Smart Labelling

BSc Thesis

Suzanne J. Spink

Supervisors:
Dr. J.W. Kamminga
Dr. A. Kamilaris

UNIVERSITY OF TWENTE
Netherlands, Enschede
July 2021

Abstract

The literature shows that active learning (AL) has great potential in the field of activity recognition. The AL algorithm is potentially much more cost-efficient and less time-consuming when labelling activities than with supervised learning. However, AL has only been applied to animal activity recognition (AAR) in a limited fashion. This thesis investigates the potential role active learning can play in the field of AAR, by finding the AL strategy which is the quickest in converging to the most adept performance for AAR. This is done by applying three uncertainty sampling algorithms and two disagreement based sampling algorithms to a DNN classifier, namely least certain, margin, uncertainty entropy, consensus entropy, and maximum disagreement.

Comparing these to each other showed favouritism towards using least confident or maximum disagreement for their respective divisions. Both showed a great advantage over random sampling and the least confident algorithm was quicker to reach its maximum potential than maximum disagreement. However, the differences were minor, where a bigger factor was the impact the initial training set sizes had and how many times the oracle was queried iteratively. For this data set, the optimal size was at 350 with an additional 18 iterations. This showed the great potential of AL over supervised learning, as this data set consisted of 81 332 points which were previously all manually annotated. Using AL, this would have saved a person more than a month in labelling time when assuming a full working week of 40 hours. However, the performance is lower, as AL had a MCC of 0.694 and manual annotation has a certainty of 94.3% when allowing for human error.

Contents

1 Introduction **5**

2 Background **6**
2.1 Querying Methods . . . 6
2.1.1 Pool-Based Sampling . . . 6
2.1.2 Stream-Based Sampling . . . 6
2.1.3 Membership Query Synthesis . . . 7
2.2 Algorithm Types . . . 7
2.2.1 Uncertainty Sampling . . . 7
2.2.2 Disagreement-based Sampling . . . 8
2.3 General AL Training . . . 8
2.3.1 Data Collection . . . 9
2.3.2 Training . . . 9
2.3.3 Evaluation . . . 10

3 State of the Art **11**
3.1 AL Algorithms' Assets and Liabilities . . . 11
3.1.1 Uncertainty Sampling . . . 11
3.1.2 Disagreement-Based Sampling . . . 12
3.2 Animal Activity Recognition . . . 12
3.2.1 AAR Algorithms . . . 12
3.3 Human Activity Recognition . . . 13
3.3.1 HAR Algorithms . . . 13
3.3.2 AL strategies in HAR . . . 14
3.4 Activity Recognition . . . 15
3.4.1 Classification Result Reliability . . . 16
3.4.2 Sliding Window . . . 16
3.4.3 Activity Classification Limitations . . . 17
3.4.4 Data Set Size Performance . . . 18
3.4.5 Stopping Point Maximisation . . . 18
3.5 Challenges . . . 19
3.5.1 AL Challenges . . . 19
3.5.2 AR challenges . . . 19

4 Approach **21**
4.1 Focus . . . 21
4.2 Design Decisions . . . 21
4.2.1 Baseline Variables . . . 21

5 Methodology **23**
5.1 Dataset . . . 23
5.2 AL Process . . . 25
5.2.1 Active Learning Variables . . . 25
5.3 Horse activity recognition pipeline . . . 26

5.3.1 Preprocessing data . . . 27
5.3.2 Classification . . . 29
5.3.3 Evaluation . . . 30
5.4 Tools . . . 30
5.4.1 ITC Geospatial Computing Portal . . . 30
5.4.2 Programming language and packages . . . 30
5.5 Database . . . 31

6 Results **33**
6.1 Linear SVM Classifier with AL . . . 33
6.2 DNN without AL . . . 35
6.3 AL on DNN . . . 36
6.3.1 AL Variables . . . 36
6.3.2 Least Confident . . . 37
6.3.3 Margin of Confident . . . 39
6.3.4 Uncertainty Entropy . . . 41
6.3.5 Consensus Entropy . . . 43
6.3.6 Maximum Disagreement . . . 45

7 Evaluation **48**
7.1 Comparing Algorithms Per Sampling Type . . . 48
7.2 Comparing Sampling Types . . . 52
7.3 Comparing To DNN Without AL . . . 53
7.4 Manual Annotation vs Active Learning . . . 56

8 Discussion **58**
8.1 Comparing AL Variables . . . 58
8.2 Algorithm Analysis . . . 58
8.2.1 Uncertainty Analysis . . . 59
8.2.2 Disagreement Analysis . . . 59
8.2.3 Uncertainty vs Disagreement Analysis . . . 60
8.3 DNN without AL vs DNN with AL analysis . . . 60
8.4 Manual Annotation Results . . . 60

9 Conclusion **62**
9.1 Recommendations . . . 63
9.2 Future Work . . . 63

10 References **65**

11 Appendixes **69**
11.1 Appendix A: SVM classifier vs AL algorithm, 2 iterations . . . 69
11.2 Appendix B: SVM classifier vs AL algorithm, 10 iterations . . . 71
11.3 Appendix C: Code For Database . . . 76
11.4 Appendix D: Code comparing Uncertainty AL algorithms . . . 79
11.5 Appendix E: Initial Training Set Size Comparing Uncertainty AL algorithms . . . 93

11.6 Appendix F: Initial Training Set Sizes Comparing Disagreement AL algorithms . . . 98

1 Introduction

Currently, there is a lot of data available, more than can be considered manually. The process of identifying, classifying and labelling this data is tedious, expensive and labour intensive. This process can be automated using Machinel Learning. Machine learning (ML) has three branches. Firstly, there is supervised learning, where labelled instances are used to train a classifier. Secondly, there is unsupervised learning, where an algorithm with parameters finds correlations by itself without labelled instances. Lastly, there is semi-supervised learning, which uses a combination of a big amount of unlabelled data and a small amount of labelled data used to train a classifier. One interesting technique in semi-supervised learning is Active Learning. This technique only queries the data it is uncertain about, instead of randomly selecting a few data points, which thereby optimising the process. The querying happens by asking a human oracle for the correct label and adding this to the small training subset. As a result, classification can become much quicker and less costly.

This paper focuses on the impact Active Learning has in the field of animal activity recognition, where insights are given into animals' environment, health and well-being. Researchers collect lots of data by tracking animals, whose behaviour can be visualised and changes can be detected, subsequently potentially indicating other possible factors worth investigating. However, it would take a long time to manually go through this data. There are already patterns present in the data, showing correlation between the data and activities, but these are currently unidentified and just seen as a bunch of numbers. By using Active Learning, data can be more efficiently classified into activities and their behaviour can be analysed, without investing in too much time and money.

To reach this goal, Active Learning strategies are applied to an already existing IMU data set of horses. The aim will be to find the AL strategy with the best performance, by looking at a combination of the highest F1 score and MCC and the lowest number of labelled instances. Therefore, the main goal of this report is to give insight into: *Which Active Learning strategy is the quickest in converging to the most adept performance for Animal Activity Recognition when applied to an IMU horse data set?* This question is answered by focusing on two subcategories in Active Learning algorithms which were established by empirical research, namely uncertainty sampling and disagreement based sampling.

This paper consists of two parts. First empirical research is done. This focuses on existing Animal Activity Recognition and Human Activity Recognition methods, then a more general overview of potentially important factors in Activity Recognition and lastly, different AL algorithms are considered. Secondly, Active Learning strategies are tested on the horse data set. After testing different approaches with different parameters based on this research, all approaches can be evaluated based on its performance. Lastly, a conclusion is drawn on the optimal AL strategy for this data set.

2 Background

Active Learning is a niche field with many strategies and approaches which are not commonly or actively used in other methods in Machine Learning. Therefore, they are not well known. This section includes background information on the general structure of Active Learning, on approaches to querying data from a data set and some specific algorithms that have been used in AL.

2.1 Querying Methods

As mentioned, the AL strategy consists of two parts, the first being the querying method. This is the broader approach on how a data set will have to be used. This can be done by looking into how instances are selected from a data set, which are to be queried. These instances will later be decided on by an algorithm and used to train the classifier. There are three approaches that are mainly discussed in the literature, namely: Pool-Based Sampling, Stream-Based Sampling and Membership Query Synthesis.

2.1.1 Pool-Based Sampling

In Pool-Based sampling, the entire data pool is considered collectively to assess informativeness. This method looks at all unlabelled instances simultaneously and from there, an algorithm can be developed to decide which instances to query [1]. According to Settles [1], Wang [2] and Sabato [3], this is a good approach, as it can consider the full picture and can then make the most informed decision. However, this is only the case when the entire data set is already available, otherwise the data set will be biased and categories are misrepresented.

According to Kachites [4], while this method theoretically works well, there is one main disadvantage of using Pool-based sampling. This is, that while it may accurately choose instances that are best to train the algorithm most of the time, sometimes it can also choose those that are unimportant. To solve this, a new method was developed by Kachites in Pool-Based Sampling, called density-weighted Pool-Based sampling. This method looks at similarities in instances and focuses on those that have a high variance while having many similarities. This prevents querying negligible instances and improves efficiency [4]. Therefore, the Pool-Based sampling method is extremely useful in Active Learning, but will often have to be used in combination with the density-weighted method to counter imbalance.

2.1.2 Stream-Based Sampling

Stream-Based Sampling is used less often, but can still be useful in certain circumstances. As the name suggests, the instances are evaluated on informativeness individually in a stream. One at the time, they are immediately evaluated and labelled or discarded, based on the current data set available. As a consequence, every unlabelled instance is drawn [1]. This works well for data which comes in a live-stream, e.g. a spam filter or twitter feed [3]. Not all data is available beforehand, therefore the best way to efficient look at the data is by constantly making a new decision for every new instance.

This method is seen as weaker than pool-based sampling, due to it not seeing the full picture yet and subsequently lacking information [3]. Something which both Settles [1] and Sabato [3] agree

on, is that in theory it will not be able to give the best query for the whole data set and it will take more queries to get the same efficiency. However, according to Sabato [3], there is not always the option to wait for the full data set to be available in reality. This can be due to time, storage or retrieval constraints and is the main reason this method is being used in combination with Active Learning. As a result, Pool-Based sampling might seem better in theory, but Stream-Based sampling is still used in specific cases, e.g. spam filtering.

2.1.3 Membership Query Synthesis

This method differs greatly from the other two in its approach to selecting a new query, as it is a type of generative adversarial network [5]. Where Stream-Based and Pool-Based rely on the data pool or stream for the new queries, this method "generates artificial Active Learning instances" [6]. In other words, this is not an existing data instance, but a new fictitious instance which is created to optimally teach the classifier. This instance can then be used to train the data set efficiently, as the instance will be the most informative due to its creation without the boundaries of having to pick an existing one.

Membership Query Synthesis (MQS) has some pros and cons and two main strategies have been developed to advance these positives and circumvent the negatives. The main added value of MQS is that the predictive error rate is reduced more quickly [2]. This means it is more efficient and less time consuming than the other two querying methods. However, the most significant problem that arises when actually putting this method into practice, is that the human oracle who is queried for the label might not recognise the fictitious instance and will therefore not be able to categorise it [6]. Firstly, Wang [2] has solved this by combining the MQS approach with pool-based sampling. This gives the advantages of both. Only a small labelled data set is necessary, which is much more efficient. In reality, this means nearest neighbour search is applied. In this case, this results in finding the instance which is most similar to the fictitious query, so this instance can be used. Secondly, Awasthi [7] has found another way of circumventing this issue, which is by restricting the MQS approach to only producing queries that lie close to random original examples. This will help the oracle to recognise the query. This is most useful for instances that look a lot alike, so the fictitiously developed instance can be related to another existing instance. These two solutions are quite similar, but even with these complementary methods, accuracy is not always high. This was the main reason Pool-based and Stream-based sampling was developed, as these do not have such limitations. Therefore, for each non-theoretical experiment which is performed, either of the other two methods is preferred.

2.2 Algorithm Types

Once it has been decided how a data pool is considered, an algorithm can be chosen to find which instances to query. There are many algorithms which are all fine-tuned differently, but they can all be categorised into two categories. These are: Uncertainty sampling and Disagreement-based sampling.

2.2.1 Uncertainty Sampling

The most frequent and widely used algorithm in Active Learning is the Uncertainty Sampling algorithm. This algorithm focuses on the instances that are ambiguous to the algorithm, so it does

not query those instances that are confidently known. This confidence is based on a prediction of its correctness when labelling instances. Therefore, the instance with the lowest prediction is chosen. There are variations on this algorithm, as is summed up in Table 1, but they all ultimately rely on the uncertainty principle, which uses the confidence prediction. Figure 1 shows the benefit of using Uncertainty sampling compared to simply using uniform random sampling on the same data set.

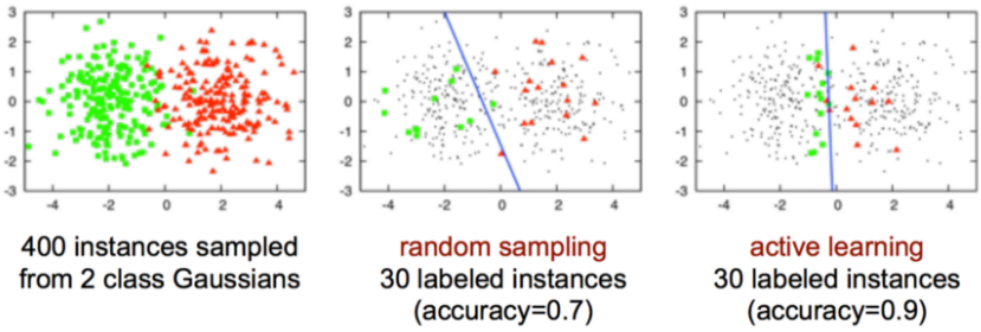


Figure 1: Random sampling and Uncertainty Sampling algorithm in AL [1]

2.2.2 Disagreement-based Sampling

Additionally, disagreement-based sampling compares more than two entities with each other, the most common variations listed in Table 1. This algorithm type most often refers to the comparison of classifiers, called a committee of classifiers [1], [8], [9]. This committee has two or more classifiers and both run simultaneously. The instances they disagree on the most, are queried. This means these classifiers give a single result together. It is a good way of combining different networks, so different experts, into a single output. These usually include a different approach, e.g. different types of errors or learning methods.

However, Disagreement-based sampling can also refer to a comparison of teachers, or oracles [10]. These teachers have the information on the labelled instances and use this to compare with. The committee of classifiers is similar to a comparison of teachers, where each teacher, has a different expertise with an unknown accuracy of how well the expertise is established. However, the instances that are most disagreed upon are not queried, but this information is used to establish a confidence level of the teachers. Here, Dekel [10] wants to "measure the consistency of the binary labels provided by each teacher in different regions of the instance space." This means that by using this method, it is clearer how accurate the teachers are in labelling instances [10]. There are two ways to establish this confidence level. This can be accomplished by testing all the teachers at once, to find the confidence level for each teacher. The second way, is by only querying the teacher of the expert of the instance, to minimise the effort yet still get a result.

2.3 General AL Training

The aim of Active Learning (AL) is to minimise the cost of annotating labels of data sets, often using the minimisation of the bias in unbalanced data sets. Which way is most efficient and balancing this with effectiveness, differs greatly per situation and its circumstances. There are many

Table 1: Variations on the algorithms

Algorithm	Variation	Description
Uncertainty Sampling	Least Confidence	Difference between most confident prediction and 100% confidence
	Margin of Confidence	Difference between top two most confident predictions
	Entropy	Difference between all predictions

methods and algorithms within active learning, but the general procedure always stands.

2.3.1 Data Collection

The first step to classification, is the collection of data. Ideally, this should be a good representation of the actual data, so it is not imbalanced and skewed. However, in practise this is not the case and AL is a way to help this unbalance. All this data is first unlabeled. Then the data set will be divided into one large unlabelled data set and a very small labelled data set, called the seed set. In literature, this labelling process is often referred to as querying the oracle or teacher, but in practice there is often a seed set available, which is a data set which has labelled data in it. This seed set will be queried instead of an oracle or teacher.

2.3.2 Training

Before a data instance can be selected, the learner must first be trained. This establishes how sure it is of this potential label and thereby gives predictions on the label of unlabelled data instances. These predictions will be used in the active learning algorithm. After the learner has been trained and these predictions are established, unlabelled instances can be chosen.

The active learning algorithm chooses a data instance to query which is most informative. Active learning is divided in two main parts, when deliberating on its approach. Any combination can be used of the two. First, a querying method is decided on. This method looks at how a data set is considered, by either looking at the data pool, which is a collection of data instances, or a stream of data coming in. Secondly, the algorithm is decided on. This algorithm looks at an "uncertainty region", which is the region the algorithm has decided it is most uncertain about. This is based on the predicted accuracy of a label if it was classified. After establishing an uncertainty region, one data instance is picked from this region. Exactly how this uncertainty region is calculated and how an instance is picked from this, depends on the algorithm.

After this distinction, the batch size is decided on. This is often a size of one or two. This batch size is the number of unlabelled instances that are queried per iteration. These instances are decided on by the algorithm and added to the labelled data set each time. This process of iteration is repeated until some stopping criteria. Often, the stopping point is at a certain number of instances that are queried, at a number of iterations or when performance does not significantly improve anymore.

2.3.3 Evaluation

To evaluate the performance of the algorithm, there are two metric which are practical to use in the field of activity recognition. Additionally, there are two metric which are used mostly in active learning. Two widely used metrics, which is often used in machine learning algorithms, are the F1-score and Matthews Correlation Coefficient (MCC). The F1-score is a good measure for the performance of activity recognition with a number between 0 and 1 and MCC gives a good measure of how good the prediction is, with a measure between -1 and 1, with 0 being as good as random. Both can be concluded from a confusion matrix. This is a table which shows the performance well, as it shows the concordance between the predicted yes/no values and the actual correct values per activity. Therefore, the values for the True/False and Negative/Positive are given.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Furthermore, two other metrics which are important in Active Learning, but often not in other ML algorithms, is the time the algorithm takes to run and number of unlabelled data instances that are used. This shows how efficient the algorithm is, as the aim is to minimise the number of instances used and thereby minimising the time it takes to label activities.

Therefore, a confusion matrix will be created and the metrics MCC, F1-score, the number of unlabelled instances and run time of the algorithm will be determined to illustrate the performance of the active learning algorithm.

3 State of the Art

Lots of research has already been done on the topic of both activity recognition and active learning. This section includes all essential empirical research on them both, including the different types, strategies and techniques that have already been used and analysed before.

3.1 AL Algorithms' Assets and Liabilities

Active Learning algorithm has been applied in many situations. There are two main divisions in AL, namely Uncertainty sampling and Disagreement-based Sampling, which have been thoroughly explored in many fields. This chapter will consider applications of all kinds of AL applications, divided in these two divisions. Both look at the pros and cons. All algorithms which are considered, make use of pool-based sampling. This was established in the Background section as the best method for experiments of a similar nature.

3.1.1 Uncertainty Sampling

Many agree Uncertainty sampling is useful for Active Learning [1], [11], [12]. It gives an intuitive view into how the AL algorithm works and has a low computational complexity [13], where it still only needs one classifier to train the data with.

From as early as 1994, Uncertainty sampling has achieved better results than random sampling for 9 out of 10 categories which were considered [14]. Nowadays, Uncertainty sampling in AL is applied more and more. For instance, many successful applications can be found in natural language processing (NLP) tasks. These tasks require a lot of data and costs are consequently high. Dredze and Crammer [15] use the confidence margin technique of Uncertainty Sampling on four different NLP tasks and compared the results with random sampling and margin sampling. Accuracy of AL was significantly higher, with 82.5% accuracy for random, 88.06% accuracy for margin, but 95.5% accuracy for confidence margin active learning. In addition, AL required only 63% of the labels of random sampling, while margin needed 73% of the labels of random sampling.

However, Zhu [16] has found that Uncertainty sampling does not work if there are many or significantly large outliers. This is because these are not useful for the system to learn from, yet is hard to recognise for the algorithm. It will need many more instances for the classifier to be successful in learning from the training set. Therefore, Cohn [17] and Trasarti [18] both concluded most variations on this algorithm alone are impractical in real life due to its high computational cost. Therefore, this method is very useful, but will not work optimally if the data is extremely noisy. By combining Uncertainty Sampling with some outlier detecting techniques, this problem can be helped. The research by Zhu and Tsou [16], applied sampling by cluster (SBC) and selective sampling by uncertainty and density (SUD) techniques. The SBC builds a training set which is representative for AL. Usually, a training set is built from random samples, assuming this was representative, but this would then include the outliers. While this worked well, redundancy issues arose. The SUD uses a Nearest-Neighbour-based density measure to determine the outliers. A combination of both methods showed a higher performance than other methods, including Uncertainty sampling.

3.1.2 Disagreement-Based Sampling

Cortes et al. [19] shows the benefits of adding properties of disagreement among hypotheses sets. There was a big improvement, where it first predicted just 3000 instances correctly, and now 12000 instances of the correctly predicted hypothesis set of functions. Additionally, Copa et al. [20] have used a type of Disagreement-based sampling called Entropy Query by Bagging on high resolution imagery. This results showed that especially at the first few iterations the model results became much better than random sampling, as it converged much quicker with less variations. This was the case for two types of classifiers, namely SVM and LDA.

The committee of classifiers algorithm is used most often, which has its benefits. It is especially useful when there are a lot of redundant views within a data set. This is when one instance has two or more mutually exclusive sets of features. Muslea [21] maximizes its efficiency in finding the correct label, by introducing co-testing. This is a combination of a committee of classifiers, while also training the classifiers. This uses several classifiers and uses the points where they disagree on the most, called contention points, to train the classifier. The reason that this works well, is because "it can identify the rarely occurring cases that are relevant" [21], i.e. these points will be most efficient to train the classifier with.

However, co-testing can also be unfavourable. Di [9] elaborates on this by saying he found some circumstances when this way of measuring the contention points is not ideal. This is the case if future data acquisition gives substantially different data points, e.g. a new activity is introduced. According to Di, one way to solve this is to use the instances that have the highest disagreement in the current committee model on extra samples instead of immediately adding them. This means that this "intercommittee distance" can be used to find how far current predictions are from the actual, target, labels [9].

3.2 Animal Activity Recognition

Currently, very limited research has been done on active learning related to Animal Activity Recognition (AAR). However, many other methods and algorithms have already been applied to help the classification process of AAR. This section will analyse these methods and identify potential limiting factors that are applicable in the case of an AL algorithm with AAR.

3.2.1 AAR Algorithms

Many algorithms and classifiers have been applied to AAR. A few which are used often and discussed in this part, are the Naive Bayes method, Convolution Neural Networks and Support Vector Machine techniques. These algorithms are quite different, yet have the same goal and therefore will give a good overview of the field and its challenges and results.

Kamminga et al. [22] have already applied a Naive Bayes (NB) classifier on the data set used in this report. This classifier assumes independence in the features and has a good complexity to performance ratio for AAR. With the use of tuning and balancing, the data was classified in five or six activities. The highest accuracy and F-1 score could be found in the smallest number of activities, so five, with a tuned and balanced data set. The biggest difference can be found in the tuning process of the data set. This combination gave an accuracy of 90% for the five activities.

Many other algorithms have already been applied to other data sets in AAR, including Convolutional Neural Networks (CNNs) and Support Vector Machine (SVM) techniques. Bocaj, Uzunids, et al. [23] use CNN for the AAR of IMU data of horses and goats, classifying them into 6 and 5 different activities respectively. The CNN has four layers, consisting of input, output and other operations, and neurons with learnable parameters. This algorithm surpasses the performance of a Naive Bayes algorithm and its accuracy and F-1 increased with the size of the used labelled data set. However, it did not increase with the amount of convolutional filters, probably due to overfitting.

In addition, a completely different algorithm used in AAR, are Support Vector Machine (SVM) techniques. This technique uses a statistical approach, which is a prediction model which tries to find patterns [24]. This has been applied widely, using 3D accelerometer data. One research by Gao, Campbell, Bidder and Hunter [25] used not only 3D accelerometer data, but combined it with videos. Spatial-domain features, e.g. standard deviation and signal magnitude area, and frequency-domain features were extracted. Another research by Sturm, Efrosinin, et al [26] used this technique on IMU data of calves, classifying their activity into six categories. He split the data in 70/30 for training/validating. However, also other models were applied, e.g. nearest neighbour search, Random Forest and CNN. Eventually, these models were combined, only using the model with the highest accuracy for each activity. This combination resulted in only 71% accuracy.

3.3 Human Activity Recognition

In contrast to AAR, Active Learning has already been applied quite widely in the field of Human Activity Recognition (HAR). HAR is easier to have control over, as humans can be told what to do, like carefully moving to minimise noise. This will most likely give cleaner results with less outliers or imbalance. Additionally, this market is much larger, as more people are leisurely interested in, for example, IMU tracking of humans than horses. However, these fields are rather similar, where both AAR and HAR are most often based on IMU data and both aim to classify activities into categories, e.g. sitting and walking. Therefore, analysing the role AL can play in HAR, can be very helpful in the research about the role AL can play in AAR.

3.3.1 HAR Algorithms

Many other algorithms have also been applied in the classification process of HAR, which reaches across the machine learning spectrum, including supervised learning and unsupervised learning. One common approach is an artificial neural network, which is part of supervised learning. As in AAR, CNNs are often used in the classification process of HAR. The reason this is used often, is because of its effectiveness in recognition. Cho and Yoon [27] use this method in HAR with 1D CNN, by first differentiating between dynamic and static movement. From here, they differentiate between activities. This gave a high accuracy of 94.3%. However, 2D CNN has also been used, as by Jiang and Yin [28]. This creates a "single image", which helps the CNN to extract features.

Furthermore, unsupervised learning has also been applied. Kwon [29] uses Gaussian, hierarchical clustering and DBSCAN when training and uses the calinski-Harabasz index to identify the

number of activities. This combination gives an accuracy of above 90%. Additionally, Li [30] uses auto-encoders and PCA. The best result came from the sparse auto-encoder, with an accuracy of 92.2%.

3.3.2 AL strategies in HAR

Active Learning has already been applied in several situations in HAR. Different querying methods and algorithms have been used and all had different findings. The most commonly applied AL models are most often used in combination, namely Pool-based sampling with Uncertainty sampling. In Table 2, all papers on AL named are shown in an overview.

A very general research was conducted by Liu [31] on the potential role of Active Learning in HAR. Both Pool-based sampling and Stream-based sampling are considered, but only pool-based sampling is applied to find the best algorithm type. However, the two types of algorithm are both tested on a data set, namely Uncertainty sampling and Disagreement-based sampling. To find a stopping point for the AL algorithm, the minimum mean squared prediction error (MSE) was applied. Optimally, the MSE should be small, which is achieved with a low bias and variance, but this is difficult in practice. To train the classifier, first a small labelled data set of 20% is used and later a size of 30%. After training and classification, the most informative instances were added to the training data set and reached up to 40%. When the two algorithms were tested on the same data sets, it could be concluded that indeed it found the instance which was most uncertain or disagreeable, was most informative to train the classifier with. Lastly, it was also concluded that indeed AL outperformed supervised learning or random selection and needed less samples on those same data sets to achieve this.

Other papers used the same querying methods and algorithms. Stikic [32] used a combination of Pool-Based sampling with two different algorithms, Uncertainty sampling and Disagreement-based sampling in HAR. This was applied to categorise ten activities, both on the same data set. For Uncertainty sampling, two samples are chosen each time iteratively which are reckoned the most informative, while for disagreement-based sampling, one sample with the highest disagreement is chosen. The results showed that there is little difference in accuracy between the two, but both did see a large increase when the number of labelled instances increased.

Additionally, Vaith et al. [12] have done research on the role active learning can play within IMU data of humans, by doing a human gait analysis. The AL approach they used was Pool-based and used variable strides of the IMU data of one time step. The algorithm is based on iteratively feeding new labelled data to the classifier which it is most uncertain on, so by using the Uncertainty method. These labelled data instances are based on an acquisition function and they found the Variation Ratio (VR) strategy and Maximum entropy (EM) strategy to be the most accurate within this Uncertainty method.

Another research, by Adaimi [33], used Uncertainty sampling to compare Pool-based sampling and Stream-based sampling with supervised sampling. Both were tested on four different data sets. The taret batch size was 2% and ended op not querying more as this was suffice, for both strategies. For all four data sets, both AL methods outperformed the supervised algorithm. However, not a clear comparison can be made, as the AL methods all used different data sets and parameters.

Table 2: Paper summary AL applications

Paper	**Problem**	**AL technique**	**AL advantage**
Lewis and Gale [13]	Text classification	Uncertainty sampling	9/10 categories with Uncertainty sampling better than random
Dredze and Crammer [15]	NLP task	Uncertainty sampling: margin	82.5% accuracy for random, 88.1 % accuracy for margin, 95.5% for confidence margin AL
Cortes [19]	Incorporate disagreement into AL	Disagreement-based	3000 to 12000 of the correctly predicted set of hypothesis functions
Copa [20]	Testing Entropy Query by Bagging	Disagreement-based	beginning iterations converge quicker than random
Liu [31]	Find stopping point AL	Uncertainty and disagreement	AL accuracy was up to 75.96%, while the accuracy of supervised learning was often 4 to 5% lower
Stikic[32]	Categorise activities AL with supervised, self-training and co-training	Uncertainty and disagreement with co-training	lowest accuracy was supervised, then self-training, the co-training. This increased from 0.25, to 0.3 to 0.35 accuracy
Vaith [12]	Human gait analysis	Uncertainty: Ratio and Entropy	max entropy was the quickest in convering and had a F1 score of 96% with the least amount of labelled instances. The lowest was random, with a F1 score of 95%, but needed more labelled instances.

3.4 Activity Recognition

While the type of algorithm plays an important role in Activity Recognition, this is not the only factor which has an influence on the performance of the algorithm. There are several pre-processing steps and parameters which can be adjusted to maximise performance for AR, together with some limitations which will impact the performance of the AL algorithm, which were found in

Chapter 2.2 and 2.3. These are discussed in this section.

3.4.1 Classification Result Reliability

It is important to understand how well the classification of the data resemble the actual behaviour of the animals, to know if the results of the experiment are meaningful. In the study on horses' gait by Casella, Khamesi and Silvestri [34], several factors were established which influenced the natural activity of the horses and therefore the validity of the data. The two main factors were on placement and obtrusiveness. With this combination in mind they considered the type of data which has to be collected. Tracking devices such as cameras have to be deployed in a controlled environment, which is usually not their natural habitat, while on-body sensors can be used regardless of habitat. However, on-body sensors are loose, can fall off and are potentially noisier. The latter is solved by preprocessing.

The research by Sturm, Efrosinin, et al [26] found a way to use pre-processing to eliminate noise. This was by the use of filtering, namely lowpass and bandpass filters, and by finding the outliers. The research shows the large impact this transformation has on the AL strategy. Furthermore, Casella, Khamsei and Silvestri [34] also identify and remove outliers. However, this was done differently, namely by calculating a "global" feature, then comparing that to a single feature and then those that are below a threshold from the average, are named the outliers and removed.

In addition, the sensor may have moved around a lot. This can give unreliable data. This was also the case for Sturm, Efrosinin, et al [26]. Due to this rotating, individual coordinates cannot be used, therefore an "orientation-independent signal has to be evaluated" [26]. A solution their paper gives, is to use a signal vector magnitude of the data, which will make use of all axis of movement for more stable results.

3.4.2 Sliding Window

The data which is being processed is continuous time series data. Consequently, the data first has to be sliced into parts, to be able to extract activities. This approach is called the sliding window approach. This is a window which is put over part of the data and by sliding it, gives different parts.

Data instances of continuous AAR data often overlap, because there is not always a clear cut between different activities, e.g. to go from standing to running, the animal will have to accelerate and set off. Usually, this overlap is either at 25% or 50% as this gives the best results. For the Naive Bayes classifier which was applied to the horses' IMU data set [22], each section had an overlap of 50% and a window length of two-seconds. The maximum length was ten seconds, as they found this improves the class balance. Additionally, other researches with a similar data set have done comparable. The researchers Bocaj, Uzenidis et al. [23] also used an overlapping of 50% and a two second window, based on this work [22]. Furthermore, another study by Gao, Campbell, Bidder et al. [25], has similar data. They used a window size of three seconds, with a one-second overlap. This is because of the high accuracy of a window of 50%. An activity can be captured with a window of two seconds, so this results in a window of three seconds with one second overlap. This gave two sampling points for a 1 Hz sampling rate. When calculating a FFT, this usually works best with a time window length that is a power of two, so this is beneficial too [25].

To find the impact of this sliding window, an analysis was conducted [34] on the sliding window size and frequency rate. The result of using 10-fold cross validation of 75/25 training/testing distribution was that 5 Hz was not significant in differentiating activities. However, an 8 seconds sliding window size and 10 Hz frequency rate or more gave a approximate equal result. In the end, a 6 seconds sliding window size was used with a frequency rate of 20Hz on a smaller data set, as this would give a higher accuracy. Therefore, the sliding window size and frequency depend on the size of the data set and have a significant effect on the algorithm's accuracy.

3.4.3 Activity Classification Limitations

Potential application problems of active learning are also important to consider, which can be recognised and learned from other classification methods. In the Naive Bayes approach on the horse data set [22], some problems were encountered. The differentiation between walking, standing and eating was not clear, because often grazing is done while standing or walking. In addition, galloping and trotting were often confused, because these are similar and the transition from one to the other is not clear-cut.

Furthermore, the walking class in the horse data set is much larger than the other classes, which creates a bias for the NB classifier. This means the data set is imbalanced. According to Sturm, Efrosinin, et al [26], when rare classes have to be found, this can create inaccuracies. They propose some remedies: to oversample these classes, undersample the other classes which are present more frequently or a combination of both. However, a direct implementation was not presented. Kamminga [22] tried solving this with random undersampling. However, this meant that labelled data is disregarded, even though this could give useful information.

Additionally, Kamminga [22] has not only balanced the data set, but also applied parameter tuning and has measured its affect on accuracy and F1 score. Tuning improved the F1 score by 1.6% and balancing with 0.5% to 0.6%, dependent on the amount of activities classified. However, for the lowest increase in F1 score, accuracy went down again after tuning.

Moreover, in the SMV approach of Gao, Campbell, Bidder and Hunter [25], they compared several data sets. There was shown that accuracy and precision rapidly declined when classification was more fine-grained, with harder to distinguish activities, e.g. foraging and climbing. In this case, accuracy was never above 87% and precision differed greatly per activity. However, for another data set where the categorisations could more easily be established, e.g. walking and standing, all results of the performance were high. The accuracy was always more than 95% and the other three were always more than 90%.

Lastly, research was conducted on the effect of the algorithm' performance of the number of activities that are to be classified. This research was conducted by Yang, Ma and Nie [35] and a KTH action data set was used with either six or eleven activities to classify. The results of his research can be found in Figure 2. There can be concluded that the amount of activities that are to be classified have a big influence on the accuracy of the labelled instances and that this effect differs per AL algorithm strategy. While this is a different type of data, namely video data instead of accelerometer data, this does give insight into classification parameters. This concurs with other classifiers, e.g. with the NB classifier which was applied to the horse data set [22].

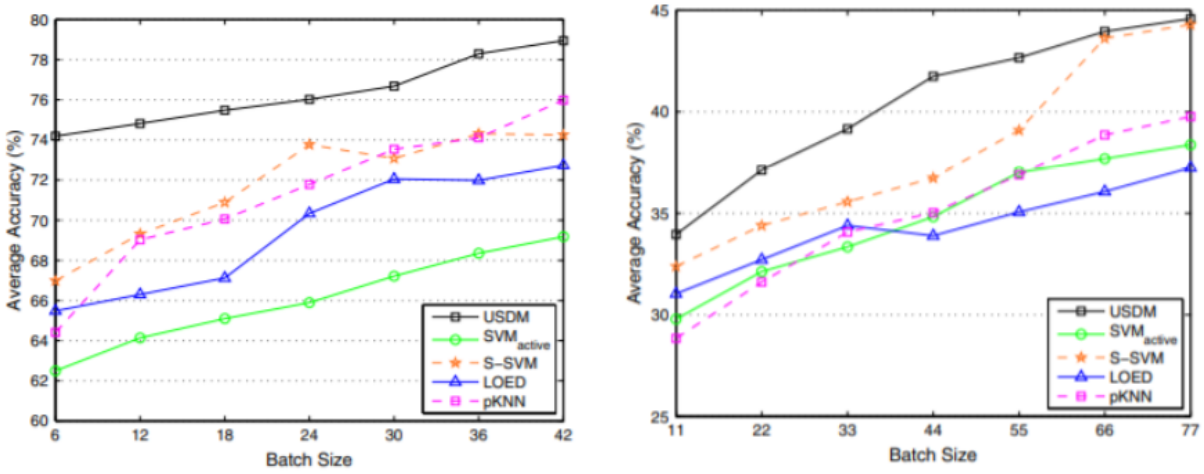


Figure 2: AL algoriths on action recogition using KTH data set with 6 actions and YouTube data set with 11 actions

3.4.4 Data Set Size Performance

Adaimi [33] found that both Pool-based and Stream-based sampling outperformed supervised sampling in HAR. However, for this to always be the case, the unlabelled data set must be sufficiently large. The benchmark which was identified was at a size of at least as large as "ExtraSensory" data set, which was 10 times bigger than the others. The noisier and more variations the data instances have, the bigger the data set should be. Therefore, a smaller data set could still cultivate a good performance, but this decreases with the increases in unclean data.

Furthermore, all research [31], [32], [33] showed that when the training set was bigger, there were better results. This means that the AL strategy is allowed to make more iterations and the accuracy goes up. However, a clear stopping point was not often established, as usually this is determined by estimating and tweaking parameters. This meant that while indeed bigger is better, no conclusions were drawn as to its optimal size.

3.4.5 Stopping Point Maximisation

The research by Adaimi [33], focuses on the issue of finding a stopping point in the AL algorithm applied to HAR. This is important, as this can maximise the performance of the AL, by finding a balance between labelled instances needed and accuracy. He introduced a way to determine a stopping point in the AL process which maximises its performance. This stopping point is based on a Conditional Mutual Information (CMI). It was tested on a Pool-based sampling strategy, as Stream-based sampling cannot be used, because its mechanism rests on the use of entropy of a unlabelled instances pool. Using this criterion as stopping point, nearly no information gain is missed out on, meaning it has maximised its potential. However, while the results were promising, there was acknowledged that this varies highly per data set and the diversity of the data. Alternatively, a stopping point based on a target performance was proposed, but this gives different issues due to the unreliability and variability of the AL algorithm. The classifier is not constant and sometimes accuracy goes down before going up again, hence stopping too early, or it

will never reach its potential, which means it would never stop.

3.5 Challenges

In this chapter, some challenges arose which have to be taken into account when designing the Active Learning strategy, as they could potentially influence the result. These points can be divided into those that are specific for the active learning algorithm and for activity recognition.

3.5.1 AL Challenges

The aim of this paper is to establish the best AL algorithm for the IMU horses data set. Therefore both Uncertainty sampling and Disagreement based sampling will be applied. However, some challenges arose during the empirical research in this chapter.

One challenge which was established, was the presence of outliers, as the algorithm is prone to mistakes. Accuracy can drastically plummet. However, there are many techniques to combat outliers with pre-processing. In combination with Uncertainty sampling, SUD and SBC have been applied and both showed a significant improvement. Therefore, outlier detecting will have to be investigated if necessary.

Additionally, an imbalanced data set can negatively impact any classifier, as it will favour one class over the other. While AL already helps enormously with this problem, under or oversampling the data will also help the algorithm consider the data fairly.

Moreover, computational power is also higher for disagreement based sampling and must be considered when developing the algorithm. This trade off can be substantial in many instances, as AL is often used to reduce these computational costs.

3.5.2 AR challenges

Also activity recognition in combination with AL will have its challenges. One challenge of activity recognition, is the high computational cost. Computational cost will always be an issue and the aim will always be to minimise it. The fastest way with the least amount of effort is the ultimate goal of any algorithm, as this means it cannot be improved further. This is something which AL aims to resolve, as it will minimise the amount of labelled instances used. However, there are still ways to minimise this cost in AL and this must be considered when developing the algorithm.

Furthermore, pre-processing can play a big role in the increased performance of the AL algorithm. A challenge which has to be considered, is that of noise. This has a significant impact on the AL results. Several methods to resolve this problem have already been suggested, including: outlier detection and elimination and filtering. Additionally, the imbalance of the data set will have an effect on the performance of the AL. As was already established, the walking class is much larger in the horse data.

Also, the choice of activities to be classified is important. Firstly, the number of classes that are classified affect the performance. The decision on the number of classes must be considered carefully, to find this balance of performance and classes. Secondly, exactly which activities are

chosen to distinguish between, will also affect performance. If the activities are harder to distinguish, due to e.g. overlapping values, the performance will goes down.

Moreover, the size of the unlabelled data set and that of the end training set affects performance greatly. The bigger the unlabelled data set, the better the performance. Additionally, the bigger the end training set can become, the better the performance. However, the aim of AL is to minimise both of these. Therefore, the correct balance must be found. To find the latter, a good definition for the stopping point must be found.

4 Approach

4.1 Focus

There are many parameters and methods that could be tested. To answer the research question, the focus will be on comparing the two different sampling types disagreement based sampling and uncertainty sampling. These will be applied to a deep neural network (DNN) classifier. There are three algorithm types within uncertainty sampling which will be compared, namely: least confident, margin uncertainty and entropy. Disagreement based sampling will consider two types, namely: maximum disagreement and consensus entropy. The difference in performance of the DNN with and without AL show the benefit of AL. Additionally, all algorithms will be compared to manual annotation, to find the benefit of using AL over labelling all instances manually.

4.2 Design Decisions

Some assumptions and design decisions have been made before application, to efficiently and accurately compare the various algorithms. These include deciding which assumptions can be made with which baseline variables and which Active Learning variables should be compared.

4.2.1 Baseline Variables

Three querying methods were discussed in the background, with pros and cons and subsequent concerns. As can be concluded, pool-based and stream-based sampling are the two best options when used in real experiments. Seeing as the whole data pool is already available beforehand, Pool-based sampling will be the best method to use. A lot of thorough research has gone into the differences between the sampling techniques and all research in the field of active learning supports the decision to use Pool-based sampling in this kind of experiment. Therefore, this conclusion can be drawn without further investigation and pool-based sampling will be used.

Additionally, two divisions were made within active learning algorithm types. These were uncertainty sampling and disagreement based sampling. To find the affect AL can have on a classifier, AL is applied to the CNN and compared to it with and without AL. Both divisions can be considered and compared.

The amount of activities to classify is another variable which has to be considered beforehand. The more classes, the higher the inaccuracy and room for confusion there is. However, more classes also give more information. This AL algorithm will classify six activities. The Naive Bayes classifier which was applied to the data set before [22], showed sufficiently good results with six activities, where accuracy was high, yet the results were still of significant gravity.

Moreover, a variable which impacts the performance of the AL algorithms, is that of the size of the pool set and test set. The pool and test will first be divided by using 1 horse for testing and the others for testing. This divide of approximately 20/80 for training/testing is most often used in literature. There is iterated for all four horses and the average of these results is used in the evaluation.

Furthermore, how the samples for the initial training set are selected is also of importance. This can be done in several ways, for instance by stratified sampling or random sampling. Stratified

sampling has the advantage of countering imbalance and to already get a good idea of the results. However, in this case there is chosen for random sampling. This is because this method most fairly gets an indication of the effect of AL, as AL is usually applied without prior knowledge of the dataset. Moreover, the affect of AL can be shown, as the baseline of the performance of the classifier without AL is at this beginning point, seeing as this is random sampling without AL.

Additionally, the data which will be used has already been collected. However, before this can be classified, some preprocessing steps must be considered, which will influence the performance of the AL results too. These basic steps include feature selection, splitting the data, feature scaling, windowing, shuffling, reshaping and encoding labels. Additionally, the data is filtered with a low pass filter. Furthermore, the data set is unbalanced, as can be seen in Figure 3. This will be facilitated by preprocessing too.

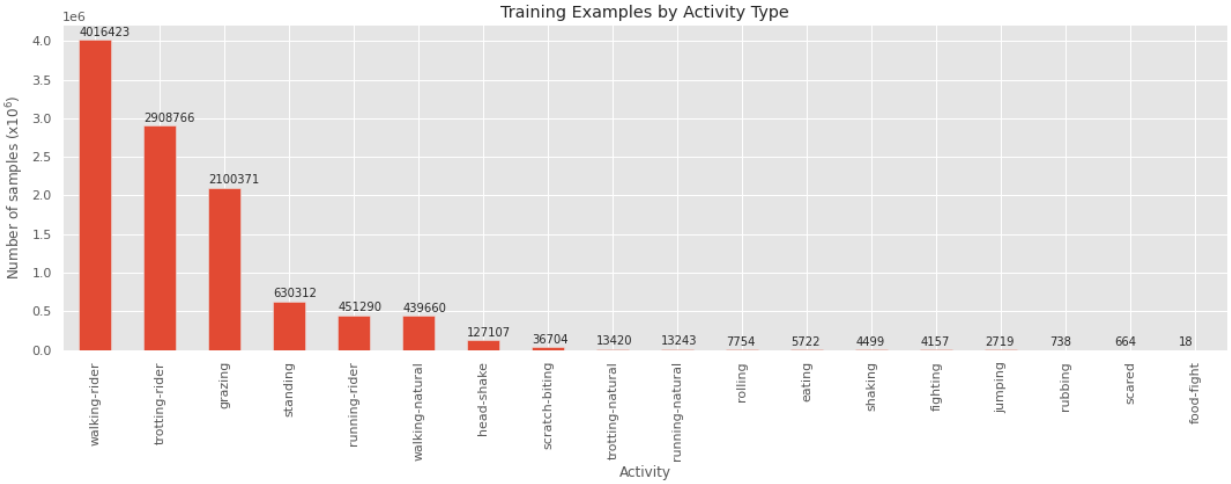


Figure 3: The distribution of labeled activities for all horses. [22]

5 Methodology

5.1 Dataset

The data used in this Graduation Project was gathered by Kamminga et al. [22]. This dataset consists of data gathered from 18 different horses and ponies across a period of seven days, during which the horses participated in both riding and free roaming activities throughout their pasture. The animals wore an inertial measurement unit (IMU), containing an accelerometer, gyroscope and magnetometer, in a collar. These IMUs made use of a 100 Hz sampling rate, recording a total of 1.2 million data samples, each describing a 2 second segments, by the end of the week.

As the collar containing the IMU sensor can still slightly move and rotate around the animal's neck, the dataset also includes l2-norm values for each of the sensors. These values can be used to compensate for any recorded movement of the collar that does not correspond to movement related to the horse's activity.

The dataset consists of labeled and unlabeled data, of which only labeled data will be used. The used data as a whole is not equally labeled; only data from 11 subjects were labeled, of which four subjects and six activities were labeled more extensively and gave enough information, which can be seen in Figure 4 and 3. Therefore, these are the activities and subjects that will be used for this project.

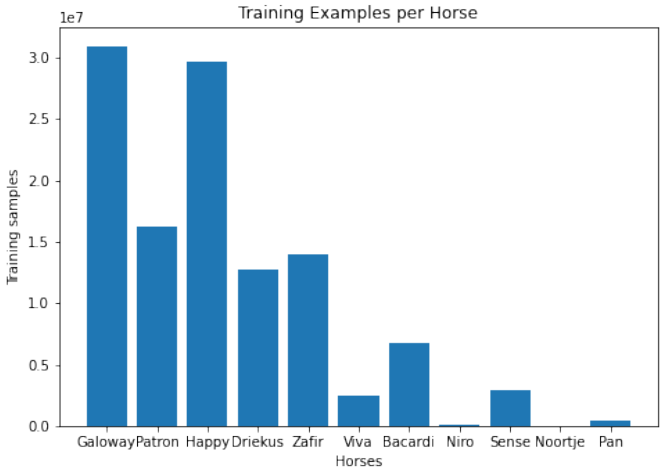


Figure 4: The distribution of labeled samples over the different horses.

The data is contained in CSV files, describing the x, y and z and 3D vector with l2-norm values of the accelerometer, gyroscope and magnetometer. Next to that, the subject, segment, label and date and time are denoted.

In Figure 5, 6 and 7 one data sample can be found for the activities eating, running-natural and

shaking.

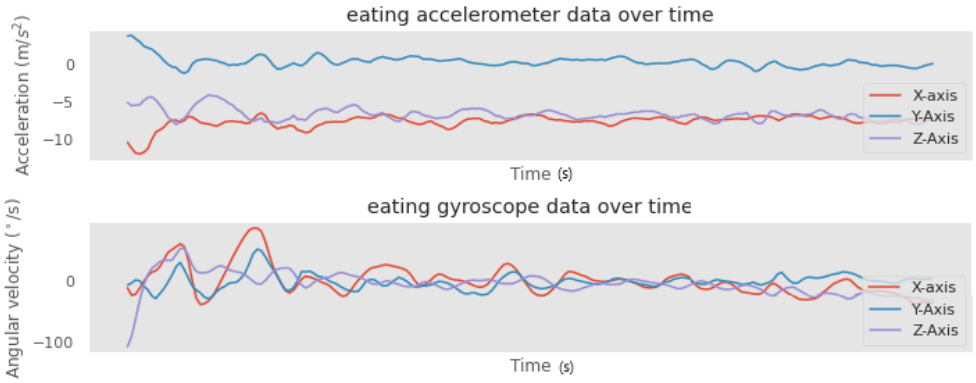


Figure 5: An accelerometer and gyroscope measurement of a horse eating.

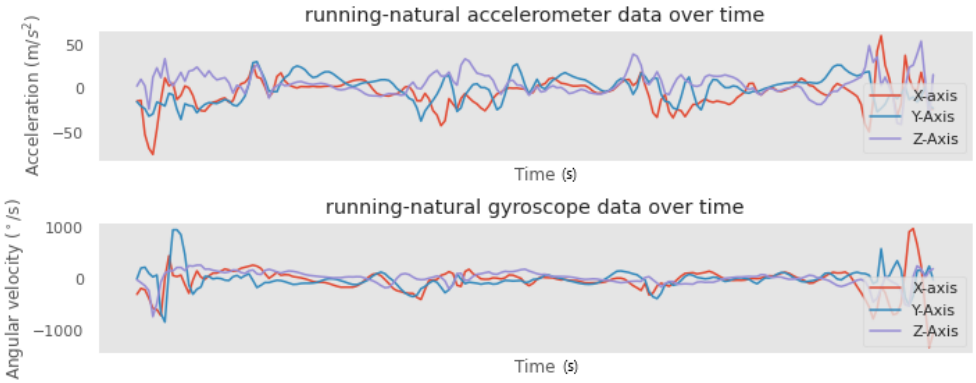


Figure 6: An accelerometer and gyroscope measurement of a horse running naturally (without a rider).

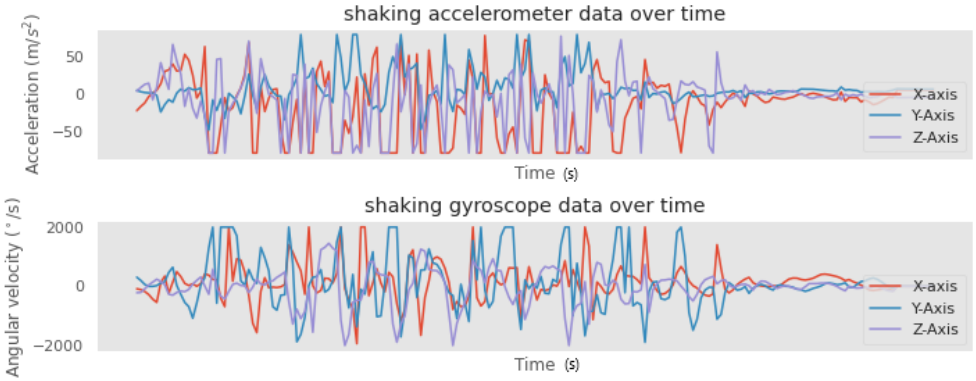


Figure 7: An accelerometer and gyroscope measurement of a horse shaking.

5.2 AL Process

The process of developing and evaluating the active learning algorithm for the horse data set can be summarised in six steps. These steps will take the preprocessing steps and baseline variables established in the Approach section into consideration.

1. The data will be divided in a pool subset and test subset, with a split of approximately 80/20. The pool set is divided into a small initial training set (e.g. 50 labelled data points) and a subset of data which includes the rest of the pool set.
2. This data is then preprocessed for optimal use and reshaped to input into the classifier.
3. Some initial choices are made. Pool-based sampling is used with a batch size of 1. First, one type of uncertainty sampling will be applied to a SVM classifier to visualise the power of AL. Then, for the rest of the experiments, the learner will be trained with a Deep Neural Network (DNN) each time. Three types of uncertainty sampling and two types of disagreement based sampling are applied to the DNN and compared.
4. The model is trained by iteratively letting the AL algorithm choose the most informative instance, which is the most ambiguous, and then adds this to the training set. An algorithm is chosen to find which instances to query, e.g. marginal uncertainty sampling. This will differ per experiment and the results will be compared. The model is then trained again with the newest information.
5. A stopping criterion is used to decide when to stop querying for unlabelled instances, in this case established by trail and error. This will depend on the size of the initial training set and when the AL stopping giving informative instances. During the experiments, trail and error will be used to decide which iteration number this will be.
6. The evaluation will be conducted by comparing the algorithms applied to the DNN with each other, to the DNN without AL and manual annotation. Firstly, the different algorithms are compared to each other by plotting the accuracy against the amount of labelled instances. Then, the best uncertainty sampling and disagreement based sampling will be compared to each other. Secondly, a comparison between the labelling time and manual labelling time is made. By comparing these, the advantage of the minimisation of the number of labelled instances used in AL is clearly highlighted and there can be shown which AL strategy does this most efficiently. The metrics which will be used, are F1-score, MCC, labelled instances used in the initial training set, number of instances queried and the run time.

5.2.1 Active Learning Variables

The first factor to take into consideration is the number of times the AL algorithm has to query labels of instances, so how many iterations the AL needs. The number of iterations has a great effect on accuracy, especially since AL is not very consistent. If it is too small, it will never reach its full potential and will subsequently give a low accuracy. However, if it is too large, the benefit of AL is wasted, as it has already reached its maximum and would still ask for labels. Furthermore, it could be that this point is different for each algorithm. Therefore, the number of iterations will be a variable, which will be considered.

Additionally, a variable to consider is that of the size of the initial training set. The data set was already divided in pool data and test data. However, the next step would be to take out a certain number of data points and use that as training data. This will be used as initial training data, where one, most ambiguous, data point will be queried, labelled and added to the training set per iteration. This is illustrated in Figure 8, where X is the variable which will used to investigate in this thesis.

Lastly, the algorithm with which the find most ambiguous function works will be investigated. This way, various strategies can be compared. These algorithms are uncertainty sampling types, namely least confident, margin and entropy and for disagreement based sampling consensus entropy and maximum disagreement.

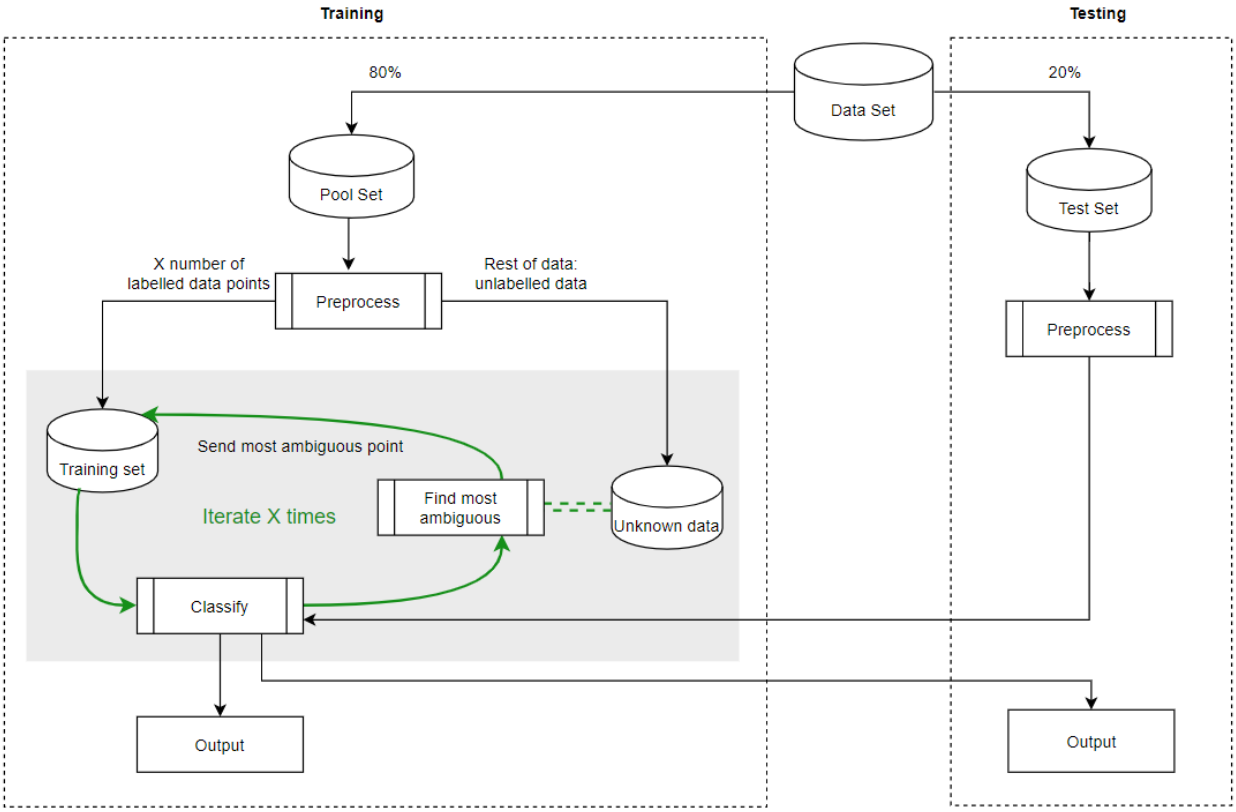


Figure 8: Active Learning application

5.3 Horse activity recognition pipeline

The pipeline is split up into three classes: preprocessing of the data, the database interactions, and the main class, which also contains the classifier, active learning structure and uncertainty algorithm.

5.3.1 Preprocessing data

Before the data can be used for classification purposes, it has to be preprocessed for optimal use. The data must select sensor features, filter the data, split the data, scale, window, shuffle and reshape that data and encode the labels.

Dataset usage A sub-selection of horses is first made as for some horses there was comparatively little data available to facilitate unbalance. The activities that do not have a lot of data points were removed and some smaller activities have been combined into one bigger activity. Additionally, only the best four horses are used, to counter imbalance, but also due to time constraints. After this selection, the activities are more evenly distributed, as can be seen in Figure 9

The selected horses are Galoway, Patron, Happy and Driekus. For the activities, trotting-rider and trotting-natural are combined, as well as for the running-rider and running-natural activities, since these are similar activities of which not both activities contain enough samples to be used in this project. Thus, only data from the horses Galoway, Patron, Happy and Driekus will be used for the activities walking-rider, trotting (rider and natural), grazing, standing, running (rider and natural) and walking-natural. The remaining dataset contains 9403903 labeled (unwindowed) samples. The corresponding data sets are combined into a single dataframe and rows where values are missing are removed.

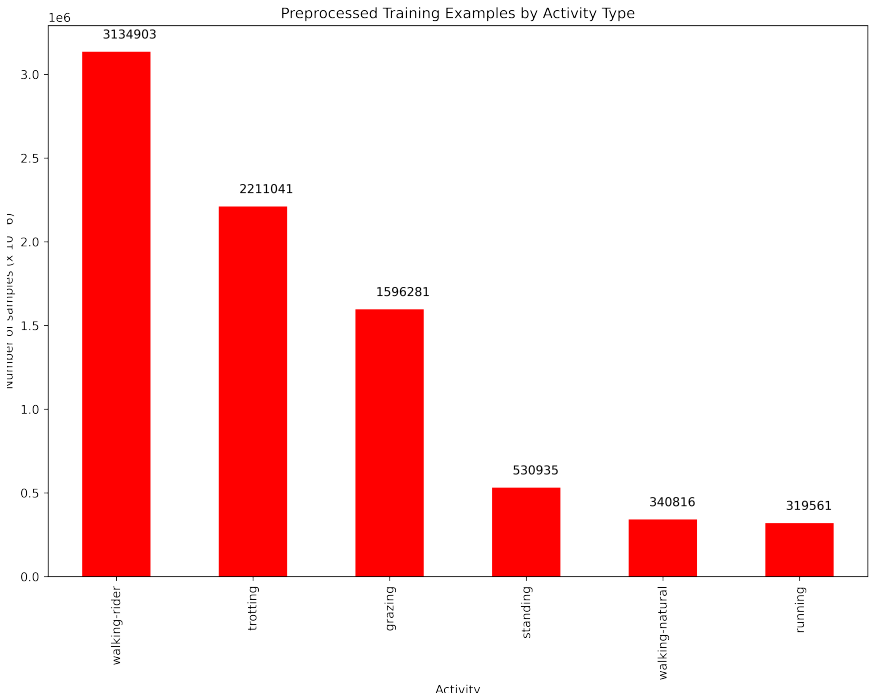


Figure 9: The distribution of labeled activities for selected activities and four horses

Sensor selection The three features describing the various magnetometer and gyroscope axes are dropped, as this data was found to be too prone to alterations as a result external distur-

bances, such as magnetic fields, and thus unreliable as a whole. The 3 axes of the accelerometer combined in a vector are used. Additionally, the l2-norms of the various sensors are also not used during the study, and as such are removed from the feature set. This results in the following feature set, seen below in table 3.

Table 3: Overview of the feature set. Adapted from [22, p.4]

Feature	**Description**
A3D	Raw data from the accelerometer in a 3D vector
label	Label that belongs to each row's data
segment	Each activity has been segmented with a maximum length of 10s. Data within one segment is continuous. Segments have been numbered incrementally.
subject	Subject identifier

Data filtering The accelerometer and gyroscope measurements are inherently noisy. Thus, it is important to filter out high-frequency noise from the measurements. This is done with a low-pass Butterworth filter with a cut-off frequency of 30 Hertz. Arablouei et al. [36] show that the most power in the signals ranges from 0 to 25 Hz, making 30Hz a reasonable cut-off frequency for high-frequency noise.

Splitting the data The dataset is split into a pool data set and testing subset and later the pool data subset is divided into a training and unknown data subset. This process is visualised in Figure 8 . This method was also used by Kamminga et al. [37] for the same dataset, to address heterogeneity. During the testing phase, one of the four horses is selected as the testing subject and the other four horses are included in the pool subset. For example, if Galoway is used as a test subset, then Patron, Happy and Driekus are included in the pool subset. These steps are performed for all four horses, therefore testing is done four times.

Feature scaling Since some of the values are of a much larger size than others, it is important to scale them so that they are easily comparable. This is done after the splitting of the data, so no test data is used for training or vise versa. To do so, the accelerometer and gyroscope samples are divided by the highest value within the corresponding axis to obtain normalized values between 0 and 1.

Windowing, Shuffling, Reshaping Windowing is done with a sliding window of two seconds with 50% overlap, so a step distance of one second. These windows can now be used as data instances for training. All training data is shuffled with the use of the `shuffle()` method from sklearn. After shuffling, the training data is reshaped from a array with dimensions [number of labels, 200] into a one-dimensional array to fit into the classifier.

Encoding labels In order to make the dataset more suited for most machine learning algorithms, the categorical labels should be converted into numerical ones. However, as there is no ordinal relationship between the original categorical labels, one-hot encoding should be applied to

any numerical label representation to avoid the algorithm potentially trying to make use of any non-existent ordinal relationship.

To do so, the various activity labels are first converted into numerical labels using the `LabelEncoder()` function provided within scikit-learn's preprocessing library. Following this, one-hot encoding is applied to the integer representation of the labels. The resulting encoded labels are added as an extra column to the dataframe.

5.3.2 Classification

Active Learning After splitting the data into a pool data set and test set, the initial training set can be formed from the pool data set, which will be variable DP, e.g. 50, data points. The rest of the data points from the pool data set will be in an unlabelled data subset, known as the rest subset. With the initial training data, the pool is trained with the classifier. Then, the most ambiguous point is selected from the unlabelled rest subset, labelled, and added to the training set. This is trained again. This is iterated variable IT, e.g 20, number of times, to finally get a definite result. These two variables will try to be found optimally in the experiments. Additionally, the way the most ambiguous point is selected is done by three different types of uncertainty sampling and two types of disagreement based sampling, namely least certain, uncertainty margin, uncertainty entropy, consensus entropy and maximum disagreement

Algorithms The uncertainty sampling algorithms use a formula to calculate which point to query from the unknown data subset, so all data from the pool set which is not used in the training set. This uses just one classifier, the DNN, to train the data. However, in disagreement based sampling several classifiers are used and compared to each other. After a prediction, they vote on the one with the most disagreement. These algorithms use two DNN classifiers, but both start with different samples in the initial training set.

DNN classifier The classifier used is a sequential classifier from Keras, which represents a neural network with multiple layers, as depicted in Figure 10. The first layer is a Reshape layer, where the training data is reshaped back into 6 dimensions. Next, three Dense layers are added, representing three hidden layers in the neural network, each with 100 fully connected nodes. The activation function used in each of these layers is a rectifier, which is the ReLu activation function. After the three hidden layers, a Flatten layer is added to flatten the data. The last layer is the output layer, which is a Dense layer with the same number of nodes as the number of activities and a softmax activation function. After training and having iterated IT times, an evaluation is performed and a confusion matrix is constructed from this data.

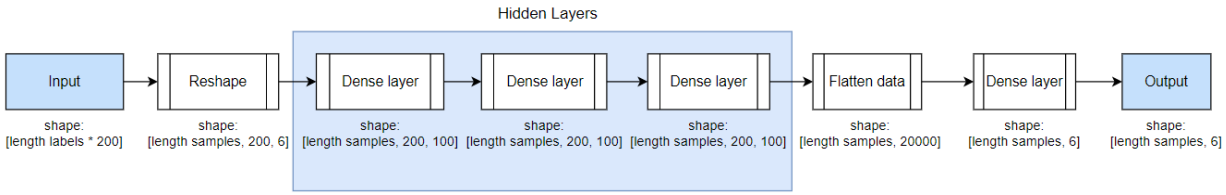


Figure 10: DNN structure

5.3.3 Evaluation

As mentioned above, the testing phase is performed four*IT times, once per tested horse per iteration. To get the results of the F1-score and MCC per the iteration, the average of the horses is taken. The steps are performed in the same order as described above: splitting the data, feature scaling, windowing, shuffling, reshaping, encoding, training the classifier and lastly, testing. Firstly, the sensor data in the test data are normalized, after which windowing and reshaping is performed. The only difference in the preprocessing of the test data and the training data, is that the test data does not get shuffled. Lastly, the model is tested with the Keras `predict()` method. Per horse, the experiment performance is saved in the database. Also, the performance of each activity per horse is also saved in the database.

5.4 Tools

In this Graduation Project multiple tools are used, which will be described below.

5.4.1 ITC Geospatial Computing Portal

The ITC Geospatial Computing Portal from the University of Twente is used to run the code on [38]. This portal allows, among other programming languages, Python 3 code through the JupyterLab environment [39]. Apart from running the code, the CSV files containing the raw IMU data can also be stored on this server.

5.4.2 Programming language and packages

For this project Python 3 is used with a couple of packages.

pandas The Pandas package supports data analysis and data manipulation, allowing the user to retrieve and shape the data [40].

numpy The Numpy package allows mathematical calculations [41]. In this case, these calculations are used for conversions between number types (integer, float, etc.), retrieving maximum values and shaping arrays.

scikit-learn The scikit-learn (or sk-learn) library is made for predictive data analysis [42]. In this project, it is used to define the metrics and for preprocessing.

keras The keras package focuses on Deep Learning, including methods for implementing a Deep Learning algorithm [43].

peewee The peewee package is used to implement a database to store and retrieve the results of experiments [44].

seaborn Seaborn is used for data visualization [45]. In this case, seaborn is used to plot a confusion matrix.

Other packages Other packages used are scipy, glob, matplotlib, uuid, IPython, datetime, re and ast.

5.5 Database

By the use of the Python package Peewee, a database is created to easily store experimental data. This database is connected to a virtual server. There are two tables, one for the experiment results per horse and one for the experiment results for a specific activity per horse.

The experiment table consists of several columns, containing a summary of the results from all activities:

- Key: a unique key is generated per experiment with the uuid Python package.
- Username: The name of the person who performed this experiment.
- Horse: The name of the horse which is in the test dataset.
- Date: The date on which the experiment was conducted.
- Accuracy: The overall accuracy of this experiment.
- Balanced accuracy: The overall accuracy of this experiment, balanced on the amount of samples per class.
- F-score: The overall weighted F-score of this experiment.
- MCC: The overall MCC of this experiment.
- Recall: The recall of this experiment.
- Confusion matrix: The confusion matrix of this experiment.
- Parameters: The parameters used for this experiment, including Time Periods, Step Distance, Epochs and Batch size.
- Description: A description of the specific settings used for this experiment. This can, for example, entail of the structure of the classifier. The description is added in text-format.

The activity table also has several columns, focusing on each activity within an experiment:

- Key: A unique key, corresponding to the experiment.
- Horse: The name of the horse which is in the test dataset for this activity.
- Activity: The name of this activity.
- Accuracy_activity: The accuracy of this activity.
- Recall_activity: The recall of this activity.
- Specificity: The specificity of this activity.
- Precision: The precision of this activity.

- TP: The number of times the system predicted Positive and the actual value was also Positive (True Positive).
- TN: The number of times the system predicted Negative and the actual value was also Negative (True Negative).
- FP: The number of times the system predicted Positive and the actual value was Negative (False Positive).
- FN: The number of times the system predicted Negative and the actual value was Positive (False Negative).

6 Results

The experiments have been divided into two parts. The first, is to visualise and consider the potential performance of the AL algorithm. This is done by showing the power of an AL algorithm step by step with a linear SVM classifier, by depicting the iterative process in a 2D graph. Secondly, the three uncertainty sampling AL algorithms and the two disagreement based sampling AL algorithms were evaluated on a DNN pipeline. The uncertainty sampling uses the DNN classifier once and the disagreement based sampling uses two DNN classifiers, but each with different samples in the initial training set.

6.1 Linear SVM Classifier with AL

To visualise the power of AL in practice, two features have been chosen to be plotted against each other, as it is clearer to interpret 2D plotted data. The features chosen are the standard deviation and the magnitude. The activities have been divided in two, to depict the boundary more clearly in a graph, comparing stationary activity to dynamic activity of a 3D vector of accelerometer movement. For clarity's sake, grazing, standing and eating were chosen for stationary; trotting and running were chosen for dynamic. Walking was not considered, as this overlapped with grazing and consequently cannot clearly be depicted in the graph. Some other activities, like fighting and rolling, were also left out because of the large amount of outliers. As the point of this visualisation is to depict the power of Active Learning in a graph, this scattered depiction of the activities was not convenient and therefore excluded.

The classifier has been trained on all the data with a Linear SVM classifier and its result can be found in Figure 11. This Figure does not yet include the AL algorithm. The purple line shows the subsequent decision boundary, which will be seen as an "ideal" decision boundary, to compare the performance of the AL algorithm to.

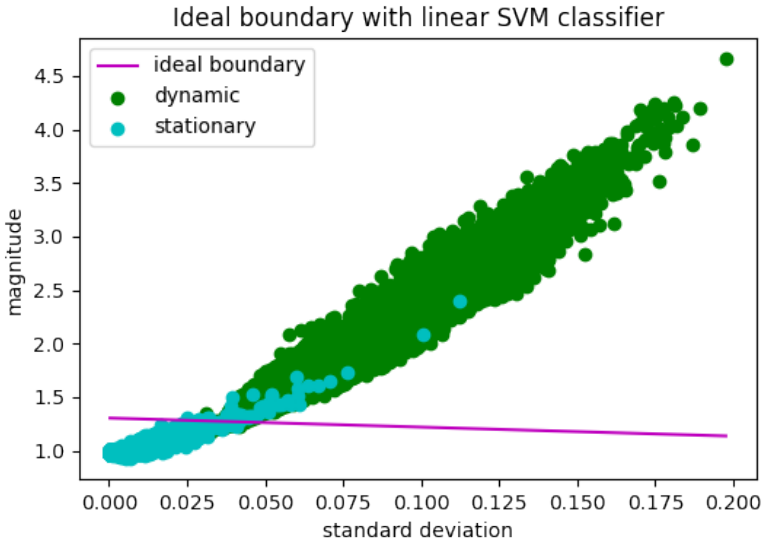


Figure 11: linear SVM on dynamic vs stationary activity

Now we have a baseline for the optimal performance, the AL algorithm can train. The data set is split into two, dividing pool data and test data in a 80/20 divide. The training data set is then created from the first 10 instances of the pool data set and iterated 2 times and then again with 2 instances and iterated 10 times.

To decide which instances to query, a type of uncertainty sampling is used, namely a least certain algorithm. This instance is shown in the figure with the yellow star, as shown in Figures **??** and 15. This point is found in the rest of the pool data set with unlabelled data and then added to the training data set. It then trains the boundary again. This process of finding, adding and training is done iteratively. After each iteration, a new decision boundary is estimated based on the newest information, which is visualised with the green dotted line. To compare the accuracy of the AL decision boundary, the purple ideal boundary found by the linear SVM line is still visualised too.

Firstly, the AL algorithm will do 2 iterations with an initial training set of 10 data points, linked to a random state of 1 for reproduction for the second experiment, as shown in Figure 12. This visualises step by step how beneficial asking a specific label can be. This means in total, it will have used 12 labelled data points to classify. Secondly, the AL algorithm will do 10 iterations with an initial training set of 2 data points, again using 12 labelled data points. Some iterations are shown in Figure 15, the rest can be found in Appendix B.

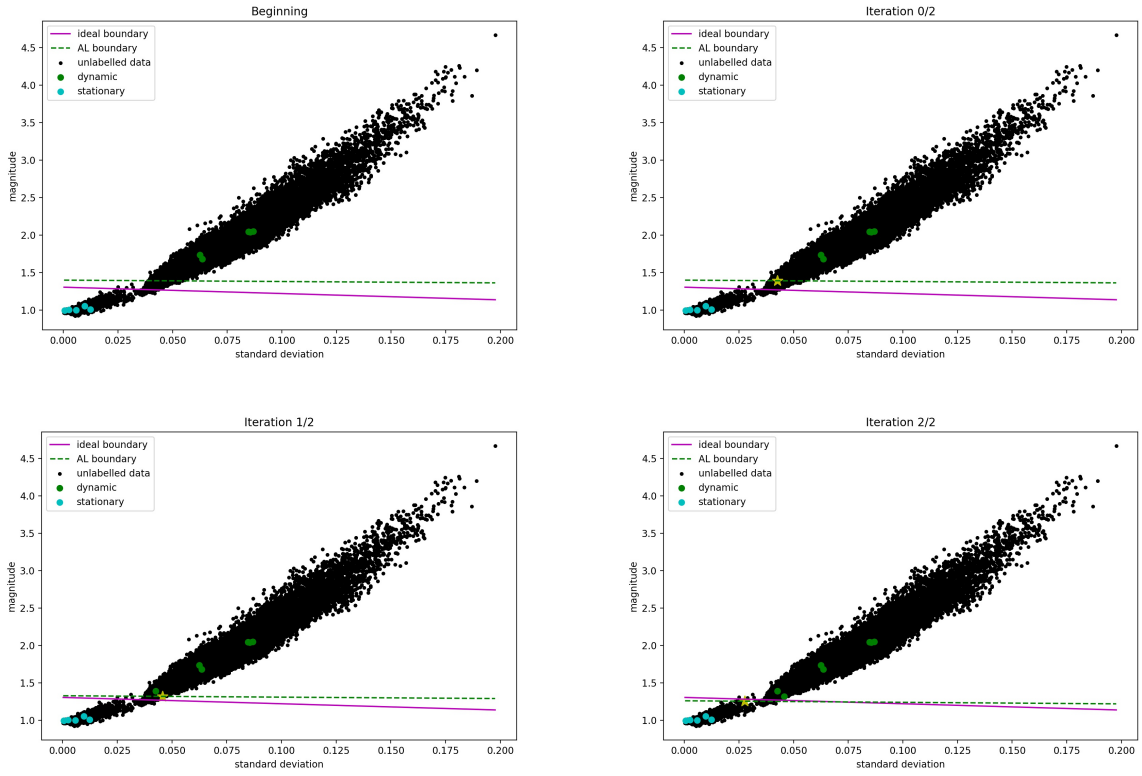


Figure 12: AL with 2 iterations and training set of 10 with linear SVM classifier

While both use the same amount of data points and get good results, there is a clear difference in how consistently the algorithm works. With more initial samples, it goes to the approximate correct classification quicker, as it better knows where to look. However, with less initial samples but more iterations, the correct boundary can be found more accurately, as the specific line will eventually be found by asking for more of the ambiguous labels. How fast this goes hugely depends on which initial samples are chosen, as they might include outliers and then the AL algorithm will need many more iterations to establish a boundary.

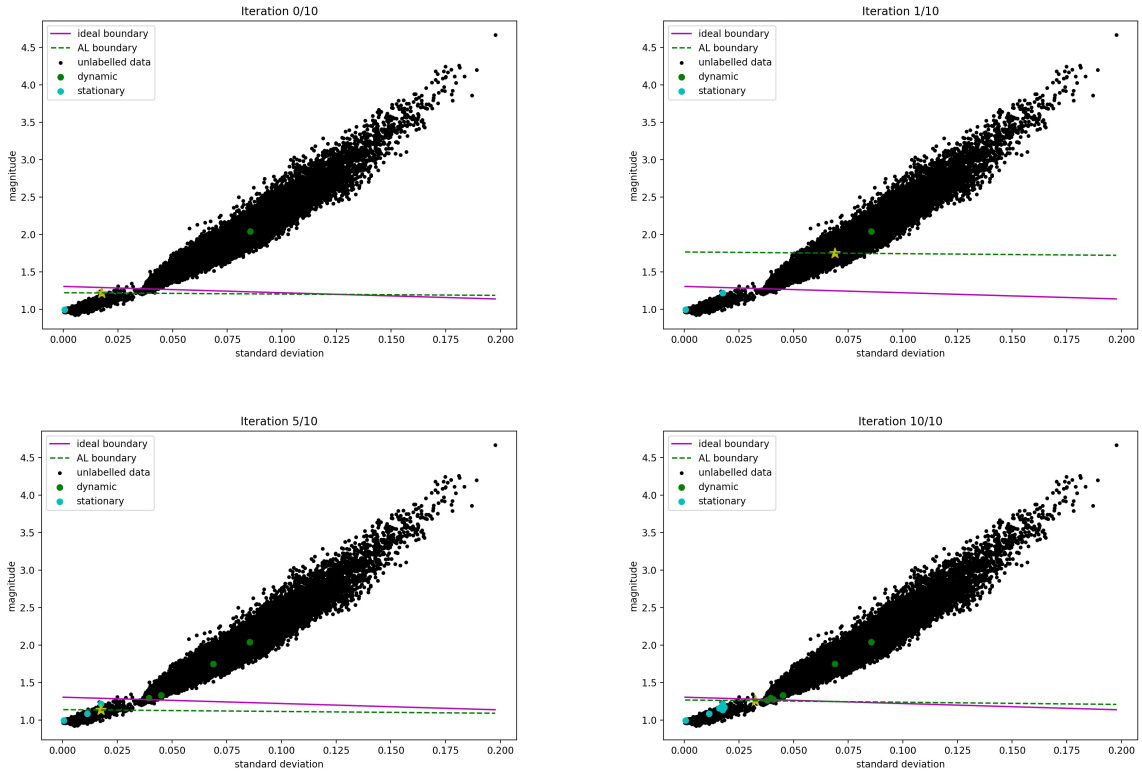


Figure 13: AL with 10 iterations and training set of 2 with linear SVM classifier

6.2 DNN without AL

The DNN trains with a set amount of labelled data points, but the amount of epochs can be varied. The training process also stops the epochs once the accuracy goes down again. After trial and error, there was found that this stop was at a maximum of around 20 epochs. Therefore, 25 epochs were chosen to get the best result from the DNN for every horse. The testing was done for each horse, so four times, and the average of this was used for the results and confusion matrix.

Results The first horse Galoway only needed 9 epochs, after which it stopped because the accuracy went down again. The horse Patron only needed 7 epochs, horse Happy 14 epochs and horse Driekus 6 epochs. This gave a F1-score of 0.774 and a MCC of 0.723. This shows the pre-

diction is quite good.

Confusion Matrix The resulting confusion matrix shows good results for most activities, as shown in Figure 14. Walking-rider was identified the best and only walking-natural was predicted very badly. Grazing and walking-rider was most often confused with each other.

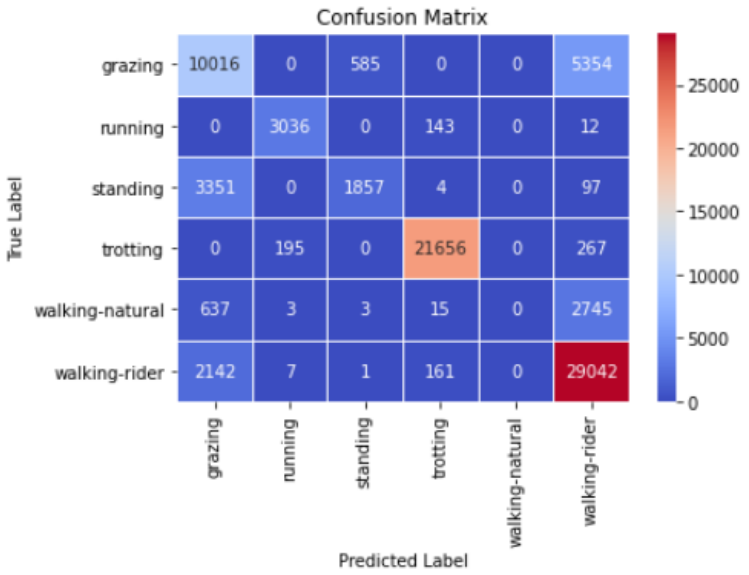


Figure 14: Confusion Matrix DNN without AL

6.3 AL on DNN

After visualising active learning on a linear SVM classifier with different initial training set sizes and number of iterations, active learning was applied to the DNN pipeline which was described in the methodology. First the DNN was tested without AL and then the algorithms least confident, margin, uncertainty entropy, consensus entropy and maximum disagreement were applied. The package used for all algorithms was modAL [46].

6.3.1 AL Variables

To be able to compare the effects the initial training set size has and to find the number of iterations needed to get the full potential of AL, trial and error was used at first. This was based on when AL still has affect, while also considering the aim is to use the least amount of labels possible. Two experiments are done per algorithm and four graphs have been developed to most informatively visualise the results. Two use the MCC value and the other two use the F1-score, while comparing small and big initial training set sizes. The iterations are the average of the four horses, each time testing on a different horse and training the others. The initial training set uses random sampling, therefore the beginning point at iteration 0 can be seen as the performance of the classifier without AL. Up from there shows the increase in performance by the AL algorithms. Additionally, a confusion matrix is used to depict how well classification works per activity.

The first experiment focuses on the low initial training set sizes. As was found by trial and error, the best performance was depicted in the first 40 iterations. It differed greatly per algorithm which iteration should be chosen, but all showed no significant increase in MCC or F1-score after 40 iterations. The initial training set sizes were chosen at 10, 20, 30 and 50. This shows the absolute minimum training set size which can be considered to get meaningful results.

The second experiment focuses on the higher initial training set sizes. As could be concluded from literature /sources/, the SVM classifier and some trial and error, indeed much less iterations were needed to get to most out of it. The rapid increase stopped at around 8 iterations. However, while the smaller initial training set sizes did not go up anymore, the bigger ones did. First gradually, but then also significantly. This was between iteration 30 to 50, dependent on the algorithm and training set size. Therefore, 50 iterations were chosen to depict the effect. The initial training set sizes that showed significant differences between each other, yet are still relatively small compared to the data set, were at 80, 150, 250, 350. The highest ones already give results that are closer together and above 350 it does not give significantly better results.

The confusion matrix was created for all horses at an initial training set size of 50 and 350, using in total 90 and 400 labelled data points. This shows how well labels were predicted on average for the horses after iterating.

6.3.2 Least Confident

Description This method looks at the probability (prediction) of each point. Then it subtracts (1 - prediction), to find the lowest prediction and therefore the highest uncertainty. This was done with the modAL package classifier_uncertainty.

Comparing initial training set sizes The smaller initial set sizes were not very consistent when reaching more iterations, however, in the end they do go up. The initial F1-score and MCC were low and they needed many iterations to achieve good results. As can be seen in Figure 15, consistency becomes higher with a bigger initial training set. Here, they stop learning as rapidly at iteration 25.

Interestingly, for the smallest three initial training set sizes, the F1-score starts of at around 0.22, but they go down until iteration 10, from where it goes up again, even though it does pick the most informative instance. The MCC also shows a decrease at first, illustrating the prediction value indeed gets worse and even worse than random. After iteration 10, both F1-score and MCC go up again.

Furthermore, there can be seen that the bigger the initial training set, the quicker it learns per iteration and the higher the total accuracy can become. There is a rapid increase in both F1-score and MCC up and until iteration 10. From here, the F1-score and MCC continue to increase with approximately 0.1 for another 20 iterations. The highest two, so 250 and 350 samples, are much less consistent and increase from iteration 20 on. If they query a convenient informative one, its values increase, but if it causes confusion, it decreases a bit. This variance between iteration 20 and 50 can make the difference of approximately 0.1 MCC, as can be seen in Figure 15.

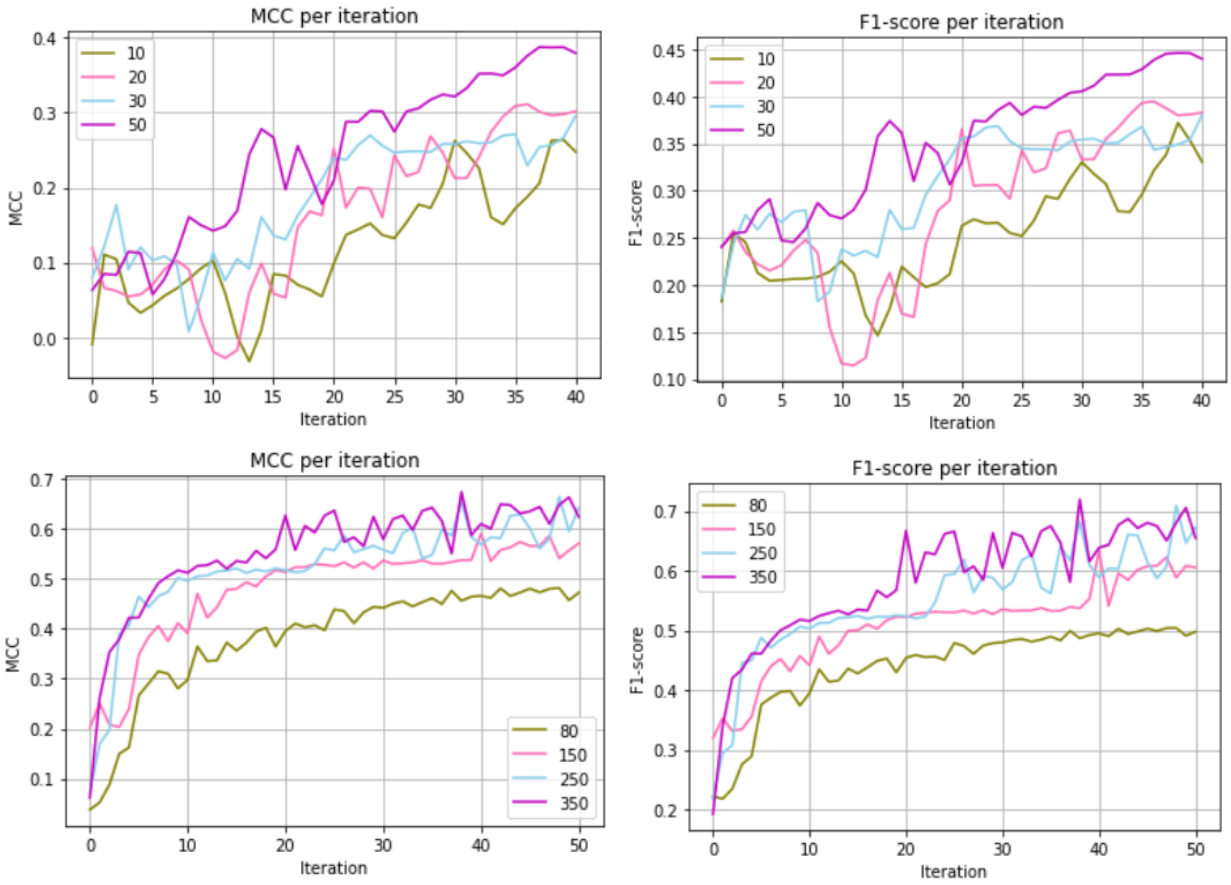


Figure 15: least confident MCC and F1 per iteration, for small and big initial training set sizes of 10, 20, 30, 50 and 80, 150, 250, 350

Confusion matrix The confusion matrices show clear differences, as shown in Figure 16. The first confusion matrix, using 90 labelled instances, most often predicted the sample to be in the walking-rider class, which is also the biggest. However, it still confuses this with many other activities, mainly grazing and trotting. Trotting was also identified reasonably well, but as mentioned was most confused with walking-rider.

The second confusion matrix, using 400 labelled instances, shows it preformed much better. The activity trotting was identified quite well, with relatively little confusion. The activities grazing and walking-rider were also classified well, although they were still often confused with others.

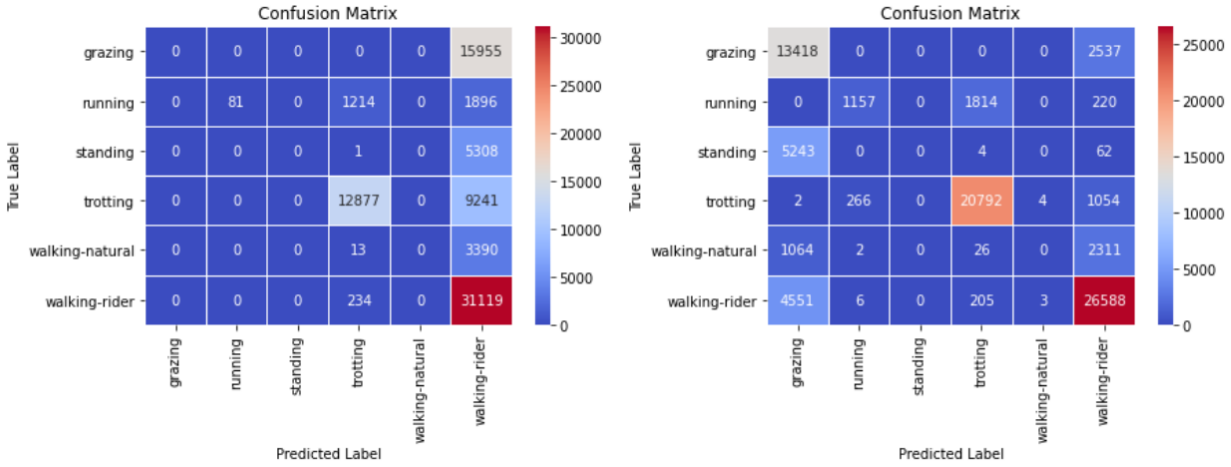


Figure 16: Least Confident confusion matrices using 90 and 400 labelled instances

6.3.3 Margin of Confident

Description This method considers the difference in the probabilities of the two highest predictions. This margin is then used to establish the smallest margin and this is used to find the most ambiguous data point. The modAL package which was used was: classifier_margin

Comparing initial training set sizes The results, as shown in Figure 17 show that for the lower initial training set sizes, MCC and F1-score are low and inconsistent. The difference per size is still significant, where even labelling 10 points gives a F1-score of only 0.15 and 20 already gives 0.35 after 40 iterations. The F1-score did again decrease for the three smallest, this time even to a score of almost 0 F1-score and below 0 MCC till iteration 20, but then goes up again.

The higher training set sizes were as expected, where F1-score and MCC go up rapidly until iteration 8. From iteration 8 till iteration 20 it still increases with 0.1. From iteration 20 to 50, it is less stable again. The biggest initial training set size of 350 gave the best results again, with it being reasonably consistent around 0.64 for the MCC and 0.68 for the F1-score.

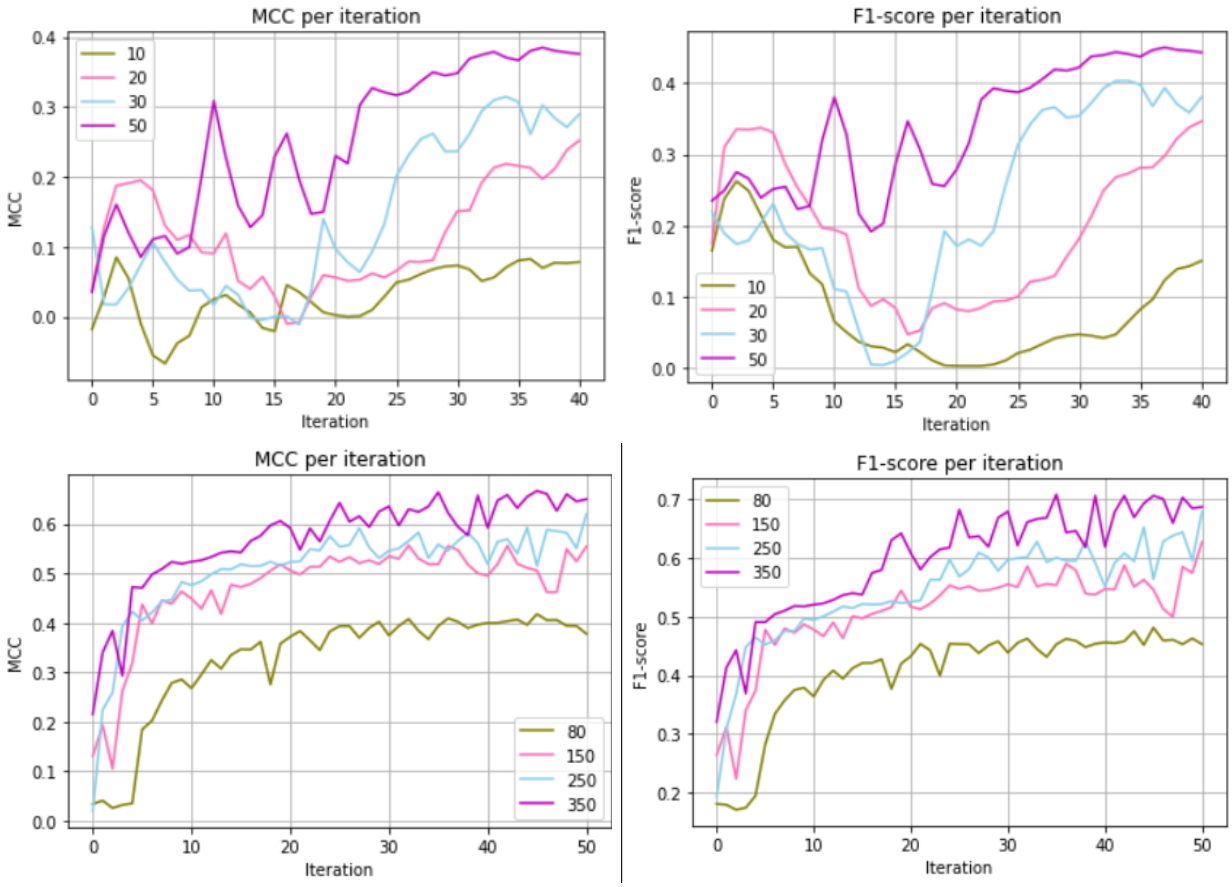


Figure 17: Margin of Confidence MCC and F1 per iteration, for small and big initial training set sizes of 10, 20, 30, 50 and 80, 150, 250, 350

Confusion matrix These confusion matrices look similar to those of the least confident, as shown in Figure 18. Again, for the confusion matrix which used 90 labelled instances, walking-rider was best predicted, yet also gave the most inaccuracy. Trotting still had some good results, but was also confused with walking-rider. However, many more activities were seen and considered and also others beside walking-rider were correctly identified.

The confusion matrix which used 400 labelled instances was again much more accurate in its predictions. Walking-rider was mostly correct, or confused with grazing. Trotting and grazing were often classified correctly too.

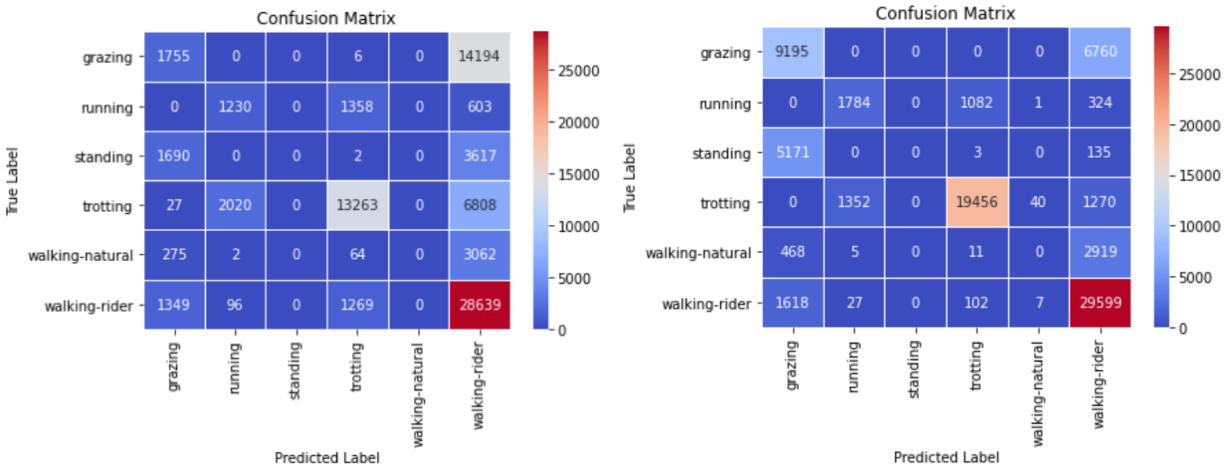


Figure 18: Margin of Confidence confusion matrices for 90 and 400 labelled instances

6.3.4 Uncertainty Entropy

Description This method focuses on the entropy of the class probabilities. This is a measure of how random data is, so the higher the entropy, the more ambiguous the instance. The modAL package used was: classifier_entropy.

Comparing initial training set sizes The results are again very similar, as shown in Figure 19. The three smallest initial training set sizes go down to a F1-score of zero. The smallest size of 10 even goes below 0 MCC, showing it is worse than random prediction. The other two lowest, those at 20 and 30, again first go down, but then up again and give a F1-score of 0.35. The last of the lowest, that of 50, does not first go down. However, it is very inconsistent with many peaks up and down. At iteration 20, all 4 go up and at iteration 32 reach a point of stability.

The higher training set sizes are also similar. All go up rapidly until iteration 9 and reach stability, with more variations, around iteration 20. Again, the biggest initial training size of 350 gives the best results, with a MCC of 0.64 and a F1-score of 0.69.

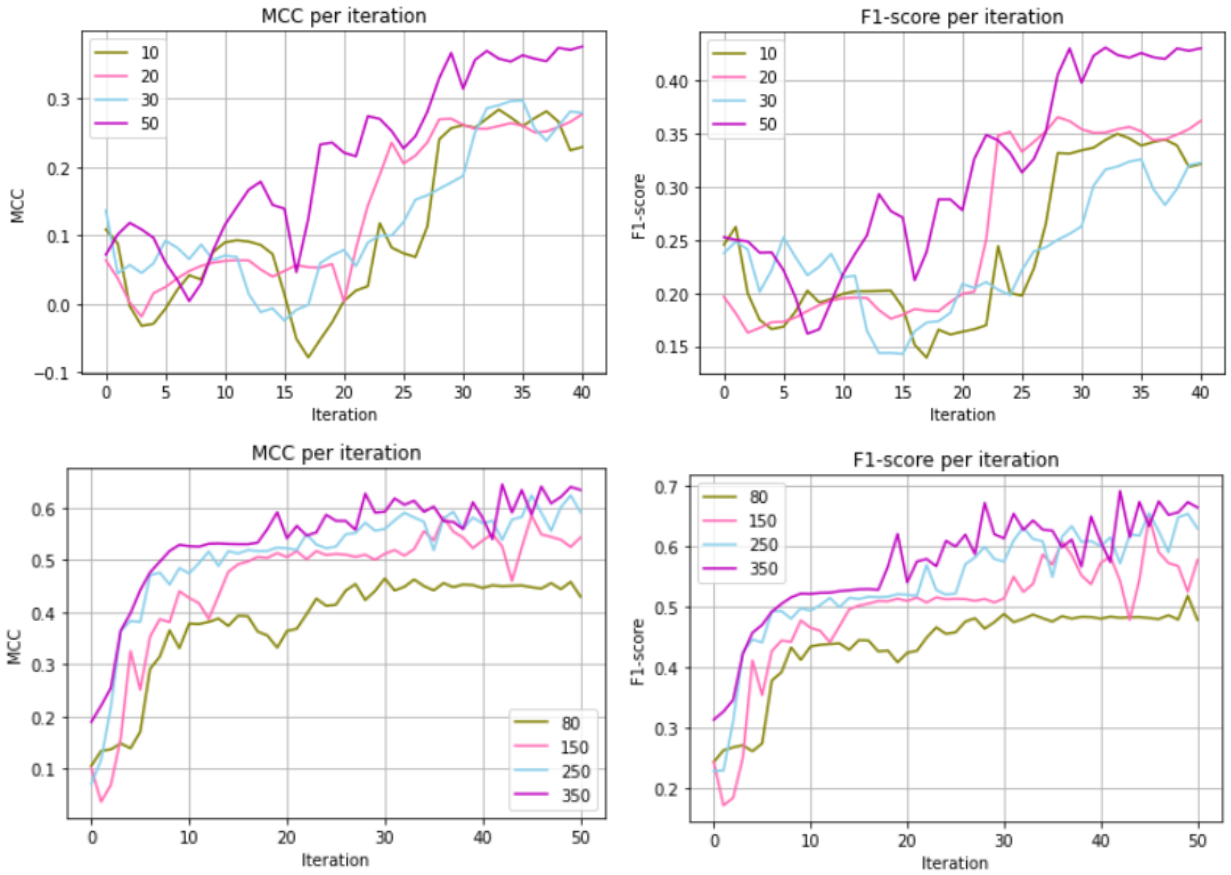


Figure 19: Entropy MCC and F1 per iteration, for small and big initial training set sizes of 10, 20, 30, 50 and 80, 150, 250, 350

Confusion matrix Both confusion matrices are very similar to the least certain algorithm. The walking-rider activity was classified correctly the most times, but for the lowest training set of 90, it was also most confused. The activity trotting was also classified well, but had less confusion with walking-rider than the other algorithms. For the training set of 400, grazing was often also classified correctly and the other activities were classified right sometimes, although often poorly.

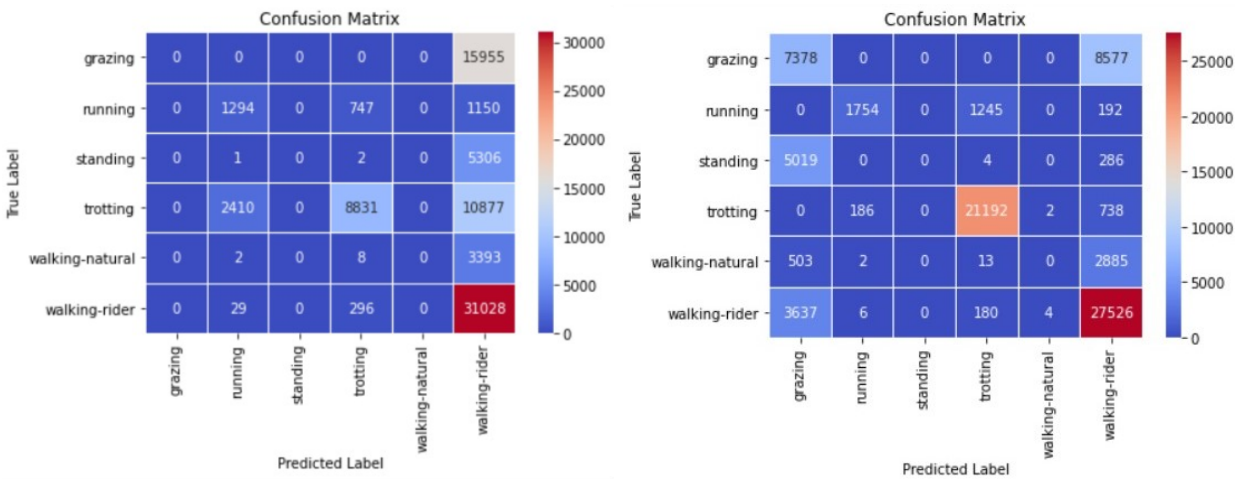


Figure 20: Uncertainty entropy confusion matrices for 90 and 400 labelled instances

6.3.5 Consensus Entropy

Description This method is part of disagreement based sampling. It first considers the averages of the class probabilities per classifier, from which the entropy is calculated. This largest entropy is selected as having the most disagreement. The method uses the modAL package consensus_entropy_sampling.

Comparing initial training set sizes The results showed something similar to that for uncertainty sampling. Again, the higher the initial training set, the better the performance. For the smaller sizes, the F1-score and MCC did go down a little again and at around iteration 20 it went up again. At iteration 30, it does not go up as rapidly and is reasonably stable.

The bigger initial training set sizes were quite stable. It went up rapidly till around iteration 9. From iteration 9 to 25, this is quite consistent, but from there it is less stable and has some peaks and troughs.

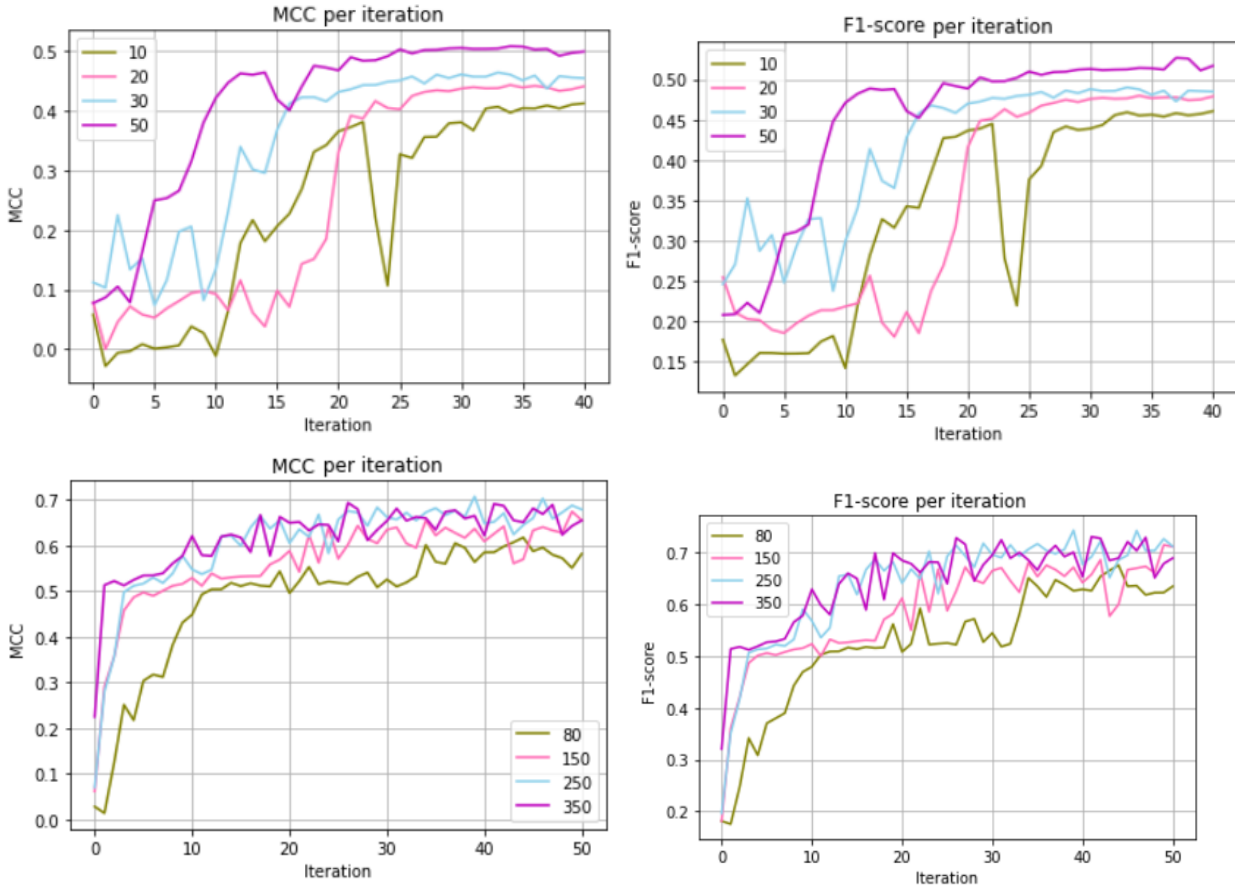


Figure 21: Consensus entropy MCC and F1 per iteration, for small and big initial training set sizes of 10, 20, 30, 50 and 80, 150, 250, 350

Confusion matrix The confusion matrix with the lowest amount of labelled instances used, which is 90, had decent results. It could differentiate the trotting quite well from the others and walking-rider even more often than for the 400 labelled instances used. However, every class did have more confusion for the lower one and walking-natural and standing was never identified correctly. The higher one did identify standing sometimes and running and grazing even a lot more times.

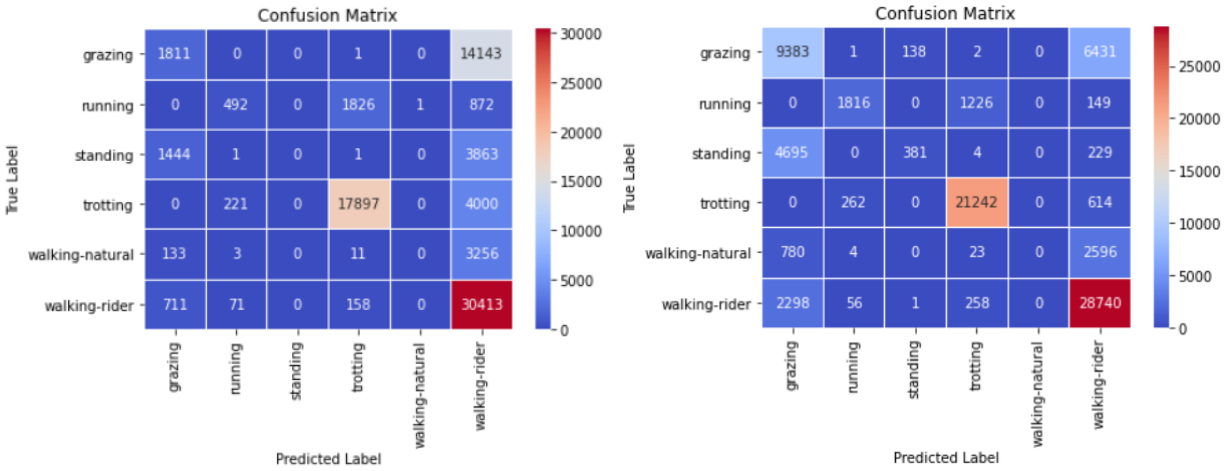


Figure 22: Consensus entropy confusion matrix for 90 and 400 labelled instances

6.3.6 Maximum Disagreement

Description Unlike the other methods, this method does not just pick the ones all classifiers disagree on the most, but considers the instances that have the largest disagreement. This method uses the Kullback-Leibler divergence per classifier, to measure how the probabilities differ per classifier. The modAL package max_disagreement_sampling is used.

Comparing initial training set sizes The results showed that it took a little longer to get to a stable point and stay consistent for this algorithm. The lower initial training set sizes went down again, but after iteration 5 already went up. This increase did take quite long, with a increase till iteration 30. Even from there, it was not very stable.

The higher initial training set sizes look very similar to the consensus entropy. It rapidly goes up until iteration 9, then finds a stable point and hangs around this til iteration 20. From here there are more peaks and troughs, which decreased with the higher training set size.

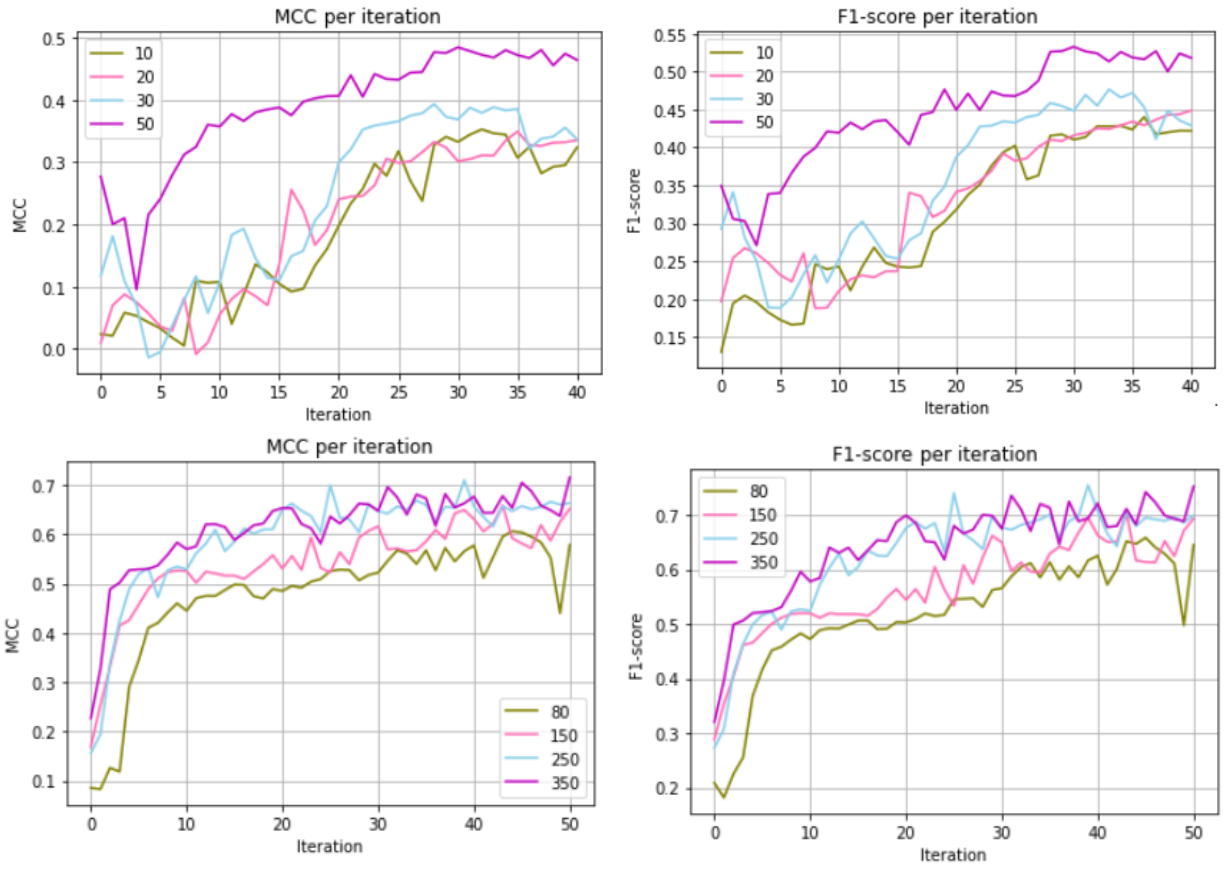


Figure 23: Maximum disagreement MCC and F1 per iteration, for small and big initial training set sizes of 10, 20, 30, 50 and 80, 150, 250, 350

Confusion matrix These two confusion matrices show quite good results. The smallest training set size of 90 could even differentiate walking natural a few times, which was barely done before. However, it could not find the standing activity and all activities were confused with walking-rider many times. The training set size of 400 showed good results too. All activities had some reasonable results, except for walking-natural, which was confused with walking-rider a few times and only labelled correctly 18 times.

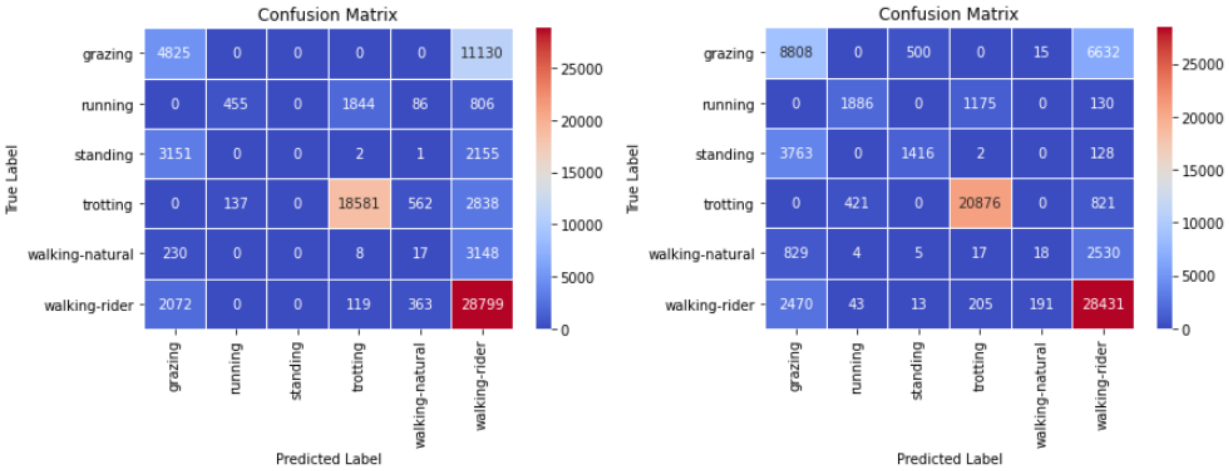


Figure 24: Maximum disagreement confusion matrix for 90 and 400 labelled instances

7 Evaluation

To evaluate which algorithm performs best, the algorithms are compared to each other within their division of uncertainty or disagreement. This is done by showing the differences in the number of instances that need to be queried for AL to be efficient and the size of the initial training set. Additionally, the best algorithms are chosen for each sampling type and are compared. Furthermore, the AL algorithms on the DNN are compared to the DNN without AL. Lastly, all algorithms are compared to manual annotation to show how time-efficient AL is.

7.1 Comparing Algorithms Per Sampling Type

The first experiment used an initial training set size of 10, 20, 30 and 50 data instances and iterated 40 times. The second experiment used an initial training set of 80, 150, 250 and 350 and iterated 50 times. All three algorithms were tested and depicted against their F1-score and MCC. To visualise a comparison between the algorithms and show the effect of AL, and how it varies with the initial training size, the initial training set sizes of 50 and 350 are used to compare the various algorithms, as shown in Figure 25 and Figure 26. Both are iterated 50 times.

Uncertainty sampling The three uncertainty algorithms overlap hugely and with an increase in iteration, all become more consistent and overlap more, both for the smaller and bigger initial training set sizes. The initial training based on 0 iterations is around 0.1 MCC and 0.22 F1-score for all algorithms for both sizes. This reaches up to 0.4 MCC for the initial training set size of 50 and an F1-score of 0.45 after 50 iterations. The initial training set size of 350 goes up to 0.64 MCC and an F1-score of approximately 0.65. The exact results can be found in Table 4 and 5.

For the low initial training set size of 50, when the iterations are low, the algorithms overlap less. There can be seen in Figure 25 that all algorithms have peaks going up and down at first and become more consistent around iteration 20. From there on, it reaches stability around iteration 40. The least certain algorithm does have some extra peaks, but on average does give the same result.

The algorithms of the higher initial training set size of 350 overlap even more. The results are very similar, where all algorithms go up in F1-score and MCC. These rapidly increase and are reasonably consistent until iteration 9. From there on, all algorithms are very inconsistent, but stays around a MCC score of 0.6 and F1-score of 0.63. What can be seen, is that the margin algorithm is the least consistent, as it goes up and down in little peaks, also before iteration 9. Entropy seems the most consistent at the lower iterations. However, the least certain algorithm seems to be most consistently stable at the higher iterations.

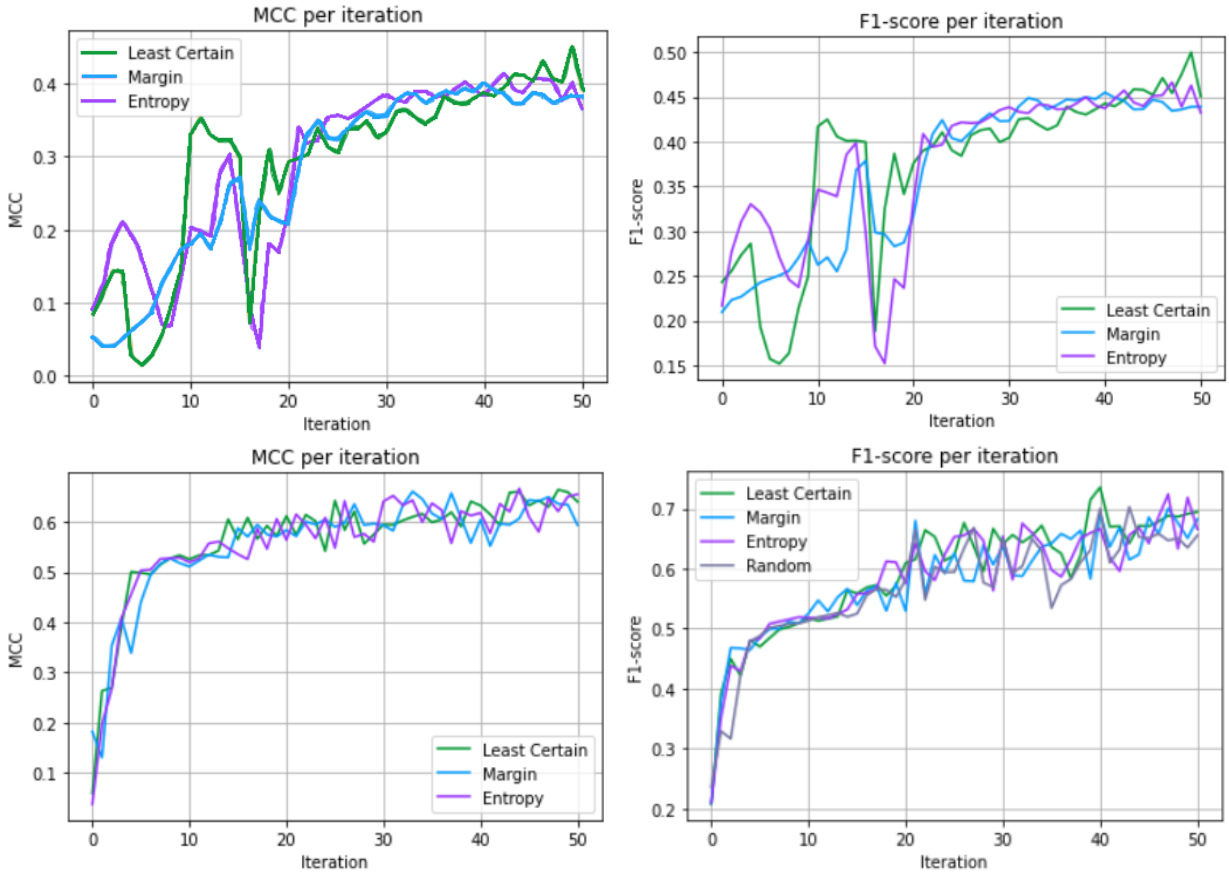


Figure 25: Least Certain vs Margin vs Entropy, for small and big initial training set sizes of 50 and 350

Disagreement based sampling The two disagreement based sampling overlapped even more, especially for the low initial training set size, as shown in Table 4 and 5. The differences are small and the results very comparable. Both got to a MCC of 0.5 and F1-score of 0.55 for the small training set size and a MCC of 0.65 and F1-score of 0.7 for the big training set size. There is a small difference which can be found, which is that the maximum disagreement algorithm is more slightly more consistent. Additionally, both algorithms start at a F1-score of 0.2 and MCC of 0.05 for the low initial training set size. However, the bigger initial training set size was at a F1-score of 0.2 and MCC of 0.1 for consensus entropy and F1-score of 0.35 and MCC of 0.28 for maximum disagreement. This also means that the maximum disagreement has to rise much less per iteration to get to the same performance.

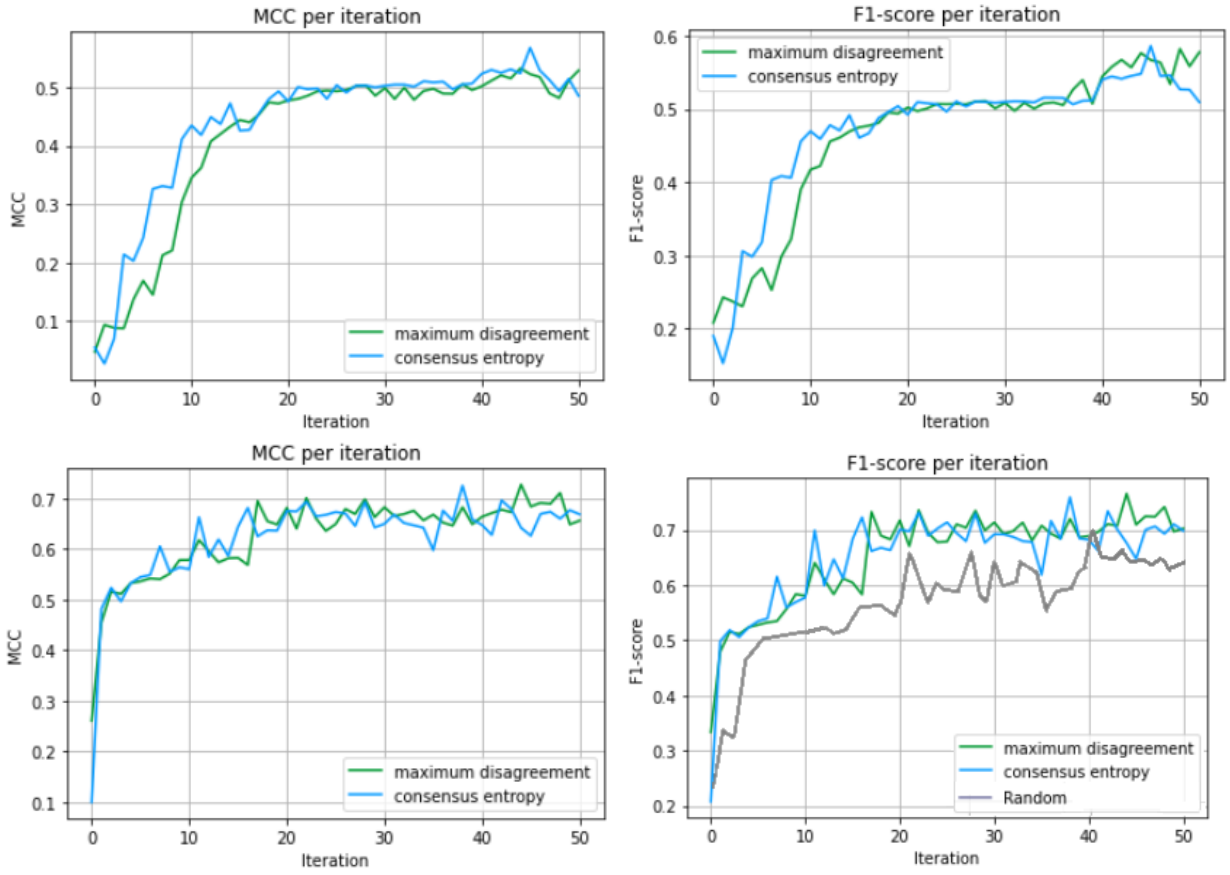


Figure 26: Maximum Disagreement vs Consensus Entropy, for small and big initial training set sizes of 50 and 350

Table 4: F1 of algorithms per iteration with initial training set size 50 and 350

Algorithm	**0**	**2**	**4**	**6**	**8**	**10**	**15**	**20**	**25**	**30**	**40**	**49**	**50**
Least Confidence (50)	0.242	0.273	0.193	0.152	0.214	0.417	0.400	0.376	0.384	0.404	0.443	0.450	0.450
Margin of Confidence (50)	0.209	0.227	0.243	0.250	0.270	0.262	0.379	0.317	0.400	0.423	0.455	0.439	0.439
Uncertainty Entropy (50)	0.217	0.311	0.321	0.271	0.237	0.346	0.300	0.334	0.422	0.439	0.436	0.463	0.432
Consensus Entropy (50)	0.190	0.202	0.298	0.403	0.406	0.470	0.461	0.492	0.510	0.510	0.540	0.526	0.509
Maximum Disagreement (50)	0.208	0.237	0.269	0.253	0.322	0.417	0.469	0.494	0.510	0.508	0.544	0.558	0.578
Least Confidence (350)	0.224	0.340	0.508	0.503	0.520	0.519	0.574	0.631	0.672	0.626	0.666	0.696	0.676
Margin of Confidence (350)	0.309	0.422	0.383	0.503	0.520	0.513	0.608	0.614	0.630	0.646	0.644	0.678	0.630
Uncertainty Entropy (350)	0.177	0.359	0.482	0.510	0.520	0.517	0.531	0.568	0.555	0.669	0.650	0.677	0.699
Consensus Entropy (350)	0.207	0.518	0.523	0.539	0.600	0.699	0.722	0.731	0.694	0.693	0.682	0.710	0.698
Maximum Disagreement (350)	0.333	0.515	0.524	0.532	0.557	0.580	0.583	0.736	0.685	0.714	0.689	0.697	0.703

Table 5: MCC of algorithms per iteration with initial training set size 50 and 350

Algorithm	**0**	**2**	**4**	**6**	**8**	**10**	**15**	**20**	**25**	**30**	**40**	**49**	**50**
Least Confidence (50)	0.084	0.144	0.027	0.028	0.095	0.329	0.298	0.293	0.306	0.333	0.388	0.450	0.392
Margin of Confidence (50)	0.054	0.041	0.063	0.088	0.150	0.209	0.218	0.350	0.361	0.384	0.372	0.382	0.381
Uncertainty Entropy (50)	0.093	0.182	0.193	0.121	0.068	0.203	0.204	0.224	0.357	0.384	0.383	0.402	0.364
Consensus Entropy (50)	0.055	0.070	0.203	0.326	0.328	0.436	0.462	0.475	0.503	0.502	0.523	0.514	0.485
Maximum Disagreement (50)	0.048	0.088	0.137	0.145	0.221	0.345	0.444	0.477	0.493	0.498	0.502	0.511	0.528
Least Confidence (350)	0.059	0.269	0.501	0.496	0.527	0.525	0.565	0.612	0.642	0.595	0.633	0.659	0.640
Margin of Confidence (350)	0.182	0.353	0.339	0.494	0.526	0.511	0.588	0.584	0.590	0.593	0.604	0.634	0.593
Uncertainty Entropy (350)	0.038	0.267	0.456	0.505	0.527	0.520	0.536	0.564	0.548	0.641	0.618	0.651	0.655
Consensus Entropy (350)	0.100	0.523	0.532	0.549	0.555	0.560	0.644	0.674	0.670	0.649	0.647	0.676	0.668
Maximum Disagreement (350)	0.261	0.516	0.532	0.542	0.551	0.578	0.582	0.681	0.679	0.669	0.672	0.648	0.656

7.2 Comparing Sampling Types

Additionally, the best uncertainty sampling type is the least certain algorithm, as this is most consistent throughout the iteration process, and the best disagreement based is maximum disagreement sampling type, as this is slightly more consistent in the higher iterations. A very clear difference is shown between the two, as shown in Figure 27, where the least confident algorithm has a substantially higher performance than the maximum disagreement throughout. However, at the biggest training set size, at around iteration 45, finally there is no difference anymore and every new iteration even overlaps exactly in MCC and almost exactly in F1-score.

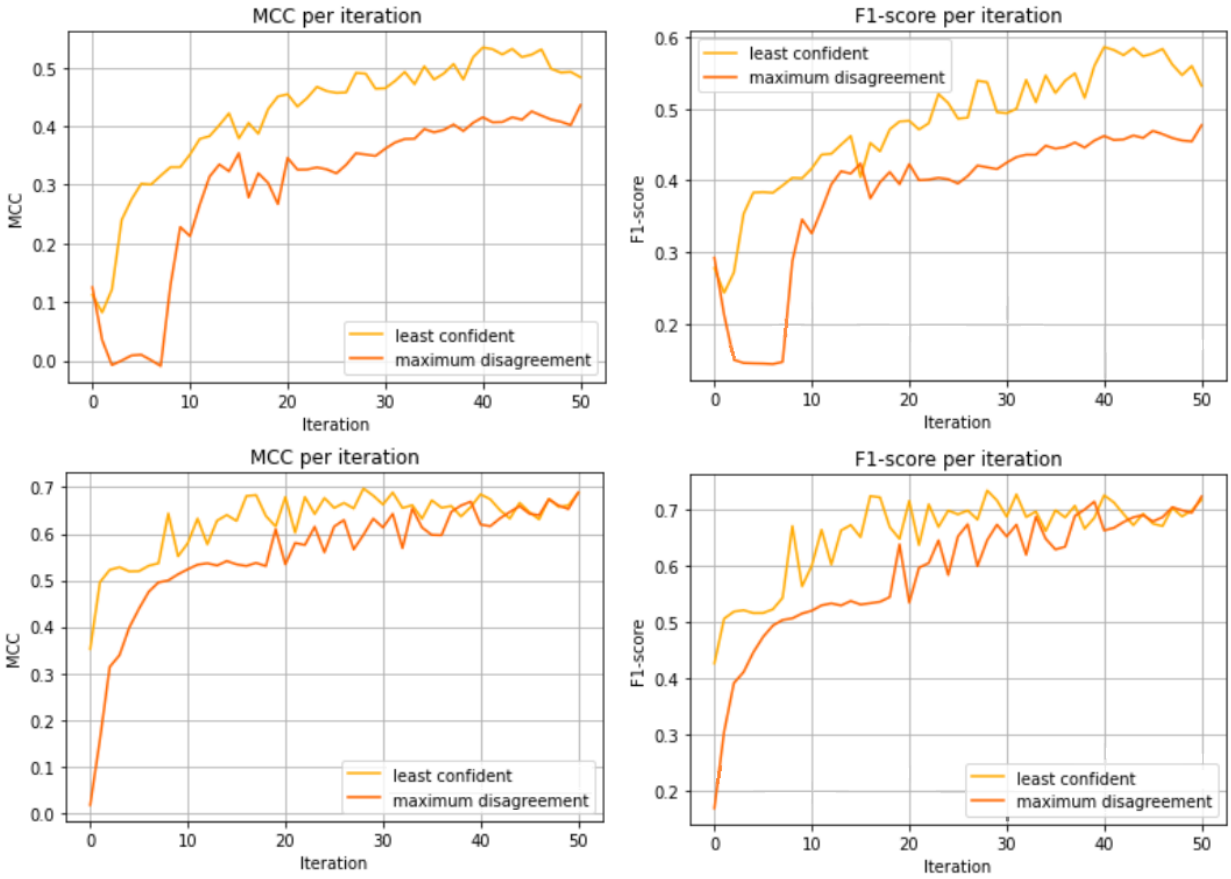


Figure 27: Maximum Disagreement vs Consensus Entropy vs Random Sampling, for small and big initial training set sizes of 50 and 350

7.3 Comparing To DNN Without AL

The results of the uncertainty and disagreement AL algorithm can be compared to the DNN training without AL. The DNN classifier is a type of supervised learning, where all of the labelled data is used to train the DNN. Where the AL algorithm splits the pool data into a labelled training subset and a rest unknown subset of unlabelled data, the DNN uses all of the pool data as training data. These are 81332 data points in this dataset. The AL algorithm can use variable X amount to train with. The DNN will be compared without and with AL on the best uncertainty sampling and disagreement based sampling algorithms, namely least confident and maximum disagreement.

The smaller initial training sets needed more iterations to get the results from AL. This was the same for all algorithms. For AL to be beneficial, it must reach a high consistency, as otherwise it could coincidentally be at a trough and subsequently have a low F1-score and MCC. Additionally, at a given point it reaches a point where the labelled instances are not substantially useful anymore. Therefore, the point where AL is stable and effective will be chosen to compare the AL algorithm with the DNN. The uncertainty sampling algorithm uses the DNN once and the disagreement based sampling twice. This is compared with a low and higher initial training set.

The first instance where a stability is reached with a sufficient performance will be used. This differs a bit per algorithm and initial training set size, as shown in Table 6.

Additionally, seeing as the benefit of AL is that is saves labelling time, it is also important to consider the run time of the algorithms. The run time will again be based on an average of the horses.

MCC, F1-score, Run time The MCC and F1 score can be compared with the labelled instances used and the three different uncertainty sampling algorithms and two different disagreement based sampling algorithms, as shown in Figure 6. Clearly, the MCC and F1-score is better for the DNN without AL. However, the MCC is only 0.062 better and the F1-score only 0.117 for the best uncertainty sampling algorithm at the highest number of iterations and a difference of a MCC of 0.029 and F1-score of 0.041 for the best disagreement based sampling algorithm. The run time was quite similar for all algorithms.

Table 6: Comparing DNN with(out) AL

Algorithm	**Initial training Set**	**Iterations**	**MCC**	**F1-score**	**Time (s)**
DNN with Least Confident	50	44	0.446	0.439	50.02
DNN with Margin of Confidence	50	29	0.361	0.449	50.76
DNN with Uncertainty Entropy	50	30	0.384	0.439	46.94
DNN with Consensus Entropy	50	18	0.479	0.496	44.60
DNN with Maximum Disagreement	50	18	0.474	0.495	45.20
DNN with Least Confident	350	43	0.658	0.685	51.59
DNN with Margin of Confidence	350	33	0.661	0.688	50.34
DNN with Entropy	350	42	0.636	0.657	51.43
DNN with Consensus Entropy	350	18	0.625	0.622	47.63
DNN with Maximum Disagreement	350	18	0.694	0.733	46.81
DNN without AL	81332		0.723	0.774	56.34

Confusion Matrix Comparison Lastly, there can be considered exactly how classification works and if this has significant differences in DNN without AL or with AL. As the confusion matrices were very similar for all five algorithms, only two algorithms were used to compare with the DNN without AL. The algorithm which was established as slightly more stable and with a better performance above, were the maximum disagreement and least certain algorithm. Therefore this confusion matrix, as seen in Figure 28, will be used to compare the DNN without AL to, as seen in Figure 29.

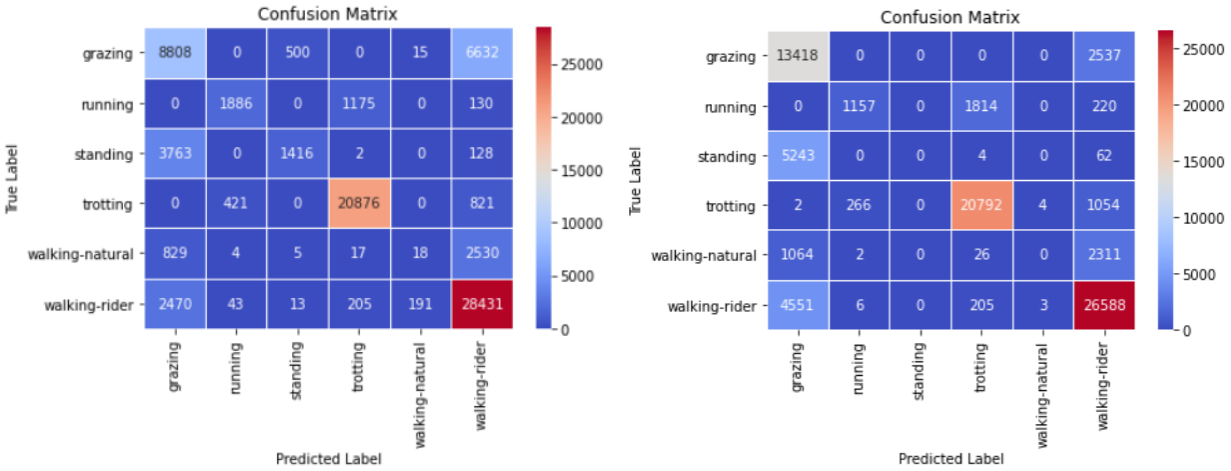


Figure 28: Maximum Disagreement and Least Confident confusion matrix using 400 labelled instances

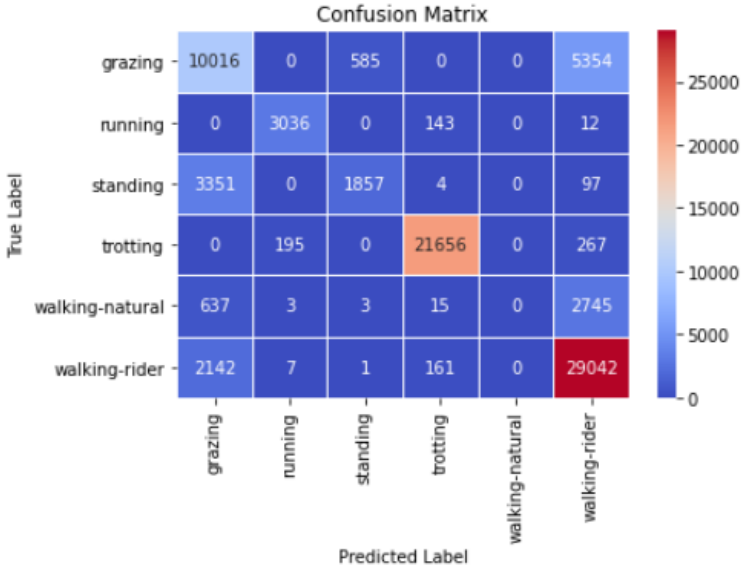


Figure 29: Confusion Matrix DNN without AL using 81332 labelled instances

The DNN confusion matrix using 81332 labelled instances is very similar to the confusion matrix of 400 labelled instances. The confusion matrix of the DNN shows that it has classified many more activities correctly than the AL algorithm. However, with an increase in correct classification, the confusion in other areas also increased, e.g. grazing was classified much more often than walking-rider. In general, the three confusion matrices do look rather similar in the sense of which activities it classified and how well, but the DNN without AL has simply labelled more. However, here a difference in uncertainty and disagreement based sampling can be found. The disagreement based sampling, using the maximum disagreement algorithm, differentiated all activities a little. The least confident was less good, with not once classifying standing and

walking-natural. However, the DNN without this never even tried to predict the walking-natural activity. The maximum disagreement therefore had predicted more different activities correctly, but least certain had less uncertainties across all activities. However, all three types were able to classify walking-rider, trotting and grazing quite well.

Compare to State of the art In literature, most research compared AL to random sampling [15], [14], [20]. The MCC shows that indeed, AL is better than random sampling. Active Learning can indeed be much quicker and show much better results, as was found in this report. This was the case for both disagreement based sampling and uncertainty sampling, but mainly for disagreement based sampling. For disagreement based sampling, there was found that results were extremely similar to uncertainty sampling. This was also the case in this report. However, uncertainty sampling was also compared to supervised sampling in literature, where Liu [31] found that AL showed a 4% to 5% higher accuracy than just supervised learning. Additionally, Stikic [32] found that co-training with uncertainty sampling even had an increase of 0.25 to 0.35 in accuracy over just supervised learning. This does not align with what was found in this research, where AL was slightly less efficient than supervised sampling. However, these compared accuracy, while the focus of this thesis was on F1-score and MCC, as this is more telling for activity recognition.

7.4 Manual Annotation vs Active Learning

The main benefit of AL can be found in the number of labelled instances which are used, which saves lots of time. To find the benefit of labelling with the AL algorithm, a comparison can be made with manual annotation. The benefits of using AL instead of manual annotation has already been established for other databases [47]. It depends greatly on the annotation time of a human, which differs per instance, and the type of data set.

With manual annotation, every instance needs to be labelled by hand. This takes approximately 3 to 10 minutes for sequential annotation tasks like this [48], with an uncertain rejection of 5.7%. Some research has been done specifically for animal activity recognition and showed that one minute of video took 3.7 minutes to label on average [48]. The time step of the data points used in this data set was at 1 second, with a 50% overlap. This gives segments of 2 seconds and the annotation time would therefore lie around 7.4 seconds on average.

The time it takes to run the algorithm can be found in Table 6. The least confident algorithm will be used to compare with, as this showed to be most consistent with best results. However, this does not yet include labelling the initial samples, which has to be added to the time. This is calculated in Table 7. The same annotation time is chosen as was found for a similar data set with IMU data and video. Therefore, the formula used is 7.4 s * size of total labels used. The manual annotation time of the data set which is used has been calculated to compare to. It is also compared to the DNN, which is the same as the manual annotation, yet also includes the learning process time, so additional similar data can be labelled in the future. The MCC and F1-score were calculated by averaging the horses, therefore the time the algorithm took will too.

Table 7: Labelling time

Type	**Number of labels**	**Labelling Time**	**Algorithm Run Time (s)**	**Total Time (s)**	**Certainty**
AL Uncertainty	94	695.6 s 11.58 min	50.02	745.62	0.446 MCC
AL Disagreement	68	503.2 s 8.4 min min	45.20	548.4	0.474 MCC
Largest AL Uncertainty	393	2908.2 s 48.47 min	51.59	2959.79	0.658 MCC
Largest AL Disagreement	368	2723.2 s 45.39 min	46.8	2770	0.694 MCC
DNN	81332	601856.8 s	56.43	601916.23	0.723 MCC
Manual	81332	601856.8 s 10030.95 min 167.18 h 6.97 full days	0	601856.8	94.3%

As can be seen, the smallest AL algorithm, which was at an initial training set size of 50, takes the least amount of time. However, the F1-score and MCC is quite low. The second best is the largest AL algorithm, which has an initial training set size of 350. This size gave much better results, namely a F1-score of 0.685 and MCC of 0.658 for uncertainty and 0. F1-score and MCC of 0.694 for disagreement. The human error was at 94.3%. This shows humans will still annotate labels better, but, this will save 601856.8 - 2959.79 = 598897.01 seconds compared to the least certain algorithm of 393 samples and 601856.8 - 2723.2 = 599133.6 seconds compared to the maximum disagreement algorithm of 368 samples. If this data set is labelled in a working week of 40 hours, this would have saved approximately 4 weeks of work with either of the AL algorithms.

Additionally, the average annotator earns around 3100 euro a month [49]. This data set is not very difficult, but annotators have to be trained as an accuracy in labelling is needed, so let's assume the pay grade will fall within the 25th percentile. This would be at 2231 euro monthly pay [49]. This algorithm still needed 48 mins to label the queried instances and samples form the initial training set. This would mean, approximately 2218 euro is already saved by using the AL algorithm over manual labeling.

8 Discussion

8.1 Comparing AL Variables

After applying and comparing all algorithms to each other, there could be found that the F1-score and MCC greatly depend on the size of the initial training set. After how many iterations the AL algorithm showed potential, also depended on this.

Small initial training set size For all five algorithm types, the three smallest initial training set sizes of 10, 20 and 30, first dipped in F1-score and MCC. This is because it has so many uncertainties, that when it takes an ambiguous point which is confusing for the classifier, another activity can become even more confusing. This is also due to it needing to classify 6 activities and it only identifies 1 ambiguous point at the time. At approximately 15 iterations, these scores go up again and become stable without a gain at around 35 iterations. The last smallest initial training set size which was tested, was at a size of 50. This still had lots of peaks, but didn't first dip to zero before climbing up. This algorithm also reached a point where it did not go up anymore in MCC or F1-score, which was around iteration 30. The MCC was around 0.39, therefore giving a result just showing AL has some potential, even when only using 90 labelled instances in total. This is not great for the 6 activities which were classified, but when classifying for instance 4, this could potentially give much better results.

The classification, as shown in the confusion matrices, showed that walking-rider was guessed most times during classification. This is due to the imbalance, where walking-rider has many more samples. If only a few random instances are selected for the training class, probably most, if not all, are from walking-rider. If the ambiguity sample is queried, most likely this will be from walking-rider. Therefore, walking-rider is most often correctly, but also incorrectly labelled.

When comparing these results to literature, there was be found that these results were slightly worse than in literature. However, other research compared accuracy, while this thesis looked at F1-score and MCC. Additionally, this research was done on a very different data set, not on one concerning the noisy IMU data with outliers and imbalance. Therefore, this comparison is difficult to make.

Big initial training set size The bigger initial training set sizes were at 80, 150, 250 and 350. Both MCC and F1-score for all algorithm types were better for bigger sizes and the bigger, the better the performance. The highest of 350 reach a maximum F1-score of somewhere between 0.65 and 0.70 and a maximum MCC of approximately 0.63. This shows that the biggest initial training set size of 350 indeed gives the best results, which was at around 35 iterations. This uses relatively more labelled instances in total, namely 385, however also shows a potential for AL. The number of labels is still relatively small, compared to the data set of 81332 number of data points and give similar results, although a bit lower.

8.2 Algorithm Analysis

After all algorithms were considered individually, they are compared to each other with a low initial training set size of 50 and a high initial training set size of 350. Both did 50 iterations.

8.2.1 Uncertainty Analysis

Similarities The similarities of the three algorithms was very big, whereas the results and comparison showed that they overlap in F1-score, MCC and the resulting confusion matrices. Also the general track it takes mostly overlap. This makes sense, as each time one point is taken, it learns a bit from it and in some way it is the most informative one. The exact data point might be similar, but the data set is so big that this point might be very similar to the point from the other algorithm.

The confusion matrices of the higher sampling size of 400, showed much overlap. Therefore, this means that activities are classified similarly.

Differences There are a few differences. The results of the low initial training set sizes overlap less for the different algorithms. They query different instances, which will have a bigger impact. This is due to the relatively big new addition that instance is to the training set. Furthermore, the margin algorithm is the least consistent and has the most peaks and troughs. The entropy algorithm is the most consistent at lower iterations and the least confident algorithm is most consistent at higher iterations. The latter is most important in AL, as this reaches the most consistently stable performance.

The confusion matrices showed that for the training set size of 90, there were some differences. The least confident algorithm, mainly classified all activities as walking-rider and a few as trotting. However, margin of confidence queried instances from all activities. This could be a coincidence, as this is just what the algorithm finds based on the random initial training set samples. This did mean, at lower initial training set sizes, margin of confidence does seem better at recognising various activities.

8.2.2 Disagreement Analysis

Similarities The algorithms were extremely similar and overlapped mostly. Seeing as both work with a DNN classifier and the classification is compared, this is a logical result. Both DNN's will probably agree, as the only difference would be the difference in the exact samples from the data set chosen for the initial training set. Additionally, the confusion matrices were very similar, showing that classification was also not affected massively by the type of algorithm.

Differences While the algorithms lay close together, some minor differences were found. The stability of maximum disagreement was a bit better than consensus entropy, therefore showing that in practice this algorithm would be good to use when simply applying it to a data set.

In the classification of specific activities, there is a big difference. The maximum disagreement algorithm was able to find more different activities for the training set sizes of 90 and 400. Especially the activity standing was classified much better by the maximum disagreement algorithm. This could again simply be a coincidence, as certain samples might be chosen which could further confuse or specifically help classification. However, this does show that maximum disagreement had the best results concerning classification and stability.

8.2.3 Uncertainty vs Disagreement Analysis

Overall, both the least confident and maximum disagreement algorithms had the same curve going up and at the same iteration stopped increasing as rapidly. This shows how big of an impact the size of the training set is compared to the type of algorithm. However, least confident was consistently much higher in lower iterations and for the lower training set size. This could be a coincidence, but this was the case for both the lower and higher training set sizes. It could be that maximum disagreement had a lower performance due to it using the same classifier to compare with, while least confident could look within the data set to find the most informative instance. If another classifier was used to compare with, the results of maximum disagreement could be vastly different, even if in principle the algorithm would work the same. At the end, both algorithms did come together at the same performance, so this did show that maximum disagreement does learn more per iteration, even if it does starts off lower.

However, it is hard to get to definite results, because all results depend greatly on which instances are chosen. It could be an unlucky addition or outlier, which throws the algorithm off. This can especially be seen in the low training set sizes, as the relative affect of an addition is more substantial. It showed that performance first went down with the first iterations, but later up again, once it had learned enough.

8.3 DNN without AL vs DNN with AL analysis

After comparing the AL algorithms on the DNN to each other, it was compared to the DNN without AL. Two initial training set sizes were chosen with different amount of iterations. There it can be seen that the DNN without AL still majorly outperforms the DNN with AL. The best AL results came from margin of confidence and maximum disagreement, although this was the least consistent and in practice works least well. The margin of confidence used 23 iterations and a total of 383 labels and has a MCC of 0.661 and a F1-score of 0.688. The maximum disagreement used 18 iterations and a total of368 labels. It has a MCC of 0.694 and F1-score of 0.733. The DNN has a MCC of 0.723 and F1-score of 0.774, but used 81332 labels. The difference in MCC and F1-score was low. However, this difference is still significant.

However, what must not be forgotten is the impact of the number of labels. After only using approximately 1/212 of the labels, it already got quite close in terms of MCC and F1-score and gave correct results. Even if this does mean DNN on its own is better, it shows the potential AL has for application where MCC and F1-score does not have to be high for all activities or if less activities are used.

8.4 Manual Annotation Results

Finally, the run time of the program was also taken into consideration. There was determined that the run time of the DNN with the AL only differed a maximum of 6.41 seconds. This is less time than it takes to manually label 1 sample on average. Therefore, this difference is not very significant. However, the difference in manually annotating every point and using the algorithm, was substantial. It was calculated for this very specific data set, which all contained labels, that it saved approximately a month on work and 2218 euros.

This assumed that one person would label all instances. However, annotators are often checked,

to ensure reliability. This also takes time. There is not one set way of doing this. The exact way this is done differs, e.g. depending on the data which has to be annotated. In literature, four methods are identified for finding the reliability of annotators of activity recognition and the method *Labeling instances of behavior with a frequency-based comparison* is most applicable to this data set and aim [50]. This paper works with a program which would optimise this process. There is a lot of research done on how long this process would take, all using various approaches and programs, but no agreement on a standard or minimum amount of time was established. However, this does mean that annotation would take even long than calculated in Table 7.

The results already seem very positive towards the potential of AL. However, as state of the art has found, this can even be improved by knowing some variables [47]. What was found, is that the cost of annotation should be taken into consideration, as this will not be the same for every data instance. This is found in how long the time instances take to be labelled, but also if the annotator is an expert or not and what their error is. If the AL algorithm can take this cost into account, the labelling process would be even quicker and more efficient.

9 Conclusion

The aim of this thesis was to find which Active Learning strategy is the quickest in converging to the most adept performance for Animal Activity Recognition when applied to an IMU horse data set. This was done by first applying AL to a linear SVM classifier and visualising the effect AL has when classifying two activities. Then, AL was applied to a deep neural network (DNN), which classified 6 activities. Three AL algorithms using uncertainty sampling were compared to each other, compared to the DNN without the AL algorithm and lastly, compared to manual annotation time.

The SVM classifier showed that the higher initial training set sizes needed less iterations for a better performance and was quite stable for each iteration. The lower initial training set sizes needed more iterations and gave less consistent results. However, they did have more potential to get to a high accuracy.

When comparing the three uncertainty sampling types, namely least confident, margin and entropy, it could be concluded that least confident sampling was the best for the bigger initial training set size and high number of iterations, while entropy gave the best result for lower iterations. However, all algorithms greatly depend on the exact instances which are used for the initial training set, because its size is so small. If it contains outliers or it is unlucky in finding the right instances, the results can be very different. Additionally, the two disagreement based sampling types, namely consensus entropy and maximum disagreement, could also be compared. While both are very similar, the maximum disagreement seemed better at differentiating between activities and was more stable in doing so. However, as both use the DNN classifier with different random sampling points, this could be a coincidence. Similarly for uncertainty sampling, having a specific instance in the training set can massively influence the results, e.g. by having outliers. However, the performance of all algorithms were still very similar. What had the most substantial impact for all, was that of the initial training set sizes. The bigger the initial training set, the less iterations there were needed for a good performance.

Moreover, when comparing the two sampling types, the least confident algorithm performs much better from the beginning, but after enough iterations, these start overlapping. However, again this could be a coincidence. Due to this and the results being very close together, no conclusive judgement can be made on the best algorithm for either sampling type. To take this uncertainty away, the experiment would need to be run many times with different instances being sampled each time. However, what was clear, is that both are much better and preferred over random sampling.

The results of comparing the DNN without and with AL were promising. The DNN without AL still outperformed the DNN with the AL algorithms, but at the highest initial training set size of 350, this difference was small. The performance can be helped by preprocessing the data more thoroughly, which should result in a higher MCC and F1-score. The amount of labelled instances needed with the AL algorithm was reduced substantially, saving a lot of time. Therefore, there could be concluded that while DNN without AL does slightly outperform DNN with AL, AL still has considerable potential. Compared to literature, the results are a little disappointing, but they cannot properly be compared to reach any definite conclusion.

Lastly, AL clearly showed a huge benefit over manual annotation. How much time annotation

would take and how much money this would cost, depends on the type of data which has to be labelled and how reliable this annotation should be. However, what can be concluded is that it can potentially save months of time and thousands of euros.

9.1 Recommendations

It can be concluded that even though AL shows potential, as it does clearly increase in MCC with each iteration if the training set is significantly high enough, the MCC was still low, at around 0.65 after using approximately 400 labelled instances in total. There are several factors which could cause this. Therefore, in further research, I would recommend some improvements. The most significant factor, is that of the number of activities which are to be classified. When classifying 2 activities, the results showed very positive results. However, when classifying 6 activities, these results were much lower. This is most likely because if it queries 1 instance at the time, it might need 20 iterations before it will query an instance from one of the smaller activity classes. Especially since the data set is imbalanced. Therefore, I would recommend using AL when there are less activities to be classified.

Additionally, it would be beneficial to further counter the imbalance present in the data set. Right now, this will give skewed results and a bias towards the activities with more data points. AL already helps this, but as this is the case for several of the activities, it needs to query quite a lot of instances before it can get significant results. This is especially useful since research often needs more information on those smaller activities. The AL will now ask for the bigger activities much more often and the smaller activities will be classified less well. To get better results, oversampling the smaller classes and undersampling the bigger ones is recommended. Additionally, outliers will be present, which will also potentially skew the results. If an outlier is the instance the AL queries, it will learn that a rare deviation is normal and will try to compensate all other classifications. Therefore, compensating for outliers would be beneficial to the results. Furthermore, the sensors attached to the horses were able to move. This means that the data may be noisy. While a low-pass Butterworth filter was already applied to try and compensate for this, this did not yield significant results. What can be considered is to use other methods to counter this.

Lastly, no unequivocal conclusion could be drawn about the type of algorithm which showed the best performance. This was because the results all lay so close together, yet its performance and specific instances that were queried, greatly depend on the initial instances that were chosen for the initial training set. Therefore, I would recommend running all experiments many times, to find if this impacts the behaviour of the algorithms and to see the effect of what the luck value of finding a particular value is.

9.2 Future Work

These recommendations should be addressed in future studies to give a more complete picture of the potential effect of AL. Currently, no definite conclusion could be drawn on the best AL algorithm. Therefore, guidelines should be set up to find the best AL algorithm for their particular dataset which fully consider all factors. These include the preprocessing steps, but also all limitations in the field which should be addressed in this consideration. This way, a combination of initial training set size and optimal number of iterations can be linked to the AL algorithms and

their performance for particular datasets.

Additionally, there are some other aspects within AL which were not applied, but do have potential in this area. The DNN classifier was used to find the potential of AL for AAR. However, other classifiers also have the potential to work well, e.g. a SVM classifier. To thoroughly find the best AL strategy on this data set, other classifiers should also be applied and compared. This is especially the case for disagreement based sampling, where a focus could be on finding the effect of different combinations of classifiers. The performance of this algorithm can in this way be optimised fully.

Furthermore, it was established that the number of classes has a big influence on the effectiveness of AL. To find what affect is has, more thorough research must be done to see how a different number of classes and the type of instances, e.g. overlapping or clear differences, within these classes affect the performance. Additionally, again the initial training set size must be considered here, as these could change for a different data set with different samples.

In addition, there are also other types of algorithms within both uncertainty sampling and disagreement based sampling which can be considered. To fully find the potential of AL, a more in depth look can be taken into the variations and the affect it has on different datasets.

Lastly, the advantage of AL over manual annotation was considered. However, there was established that performance is lower when using AL. More research should be done on how big this error can be to still have an significant performance and where this error arises. Subsequently, there can be considered how this error can be helped further.

10 References

[1] B. Settles, *Active Learning, Burr Settles*. 2013, p. 100, ISBN: 9781608457267.

[2] L. Wang, X. Hu, B. Yuan, and J. Lu, "Active learning via query synthesis and nearest neighbour search," *Neurocomputing*, vol. 147, no. 1, pp. 426–434, 2015, ISSN: 18728286. DOI: 10.1016/j.neucom.2014.06.042. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2014.06.042.

[3] S. Sabato and T. Hess, "Interactive algorithms: From pool to stream," *Journal of Machine Learning Research*, vol. 49, no. June, pp. 1419–1439, 2016, ISSN: 15337928.

[4] A. Kachites, "Employing EM and Pool-Based Active Learning for Text Classi cation 2 Probabilistic Framework for Text Classi cation," *Learning*, 1998.

[5] J. Luo and J. Huang, "Generative adversarial network: An overview," *Yi Qi Yi Biao Xue Bao/Chinese Journal of Scientific Instrument*, vol. 40, no. 3, pp. 74–84, 2019, ISSN: 02543087. DOI: 10.19650/j.cnki.cjsi.J1804413.

[6] R. Schumann and I. Rehbein, "Active learning via membership query synthesis for semi-supervised sentence classification," *CoNLL 2019 - 23rd Conference on Computational Natural Language Learning, Proceedings of the Conference*, pp. 472–481, 2019. DOI: 10.18653/v1/k19-1044.

[7] P. Awasthi, V. Feldman, and V. Kanade, "Learning using local membership queries," *Journal of Machine Learning Research*, vol. 30, no. iii, pp. 398–431, 2013, ISSN: 15337928.

[8] F. Laviolette, M. Marchand, and S. Shanian, "Selective sampling for classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5032 LNAI, pp. 191–202, 2008, ISSN: 03029743. DOI: 10.1007/978-3-540-68825-9{_}19.

[9] W. Di and M. M. Crawford, "View generation for multiview maximum disagreement based active learning for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 5 PART 2, pp. 1942–1954, 2012, ISSN: 01962892. DOI: 10.1109/TGRS.2011.2168566.

[10] O. Dekel, C. Gentile, and K. Sridharan, "Selective sampling and active learning from single and multiple teachers," *Journal of Machine Learning Research*, vol. 13, pp. 2655–2697, 2012, ISSN: 15324435.

[11] N. V. Cuong, W. S. Lee, and N. Ye, "Near-optimal adaptive pool-based active learning with general loss," *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference, UAI 2014*, pp. 122–131, 2014.

[12] A. Vaith, "Uncertainty based active learning with deep neural networks for inertial gait analysis," *FUSION*, vol. 23, p. 8, 2020. DOI: 10.23919/FUSION45008.2020.9190449.

[13] P. Ren, Y. Xiao, X. Chang, P. Y. Huang, Z. Li, X. Chen, and X. Wang, "A survey of deep active learning," *arXiv*, 2020, ISSN: 23318422.

[14] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1994*, pp. 3–12, 1994. DOI: 10.1007/978-1-4471-2099-5{_}1.

[15] M. Dredze and K. Crammer, "Active Learning with Confidence 2008," no. June, pp. 233–236, 2008.

[16] J. Zhu, H. Wang, B. K. Tsou, and M. Ma, "Active learning with sampling by uncertainty and density for data annotations," *IEEE Transactions on Audio, Speech and Language Pro-*

cessing, vol. 18, no. 6, pp. 1323–1331, 2010, ISSN: 15587916. DOI: 10.1109/TASL.2009.2033421.

[17] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of Artificial Intelligence Research*, vol. 4, pp. 129–145, 1996, ISSN: 10769757. DOI: 10.1613/jair.295.

[18] R. Trasarti, S. Rinzivillo, F. Pinelli, M. Nanni, A. Monreale, C. Renso, D. Pedreschi, and F. Giannotti, *Exploring real mobility data with M-atlas*, PART 3. 2010, vol. 6323 LNAI, pp. 624–627, ISBN: 3642159389. DOI: 10.1007/978-3-642-15939-8{_}48.

[19] C. Cortes, G. DeSalvo, C. Gentile, M. Mohri, and N. Zhang, "Active learning with disagreement graphs," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 2468–2483, 2019.

[20] L. Copa, D. Tuia, M. Volpi, and M. Kanevski, "Unbiased query-by-bagging active learning for VHR image classification," *Image and Signal Processing for Remote Sensing XVI*, vol. 7830, no. October 2010, 78300K, 2010, ISSN: 0277786X. DOI: 10.1117/12.864861.

[21] I. Muslea, I. Muslea, S. Minton, S. Minton, C. a. Knoblock, and C. Knoblock, "Selective sampling with redundant views," *Proceedings of the National Conference on Artificial Intelligence*, pp. 621–626, 2000. [Online]. Available: http://www.aaai.org/Papers/AAAI/2000/AAAI00-095.pdf.

[22] J. W. Kamminga, N. Meratnia, and P. J. Havinga, "Dataset: Horse movement data and analysis of its potential for activity recognition," *DATA 2019 - Proceedings of the 2nd ACM Workshop on Data Acquisition To Analysis, Part of SenSys 2019*, pp. 22–25, 2019. DOI: 10.1145/3359427.3361908.

[23] E. Bocaj, D. Uzunidis, P. Kasnesis, and C. Z. Patrikakis, "On the Benefits of Deep Convolutional Neural Networks on Animal Activity Recognition," *Proceedings of 2020 International Conference on Smart Systems and Technologies, SST 2020*, pp. 83–88, 2020. DOI: 10.1109/SST49455.2020.9263702.

[24] "Support Vector Machine," *Analutica Chimica Acta*, pp. 1–47, 2004. [Online]. Available: http://www.dbpia.co.kr/Journal/ArticleDetail/401581.

[25] L. Gao, H. A. Campbell, O. R. Bidder, and J. Hunter, "A Web-based semantic tagging and activity recognition system for species' accelerometry data," *Ecological Informatics*, vol. 13, pp. 47–56, 2013, ISSN: 15749541. DOI: 10.1016/j.ecoinf.2012.09.003. [Online]. Available: http://dx.doi.org/10.1016/j.ecoinf.2012.09.003.

[26] V. Sturm, D. Efrosinin, N. Efrosinina, L. Roland, M. Iwersen, M. Drillich, and W. Auer, "A Chaos Theoretic Approach to Animal Activity Recognition," *Journal of Mathematical Sciences (United States)*, vol. 237, no. 5, pp. 730–743, 2019, ISSN: 15738795. DOI: 10.1007/s10958-019-04199-9.

[27] H. Cho and S. M. Yoon, "Divide and conquer-based 1D CNN human activity recognition using test data sharpening," *Sensors (Switzerland)*, vol. 18, no. 4, pp. 1–24, 2018, ISSN: 14248220. DOI: 10.3390/s18041055.

[28] W. Jiang and Z. Yin, "Human activity recognition using wearable sensors by deep convolutional neural networks," *MM 2015 - Proceedings of the 2015 ACM Multimedia Conference*, pp. 1307–1310, 2015. DOI: 10.1145/2733373.2806333.

[29] Y. Kwon, K. Kang, and C. Bae, "Unsupervised learning for human activity recognition using smartphone sensors," *Expert Systems with Applications*, vol. 41, no. 14, pp. 6067–6074,

2014, ISSN: 09574174. DOI: 10.1016/j.eswa.2014.04.037. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2014.04.037.

[30] K. S. Perera, B. Neupane, M. A. Faisal, Z. Aung, and W. L. Woon, *Mining Intelligence and Knowledge Exploration*. 2013, vol. 8284, pp. 370–382, ISBN: 978-3-319-03843-8. DOI: 10.1007/978-3-319-03844-5. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84893366349&partnerID=tZOtx3y1.

[31] R. Liu, T. Chen, and L. Huang, "Research on human activity recognition based on active learning," *2010 International Conference on Machine Learning and Cybernetics, ICMLC 2010*, vol. 1, no. July, pp. 285–290, 2010. DOI: 10.1109/ICMLC.2010.5581050.

[32] M. Stikic, K. Van Laerhoven, and B. Schiele, "Exploring semi-supervised and active learning for activity recognition," *Proceedings - International Symposium on Wearable Computers, ISWC*, pp. 81–88, 2008, ISSN: 15504816. DOI: 10.1109/ISWC.2008.4911590.

[33] R. Adaimi and E. Thomaz, "Leveraging Active Learning and Conditional Mutual Information to Minimize Data Annotation in Human Activity Recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 3, pp. 1–23, 2019, ISSN: 2474-9567. DOI: 10.1145/3351228.

[34] E. Casella, A. R. Khamesi, and S. Silvestri, "Smartwatch application for horse gaits activity recognition," *Proceedings - 2019 IEEE International Conference on Smart Computing, SMARTCOMP 2019*, pp. 409–416, 2019. DOI: 10.1109/SMARTCOMP.2019.00080.

[35] Y. Yang, Z. Ma, F. Nie, X. Chang, and A. G. Hauptmann, "Multi-Class Active Learning by Uncertainty Sampling with Diversity Maximization," *International Journal of Computer Vision*, vol. 113, no. 2, pp. 113–127, 2015, ISSN: 15731405. DOI: 10.1007/s11263-014-0781-x. [Online]. Available: http://dx.doi.org/10.1007/s11263-014-0781-x.

[36] R. Arablouei, L. Currie, B. Kusy, A. Ingham, P. L. Greenwood, and G. Bishop-Hurley, "In-situ classification of cattle behavior using accelerometry data," *Computers and Electronics in Agriculture*, vol. 183, p. 106 045, Apr. 2021. DOI: 10.1016/j.compag.2021.106045.

[37] J. Kamminga, P. Havinga, N. Meratnia, and V. L. Duc, *Towards Deep Unsupervised Representation Learning from Accelerometer Time Series for Animal Activity Recognition*, Aug. 2020.

[38] *Geospatial Computing Portal*, 2020. [Online]. Available: https://crib.utwente.nl/.

[39] *Project Jupyter*, Apr. 2021. [Online]. Available: https://jupyter.org/.

[40] *pandas - Python Data Analysis Library*, Apr. 2021. [Online]. Available: https://pandas.pydata.org/.

[41] *NumPy*, 2020. [Online]. Available: https://numpy.org/.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[43] *Keras: the Python deep learning API*. [Online]. Available: https://keras.io/.

[44] C. Leifer, *peewee*. [Online]. Available: http://docs.peewee-orm.com/.

[45] M. Waskom, *seaborn: statistical data visualization*, 2020. [Online]. Available: https://seaborn.pydata.org/.

[46] *modAL.uncertainty — modAL documentation*. [Online]. Available: https://modal-python.readthedocs.io/en/latest/content/apireference/uncertainty.html.

[47] A. Olszówka-Myalska and J. Chrapońskib, "Active Learning with Real Annotation Costs Burr," *Solid State Phenomena*, vol. 227, pp. 178–181, 2015, ISSN: 16629779. DOI: 10.4028/www.scientific.net/SSP.227.178.

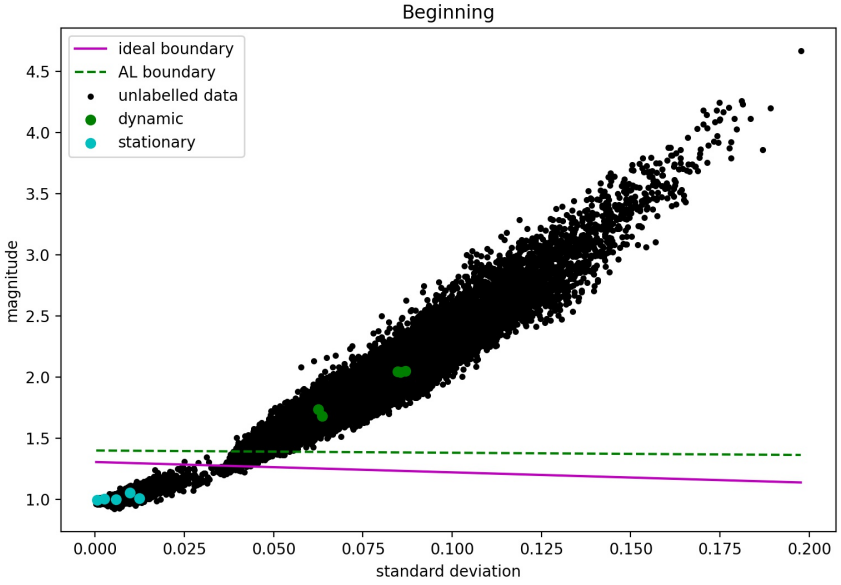
[48] M. Lorbach, R. Poppe, and R. C. Veltkamp, "Interactive rodent behavior annotation in video using active learning," *Multimedia Tools and Applications*, vol. 78, no. 14, pp. 19 787–19 806, 2019, ISSN: 15737721. DOI: 10.1007/s11042-019-7169-4.

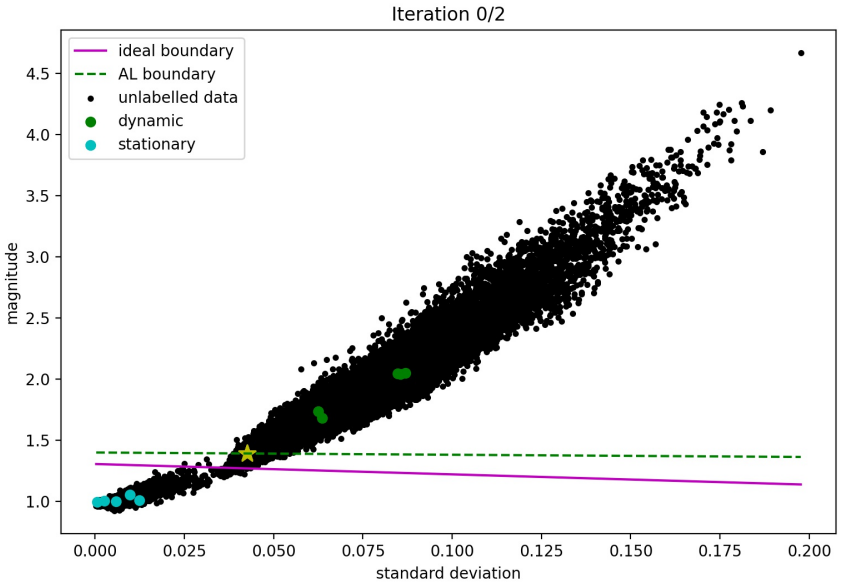
[49] *2 Annotator Jobs — ZipRecruiter*. [Online]. Available: https://www.ziprecruiter.co.uk/jobs/search?q=annotator&l=&lat=52.2385&long=6.8706&d=.

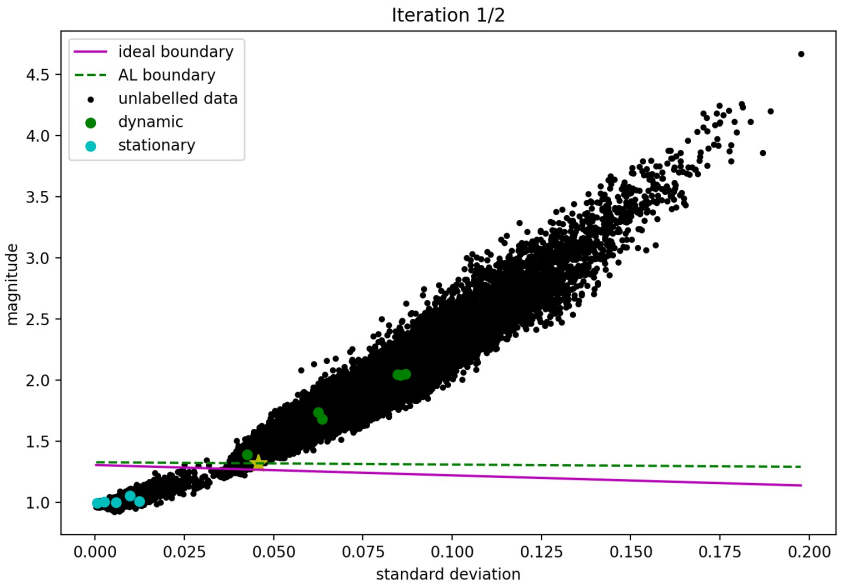
[50] R. G. Jansen, L. F. Wiertz, E. S. Meyer, and L. P. Noldus, "Reliability analysis of observational data: Problems, solutions, and software implementation," *Behavior Research Methods, Instruments, and Computers*, vol. 35, no. 3, pp. 391–399, 2003, ISSN: 07433808. DOI: 10.3758/BF03195516.

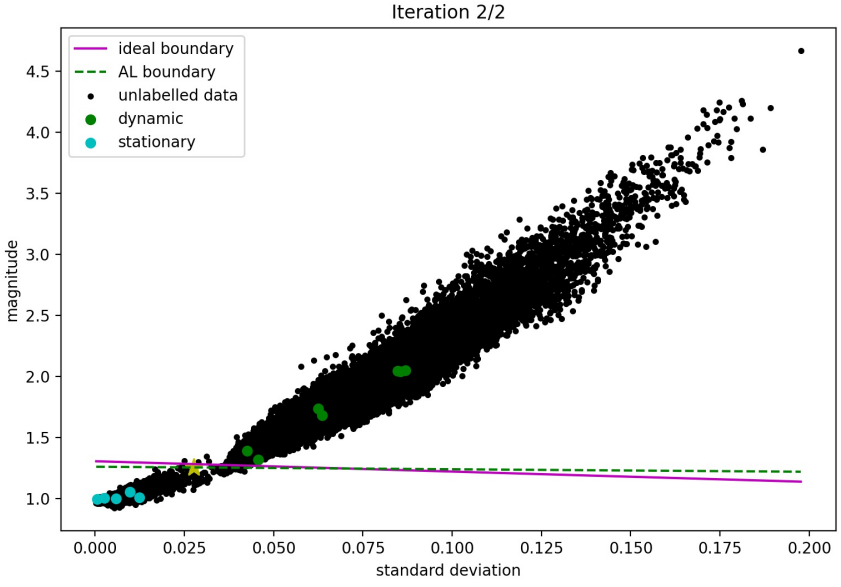
11 Appendixes

11.1 Appendix A: SVM classifier vs AL algorithm, 2 iterations

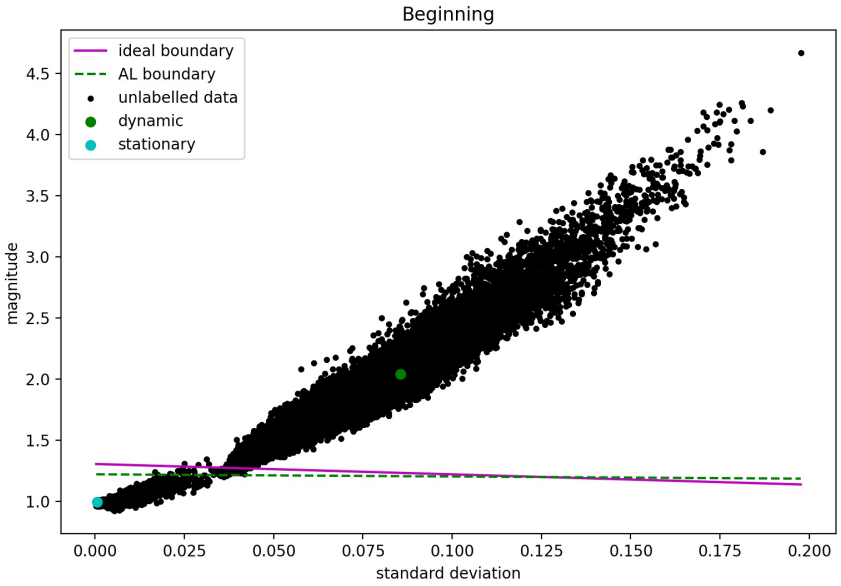


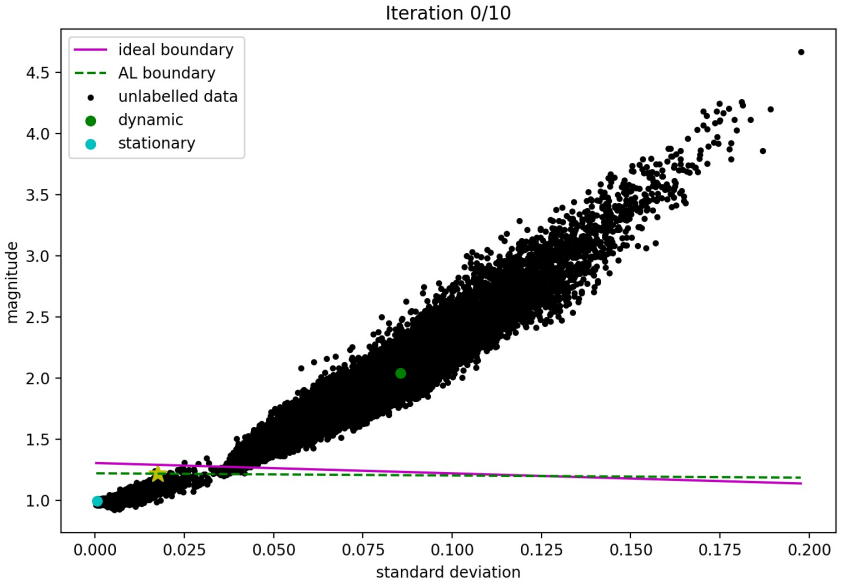


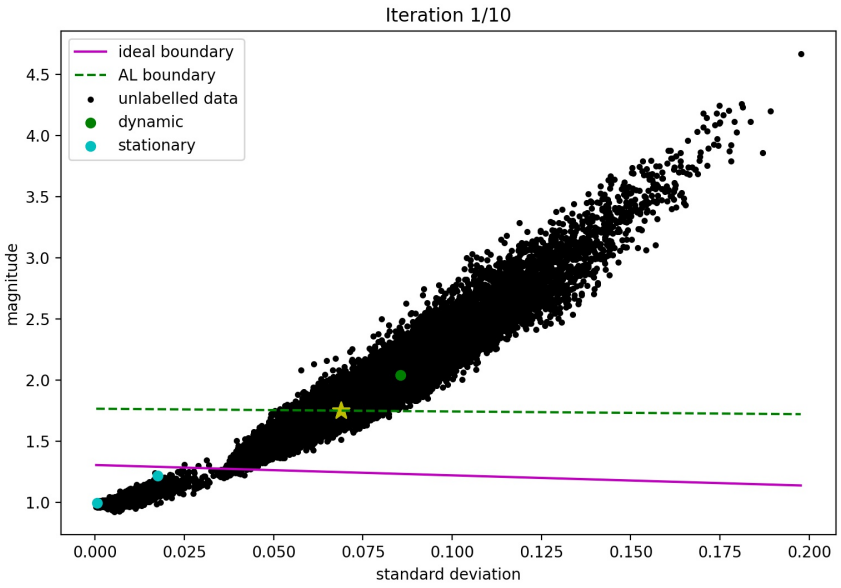

Iteration 1/2
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

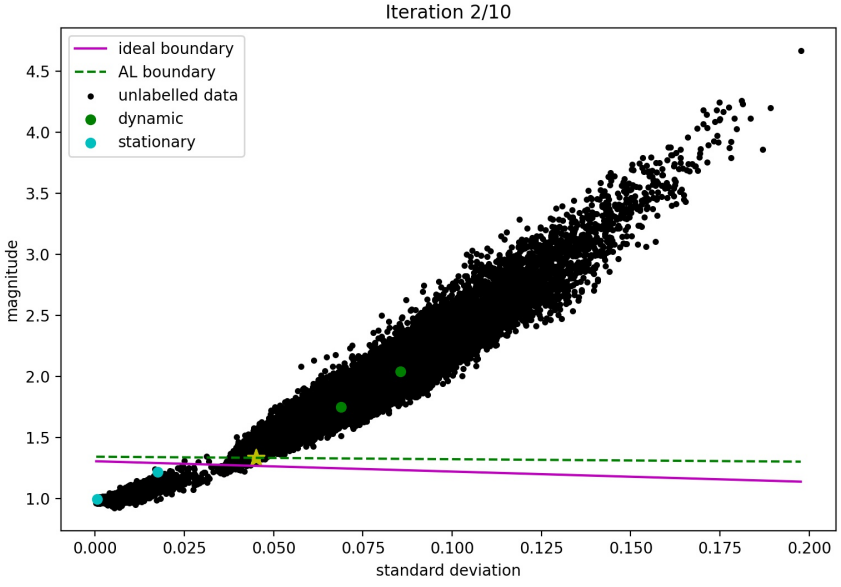

Iteration 2/2
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

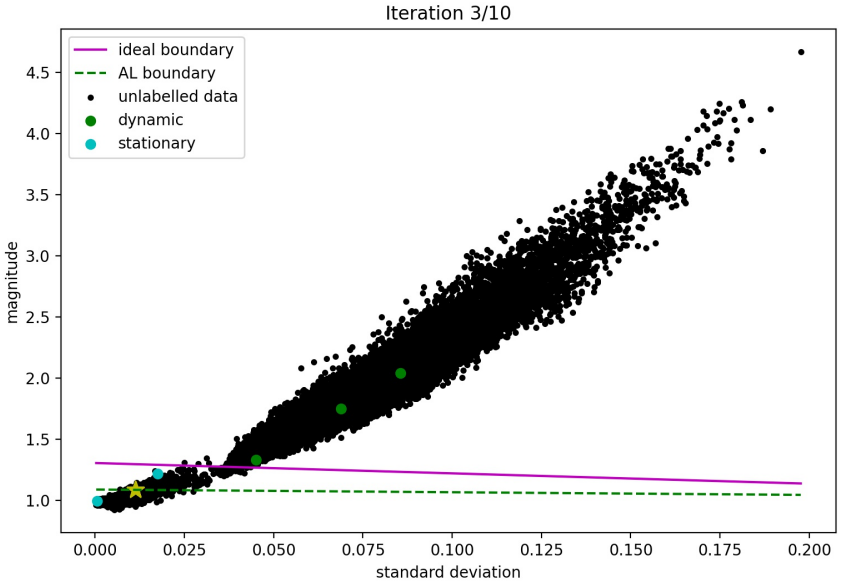
11.2 Appendix B: SVM classifier vs AL algorithm, 10 iterations

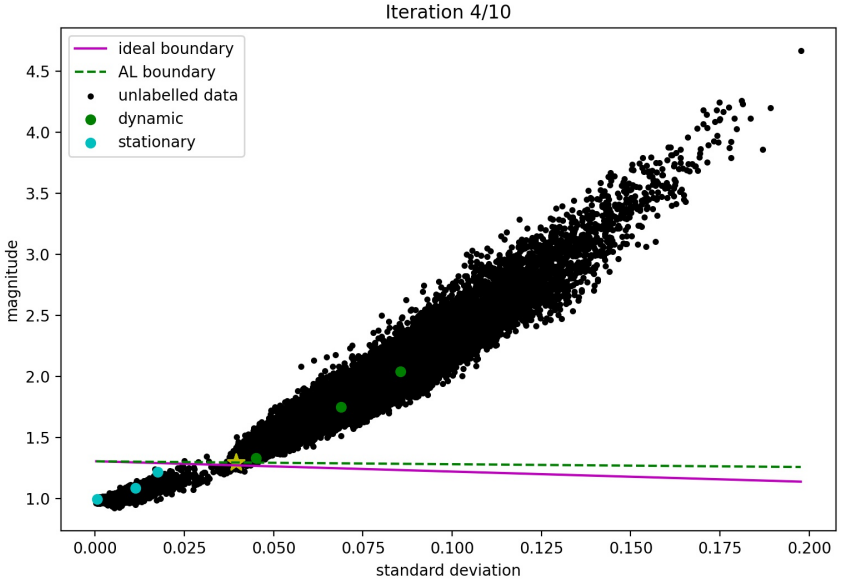


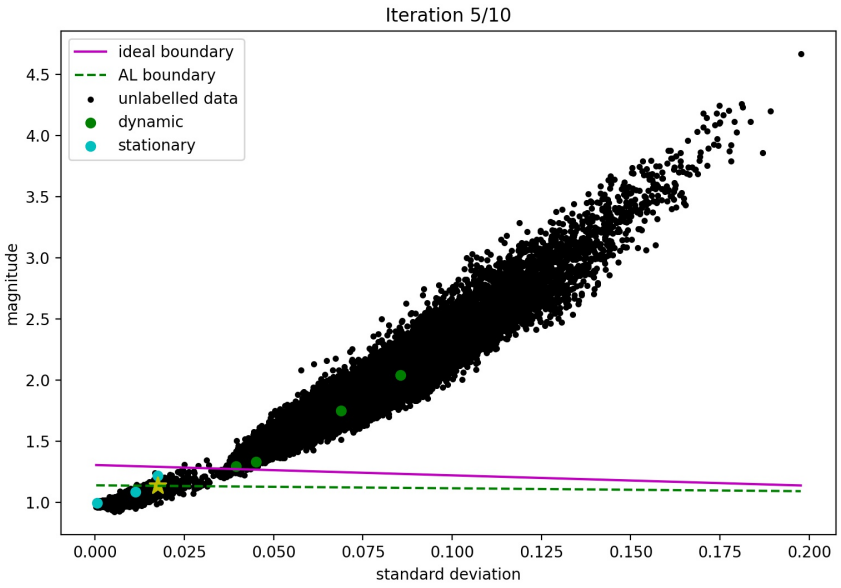


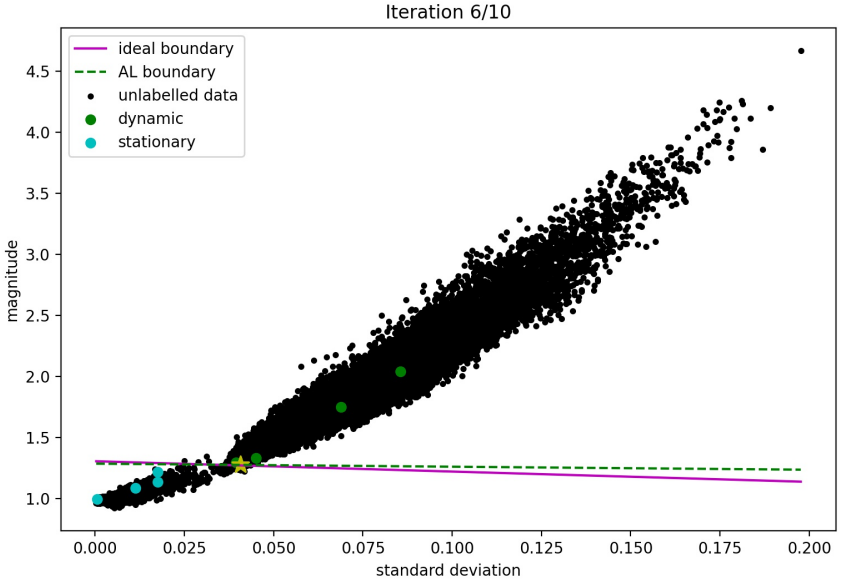

Iteration 1/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

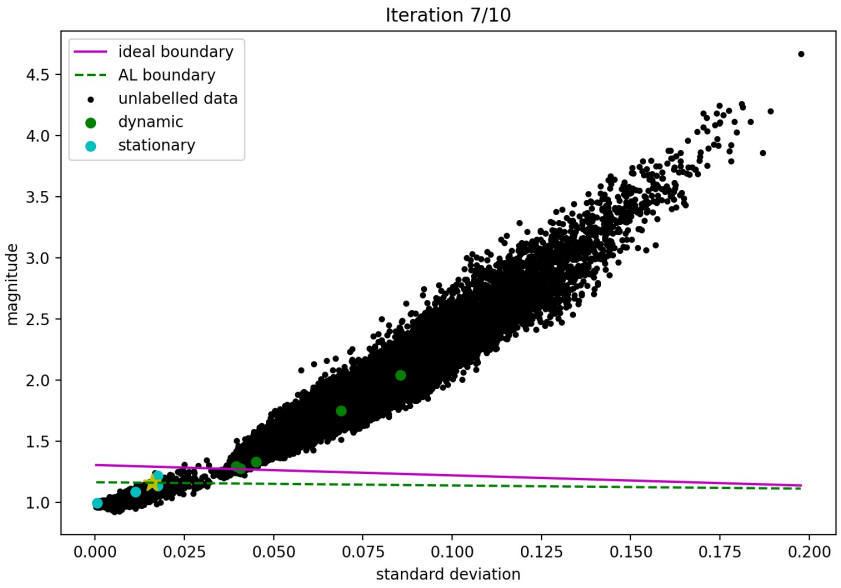

Iteration 2/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

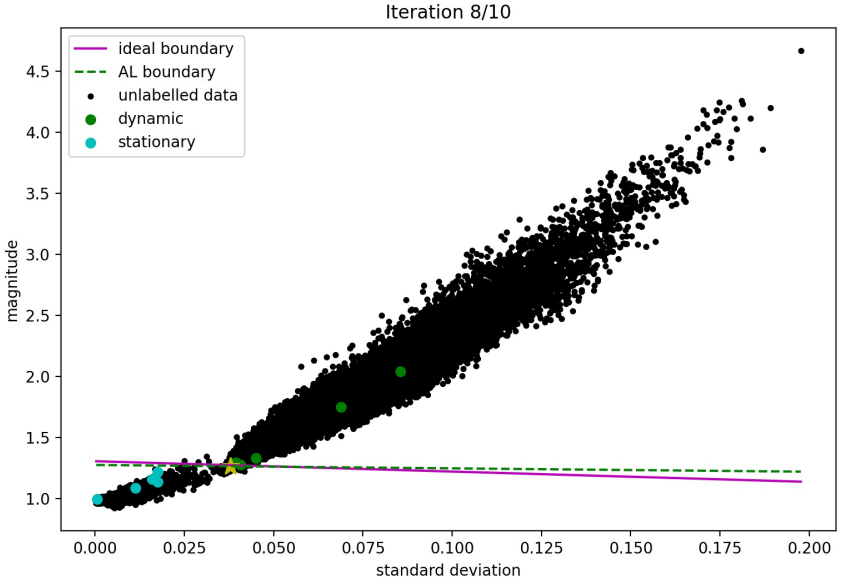

Iteration 3/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

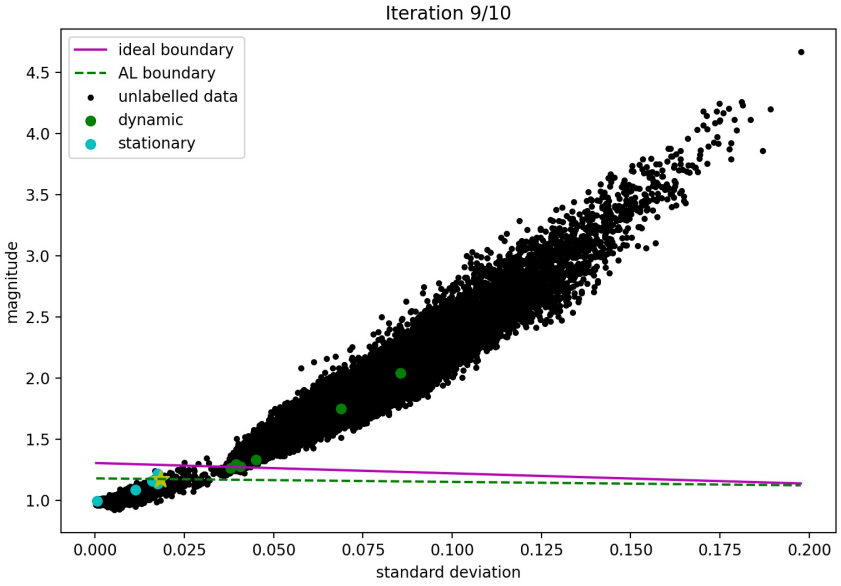

Iteration 4/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

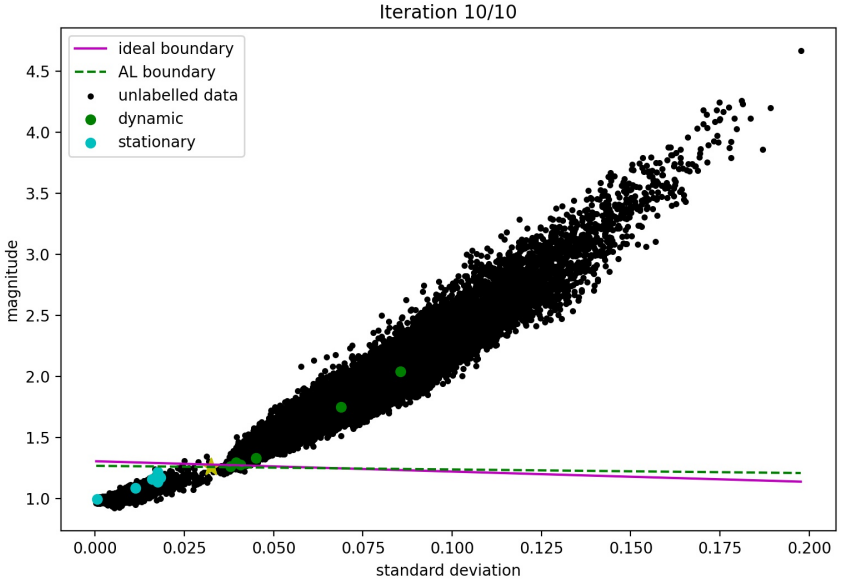

Iteration 5/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Iteration 6/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Iteration 7/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200


Iteration 8/10
ideal boundary
AL boundary
unlabelled data
dynamic
stationary
magnitude
standard deviation
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200





11.3 Appendix C: Code For Database

```
# import packages
from peewee import *
from datetime import date
import pandas as pd
```

```
import re
from ast import literal_eval
import seaborn as sns
from matplotlib import pyplot as plt

# database credentials
DB_NAME = 'HorsingAround'
USER = 'suzannespink'
PASSWORD = 'LBtBuxVfrmqvBufR'
HOST = 'www.jacobkamminga.nl'
PORT = 3306


# Establish connection to database
def connect(db_name, username, password, host, port):
    db = MySQLDatabase(db_name, user=username, password=password,
        host=host, port=port)
    db.connect()
    print("Connected to: " + host + ":" + str(port))
    return db

# save experiment in database
def save_experiment(Experiment, uid, horse, acc, balanced_acc,
    f_score_avg, mcc_score, recall, matrix, params, desc):
    Experiment.create(key=uid,
                      username="suzannespink",
                      test_horse=horse,
                      date=date.today(),
                      accuracy_experiment=acc,
                      balanced_accuracy_experiment=balanced_acc,
                      fscore=f_score_avg,
                      mcc=mcc_score,
                      recall=recall,
                      confusion_matrix=matrix,
                      parameters=params,
                      description=desc)

#     save each activity in database
def save_activity(Activity, uid, horse, activity, matrix,
    index, recall, precision):
    TP = matrix[index][index]
    FN = matrix[index].sum() - TP
    FP = 0
    for j in range(0, len(matrix)):
        FP += matrix[j][index]
```

```
    FP = FP - TP
    TN = matrix.sum() - TP - FN - FP

    specificity = TN/(TN+FP)
    accuracy = (TP+TN)/(TP+TN+FP+FN)
    #save results per activity per horse in the database
    Activity.create(key=uid,
                    test_horse=horse,
                    activity=activity,
                    accuracy_activity=accuracy,
                    recall_activity=recall,
                    specificity=specificity,
                    precision=precision,
                    TP=TP,
                    TN=TN,
                    FP=FP,
                    FN=FN)

#     differentiate classes in database
def get_classes(db):
    class Experiment(Model):
        key = UUIDField()
        username = TextField()
        test_horse = TextField()
        date = DateField()
        accuracy_experiment = FloatField()
        balanced_accuracy_experiment = FloatField()
        fscore = FloatField()
        mcc = FloatField()
        recall = FloatField()
        confusion_matrix = BlobField()
        parameters = TextField()
        description = TextField()

        class Meta:
            database = db

#     differentiate activity metrics to save in database
    class Activity(Model):
        key = UUIDField()
        test_horse = TextField()
        activity = TextField()
        accuracy_activity = FloatField()
        recall_activity = FloatField()
        specificity = FloatField()
```

```
        precision = FloatField()
        TP = IntegerField()
        TN = IntegerField()
        FP = IntegerField()
        FN = IntegerField()

        class Meta:
            database = db

    return Experiment, Activity
```

11.4 Appendix D: Code comparing Uncertainty AL algorithms

```
from sklearn.svm import SVC, LinearSVC
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import glob
from sklearn import preprocessing
from scipy import stats, signal
from keras.utils import np_utils
from sklearn.utils import shuffle

import importlib
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape
from sklearn import metrics
from matplotlib import pyplot as plt
import seaborn as sns
import uuid

import glob
import numpy as np

# ------------------------------------------------------------ #
#  Pre-Processing Constants:
# ------------------------------------------------------------ #

# Label encoder used to get a numeric representation of a label
le = preprocessing.LabelEncoder()
```

```
# The activities
LABELS = ['grazing', 'running', 'standing',
    'trotting', 'walking-natural', 'walking-rider']
print(len(LABELS))

# Add columns to drop from dataframe
REMOVE_COLUMNS = ['Mx', 'My', 'Mz', 'G3D', 'M3D',
    'Ax', 'Ay', 'Az', 'Gx', 'Gy', 'Gz' ]

# Add subjects you want to include
SUBJECTS = ['Galoway', 'Patron', 'Happy', 'Driekus']
# SUBJECTS = ['Galoway']

# Amount of features (xyz acc / xyz gyr)
N_FEATURES = 1

# number of iterations
IT_NUM = 50

# Name of the column used as output
OUTPUT_LABEL = 'ActivityEncoded'

# Sliding windows parameters
TIME_PERIODS = 200
STEP_DISTANCE = 100


# Datasets
PATH = 'horse_data/*'
FILES = sorted(glob.glob(PATH))


# ------------------------------------------------ #
#  Helper functions:
# ------------------------------------------------ #

def create_dataframe(files):
    """
    Simple function to set up dataframe and initial clean-up of the data
    files: path to files
    returns: dataframe
    """
    result = pd.DataFrame()
    # Pick only the files in SUBJECTS
    matching = [f for f in files if any(s in f for s in SUBJECTS)]
```

```
    for file in matching:
        csv = pd.read_csv(file)
        csv['filename'] = file
        result = result.append(csv)

    # remove redundant columns
    result.drop(REMOVE_COLUMNS, axis=1, inplace=True)
    result = select_activities(result)
    # create a new column with a unique integer value for each label
    result[OUTPUT_LABEL] = le.fit_transform(result['label'].values.ravel())
#     print(result)
    return result

def select_activities(df):
    df['label'] = df['label'].replace(to_replace=['trotting-natural'],
    value='trotting')
    df['label'] = df['label'].replace(to_replace=['trotting-rider'],
    value='trotting')
    df['label'] = df['label'].replace(to_replace=['running-natural'],
    value='running')
    df['label'] = df['label'].replace(to_replace=['running-rider'],
    value='running')
    result = df[df['label'].isin(LABELS)]
    return result

def filter(df):
    sos = signal.butter(N=3, Wn=30, btype='lowpass', fs=100, output='sos')
    df['A3D'] = signal.sosfilt(sos, df['A3D'])
    return df


def split_by_subject(df, name):
    test = df[df['filename'].str.contains(name)]
    pool = df[~df['filename'].str.contains(name)]
    return pool, test


def feature_scaling(df):
#     print("df=")
#     print (df)
#     print("A3D row=")
#     print(df['A3D'])
    train_A3D_max = df['A3D'].max()
```

```
    pd.options.mode.chained_assignment = None

    df['A3D'] = df['A3D']/train_A3D_max

    return df


def create_windows(df, time_steps, step, label_name):
    windows = []
    labels = []
    for i in range(0, len(df) - time_steps, step):
        A3d = df['A3D'].values[i: i + time_steps]
        # Retrieve the most often used label in this segment
        label = stats.mode(df[label_name][i: i + time_steps])[0][0]
        windows.append(A3d)
        labels.append(label)
    # Bring the segments into a better shape
    reshaped_windows = np.asarray(windows, dtype=np.float32).
        reshape(-1, time_steps, N_FEATURES)
    labels = np.asarray(labels)

    return reshaped_windows, labels



# Reshape input into a format compatible with the NN
def reshape_input(x, shape):
    result = x.reshape(x.shape[0], shape)
    return result


# Apply one hot coding to output
def encode_output(y, classes):
    result = np_utils.to_categorical(y, classes)
    return result


def preprocess_training(df, input_shape, num_classes):
    train = feature_scaling(df)
    X_pool, y_pool = create_windows(train,
            TIME_PERIODS, STEP_DISTANCE, OUTPUT_LABEL)
    X_pool, y_pool = shuffle(np.array(X_pool), np.array(y_pool))
    X_pool = X_pool.astype('float32')
    y_pool = y_pool.astype('float32')
    y_pool = encode_output(y_pool, num_classes)
```

```
    return X_pool, y_pool


def preprocess_test(df, input_shape, num_classes):
    test = feature_scaling(df)
    x_test, y_test = create_windows(test,
        TIME_PERIODS, STEP_DISTANCE, OUTPUT_LABEL)
    x_test = reshape_input(x_test, input_shape)
    x_test = x_test.astype('float32')
    y_test = y_test.astype('float32')
    y_test = encode_output(y_test, num_classes)

    return x_test, y_test


def preprocess(df, test_subject):
    input_shape = (TIME_PERIODS * N_FEATURES)
    num_classes = len(LABELS)
    df = filter(df)
    pool, test = split_by_subject(df, test_subject)
    X_pool, y_pool = preprocess_training(pool, input_shape, num_classes)
    x_test, y_test = preprocess_test(test, input_shape, num_classes)

    return X_pool, y_pool, x_test, y_test, input_shape, num_classes

# visualise the MCC and F1-score per iteration for the horses
def visualise(F11, F12, F13, MCC1, MCC2, MCC3):
#     get iteration length
    iteration  = list(range(0, IT_NUM+1))

#     plot F1-score for each algorithm
    plt.plot(iteration, F11, color="#009933")
    plt.plot(iteration, F12, color="#0099ff")
    plt.plot(iteration, F13, color="#9933ff")
    plt.title("F1-score per iteration")
    plt.xlabel("Iteration")
    plt.ylabel("F1-score")
    plt.legend(['Least Certain', 'Margin', 'Entropy'])
    plt.grid(True)
    plt.savefig("F1", dpi=600)
    plt.show()

#     plot MCC for each algorithm
    plt.plot(iteration, MCC1, color="#009933")
    plt.plot(iteration, MCC2, color="#0099ff")
```

```
    plt.plot(iteration, MCC3, color="#9933ff")
    plt.title("MCC per iteration")
    plt.xlabel("Iteration")
    plt.ylabel("MCC")
    plt.legend(['Least Certain', 'Margin', 'Entropy'])
    plt.grid(True)
    plt.savefig("MCC", dpi=600)

    plt.show()

# import modAL packages
from modAL.uncertainty import classifier_uncertainty as uncer
from modAL.uncertainty import classifier_entropy as entrop
from modAL.uncertainty import classifier_margin as margin

# use various uncertainty algorithm types
def algorithms(clf, X_rest):
    UNCERT = uncer(clf, X_rest)
    MARGIN = margin(clf, X_rest)
    ENTROPY = entrop(clf, X_rest)
    return UNCERT, MARGIN, ENTROPY

# find most ambiguous instance using the algorithm
def find_most_ambiguous(clf, X_rest, y_rest, alg):
 #     max uncertainty
    uncerta = alg
    uncertain = np.amax(uncerta)
#     max uncertainty index
    loc = np.argmax(uncerta, axis=-1)
    loc = np.unravel_index(uncerta.argmax(), uncerta.shape)
    # find value of max uncertainty
    numbers = X_rest[loc]
    number = np.reshape(numbers, (1, 200, 1))
    # find label of max uncertainty
    labelInd = y_rest[loc]

    return loc, uncertain, number, labelInd

# least certain AL
def AL_uncert(model, INITIAL):
#     initialise training set size
    X_train = X_pool[:INITIAL]
    y_train = y_pool[:INITIAL]

#     initialise rest of training set
```

```
    X_rest = X_pool[INITIAL:]
    y_rest = y_pool[INITIAL:]

#     classifier
    clf = model
    clf.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])
    clf.fit(X_train, y_train)

#     find instance to add
    UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
    loc, uncertain, number, labelInd = find_most_ambiguous(clf,
        X_rest, y_rest, UNCERT)

#     delete instance from rest
    X_rest = np.delete(X_rest, loc, axis = 0)
    y_rest = np.delete(y_rest, loc, axis = 0)

#     prediction on test
    test_prediction = clf.predict(xtest)

#     best test prediction
    max_test_prediction = np.argmax(test_prediction, axis=1)
#     best ytest value
    max_y_test = np.argmax(ytest, axis=1)

#     get evaluation results of the test data
    precision_per_class, recall_per_class, f_score_per_class, recall_avg,
        f_score_avg, mcc_score, balanced_acc, acc =
            metrics_call(max_y_test, max_test_prediction, model)

#     for the average per iteration for visualisation
    F1 = []
    MCC = []
    calc_F1 = f_score_avg
    calc_MCC = mcc_score
    F1.append(calc_F1)
    MCC.append(mcc_score)

#     iteratively adding most ambiguous instance IT_NUM number of times
    for i in range(IT_NUM):

#         add most ambiguous to training set
        X_train = np.vstack((X_train, number))
        y_train = np.vstack((y_train, labelInd))
```

```
#           train model again
            clf = model
            clf.fit(X_train, y_train)

#           find new most uncertain
            UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
            loc, uncertain, number, labelInd = find_most_ambiguous(clf,
                X_rest, y_rest, UNCERT)

#           remove most uncertain from unknown rest set
            X_rest = np.delete(X_rest, loc, axis = 0)
            y_rest = np.delete(y_rest, loc, axis = 0)

#           prediction on test
            test_prediction = clf.predict(xtest)
#           best test prediction
            max_test_prediction = np.argmax(test_prediction, axis=1)
#           best ytest value
            max_y_test = np.argmax(ytest, axis=1)
#           get evaluation results of the test data
            precision_per_class, recall_per_class, f_score_per_class, recall_avg,
                f_score_avg, mcc_score, balanced_acc, acc =
                    metrics_call(max_y_test, max_test_prediction, model)
#           add F1 and MCC values per iteration
            calc_F1 = f_score_avg
            calc_MCC = mcc_score
            F1.append(calc_F1)
            MCC.append(mcc_score)

#           print iteration during running
            print("ITERATION:")
            print(i)

    return X_rest, y_rest, clf, F1, MCC
# AL margin algorithm
def AL_margin(model, INITIAL):
#     initialise training set size
    X_train = X_pool[:INITIAL]
    y_train = y_pool[:INITIAL]

#     initialise rest of training set
    X_rest = X_pool[INITIAL:]
    y_rest = y_pool[INITIAL:]
```

```
#     classifier
    clf = model
    clf.compile(loss='categorical_crossentropy', optimizer='adam',
        metrics=['accuracy'])
    clf.fit(X_train, y_train)

#     find instance to add
    UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
    loc, uncertain, number, labelInd = find_most_ambiguous(clf,
        X_rest, y_rest, MARGIN)

#     delete instance from rest
    X_rest = np.delete(X_rest, loc, axis = 0)
    y_rest = np.delete(y_rest, loc, axis = 0)

#     prediction on test
    test_prediction = clf.predict(xtest)

#     best test prediction
    max_test_prediction = np.argmax(test_prediction, axis=1)
#     best ytest value
    max_y_test = np.argmax(ytest, axis=1)

#     get evaluation results of the test data
    precision_per_class, recall_per_class, f_score_per_class, recall_avg,
        f_score_avg, mcc_score, balanced_acc, acc =
            metrics_call(max_y_test, max_test_prediction, model)

#     for the average per iteration for visualisation
    F1 = []
    MCC = []
    calc_F1 = f_score_avg
    calc_MCC = mcc_score
    F1.append(calc_F1)
    MCC.append(mcc_score)

#     iteratively adding most ambiguous instance IT_NUM number of times
    for i in range(IT_NUM):

#         add most ambiguous to training set
        X_train = np.vstack((X_train, number))
        y_train = np.vstack((y_train, labelInd))

#         train model again
        clf = model
```

```
        clf.fit(X_train, y_train)

#         find new most uncertain
        UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
        loc, uncertain, number, labelInd = find_most_ambiguous(clf,
            X_rest, y_rest, MARGIN)

#         remove most uncertain from unknown rest set
        X_rest = np.delete(X_rest, loc, axis = 0)
        y_rest = np.delete(y_rest, loc, axis = 0)

#         prediction on test
        test_prediction = clf.predict(xtest)
#         best test prediction
        max_test_prediction = np.argmax(test_prediction, axis=1)
#         best ytest value
        max_y_test = np.argmax(ytest, axis=1)
#         get evaluation results of the test data
        precision_per_class, recall_per_class, f_score_per_class, recall_avg,
            f_score_avg, mcc_score, balanced_acc, acc = metrics_call(max_y_test, m
#         add F1 and MCC values per iteration
        calc_F1 = f_score_avg
        calc_MCC = mcc_score
        F1.append(calc_F1)
        MCC.append(mcc_score)

#         print iteration during running
        print("ITERATION:")
        print(i)

    return X_rest, y_rest, clf, F1, MCC

# AL entropy algoritm
def AL_entropy(model, INITIAL):
#     initialise training set size
    X_train = X_pool[:INITIAL]
    y_train = y_pool[:INITIAL]

#     initialise rest of training set
    X_rest = X_pool[INITIAL:]
    y_rest = y_pool[INITIAL:]

#     classifier
    clf = model
    clf.compile(loss='categorical_crossentropy', optimizer='adam',
```

```
        metrics=['accuracy'])
    clf.fit(X_train, y_train)

#     find instance to add
    UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
    loc, uncertain, number, labelInd = find_most_ambiguous(clf,
        X_rest, y_rest, ENTROPY)

#     delete instance from rest
    X_rest = np.delete(X_rest, loc, axis = 0)
    y_rest = np.delete(y_rest, loc, axis = 0)

#     prediction on test
    test_prediction = clf.predict(xtest)

#     best test prediction
    max_test_prediction = np.argmax(test_prediction, axis=1)
#     best ytest value
    max_y_test = np.argmax(ytest, axis=1)

#     get evaluation results of the test data
    precision_per_class, recall_per_class, f_score_per_class, recall_avg,
f_score_avg, mcc_score, balanced_acc, acc =
        metrics_call(max_y_test, max_test_prediction, model)

#     for the average per iteration for visualisation
    F1 = []
    MCC = []
    calc_F1 = f_score_avg
    calc_MCC = mcc_score
    F1.append(calc_F1)
    MCC.append(mcc_score)

#     iteratively adding most ambiguous instance IT_NUM number of times
    for i in range(IT_NUM):

#         add most ambiguous to training set
        X_train = np.vstack((X_train, number))
        y_train = np.vstack((y_train, labelInd))

#          train model again
        clf = model
        clf.fit(X_train, y_train)

#         find new most uncertain
```

```
        UNCERT, MARIGN, ENTROPY = algorithms(clf, X_rest)
        loc, uncertain, number, labelInd = find_most_ambiguous(clf,
            X_rest, y_rest, ENTROPY)

#         remove most uncertain from unknown rest set
        X_rest = np.delete(X_rest, loc, axis = 0)
        y_rest = np.delete(y_rest, loc, axis = 0)

#         prediction on test
        test_prediction = clf.predict(xtest)
#         best test prediction
        max_test_prediction = np.argmax(test_prediction, axis=1)
#         best ytest value
        max_y_test = np.argmax(ytest, axis=1)
#         get evaluation results of the test data
        precision_per_class, recall_per_class, f_score_per_class, recall_avg,
            f_score_avg, mcc_score, balanced_acc, acc =
                metrics_call(max_y_test, max_test_prediction, model)
#         add F1 and MCC values per iteration
        calc_F1 = f_score_avg
        calc_MCC = mcc_score
        F1.append(calc_F1)
        MCC.append(mcc_score)

#         print iteration during running
        print("ITERATION:")
        print(i)

    return X_rest, y_rest, clf, F1, MCC


# build classifier
def build_classifier(input_shape, num_classes):
    model_m = Sequential()
#     reshape
    model_m.add(Reshape((TIME_PERIODS, ), input_shape=(input_shape, )))
#     hidden layers
    model_m.add(Dense(100, activation='relu'))
    model_m.add(Dense(100, activation='relu'))
    model_m.add(Dense(100, activation='relu'))
#     flatten
    model_m.add(Flatten())
    model_m.add(Dense(6, activation='softmax'))
    return model_m
```

```
# construct confusion matrix
def show_confusion_matrix(validations, predictions, model_m):
    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(6, 4))
    sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS,
                yticklabels=LABELS,
                annot=True,
                fmt='d')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
    return matrix

# get metrics
def metrics_call(validations, predictions, model_m):
#     per class precision, recall, f1-score
    precision_per_class, recall_per_class,
        f_score_per_class, _ = metrics.precision_recall_fscore_support
            (validations, predictions, zero_division=0)
#     average recall, f1-score
    _, recall_avg, f_score_avg, _ = metrics.precision_recall_fscore_support
        (validations, predictions, pos_label=None,
            average="weighted", zero_division=0)
#     mcc
    mcc = metrics.matthews_corrcoef(validations, predictions)
#     accuracy balanced
    balanced_accuracy = metrics.balanced_accuracy_score(validations, predictions)
#     accuracy
    accuracy = metrics.accuracy_score(validations, predictions)

    print('\nAccuracy on test data: %0.2f' % accuracy)

    return precision_per_class, recall_per_class, f_score_per_class,
        recall_avg, f_score_avg, mcc, balanced_accuracy, accuracy

PATH = 'horse_data/*'
FILES = sorted(glob.glob(PATH))

df = create_dataframe(FILES)
```

```
for s in SUBJECTS:
#       print name horse
    print("Test subject: "+str(s))
#           preprocess all inputs
    X_pool, y_pool, xtest, ytest, input_shape, num_classes = preprocess(df, s)

#       build model
    model = build_classifier(input_shape, num_classes)
#       run uncertainty algorithm
    X_rest, y_rest, clf, F11, MCC1 = AL_uncert(model, 50)
#       run margin algorithm
    X_rest, y_rest, clf, F12, MCC2 = AL_margin(model, 50)
#       run entropy algorithm
    X_rest, y_rest, clf, F13, MCC3 = AL_entropy(model, 50)


#     categorise which score belongs to which horse and which algorithm
    if (str(s)=="Galoway"):
        GalF1 = F11
        GalMCC1 = (MCC1)
        GalF2 = (F12)
        GalMCC2 = (MCC2)
        GalF3 = (F13)
        GalMCC3 = (MCC3)

    if (str(s) == "Patron"):
        PatF1 = (F11)
        PatMCC1 = (MCC1)
        PatF2 = (F12)
        PatMCC2 = (MCC2)
        PatF3 = (F13)
        PatMCC3 = (MCC3)

    if (str(s) == "Happy"):
        HapF1 = (F11)
        HapMCC1 = (MCC1)
        HapF2 = (F12)
        HapMCC2 = (MCC2)
        HapF3 = (F13)
        HapMCC3 = (MCC3)

    if (str(s) == "Driekus"):
        DrieF1 = (F11)
        DrieMCC1 = (MCC1)
```

```
DrieF2 = (F12)
DrieMCC2 = (MCC2)
DrieF3 = (F13)
DrieMCC3 = (MCC3)
```

```
# average of all F1 scores and MCCs for all algorithms
avg_F11 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF1, PatF1, HapF1, DrieF1) ]
avg_F12 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF2, PatF2, HapF2, DrieF2) ]
avg_F13 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF3, PatF3, HapF3, DrieF3) ]
avg_MCC1 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC1, PatMCC1, HapMCC1, DrieMCC1)]
avg_MCC2 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC2, PatMCC2,  HapMCC2, DrieMCC2)]
avg_MCC3 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC3, PatMCC3, HapMCC3, DrieMCC3)]

# Confusion matrix
test_prediction = clf.predict(xtest)

# best test prediction
max_test_prediction = np.argmax(test_prediction, axis=1)
# best ytest value
max_y_test = np.argmax(ytest, axis=1)

#print confusion matrix and get evaluation results of the test data
matrix = show_confusion_matrix(max_y_test, max_test_prediction, model)

# visualise
visualise(avg_F11, avg_F12, avg_F13,avg_MCC1, avg_MCC2, avg_MCC3)

# get metrics
precision_per_class, recall_per_class, f_score_per_class, recall_avg,
    f_score_avg, mcc_score, balanced_acc, acc =
        metrics_call(max_y_test, max_test_prediction, model)
```

11.5 Appendix E: Initial Training Set Size Comparing Uncertainty AL algorithms

Will excluded Preprocessing, importing packages, algorithm types, find most ambiguous function and metric functions:

```
# visualise initial training set sizes
def visualise(F11, F12, F13, F14, MCC1, MCC2, MCC3, MCC4):
    iteration  = list(range(0, IT_NUM+1))
```

```
    plt.plot(iteration, F11, color="olive")
    plt.plot(iteration, F12, color="hotpink")
    plt.plot(iteration, F13, color="skyblue")
    plt.plot(iteration, F14, color="m")
    plt.title("F1-score per iteration")
    plt.xlabel("Iteration")
    plt.ylabel("F1-score")
    plt.legend(['80', '150', '250', '350'])
    plt.grid(True)
    plt.show()

    plt.plot(iteration, MCC1, color="olive")
    plt.plot(iteration, MCC2, color="hotpink")
    plt.plot(iteration, MCC3, color="skyblue")
    plt.plot(iteration, MCC4, color="m")
    plt.title("MCC per iteration")
    plt.xlabel("Iteration")
    plt.ylabel("MCC")
    plt.legend(['80', '150', '250', '350'])
    plt.grid(True)
    plt.show()

# AL algorith structure
def AL(model, INITIAL):
#     initialise training set size
    X_train = X_pool[:INITIAL]
    y_train = y_pool[:INITIAL]

#     initialise rest of training set
    X_rest = X_pool[INITIAL:]
    y_rest = y_pool[INITIAL:]

#     classifier
    clf = model
    clf.compile(loss='categorical_crossentropy',
        optimizer='adam', metrics=['accuracy'])
    clf.fit(X_train, y_train)

#     find instance to add
    loc, uncertain, number, labelInd =
        find_most_ambiguous(clf, X_rest, y_rest)

#     delete instance from rest
    X_rest = np.delete(X_rest, loc, axis = 0)
    y_rest = np.delete(y_rest, loc, axis = 0)
```

```
#     prediction on test
    test_prediction = clf.predict(xtest)

#     best test prediction
    max_test_prediction = np.argmax(test_prediction, axis=1)
#     best ytest value
    max_y_test = np.argmax(ytest, axis=1)

#     get evaluation results of the test data
    precision_per_class, recall_per_class, f_score_per_class, recall_avg,
        f_score_avg, mcc_score, balanced_acc, acc =
            metrics_call(max_y_test, max_test_prediction, model)

#     for the average per iteration for visualisation
    F1 = []
    MCC = []
    calc_F1 = f_score_avg
    calc_MCC = mcc_score
    F1.append(calc_F1)
    MCC.append(mcc_score)

#     iteratively adding most ambiguous instance IT_NUM number of times
    for i in range(IT_NUM):

#         add most ambiguous to training set
        X_train = np.vstack((X_train, number))
        y_train = np.vstack((y_train, labelInd))

#          train model again
        clf = model
        clf.fit(X_train, y_train)

#         find new most uncertain
        loc, uncertain, number, labelInd =
            find_most_ambiguous(clf, X_rest, y_rest)

#         remove most uncertain from unknown rest set
        X_rest = np.delete(X_rest, loc, axis = 0)
        y_rest = np.delete(y_rest, loc, axis = 0)

#         prediction on test
        test_prediction = clf.predict(xtest)
#         best test prediction
        max_test_prediction = np.argmax(test_prediction, axis=1)
```

```
#           best ytest value
          max_y_test = np.argmax(ytest, axis=1)
#           get evaluation results of the test data
          precision_per_class, recall_per_class, f_score_per_class, recall_avg,
              f_score_avg, mcc_score, balanced_acc, acc =
                  metrics_call(max_y_test, max_test_prediction, model)
#           add F1 and MCC values per iteration
          calc_F1 = f_score_avg
          calc_MCC = mcc_score
          F1.append(calc_F1)
          MCC.append(mcc_score)

#           print iteration during running
          print("ITERATION:")
          print(i)

    return X_rest, y_rest, clf, F1, MCC

PATH = 'horse_data/*'
FILES = sorted(glob.glob(PATH))

df = create_dataframe(FILES)


for s in SUBJECTS:
#     print name horse
    print("Test subject: "+str(s))
#     preprocess all inputs
    X_pool, y_pool, xtest, ytest, input_shape, num_classes = preprocess(df, s)
#     build model
    model = build_classifier(input_shape, num_classes)
#     run AL with different initial training set sizes
    X_rest, y_rest, clf, F11, MCC1 = AL(model, 80)
    X_rest, y_rest, clf, F12, MCC2 = AL(model, 150)
    X_rest, y_rest, clf, F13, MCC3 = AL(model, 250)
    X_rest, y_rest, clf, F14, MCC4 = AL(model, 350)

#     categorise which score belongs to which horse and which algorithm
    if (str(s)=="Galoway"):
        GalF1 = F11
        print("Gal")
        print(GalF1)
        GalMCC1 = (MCC1)
        GalF2 = (F12)
        GalMCC2 = (MCC2)
```

```
        GalF3 = (F13)
        GalMCC3 = (MCC3)
        GalF4 = (F14)
        GalMCC4 = (MCC4)
    if (str(s) == "Patron"):
        PatF1 = (F11)
        PatMCC1 = (MCC1)
        PatF2 = (F12)
        PatMCC2 = (MCC2)
        PatF3 = (F13)
        PatMCC3 = (MCC3)
        PatF4 = (F14)
        PatMCC4 = (MCC4)
    if (str(s) == "Happy"):
        HapF1 = (F11)
        HapMCC1 = (MCC1)
        HapF2 = (F12)
        HapMCC2 = (MCC2)
        HapF3 = (F13)
        HapMCC3 = (MCC3)
        HapF4 = (F14)
        HapMCC4 = (MCC4)
    if (str(s) == "Driekus"):
        DrieF1 = (F11)
        DrieMCC1 = (MCC1)
        DrieF2 = (F12)
        DrieMCC2 = (MCC2)
        DrieF3 = (F13)
        DrieMCC3 = (MCC3)
        DrieF4 = (F14)
        DrieMCC4 = (MCC4)


# average of all arrays

avg_F11 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF1, PatF1, HapF1, DrieF1) ]
avg_F12 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF2, PatF2, HapF2, DrieF2) ]
avg_F13 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF3, PatF3, HapF3, DrieF3) ]
avg_F14 = [((i+j+k+l)/4) for i, j, k, l in zip(GalF4, PatF4, HapF4, DrieF4) ]
avg_MCC1 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC1, PatMCC1, HapMCC1, DrieMCC1)]
avg_MCC2 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC2, PatMCC2,  HapMCC2, DrieMCC2)]
avg_MCC3 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC3, PatMCC3, HapMCC3, DrieMCC3)]
```

```
avg_MCC4 = [((i+j+k+l)/4) for i, j, k, l in
    zip(GalMCC4, PatMCC4, HapMCC4, DrieMCC4)]


# Confusion matrix
test_prediction = clf.predict(xtest)

# best test prediction
max_test_prediction = np.argmax(test_prediction, axis=1)
# best ytest value
max_y_test = np.argmax(ytest, axis=1)

#print confusion matrix and get evaluation results of the test data
matrix = show_confusion_matrix(max_y_test, max_test_prediction, model)

# visualise
visualise(avg_F11, avg_F12, avg_F13,avg_MCC1, avg_MCC2, avg_MCC3)

# get metrics
precision_per_class, recall_per_class, f_score_per_class, recall_avg, f_score_avg,
    mcc_score, balanced_acc, acc =
        metrics_call(max_y_test, max_test_prediction, model)
```

11.6 Appendix F: Initial Training Set Sizes Comparing Disagreement AL algorithms

```
from modAL.disagreement import max_disagreement_sampling as max_dis
from modAL.disagreement import consensus_entropy_sampling as consens_entrop
from modAL.disagreement import vote_entropy_sampling as vote
from sklearn.ensemble import RandomForestClassifier as forest
from sklearn.neighbors import KNeighborsClassifier as neigh
from modAL.models import ActiveLearner, Committee

# AL algorith structure
def AL(X_pool, y_pool, xtest, ytest, model1, INITIAL):

#     copy pool data
    X_pooled = deepcopy(X_pool)
    y_pooled = deepcopy(y_pool)

#     learner list
    learner_list = list()
#     initiate active learners
    for x in range(2):
#         initialise model with random instances
        clf_CNN = model1
```

```
        clf_CNN.compile(loss='categorical_crossentropy', optimizer='adam', metrics=

#         initiate training set
        train_idx = np.random.choice(range(X_pooled.shape[0]), size=INITIAL, replac
        X_train = X_pooled[train_idx]
        y_train = y_pooled[train_idx]

#         delete training from pool
        X_pooled = np.delete(X_pooled, train_idx, axis=0)
        y_pooled = np.delete(y_pooled, train_idx, axis=0)

#         initiate learners
        learner = ActiveLearner(
            estimator=clf_CNN,
            X_training=X_train, y_training=y_train
        )

        learner_list.append(learner)

#     iteratively find best instances
    for i in range(IT_NUM):
#         query committee
        query_idx, query_instance = committee.query(X_pooled)
#         teach committee
        committee.teach(
            X=X_pooled[query_idx].reshape(1, -1),
            y=y_pooled[query_idx].reshape(1, -1)
        )
#         delete new instances from pool
        X_pooled = np.delete(X_pooled, query_idx, axis=0)
        y_pooled = np.delete(y_pooled, query_idx, axis=0)

    return committee
```