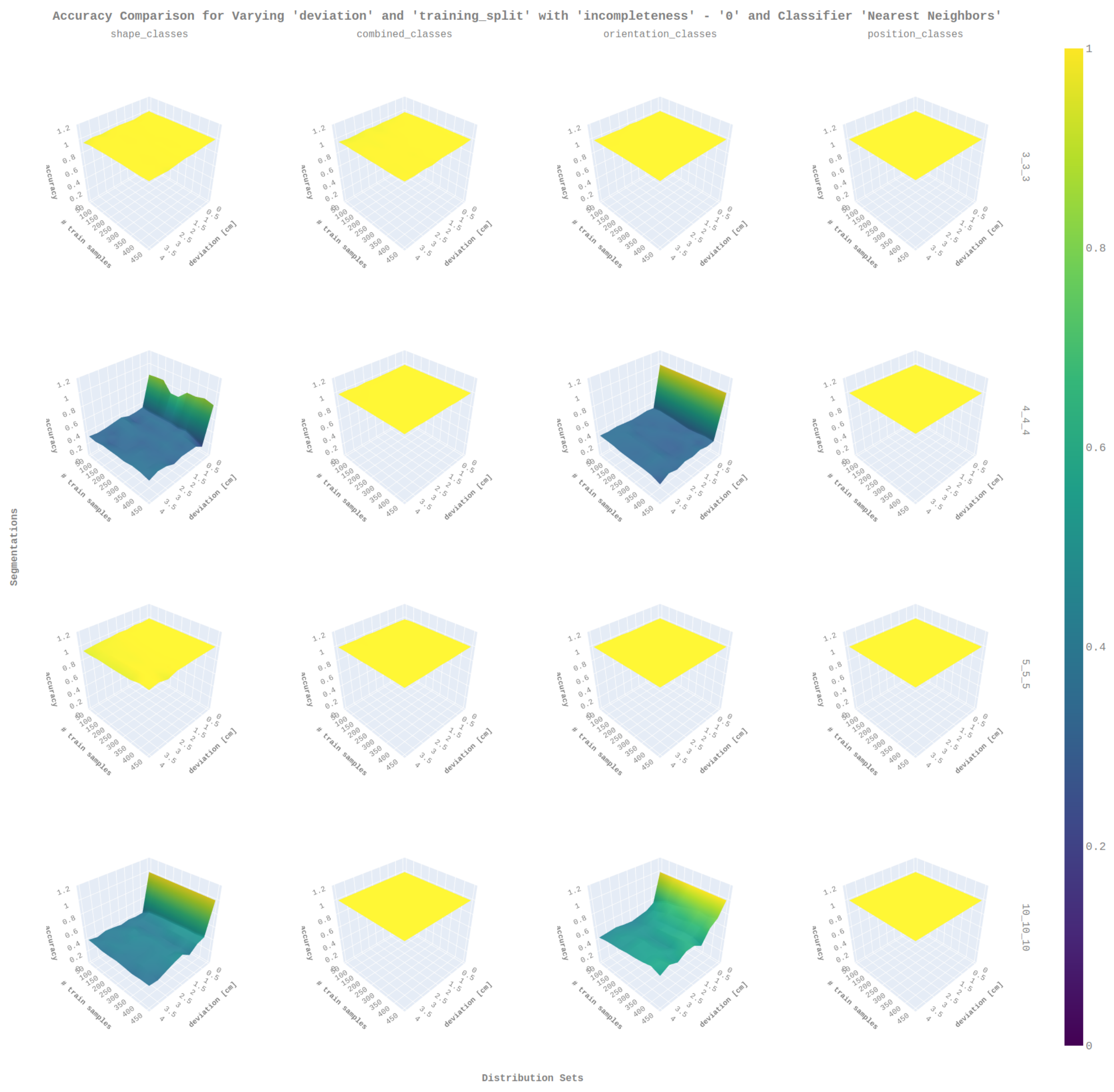


Figure 23: Results with Deviation for Nearest Neighbors Classifier

3.6 Summary

To sum up this section, so far I introduced different segmentations and distributions, a feature system and different classifiers. The experiments conducted show that the "Nearest Neighbors" classifier works best on the data I simulated, and is quite robust with regard to the amount of training data, incompleteness of data and deviation of the measurement positions. Nonetheless, the experiments also showed that imcompleteness and deviation should be kept as low as possible, as well as that the accuracy also depends on smart choices of the segmentation. This is besides the general resolution also important for increasing the resolution only in one particular direction, which in some cases increased the accuracy and in others decreased it. Generally speaking it also became evident that the positional property of a distribution is the most expressive and leads to the highest accuracies. Now these insights can be incorporated into the requirements of the robot, which follow in the next section.

4 Robotics

In the following I will detail the respective requirements for each part of a Autonomous Underwater Vehicle (AUV), that is suitable for the use with the proposed system. The goal is to create a system which is completely wireless in order to disturb the environment as little as possible, as well as small enough to fit into the environment and capable of exploring it with regard to movement and sensing. To do that, this section will also include possible technologies to meet the requirements and direct reference to the results from Section 3.

4.1 Electronics and Computation

A good choice for the computational robot would be a Raspberry Pi 3 or 4. For once it offers more than necessary computational power for almost all imaginable tasks and is capable of running a standard OS like Ubuntu, which in return allows the implementation of countless software packages. Besides that, it has Wi-Fi, connections especially for Raspberry Pi Cameras, the HAT adapter which allows other boards like motor-controllers to be connected. To connect further periphery devices such as antennas, I2C and SPI are available as well.

Since the System will operate underwater, waterproofness is another important topic when it comes to electronics. Most underwater thrusters can be placed outside of the body, but they still need to be connected to a power source inside the body. In some cases water might travel inside the cabels if the motor is not waterproof, which for example brushless motors are not. To protect the other electronic components they should be coated with a sealant.

Regarding power supply it should be expected that two kinds of batteries are necessary. The Raspberry Pi require 3.3 V to 5 V, while most motors will require 6 V to 24 V. While it is possible to reduce voltage, elements which can be used for this are either big or will generate a lot of heat, which can become a problem in a closed system.

4.2 Propulsion and Steering

Propulsion and Steering heavily relies on the environment, which the robot has to operate in. While robots operating in large marine waters often are equipped with fins[11], in small water bodies where we have a need for precise movements, a thruster-based system is more viable. Since the robot should be as small as possible, the minimal number of three thrusters would be optimal. There are different configurations, which can be found for

[11]International Submarine Engineering Ltd. Explorer AUV

example in "The ROV Manual" by Christ and Wernli [3]. To work properly the thruster have to be placed carefully to avoid any imbalance as the deviation can have a significant effect on the classification of the data taken by the robot as shown in 3.5.4.

4.3 Communication

Although it is possible to communicate data through water, the data rate is usually quite slow and it is necessary to include intense testing of all data packages to ensure the data is complete and free of corruptions, which also takes time and can't always be guaranteed. As it was shown in 3.5.3, it is important to rate of incomplete data as low as possible to ensure a high accuracy. Moreover, those systems are sometimes quite expensive and big as well. Because of that I suggest to refrain from using underwater communication and instead include a short time of surfacing and transmitting all data through Wi-Fi after each tour. This also allows for the use of high-level communication and messaging systems such as MQTT and Kafka. This specific part will be subject of Section 5 among others.

4.4 Localization

An essential part of the robot is its localization system. There are several methods for localization, and they all have different advantages and disadvantages. One option is to start from a known position and then iteratively add the distance traveled, which can either be calculated based on the known revolutions of the thrusters or the double-integrated acceleration measured by an Inertial Measurement Unit (IMU). While this sounds like a very nice closed system, unfortunately using the thrusters to calculate the traveled way is extremely inaccurate and the IMU has a significant drift, which means that a system using the environement is needed. Using beacons sending EM waves to do this is one option, which for example is used by the GPS. A general problem is that EM waves attenuate much quicker in water than they do in air, which is why GPS does not work underwater below a few centimeters. Two different categories of EM-wave based underwater localization are Long Baseline (LBL) and Ultra Short Baseline (USBL). For the LBL systems, there are multiple beacons sending their signals, which are received by the robot. Depending on the method for calculating the distance, the robot can also send back a signal to be received by the beacons. Based on the distances between the robot and the beacons, the robot's position can be calculated through trilateration is discussed in 4.4.2. For the USBL systems use only one beacon sending a signal to the robot, which is direction sensitive. Since the USBL method is more complex in design, the LBL seems to be the simpler approach. There are different kinds of EM waves, for example ultrasonic and ISM band. Since ultrasonic transducers are either extremely limited in sending angle or extremely expensive, transducers using the ISM band seem like the more reasonable choice. Finally, there is also the possibility to combine several localization systems through a Kalman Filter, which increases the overall accuracy.

4.4.1 ISM Band

Two popular frequencies in the ISM Band are 2.4 GHz and 433 MHz. To understand how useful they are for localization, I examined their transmission properties. An important measurement here is Decibel (dB). In the context of wave transmissions, it describes the loss of wave energy.

$$dB = 10 * log(\frac{P_1}{P_2}) \Leftrightarrow \frac{P_1}{P_2} = 10^{\frac{dB}{10}}$$

where P_1 is the power of the wave at point 1 and P_2 the power at point 2.

According to Qureshi et al [17], the underwater path loss is 26.5 dB per 0.4 m for an EM signal with a frequency of 2.4 GHz. If this value is inserted into the equation, we obtain a relation $\frac{P_1}{P_2} = 446$, which means that the measured power of the wave will be 446 times smaller than the original signal after 0.4 meters. While this value might still seem acceptable, after 4 m it reaches $316 * 10^{24}$, which is completely unrealistic.

In comparison to that, Muhammad M.S. et al [14] calculated the underwater path loss for 433 MHz to be 20 db per 1.0 m, which means that relation between the power levels at points 1 and 2 is 100 after 1 m and $100 * 10^6$ after 4 m, which might sounds impossible as well, but actually is still viable with a fitting transducer.

4.4.2 Trilateration

Independent of the method chosen, we end up with having the distance to every individual beacon. To calculate the robot's position, we will now use a method called trilateration. The goal is to find the position by calculating the point, which fulfills all distance requirements. This can be done either mathematically or through optimization. Since the mathematical solution requires the distances to be exact, the optimization approach should be used, which minimizes the error.

4.5 Sensing

In Section 3 contamination was treated as something abstract, but now with regard to robotics, it is important to consider which kind of contamination is supposed to be monitored, as this is basis for choosing a proper sensor. In the case of aquaponics for example this might be done through a Microbial Fuel Cell, which can be used to detect and identify biological contamination [5]. General requirements for the sensing would be a high accuracy, and a extended range for measurements, as it is necessary to have a complete depiction of each spatial feature. A very versatile sensor is a camera, which can be used to find all contaminations, which are also visible to the human eye. But eventually the decision of which sensor to use depends on the use case, and has great potential for including expert knowledge.

4.6 Path Planning and Navigation

Path planning is a difficult issue and heavily relies on the waypoints which are supposed to be traveled along. Since those waypoints are the centers of the single segmentations, the path depends on the respective segmentation. In case of a cuboid segmentation, the path can be easily set as shown in Figure 3. For more complex waypoint structures in a arbitrary graph, there are different approaches as for example presented by Petres et al [16]. But those algorithms base their result on a set length for every edge within the graph, mostly the euclidean distance. But when examining the movement in such a small space, other factors such as the turning circle of the robot play a role as well, which might elongate the way for small angles between edges and increase the deviation between the point where the robot should take the measurement and where it will actually take it. To include this problem in path planning and find the fastest and most accurate way through all waypoints, one option would be to use Reinforcement Learning (RL) with a simulation of the robot to learn the best actual path, how it is done by Wang et al [19].

4.7 Control

With the single components treated, the next step is establishing the control of the robot. On the one hand that means the high-level control of tying all elements together, for example running the communication and localization of the robot, navigation and low-level control of the thrusters. This can be found in the Robot Operating System (ROS), which offers an abundance of different packages for the afore mentioned and additional functionalities such as a simulation suite. The low-level control of the thrusters is tightly connected to the navigation and path planning, as it all about keeping the robot on track. Once again there are several options, as they are for example presented by Ye et al [11]. The low-level control to work properly, in many cases the hydrodynamic parameters of the robot need to be determined first, which is presented by Chen et al [2]. Other approaches even promise robust control without the need for parameters [21], which, like Path Planning, can also use RL.

4.8 Design

As stated before, the robot should be as small as possible, so that it can maneuver and operate in limited space. To achieve this the body volume should be limited to the absolute minimum, only leaving space for the main components. Additionally, attachments for the thrusters are required as well if they are note even completed integrated into the body. Many existing underwater robots use acrylic tubes, but in this case a custom designed body is probably necessary, as tubes often waste space.

4.8.1 Fabrication

An easy to implement, quick and cost-effective way of fabrication is 3D printing. 3D printing offers a variety of different materials and printing methods. One of the most popular and available methods is Fused Deposition Modeling (FDM), which deposits layer after layer of material. A popular choice for the material is ABS, which offers great durability and stiffness. Because of the nature of layering, there will always be tiny spaces within the printed body, tarnishing its waterproofness. One option to deal with this would be to coat the body, but evenly distributing the coating over a complex surface is quite challenging. A second option is to print a mold, which then can be filled with a material that offers the same durability, but is waterproof, for examples epoxy resin. In Figure 24, the process of molding is depicted. First the mold itself is printed, and then afterwards filled with the molding material. For this demonstration I used PDMS. Afterwards the mold is broken off, and the molded body is freed. As it can be seen, there are some small air bubbles trapped inside the body, but those can be removed through the use of a vacuum chamber before the mold is left for curing.

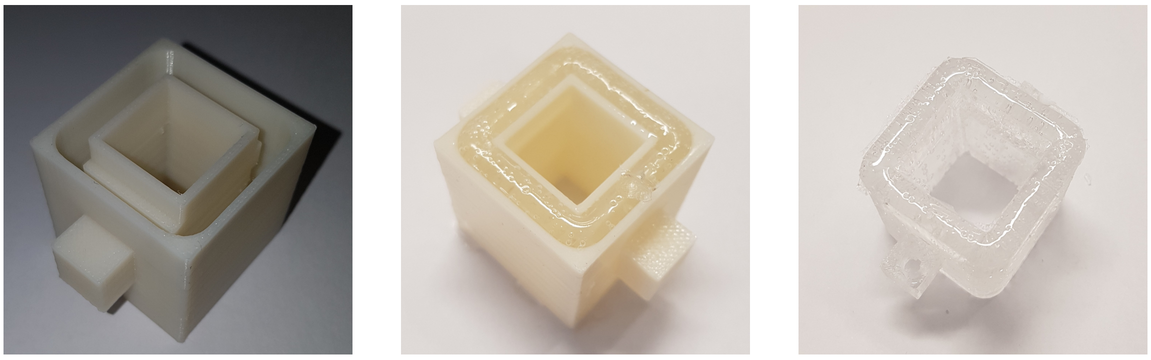

(a) Empty Mold (b) Filled Mold (c) Molded Body

Figure 24: Molding Process

For this demonstration we used a sacrificial mold, which was destroyed in the process on freeing the body, but it is also possible to design a mold, which consists of several parts, so it can be simply disassembled, which is less work and the mold can be reused as well.

5 System Setup

5.1 Production Setup

With a valid approach for analyzing data and a list of requirements for the robot, the next step is now to connect those two parts and create a production setup.

First, it is necessary to setup a server, on which the analysis can run. The choice of a machine for the server heavily depends on the amount of data and number of operations which should be run simultaneously, but in most cases, a Raspberry Pi 3 or 4 should be enough like for the robot. This server can be run for example with Ubuntu 18.04 or Ubuntu 18.04 Server, the lightweight version without a GUI, as its OS, but this is up to the personal preference.

Now that the server and the robot are in place, a way of communication must be established. We achieve this through the use of the MQTT protocol. This protocol allows different components to publish on and subscribe to different topics over a broker. A good choice is the Mosquitto broker. Since the components exist in a distributed system, a bridge with two brokers is needed, which communicate with the local components and sending messages between each other, allowing communication between remote components.

The final setup can be seen in Figure 25.

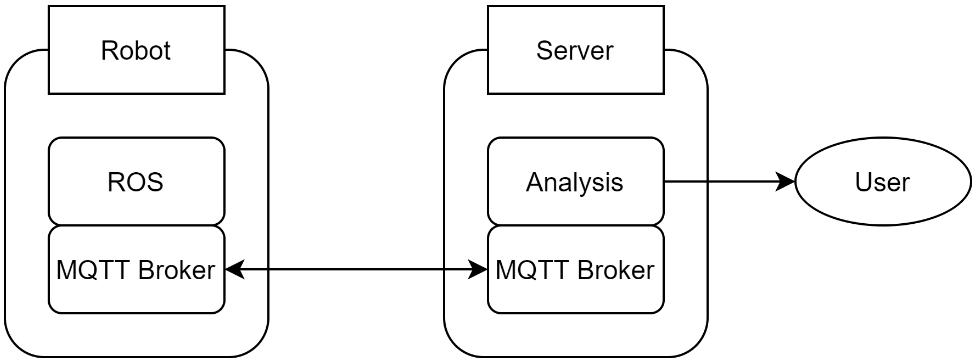


Figure 25: Production Setup

5.2 Analysis Implementation

In the following, all functions used for this thesis will be explained, as well as how to use them for other applications. The code can be downloaded from Github. All scripts which are used for the analysis can be found inside the "analysis" package, while all other files are placed inside the "analysis_files" directory.

Additional functions, which are included with the intention of developing them in the future, but are not included in the approach detailed in Section 3, are included the files starting with "_future_".

To create all necessary directories, the "setup.py" script has to be run. To create all necessary directories for the future scripts as well, use "setup.py –f". Any new scripts added by users using the given experiments should be added in the "TrainingCenter" directory, due to files being taken from and saved in the directory "analysis_files". Besides the following explanations, the code is completely documented through docstrings, with the exception of the future scripts.

To carry out the experiments described in Section 3, I also included some functions, which iterate the following functions over multiple instances. For example, the function `create_cuboid_segmentation()` in "sampling.py", which creates a single segmentation and is explained in detail in 5.2.2, is used in the function `create_segmentations()` to create all segmentations of multiple segmentation sets. Those functions can be found in "experiments.py". A few of them are showcased in "run.py". General functions, which are needed by multiple other functions across files can be found in the "functions.py" file

5.2.1 Environment

Each environment is saved in the JSON format inside the "environments" directory. While right now there is only one environment, and the approach only works for cuboid environments, the structure of the environment file is designed to be compatible in the future for other shapes as well. The currently used environment is described as follows:

```
{
  "basic_shape": "cuboid",
  "params": {
    "length": 100,
    "width": 100,
    "height": 100
  }
}
```

All files are intended to have a "basic_shape" and a "params" parameter, but while the former should always be a simple string, the latter can look different for every shape, for example in the case of a cylinder having "diameter" and "height" instead of "length", "width" and "height".

5.2.2 Segmentation

In the first step, a segmentation set has to be defined in a file of the JSON format. As seen in the following example, the file is structured as a dictionary, where the key is the name of the segmentation and the respective value is a list containing, in order, the number of segments the environment should be spit into in X, Y and Z dimension.

```
{
  "3_3_3: [3, 3, 3],
  "4_4_4: [4, 4, 4],
  "5_5_5: [5, 5, 5],
  "10_10_10: [10, 10, 10]
}
```

As shown in Figure 26, function `create_cuboid_segmentations()` in "sample.py" takes a cuboid environments and segments it into smaller cuboids based on given parameters, calculates the centers of the segments and then saves the centers in an array as follows:

```
[[0, 0, 0, 16.66667, 16.66667, 16.66667],
 [0, 0, 1, 16.66667, 16.66667, 50],
 ...
]
```

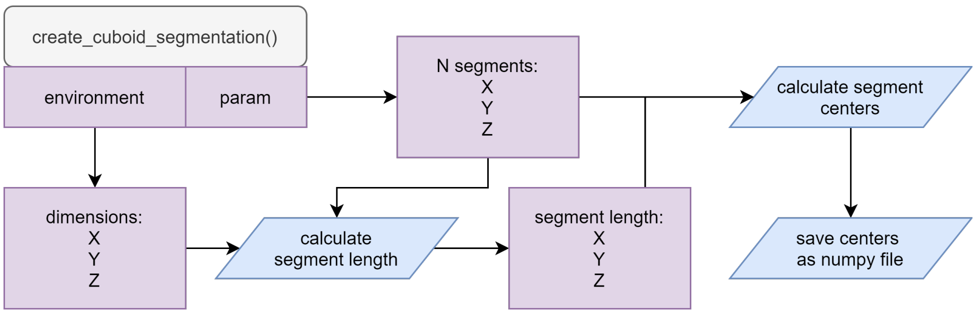


Figure 26: Segmentations Flow

5.2.3 Distributions

Just as the segmentations, the distributions are saved in the JSON format as well. The following example contains the information for the "Shape Classes" set as described in Table 2.

```
{
  "1": [[50, 50], [50, 50], [50, 50], [5, 10], [5, 10], [5, 10]],
  "2": [[50, 50], [50, 50], [50, 50], [15, 20], [5, 10], [5, 10]],
  "3": [[50, 50], [50, 50], [50, 50], [30, 40], [5, 10], [5, 10]]
}
```

Figure 27 show how the function `create_distributions()` in "sample.py" works. It is given the ranges for one class, i.e. one row in the JSON file shown above and the number of samples to be drawn from those ranges.

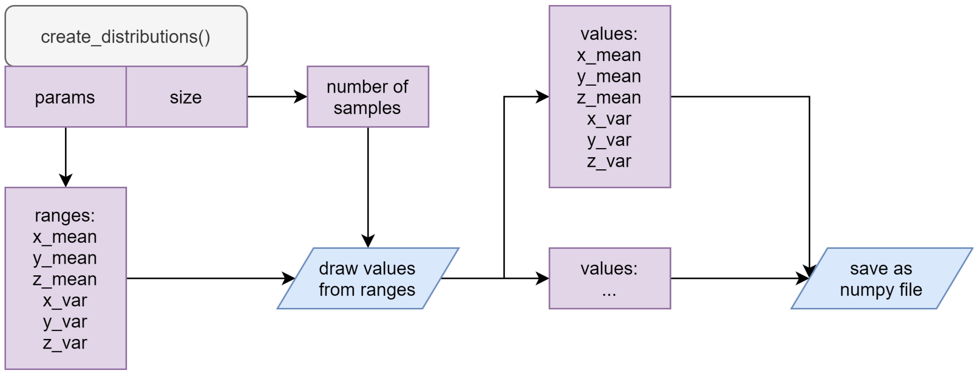


Figure 27: Distributions Flow

5.2.4 Features

The creation of features happens through three different functions, `generate_features()`, `generate_features_with_incompleteness()` and `generate_features_with_deviation()` in "features.py", which can be seen in Figures 28, 29 and 30 respectively. They share the basic principle of drawing values from the distributions, creating grids and afterwards features through the function `turn_grid_into_features()` from "functions.py", which can be seen in Figure 31. To simulate the incompleteness, values are randomly completed and then replaced through mean values, as shown in Figure 20. Deviation is included by introducing distributions for segmentation centers. For each distribution, the segmentation centers are drawn again, and then used to draw values from the contamination distribution.

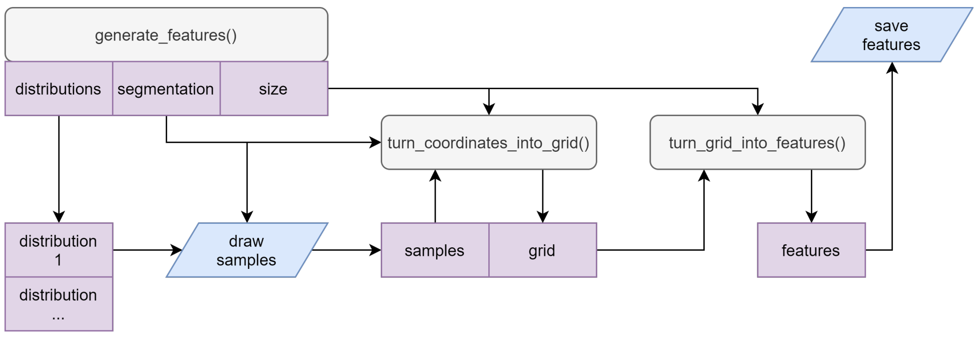


Figure 28: Features Flow

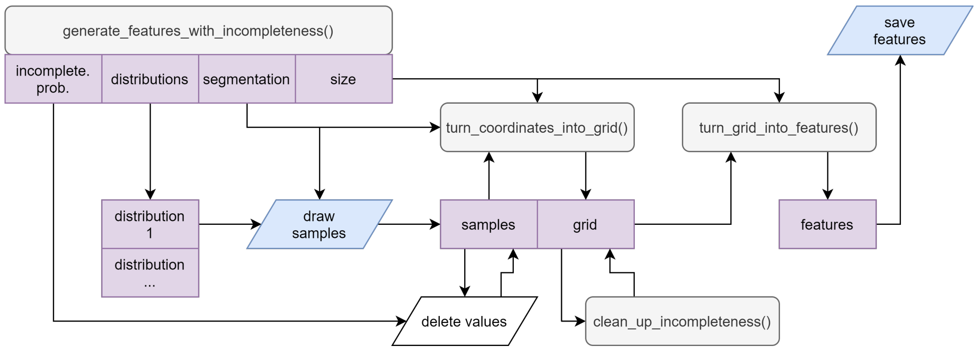


Figure 29: Features with Incompleteness Flow

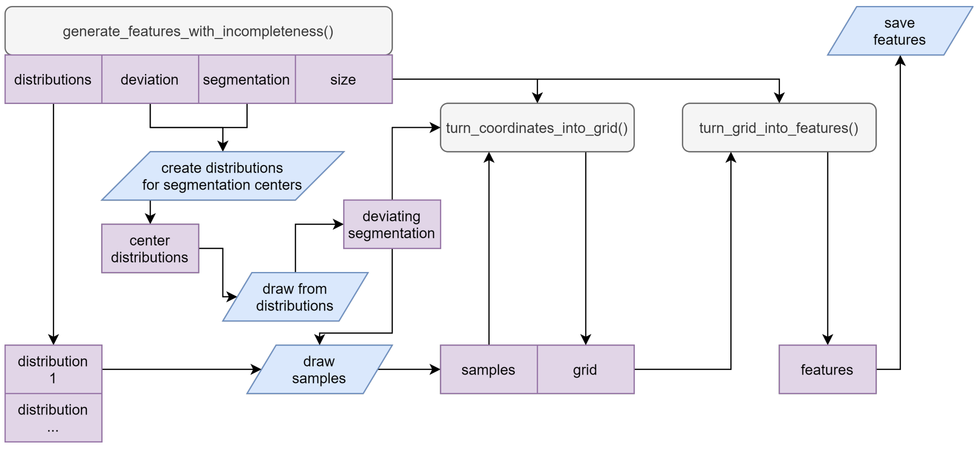


Figure 30: Features with Deviation Flow

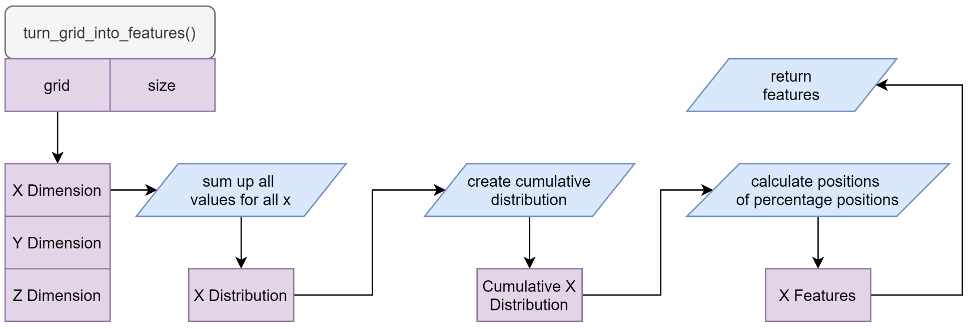

turn_grid_into_features()
grid
size
return features
X Dimension
Y Dimension
Z Dimension
sum up all values for all x
X Distribution
create cumulative distribution
Cumulative X Distribution
calculate positions of percentage positions
X Features

Figure 31: Feature Creation

5.2.5 Analysis

The general analysis is done through the function `features_training()`, which can be seen in Figure 32. This allows the use of different sets of classifiers and different training splits. The functions `features_training_duration()` and `train_and_save_model()` differ only very slightly. The former measure and prints the time every step takes for each classifier, while the latter actually saves the model, as well as the unused testing data. All functions can be found in "training.py".

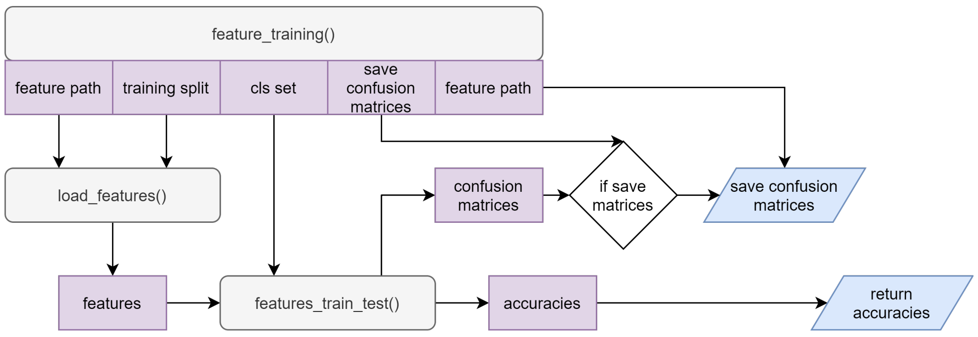


Figure 32: Training

5.2.6 Visualization

All visualizations use the Plotly Library [9]. All functions, which directly relate to the sampling, feature generation or analysis can be found in the "visualize.py" file. In some cases, they are implemented to work with already existing data from e.g. segmentation generation, in other cases it is necessary to create a CSV file first by using the `convert_results_to_csv()` function in the "functions.py" file, which can be then given to the visualization function. Additional visualizations, e.g. for visualizing how incompleteness is treated, can be found in "support_visualize.py".

5.3 Mock Setup

To demonstrate the communication system, I created a mock setup, which runs on a single machine. It consists of a server program, which would normally run in this or a similar form on a dedicated server. Moreover, a single Mosquitto broker is used, as opposed to the two brokers suggested earlier in 5.1, since no remote communication takes place. Lastly, a robot program simulates the robot. The code for this setup can be found in the "mock_setup" directory. To run it, first mostquitto has to be installed. A model trained on the "Combined Classes" distribution set is provided, but can be replaced by using the function `train_and_save_model()` in "training.py".

The operation of the mock setup is shown in Figure 33. Once the broker is started and "robot.py" and "server.py" are running, robot and server are in state 0, which represents the idle mode. The process can then be started by giving "yes" to the input in the "server.py" terminal. During step 1, the server sends a message to the robot through the broker over the topic "state", requesting it to switch to state 1, and then switching to state 1 itself. Now in step two, the robot is in send mode, sending out data packages, which represent measurements the robot would usually take, over the topic "data" to the server, which is in listening mode and saves all the data. Once all data packages are send, in step three the robot sends a message over the topic "state" to the server, telling it that the transmission is finished, and switches to state 0 afterwards again. The server switches to state 2 and starts checking the data for completeness, treats any problems, builds the features and then runs a classification. Afterwards it announces the results and in step 4 switches back to state 0, from where a new iteration can be started.

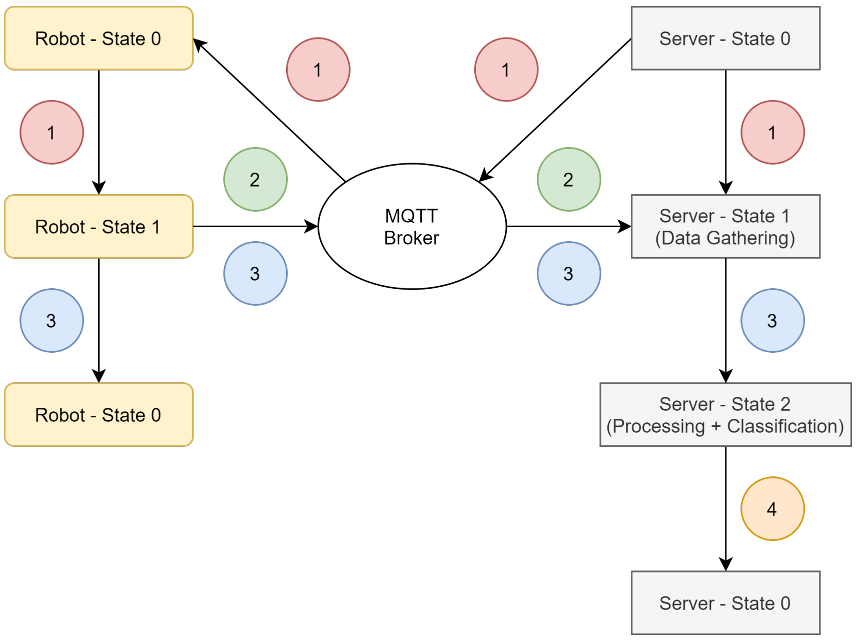


Figure 33: Mock Setup Operation

6 Conclusions and Future Work

6.1 Conclusions

This thesis successfully provided an approach for the analysis of spatial features, allowing it to classify different kinds of 3D distributions. I tested this approach extensively, examining different kinds of distributions, different segmentations and classifiers. By developing additional scenarios such as incomplete data and segmentations deviating from the perfect layout, I also made sure that the approach works under advanced conditions.

Through carefully designed distribution sets I was able to segregate the different properties "Position", "Orientation" and "Shape" which distributions can differ through. With a fourth class "Combined" I then brought all properties together. Besides that I also created multiple segmentation to split up the environment, and features which convert the data from 3D spatial data into a simpler representation.

My experiments showed that for the given data the "Nearest Neighbors" classifier outperformed all other classifiers, even for small amounts of data. Almost all the classifiers were able to train on the given data in less than a second, with the exception of the Neural Network, which took up to 5 seconds, and the Gaussian Process, which took up to 30 minutes. Besides that, all classifiers performed best on the "Position" distributions, followed by the "Combined" set, which also has strong position properties. While the other two classes usually performed less good, this is acceptable as the "Combined" is the closest to what can be expected in real-world cases. The introduction of deviation for segmentations showed that this has a strong influence on the results for distributions which do not differ strongly in position for certain segmentations, while for other segmentations and distributions there is almost no impact. Through simulating incomplete data I was also able to show that losing data packages has a significant influence on the accuracy.

As it was stated in the introduction, the system it also supposed to be used by others as well. To do that, I designed an easy-to-use API for Python, which allows a user to access all parts of the program, combine them as I did, or adapt them in any way necessary, for which I showed some options as well. This includes the data generation, feature generation, training, testing and visualization. That way anyone can for example repeat my experiments with new data, train new classifiers on the same data, switch to other features and so forth.

For the robotics part a guide to select, adapt or develop a robot is presented. For computation a Raspberry Pi computer is recommended, as it is powerful enough for everything the robot has to do computation wise, can use a standard OS and has great options for periphery devices. For the propulsion of the robot I suggest a very simple configuration with 2 vertical thrusters and 1 or 2 vertical thrusters. Regarding communication a stop at the surface to transmit data via Wi-Fi is recommended. There are several options for localization, with acoustic and 433MHz waves being the most reasonable ones. For the 433MHz waves I also showed that using them in a small water body seems viable. For the sensing part I discussed different options, but concluded that this strongly depends on the use case. Path Planning is tightly connected to the positions of measurements. In a simple case with a cuboid environment and segmentation, this can be done manually, for more complex segmentations RL might be a good option. My suggestion for controlling the robot is ROS, as it is perfect in tying all necessary software components together. Finally, I also showcase an example on how to build a waterproof body through 3D printing and casting.

Moreover, a blueprint for a setup to connect robotics and analysis is provided as well, together with a basic example of how it will work. It includes instructions on how to install the necessary software, as well as how to configure it.

6.2 Future Work

The current analysis only focuses on stationary data, but of course water is a very dynamic medium, so implementing time-series analysis on the spatial data taken during consecutive iterations will allow to understand the dynamics of the system, rather than only its state. While I initially dismissed using complex models such as 3D CNNs or Graph Neural Network (GNN)s as too computationally expensive, when moving forward they might be a good idea to reconsider, as they offer a few advantages. For once, they don't require the measurements to be transformed into features, which, even with good features, always is accompanied by a loss of data. Besides that currently it is problem to create features from data which is not ordered as a grid, which would especially be an argument for Graph Neural Networks. Not having to rely on data in grid form would also mean that the system can be applied to more than cuboid environments.

Given that the approach was so far tested only with synthetic data based on general assumptions, it will interesting to develop more realistic models of contamination or use actual measurements of contamination. Moreover, the model of the environment is overly simplistic right now and only allows a cuboid segmentation. Further developing this into a more general approach would be an important step towards using real data.

Besides that, I would like to implement the system with a robot to see how well this systems works in a real-world application. In 3.1 I postulated that in a real system a too high segmentation will eventually lead to a drop in accuracy through length of path, deviation from path and disturbance of the environment. This could be tested in a real system. It would additionally also grant to possibility to integrate the search for the perfect resolution and path.

7 Acknowledgments

I would like to thank the University of Bristol (UoB) and the Bristol Robotics Laboratory (BRL) for hosting me during a portion of my thesis, which allowed me to work towards the robotics part presented through access to various facilities and workshops.

My special thanks go to Dr. Hemma Philamore, who made it possible for me to visit the UoB and BRL and supported me tremendously throughout the entire thesis, through insight and guidance.

Besides that, I would also like to thank the ERASMUS+ program, which financially supported my stay in Bristol.

A Experiments

A.1 Base Case

Distribution Set	Classifier	Segmentations							
		1x1x1	2x2x2	3x3x3	4x4x4	5x5x5	10x10x10	15x15x15	20x20x20
Shape	Nearest Neighbors	0.8400	0.3400	1.0000	0.7400	1.0000	1.0000	1.0000	1.0000
	Linear SVM	0.7267	0.3333	0.3333	0.3333	0.4733	0.8200	1.0000	0.9467
	RBF SVM	0.7267	0.3333	0.4600	0.4067	0.6667	0.8600	1.0000	1.0000
	Gaussian Process	0.8133	0.3333	0.5133	0.3333	1.0000	1.0000	1.0000	1.0000
	Decision Tree	0.8200	0.3333	0.3333	0.4000	1.0000	1.0000	1.0000	1.0000
	Random Forest	0.8533	0.3333	0.3333	0.4000	1.0000	1.0000	1.0000	1.0000
	Neural Net	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	1.0000	1.0000
	AdaBoost	0.5800	0.3333	0.3333	0.4000	1.0000	1.0000	1.0000	1.0000
	Naive Bayes	0.8533	0.3467	0.7467	0.6667	1.0000	1.0000	1.0000	1.0000
	QDA	0.5467	0.3333	0.3333	0.3333	0.6867	0.6067	0.6333	0.6333
Combined	Nearest Neighbors	0.4025	0.8725	0.9925	1.0000	1.0000	1.0000	1.0000	1.0000
	Linear SVM	0.1650	0.5775	0.8975	1.0000	0.9825	1.0000	1.0000	1.0000
	RBF SVM	0.1650	0.7150	0.9200	1.0000	0.9925	1.0000	1.0000	1.0000
	Gaussian Process	0.2500	0.7225	0.4975	1.0000	1.0000	1.0000	1.0000	1.0000
	Decision Tree	0.2500	0.7675	0.9625	1.0000	1.0000	1.0000	1.0000	1.0000
	Random Forest	0.2500	0.7350	0.9825	1.0000	1.0000	1.0000	1.0000	1.0000
	Neural Net	0.1250	0.6400	0.9100	1.0000	0.9650	1.0000	1.0000	1.0000
	AdaBoost	0.2500	0.2950	0.2500	0.6250	0.6250	0.7500	0.6250	0.7500
	Naive Bayes	0.2500	0.6950	0.9600	0.9925	0.9875	1.0000	1.0000	1.0000
	QDA	0.1250	0.1250	0.1250	0.1250	0.1250	0.9975	1.0000	1.0000
Orientation	Nearest Neighbors	0.3133	0.3600	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Linear SVM	0.3800	0.3333	0.3333	0.3333	1.0000	1.0000	1.0000	1.0000
	RBF SVM	0.3800	0.3333	0.8733	0.8400	1.0000	1.0000	1.0000	1.0000
	Gaussian Process	0.3467	0.3333	0.8733	0.3333	1.0000	1.0000	1.0000	1.0000
	Decision Tree	0.3333	0.3333	0.3333	0.3733	1.0000	1.0000	1.0000	1.0000
	Random Forest	0.3800	0.3333	0.3333	0.3733	1.0000	1.0000	1.0000	1.0000
	Neural Net	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	1.0000	1.0000
	AdaBoost	0.3533	0.3333	0.3333	0.3733	1.0000	1.0000	1.0000	1.0000
	Naive Bayes	0.3200	0.3600	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	QDA	0.3333	0.3333	0.3333	0.3333	1.0000	0.5667	1.0000	1.0000
Position	Nearest Neighbors	0.1500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Linear SVM	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	RBF SVM	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Gaussian Process	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Decision Tree	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Random Forest	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Neural Net	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AdaBoost	0.1250	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Naive Bayes	0.1175	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	QDA	0.1250	0.1250	0.1250	0.1250	0.9975	1.0000	1.0000	1.0000

Table 6: Accuracies Segmentation Set "Basic"

Distribution Set	Classifier	Segmentations							
		5x3x3	3x5x3	5x5x3	10x3x3	3x10x3	10x10x3	20x3x3	3x20x3
Shape	Nearest Neighbors	1.0000	0.7400	1.0000	1.0000	0.5067	1.0000	1.0000	0.3600
	Linear SVM	0.4333	0.3333	0.4333	0.7333	0.3333	0.7400	0.9400	0.3267
	RBF SVM	0.5733	0.4200	0.5733	0.8000	0.3067	0.8000	1.0000	0.3267
	Gaussian Process	1.0000	0.4333	1.0000	1.0000	0.3067	1.0000	1.0000	0.3667
	Decision Tree	1.0000	0.3333	1.0000	1.0000	0.3533	1.0000	1.0000	0.2467
	Random Forest	1.0000	0.3333	1.0000	1.0000	0.3400	1.0000	1.0000	0.3133
	Neural Net	0.3333	0.3333	0.3333	0.5067	0.3333	0.3333	1.0000	0.3333
	AdaBoost	1.0000	0.3333	1.0000	1.0000	0.3667	1.0000	1.0000	0.3267
	Naive Bayes	1.0000	0.6467	1.0000	1.0000	0.6533	1.0000	1.0000	0.3667
	QDA	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333
Combined	Nearest Neighbors	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Linear SVM	0.9825	0.9950	0.9725	1.0000	1.0000	1.0000	1.0000	1.0000
	RBF SVM	0.9875	0.9950	0.9850	1.0000	1.0000	1.0000	1.0000	1.0000
	Gaussian Process	0.9350	0.9975	0.9725	1.0000	1.0000	1.0000	1.0000	1.0000
	Decision Tree	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Random Forest	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Neural Net	0.9875	0.9950	0.9850	1.0000	1.0000	1.0000	1.0000	1.0000
	AdaBoost	0.6250	0.5000	0.6250	0.7500	0.5000	0.7500	0.7500	0.5000
	Naive Bayes	0.9925	1.0000	0.9975	0.9875	0.9925	0.9975	0.9900	0.9925
	QDA	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250
Orientation	Nearest Neighbors	1.0000	1.0000	1.0000	0.8800	0.8733	1.0000	0.6333	0.6467
	Linear SVM	0.4533	0.5333	0.6267	0.6200	0.6133	0.9000	0.6600	0.6800
	RBF SVM	0.7333	0.8200	0.6800	0.6733	0.6533	1.0000	0.6600	0.6733
	Gaussian Process	0.9933	1.0000	1.0000	0.9733	0.9933	1.0000	0.6733	0.6867
	Decision Tree	0.6667	0.6667	1.0000	0.6333	0.6467	1.0000	0.6600	0.6467
	Random Forest	0.6667	0.6667	1.0000	0.6667	0.6667	1.0000	0.6667	0.6800
	Neural Net	0.3333	0.3333	0.3333	0.3533	0.3333	0.3333	0.6667	0.6667
	AdaBoost	0.6667	0.6667	1.0000	0.6667	0.6667	1.0000	0.6667	0.6667
	Naive Bayes	0.9933	1.0000	1.0000	0.8133	0.6933	1.0000	0.6667	0.6733
	QDA	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333	0.3333
Position	Nearest Neighbors	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Linear SVM	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	RBF SVM	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Gaussian Process	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Decision Tree	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Random Forest	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Neural Net	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	AdaBoost	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Naive Bayes	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	QDA	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250

Table 7: Accuracies Segmentation Set "Advanced"

A.2 Changing Amount of Training Data

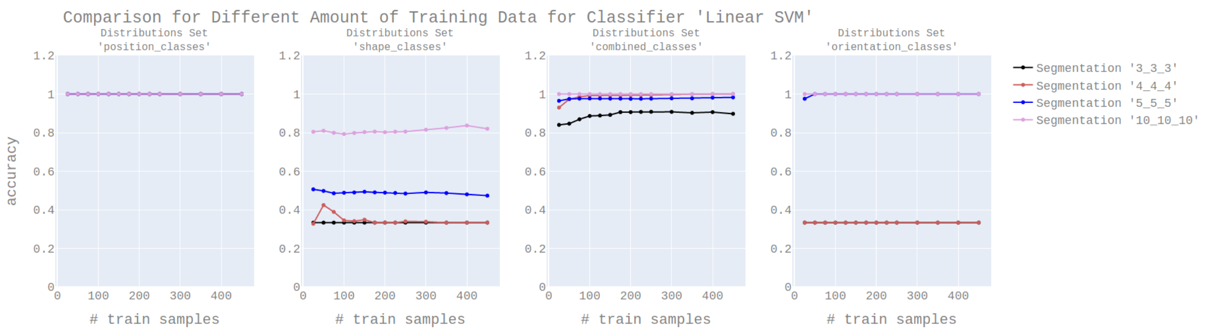


Figure 34: Influence of Amount of Training Data for Linear SVM Classifier

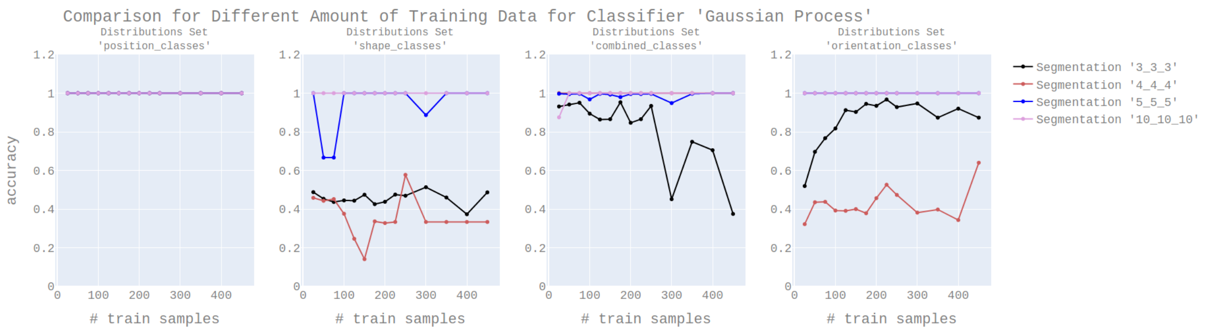


Figure 35: Influence of Amount of Training Data for Gaussian Process Classifier

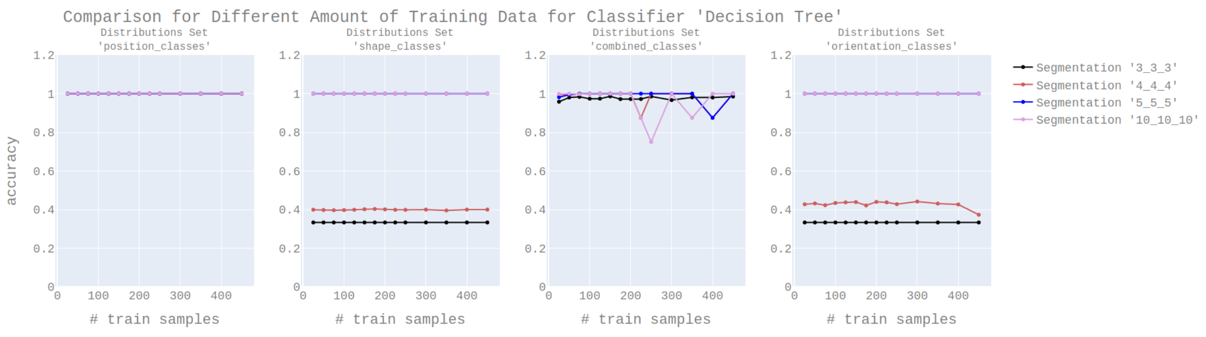


Figure 36: Influence of Amount of Training Data for Decision Tree Classifier

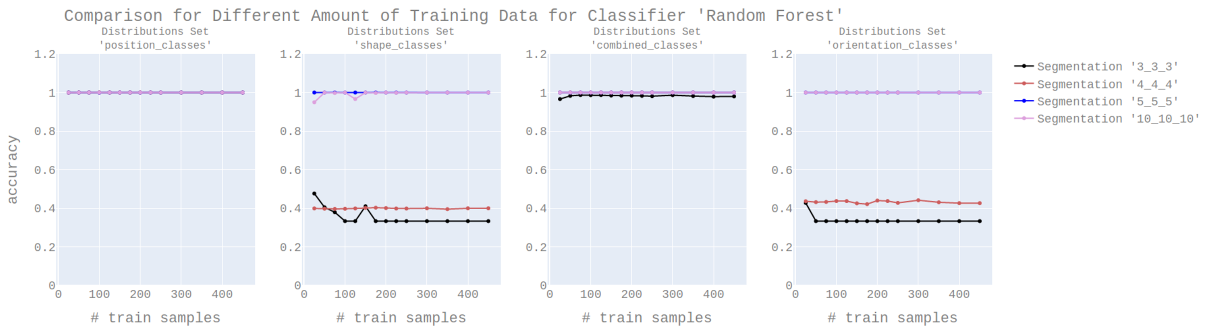


Figure 37: Influence of Amount of Training Data for Random Forest Classifier

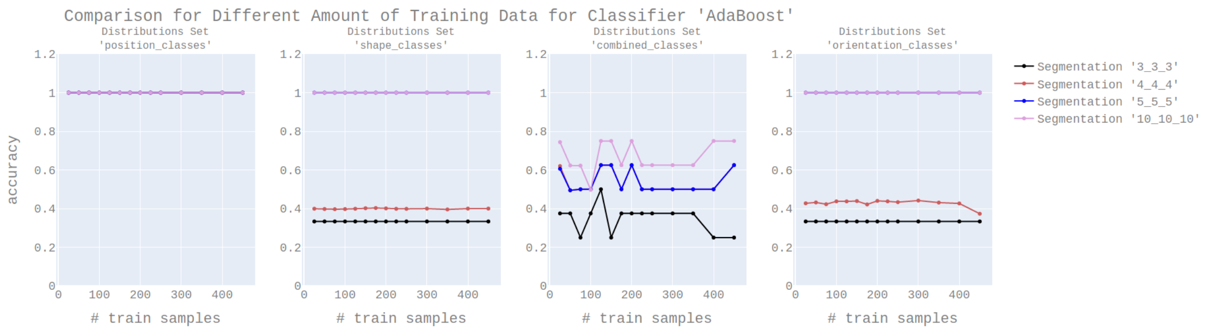


Figure 38: Influence of Amount of Training Data for AdaBoost Classifier

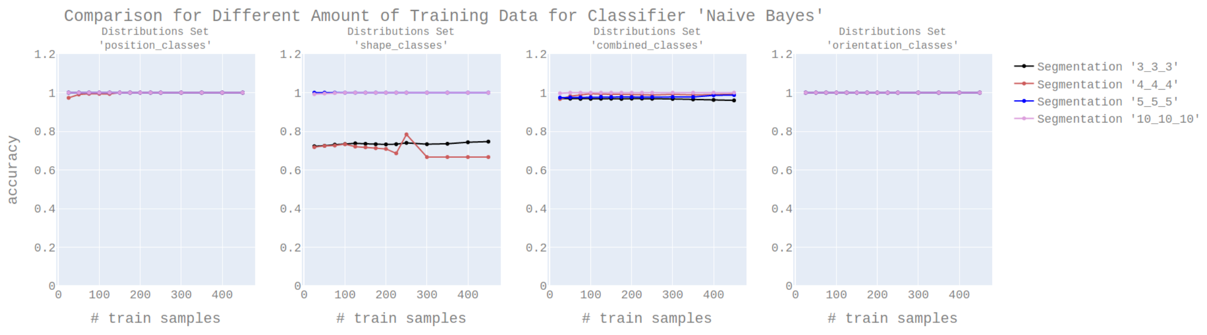


Figure 39: Influence of Amount of Training Data for Naive Bayes Classifier

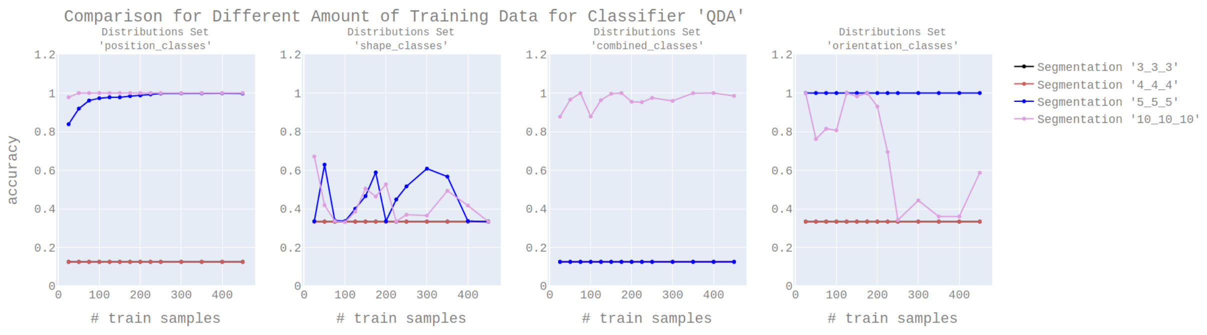


Figure 40: Influence of Amount of Training Data for QDA Classifier

A.3 Incomplete Data

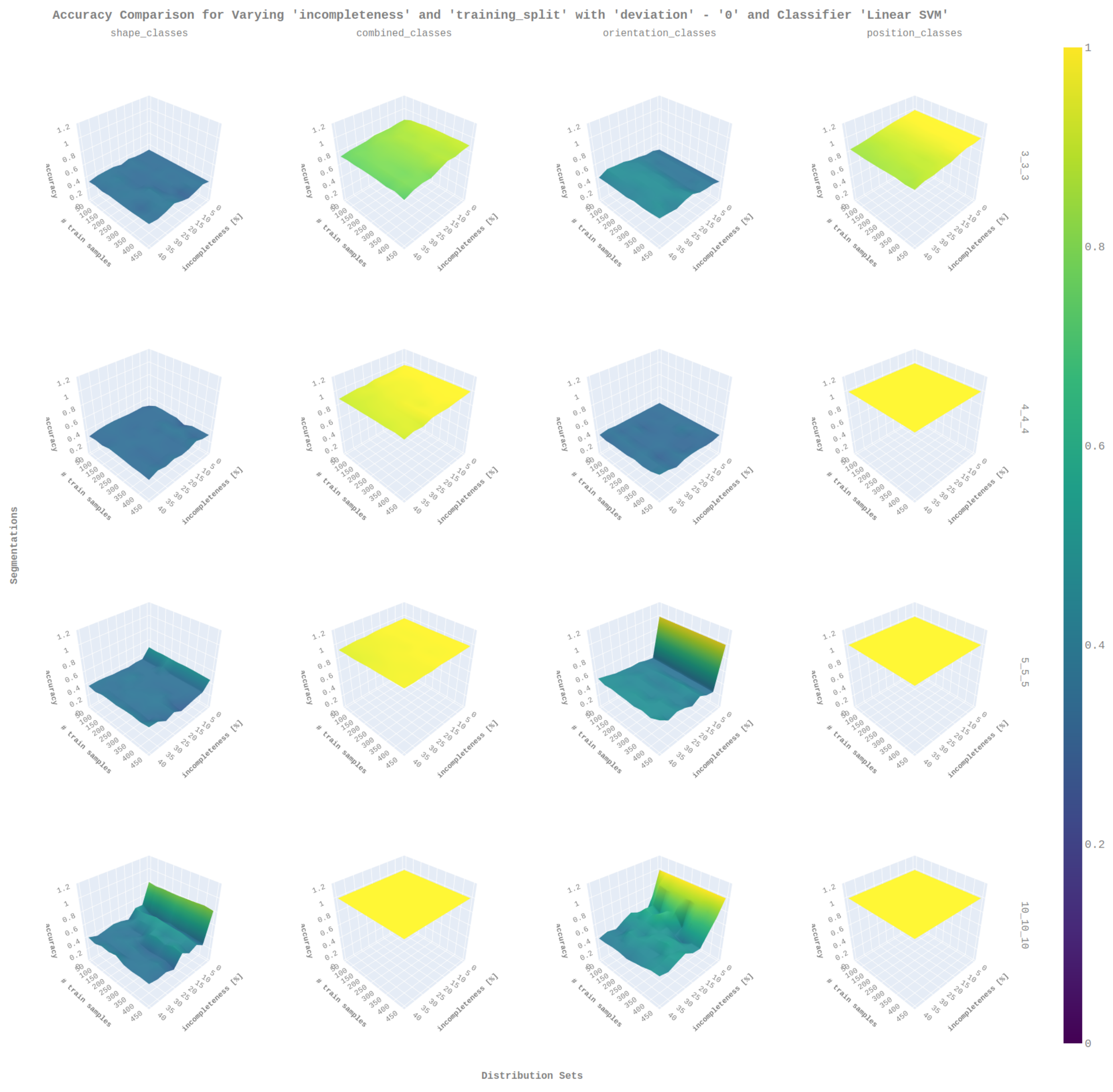


Figure 41: Influence of Incompleteness for Linear SVM Classifier

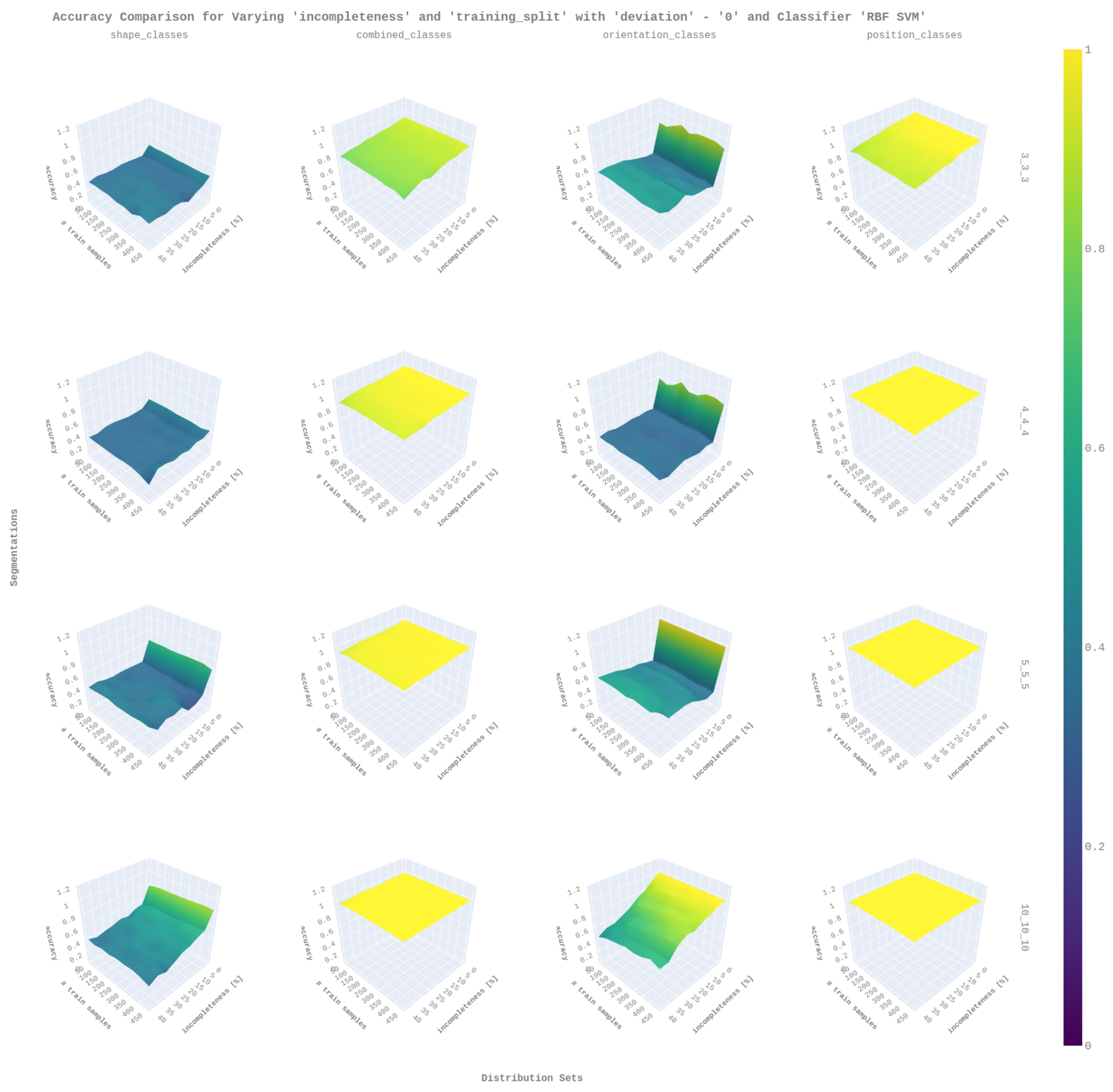


Figure 42: Influence of Incompleteness for Gaussian Process Classifier

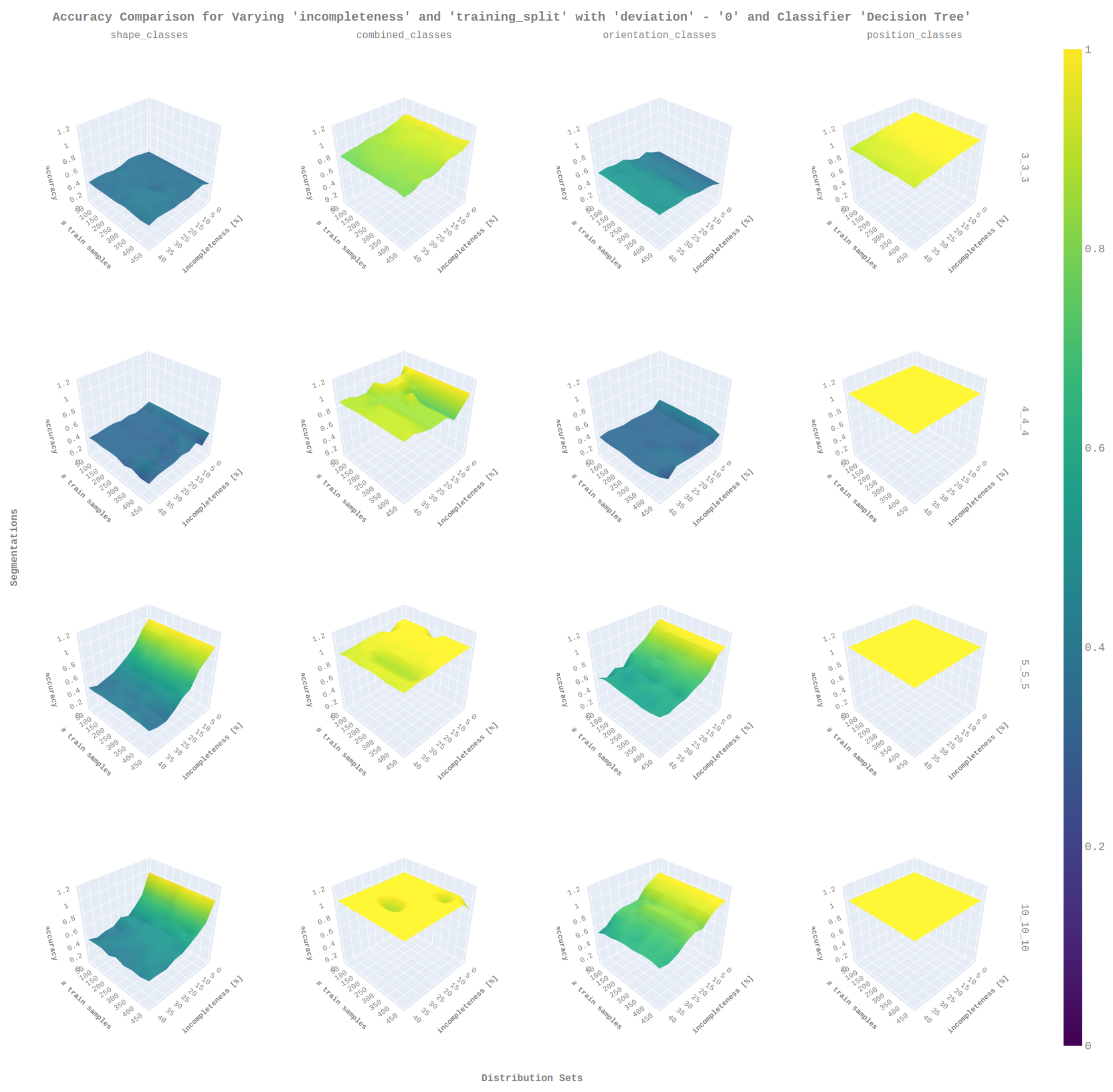


Figure 43: Influence of Incompleteness for Decision Tree Classifier

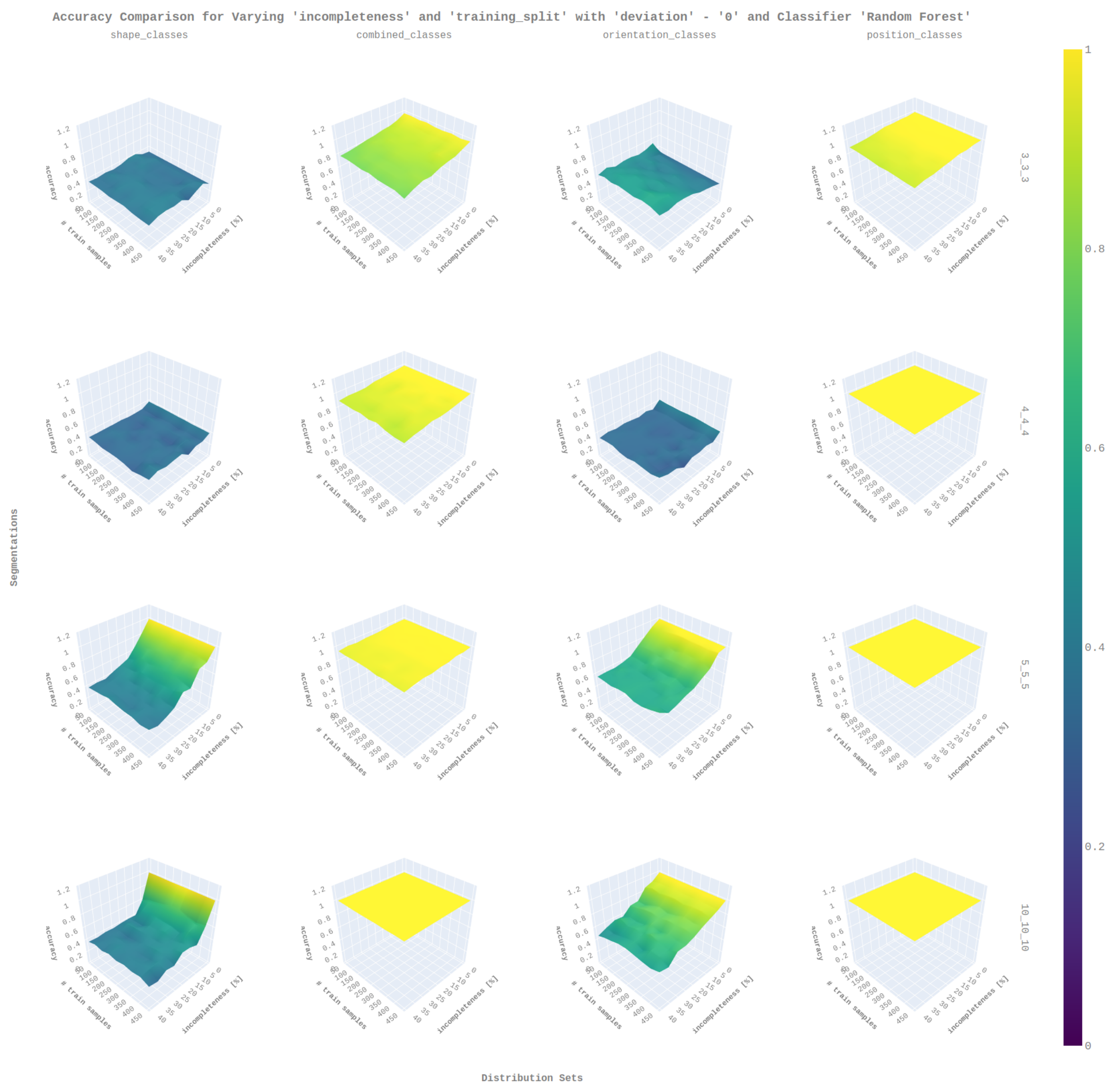


Figure 44: Influence of Incompleteness for Random Forest Classifier

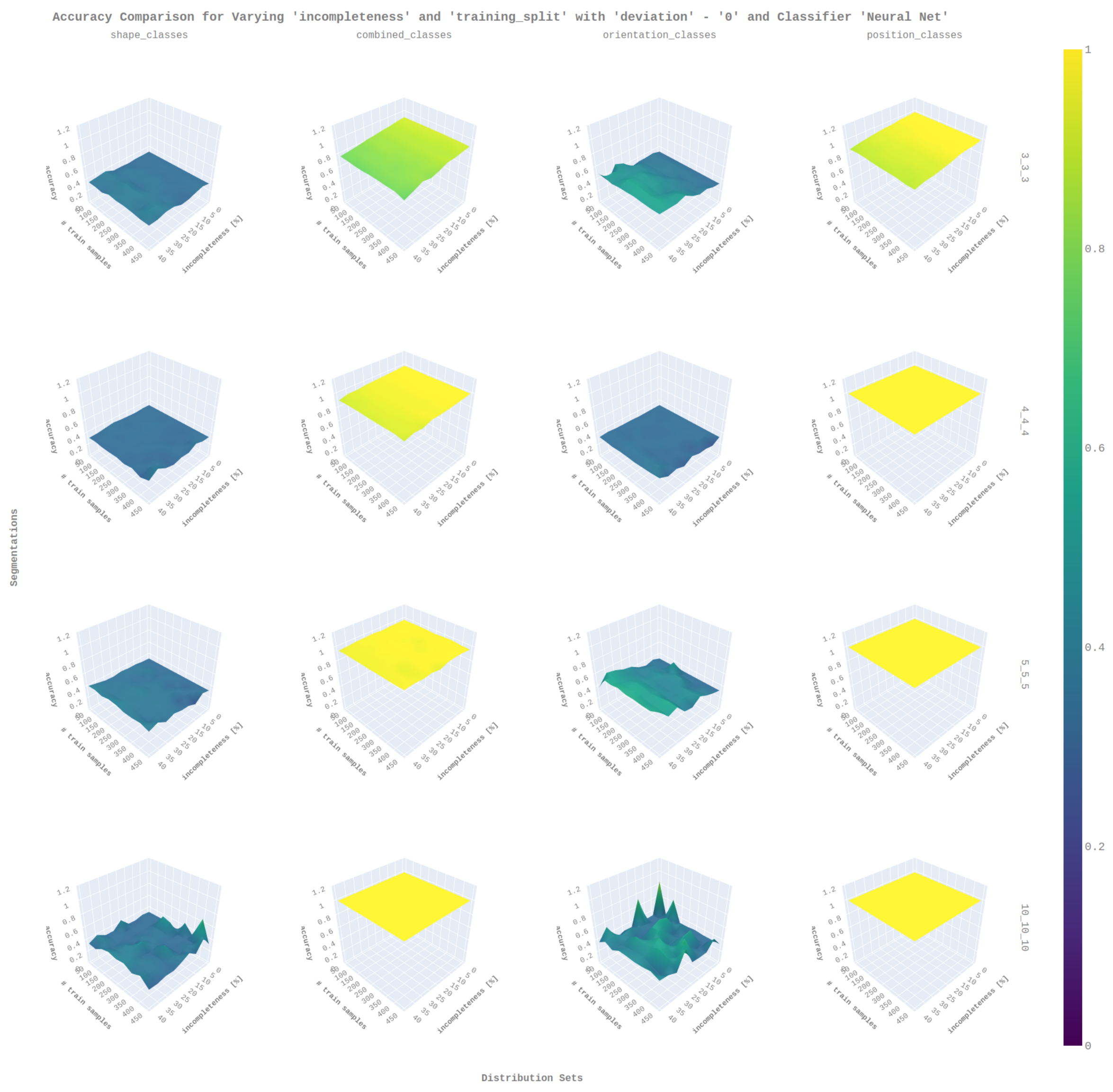


Figure 45: Influence of Incompleteness for Neural Net Classifier

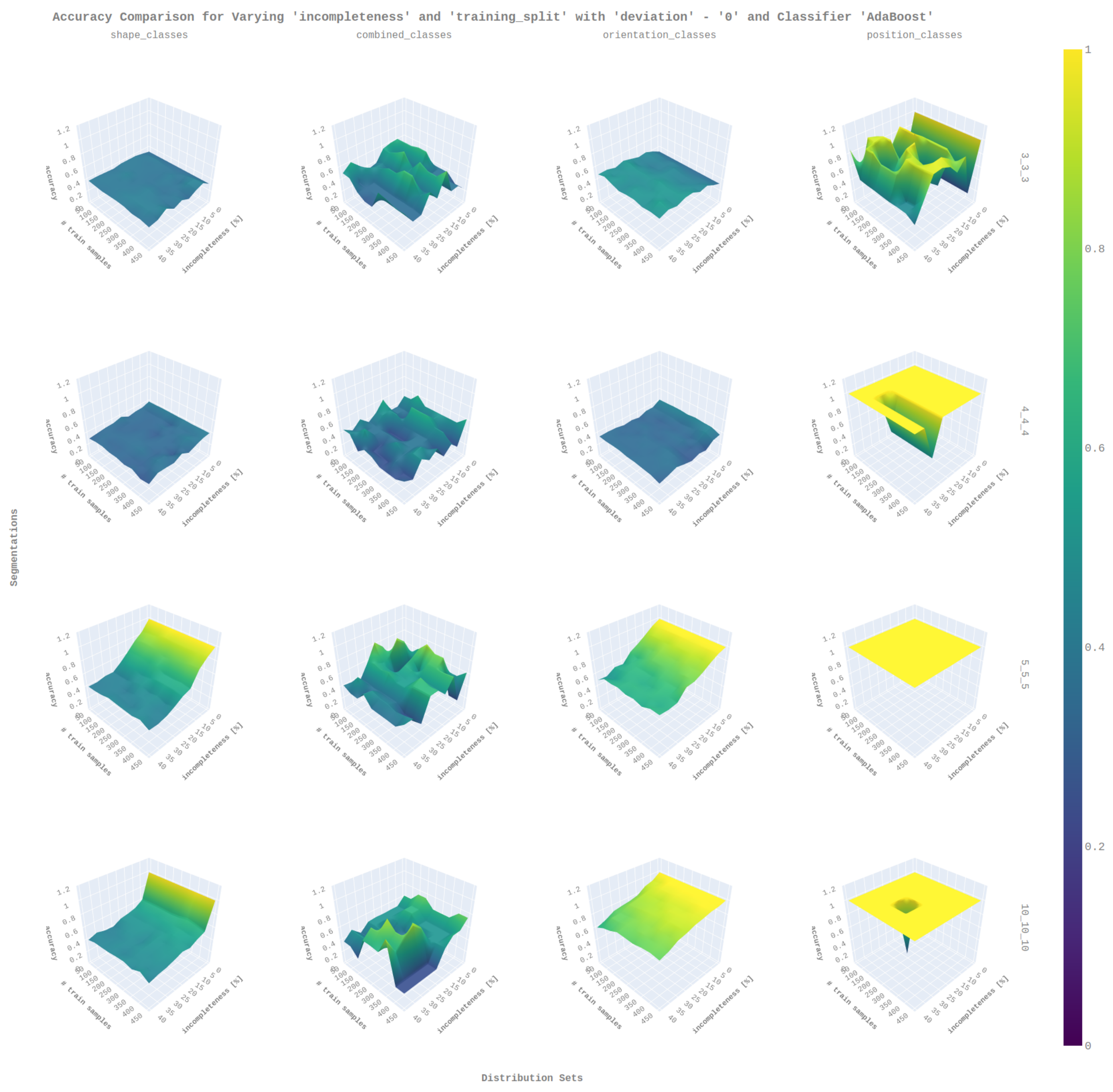


Figure 46: Influence of Incompleteness for AdaBoost Classifier

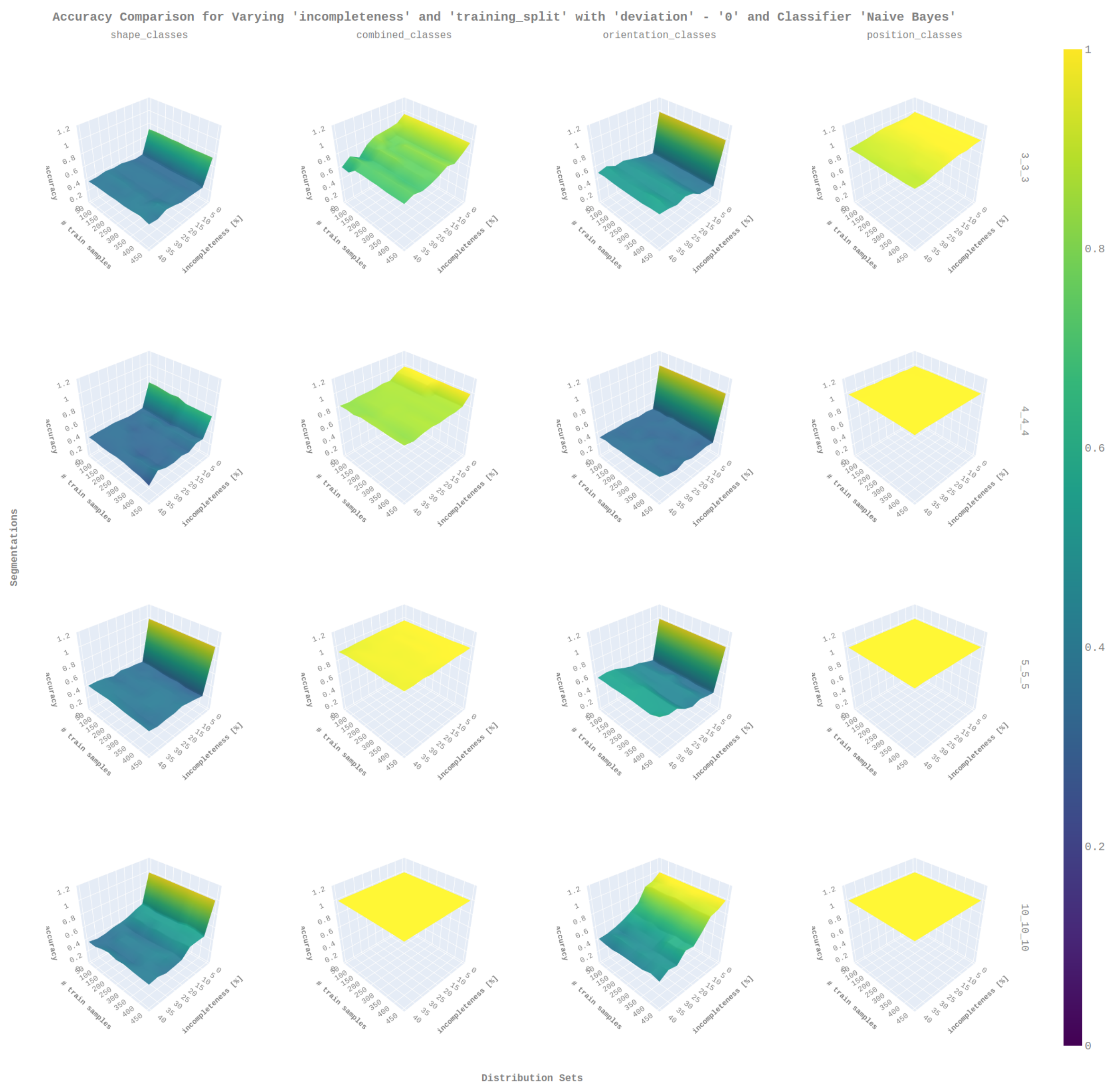


Figure 47: Influence of Incompleteness for Naive Bayes Classifier

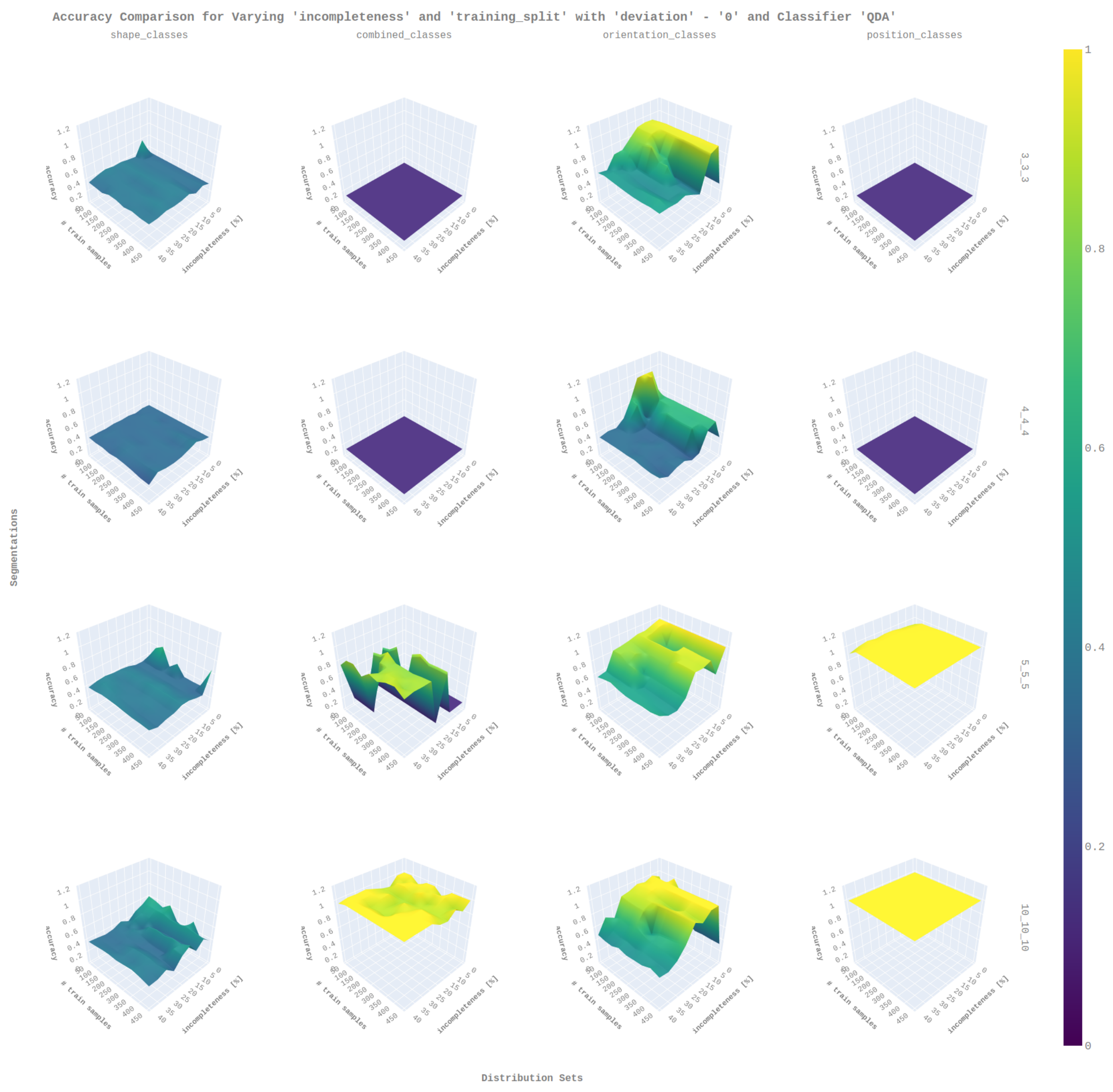


Figure 48: Influence of Incompleteness for QDA Classifier

A.4 Deviation in Segmentation

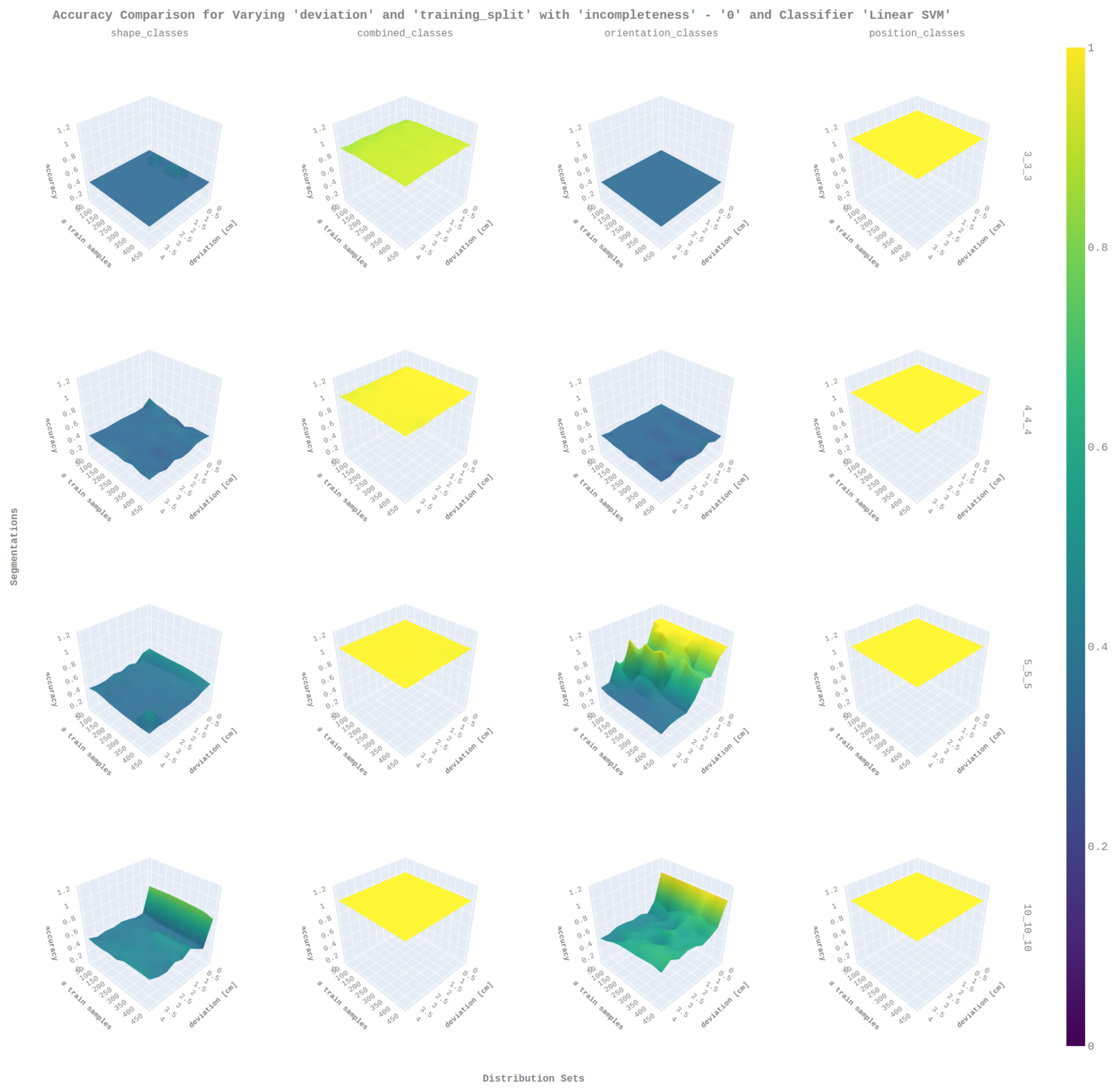


Figure 49: Influence of Deviation for Linear SVM Classifier

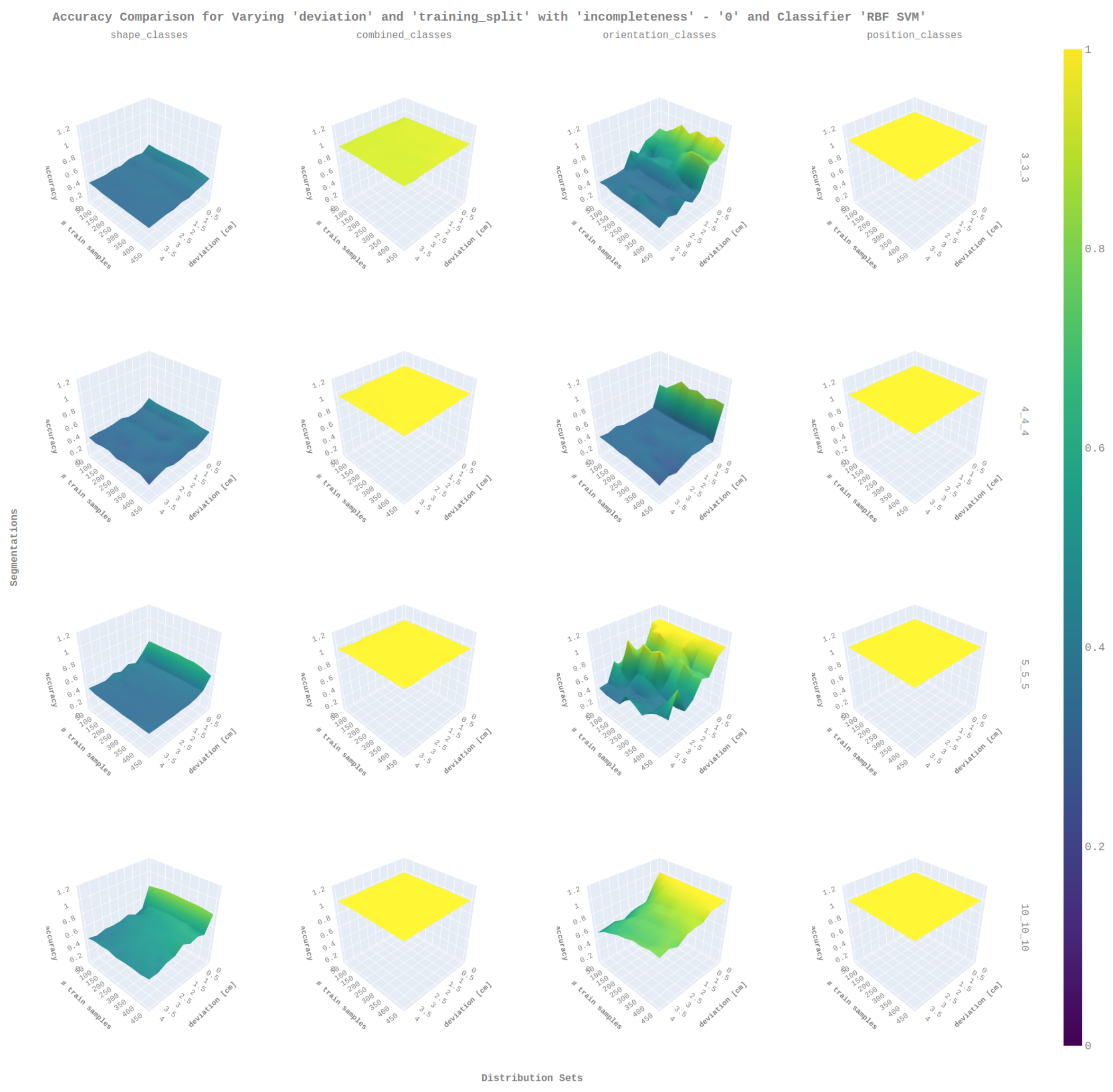


Figure 50: Influence of Deviation for Gaussian Process Classifier

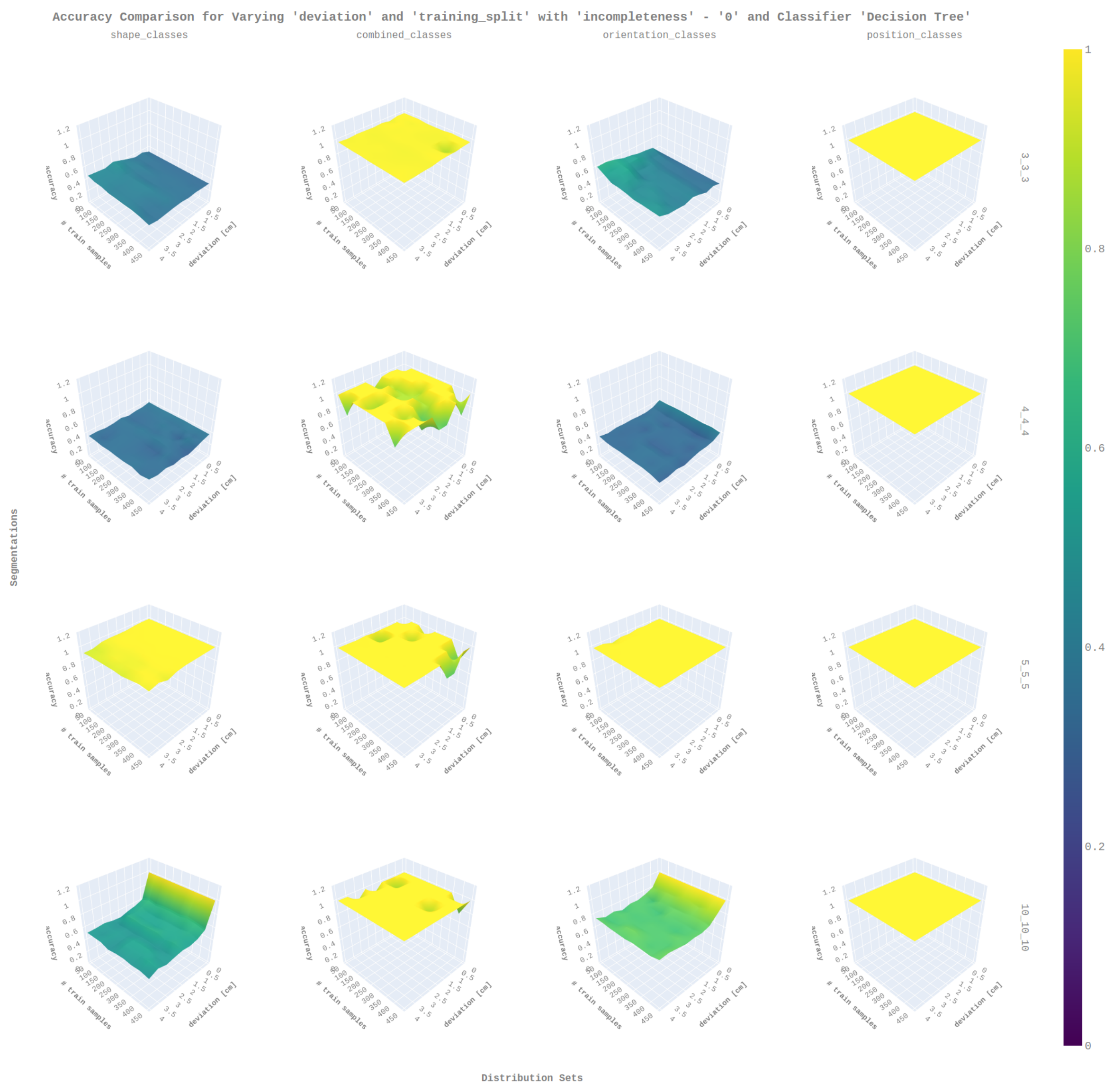


Figure 51: Influence of Deviation for Decision Tree Classifier

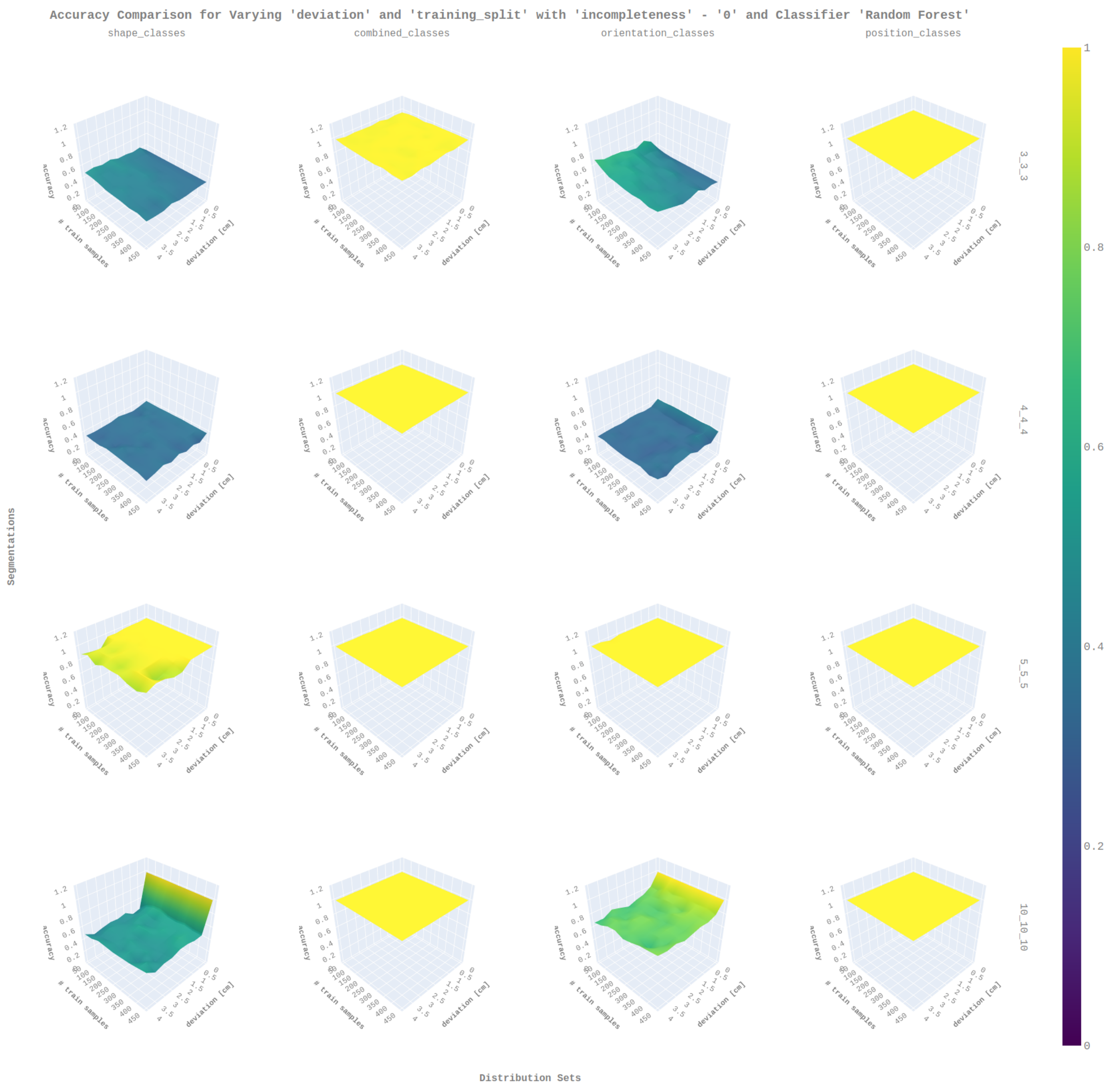


Figure 52: Influence of Deviation for Random Forest Classifier

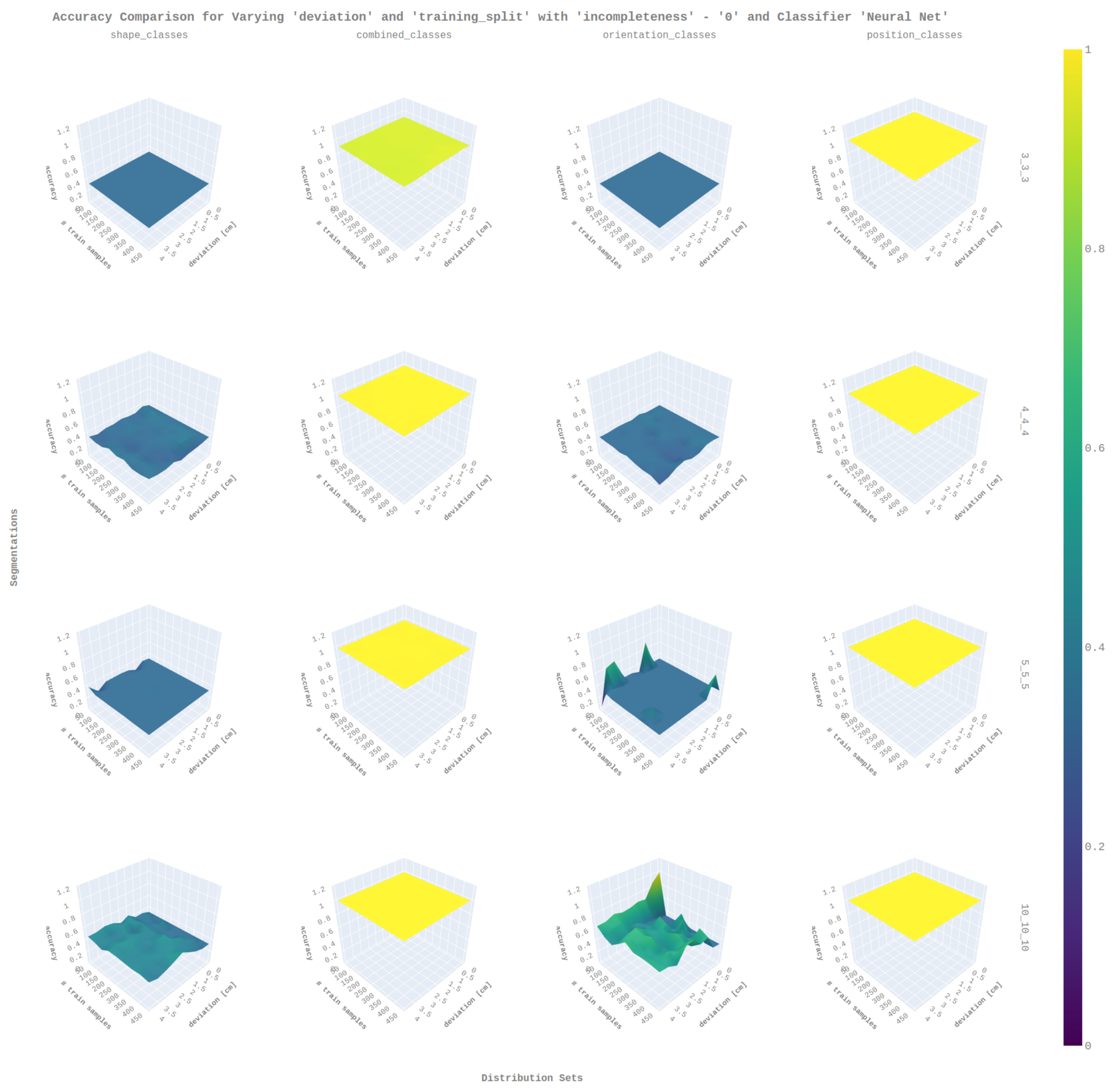


Figure 53: Influence of Deviation for Neural Net Classifier

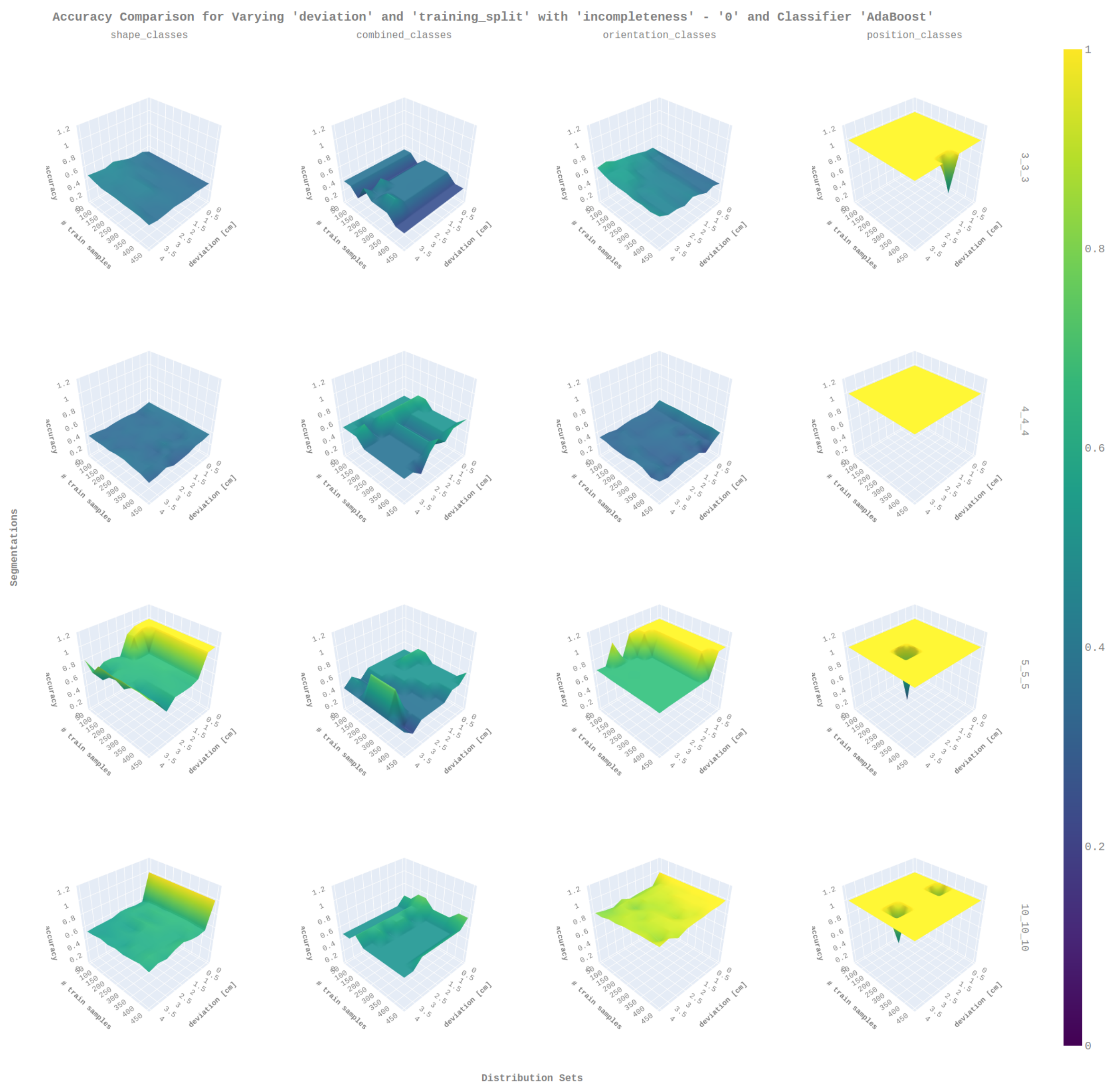


Figure 54: Influence of Deviation for AdaBoost Classifier

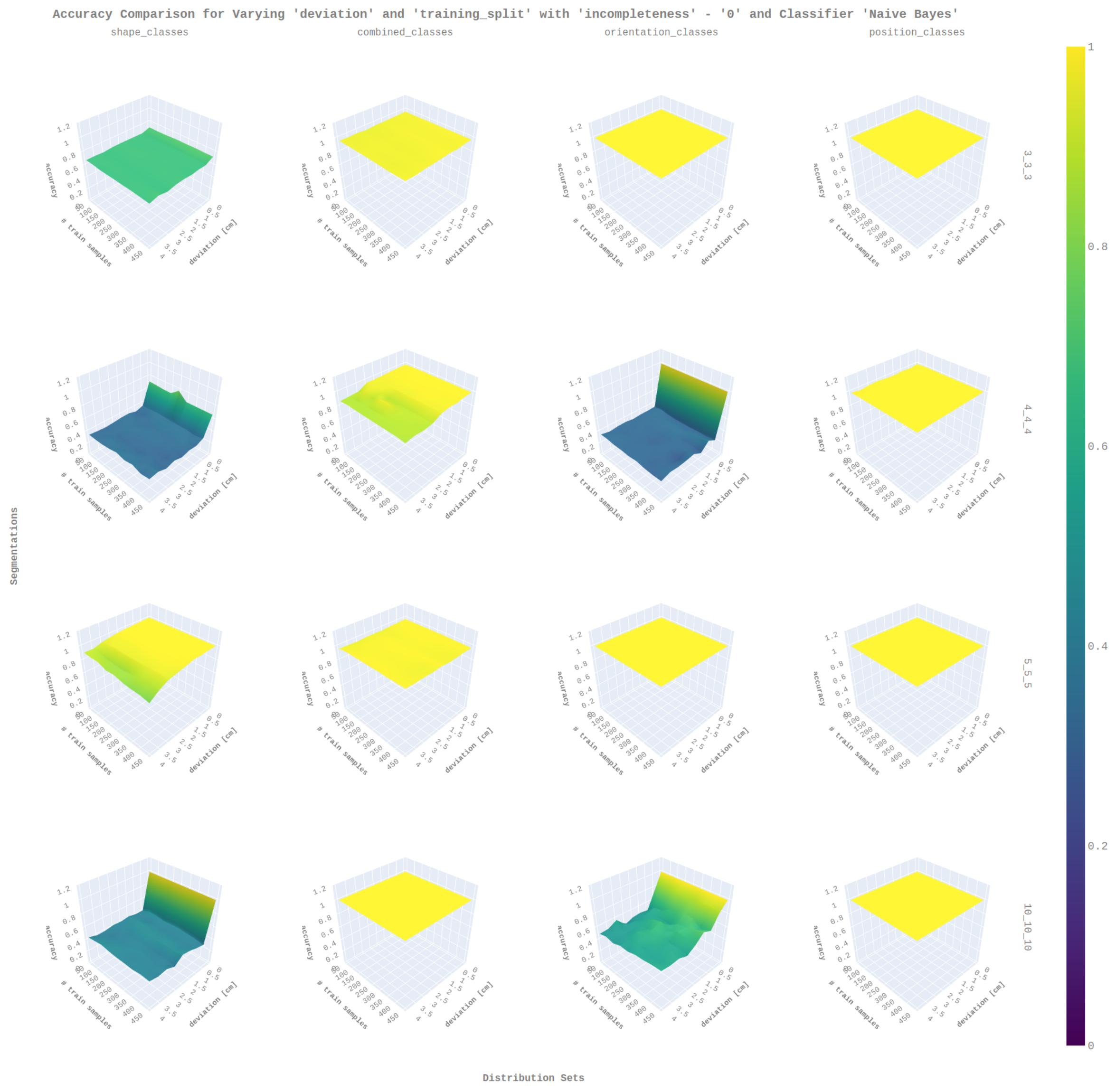


Figure 55: Influence of Deviation for Naive Bayes Classifier

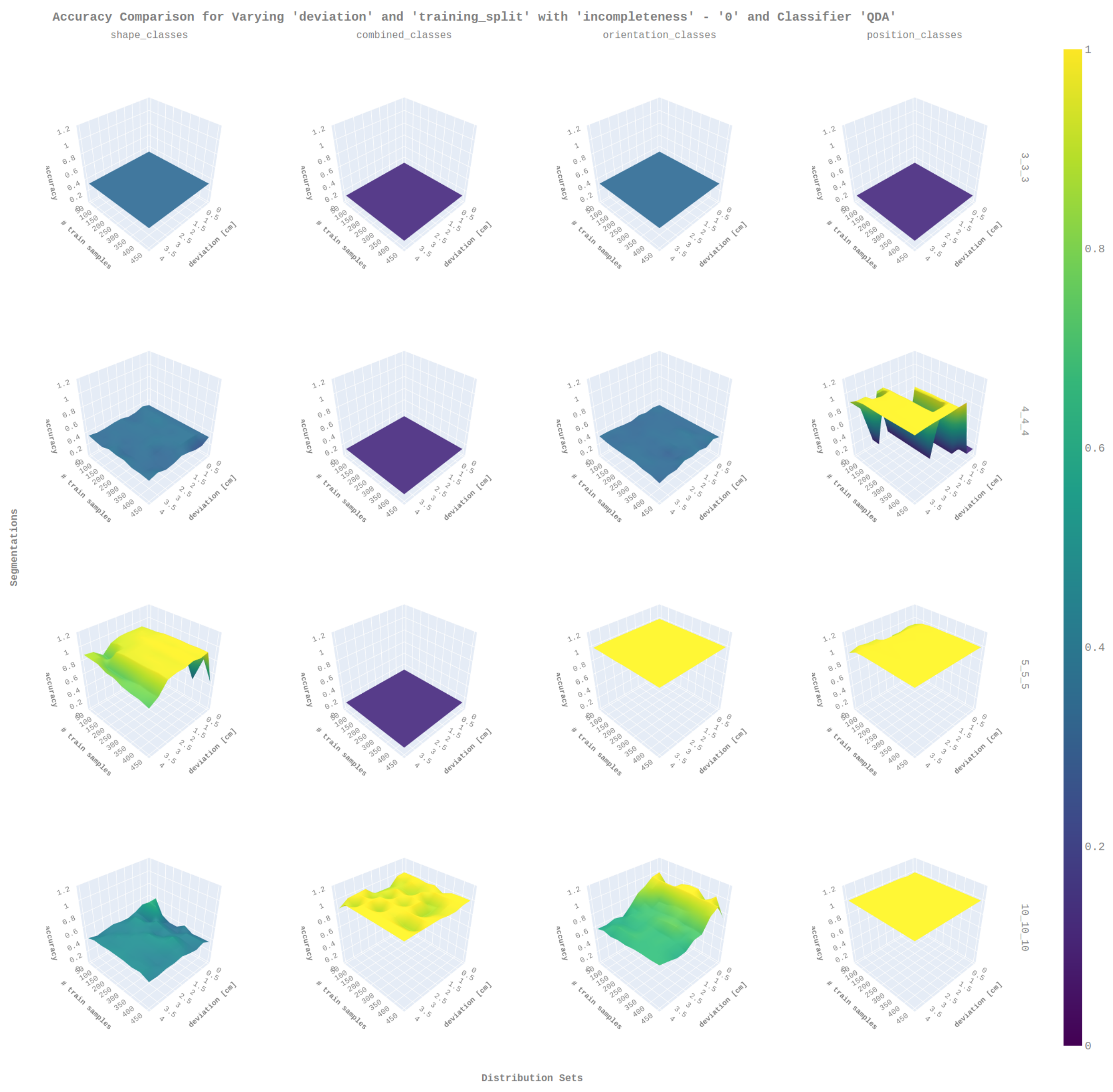


Figure 56: Influence of Deviation for QDA Classifier

References

[1] Ahmed, E., Saint, A., Shabayek, A. E. R., Cherenkova, K., Das, R., Gusev, G., Aouada, D., and Ottersten, B. E. Deep learning advances on different 3d data representations: A survey. *CoRR abs/1808.01462* (2018).

[2] Chen, Y., Wenjin, W., and Xu, G. System identification of auv hydrodynamic model based on support vector machine. pp. 1–7.

[3] Christ, R. D., and Sr], R. L. W. Chapter 3 - rov components. In *The ROV Manual*, R. D. Christ and R. L. W. Sr], Eds. Butterworth-Heinemann, Oxford, 2007, pp. 46 – 80.

[4] Dutta, A., Tamang, P., Dahal, P., Prajapati, R., and K.C., S. Iot based aquaponics monitoring system.

[5] Feng, Y., Barr, W., and Harper, W. Neural network processing of microbial fuel cell signals for the identification of chemicals present in water. *Journal of Environmental Management 120* (2013), 84 – 92.

[6] Gupta, A., and Ruebush, E. Aquasight: Automatic water impurity detection utilizing convolutional neural networks. *CoRR abs/1907.07573* (2019).

[7] Hafeez, S., Wong, M. S., Abbas, S., Kwok, C. Y. T., Nichol, J., Lee, K. H., Tang, D., and Pun, L. Detection and monitoring of marine pollution using remote sensing technologies. In *Monitoring of Marine Pollution*, H. B. Fouzia, Ed. IntechOpen, Rijeka, 2019, ch. 2.

[8] Hu, H., Oyekan, J., and Gu, D. *A School of Robotic Fish for Pollution Detection in Port.* 12 2011, pp. 85–104.

[9] Inc., P. T. Collaborative data science, 2015.

[10] Jena, A. K., Biswas, P., and Saha, H. Advanced farming systems in aquaculture: Strategies to enhance the production. *Innovative Farming, 2455-6521 2* (01 2017), 84–89.

[11] Li, Y., Yanqing, J., Wang, L.-F., Cao, J., and Zhang, G.-c. Intelligent pid guidance control for auv path tracking. *Journal of Central South University 22* (09 2015), 3440–3449.

[12] Li, Y., Zhang, H., and Shen, Q. Spectral–spatial classification of hyperspectral imagery with 3d convolutional neural network. *Remote Sensing 9*, 1 (Jan 2017), 67.

[13] Mohammed, H., Hameed, I., and Seidu, R. Machine learning: based detection of water contamination in water distribution systems. pp. 1664–1671.

[14] M.S, M., Ali, M., G, N., Ali, S., and M.Y, K. Measuring the underwater received power behavior for 433 mhz radio frequency based on different distance and depth for the development of an underwater wireless sensor network. *Bulletin of Electrical Engineering and Informatics 8* (09 2019).

[15] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research 12*, Oct (2011), 2825–2830.

[16] Petres, C., Pailhas, Y., Patron, P., Petillot, Y., Evans, J., and Lane, D. Path planning for autonomous underwater vehicles. *Robotics, IEEE Transactions on 23* (05 2007), 331 – 341.

[17] Qureshi, U. M., Shaikh, F. K., Aziz, Z., Shah, S. M. Z. S., Sheikh, A. A., Felemban, E. A., and Qaisar, S. B. Rf path and absorption loss estimation for underwater wireless sensor networks in different water environments. *Sensors (Basel, Switzerland) 16* (2016).

[18] Shafahi, M., and Woolston, D. Aquaponics: A sustainable food production system.

[19] Wang, C., Wei, L., Wang, Z., Song, M., and Mahmoudian, N. Reinforcement learning-based multi-auv adaptive trajectory planning for under-ice field estimation. *Sensors 18* (11 2018), 3859.

[20] Wolpert, D., and Macready, W. No free lunch theorems for search.

[21] Yujia, H., Li, Y., and Feng, X. Model-free recurrent reinforcement learning for auv horizontal control. *IOP Conference Series: Materials Science and Engineering 428* (10 2018), 012063.

Assertion

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

[illegible]

Karlsruhe, July 21, 2020 Hendrik Emil Eichhorn