Improving the efficiency of robotic packaging lines using AI

Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne

Thèse de Master de

Vivien du Bois de Dunilac
Chemin de l'Oche-l'Abbé 2, 1112 Echichens

Supervisée par :

Prof. Rüdiger Urbanke, LTHC, EPFL
Mr. Christian Vouillamoz, Demaurex SA

Réalisée chez :

Demaurex SA
Avenue du Tir-Fédéral 44, 1024 Ecublens

Lausanne, EPFL, 2020

Abstract

Today, more and more is being produced every day in most aspects of our lives. Industrial production lines are no exception to this trend and most companies are now looking to extract value out of the data that their machines generate. From simply providing their clients with additional statistics to better monitor the behavior of the machines to fully automating large parts of the control of the production facilities, the range of applications that have been created is vast.

In this work we explore three ways of exploiting data produced by robotic pick and place lines used in industrial packaging with delta robots. We first present a regression model that suggests parameters to tune the machines based on their hardware configuration and the characteristics of the products that they need to handle, which unfortunately failed to meet the performance requirements set by our supervisors.
In the second part we discuss a diagnosis tool that lets experts explore data collected on the machines to find out whether or not the machine is performing as expected. It uses pressure sensors to monitor each motion performed by the robot and find those that are not optimal, and searches for abnormal samples in data mined from the machine's software which controls all the mechanical parts and sensors present on the machines. We show experimentally that experts are capable of finding problems on machines on which issues have been created.
Finally we propose a modified version of the AUE2 ensemble stream classifier to perform quality control on the products handled by the machines using pressure data from the machine's vacuum system taken as they pick the products up. Our method adds a way of weighting the classes when training the model, modifies the way the weights of the ensemble members are updated to make it more robust to label noise, and uses convolutional neural networks as base estimators. In our use case no labels are available in the data stream, we therefore use unsupervised outlier detection techniques to partially label it. Our results show that neural networks can outperform Hoeffding trees and Naive Bayes base estimators when the elements of the stream are time series.

Contents

Introduction

As more and more companies are seeking to advance to Industry 4.0, AI and machine learning are the focus of a lot of attention from the industrial world. These techniques have been used in predictive maintenance to predict the failures of specific parts, machines , or systems thus reducing the need for planned downtime and the number of technicians needed to maintain equipments. Detecting problems before they occur also prevents eventual damage that could occur to the machines due to parts malfunctioning during production which increases their lifespan.

Deep convolutional networks are also used in many computer vision tasks in the industry, for instance Audi is using this approach to detect cracks in the parts produced in their press shop. The main advantages of using AI for this kind of application are the speed and the accuracy of the resulting models which can be vastly superior to what humans can achieve. Typical use cases for vision include detecting and tracking objects, and quality control.

Overall AI techniques are particularly well suited to tasks that necessitate processing very large amounts of data quickly and finding pattern in these large datasets.

In this project we focus on exploring ways of using data generated by Demaurex's robotic pick and place lines used in industrial packaging applications. Demaurex does not have a strong AI expertise yet and does not possess the infrastructure to collect large amounts of data from their machines once they are installed in a production facility. Because of this, the small amount of machines that are produced, and how unique each of them are it is very complicated to produce solutions that work for every machine.

The lack of a large scale data collection infrastructure also prevents us from testing applications in a real production environment without physically going to a client's production facility as the company's workshop is naturally not equipped with the same production equipment.

In the remainder of this report we will present the three use cases that we explored, in Part I we present a regression model that suggests parameters for the machines based on how the machine is built and what types of products it has to process. Our goal was to speed up the process of commissioning the machines for production. We designed experiments to evaluate the model according to the average errors it makes when predicting parameter values and the distribution of the magnitude of the errors, the goal was to obtain a model that performed well on average and to guarantee that it did not make many large mistakes which could cause damage on the machines. Part II discusses a diagnosis tool designed to help understand why machines are underperforming and if some machines that seem to work well can be optimized further. The tool looks for unusual patterns in the data generated by the machine and presents it to the user who can then judge if they are concerning or normal. When available the tool also uses sensors to measure the air flow in the vacuum system of the machine which allows us to know precisely when the robots hold a product or not, letting us observe which of the robots' motions are failures. We evaluate the performance of this tool by creating problems on a machine then having experts use it to diagnose these problems. Finally, Part III presents a way of performing quality control on the products that are processed by the machines using stream classification techniques on the data collected by pressure sensors. This is made challenging by the fact that the machines' vacuum system changes over time as the components age and become dirty, and as the machines' parameters are changed by the operators. We test this quality control system by simulating the aging of a machine as it processes products that are known to be good and others that are known to be defective.

1 Overview of the machines

In this chapter we will present the machines that are built by Demaurex SA to familiarize the reader with the way they operate, the tasks they perform, and the problems that need to be addressed when designing and operating them. Section 1.1 explains the different applications in which the machines are used. In section 1.2 we talk about the delta robot which the company is specialized in and is found in most of their products. Section 1.3 contains a discussion of the various methods of detecting products that move through the robotic line. In section 1.4 we present an overview of how the robot's trajectories are planned and of the parameters that the operators can manipulate to optimize the process.

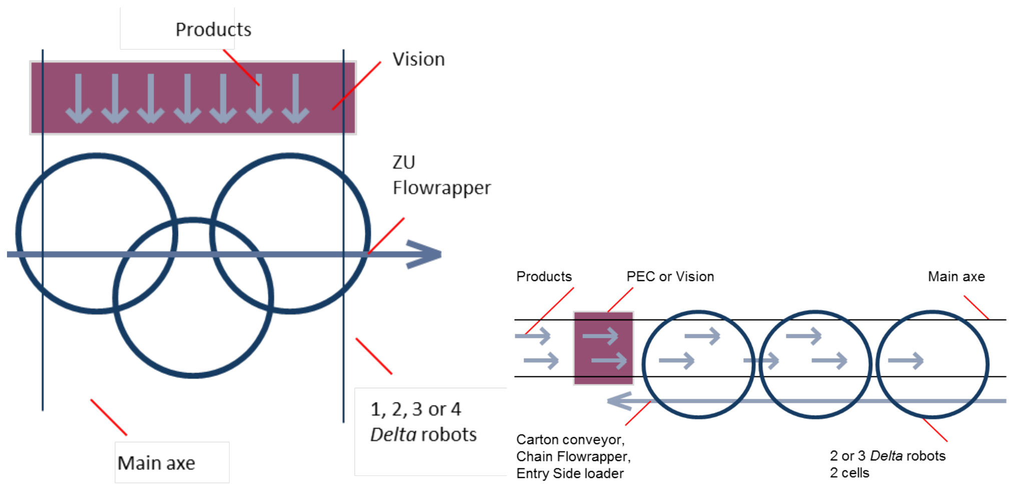
1.1 Applications

Demaurex SA designs and installs pick and place robotic lines for industrial packaging of any sort of products. The products can have any shape or weight and can be placed in boxes or cleat chains that lead to wrapping machines or piled on conveyors to later be pushed sideways inside a box by another machine. The machines can process products that have already been packaged as well as raw ones, i.e. they can perform secondary as well as primary packaging. The most common use case is packaging of industrially produced food items such as cookies, chocolates or pastries but Demaurex machines have also processed light bulbs, sponges, syringes and many other types of products.

Industrial packaging is a very competitive sector and the main considerations of the clients when they decide to buy a machine are its cost and its efficiency. Clients usually expect to offset the cost of buying the machine in the two years that follow the purchase and the efficiency of the machines is key to fulfilling these objectives, making it one of the main focus of research and development in this field.

During their lifetime, packaging lines are often modernized or extended to improve the rate at which they can work or fix unforeseen issues, these modifications often imply a research and development process and the solutions that are initially developed for a particular case can become standard options for future machines.

Currently the company proposes five standard machines that are adapted to particular use cases. The Delfi and Paloma are designed to efficiently manipulate products arriving on a single conveyor. A vision system is placed at the beginning of the work area of the robots to detect the position and orientation of the products which are then placed on an output belt, in boxes or in a cleat chain that lead to the next step of the processing. Their modular architecture allows them to be placed one after the other without any adaptation making expanding the production capability of the facility easier. The main difference between the



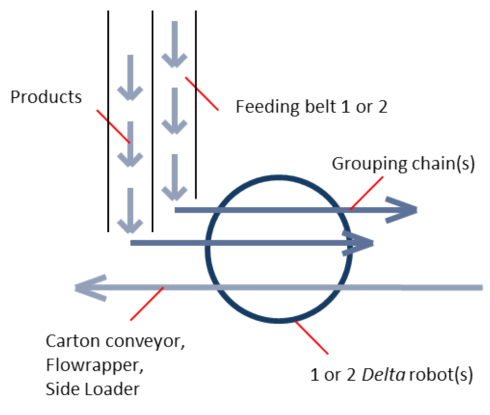
(a) Delfi (b) Paloma

Figure 1.1 – The layouts of the Delfi and Paloma machines. The black circles represent the work volumes of the robots and the arrows show the path of the products.

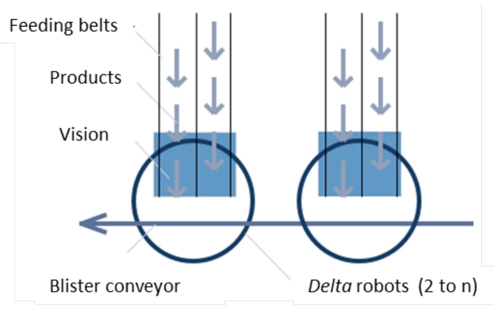
two types of machines is that the output of the Paloma is parallel to the input flow while the Delfi's is perpendicular to its input as shown in Figure 1.1. This makes the Delfi slightly faster than the Paloma because the robots need to travel less distance to place the products; with three robots the Delfi is able to perform up to 360 motions per minute while the Paloma can only reach 290.

The Presto design uses a feeding and grouping unit to ease the job of the robot. Since the location and orientation of the products are fixed by a cleat chain the robot can easily pick as many products as needed to satisfy the client's needs in a single motion. The performance bottleneck becomes how fast products can be inserted into the chain, using a single chain and a single robot up to 500 products can be processed per minute.

The Astor machine is made to fill blisters with many different types of products. A central belt carries the blisters through the working volumes of the robots while conveyors coming from the side bring the products to be placed inside. In theory an infinite number of such machines could be chained of after the other to handle any type of product, in practice the largest Astor lines count sixteen robots where each robot can process up to 130 products per minute. The



(a) Presto



(b) Astor

Figure 1.2 – The layouts of the Presto and Astor machines. The black circles represent the work volumes of the robots and the arrows show the path of the products.

layout of the Presto and Astor machines can be seen in Figure 1.2.

1.2 Delta robots

Designed during a PhD thesis at EPFL by Clavel (1991), the delta robot was made to manipulate light objects at high rate of three motions per second. Bonev (2001) provides a summary of the robot's history. The defining features of the delta are the use of parallelograms to link the motors to the end-effector. A parallelogram constrains a straight joint to keep the same orientation as the parent it is linked to, the use of three parallelograms can therefore constrain a platform to remain level with the ground at all times. Attaching the platform in this way also permits attaching the engines to the fixed base of the robot meaning that only pieces that move when the robot operates are the links between the platform and the motors, the platform itself and the end-effector that will grip the products. The total moving mass is kept low which allows the platform to reach accelerations of up to 50G in experimental settings and 12G during industrial operation. Clavel's original design also includes a fourth motor that is connected to the platform by a telescopic bar allowing an additional degree of freedom that was originally meant to allow the rotation of the end-effector to place the products at any angle though more than one telescopic arm can be attached to the platform allowing for example the end-effector to pick up a line of twelve products then separate them in two groups of six in a single motion or to place the products at an angle relative to the floor.

These design characteristics make the delta robot particularly suited for tasks that satisfy the following points: the products are light, a small and therefore simple end-effector can be used, a large quantity of products are made, the precision required to pick and place the products is not too great. The precision constraints stems from the fact that all three axes of the robot need to move at same time to translate the moving platform vertically lessening the precision when compared with a design that only needs one axle to move vertically.

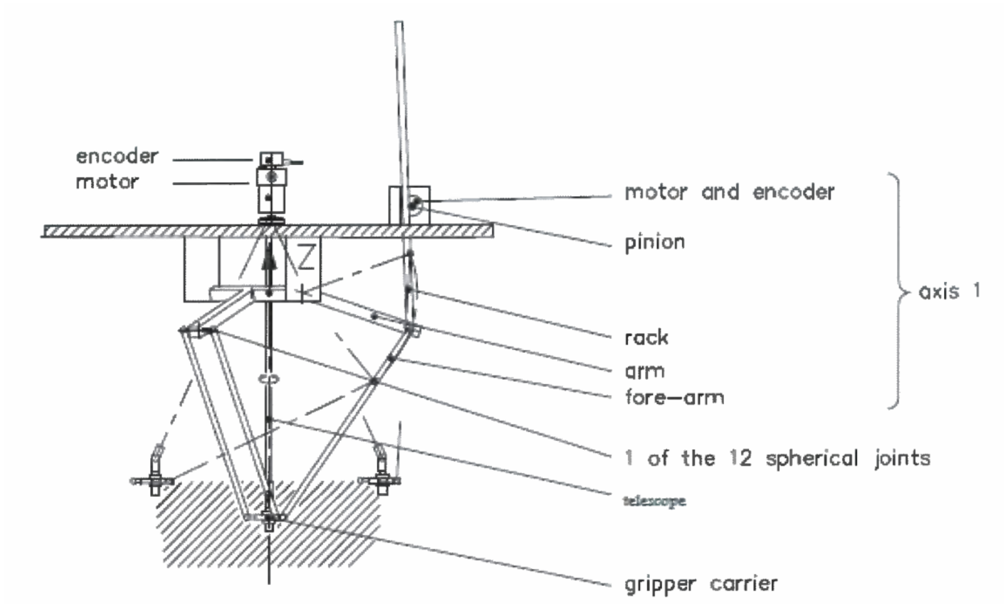


Figure 1.3 – A schematic view of a delta robot, only one of the three motors that control the arms is shown.

Due to mechanical constraints the moving platform cannot reach some points in space. As an additional safety precaution the software used by Demaurex SA restricts the platform to a subset of the positions that it can reach and triggers an emergency stop if one of the robots goes out it. This is called the working volume of the robots.

1.3 Product detection

When products enter a robotic line their position needs to be detected for the robots to pick them up. Depending on the application their orientation might also be needed and quality checks may need to be performed. These necessities are combined with additional constraints resulting from the nature of products that can come from their shape, their color, the way they reflect light, their motifs or even the range of deviations that are considered acceptable when performing quality checks. For a very simple application such as finding a rectangular box of known dimensions a single laser ray can be shot; the duration during which the light is blocked gives a good indication of both the position and orientation of the box. For more complex cases a camera is usually used to detect the products. In this case the choice of lighting is primordial to the application's performance. Colored lights can be used as well as infrareds or ultraviolets. The light source can also be placed behind the conveyor belt to make detecting the contours of the products easier but this approach also makes it impossible to perform some quality checks since the resulting images become black and white. The image processing algorithms that are used are also essential, they usually work by either detecting a shape or clusters of similar pixels called blobs. Shape based methods are better at detecting the orientation of the objects while blob based methods are better at finding their center of mass.

Recently other kinds of sensors have been used to perform quality checks such as pressure sensors to make sure that there are no leaks in the machine's vacuum system which would happen if it tried to pick up a product and failed or if the product was lost during transit decreasing the chance that incomplete boxes leave the line, and other methods that are still in development and are considered secret by the company.

1.4 Computing trajectories

The trajectory of a robot is made of three separate phases: first it moves horizontally to reach its target, then it descends to interact with it and finally it moves back up to prepare for its next movement; the speed profile of the robot during one phase of the trajectory can be seen in fig. 1.4. During a movement phase the robot starts by accelerating at a constant rate given by its accel parameter until it reaches the velocity set by the speed parameter. It then decelerates at the same rate until it reaches its destination. For robots that are equipped with

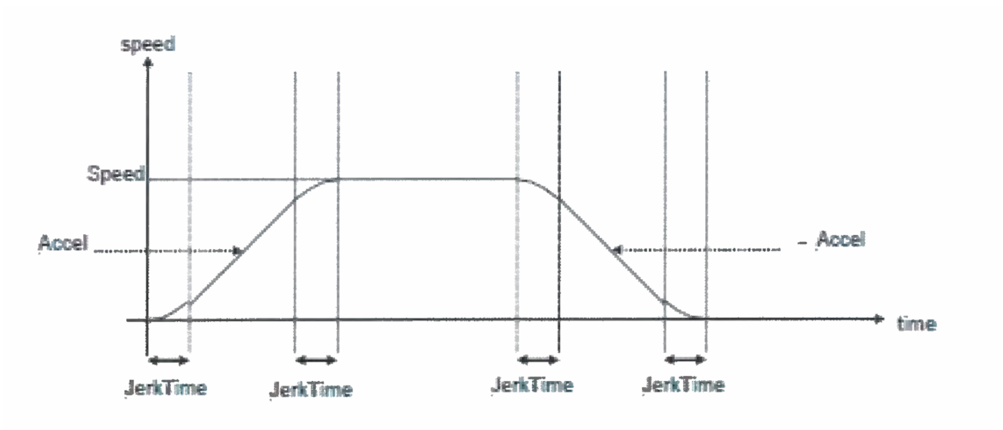


Figure 1.4 – The speed profile of a robot's movement. It is influenced by the acceleration and speed parameters. The jerk time prevents the machine from vibrating too much by limiting the amount of force that is applied at once by the motors.

additional rotation axes the same technique is used to compute the speed profile of those axes with the caveat that all the rotation of all the axes must finish at the same time as the robot's translation. The faster operations are slowed down to match the execution speed of the slowest one. To smooth the movements of the robots they are allowed to start the next phase of their movements before completing the current one. This behavior is called overlap and an example of it is shown in Figure 1.5. For each phase of the robot's movement two parameters control the point at which overlap is allowed to happen: startOverlapDistance and endOverlapDistance. Whenever two phases are scheduled one after the other the controller looks at the endOverlapDistance of the first and the startOverlapDistance of the second, the shortest trajectory that satisfies both constraints is chosen. Electric valves control the flow of

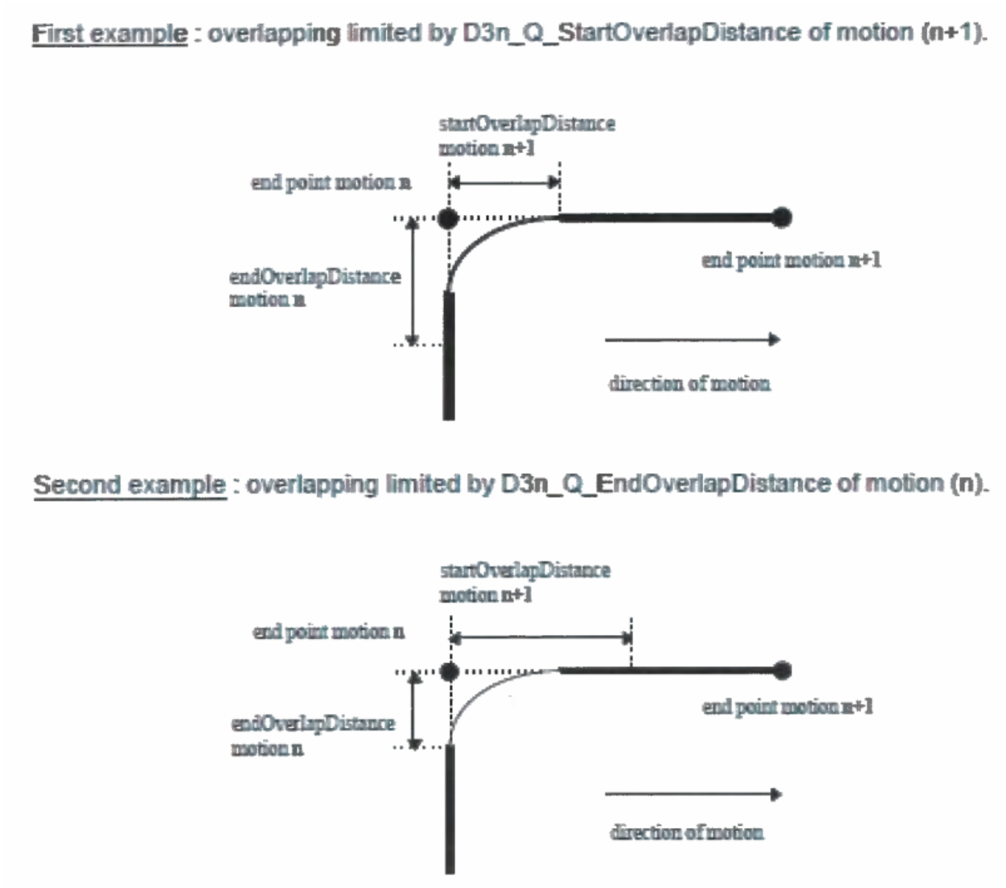

First example : overlapping limited by D3n_Q_StartOverlapDistance of motion (n+1).
startOverlapDistance motion n+1
end point motion n
endOverlapDistance motion n
end point motion n+1
direction of motion
Second example : overlapping limited by D3n_Q_EndOverlapDistance of motion (n).
startOverlapDistance motion n+1
end point motion n
endOverlapDistance motion n
end point motion n+1
direction of motion

Figure 1.5 – The speed profile of a robot's movement. It is influenced by the acceleration and speed parameters. The jerk time is defined in the material configuration and is not editable.

air in the tool that grabs the products. Their activation is normally programmed at the lowest point of the robot's trajectory but because the valves are not perfect and cause some delay between when the signal is sent to activate them and the moment the product can actually be picked up it can be started earlier using the anticipation parameter. When placing a product with a suction cup it is necessary to blow air in the circuit or the vacuum would keep the product stuck to the robot for too long. The duration parameter controls the amount of time that the blowing valve is kept open.

On the machines the robots work in predefined zones that serve to keep them inside their working volume. The zones contain values for each of the trajectory parameters discussed above as well as additional shape parameters that can be used to ensure that the robot will not hit any static obstacles when it moves.

Machine parameter suggestion Part I

One of the tasks that technicians need to accomplish when preparing a machine for a client is setting the production parameters to ensure that the required quality is met Choosing improper parameters could cause the machine to destroy products, fail to meet the required cadence or place products inaccurately which could result in waste if the client's quality control system rejects them or defective products.

To achieve this, they start by creating a production recipe which holds the values of the parameters then they need to test values until they reach the limit of what the product can take without breaking or of what the machine can take before losing accuracy. Since the production recipes do not offer any default values the technicians have to guess where to start based on their own experience. Testing a set of parameters can take several minutes and a dozen parameters need to be set for the robot to pick up then place a product. To try and reduce the time required for this process we created a model to suggest default parameters that would be good enough to only necessitate minor adjustments from the technicians for the machine to function properly allowing them to spend more time working on optimizing the machine.

This part explains the process of building and evaluating the model and presents the results, it is structured as follows. In Chapter 2 the techniques and tools that were used to create the model are presented. Chapter 3 details the process of creating the model and the analysis of the results.

2 Background, regression

Mendes-Moreira et al. (2012) give the following definition of regression:

> With a potentially infinite input space X, the goal is to induce a function $\hat{f} : X \to \mathbb{R}$ that approximates an unknown true function f.

They add that the function $\hat{f}$ called the model or the predictor is obtained by running a learning algorithm on a finite set of data consisting of n training examples for which the true value of f is known called the training set and that since there is no feasible way of measuring the error between the predictor's outputs and the true value of f on the whole domain of the function, the error is approximated on another finite set of examples called the validation set. The error measure can be any distance metric but mean squared error is often used because it is easy to optimize and to compute.

This chapter is structured as follows: In Section 2.1 we give an overview of some of the most used learning algorithms to learn from the training set and explain their strengths and weaknesses. Section 2.2 explores different methods to evaluate the performance of regression models and provide some guarantees on the mistakes they make. Section 2.3 takes a closer look at ensemble methods which aggregate the results of several predictors to provide better predictions. Finally Section 2.4 explains the specific challenges of multi-output regression and some of the techniques that can be used to address them.

2.1 Regression models

One of the oldest regression models is linear least squares. According to Björck (1996) it came as the solution to the problem of fitting a linear mathematical model to given observations. The model can be defined as approximately solving an overdetermined linear system of equations in which a matrix A of size mxn multiplies an unknown vector x such that the result of the multiplication minimizes the mean squared error relative to a target vector b. In mathematical form the problem is to find

$$argmin_x(\sum_{i=0}^{m}(Ax-b)_i^2)$$

One of the advantages of this method is that it is very fast to compute but this simplicity comes at the cost of only being able to model linear relationships between the inputs and the target. To overcome this limitation feature extension methods are used; this means combining the inputs in nonlinear ways and training the model on the modified training set. When using this technique one needs to be careful in selecting the features to add to the training set as some features may be useless and adding too many of them can make the input matrix very large and cause numerical issues. Adding too many features may also cause overfitting, meaning that the model is too close to the training examples which hurts its generalization capabilities. A solution to overfitting is the use of a regularization term λ in the optimization problem, the equation becomes

$$argmin_x(\lambda+\sum_{i=0}^{m}(Ax-b)_i^2)$$

Lambda can be any measure but some choices are more popular: when using the L2 norm of x as discussed by Rifkin and Lippert (2007) we talk of ridge regression which tends to shrink the coefficients of x yielding a simpler classification model and when using the L1 norm we talk of lasso which tends to give a solution with fewer nonzero coefficients.

Another useful model is the support vector machine as described in Vapnik (1995). Originally designed for classification problems the SVM tries to find the best possible hyperplane to separate the different classes of data. The quality of the hyperplanes is determined by the distance between the hyperplanes and the data points that are closest to them, the portion of space that is situated between these data points is called the margin. When applied to regression this technique is called support vector regression as explained by Smola and Schölkopf (2004). The idea is to find a function $f(x) =< w, x > +b$, where <., .> is the dot product, that has an L1 distance of at most ϵ with respect to all the examples of the training set, i.e. the margin is at most 2ϵ wide, while being as simple as possible. The measure of the solution's simplicity is the L2 norm of the weight vector w. This model guarantees that we will not make mistakes greater than ϵ on examples that have been seen in the training set and generalizes well for data that is similar to the training examples but its behavior is unpredictable for data that is distant from the training set. Smola and Schölkopf (2004) also show how implicit mappings via kernels can be used to obtain nonlinear models for support vector machines. The idea is that since the algorithm only depends on the results of dot products between data samples one can compute the dot products in a feature space of higher dimensionality than the input space without needing to compute the mapping from the input space to the feature space explicitly. For the details of the computations see the article.

Decision trees can also be used as regression models. They work by creating a chain of if-then-else rules to separate the training examples into small groups that have similar target values then assigning an output value to each group. The training procedure determines the rules greedily by always selecting the best available split, in the case of regression the cost a split is the sum of the minimized errors made by optimizing a cost metric on both sides of the split. The cost metric is usually mean squared error or mean absolute error. To predict a value for a new data point it is tested against the tree's decisions rules to find the group that it belongs to and the value associated to the group is predicted. The main advantages of decision trees

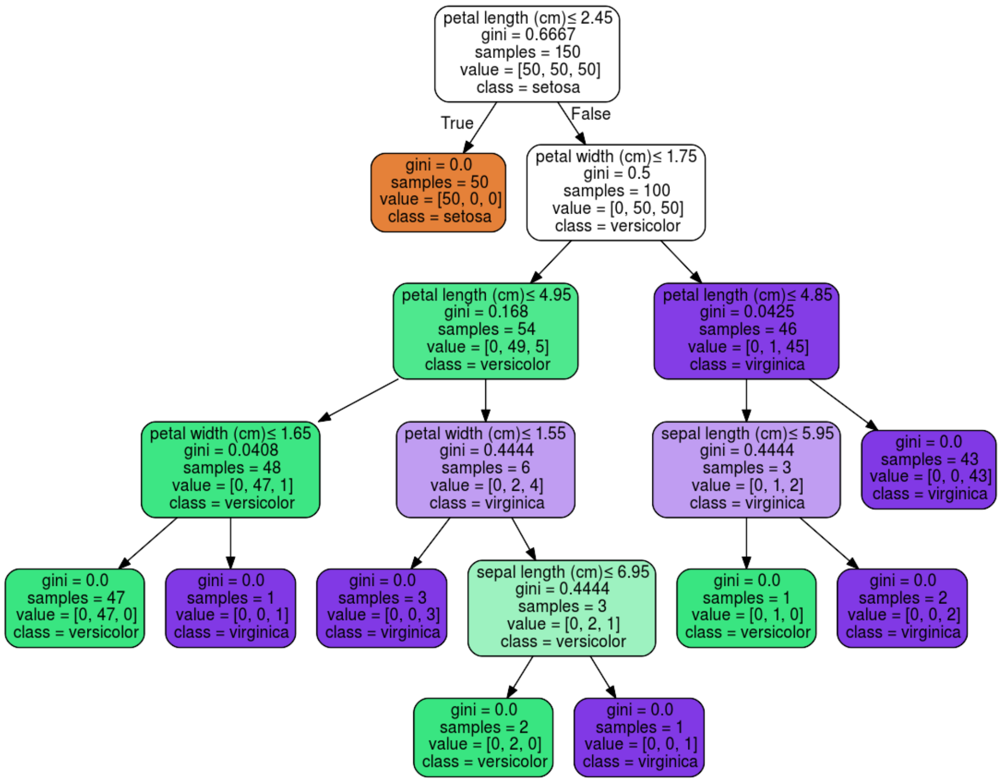


Figure 2.1 – A representation of a decision tree built for the iris dataset, from the scikit-learn user guide.

are that they are easy to understand and visualize as shown in Figure 2.1, and they predict values in logarithmic time in the number of training data points. They are however particularly sensitive to outliers in the training data and tend to overfit. Several methods exist to train decision trees, in this work we use the CART algorithm of Breiman et al. (1984).

The last type of model we will present is the dense feedforward neural network or multi-layer perceptron. It consists of several layers of perceptrons in which every perceptron of layer i is connected to every perceptron of layers $i-1$ and $i+1$ as seen in Figure 2.2. The output of each perceptron is given by

$$\theta(\sum_{i=0}^{n} w_i x_i + b)$$

where x_i denotes the output of the i-th perceptron of the previous layer, θ is any differentiable function, w_i is the i-th element of the perceptron's weight vector and b is the perceptron's bias and the output of the model is given by the output of the final layer. The weights and

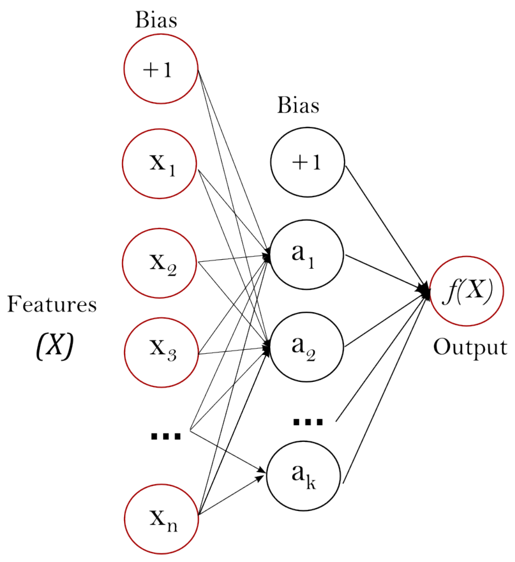


Figure 2.2 – A multi-layer perceptron with one hidden layer, from the scikit-learn user guide.

bias of each perceptron are updated with the backpropagation algorithm which consists of two phases, in the first phase the input is presented to the model which produces an output and in the second phase the output is compared to a target value using a loss function to create an error signal which is then propagated backwards through the layers to update the weights using the gradient of the error signal to minimize it. The loss function plays a crucial role in the optimization procedure since it determines the optimal solution that the model should reach, many such functions have been studied in the literature for different purposes in e.g. Janocha and Czarnecki (2017) and Cherkassky and Yunqian Ma (2004). A more detailed explanation of the multi-layer perceptron and the backpropagation algorithm can be found in Bishop et al. (1995) or any other textbook on the subject of neural networks.

Neural networks are powerful models that have been shown to be able to approximate any smooth function by Hornik et al. (1989) but they also have limitations, they are not able to give accurate predictions for data points situated far away from the training examples, they require a lot of training examples for which the true value of the function they need to approximate is known, and different model architectures must be tested to select one that is adapted to both the approximated function and the available data as explained by Gardner and Dorling (1998).

2.2 Model performance evaluation

Since the goal of regression is to create a model that approximates the value of an unknown function on unseen inputs it is impossible to know for certain which model is the best. Many methods exist to measure the performance of regression models and each one measures a different desirable property that models can have. When training the model it is not always possible to optimize for a given metric on the training data, in such a case one must create many different models then compare them using the desired metric.

Some of the most well-known measures are the following where y is the true value of f, $\hat{y}$ is a prediction, $\bar{y}$ is the mean of the values of y and n is the size of the validation data set.
Mean absolute error: $\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i|$
MAE measures the expected value of the difference between one of the model's predictions and the true value of the unknown function.

Mean squared error: $\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2$
MSE measures the expected value of the square of the difference between a prediction and the true value of f. Both MSE and MAE are indicators of how well the model fits the true function on average, they do not provide any guarantees on how large the errors are or how they are distributed although MSE tends to allow less large mistakes because the errors are squared. To gain more insight into what mistakes the model makes it is possible to use the median or the maximum.

Median absolute error: $median(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$
The use of the median gives a better idea of what a typical error looks like but this metric is never possible to optimize a model using this metric.

Maximum absolute error: $max(|y_1 - \hat{y}_1|, ..., |y_n - \hat{y}_n|)$
Finding the maximum error can give some guarantees about how large the mistakes of the model can be but one needs to be very careful because the testing data needs to be representative of every situation the model might face in practice to ensure that this metric is meaningful.

Coefficient of determination, R^2 score: $1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$
The R^2 score represents the proportion of the variance of y that is explained by the model, it is computed as the ratio of the sum of the model's squared errors and the sum of the squared errors of a model that always outputs the mean value of y, $\bar{y}$.

All of these scoring methods share the same shortcoming, they require a dataset that is representative of the true situations that the model will face in order to give good information. Since labeled data can be difficult and expensive to get it is important to use the amount that is available effectively. To this end it is usual to split the full dataset into k pairs of training and validation sets either randomly or in a stratified way taking for instance the first ten samples as the validation set for the first pair then samples eleven to twenty for the second and so on. A model is then trained for every pair of data sets and the results can be aggregated to compute useful statistical measures about them.

Despite all the methods that allow us to quantify and limit the errors made by regression models engineers are still very reticent to rely on them in tasks where safety is critical such as autonomous driving or robotics because providing actual bounds on the quantity and magnitude of errors made by the models relies on strong assumptions and using specific models as described by Lederer et al. (2019). Often engineers rely on empirical studies such as this study on the performance of deep learning based vision techniques applied to highway driving by Huval et al. (2015).

2.3 Ensemble methods

Mendes-Moreira et al. (2012) define an ensemble method as one that generates several models, each of them obtained by applying a learning process to a given problem, then integrates them in some way to obtain the final prediction. They also provide a good overview of the most common ensemble learning methods that are applicable to regression, among those we will be particularly interested in bagging and random forests.

Bagging consists of training different models on different subsets of the training data created by randomly choosing data points with replacement from the original dataset. This gives a collection of models that are specialized to handle their subset of the data and not the rest of it, the function that results from averaging the results of the members of the ensemble can be more complex than that of a single model trained on the whole dataset.

Random forests are collections of decision trees that have had randomness introduced in their training algorithms, instead of evaluating all the possible splits and choosing the best every time they either look at a random subset of possible split thresholds or features to consider when splitting. A more random way of creating decision trees called extremely randomized trees has been proposed by Geurts et al. (2006) specifically to be used for ensemble methods.

2.4 Multi-output regression

Multi-output regression is a specific case of regression in which we need to approximate a function $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}^d$ where d is strictly greater than one. Compared to the case where d is equal to one, most of the model creation techniques can be modified to profit from the dependencies between the different output variables and the model performance evaluation techniques can be adapted to provide information either for each output separately or for all of them at once. A survey of the multi-output regression techniques has been written by Borchani et al. (2015) in which they present the most important developments of the field, among those we will be particularly interested in the following.

Multi-target regressor stacking
MTRS is a two stage process, the first stage consists of training d models, one for each output variable, in the second stage we train a model that takes as input the features of the training set and the predictions of all the models of the first stage. The idea is that the model of the second stage will correct the predictions of the first stage by observing the relations between the outputs.
Regressor chains

The RC method is similar to MTRS but the process is incremental. First an ordering of the output variables is chosen randomly then a model is trained for each output in the chosen order. Every model is trained on the features of the training set and the predictions of all the models that have been trained before. Because some orderings perform better than others, regressor chains are often used in ensemble methods to improve performance, in this case they are called an ensemble of regressor chains.

Multi-output support vector regression

Support vector regression was designed as a single output method and because of this it is rare to find an implementation that natively supports multi-output problems, for instance LibSVM (Chang and Lin (2011)) does not offer this feature. Modified SVR algorithms that support multi-output problems are an active area of research and many solutions have been proposed but none has become a widespread standard yet, for references to these papers see Borchani et al. (2015) One way to use SVR for multi-output problems is to transform the training set by adding indicator variables for each output. If the training set consists of n samples with d features and m targets, the transformed set consists of nm samples with $d+m$ features and a single target as follows:

$$(x_i, y_i) \rightarrow (I_1, x^i, y_1^i), ..., (I_m, x^i, y_m^i)$$

for each of the n pairs of features x^i and targets y^i where I_x is an m dimensional vector in which the x-th element is one and all the others are zero. The resulting training set can then be used to train a standard one output SVR model.

Multi-output regression trees

When creating a regression tree the CART algorithm evaluates possible splits using mean squared error which can easily be extended to the multi-output case by averaging over the errors the model makes for each target variable.

Artificial neural networks

Similar to decision trees neural nets can be optimized using the gradient of the mean of the loss function over the output variables.

Once we have built a multi-output model we need to measure its performance, this can be done by averaging the results of single output performance metrics computed for each target variable. To ensure that the scale of the targets does not affect the results too much the results are usually divided by the mean of each target. Alternatively one can keep the scores of each target separate and report them all instead of averaging, this allows for more precise evaluation of specific targets which can be useful if some are more important or safety critical than others.

3 Experiments

To create a parameter suggestion model we need to obtain data about the parameters of existing machines, the products that they process, and their hardware configuration. The company's archives contain backups of the machines' software which include the parameters used for every type of product that they work with in xml files called recipes. Information about the products is less standardized and can usually be found in the archives in the form of .docx files or Excel spreadsheets. In some cases the client did not send samples for tests or they arrived late and the files were not updated or they only sent some of the products so only partial data is available, for machines that are made for new production facilities the products may not even exist at the time of the machine's construction. It can also happen that the machine was not optimally tuned when it was delivered to the client since some problems will only occur when the machine is being used intensively for long periods of time. Because of this we decided to use the latest available backup every time, this ensures that the parameters are good enough to be used in production but there is no guarantees that the client did not change the way they make their products. The lack of standard for product data and the fact that the only way of linking it with parameters is the name of the recipe file we had to enter this data manually.

Informations about the hardware configuration of the machines are not available through the software backups and we had to access another database to obtain them. They include the type of conveyor and vacuum pump that were used, sometimes no pump is used and air flow is created using the Venturi effect and an air compressor..

3.1 Setup

Data samples are created separately for each robot to account for the fact that they do not always have the same role depending on where they are in the machine. The samples are cleaned of outliers using an isolation forest (Liu et al. (2008)). The dataset is also split in two parts for which separate models are trained depending on whether the robot is picking up, or placing a product because unladen robots can go faster.

The final dataset includes the following inputs: **machine_type**: the type of the machine as described in Section 1.1, **nb_robots**: the number of robots in the machine, **robot_id**: the id of the robot, **flow_kind**: the type of conveyor, **pump_type**: the type of pump, can be Venturi, vane pump or piston pump, **p_size**: the length, width and height of the product, **p_weight**: the weight of the product, **p_type**: the type of the product, e.g. chocolate or biscuit, **packaging**: the type of package that the product comes in, e.g. none, flowpack, **ppm**: the nominal cadence of one robot in products per minute.

As well as the following outputs: **speed, accel**: the acceleration and maximum speed for the horizontal part of the motion, for vertical parts they are called **upspeed and upaccel** when ascending and **downspeed and downaccel** when descending, **toolspeed, toolaccel**: the acceleration and maximum speed of the end effector's rotation, **anticipation**: controls how soon the valves will be commanded to change position, **duration**: when placing, controls how long air will be blown in the tube to break the vacuum, **waiting**: controls how long the robot will wait after finishing its move, **overlap, exitoverlap**: control how much different movements can overlap to smooth the robot's trajectory

3.2 Models and performance evaluation

The following model types were used:

Ridge regression: with polynomial feature expansion of degree two, the strength of the regularization term was set to 1.

Neural network (MLP): using ten hidden layers with one hundred neurons each and training with mean squared error as the loss function. We use a learning rate of $1e^{-4}$ and an L2 regularizer on the weights of the layers with magnitude $1e^{-4}$.

Random forest: using CART trees (Random Forest) and extremely randomized trees (Extra Trees) as base estimators, the number of estimators and the depths of the trees were set to 500 and 23 experimentally.

Multi-output support vector regression: we extend SVR to the multi-output case by training multiple single target models (SVR-mult), using regressor chains (SVR-chain), and regressor stacking (SVR-stack) using a neural network with the same architecture as the MLP model as the model for the second stage. With the epsilon and tolerance parameters set experimentally to 0.05 and 0.001.

To build the models we used tensorflow (Abadi et al. (2015)) for the neural networks and scikit-learn (Pedregosa et al. (2011)) for all of the others.

To evaluate the performance of the models we are interested in two things. First the models must on average give good predictions and make small mistakes and second the models must not make large mistakes. The second condition is only true for the parameters that control the

	Ridge	MLP	Random Forest	Extra Trees	SVR-Chain	SVR-Mult	SVR-Stack
Pick	0.105	0.0923	0.0821	**0.0638**	0.112	0.112	0.0945
Place	0.205	0.165	0.158	**0.146**	0.209	0.207	0.181

Table 3.1 – Relative mean absolute error of the different models.

	Ridge	MLP	Random Forest	Extra Trees	SVR-Chain	SVR-Mult	SVR-Stack
Pick	0.0685	0.0431	0.0381	**0.0257**	0.0713	0.0754	0.0426
Place	0.207	0.109	**0.0991**	0.100	0.192	0.196	0.111

Table 3.2 – Relative mean squared error of the different models.

speed and acceleration of the robots because values that are either too low or too high can cause mechanical issues and damage the robots. We measure these two qualities using the relative mean absolute error, relative mean squared error, and the empirical quantiles of the relative absolute error for every output on the validation set.

To make sure that the comparisons are as fair as possible we use the same three random splits of the data into training and validation sets for every type of model.

3.3 Results

Table 3.1 shows the relative mean absolute error of the models and Table 3.2 shows the relative mean squared error, both are averaged over all the target variables. We can see that the extra trees model performs best, closely followed by the random forest, then by both the MLP and the SVR-stack. This seems to indicate that tree based models give overall better predictions than the other methods.

Table 3.3 and Table 3.4 report the fraction of samples from the validation set for which the different models obtain a relative absolute error of more than 0.5 for picking and placing motions respectively, this threshold represents the limit after which a prediction is considered bad. Once again the extra trees model performs better than the others, for this metric the MLP, random forest and SVR-Stack models are closer to the performance of the best model.

We believe that the good performance of the tree based models is due to the nature of the dataset, since Demaurex produces relatively few machines every year and because each machine includes some uniques pieces, many configurations are truly unique. The lack of training samples in some areas of the data hurts the other models more than the trees because their outputs are naturally constrained to the range of values they have seen in the training set whereas ridge, MLP, and the SVR based methods can output any value.

Since the extra trees model had the best performance on the two selected metrics we present the distribution of the error the model makes in more details in Figure 3.1 and Figure 3.2. We showed these results to the chief of engineering and the chief of the workshop who decided

	upspeed	upaccel	downsp	downac	speed	accel	toolsp	toolac	ant	wait	overlap	exitover
Ridge	0.0129	0.207	0.0142	0.0220	0.207	0.00518	0.0259	0.0272	0.0543	0.169	0.0660	0.0375
MLP	0.0168	0.00647	0.0181	0.00776	0.0142	0.00614	0.0246	0.0220	0.0582	0.128	0.0582	0.0440
Forest	0.0181	0.00776	0.0181	0.00776	0.0129	0.00259	0.0233	0.0220	0.0466	0.128	0.0569	0.0285
Extra Trees	**0.00647**	**0.00518**	**0.00518**	**0.00518**	**0.00776**	**0.0**	**0.0207**	**0.0155**	**0.0362**	**0.0931**	**0.0440**	**0.0129**
SVR-Chain	0.0233	0.00906	0.0323	0.0259	0.0298	0.00388	0.0362	0.0401	0.0919	0.153	0.102	0.0944
SVR-Mult	0.0259	0.00906	0.0298	0.0323	0.0336	0.00259	0.0349	0.0479	0.105	0.177	0.150	0.138
SVR-Stack	0.0207	0.00647	0.0168	0.00905	0.0129	0.00129	0.0207	0.0232	0.0517	0.134	0.0595	0.0427

Table 3.3 – Fraction of validation samples for which the relative absolute error is larger than 0.5 for the different models for picking motions.

	upspeed	upaccel	downsp	downac	speed	accel	toolsp	toolac	ant	wait	overlap	exitover
Ridge	0.0341	0.0495	0.0310	0.0851	0.0418	0.0201	0.0743	0.0495	0.135	0.228	0.125	0.0728
MLP	0.0232	0.0232	0.0294	0.0666	0.0201	0.0201	0.0929	0.0573	0.115	0.211	0.0820	0.0557
Forest	0.0232	0.0248	**0.0201**	0.0650	**0.0155**	0.0263	0.0867	0.0433	**0.105**	0.181	**0.0728**	0.0573
Extra Trees	**0.0217**	0.0294	0.0294	**0.0619**	0.0186	0.0263	0.0851	**0.0418**	**0.105**	**0.173**	**0.0728**	**0.0495**
SVR-Chain	0.0480	0.102	0.0573	0.153	0.0495	0.118	0.136	0.203	0.156	0.193	0.181	0.153
SVR-Mult	0.0418	0.102	0.0557	0.159	0.0433	0.0248	0.141	0.209	0.150	0.192	0.178	0.159
SVR-Stack	0.0248	**0.0217**	0.0263	0.0743	0.0248	**0.0186**	**0.0588**	0.0480	0.107	0.201	0.0774	0.0650

Table 3.4 – Fraction of validation samples for which the relative absolute error is larger than 0.5 for the different models for placing motions.

that the performance of the models was not good enough and this part was abandoned.

3.4 Conclusion

In this part we presented how we built and evaluated a regression model to suggest default machine parameters using the machine's hardware specification and the characteristics of the products it has to handle.

This part of the project was abandoned after reviewing the performance of the models with the company's chief of engineering and head of the workshop.

We believe that the poor performance of the models is due to the small quantity of data available, the fact that the machine's hardware changes rapidly over the years, the data of the products handled by the machines not being available in a format that can be parsed automatically which forced us to enter it manually, and the fact that old and unused recipes were sometimes present in the backups of the machines which caused noise in the labels of our dataset.

Additional work could focus on suggesting what type of lighting and end-effector should be used for a given product, and on objectively quantifying the characteristics of the products in a way that captures enough information for these tasks.

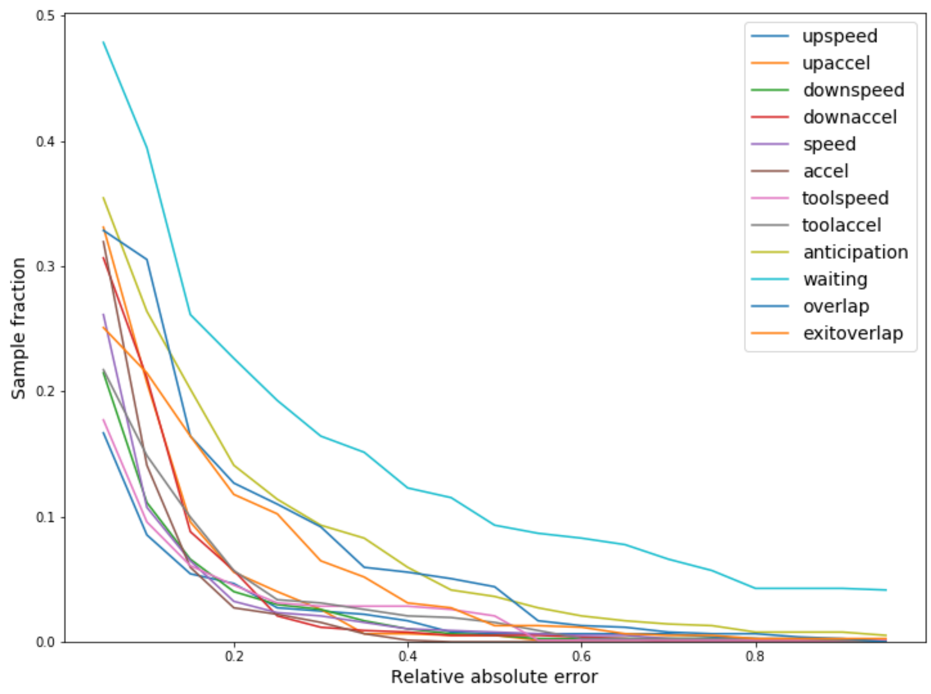


Figure 3.1 – A visualization of the distribution of the magnitude of errors made by the Extra Trees model for picking motions. The values on the y-axis are the fraction of samples in the validation set for which the relative absolute error is larger than the corresponding value on the x-axis.

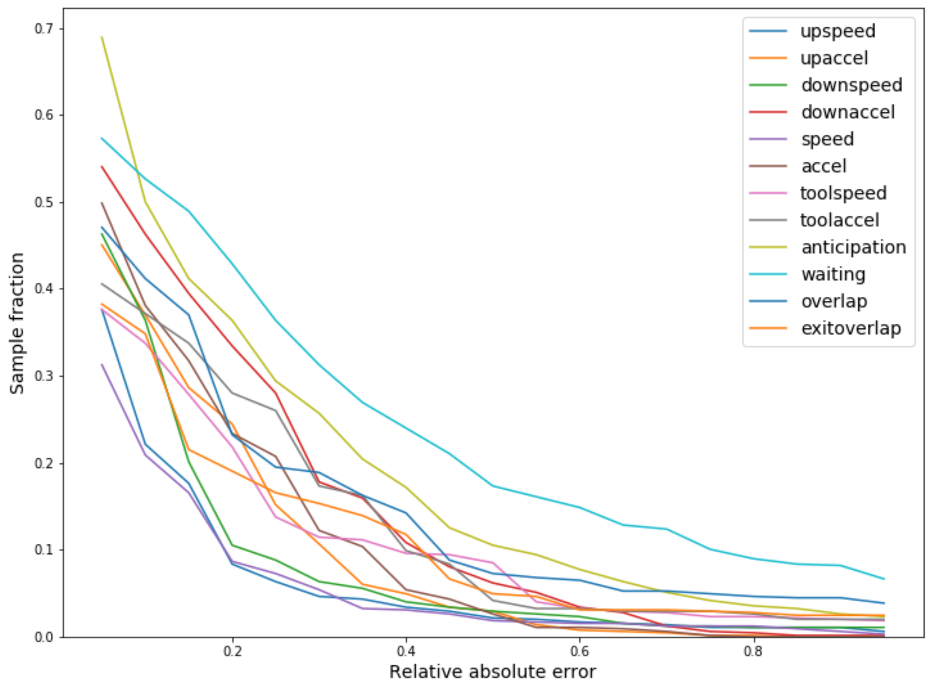


Figure 3.2 – A visualization of the distribution of the magnitude of errors made by the Extra Trees model for placing motions. The values on the y-axis are the fraction of samples in the validation set for which the relative absolute error is larger than the corresponding value on the x-axis.

Machine diagnosis Part II

The machines produced by Demaurex are complex and many pieces of software and hardware from different sources need to interact for them to function. The cause of failures can therefore come from many different places, for example a sensor might get dirty and send erroneous data, the vacuum system can get dirty and lose some power, a suction cup can have holes, or some mechanical part of a conveyor might break causing the machine to think the products are in an another place than they really are. The clients are also not experts and often give confused or wrong explanations of what is going wrong. Because of this it is often necessary for the company to send a technician to assess the situation before they can even begin to think of a solution.

Being able to recognize patterns in the machine's behavior when products are not handled correctly could help the company fix machine failures faster and more efficiently by finding which part behaves differently.

One of the new features that has been integrated in the newest version of the machines' software is the vacuum check, it is able to detect whether or not a product is being transported by the robot at any given time by measuring the air flow inside the vacuum system, a high flow when the valves are open means that the product was not picked up or lost during transit. Using labels provided by this feature we built a classification model that takes as input as much as possible of the data that is generated by the machine while it runs to find explanations for the failures.

With a more sophisticated method that could detect smaller mistakes in the way the robots handle the products, such as not picking a product on its center of mass, it would also be possible to optimize the machines and gain efficiency by reducing the number of products that are misplaced. To this end we investigated ways of finding unusual behavior without using the vacuum check's output as labels.

This part explains how we built interpretable classification models using the vacuum check's data, how we searched for anomalous behavior in the results without labels, how we looked for soboptimal robot motions using raw pressure data from the vacuum check's sensors and how all of this was put together as a tool to assist with debugging and optimizing the machines. Chapter 4 provides an overview of previous work done on similar subjects and Chapter 5 details the process of creating and evaluating each part of our solution.

4 Background

In order to build the diagnosis tool we had to use techniques from different fields, Section 4.1 presents the problem of binary classification, the models that were used in the project, and the visualization techniques that we used to obtain insights from the models. Section 4.2 explains how we searched for anomalies in the data extracted from the machines which includes outlier and novelty detection.

4.1 Binary classification

Classification is similar to regression in that we are looking to approximate a function $f(x)$ but instead of a continuous output i.e. $f(x) : \mathbb{R}^m \rightarrow \mathbb{R}^n$ we have a discrete output i.e. $f(x) : \mathbb{R}^m \rightarrow \{0,1\}^n$ where a value of one at the i-th position of the output indicates that x belongs to the i-th class. For our use case of deciding whether a robot motion was good or bad we have only two mutually exclusive classes which is a special case called binary classification, many algorithms can be simplified when working with binary classification because we do not have to worry about data points belonging to more than one class at once and we can use a single output value instead of one per class. This allows us to optimize a model $\hat{f}(x) : \mathbb{R}^m \rightarrow [0,1]$ then round its output to get the prediction of the class of x.

4.1.1 Models

Logistic regression is a linear model that is used for classification. It differs from a regression model in that it imposes a limit on the outputs of the model using the logistic function $g(x) = \frac{1}{1+e^{-x}}$. The training procedure minimizes the following cost:

$$\lambda + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

where X is the matrix of training samples, y is the array of labels in the set $\{-1,1\}$, w and c are the parameters of the model that are being optimized, and λ is a regularization term on the parameters of the model, usually the L1 or L2 norm of the weight vector w. The predictions of the model are computed in the following way: $p(x) = g(x^T w + c)$ where g is the logistic function. As with other linear models discussed in Section 2.1, logistic regression can learn nonlinear relationships between the input features using feature extension.

More details are available in Kleinbaum et al. (2002) or any other textbook on the subject.

In Section 2.1 we explained that **support vector machines** can be used for classification problems. In the case of binary classification the SVM tries to find a hyperplane that separates the data samples according to their classes while maximizing the distance between the hyperplane and the samples that are closest to it.

Decision trees were introduced in Section 2.1 as regression models but they can also be used for classification, the only difference in the procedure is the cost metric that is used to determine where to split the data. The CART algorithm uses Gini impurity defined for a set of elements as $1 - \sum_{i=1}^{n} p_i^2$ where n is the number of classes and p_i is the fraction of items labeled with class i in the set, it reaches the minimal value of zero when all elements are of the same class. Other algorithms including C4.5 use the information gain defined as the difference between the entropy of the full set of elements and the sum of the entropies of the split set: $H(t) - (H(t_1) + H(t_2))$ where H is the entropy defined as $-\sum_{i=1}^{n} p_i log_2 p_i$, t is the full set, and t_1 and t_2 denote the two halves that result from the split that is being evaluated.

Neural networks can also be used for classification, depending on the loss function that are trained to optimize they can solve an equivalent problem to logistic regression or support vector classification.

4.1.2 Interpretability

Many metrics and visualization techniques have been created to assess the performance of classification models and allow us to select and compare different models such as ROC curves or precision and recall graphs. In the context of diagnosis however it is much more interesting to understand how the models make decisions to locate the causes of abnormal behavior. To this end we need ways of visualizing the decision boundaries of different classification models.

Decision trees can easily be visualized as they consist of a collection of if-then-else rules but for other models that build functions in potentially high dimensional spaces like support vector machines, neural networks or logistic regression the decision boundaries can not be drawn in a way that is interpretable for humans. Different approaches have been proposed to allow for more interpretability of classification models: Zhiyong Yan and Congfu Xu (2008) designed a method that learns a transformation that reduces the dimensionality of the data to two dimensions using self organizing maps (Kohonen (1997)) to plot the decision boundary of the classifier in the transformed space which has the inconvenient of masking the original names of the features in the plot; Elmqvist et al. (2008) built a system that allows a user to explore a dataset using 2D scatter plots of the data for different projections; Migut et al. (2015)

proposed a way of combining previously existing methods by generating a 2D scatter plot by letting a user select two input features to plot then approximating the decision boundary using a Voronoi tessellation (Aurenhammer (1991)) of the 2D space in which regions containing samples of different classes are filled with different colors. They also use a histogram of the distances of true/false positives as well as true/false negatives to the classification boundary to measure how confident the model is in the decisions it takes.

Specific methods have also been designed for some types of model, Rauber et al. (2017) provide a way of inspecting the neurons of a dense feedforward neural network, Yosinski et al. (2015) give a similar method for convolutional networks, Caragea et al. (2001) used projection based tour methods to gain insight about how support vector classifier make decisions and Hamel (2006) used self organizing maps for the same purpose.

4.2 Outlier detection

Hodge and Austin (2004) state that outlier detection is a broad topic and that many definitions of what an outlier is have been proposed, they use a definition by Grubbs which states:

> An outlying observation or outlier is one that appears to deviate markedly from other members of the sample in which it occurs.

They distinguish three different approaches to outlier detection: determining the outliers with no prior knowledge about the data which amounts to unsupervised clustering, create a model that classifies data as normal or abnormal using a labeled training set which amounts to supervised classification, and creating a model from data that is known to be normal (or abnormal) that is able to tell if new observations originate from the same process or not which is also called one class classification or novelty detection.

When we receive data from a machine that is malfunctioning it is collected from a production environment in which it is not possible to assign labels to individual motions of the robots, therefore the second use case is of no interest to us. We will now present some interesting models that are designed to handle both the first and second cases.

In the case where all of the normal data points are known to be similar to each other it is possible to detect outlier by computing a distance measure between the mean of all the samples and each individual sample assigning them a similarity score . This reduces the problem to a univariate one for which statistical measures can be used to determine a threshold of normality, see Grubbs (1969). If the normal samples occupy several regions of the input space an unsupervised clustering algorithm such as k-means can be used to separate the data into regions, similarity scores can then be computed for each region and samples can then be classified by considering the smallest difference they have with one of the cluster means (Allan et al. (1998)).

Another approach is to use a k-nearest neighbors method and use the distance of each samples to its nearest neighbors as a score of normality which has the benefit of being more general than computing the distance to the mean but is also more computationally expensive.

Liu et al. (2008) proposed a method based on random forests in which a user defined number of random trees of maximum depth $\log_2(n)$ are trained on the input data where n is the number of data samples. The idea is that anomalous samples will require less decision rules to be isolated from the others and the measure of normality of the samples is the average depth at which they are found in the trees.

A modified support vector machine called the one-class SVM was designed by Schölkopf et al. (1999). They optimize the SVM to separate the data points from the origin, creating a function that takes the value $+1$ in a small region containing most of the training samples and -1 elsewhere. The samples are classified as being normal or abnormal based on the sign of the function.

Manevitz and Yousef (2000) create a score for the samples using an autoencoder neural network, that is a network that is trained to approximate the identity function. They use a model that does not have enough neurons to fully learn the identity function in the input space which forces it to prioritize some samples over others. They count on the fact that the model will make larger errors when predicting on anomalies since fewer of them are in the training set. The score for each sample is given by the difference between the model's prediction and the sample itself.

Recurrent neural networks have been shown to perform well for detecting anomalies in continuous time series data by Hundman et al. (2018). They use an LSTM network to predict the values of the signal then smooth the errors made by the model to obtain a continuous anomaly score for the signal which is then thresholded to classify some regions of the input signal as abnormal.

4.3 Performance evaluation

For performance evaluation, classification is similar to regression. It shares the same difficulties with acquiring enough data to get meaningful results and accurately represent all of the situations that the model might face in practice. The main differences come from the ways of scoring the performance of the models because the concept of a large mistake does not exist as samples are either classified correctly or incorrectly. Powers (2008) gives an overview of the different methods that can be used to asses the quality of classification methods.

For the special case of binary classification there are only four possible outcomes for each sample: it can be of one of two classes and it can be classified as one of two classes. When a sample is correctly classified as belonging to the positive class 1 it is a true positive, for the negative class 0 it is a true negative. Similarly a sample that belongs to the positive class but was wrongly classified is a false negative, for the negative class it is a false positive. Many metrics have been proposed to quantify the quality of binary classifiers using the number of occurrences of these four cases, some of the most popular are:

Accuracy is the number of true positives and true negatives divided by the total number of samples.

Sensitivity, also known as **recall** and **true positive rate**, is the ratio of samples that were classified as positive that are really positive.

Specificity or **true negative rate** is the same measure for the negative class.

Precision or **positive predictive value** is the ratio of truly positive samples that were classified as positive.

Negative predictive value is the same measure for the negative class.

The **F1-score** is the harmonic mean of precision and recall: $F1 = 2\frac{\text{precision}*\text{recall}}{\text{precision}+\text{recall}}$. It does not take into account how the model handles the negative class and is therefore better suited to problems where the negative class is not measurable such as information retrieval.

Informedness is the sum of the true positive rate and the true negative rate minus one.

Markedness or **deltaP** is the sum of the positive and negative predictive values minus one.

To assess the performance of a classifier one can also use a Receiver Operating Characteristic curve which consists in plotting the true positive rate against the false positive rate for different parameters and/or models. An example can be seen in Figure 4.1.

Fawcett (2006) gives a good overview of how ROC curves can be used for evaluating the performance of classification models.

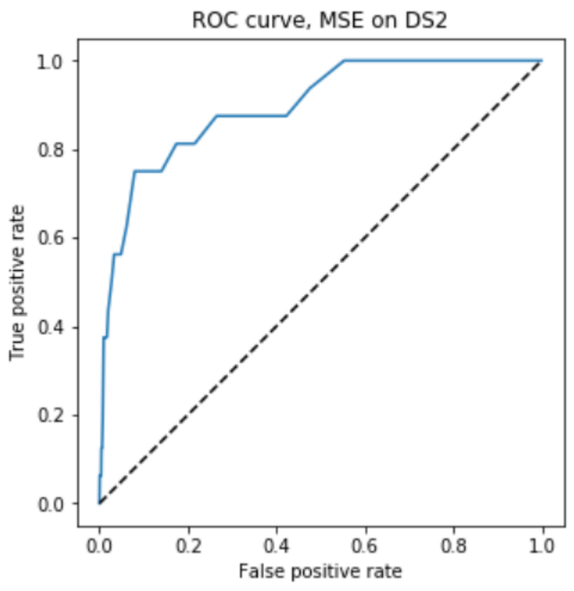


Figure 4.1 – An ROC curve, the blue line represents the scores of the classifier with each point of the curve being the results of using different parameters with the same model, the black line is the expected score of a random classifier.

5 Experiments

The machine diagnosis tool consists of two parts, the outlier detector that points out which robot motions are suboptimal or anomalous and the classifier that tries to explain which part of the machine caused the failure. To test both aspects we need labeled outliers to verify the performance of the detector and data generated by malfunctioning machines to make sure that experts are able to correctly identify the root cause of the problem with the help of the classifier. We chose to test both aspects separately as labeling every motion as good or bad is time consuming and would be impractical if it had to be done when generating large amounts of data for malfunctioning machines. We therefore created two datasets: the first contains pressure curves taken from the vacuum check's sensors labeled as either good or bad where bad motions are those for which the robot failed to pick the product on its center of mass, and the second contains unlabeled pressure curves and associated data generated by machines on which a known defect was purposefully created.

To give a better understanding of the data we will now explain how the vacuum check works in more detail. On a machine each end effector is connected to the vacuum system by a tube, an electric valve controls whether air is sucked or blown through the tube. The vacuum check itself consists of a Venturi cylinder and two pressure sensors placed in the air tube of each end effector as shown in Figure 5.1. The difference between the value measured by the first sensor in the wider part of the tube and the second one in the narrower part gives an approximation of the air flow through the tube allowing to know whether or not a product is currently held by the end effector. Values are taken every two milliseconds

The pressure curves generated by the robot's motions are very similar when they manage to pick up a product, the pressure and the air flow start by increasing in the tube, then the pressure stabilizes and the air flow drops meaning that the tube is sealed by the product that was picked up, when the robot reaches its destination air is blown in the tube to break the vacuum and the pressure falls rapidly until it reaches the level of the pressure outside of the tube as shown in Figure 5.2.

When a product is picked up off center a difference can usually be seen in the 100ms that follow

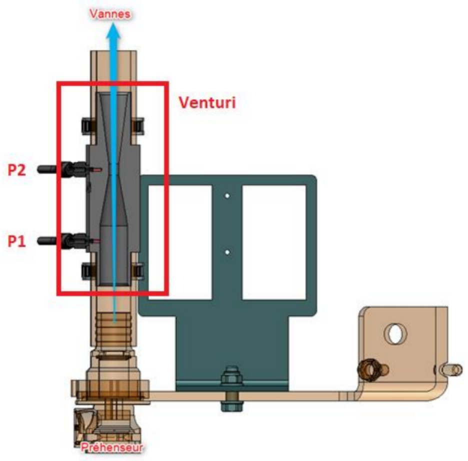


Figure 5.1 – A schematic view of the vacuum check system, the flow of air through the tube is shown by the blue arrow.

the activation of the valve as can be seen in Figure 5.3. The pressure values react differently for every product depending on the type of suction cups and end effector as well as the product's weight, shape and contact surface with the suction cup. Sometimes a difference can also be noticed at the end of the measures when the vacuum is broken.

Associated to the pressure data we also extract values from the machine's software and the vision system that detects the product when one is installed. The exact set of data that is recovered can vary from one machine to another but typical values include the positions at which the product was picked up and placed, the product's area, length, width and perimeter, the time taken by the robot to perform the motion, the speed of the conveyors, and any other measures that can be extracted from the vision and the mechanical parts of the machine.

In this part of the project we used erasers to simulate products because they are similar to some bakery items such as croissants or large biscuits. We taped them two by two to ensure that they were wide enough to be used with the machine that we used for the tests. One of the resulting products is shown in Figure 5.4.

5.1 Setup

To test the performance of the pressure outlier detector we created a dataset consisting of around 1000 good motions and around 600 suboptimal ones by using a well parameterized machine. The suboptimal motions were created by adding a 1.5cm offset to the robot to make it pick the products off center.

To make sure that the pressure curves are all aligned with each other we use a simple threshold to remove the zero values before the commutation of the valve. Correcting for alignment in a distance metric such as dynamic time warping would not correctly model the distance

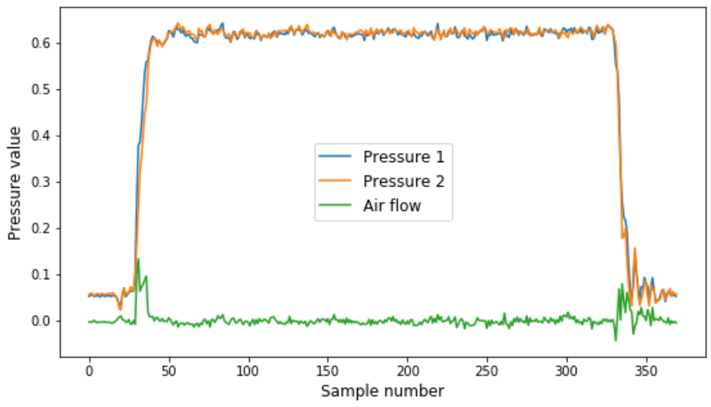


Figure 5.2 – The values given by the two pressure sensors of the vacuum check and the air flow for one motion.

between the curves because the difference between good and bad ones is often time distortion.

Then we train a model to give abnormality scores to every pressure curve in the dataset and plot them to let the user choose a threshold beyond which a motion is considered bad as shown in Figure 5.5. A default value of $Q_3 + 1.7 * \text{IQR}$ is given to the threshold where IQR is the inter quartile range $Q_3 - Q_1$ and $Q_1 and Q_3$ represent the 25% and 75% quantiles of the data. We chose a threshold value that depends on quartiles of the data because it is more robust to those that depend on the empirical mean and standard deviation, the specific value that multiplies the IQR was chosen from the results of the experiments presented in the next section.

After the outlier detection step, the average pressure curve is shown to the user as in Figure 5.6. If old data from when the machine was known to work properly is available the old and new curves are plotted together for comparison.

In addition to the pressure data we also search for outliers in the values extracted from the other parts of the machine such as the vision system because the pressure based detection is not always perfect, some parts may cause very small mistakes when they fail which are not detectable using the pressure values, and the sensors may not be installed on some machines.

Once we have obtained a list of outliers we train a classification model to differentiate between normal samples and outliers. We use forward feature selection as described by Marcano-Cedeño et al. (2010) and let the user choose the number of features to be selected by the procedure. We also let the user choose the scoring metric that is used to select the features between training accuracy and test accuracy and the fraction of samples to be reserved for the test set. The user is also allowed to choose whether or not to balance the data set by removing values from the overrepresented class or to give more weight to one class during the training of the model.

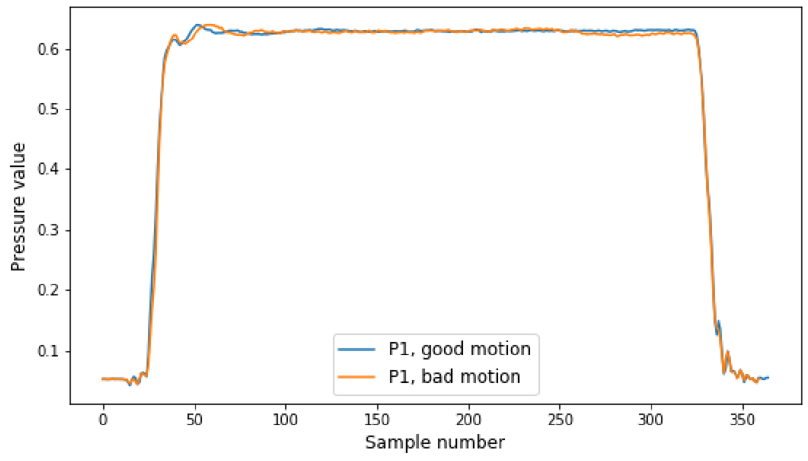


Figure 5.3 – A comparison of the means of pressure curves for good motions and one type of bad motion

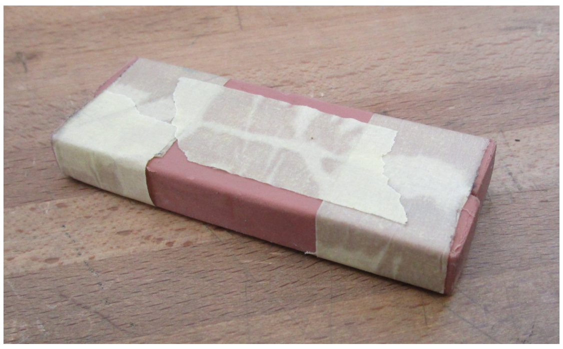


Figure 5.4 – The products used in this part of the project: two erasers stuck together with tape.

The features that are selected and the score differences between the stages of the selection process give a first indication of which components prevent the machine from working correctly.

Finally we use the scatter plot technique of Migut et al. (2015) to give a better idea of how the classifier decides which samples are outliers. We let the user choose two features for which to plot a 2D scatter plot and draw an approximation of the decision boundary using a Voronoi tessellation as shown in Figure 5.7. If the user chooses to use a decision tree in this step then they can also choose to visualize its decision rules in the form of a graph.

5.2 Performance evaluation

The method is divided in three steps, outlier detection in pressure data, outlier detection in feature vectors, and training a model to interpret the data and help and expert figure out what is causing the machine to misbehave. Therefore we need to evaluate the performance of each

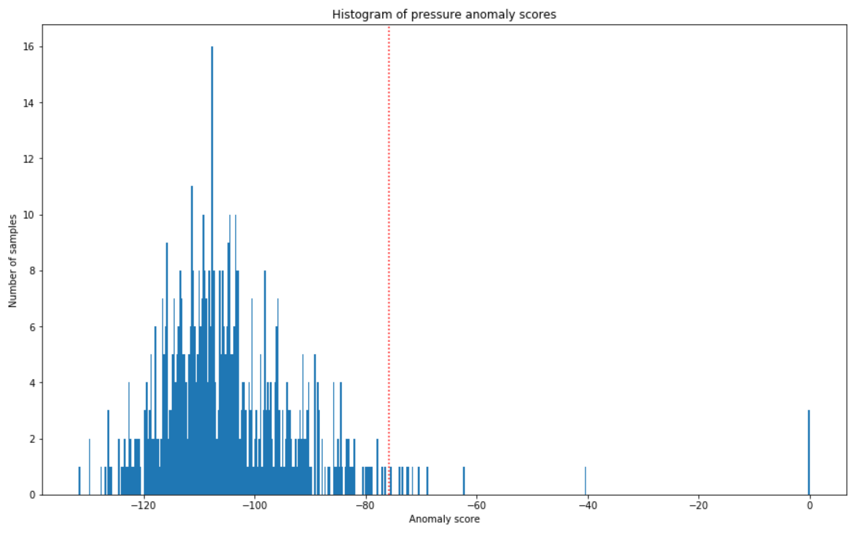


Figure 5.5 – The scores given to the data samples in the outlier detection step presented as a histogram.

of these three parts.

For the first part we are able to generate and label good and bad motions easily by setting offsets on a machine that works well to make it pick products off-center. We can therefore obtain a dataset of good quality for every product that we want to test which allows us to objectively quantify the performance of this step using the methods described in Section 4.3.

By definition we expect the number of outliers in the data to be small. A probable value for robotic pick and place lines is 99% of good motions. We therefore look for classifiers that have a very low false positive rate because if even one percent of good motions are labeled as bad there would be more false positives than actual bad motions in the produced outliers list. Among those we are interested in finding one with a high precision to retrieve as many outliers as possible. Because of this we decided to use ROC curves to rank the models and focus on the points where the false positive rate is less than one percent.

For values extracted from the other parts of the machine we have no way of reliably labeling them as causing failures of the machine. We also do not have the required knowledge to differentiate between rare values that are normal and true outliers that indicate that a part of the machine is not functioning properly and mobilizing experts long enough to create labels for a dataset large enough to compute meaningful statistical measures was not an option. To give an idea of how well this part performs we had to rely on empirical results from experiments detailed below.

The last part of the tool is the classifier that is meant to assist experts in diagnosing the machines. Since it does not give any decision itself we had to score its performance by testing it in scenarios created in the workshop. With the help of a field engineer we created problems

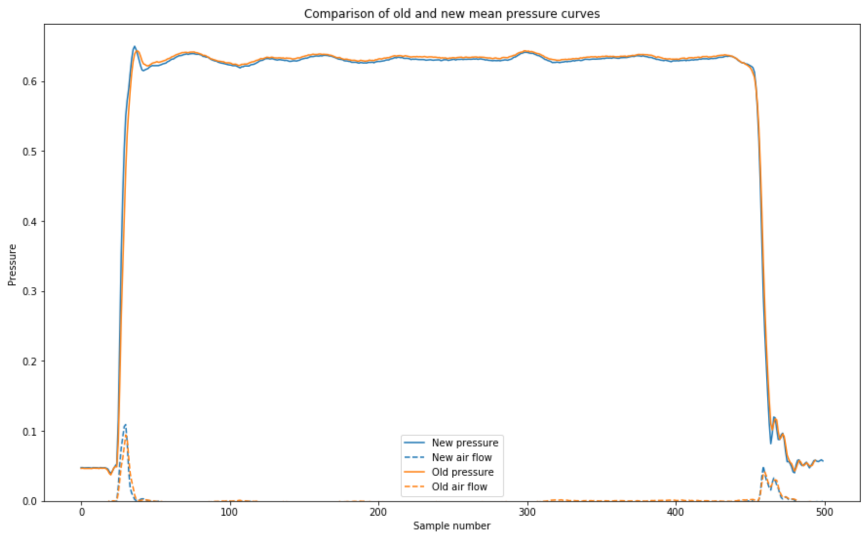


Figure 5.6 – A comparison of the average pressure curves of when the machine was known to work properly and when the experiment took place.

on a machine that resemble those that can happen in a production environment then had some experts that usually perform the diagnosis of the machines use the tool to see if they could accurately find which parts of the machine were modified.

In order to determine how useful each step of the process is and how much extra work is required to install the diagnosis tool on old and new machines we perform the test using four different amounts of available data: the vacuum check can be installed or not, and data collected from a time period when the machine was functioning properly, called reference data, may be available or not. We therefore gave experts seven cases to study with vacuum check data available for three of them. For each case we first asked them to state which part of the machine they thought was modified without having access to old data from when the machine functioned properly then gave them this additional data to see how it changed their perception of the situation.

We consider the expert to have succeeded if they can identify the part of the machine that was faulty between the following categories: mechanical, product detection, calibration, and vacuum system where calibration refers to the matching between the product detection's and the robot's coordinate systems that lets the robot move to pick up the products in the correct position. We also ask them to be as precise as possible when choosing the mechanical or vacuum system categories because these contain many different pieces that can each be malfunctioning but we do not consider it a failure if they can not identify the precise part that is faulty.

We created the following defects on the machine: bad vision coordinate calibration (calib), acceleration parameters too high (accel), dirty camera cache (vis_cache), reduced lighting

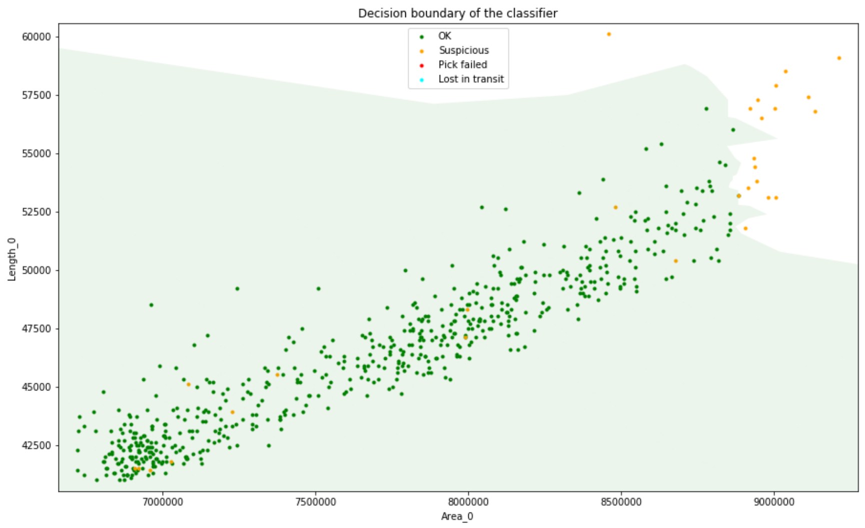


Figure 5.7 – A visualization of the decision boundary of the classifier, the outliers detected in the previous stage are shown in orange. The green region of the graph indicates where the classifier decides that the samples are normal.

(vis_light), reduced air pressure in the vacuum system (vac_pressure), hole in the suction cup (vac_leak), and products moved after being seen by the vision which simulates a defective conveyor belt (meca_belt). Vacuum check data was available for: meca_belt, vac_pressure, vac_leak, and vis_light.

For the experiment we had three participants: a field engineer, a software engineer, and the chief of engineering.

5.3 Models

For the first part of the tool we selected the following models: an autoencoder neural network (IDNN), a one-class support vector machine, an isolation forest, and a simple classifier that computes the MSE between the mean of all the pressure curves and each curve individually to score their normality. The neural networks are built using tensorflow (Abadi et al. (2015)) and the other models using scikit-learn (Pedregosa et al. (2011)).

For each model that has parameters, they were determined experimentally by finding those that maximized the performance evaluation metrics described in Section 5.2.

For the second part we do not use time series data so we decided to keep the same models as the first step except the recurrent neural network which does not make sense in this context. To avoid presenting the use with too many parameters that they do not understand we use the default parameters set by sklearn for these models, for the neural network used with pressure curves we use one hidden layer with only one neuron and an L2 kernel regularizer of magnitude $5e^{-9}$ for the hidden layers and $2e^{-4}$ for the output layer. The network that works with the data other than pressure curves has 5 neurons in its hidden layer. For the third part we can not fine tune the parameters of the models beforehand and we have the additional constraint of needing to be fast enough to let the user try many parameter combinations to find the one that works best for the machine they are diagnosing. For this reason we decided to use a support vector machine and a decision tree as possible choices for the final classification model. For the SVM we let the user choose the strength of the regularization term, the number of features that are selected through the forward feature selection method, the kernel and its parameters, and for the decision tree we let them choose its maximum depth.

5.4 Results

5.4.1 Pressure outlier detection

Figure 5.8 shows the ROC curves of the models on the pressure outliers test dataset. We can see that the identity neural network performs better than the other models and is able to recover 32.2% of outliers with 1% of false positives.

From this experiment we determined the formula to set the default outlier threshold for the tool as $Q_3 + 1.7 * \text{IQR}$ as discussed in the previous section. We also decided to use the IDNN and OSVM models in the diagnosis tool. The IDNN was selected because it had the best performance and the OSVM was added as well because it runs faster and achieves comparable results.

5.4.2 Diagnosis experiment

The experiment was composed of four different phases depending on whether vacuum check data and reference data were available. We will discuss for each of these scenarios what kinds of problems are visible in the data, how the different parts of the tool help localize it, and how the experts proceeded. The results are summarized in Table 5.1.

We consider four sources of problems: mechanical, vision, calibration, and vacuum. Depending on how many of the robots' motions are affected by the anomaly it can either be visible in the outliers of the data or only when compared with a reference good behavior of the machine which the expert may know or not in which case comparing with old data saved when the machine was known to perform well is required.

	vac_block	vis_cache	calib	accel	meca_belt	vac_pressure	vac_leak	vis_light
Bad data only	0/3	0/3	1/3	0/3	0/3	0/3	2/3	1/3
With reference data	0/3	3/3	2/3	3/3	3/3	3/3	3/3	2/3

Table 5.1 – Number of experts that correctly identified the problems for each part of the experiment. Pressure data was not available for the first four cases.

In the first phase of the experiment the expert does not have pressure data or reference data, they can only rely on the outlier detector and what they believe to be normal values for a well-behaving machine. Since no pressure data is available it should not be possible to differentiate between vacuum system and mechanical problems but vision and calibration issues should be visible.

The experts quickly understood what they could see in the data that was available and found relationships between variables that allowed them to monitor specific parts of the machine, for instance the relationship between the perimeter and the area of the products detected by the vision system changes with the shapes of the objects that are seen by the camera, and looking at the position at which the object was detected then that at which it was picked up by the robot gives information about the matching of the vision coordinates with the robot's.

They had more trouble knowing what normal values looked like and they did not manage to isolate what was going wrong most of the time. They also had a tendency to rely too much on the outlier detector and forget to look for patterns themselves.

In the outlier detector itself we noticed that the IDNN model seemed to be able to model more complex relationships between the data as the OSVM one. In Figure 5.9 we can see that the IDNN model understands that the data points are supposed to be on a line and does not consider the points in the top right corner to be anomalous while the OSVM model simply considers the regions where the points are more sparse as anomalous.

When we added the reference data the experts were able to correctly identify the problems most of the time with only one of them making a mistake on the calib case as he thought the problem was vision related.

They seemed to understand the output of the classifier better when comparing the reference data with the current observations than when classifying the outliers among the current data. They were however disturbed by the fact that it is impossible to know for sure whether or not a motion was failed without using the vacuum check data as no matter how unusual a data point may be it does not guarantee that the robot did not manage to handle the product correctly which made them hesitant.

Once they were allowed to use pressure data the experts became more confident in their predictions and formed their opinions faster. They still had trouble identifying abnormal patterns without reference data, they were usually able to find some incoherence in the data but they did not commit to a diagnosis unless the bad motions were clearly outliers.

In the case where the products were moved after being detected to simulate a defective conveyor belt all of the experts were able to correctly identify that the collected data could not explain the problem but none of them committed to declaring that the problem came from a mechanical part of the machine.

With both pressure and reference data the experts made almost no mistakes and were able to quickly identify all of the problems they were presented with. The only mistake was made in the case of the lighting of the vision being partially dimmed, the expert saw the small effect on the average pressure curve that resulted from the products being picked up off center but missed the change in the behavior of the vision data and therefore thought of a vacuum system issue.

5.5 Conclusion

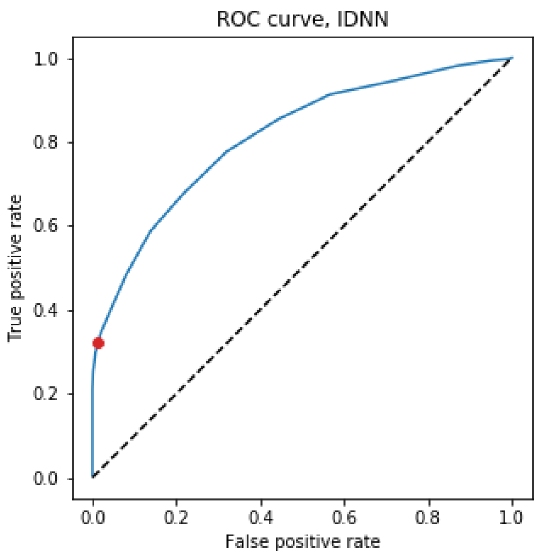
In this part we presented a machine diagnosis tool designed to help experts identify the cause of suboptimal performances on pick and place robotic lines. We presented and evaluated a way to find abnormal motions using pressure data collected in the machines' vacuum system during the motions of the robots as well as data generated by the software that controls the robots and the product detection system.

We had experts try to find defects on a machine with the help of the diagnosis tool and found that they performed very well when they had access to reference data taken when the machine was known to function well.

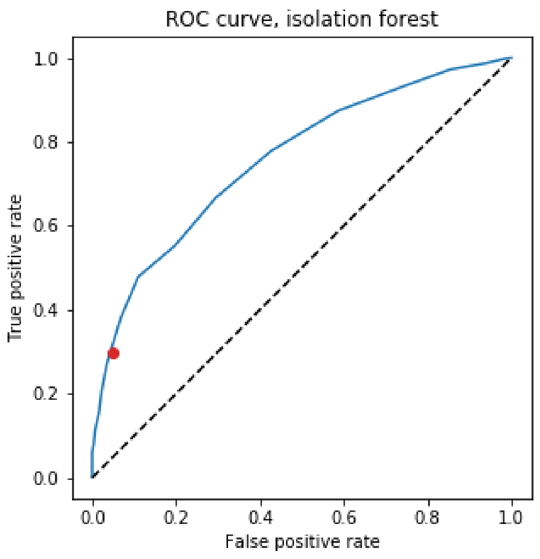
We believe that after using this tool for some time and getting more familiar with the kinds of pattern that can be expected to be seen in machines that function properly the experts could correctly diagnose machines faster than without using our tool and without needing physical access to the machine to investigate which would make customer service more efficient.

When performing experiments on machines that were available in the workshop we were able to find small problems that had not been seen by the field engineers that commissioned them, we therefore believe that our diagnosis tool could also help them in this task which would improve the efficiency of the machines that are produced by the company.

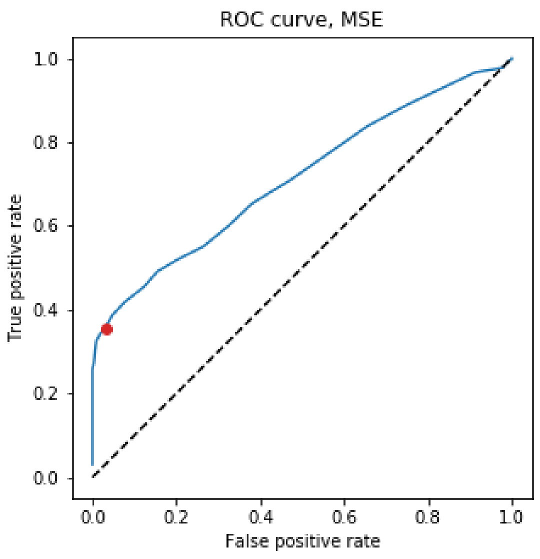
In the future the diagnosis tool should be tested on real cases and integrated to the customer service and machine commissioning workflows. An important part of this process will be determining how important it is to possess reference data when using the tool and how to obtain it if necessary. Determining if pressure data is necessary will also be important as most machines are not equipped with pressure sensors by default, obtaining data on these machines would then require sending sensors to the client or sending a technician to install them which might not be economically profitable.



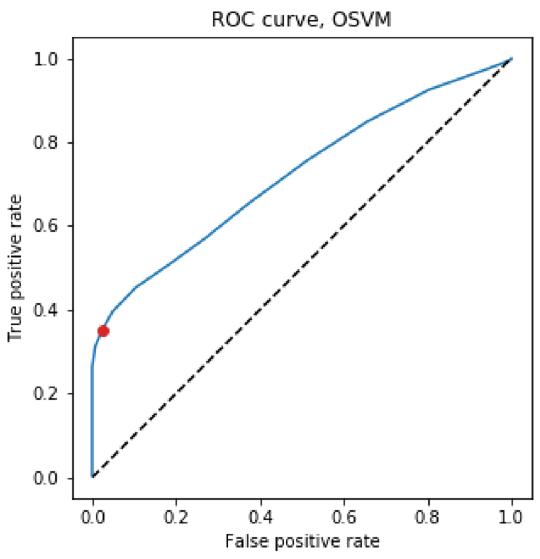
(a) IDNN



(b) Isolation forest

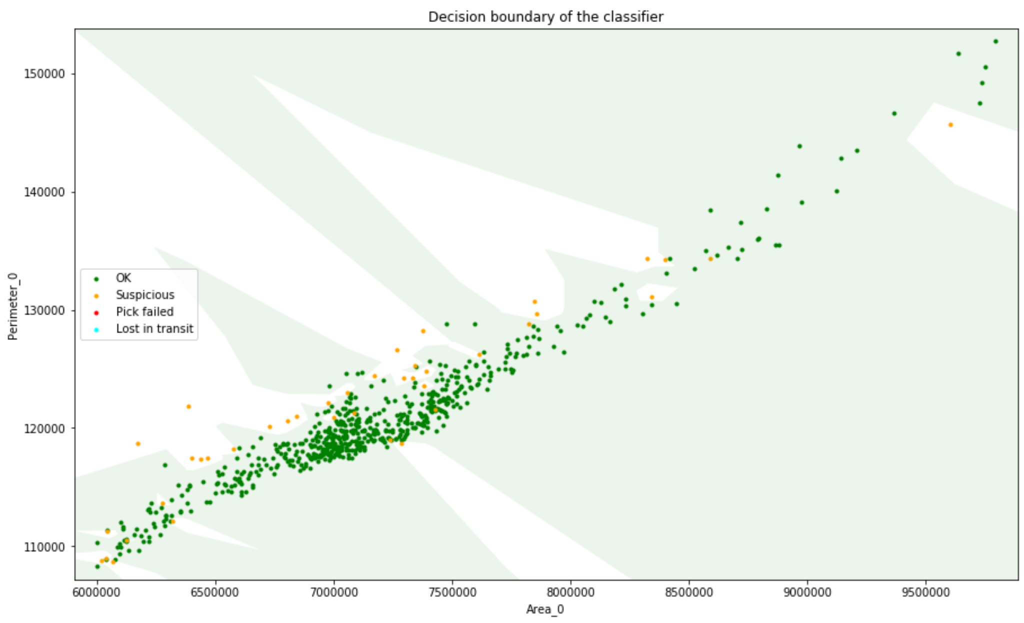


(c) MSE

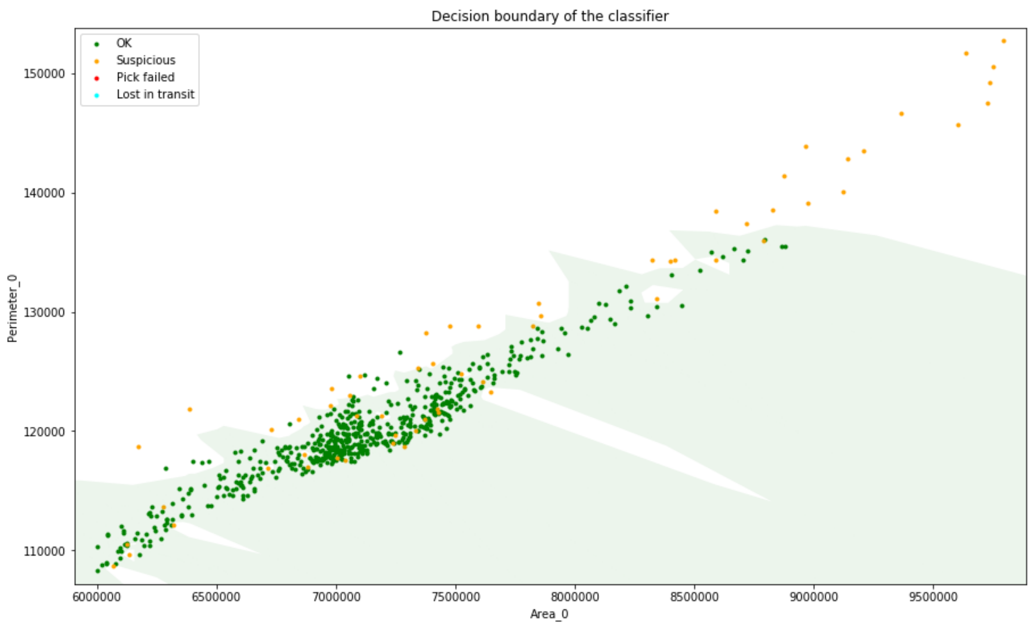


(d) OSVM

Figure 5.8 – The ROC curves of the models, the outliers were created by setting an offset of 1.5 cm on a working machine. The red dots represent the default value for the classification threshold.



(a) IDNN



(b) OSVM

Figure 5.9 – The outliers detected using the OSVM and IDNN models on non pressure data. The outliers are plotted in orange.

Product quality control Part III

When ordering a machine from Demaurex, some clients request that some quality control be performed on the products to make sure that no defective ones end up being sold to their clients. This is typically done with by integrating a quality check inside a vision based product detection system. This method is limited by time constraints that force the processing time of the check to be low and by the fact that the camera is fixed and therefore can not see the product in its entirety.

There also exists defects that are not detectable using a camera, for instance a biscuit might be broken in two with both parts still touching each other in such a way that it appears to be whole, when the robot picks it up it will only take one part and a broken product will end up in a box.

Some products also require to be lit from the back of the conveyor in order to be detected reliably which makes it impossible to obtain color images which may be required to detect if for instance the products may be missing a topping.

In these problematic situations it might be advantageous to use a vacuum check to perform the quality control when the product is picked up by the robot. The pressure in the vacuum system is sensitive to the shape, weight, and texture of the products, it could therefore be able to successfully detect the cases mentioned above. One potential problem is that the pressure is also sensitive to the state of the tubes and valves as well as the parameters of the machine.

This part explain how we designed an online classifier that is able to classify products as being correct or defective and to adapt itself to the changes of the state of the machine. Chapter 6 gives an overview of the technical background and previous work done on these subjects and Chapter 7 details the process of designing the classifier and assessing its performance.

6 Background

The problem we want to solve is complex and to our knowledge no research has been made on the same exact subject. The classifier needs to be able to take decisions in real time on a stream of incoming data, this aspect is covered in Section 6.1. It also needs to be able to adapt to changes in the meaning of the data that it receives, this is known as concept drift and it is detailed in Section 6.2. In most of the available literature, the algorithms are designed and evaluated using a test then train system where after making a prediction for a new value, the true label is revealed and the algorithm can be trained with this information. In our case no labels are available, techniques to deal with this are presented in Section 6.3. We also have to deal with a very imbalanced stream for which only about one percent of the products are defective. Finally, all of the literature we found either dealt with data streams of feature vectors or continuous time series but we receive a small time series for each sample which negatively affects many of the algorithms we found.

6.1 Stream classification

Stream classification also known as online classification is the problem of classifying data when it can not be stored entirely in memory and only some of it can be seen at once. Once data is discarded in this context it is lost forever and can not be recovered.

Oza and Russell (2001) have developed online versions of bagging and boosting and more recently Wang and Pineau (2016) adapted UnderOverBagging, SMOTEBagging, AdaC2, CSB2, RUSBoost, and SMOTEBoost to the online case.

Domingos and Hulten (2000) build an incremental decision tree algorithm called the Very Fast Decision Tree or Hoeffding tree. Later Manapragada et al. (2018) modified this solution to let it converge faster to the solution that would be obtained by building a tree on the full dataset.

In the literature the models are compared by the speed at which they reach their maximum performance i.e. the number of samples they need to see to converge and the maximum classification accuracy that they achieve on different datasets.

6.2 Concept drift

Gama et al. (2014) state that concept occurs in an online learning scenario when the relation between the input data and the target variable changes over time. They also define different type of drift: sudden drift is an instant change of the relationship between the input and the target, incremental drift occurs slowly over time, gradual drift occurs when the relationship changes from its initial state to a new one abruptly several times before settling on the new state, and recurring drift means that the link between the input and the output changes then reverts back to a previous state after some time.

Online classification algorithms naturally adapt to concept drift and handle gradual drifts very well but they struggle with more sudden drifts, to compensate for this, some models were modified to keep a window of samples in memory and make predictions based only on this window, forgetting and removing from the model all the previously seen samples. An example of this strategy is CVFDT (Hulten et al. (2001)). The first window methods used a fixed size window which forces the model to make decisions based on only the most recent data points, this creates a tradeoff between learning complex relationships using a lot of data and adapting to concept drift faster, to improve on this issue adaptive windows have been proposed by Bifet and Gavalda (2007), they detect concept drift by monitoring the performance of the classifier, shortening the window when the performance becomes worse and lengthening it when it increases.

Another technique is rebuilding the classifier when concept drift is detected. Rebuilding and windowing share the same problem, they forget old concepts which makes them struggle with gradual and recurring drifts. To account for these long term dependencies between concepts several models have to be kept in an ensemble. These ensemble methods differ from each other by the ways in which they make predictions and update the models, some give weights to the models based on their performance and perform weighted majority voting, while others keep old models and only use the one that currently performs the best. A good summary of ensemble methods can be found in Gama et al. (2014), Nguyen et al. (2015), and Brzezinski and Stefanowski (2014).

Some methods select which model to use to make predictions based on the current distribution of the input data, it is the case of the method proposed in Sethi et al. (2014) which creates clusters based on a discrete grid in the input space, and of the evolving granular neural networks of Leite et al. (2009).

Recently new methods using one model have been developed including kNN with a self adjusting memory based on the way human memory stores information (Losing et al. (2016)) and an adaptation of robust soft learning vector quantization (Heusinger et al. (2020)).

6.2.1 Models

We will now present in more details the models that we selected for our experiments, because of the specific challenges that we had to overcome we chose models that could easily be adapted to handle time series data and an imbalanced class distribution.

Online bagging
Originally proposed by Oza and Russell (2001) this online version of bagging relies on the observation that the distribution of the number of times each data sample appears in the training set of each ensemble member can be approximated by a Poisson(1) distribution when the number of training examples is large. Therefore given an ensemble of online classifiers the update rule for each new data sample is to incrementally train each classifier k times with the new sample where k is a random sample taken from a Poisson(1) distribution. Predictions of the ensemble are obtained by averaging the predictions of each of its members.

Accuracy weighted ensemble
The AWE algorithm of Wang et al. (2003) updates its base learners on chunks of data, it trains a new classifier using the newest data chunk then computes a weight for each model by measuring their performance on the same data. A maximum number of models is defined when creating the ensemble and the model with the smallest weight is discarder whenever too many models are created by the update procedure. The predictions are obtained by a weighted average of the ensemble member's predictions.

The weight update procedure relies on the mean squared error of each classifier C_i computed as follows:

$$\mathrm{MSE}_i = \frac{1}{|S_n|} \sum_{(x,c)\in S_n} (1 - f_c^i(x))^2$$

where S_n is a data chunk containing records of the form (x, c) where c is the true label of sample x, C_i's classification error is $(1 - f_c^i(x))$ where $f_c^i(x)$ is the probability given by C_i that x belongs to class c.

To serve as a reference for the performance of the classifiers, the MSE of a random classifier is also computed:

$$\mathrm{MSE}_r = \sum_c p(c)(1 - p(c))^2$$

where c is the set of all classes and $p(c)$ is the empirical probability of c in S_n.

The weight w_i of each classifier is then given by:

$$w_i = \mathrm{MSE}_r - \mathrm{MSE}_i$$

and classifiers whose weight is smaller than zero are discarded.

Since it is trained on the latest data chunk, the new classifier that is created for each new data chunk is evaluated by cross-validation on the chunk's data using the same procedure, this prevents evaluating the new model on its training set.

Dynamic weighted majority
Proposed by Kolter and Maloof (2007), DWM maintains a weighted ensemble of classifiers, it adds new classifiers to the ensemble when it performs poorly and reduces the weight of each individual classifier when they make mistakes. Classifiers that make too many mistakes and whose weight becomes too small are discarded from the ensemble.

The weight update procedure takes place once every p samples where p is a user defined parameter. For a weight update sample s the following steps happen: first every classifier that misclassifies s sees its weight reduced by a factor β defined by the user, then the weights of all the classifiers are normalized such that the maximum weight is one, if the ensemble misclassifies s a new classifier is created, trained on the last p samples and given a weight of one, finally any classifier whose weight is smaller than θ is removed from the ensemble.

Accuracy updated ensemble
Brzezinski and Stefanowski (2014) proposed the AUE2 algorithm as a way of dealing with multiple types of concept drift, it processes data in chunks and trains a new model for each chunk similar to AWE but the method in which weights are computed is different.

The MSE_r and MSE_i values are computed in the same way as AWE but the weight of the new classifier is given by:

$$w_{C^{'}} = \frac{1}{\text{MSE}_r + \epsilon}$$

with ϵ a small constant for numerical stability. This formula treats the new classifier $C^{'}$ as a perfect classifier because it was trained on the latest data which removes the costly cross validation step of AWE.

The weight of the existing classifiers are set according to:

$$w_i = \frac{1}{\text{MSE}_r + \text{MSE}_i + \epsilon}$$

the idea is to give weights that are further apart between good and bad classifiers compared to AWE.

6.3 Partially labeled streams

In the literature most papers are concerned with streams for which the truth can be either observed or confirmed by a human expert after a prediction has been made. This is not true for our use case because the goal is to replace human performed quality controls and observing the results with another automatic method is not easier or more reliable.

Some methods have been proposed to learn from both labeled and unlabeled data points in a stream: Read et al. (2015) propose using a deep belief network trained on all the samples seen by the model to extract features from the data and train classifiers using its output, and Bertini et al. (2012) use a method based on an incremental k-associated graph to propagate labels from labeled samples to unlabeled ones.

7 Experiments

In this chapter a modified version of the AUE2 algorithm is proposed to deal with the problem of automatic quality control. The proposed method is compared to state of the art stream classifiers.

In the following sections we describe the proposed method in detail, explain how we created the dataset to test it and how the experiments were set up, and analyze the results.

7.1 Proposed model

Our solution addresses three issues of the original AUE2 algorithm when it is applied to our problem: it uses Hoeffding trees as base learners which are not well suited to deal with time series data, it has no class weighting scheme which causes issues when it is used on class imbalanced data, and it was designed under the assumption that all labels for seen data are available for training.

To address the first issue we needed to find a base learner that could be trained incrementally and was well suited to handling time series data. We chose to use neural networks because they fit both criteria and are able to give predictions quickly. Because they are particularly well suited to time series data we chose to use convolutional networks.

To give more weight to a class we modify the way the MSE_i value is computed for each classifier by multiplying each part of the sum with a class weight:

$$\mathrm{MSE}_i = \frac{1}{|S_n|} \sum_{(x,c)\in S_n} w_c (1 - f_c^i(x))^2$$

where w_c is the weight assigned to class c.

Since we are using neural networks as base learners we also use a second class weight parameter that is passed on to the neural networks themselves when they are trained.

In order to obtain labels for some of the samples of the stream, we use an unsupervised outlier detection technique that is detailed in Section 7.3. This method can make mistakes and it is therefore important to use a weight update rule that is less abrupt than the one used by AUE2 to make sure that wrong labels will not deteriorate the performance of the classifier on the next chunk by too much. We chose to add and exponential decay on the weights of the classifier of the ensemble, the weight update step becomes:

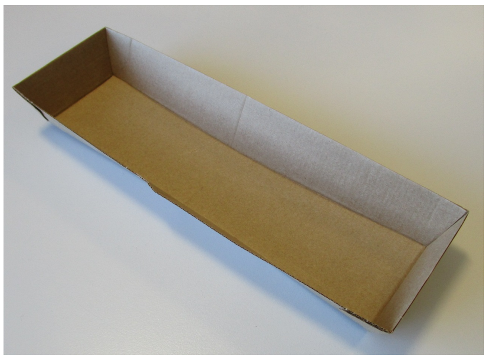
$$w_i^{(t+1)} = \frac{\beta}{\mathrm{MSE}_r + \mathrm{MSE}_i + \epsilon} + (1-\beta)\, w_i^{(t)}$$

where β is the decay parameter and t indicates the data chunk index.

7.2 Dataset

When a machine is running in a production environment we expect the pressure curves of the vacuum check to experience two types of drift: slow gradual drift that occurs as the machine ages and the vacuum tubes get dirty and sudden abrupt drift when the machine undergoes maintenance, its parameters are changed or some part of the vacuum system fails.

To simulate these situations we started by running a machine with good products and defective ones, the proportion of bad products is approximately 1.23% which is close to what was observed on the production line of a client that inspired us to investigate this problem. We used cardboard trays to simulate the products and we created three different defects: the first one is a weight added to one end of the tray to make it imbalanced, the second is another tray taped together with the first to get a heavier but still balanced product, and the third is a piece of bubble wrap sticked to the bottom of the tray to change the surface that is in contact with the suction cup. The first of the defective products is shown in Figure 7.1. Then we added a


(a) Good product

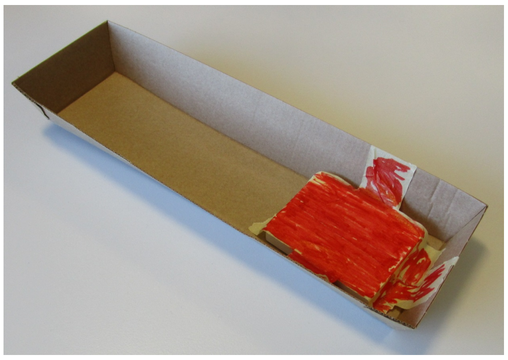

(b) Defective product

Figure 7.1 – One of the good products used in this part of the project and one of the defective ones.

dirty filter to the vacuum system to observe the effect that it has on the pressure curves. To obtain a dataset that simulates aging we then gradually interpolate the data we have collected from a normal state to one that resembles what we observed with the dirty filter, the effect of a dirty filter on the pressure curves can be seen in Figure 7.2.

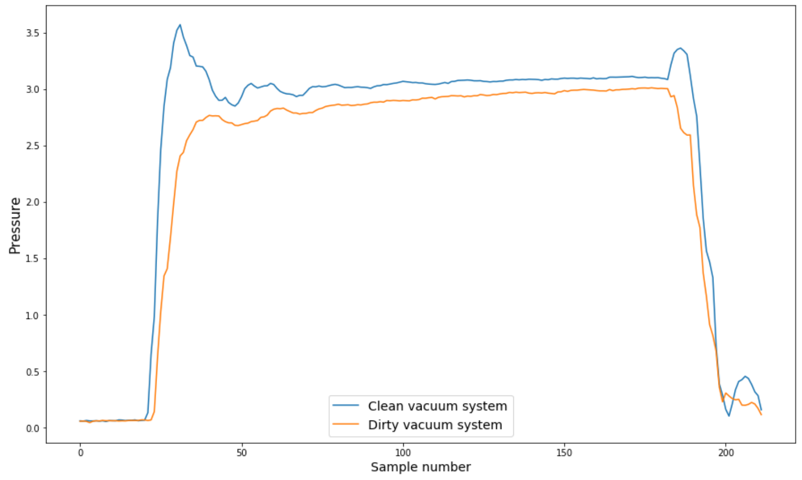


Figure 7.2 – Pressure curves obtained with a clean filter and a dirty one.

To simulate a maintenance cycle we suddenly reset the interpolation of the curves to go back to normal as if the machine was stopped to clean the filter.

Finally we collected more data after changing some of the machine's parameters. We also reset the aging interpolation when we change the parameters as if it happened during maintenance.

The resulting dataset contains around 9k samples, gradual drift, and two sudden drifts after around 3k samples each.

We tried training a neural network to classify the first 6k samples of the dataset before applying any drift and it was able to correctly classify all of the samples when using half of the data as a validation set. This proved that it was indeed possible to perform quality control in this way if concept drift was not an issue.

7.3 Setup

For the partial labeling of the data stream we compare the performance of the following models: Isolation Forest, One-class Support Vector Machine, MSE distance between each sample and their mean, and an autoencoder neural network (IDNN). For each method we test two cases: first using the complete data set to train the model, then using only a subset obtained by removing the outliers from the data.

Once we obtain the anomaly score for the samples we label half of them as follows: the n_b highest scoring samples are labeled as bad and the n_g lowest scoring samples are labeled as good where β is the rate of defective products:

$$n_b = \frac{\beta}{2}(\text{Chunk size})$$
$$n_g = \frac{1-\beta}{2}(\text{Chunk size})$$

This first experiment allows us to determine which model performs the best. We then compute partial labels for the whole data stream and use them to train the stream classifiers. We chose to use the same labels for all the classifiers to make the comparison as fair as possible even though better classifiers should theoretically obtain slightly better labels due to their ability to estimate the rate of defective products more accurately.

For the stream classifier we compare the performance of the following models on our dataset, Online Bagging, Dynamic Weighted Majority, Accuracy Weighted Ensemble, Accuracy Updated Ensemble (AUE2), and our modified version of AUE2 (ModAUE2). To establish a baseline of performance, a convolutional neural network is trained on a subset of data without any drift then tested on the full drifting dataset.

Similar to the setup of Brzezinski and Stefanowski (2014) we use ten base learners for every method, reduced to five for neural networks to keep processing time reasonably low. For methods that use Hoeffding trees we set the parameters to: grace period $n_{min} = 100$, split confidence $\delta = 0.01$, and tie threshold $\tau = 0.05$. We set the chunk size to 400. We run two experiments for each model, one with the base learner that it was developed for and a second one with a convolutional neural network.

The convolutional neural networks consist of two 1D convolutional layers with stride 2 and activation selu, the first layer has 8 filters and a kernel size of 13 while the second has 16 filters and a kernel size of 7. A dense layer with one output and a sigmoid activation function then generates the prediction.

In order to use the partial labeling procedure described above we need to estimate β, the rate of defective products in the chunk that the classifier has processed. This estimation is obtained using the predictions of the ensemble on the data chunk, we also assume that the true mean rate of bad products is known and limit the values to a 90% confidence interval for the mean of a binomial random variable of probability equal to the true defect rate and number of trials equal to the size of the chunk. This allows a good classifier to indicate when slightly more or less defective products are present in a chunk while protecting from extreme values that might result from a poorly performing classifier.

In order to measure how much the partial labeling procedure affects the performance of the classifiers we also ran an experiment where all the labels are available to the classifiers. Because the partial labeling procedure forces the classifiers to process the data in chunks, we decided to keep this constraint even when all labels are available, this only affects DWM and

OB which can update on a sample per sample basis.

Because the dataset is relatively small compared to others used in the literature we start by pre-training the classifiers on a small dataset that represents the initial state of the machine before it starts drifting. We use approximately 600 samples for this.

To build the models we use tensorflow (Abadi et al. (2015)) for the neural networks, scikit-multiflow (Montiel et al. (2018)) for the stream classification algorithms except AUE2 which we implemented ourselves based on the original implementation of Brzezinski and Stefanowski (2014), and scikit-learn (Pedregosa et al. (2011)) for the other models.

7.4 Results

7.4.1 Partial labeling

Training the models on the full dataset gives the results shown in Figure 7.3. These results show that all the models perform better when they are trained only on the good motions as long as the false positive rate is less than 1%. When the false positive rate grows larger, the MSE and OSVM model actually perform better when they are trained on the full dataset.

Using these results we decided to label half of the data stream since the outlier detectors are able to correctly classify half of the samples while maintaining a false positive rate of less than 1%. We also chose to purify the data chunks by removing the samples that correspond to defective products before training the outlier detection models as it improves their performance. We compared the performance of the MSE and OSVM models to perform this task since they obtained the best results when trained on the full dataset using between 1 and 4 iterations and keeping between 96 and 99% of the samples per iteration. The results of this experiment are shown in Figure 7.4.

To provide a better comparison of the tradeoff between the number of samples that remain in each block after the purification step and the purity of the block we also plotted the graphs as shown in Figure 7.5

Based on these results we chose to use the OSVM model with 4 iterations, keeping 98% of the samples at each iteration as we did not want to reject more than 10% of the samples in each block and these values provided the best results while satisfying this constraint.

Using these parameters we ran the partial labeling procedure ten times on the whole dataset and obtained on average 2.1 false positives, and 0.3 false negatives on the entire dataset which represents on average a false positive rate of 4.56% and a false negative rate smaller than 0.01%.

For the experiments of the next subsection we chose to use the partial labels produced by one of the partial labeling runs which produced three false positives as it is representative of the average performance of this procedure.

7.4.2 Stream classification

Figure 7.6 shows the accuracy over the whole dataset of the five models we consider using their default base estimators which are Hoeffding trees for AUE2, ModAUE2, and online bagging, and naive Bayes for DWM and AWE. Figure 7.7 presents the same metric but every method uses convolutional neural networks as base estimators.

Using convnets yields better results than Hoeffding trees and naive Bayes methods, this goes against the observations of Gama et al. (2014), we believe that this is due to the fact that we classify time series and not feature vectors as has been done in all the literature we could find on the topic.

On the CNN variants of the models we observe that online bagging and DWM handle the gradual and recurrent drift well but fail completely on the sudden drift to a new concept around sample 6000, AWE is barely than random chance on the gradual and recurrent drift and starts accepting all samples after the sudden drift of sample 6000, AUE2 and ModAUE2 both handle all the kinds of drift better than the other methods with the original AUE2 performing better on the end of the dataset and worse on the beginning compared to ModAUE2.

Figure 7.8 and Figure 7.9 present the same metrics but the models are trained using the partial labeling procedure described in the previous sections, the dotted vertical lines represent errors in the labeling.

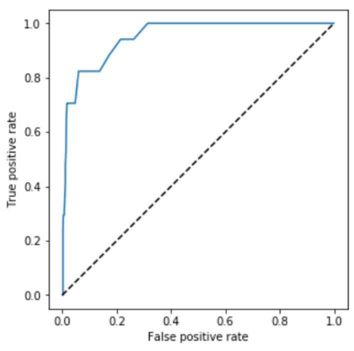
The non-CNN versions of the algorithms all fail to classify the stream, among the CNN versions AUE2 and ModAUE2 obtain the best performance with ModAUE2 performing slightly better than the original version on the gradual drifting part of the data and slightly worse when recovering from the sudden drift.

7.5 Conclusion

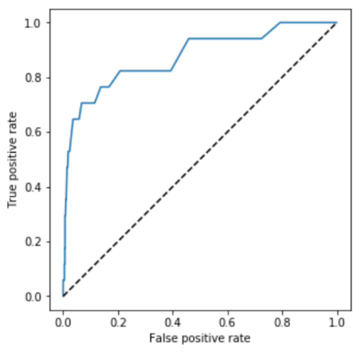
In this part we presented a way of performing quality checks on the products treated by a robotic pick and place line using pressure and air flow measures from the machine's vacuum system and showed that it could work using a partly artificial dataset for which concept drift was introduced artificially on top of data collected on actual robots. We introduced a modified version of the AUE2 ensemble stream classifier to classify the products as good or defective and showed that neural networks can outperform Hoeffding trees and Naive Bayes classifiers when each element of the stream is itself a time series.

We also used unsupervised outlier detection methods to provide labels to train the classifier as none are available in this situation and showed that the classifier could remain accurate when using only a fraction of the stream as training data.

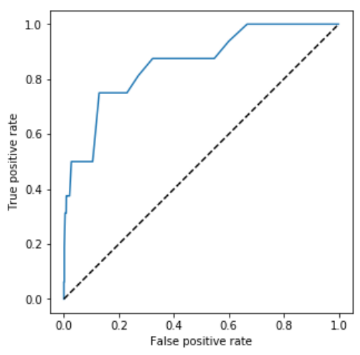
In the future we aim to test this quality control solution in a real production environment and try other approaches to classify the products using incremental unsupervised time series clustering. We also want to test the proposed solution's robustness to recurring drift more thoroughly and explore other ways of making the classifier remember all of the concepts it learns as it adapts to the stream.



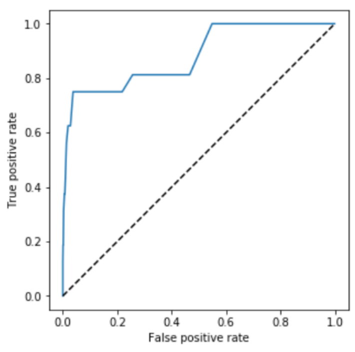
(a) MSE full data



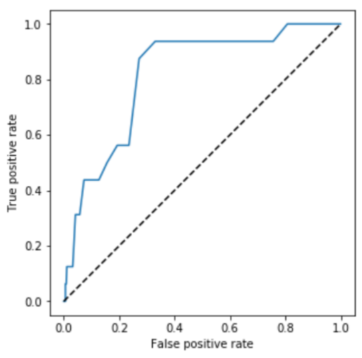
(b) MSE purified data



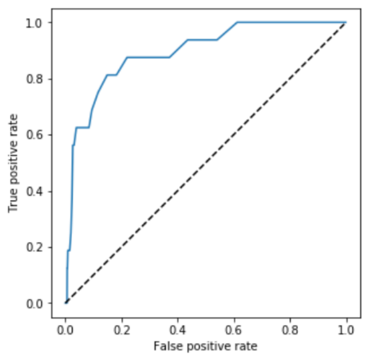
(c) IDNN full data



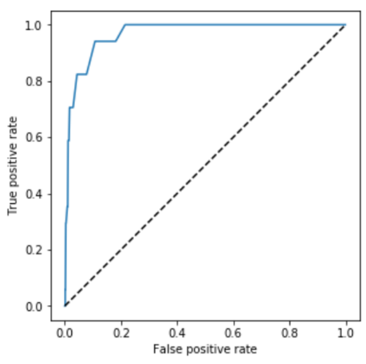
(d) IDNN purified data



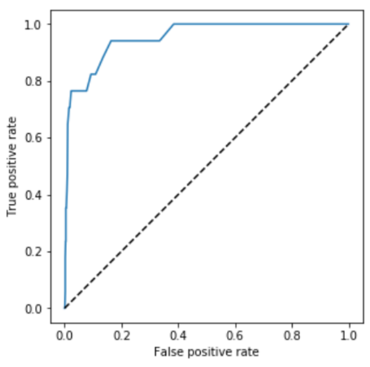
(e) Isolation forest full data



(f) Isolation forest purified data

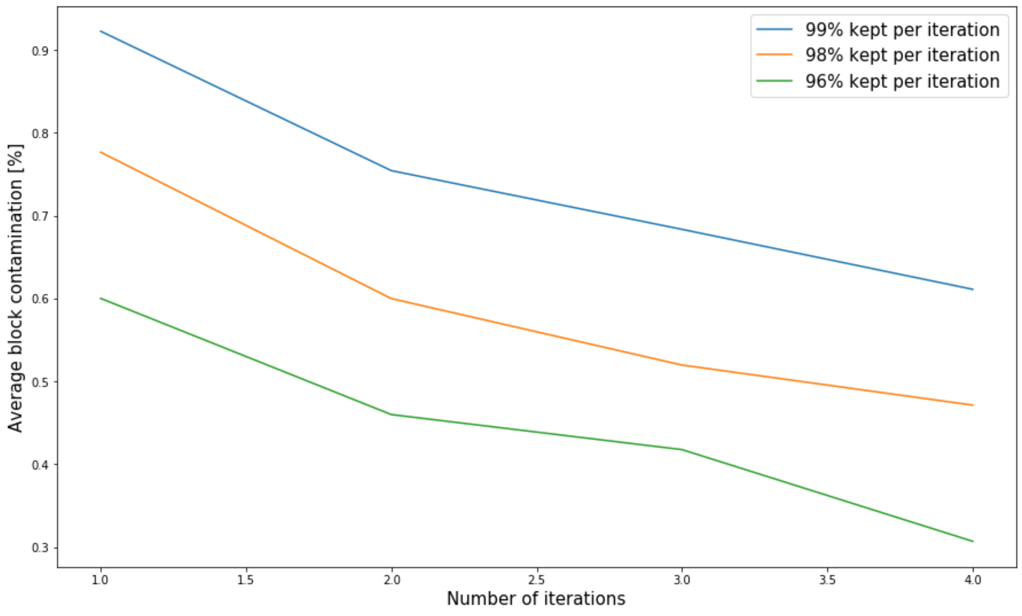


(g) OSVM full data

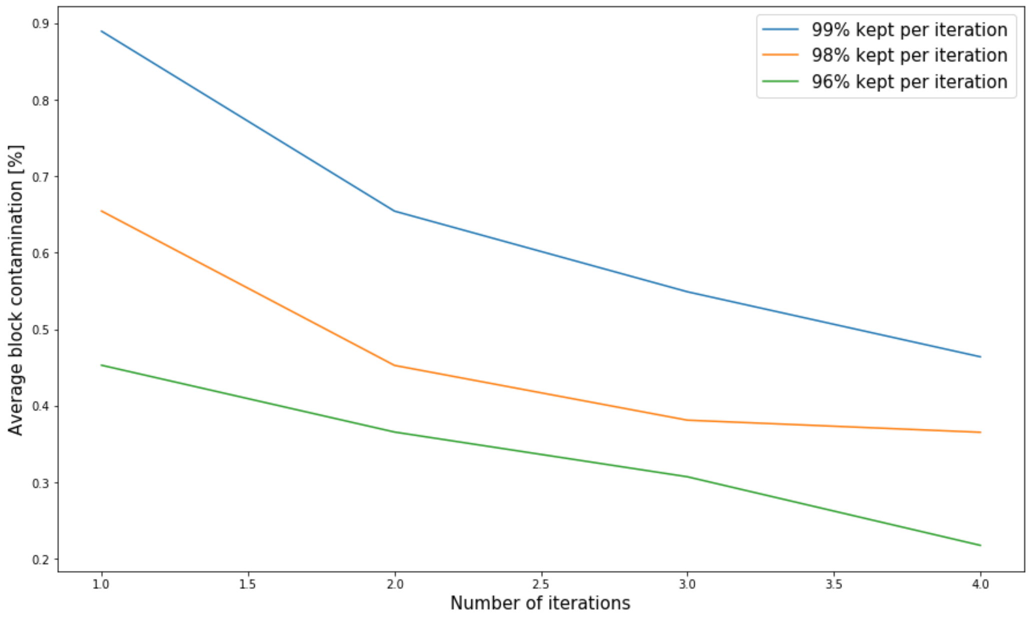


(h) OSVM purified data

Figure 7.3 – The ROC curves of the outlier detection models when trained on the full dataset (left) and only on good motions (right).

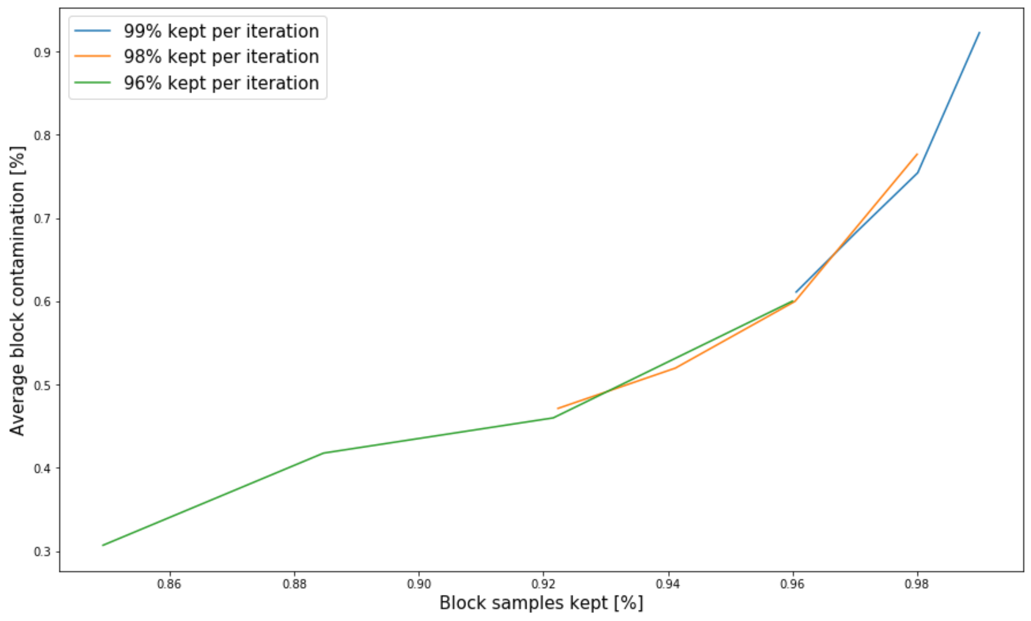


(a) MSE

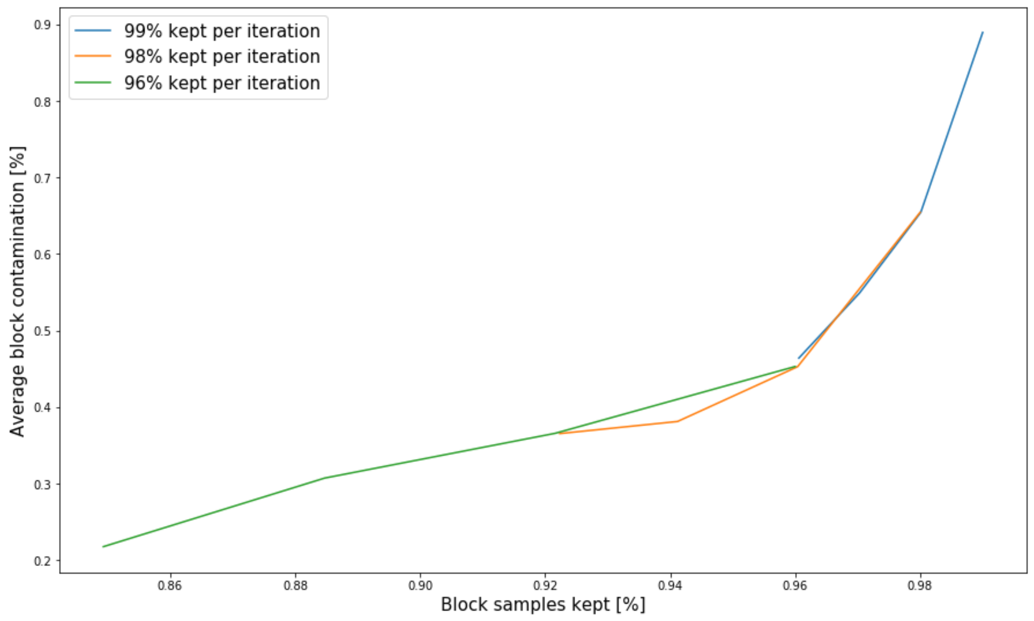


(b) OSVM

Figure 7.4 – The results of the data purification experiment, contamination is the percentage of bad motions in a data chunk after it has been purified.

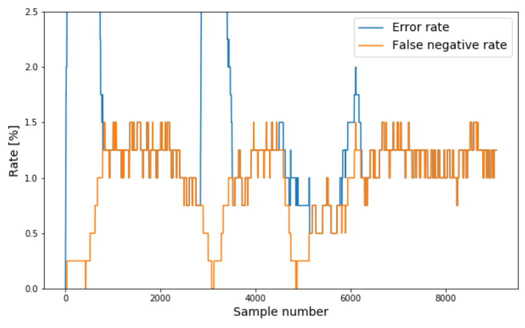


(a) MSE

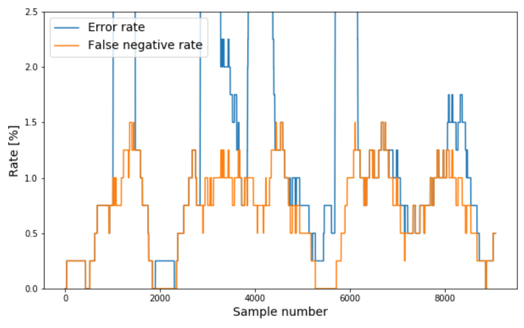


(b) OSVM

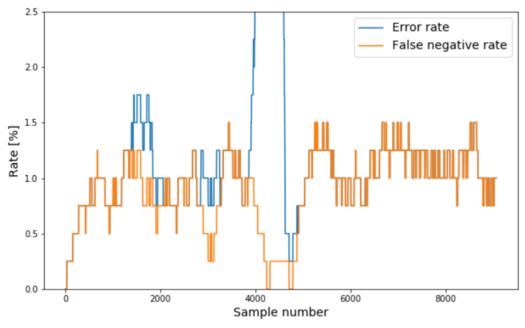
Figure 7.5 – The results of the data purification experiment, the x-axis now gives the percentage of samples kept per block for each purification model.



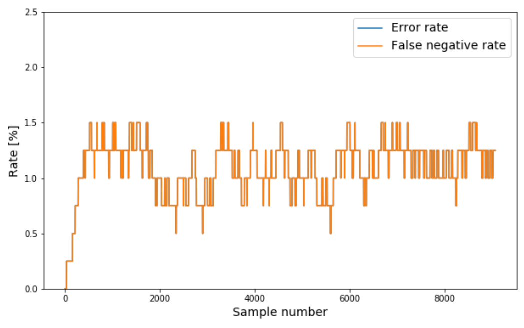
(a) Online bagging



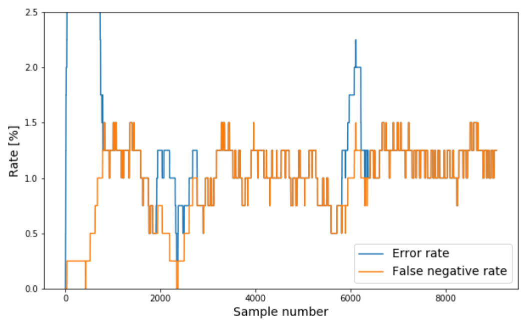
(b) DWM



(c) AWE

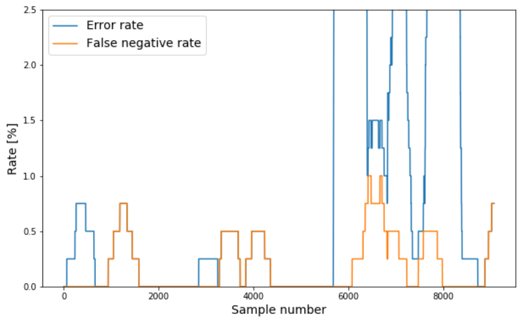


(d) AUE2

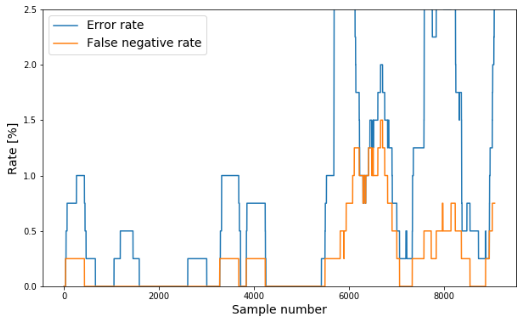


(e) ModAUE2

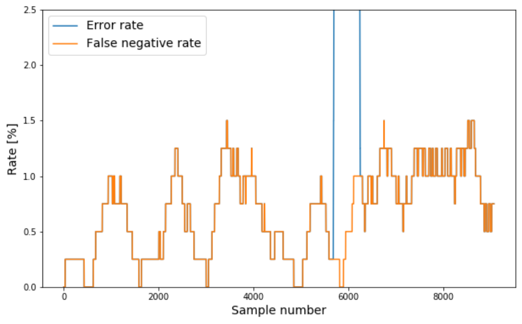
Figure 7.6 – The error rate of the different methods using their default base estimators when training with all labels available with a test-then-train scheme. The y-axis is limited to the range [0-2.5] because the stream contains 1.23% of defects and an error rate superior to that is worse than simply guessing that all the elements are correct.



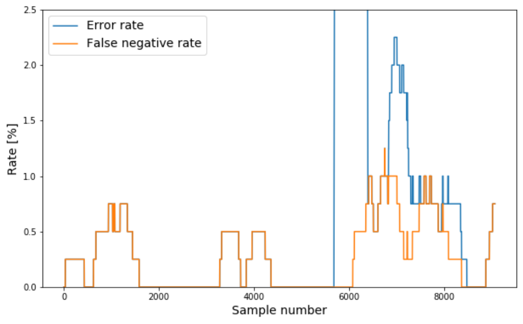
(a) Online bagging-CNN



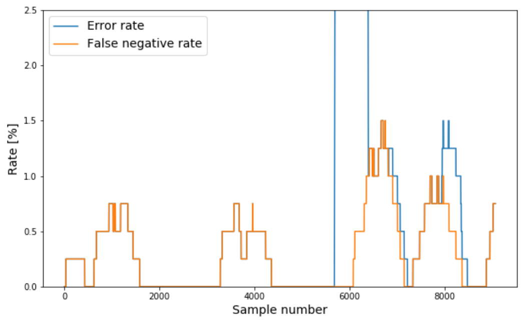
(b) DWM-CNN



(c) AWE-CNN

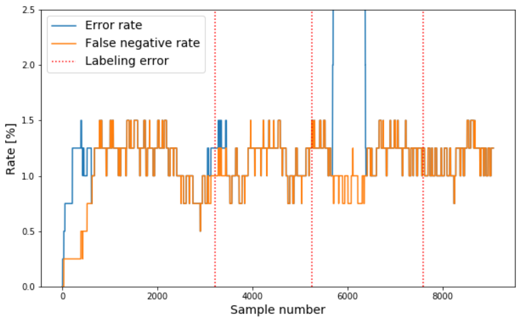


(d) AUE2-CNN

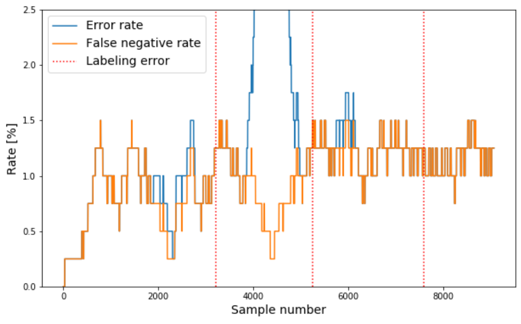


(e) ModAUE2-CNN

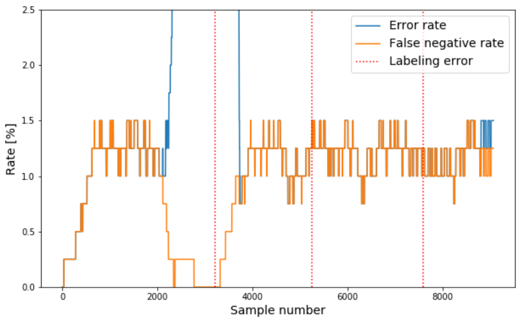
Figure 7.7 – The error rate of the different methods using convolutional neural networks as base estimators when training with all labels available with a test-then-train scheme. The y-axis is limited to the range [0-2.5] because the stream contains 1.23% of defects and an error rate superior to that is worse than simply guessing that all the elements are correct.



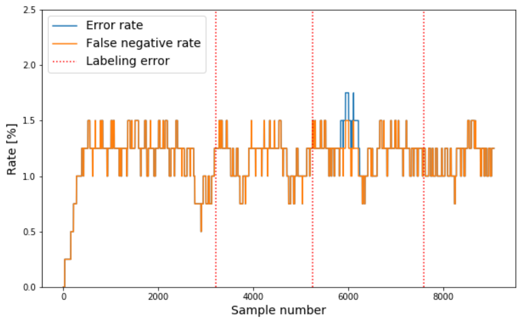
(a) Online bagging



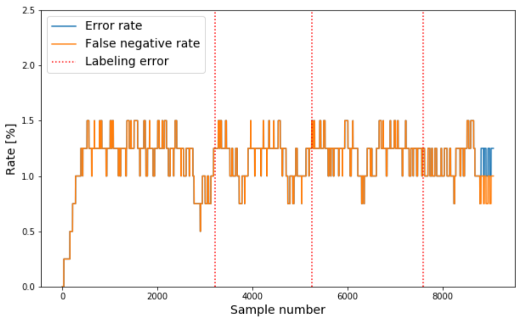
(b) DWM



(c) AWE

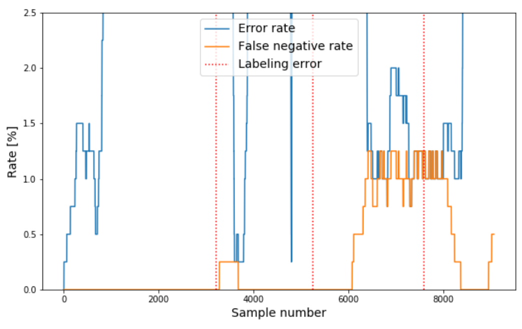


(d) AUE2

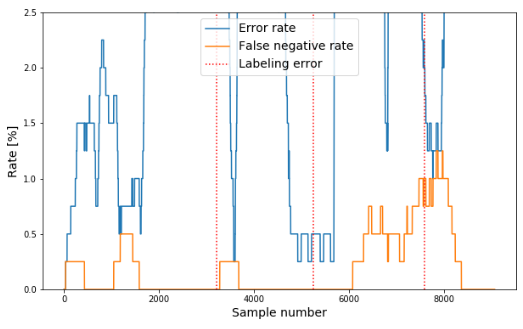


(e) ModAUE2

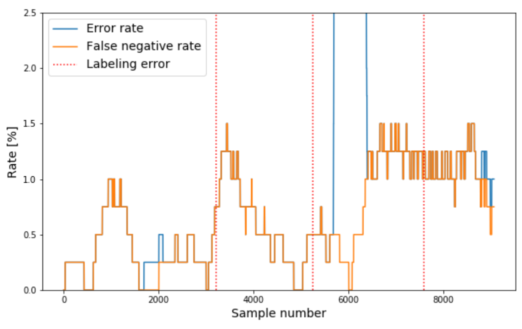
Figure 7.8 – The error rate of the different methods using their default base estimators when training with the partial labeling method. The y-axis is limited to the range [0-2.5] because the stream contains 1.23% of defects and an error rate superior to that is worse than simply guessing that all the elements are correct.



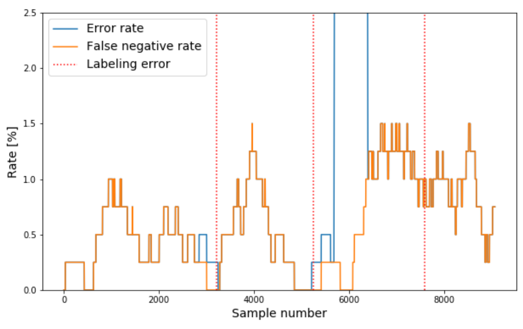
(a) Online bagging-CNN



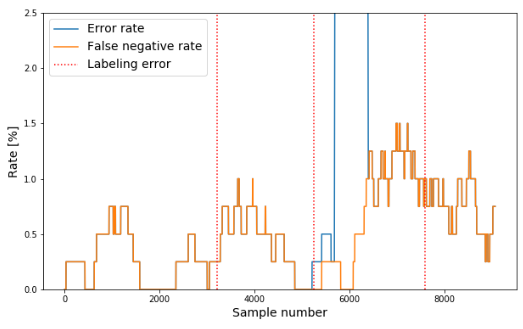
(b) DWM-CNN



(c) AWE-CNN



(d) AUE2-CNN



(e) ModAUE2-CNN

Figure 7.9 – The error rate of the different methods using convolutional neural networks as base estimators when training with the partial labeling method . The y-axis is limited to the range [0-2.5] because the stream contains 1.23% of defects and an error rate superior to that is worse than simply guessing that all the elements are correct.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Allan, J., Carbonell, J. G., Doddington, G., Yamron, J., and Yang, Y. (1998). Topic detection and tracking pilot study final report.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405.

Bertini, J. R., Lopes, A. d. A., and Zhao, L. (2012). Partially labeled data stream classification with the semi-supervised k-associated graph. *Journal of the Brazilian Computer Society*, 18(4):299–310.

Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM.

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Björck, Å. (1996). *Numerical methods for least squares problems*. SIAM.

Bonev, I. (2001). Delta parallel robot-the story of success. *Newsletter, available at http://www. parallelmic. org.*

Borchani, H., Varando, G., Bielza, C., and Larrañaga, P. (2015). A survey on multi-output regression. *WIREs Data Mining and Knowledge Discovery*, 5(5):216–233.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Brzezinski, D. and Stefanowski, J. (2014). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):81–94.

Caragea, D., Cook, D., and Honavar, V. G. (2001). Gaining insights into support vector machine pattern classifiers using projection-based tour methods. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’01, page 251–256, New York, NY, USA. Association for Computing Machinery.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

Cherkassky, V. and Yunqian Ma (2004). Comparison of loss functions for linear regression. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 1, pages 395–400.

Clavel, R. (1991). Conception d’un robot parallèle rapide à 4 degrés de liberté.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

Elmqvist, N., Dragicevic, P., and Fekete, J. (2008). Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1539–1148.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874. ROC Analysis in Pattern Recognition.

Gama, J. a., Žliobaitundefined, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4).

Gardner, M. and Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627 – 2636.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42.

Grubbs, F. E. (1969). Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21.

Hamel, L. (2006). Visualization of support vector machines with unsupervised learning. In *2006 IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology*, pages 1–8.

Heusinger, M., Raab, C., and Schleif, F.-M. (2020). Passive concept drift handling via momentum based robust soft learning vector quantization. In Vellido, A., Gibert, K., Angulo, C., and Martín Guerrero, J. D., editors, *Advances in Self-Organizing Maps, Learning Vector Quantization, Clustering and Data Visualization*, pages 200–209, Cham. Springer International Publishing.

Hodge, V. and Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126.

Hornik, K., Stinchcombe, M., White, H., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 97–106, New York, NY, USA. Association for Computing Machinery.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., and Söderström, T. (2018). Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. *CoRR*, abs/1802.04431.

Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., Mujica, F. A., Coates, A., and Ng, A. Y. (2015). An empirical evaluation of deep learning on highway driving. *CoRR*, abs/1504.01716.

Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *CoRR*, abs/1702.05659.

Kleinbaum, D. G., Dietz, K., Gail, M., Klein, M., and Klein, M. (2002). *Logistic regression*. Springer.

Kohonen, T. (1997). Exploration of very large databases by self-organizing maps. In *Proceedings of international conference on neural networks (icnn'97)*, volume 1, pages PL1–PL6. IEEE.

Kolter, J. Z. and Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8(Dec):2755–2790.

Lederer, A., Umlauft, J., and Hirche, S. (2019). Uniform error bounds for gaussian process regression with application to safe control. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 659–669. Curran Associates, Inc.

Leite, D. F., Costa, P., and Gomide, F. (2009). Evolving granular classification neural networks. In *2009 International Joint Conference on Neural Networks*, pages 1736–1743.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

Losing, V., Hammer, B., and Wersing, H. (2016). Knn classifier with self adjusting memory for heterogeneous concept drift.

Manapragada, C., Webb, G. I., and Salehi, M. (2018). Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953–1962.

Manevitz, L. M. and Yousef, M. (2000). Document classification on neural networks using only positive examples (poster session). In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 304–306, New York, NY, USA. Association for Computing Machinery.

Marcano-Cedeño, A., Quintanilla, J., Cortina-Januchs, G., and Andina, D. (2010). Feature selection using sequential forward selection and classification applying artificial metaplasticity neural network. pages 2845 – 2850.

Mendes-Moreira, J. a., Soares, C., Jorge, A. M., and Sousa, J. F. D. (2012). Ensemble approaches for regression: A survey. *ACM Comput. Surv.*, 45(1).

Migut, M. A., Worring, M., and Veenman, C. J. (2015). Visualizing multi-dimensional decision boundaries in 2d. *Data Mining and Knowledge Discovery*, 29(1):273–295.

Montiel, J., Read, J., Bifet, A., and Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5.

Nguyen, H.-L., Woon, Y.-K., and Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowledge and Information Systems*, 45(3):535–569.

Oza, N. C. and Russell, S. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, page 359–364, New York, NY, USA. Association for Computing Machinery.

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

Powers, D. (2008). Evaluation: From precision, recall and f-factor to roc, informedness, markedness and correlation. *Mach. Learn. Technol.*, 2.

Rauber, P. E., Fadel, S. G., Falcão, A. X., and Telea, A. C. (2017). Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110.

Read, J., Perez-Cruz, F., and Bifet, A. (2015). Deep learning in partially-labeled data streams. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, page 954–959, New York, NY, USA. Association for Computing Machinery.

Rifkin, R. M. and Lippert, R. A. (2007). Notes on regularized least squares.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., Williamson, R. C., et al. (1999). Estimating the support of a high-dimensional distribution. *Technical Report MSR-T R-99–87, Microsoft Research (MSR).*

Sethi, T. S., Kantardzic, M., Arabmakki, E., and Hu, H. (2014). An ensemble classification approach for handling spatio-temporal drifts in partially labeled data streams. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 725–732. IEEE.

Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression.

Vapnik, V. N. (1995). The nature of statistical learning theory.

Wang, B. and Pineau, J. (2016). Online bagging and boosting for imbalanced data streams. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3353–3366.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization.

Zhiyong Yan and Congfu Xu (2008). Using decision boundary to analyze classifiers. In *2008 3rd International Conference on Intelligent System and Knowledge Engineering*, volume 1, pages 302–307.