Tampere University

Jere Miettunen

IDENTIFYING SUITABLE PROFILE JOINTS IN CAD USING MACHINE LEARNING

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiners: Prof. Eric Coatanéa
April 2023

ABSTRACT

Jere Miettunen: Identifying suitable profile joints in CAD using machine learning
Master of Science Thesis
Tampere University
Master's Programme in Mechanical Engineering
April 2023

Profile joints serve a critical role in connecting structural profiles, such as beams and columns. The process of selecting appropriate profile joints can be both complex and time-consuming for structural engineers. This thesis explores how machine learning techniques can be used to aid in the selection of profile joints.

Six different machine learning classifiers were implemented and compared. To train the classifiers, an experimental dataset was extracted from 80 professionally designed building models, with 14 informative features identified through multiple feature evaluation methods. Data collection proved to be a significant challenge, highlighting the importance of good data collection practices. This dataset had enough samples for only 20 different profile joints. As a result, the classifiers support only this limited number of joints, while the complete solution is required to classify up to 100 joints. Several evaluation methods were used to compare the implemented classifiers. The best classification accuracy of 98.5% was achieved with XGBoost classifier. Keras Tuner was used to build a neural network classifier that achieved a classification accuracy of 97.6%. This neural network was used in a proof-of-concept tool to assist the user in the target software.

As the amount of data was limited, a large effort was made to examine and understand the data available. Dimensionality reduction methods such as uniform manifold approximation and projection were used to visualize the data. In addition, the interpretability of the model was enhanced with a method to analyze how each feature contributes to the individual predictions.

This research contributes to the growing field of artificial intelligence-assisted building design by providing a foundation for future work on AI-based profile joint selection and offering a detailed analysis of various machine learning models in this context. The most pressing future work involves collecting more data and repeating the experiments with a larger dataset, as well as exploring alternative algorithms and feature engineering techniques to improve model performance and generalizability.

Keywords: machine learning, neural network, profile joint, computer-aided design, building information modeling, structural engineering, deep learning, classification

TIIVISTELMÄ

Jere Miettunen: Koneoppimisen soveltaminen CAD-ohjelmistossa sopivien profiililiitosten tunnistamiseen
Diplomityö
Tampereen yliopisto
Konetekniikan DI-ohjelma
Huhtikuu 2023

Profiililiitokset ovat tärkeä osa rakennesuunnittelua. Ne yhdistävät profiileja, kuten palkkeja sekä pilareita. Sopivien profiililiitosten valitseminen voi olla monimutkaista ja aikaa vievää. Tässä diplomityössä tutkitaan erilaisten koneoppimistekniikoiden mahdollisuuksia helpottaa suunnittelijan työtä profiililiitosten valinnan osalta.

Työssä toteutettiin kuusi erilaista koneoppimiseen perustuvaa luokittelijaa. Niiden kouluttamista varten kerättiin 80 ammattimaisesti suunnitellusta rakennusmallista opetusdataa, josta tunnistettiin 14 informatiivista piirrettä useiden piirrevalintamenetelmien avulla. Datan kerääminen osoittautui merkittäväksi haasteeksi, mikä korostaa hyvien tiedonkeruukäytäntöjen tärkeyttä. Kerätyssä data-aineistossa oli riittävästi näytteitä 20 eri profiililiitoksen osalta. Tämän vuoksi toteutetut luokittelijat tukevat vain tätä rajallista määrää liitoksia, kun taas täyttä tukea varten luokittelijan tulisi pystyä tunnistamaan jopa 100 liitosta. Luokittelijoita vertailtiin käyttäen useita arviointimenetelmiä. Parhaan luokittelutarkkuuden 98,5 % saavutti XGBoost -luokittelija. Keras Tuner -ohjelmaa käytettiin rakentamaan neuroverkkoluokittelija, jolla saavutettiin 97,6 %:n luokittelutarkkuus. Tätä neuroverkkoa käytettiin kohdeohjelmistossa osana alustavaa työkalua, joka avustaa käyttäjää profiililiitosten valinnassa.

Käytettävissä olevan datan tutkimiseen ja ymmärtämiseen panostettiin erityisen paljon, koska data-aineiston määrä oli rajallinen. Datan visualisointiin käytettiin dimensionaalisuuden vähentämiskeinoja, kuten yhtenäisen moniston approksimointia ja projektointia. Lisäksi mallin tulkittavuutta parannettiin menetelmällä, jolla analysoidaan piirteiden arvojen vaikutusta yksittäisiin ennusteisiin.

Tämä työ edesauttaa kasvavaa tekoälyavusteista rakennussuunnittelua luomalla perustan tekoälypohjaisen profiililiitosten valinnan jatkokehitykselle ja tarjoamalla yksityiskohtaisen analyysin erilaisten koneoppimismallien hyödyntämisestä tässä asiayhteydessä. Tärkein jatkokehityksen kohde on kokeiden toistaminen suuremmalla data-aineistolla. Lisäksi tulisi tutkia vaihtoehtoisia algoritmeja sekä piirteiden käsittelytekniikoita mallin suorituskyvyn ja yleistettävyyden parantamiseksi.

Avainsanat: koneoppiminen, neuroverkko, profiililiitos, tietokoneavusteinen suunnittelu, rakennuksen tietomallintaminen, rakennesuunnittelu, syväoppiminen, luokittelu

PREFACE

Undertaking this thesis has been a significant learning experience, and I would like to acknowledge the individuals who have contributed to this accomplishment.

I want to thank my thesis supervisor, Prof. Eric Coatanéa, for his efficient guidance and profound expertise, which were instrumental in shaping my research. I also appreciate my colleagues, Joonas Hyvärinen and Marko Mätäsniemi, whose insights, support, and discussions have helped me to refine my work.

My gratitude extends also to Inka, my friends, and my family, who all provided continuous support and advice along the way.

Finally, I'd like to thank Timo Tulisalmi from Vertex Systems for providing me with the opportunity to pursue this interesting topic and encouraging me to dive deep into it.

Tampere, 28th April 2023

Jere Miettunen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
AUC-PR	Area under the Precision-Recall Curve
AUC-ROC	Area under the ROC Curve
BIM	Building Information Modeling
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
CSV	Comma-Separated Value
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DL	Deep Learning
DNN	Deep Neural Network
FEA	Finite Element Analysis
FEM	Finite Element Method
GAN	Generative Adversarial Network
KDE	Kernel Density Estimate
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
NAS	Neural Architecture Search
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SHAP	SHapley Additive exPlanations
SMOTE	Synthetic Minority Oversampling Technique
SVC	Support Vector Classifier

SVD	Singular Value Decomposition
SVM	Support Vector Machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection

1. INTRODUCTION

In the past few decades, building design has transitioned from hand-drawn plans and physical models to the digital world. The use of Computer-Aided Design (CAD) software has revolutionized the way architects and engineers approach building design. With CAD software, they can create detailed 3D models of buildings, test different design ideas, and simulate how a building will look and function before it's even built. This has not only made the design process faster and more efficient, but it has also allowed for greater accuracy and precision in the design. To increase the efficiency of the design process further, routine design tasks are automated. Automation in computer software can be achieved in various ways and the use of Artificial Intelligence (AI)-based methods is gaining traction in many fields, including Building Information Modeling (BIM) [1].

This thesis focuses on the implementation and use of AI-based methods to simplify profile joint selection in the BIM software Vertex BD. Profile joints are used to connect multiple structural profiles such as beams and columns. These connections then suppress some or all degrees of freedom between the profiles, transferring forces between them. Figure 1.1 presents an example of a profile joint that connects a steel beam and a steel column and displays the role of the proposed profile joint classifier. While many of the general profile joints are already generated automatically in Vertex BD, this automation is based on explicitly programming them to follow certain rules. This method has difficulties when there are multiple seemingly equally valid choices. Even if it were possible to program perfect rules for joints, it would likely require a lot of work and result in a significant amount of code which needs maintenance. This is where AI-based methods have potential to be useful. We can use existing data related to profile joints to teach a Machine Learning (ML) model to figure out the decision-making factors for selecting the correct profile joint. In this thesis, six different ML approaches are experimented with, and the results are compared.

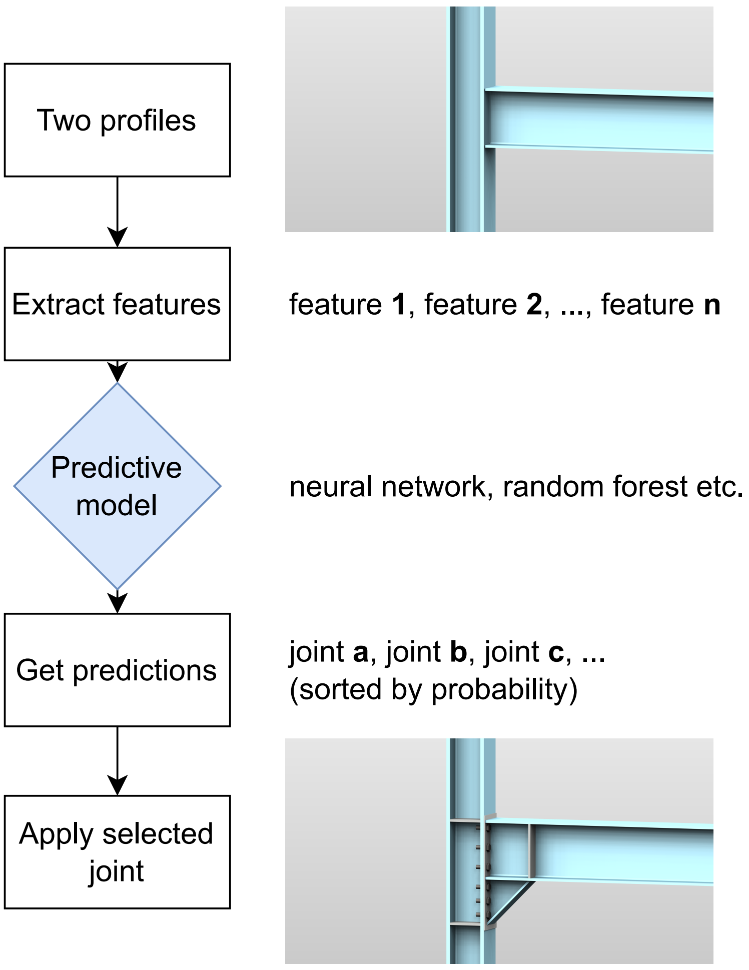

Two profiles
Extract features
feature **1**, feature **2**, ..., feature **n**
Predictive model
neural network, random forest etc.
Get predictions
joint **a**, joint **b**, joint **c**, ...
(sorted by probability)
Apply selected joint

Figure 1.1. *Profile joint classifier*

The research questions that guide this work are:

1. What are the key factors to consider when selecting a profile joint for connecting structural profiles such as beams and columns in building design?
2. How can machine learning techniques be leveraged to simplify the selection of profile joints, and what are the benefits of doing so?
3. Which machine learning models are best suited for profile joint selection in building design, and how do they compare in terms of accuracy?

The goal of this work is to provide an AI-based solution to profile joint selection, which can reduce the manual effort required, and increase the accuracy and precision of the design process. Moreover, this work has the potential to facilitate the automation of other design tasks, leading to more efficient and reliable building designs.

Chapter 2 of this thesis presents the relevant theory related to building design and ML, outlining the fundamental principles of structural engineering related to profile joints. It also provides an overview of the different ML techniques and algorithms that are experimented with in the implementation. Chapter 3 describes how the training data was collected, and how the features were selected and processed. It also visualizes the data by utilizing state-of-the-art dimensionality reduction techniques. Chapter 4 describes the implementations of the different classifiers and provides information on how they were trained. Furthermore, it discusses the software integration and design choices made during the implementation phase. Chapter 5 presents the evaluation metrics and experimental results for the different ML models. It includes a method to assist in the interpretability of the ML models. Finally, Chapter 6 summarizes the main findings of the thesis and provides recommendations for future research and development in this area.

As part of the writing process for this thesis, the language model ChatGPT, provided by OpenAI, was utilized to enhance the clarity, coherence, and overall quality of the text. ChatGPT is a state-of-the-art language model that uses deep learning algorithms to generate human-like responses to text-based prompts. In addition, ChatGPT was also used to assist with some of the programming tasks associated with this research. The software's ability to generate code snippets and provide context-specific suggestions helped to streamline the development process and ensure that the resulting code was accurate and efficient. [2]

2. BACKGROUND

2.1 Profile joints in building design

The design of a structural system can generally be divided into the design of structural members and the design of joints. The structural members are physically distinguishable parts of a structure, such as beams and columns. These parts are intended to carry the loads of the structure. The role of joints is to connect two or more structural members and transfer loads between them. They can be designed to be more flexible or rigid depending on the structural requirements and loads applied. Flexible joints, such as those utilizing bearings or expansion joints, are used to absorb movements that are caused by, for example, thermal expansion and vibrations. Rigid joints, such as those that utilize welding, are typically used to transfer large loads with minimal movement between the connected parts. [3, 4]

When selecting joints, several factors should be taken into account. The joint needs to be of appropriate type, having the correct degrees of freedom, to be able to transfer the loads effectively. Most importantly the joints should be strong and durable enough. They need to be able to withstand the forces they will be subjected to over time. They should also be cost-effective and easy to construct. Finally, the joint needs to comply with local building codes [4, 5]. Evaluating and optimizing the mechanical properties of joints can be difficult [6]. Finite Element Analysis (FEA) software can be used to approximate these properties. It is based on Finite Element Method (FEM) which is used to numerically solve differential equations. [7]

Vertex BD is a Building Information Modeling (BIM) software designed for architects, engineers and construction professionals. It has 3D Computer-Aided Design (CAD) capabilities, and it allows users to create and manage building projects from conceptual design to construction documentation. In this thesis, the structural design and more precisely, the profile joint selection is the main focus related to the software. Figure 2.1 presents an example of selecting a profile joint through the Connection Details browser in Vertex BD. [8, 9]

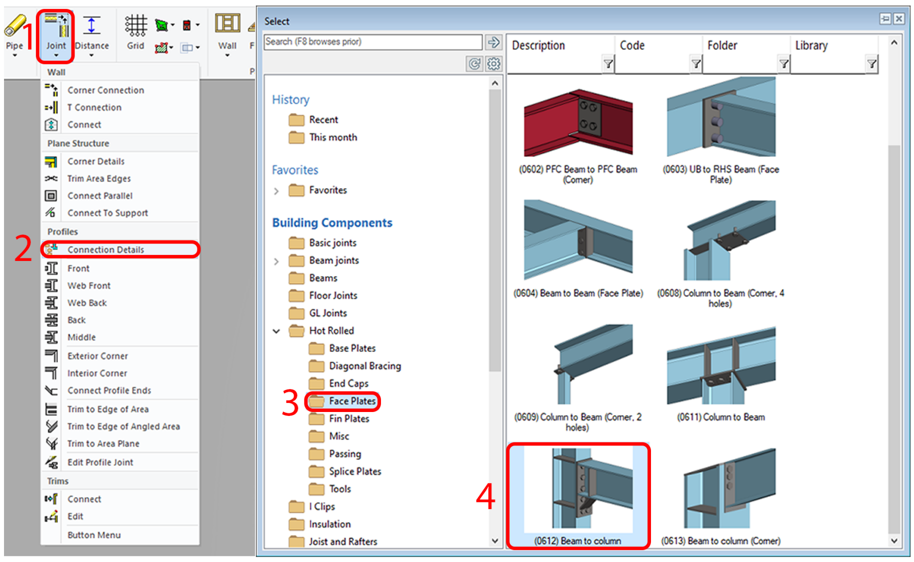


Figure 2.1. *Profile joint selection in Vertex BD*

The software aims to automate as many repetitive tasks as possible including the selection of profile joints, which can be generated automatically along with the structural elements. The building structures can often be very complex, which makes explicitly programmed automatic selection difficult or sometimes even impossible. Also, it's not unusual that the automatically generated structural elements need further modifications and tinkering. Since the users are specialists, they are able to select appropriate profile joints from the software's joint library when it's necessary. There are two main issues when profile joints are chosen manually. First, there is a risk of accidentally selecting the incorrect joint. Although the software has measures to detect such errors, it is preferable to avoid them altogether. Second, choosing profile joints manually can be time-consuming, particularly when selecting from a large library. Currently, users can assign frequently used joints as favorites, but it can still be challenging to find the optimal joint for a specific use case. [9]

To simplify the profile joint selection, AI-based solutions are experimented with. The next subchapters explain the theory behind ML and the different approaches more thoroughly and Chapter 4 describes the specific implementation and the steps required to form the final system.

2.2 Machine learning

Machine Learning (ML) is a type of Artificial Intelligence (AI) that allows systems to automatically learn and improve from experience without being explicitly programmed [10]. ML algorithms learn patterns from data, which can be either labeled or unlabeled, and then use those patterns to make predictions or take actions on new data. There are different types of ML algorithms, such as supervised, unsupervised, and semi-supervised learning.

Supervised learning is a type of ML where a model is trained on labeled data. During training, the model learns to connect a certain type of input to a corresponding output. The goal for the model is to make accurate predictions on new, but similar, unseen data. [11, 12]

In unsupervised learning the training data is not labeled beforehand. Instead, the algorithm's goal is to find patterns or structure in the data. This is useful when the objective is to explore and understand the underlying structure of the data, rather than making predictions or taking actions based on it. Another common use for unsupervised learning is generative modeling, where the model attempts to create new data based on the unlabeled training data. [12].

Semi-supervised learning is a combination of supervised and unsupervised learning. This means that only some portion of the training data is labeled, while the rest is unlabeled. The model then uses both labeled and unlabeled data to learn patterns and make predictions. Semi-supervised learning is useful when obtaining labeled data is expensive or time-consuming [13, p. 31]

Deep Learning (DL) is a subfield of ML that is based on using neural networks with many layers to perform tasks that can be difficult or even impossible for traditional ML algorithms to handle. DL has been successfully applied in various fields such as autonomous driving, medical imaging, natural language processing, and pattern recognition. In the construction industry, DL is used in, for example, crack detection, equipment tracking, construction work management, server assessment, and 3d point cloud enhancement [14]. The popularity of DL is due to the increased amount of computer-accessible data, availability of benchmark datasets, and the increase in computing power. Advancements in algorithms have enabled the training of deep neural networks with almost arbitrary depth. With the potential to automate pattern recognition and feature detection, DL is poised to make even greater strides in the future. [12, 13]

2.2.1 Traditional machine learning classifiers

In the broader field of ML, traditional classifiers have proven to be simple, interpretable, and effective in handling high-dimensional data too [15]. This subchapter will review some common traditional ML methods for classification, such as random forest classifiers, Support Vector Machines (SVMs), and Naive Bayes classifiers.

Ensemble learning is a ML technique where multiple models are trained and combined to make a final prediction. This group of multiple individual models is called an ensemble. While DL methods, such as Deep Neural Networks (DNNs), have been revolutionary in their ability to learn from complex data, they may not always be the best option for every problem. Ensemble methods, on the other hand, can perform well on cases that are not as suited for DL. One reason for this is that ensemble methods can be more effective with smaller datasets. Ensemble learning is a broad topic that can be utilized in simple traditional ML methods like decision forests, but they can similarly be utilized to combine more complex models, such as DNNs too. The idea is that by combining the predictions of multiple models, the overall performance of the ensemble can be better than any single model alone. This can be done in several ways, such as by averaging the predictions of the models, or by training a separate model to make the final prediction based on the outputs of the other models. Ensemble methods are often used to improve the performance of a model, particularly in cases where the individual models have high bias or high variance. Ensembling may lead to increased complexity when comparing to the individual models. This can cause, for instance, longer training times and decreased interpretability. [16–18]

An example of ensemble learning is Random Forest. It is a supervised ensemble method for decision trees. A decision tree is a model that makes predictions based on a series of decisions made by the model based on the input data. A decision tree consists of condition nodes and leaf nodes, which represent predictions. The input is processed through the condition nodes until a leaf node is reached. In Figure 2.2 the rectangles represent the condition nodes and the ellipses the leaf nodes.

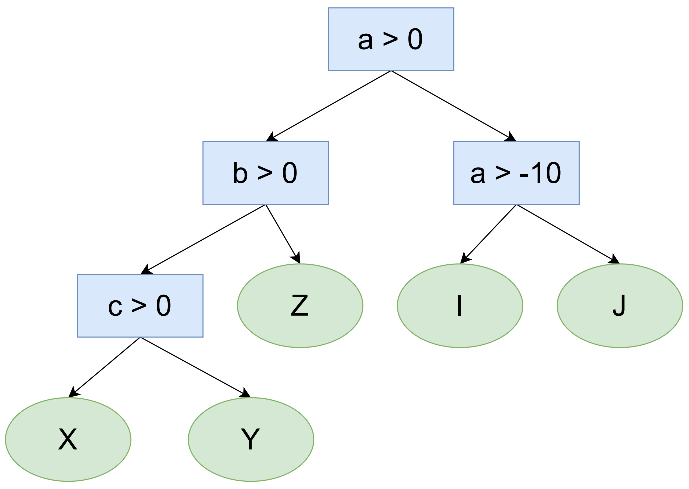


Figure 2.2. *Decision tree*

Combining multiple decision trees results in a decision forest, and an example of one is presented in Figure 2.3. Random forest is a popular type of decision forest. In a random forest, multiple decision trees are trained on different subsets of the data and with different subsets of the features. The final prediction is then based on the combined predictions of the individual decision trees. The key idea behind random forests is that by training multiple decision trees on different subsets of the data, the model will be less prone to overfitting, and the final predictions will be more robust. Additionally, since each tree in the forest is trained on a different subset of the data, the trees will have different levels of bias and variance, which helps to balance out the overall performance of the ensemble. Overall, Random Forest is a very powerful algorithm that can be used for both classification and regression tasks. [19–21]

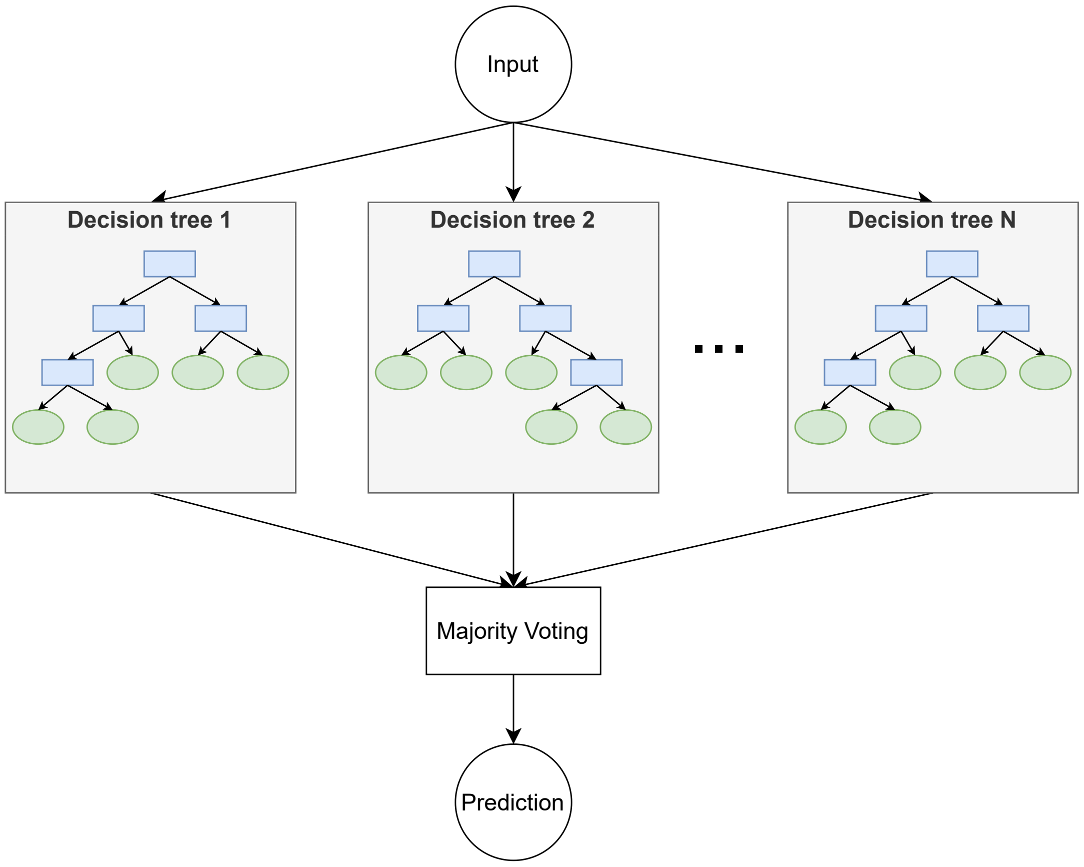


Figure 2.3. *Decision forest*

In the decision forest presented as an example, predictions are made by combining the results from multiple decision trees. One common method for combining the predictions is the majority voting approach, where the prediction that occurs most frequently across the trees is selected as the final prediction. However, there are other methods available for aggregating predictions, such as the weighted average and median. These methods may be preferred in certain scenarios, such as when certain trees or predictions are deemed more reliable or informative than others. In regression tasks, the approach to aggregating predictions may differ. Rather than selecting the most frequent prediction or using a median, a common method is to take the mean of the predicted values. [13]

Support Vector Machines (SVMs) are a type of supervised ML models that can be used for both classification and regression tasks. Those SVMs that are used for classification are often called Support Vector Classifiers (SVCs). The basic idea behind SVCs is to find a hyperplane between the different classes in the data. If each entry in each class can be separated with these hyperplanes, the data is linearly separable. This also means that there are an infinite number of possible hyperplanes that can be fitted to the training set. In linear SVCs, the training is done by finding the hyperplane that is the furthest distance away from the closest training instances. This can be also thought of as creating the widest possible decision boundary between the classes, where the hyperplane is extended with two support vectors. If all the data is to be kept outside the decision boundary, it is called hard margin classification. Hard margin classification does not work if the data is not linearly separable, e.g. there is even a single entry that is inside the other classes cluster, which could easily happen from a labeling or measuring error, for instance. Even if the data would be linearly separable, hard margin classification is sensitive to outliers reducing the final classification accuracy. [17, p. 155]

Compared to the hard margin classification, a more flexible approach would be to use soft margin classification. It differs from hard margin classification by allowing some margin violations, where some points might end up inside or even on the wrong side of the decision boundary. The result depends on the slack variables, which control the allowed violations when calculating the solution. This can be tuned by using the C parameter, where a low value of C leads to a wider decision boundary but more misclassified points. A high value of C results in a smaller decision boundary but fewer misclassified points. With its flexibility, soft margin classification also might need more tuning than hard margin classification. [11, 17]

Figure 2.4 presents the decision boundaries formed by a SVC when applied to a synthetic dataset with three classes. The SVC is trained using a linear kernel and a regularization parameter $C = 1$, attempting to balance margin maximization and classification accuracy. While the linear SVC may not always be the best choice for complex, non-linearly separable data, it can provide a reasonable solution in situations with well-separated classes in a low-dimensional feature space.

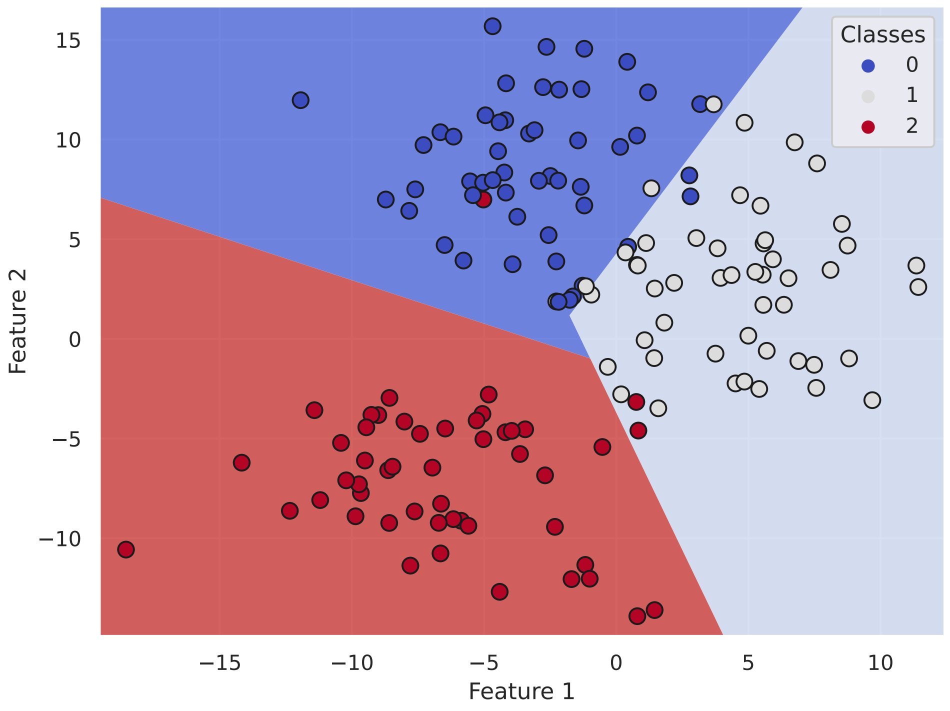


Figure 2.4. *Linear support vector classifier*

When working with a dataset that is far from linearly separable, linear SVCs likely won't provide good results even when using soft margin classification. If adding new input features is possible, it might sometimes result in a linearly separable dataset. The other approach would be to use non-linear SVCs for these datasets. This means that instead of a straight line or a hyperplane, a non-linear decision boundary is used instead. This can be achieved by first transforming the data into a higher-dimensional space, called the feature space, and then applying the linear boundary. Non-linear SVCs are more flexible and capable than linear SVCs, but at the same time more complex. [22, p. 14] [17, p. 159]

In Figure 2.5, the decision boundaries of a linear SVC and a non-linear SVC using the Radial Basis Function (RBF) kernel are compared on a synthetic dataset with non-linear class boundaries [23]. The linear SVC struggles to separate the classes effectively due to the inherent non-linearity of the dataset, resulting in an unsatisfactory classification performance.

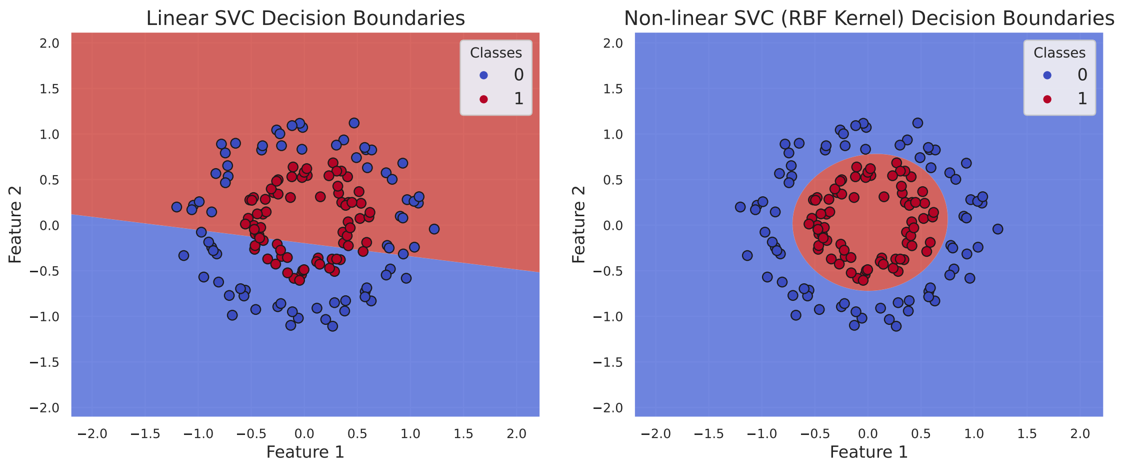


Figure 2.5. *Linear SVC compared to a non-linear SVC*

On the other hand, the non-linear SVC with the RBF kernel is capable of handling the non-linear nature of the dataset and successfully separates the classes, as shown in the second plot of Figure 2.5. The RBF kernel maps the input data into a higher-dimensional feature space, enabling the SVC to find a non-linear decision boundary that accurately separates the classes. This example highlights the importance of choosing an appropriate kernel function for the specific characteristics of the dataset when working with SVMs, as non-linear kernels can significantly improve classification performance on non-linearly separable data.

The Naive Bayesian classifier is a popular supervised ML algorithm used to predict the class of a new observation based on the Bayes theorem of conditional probability. The algorithm assumes that the predictor variables are independent and follow a normal distribution. It works efficiently if its assumption is observed and is more useful in determining the most likely class rather than the actual probabilities for various classes. The classifier requires little explicit training and can perform well with high-dimensional or large datasets. Before training, the conditional probability distribution should be determined, and the individual probability distributions of features should be estimated from the training dataset. There are different types of naive bayes classifiers, including Multinomial Naive Bayesian, Bernoulli Naive Bayesian, and Gaussian Naive Bayesian. The algorithm has been successfully applied in spam classification, among other applications. [13, p. 63]

In conclusion, this subchapter has provided an overview of traditional machine learning methods for classification, including random forest classifiers, SVCs, and naive bayes classifiers. While deep learning has revolutionized the field of machine learning, traditional classifiers continue to hold value due to their simplicity, interpretability, and effectiveness in handling high-dimensional data. Ensemble methods, such as Random Forest, are particularly useful for smaller datasets and can be more effective in some cases than deep learning methods. The choice of classifier depends on the specific problem at hand and the characteristics of the dataset. As demonstrated in the provided examples, selecting the appropriate classifier, kernel function, or ensemble method can significantly impact classification performance. Therefore, understanding the strengths and limitations of each traditional classifier is essential for tackling various classification problems in machine learning.

2.2.2 Neural network classifiers

Deep Learning (DL) methods, such as neural networks, should be considered when simpler models don't achieve adequate results. Some tasks where Deep Neural Networks (DNNs) usually excel are pattern matching in images, natural language processing and audio related tasks. These tasks are usually high dimensional which means that the dataset has numerous features. [16, p. 307]

Artificial Neural Network (ANN) is a type of machine learning model that is originally inspired by the structure and function of the human brain. Some researchers suggest dropping this biological analogy to emphasize that the ANNs aren't limited to biologically plausible systems [17, p. 277]. ANN consists of a number of connected nodes or artificial neurons, each of which performs a simple mathematical operation on its inputs.

A supervised ANN is trained by introducing example data associated with the correct outputs. During the training phase called forward pass, a batch of examples is processed through the network to yield a predicted output. The difference between the predicted output and the true output is called the loss value, which is high for an untrained network. During the iterative process called backpropagation, the gradient of the weights and biases is calculated with respect to the loss value. These gradients are then used to update the weights and biases using an optimization algorithm such as Stochastic Gradient Descent (SGD), Adam, or other variants of the gradient descent [24]. Modern ML software libraries, such as the popular TensorFlow created by Google, have the backpropagation algorithm built-in and the user can choose a suitable optimizer and possible hyperparameters. One of these hyperparameters is the learning rate which tells the gradient descent algorithm how strongly should the weights and biases be adjusted on each iteration. A low learning rate will make the training process take longer, but a too high learning rate might have trouble reaching convergence. [13]

There are plenty of different architectures for ANNs, including fully connected ANN, which is the most basic and commonly used type. Fully connected ANNs are called Multilayer Perceptrons (MLPs). An example of a simple MLP is shown in Figure 2.6. In addition to the MLPs, other types of neural network architectures include Convolutional Neural Networks (CNNs) for image recognition, Recurrent Neural Networks (RNNs) for sequential data, Long Short-Term Memory (LSTM) networks for addressing vanishing gradients in

RNNs, autoencoders for unsupervised learning and data compression, and Generative Adversarial Networks (GANs) for generative modeling. These architectures are designed to address specific types of problems and have been developed for various applications. [13, p. 123-159]

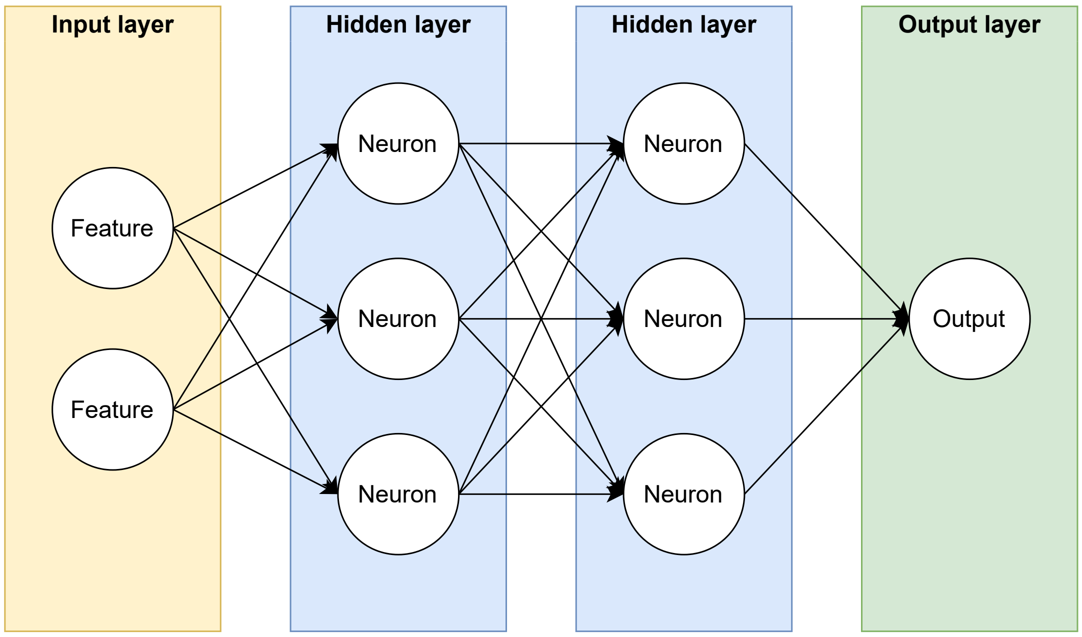


Figure 2.6. *Multilayer perceptron*

The MLP consists of dense layers, which means that each neuron on that layer is connected to every neuron on the next layer. Each arrow represents the connection between the neurons, where the output of a neuron on an earlier layer is a direct input to the neurons in the next layer. The inputs are weighted, and the outputs from each neuron are combined to make a final prediction. Also, for each neuron, a bias term is added. The weights and the bias terms are the parameters that are modified during the training. The more hidden layers there are, the higher the network's capacity will be as there will be more trainable parameters. With a larger and more complex network, the computational cost will increase. Overfitting might also occur more easily with a larger network, but this can be mitigated with proper regularization techniques. [13, p. 66-73]

In each artificial neuron, the output is calculated by some non-linear function of the sum of its inputs. These non-linear functions are usually called activation functions. Without the non-linearity introduced by the activation functions, the model couldn't accurately estimate the non-linear nature of many problems. Also, a network without non-linear activation functions could be reduced and represented as a single neuron. Some commonly used activation functions are Rectified Linear Unit (ReLU), sigmoid, tanh and softmax. The equation of ReLU activation function is defined as:

$$f(z) = \max(0, z), \tag{2.1}$$

where the z is the input value. Figure 2.7 presents the plot of the ReLU activation function.

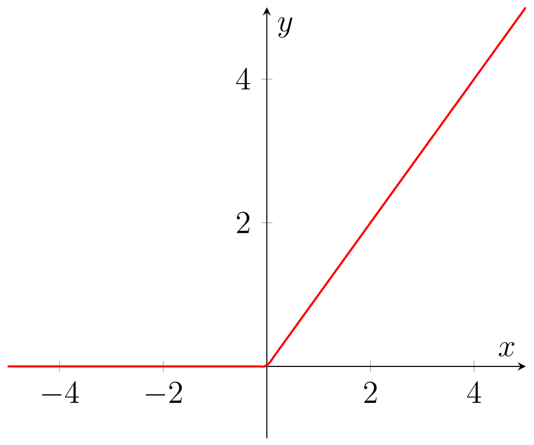


Figure 2.7. *ReLU activation function*

ReLU activation function is very simple as it only compares if the input value is greater than zero and if it is, the output is equal to the input and otherwise the output is zero. The main advantage of ReLU activation function is its speed. With a high learning rate ReLU might cause the weights to update in a way that the neuron will never update during the training again. This means that the ReLU neuron will die and therefore stop learning. This issue can be averted with a suitable learning rate or by using a leaky ReLU activation function. The equation for leaky ReLU is defined as:

$$f(z) = \max(\alpha z, z), \tag{2.2}$$

where z is the input value and α is a small constant. An example of leaky ReLU is visualized in Figure 2.8.

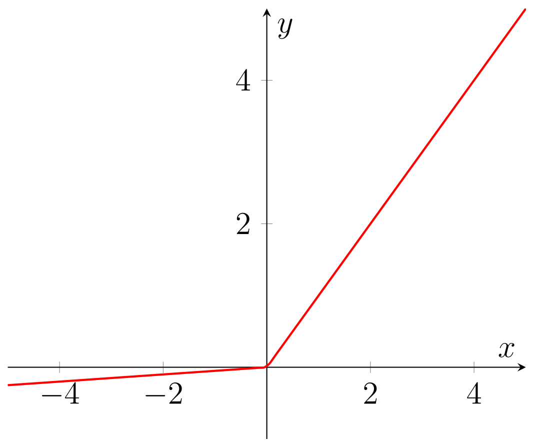


Figure 2.8. *Leaky ReLU activation function where* $\alpha = 0.05$

This means that when the input is negative, the activation is a small positive slope instead of zero. [13, p. 74-75]

The equation of the sigmoid activation function, denoted by $\sigma(z)$, is defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{2.3}$$

where the z is the input value and e is the base of the natural logarithm. The sigmoid function is plotted in Figure 2.9.

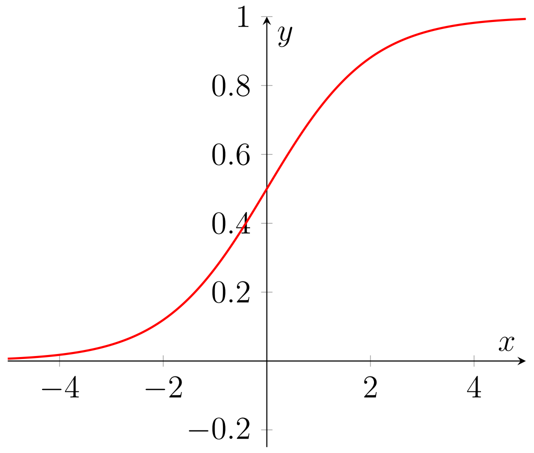


Figure 2.9. *Sigmoid activation function*

The sigmoid activation is a mathematical function that takes any input value z and squashes it to a value between 0 and 1. Large negative input values approach 0 and large positive values approach 1. The sigmoid has some drawbacks, such as the vanishing gradient problem, where the gradient becomes very small as the input to the sigmoid becomes either very large or very small. In the backpropagation algorithm, a near zero gradient will be multiplied with the latter gradients, and therefore they will vanish too. Because of this, when the gradient becomes very small, the training of the neural network can slow down. This problem is comparable to the dying ReLU neurons. The sigmoid used to be a popular choice, but due to its drawbacks it is less common nowadays. It is still commonly used for binary classification problems in the output layer. [13, p. 73-74]

The hyperbolic tangent, or tanh for short, is another commonly used activation function in neural networks. The equation of the tanh function is defined as:

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \tag{2.4}$$

where z is the input value and e is the base of the natural logarithm. The tanh function is similar to the sigmoid function, but it outputs values between -1 and 1, with a range that is twice as wide as the sigmoid. The plot of the tanh function is shown in Figure 2.10.

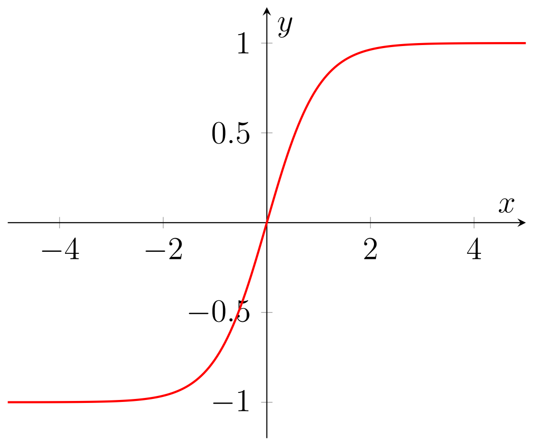


Figure 2.10. *Tanh activation function*

Compared to the sigmoid function, the tanh function has the advantage of being zero-centered, which can make the optimization of the weights in a neural network easier. However, the vanishing gradient problem still exists with the tanh function, as the gradient becomes very small as the input values get very large or very small. [13, p. 74]

For multi-class classification output layer, softmax activation function is commonly used. The softmax function takes a vector of inputs and outputs a probability distribution over multiple classes, ensuring that the sum of the probabilities equals 1. The equation of the softmax function is as follows:

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.5}$$

where z_i represents the input values, e is the base of the natural logarithm, and K is the number of classes. [25, p. 15] Figure 2.11 presents an example of how softmax output looks like.

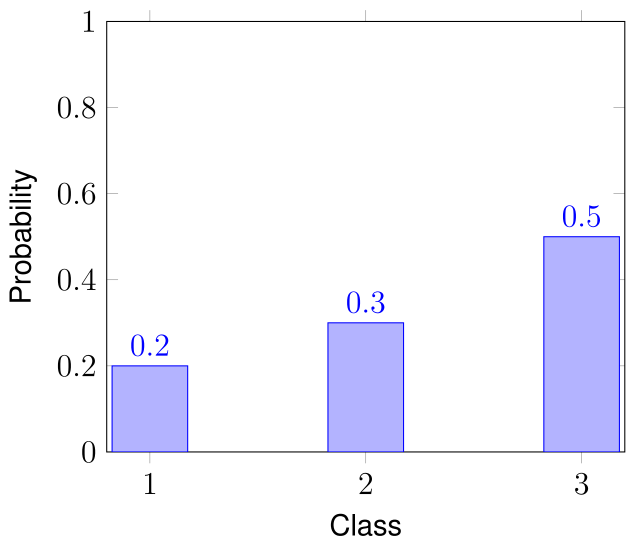


Figure 2.11. *Softmax activation function*

The softmax activation function assumes a mutually exclusive relationship between classes, forcing the probabilities to sum to 1. This behaviour is not usually useful outside the output layer. [13, p. 119]

The activation functions introduced here are some of the most common ones, but other activation functions that perform well exist too. For example, the mish activation function proposed by Misra has been shown to outperform ReLU in some experiments [26]. However, due to the simplicity and general performance that ReLU achieves, the scope on activation functions has been limited to the ones presented here.

Regularization is a critical method in machine learning that can help prevent overfitting and improve model generalization. Dropout is a powerful regularization technique that can be applied to a wide range of models [16]. It is a cost-effective way to prevent complex co-adaptations of neurons, which can result in overfitting and poor generalization on unseen data [27].

Dropout works by temporarily removing a neuron and all its connections from the network during training with a certain probability. By randomly removing neurons, dropout forces the remaining neurons to take on more or less responsibility for the inputs, making the training process noisy and leading to better generalization on held-out data. [27] Moreover, dropout can promote sparse representations of hidden units. This can be useful for autoencoder models, which are a type of neural networks where the input and output are the same. [13]

Although a dropout probability of 0.5 is often effective, the effect of dropout doesn't depend strongly on the dropout probability [27]. Dropout is often combined with other regularization techniques, such as L2 regularization, in practice [16, 25]. It is not recommended to use dropout on the first layer of a network, as it can remove important parts of the input dataset [13]. Moreover, dropout is not applied to the output layer, and it is important to note that dropout regularization is not used during testing or prediction [16].

L2 regularization is a regularization technique that works by adding a penalty equivalent to the square of the magnitude of weights in the logistic regression cost function. The hyperparameter α determines how much the model is penalized and affects the weights and complexity of the logistic regression. L2 regularization reduces complexity but does not aid in feature selection and shrinks all weights, but does not make them exactly zero. In contrast, L1 regularization works by adding a penalty equivalent to the sum of the absolute values of weights. L1 regularization uses absolute weight values for normalization and results in some weight estimates being exactly zero. Therefore, L1 regularization performs feature selection and results in sparse models. Both L1 and L2 regularization are used to reduce overfitting, but L1 regularization also has a feature selection behavior. [13, p. 111]

Data augmentation is a powerful regularization technique in ML that can enhance the performance of models. Collecting real data can sometimes be difficult and time-consuming, and data augmentation provides a cheaper alternative. The amount of data is increased by creating new variations of the existing data artificially. For example, image data can be slightly rotated, shifted, and resized, creating new instances that are still representative of the original image. By doing so, it's possible to reduce overfitting and improve generalization, as it encourages the model to learn more meaningful and invariant features. [13, 17]

Early stopping is another popular method to avoid overfitting used especially in neural networks. This technique involves measuring the performance of the model after each iteration during the iterative training process. The performance criteria could be accuracy or cost on the validation dataset. With early stopping, one of these criteria, such as cost, is monitored until a certain iteration where the cost starts to increase. This is the point where the model loses the local or global minima, causing the cost to increase, and the model's ability to generalize may weaken, resulting in overfitting of the training data. Therefore, this iteration is considered the best point to stop training the model. [13, p. 109]

Convolutional Neural Network (CNN) is a type of ANN that has been specifically designed for image recognition and computer vision tasks. The idea behind CNNs is to use a hierarchical structure that resembles the way the human visual system processes images. The network is trained to detect increasingly complex features in the data by using a series of convolutional layers. [17, p. 431]

Convolutional layers apply filters, also known as kernels, to the input image. Each filter slides over the image and performs a dot product between its weights and the pixel values within the region that it covers. The output is a feature map that indicates the presence or absence of that particular feature at each location in the input image. By stacking multiple convolutional layers on top of each other, the network can learn to detect increasingly complex features, such as edges, corners, and textures, as well as more abstract concepts like faces and objects. [13, p. 128]

Pooling layers are another important component of CNNs. These layers downsample the feature maps produced by the convolutional layers, reducing the spatial dimensionality of the data. This makes the network more efficient and reduces the risk of overfitting. The most common form of pooling is max pooling, which takes the maximum value within each pooling region. [17, p. 442-444]

CNNs have several advantages over other types of neural networks. They are particularly effective at learning hierarchical representations of the input data, where lower-level features are gradually combined to form higher-level abstractions. This makes them well-suited for tasks like object recognition, where objects can be decomposed into a set of visual features that can be detected at multiple scales and locations within the image. [13, p. 128] CNNs have achieved state-of-the-art performance on a wide range of image recognition tasks, including object recognition, image segmentation, and facial recognition. They have also been successfully applied to other domains like natural language processing and voice recognition [17, p. 431].

3. DATA COLLECTION AND PREPROCESSING

3.1 Dataset collection

The success of any ML model depends heavily on the quality and quantity of data used to train it. In this chapter, the focus is on the data collection process for a ML model that classifies profile joints using data related to the member profiles in building structures. The data consists of various features including geometric properties of the profiles and their spatial relation. Subchapter 3.2 goes into more detail on the specific types of features that were collected from the building models.

The search for suitable existing datasets to be used in the training was unsuccessful, which means that the entire dataset used in this research was collected during the thesis process. Due to time constraints, the final dataset does not cover all profile joints that are available in the software. Instead, the training is done with a subset of profile joints that should still give an acceptable approximation of how the model might perform with a larger set. The dataset was extracted from 80 professionally designed building models, which resulted in a dataset with a size of 1.65 GB when stored in Comma-Separated Value (CSV) format.

The first experiments were done with a very small dataset as the data collection began simultaneously during the implementation phase. This method allowed ML models to be trained on all currently available data, and as more models became available, the dataset was easily expanded without the need to re-extract data from the previous models. If new features were introduced to the dataset, the extraction process would still need to be repeated. For the comparisons of the different ML classifier implementations, the same dataset was used each time. Figure 3.1 displays the limited subset that was used for the comparisons. The different profile joint types are represented with a number between 0 and 19.

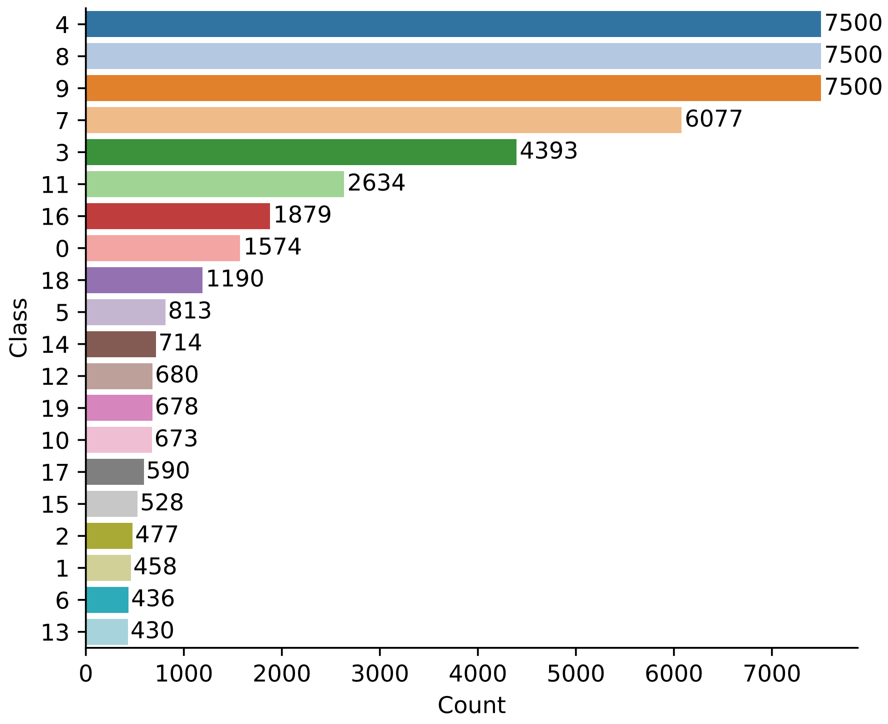


Figure 3.1. *Distribution of the classes in the dataset*

The data collection process revealed an imbalance in the usage frequency of profile joints. The effects of the imbalance were investigated and data balancing methods such as Synthetic Minority Oversampling Technique (SMOTE) was experimented with [28]. These methods and their results are discussed in Subchapter 3.3.2. Due to the nature of the building structures, a portion of the data extracted from the building models was either very similar or even identical. This is primarily due to the repetition of profile structures within a building, as shown in Figure 3.2. In addition to the repetition in a single structure, many design patterns are repeated from one building to another too.

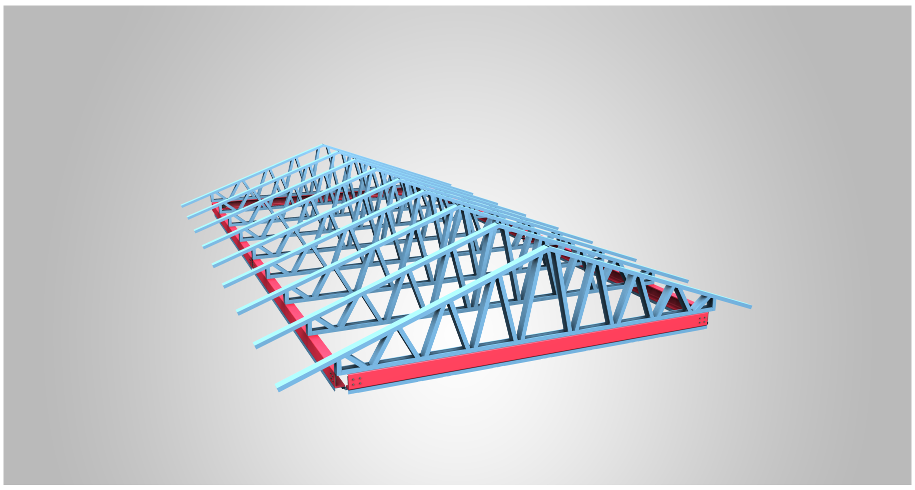


Figure 3.2. *Repetitive structures*

Repetition within the dataset poses a risk for data leakage, where some of the duplicates end up in both training and test sets, which can lead to overly optimistic performance metrics and inaccurate evaluations. The repetition proved to be an interesting topic to investigate and is discussed more in Subchapter 3.3.2.

Collecting training data from professionally designed projects was not the only option available. New data could be created manually, but this method would require lots of work and some knowledge and skills to be able to design the structural model and choose the correct profile joints. To reduce the workload, some automation could be done, but it was still considered out of the scope of this thesis. Instead, the previously mentioned method, SMOTE, was explored to augment the data by automatically generating new data samples based on the existing data. However, it should be noted that SMOTE can't create new data for those profile joints that don't already have some existing samples.

3.2 Dataset structure

The extraction process generates multiple separate Comma-Separated Value (CSV) files. The CSV format was chosen due to its versatility, as it can be easily read and written using various programming languages, and is also compatible with any basic text editors. The full dataset is split into three files that contain an equal number of rows, where each row represents a single sample. Two of these files are reserved for the input features, and the third contains the corresponding labels. The first file includes rows with two 16x16 pixel binary images of the cross-sections, which are flattened and concatenated. The binary images were not utilized in the current implementation, and the reasons for this are presented in Subchapter 4.1.2. The second file contains the rest of the input features in a structured format where each column represents a single feature.

The labels in the third file include the profile joint names as well as the names of the libraries to which each joint belongs. Label encoding was then applied to transform each unique label into a unique integer, as previously displayed in Figure 3.1. This step is essential because most ML algorithms require numerical representations [17]. The label encoding process also involves saving the relations between the original labels and the encoded numerical representations, facilitating the decoding of predictions back to the original labels. Finally, the extraction process records information related to the building models, such as the number of unique types of profile joints and the total number of samples extracted.

3.2.1 Feature descriptions

In this subchapter, a description of the features used in the final dataset is provided. The features were initially chosen based on what data was easily available, and were subsequently evaluated using methods such as mutual information scoring, random forest scoring, and SHAP scores to assess their relevance and importance within the problem domain. These methods and their results are discussed in Subchapter 3.2.2. Features that were found to have low relevance or importance were left out of the analysis. Table 3.1 summarizes the descriptions of all the features used in the final models.

Table 3.1. *Summary of the features in the dataset*

Feature	**Description**
`SlopeA`	Slope between the first profile and the horizontal plane
`SlopeB`	Slope between the second profile and the horizontal plane
`MeetRatio`	Ratio of the overlapping area between the profiles
`AngleXY`	Angle between the two profiles in the horizontal plane
`SectProportionA`	First profile's cross-section height divided by the sum of its height and width
`SectProportionB`	Second profile's cross-section height divided by the sum of its height and width
`ProfLengthProportion`	Length of first profile divided by the sum of its length and the second profile's length
`ProfEndDistProportion`	Proportion of minimum Euclidean distance between the profiles to the sum of the minimum and maximum
`SectAreaA`	Area of the first cross-section
`SectAreaB`	Area of the second cross-section
`ProfWeightA`	Weight of the first profile
`ProfWeightB`	Weight of the second profile
`ProfLengthA`	Length of the first profile
`ProfLengthB`	Length of the second profile

Each sample's input data is derived from two profiles, and several features are extracted separately from each profile. To differentiate between them, features from the first profile are labeled with the suffix `A`, while those from the second profile are labeled with the suffix `B`. Figure 3.3 illustrates how the features `SlopeA` and `SlopeB` are calculated. The selected angle is the smallest of the possible options, limiting the values to a range of 0 to 90 degrees.

Knowing the possible range, the values can be easily scaled to a range of 0 to 1 by dividing them by 90 degrees. This scaling is necessary for some of the classifier algorithms used in this thesis, as they expect input within this range [17, p. 72]. A low value indicates a horizontal profile, while a high value signifies a vertical profile.

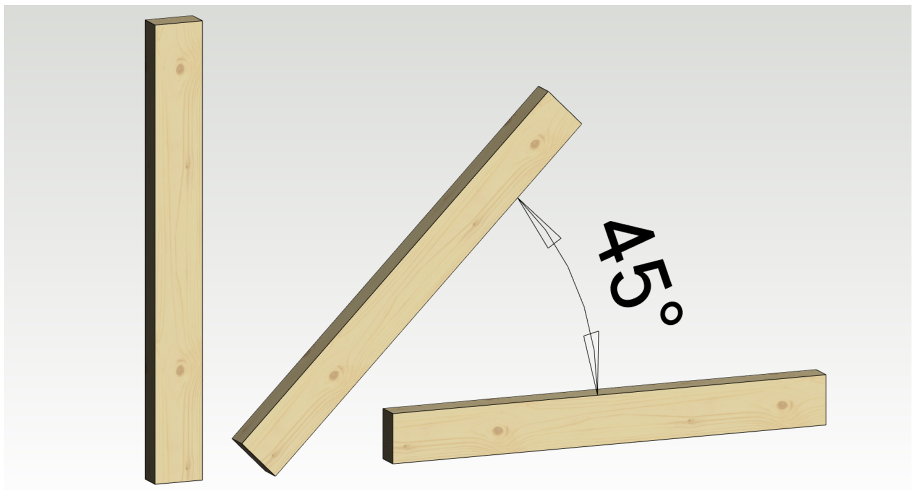


Figure 3.3. *Slope feature*

Figure 3.4 demonstrates the calculation of the `AngleXY` feature using an example. Similar to the `Slope` feature, it can be scaled from a range of 0 to 90 degrees to the optimal 0 to 1 range using a straightforward division.

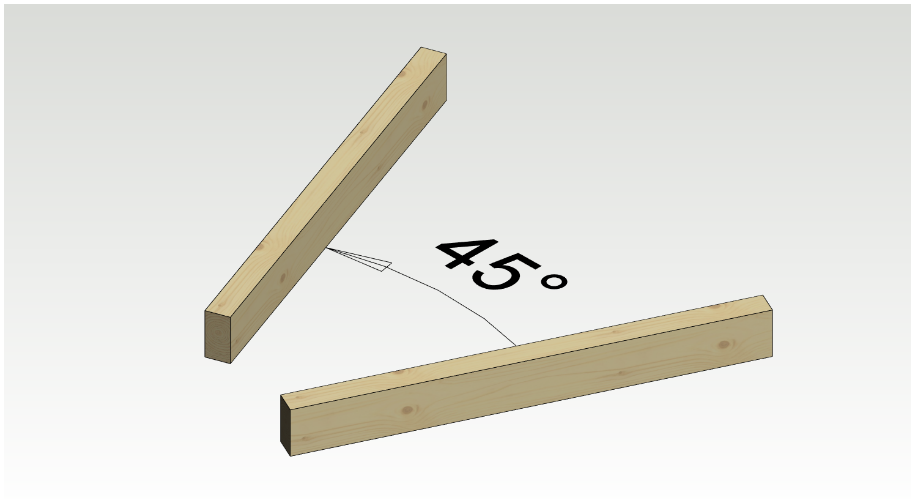


Figure 3.4. *AngleXY feature*

Figure 3.5 shows the calculation of the `SectProportion` feature. Unlike the previous two features, it is dimensionless and does not require scaling. Taller cross-sections have higher values, while wider cross-sections have lower values.

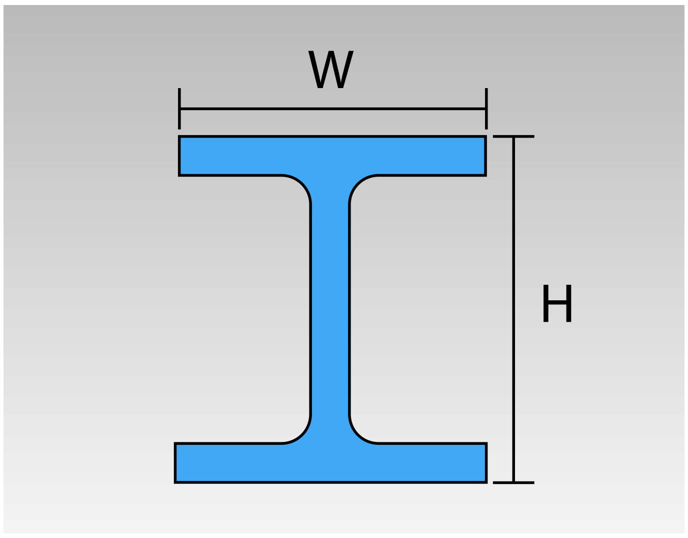


Figure 3.5. *SectProportion feature:* $H/(W+H)$

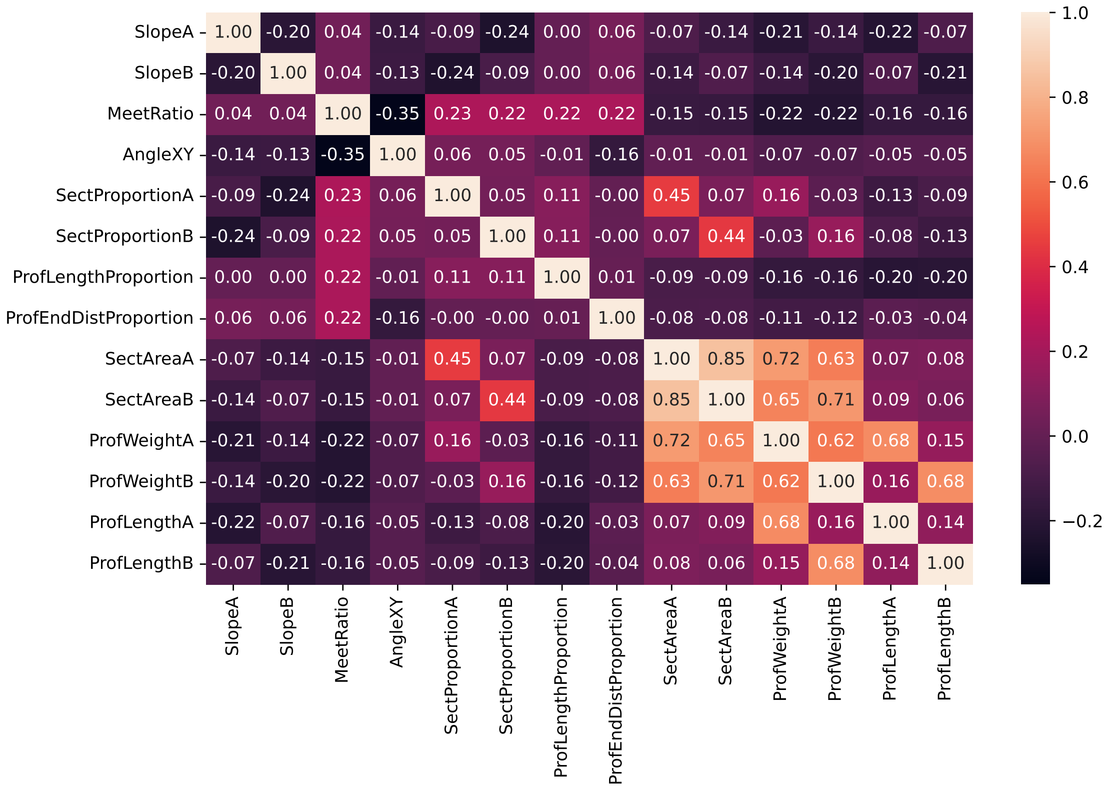
The size of the dataset increased during the project, but for most experiments and the final results, only a subset of the available data was used. This decision was made for two reasons. First, the entire dataset is so large that processing it can be time-consuming, and training the classifiers on the full dataset can be prohibitively slow. Second, modifying the dataset continuously during the course of the project could invalidate the results obtained from previous experiments.

As the dataset grew in size, so did the number of unique features. While exploring the data, new ideas for features arose. However, once the subset of data was locked in, any new features were deferred to future iterations of this project. Consequently, they have not been included in this thesis.

3.2.2 Feature evaluation and selection

Evaluation and selection of features can be accomplished in multiple ways, and in some cases, it may not be necessary at all, as some classifiers have built-in feature selection methods in their training process. To better understand how classifiers work, it is important to evaluate features in some way. In this subchapter, the correlation matrix is first utilized to visualize the linear relationships of the features. Then three different methods, mutual information, mean absolute SHAP scores, and feature importances by the `RandomForestClassifier` are also used to evaluate the features. [29, 30]

Figure 3.6 displays the correlation matrix of the features in the dataset. It was created using the `pandas.DataFrame.corr()` method which computes pairwise correlation of the features. From it the strength and direction of the linear relationships between each pair of variables can be seen. If there were any features with a perfect positive or negative correlation, the features would be redundant, and a single feature would essentially contain all the information already. In this case there are no features that perfectly correlate, but for example the two `SectArea` features have a correlation of 0.85. This means that when the other profile's cross-section area increases, the other profile's size usually increases as well. It can be tested if these features are actually redundant by removing the other one and training the model and comparing the results. In this case without the other `SectArea` feature, the classifier accuracy dropped by approximately 0.3%, which meant both features were kept in the dataset. The `ProfWeight`, `ProfLength`, and `SectArea` features have the highest correlations in the dataset. It's possible that they reveal other features that aren't present in the dataset as a distinct feature like the density of the profile, for instance. In this case feature engineering in the form of creating this separate feature wasn't found to have a noticeable effect on the classifier accuracy and was left out. This could indicate that either the density feature is not informative for the classification, or that the model may already be capturing the relevant information through the other features. [17, p. 62]



	SlopeA	SlopeB	MeetRatio	AngleXY	SectProportionA	SectProportionB	ProfLengthProportion	ProfEndDistProportion	SectAreaA	SectAreaB	ProfWeightA	ProfWeightB	ProfLengthA	ProfLengthB
SlopeA	1.00	-0.20	0.04	-0.14	-0.09	-0.24	0.00	0.06	-0.07	-0.14	-0.21	-0.14	-0.22	-0.07
SlopeB	-0.20	1.00	0.04	-0.13	-0.24	-0.09	0.00	0.06	-0.14	-0.07	-0.14	-0.20	-0.07	-0.21
MeetRatio	0.04	0.04	1.00	-0.35	0.23	0.22	0.22	0.22	-0.15	-0.15	-0.22	-0.22	-0.16	-0.16
AngleXY	-0.14	-0.13	-0.35	1.00	0.06	0.05	-0.01	-0.16	-0.01	-0.01	-0.07	-0.07	-0.05	-0.05
SectProportionA	-0.09	-0.24	0.23	0.06	1.00	0.05	0.11	-0.00	0.45	0.07	0.16	-0.03	-0.13	-0.09
SectProportionB	-0.24	-0.09	0.22	0.05	0.05	1.00	0.11	-0.00	0.07	0.44	-0.03	0.16	-0.08	-0.13
ProfLengthProportion	0.00	0.00	0.22	-0.01	0.11	0.11	1.00	0.01	-0.09	-0.09	-0.16	-0.16	-0.20	-0.20
ProfEndDistProportion	0.06	0.06	0.22	-0.16	-0.00	-0.00	0.01	1.00	-0.08	-0.08	-0.11	-0.12	-0.03	-0.04
SectAreaA	-0.07	-0.14	-0.15	-0.01	0.45	0.07	-0.09	-0.08	1.00	0.85	0.72	0.63	0.07	0.08
SectAreaB	-0.14	-0.07	-0.15	-0.01	0.07	0.44	-0.09	-0.08	0.85	1.00	0.65	0.71	0.09	0.06
ProfWeightA	-0.21	-0.14	-0.22	-0.07	0.16	-0.03	-0.16	-0.11	0.72	0.65	1.00	0.62	0.68	0.15
ProfWeightB	-0.14	-0.20	-0.22	-0.07	-0.03	0.16	-0.16	-0.12	0.63	0.71	0.62	1.00	0.16	0.68
ProfLengthA	-0.22	-0.07	-0.16	-0.05	-0.13	-0.08	-0.20	-0.03	0.07	0.09	0.68	0.16	1.00	0.14
ProfLengthB	-0.07	-0.21	-0.16	-0.05	-0.09	-0.13	-0.20	-0.04	0.08	0.06	0.15	0.68	0.14	1.00

Figure 3.6. *Feature correlation matrix*

The next method is mutual information which is a statistical measure that can be used to analyze the relationships between variables in a dataset. Unlike correlation coefficients, which measure only linear relationships, mutual information is a non-parametric measure that can capture both linear and non-linear relationships. The mutual information score represents the amount of information that a feature provides about the target variable in the dataset, which in this case is the type of the profile joint. A higher score indicates that the feature is more informative about the target variable, while a lower score indicates that the feature is less informative. [31]

To calculate mutual information, scikit-learn's `mutual_info_classif` function from the feature selection module was used. Scikit-learn is an open-source Python library that provides a wide range of ML algorithms and tools for data analysis and modeling. It includes algorithms for classification, regression, clustering, and dimensionality reduction, as well as preprocessing tools, model selection, and evaluation. [30]. The mutual information scores for each feature are displayed in Figure 3.7. A higher score indicates that the feature is more informative about the target variable. In this project, a few original features had a low mutual information scores and when they were removed from the dataset the classifier performance was unaffected. [30]

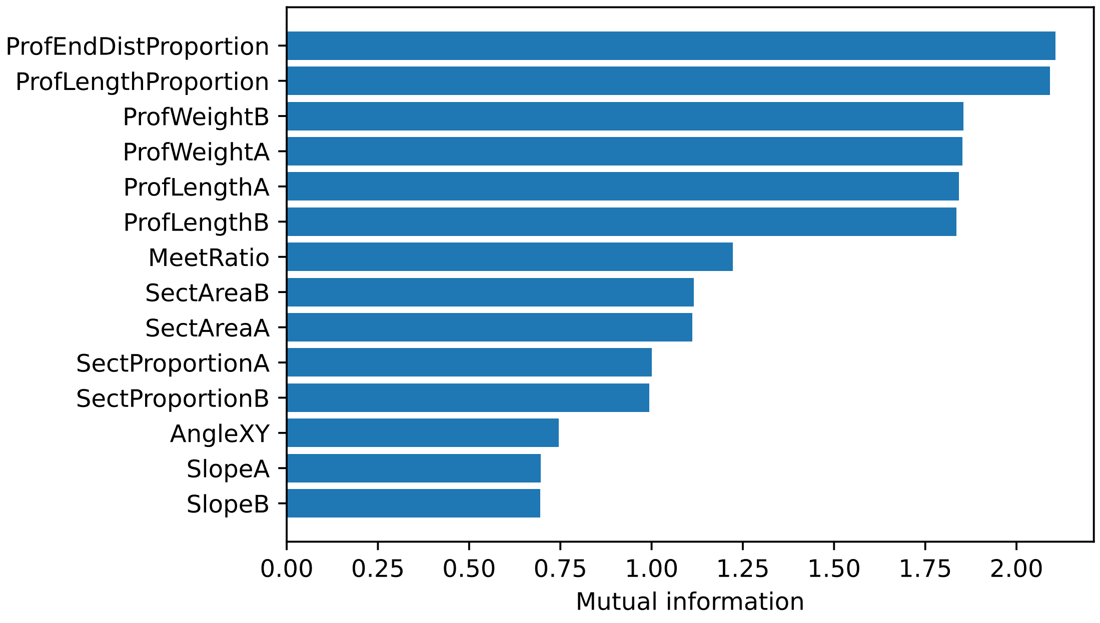


Figure 3.7. *Mutual information chart*

The next method for evaluating feature importance is the mean absolute SHAP score, which measures the average impact of a feature on the magnitude of the model's output. SHAP (SHapley Additive exPlanations) is a game-theoretic approach to explain the output of any ML model. The SHAP value of a feature represents the contribution of that feature to the difference between the actual prediction and the baseline prediction. The baseline prediction represents what the model would predict if it didn't have any information about a specific instance's feature values. [29]

To calculate the SHAP values for each feature, the SHAP library in Python was used. The results are shown in Figure 3.8. By analyzing these scores, it's possible to determine which features have the greatest impact on the model's output magnitude. [29]

Compared to mutual information and correlation coefficients, the mean absolute SHAP score can provide a more nuanced understanding of the relationships between features and the target variable. It also displays the results separately for each class indicating how some classes might be much more affected by some features than others. For example the `ProfEndDistProportion` for class 8 appears to have very high value, which indicates that the feature has an important role determining whether a sample belongs to class 8. Most samples that belong to class 8 have very low `ProfEndDistProportion` values, while all the other classes have a wider distribution for this feature. In some cases, features with low mutual information scores or weak correlations may still have a significant impact on the model's output magnitude. In this project, the SHAP results were a bit different from the mutual information results, but the lowest ranking features, which are not present in these figures, were still the same.

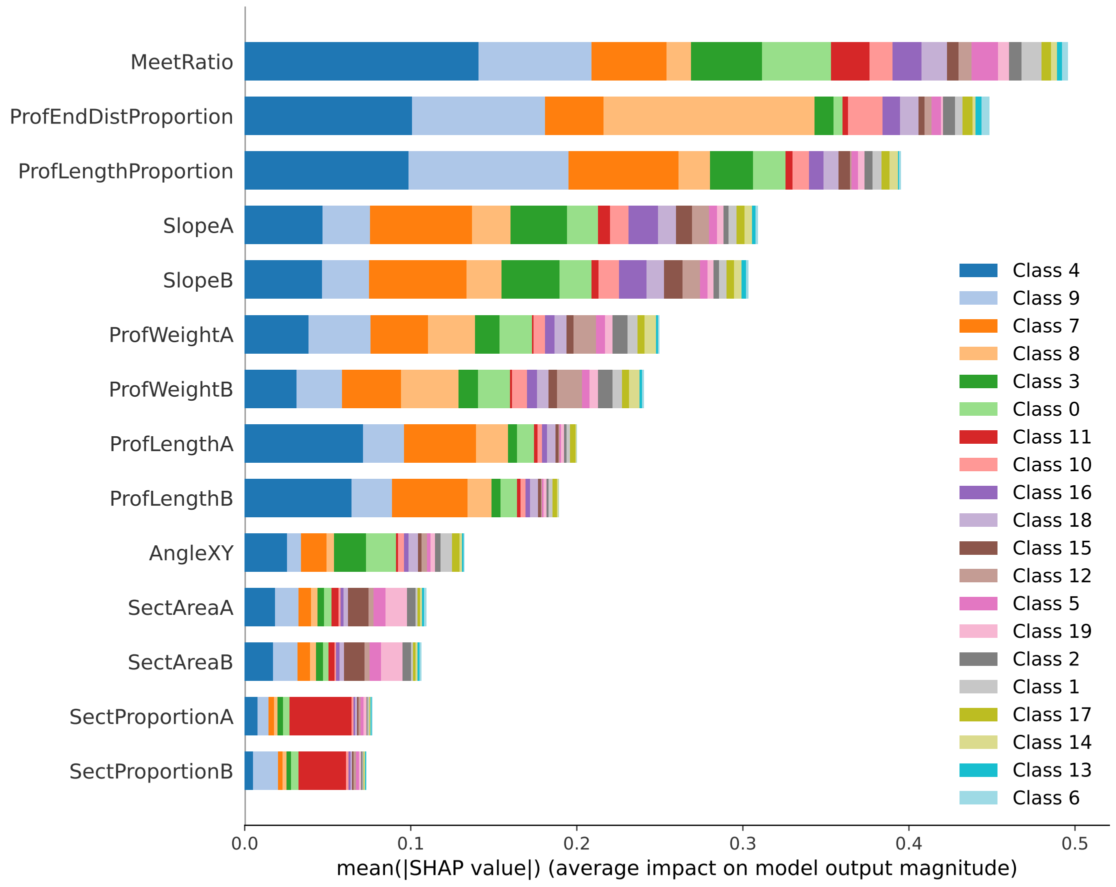


Figure 3.8. *SHAP summary*

Finally, the feature evaluation can also be achieved using the `RandomForestClassifier` algorithm in scikit-learn, which provides a measure of the importance of each feature for the classifier's decision-making process. The feature importance score represents the reduction in the impurity of the decision tree when a feature is used for a split. [21, 30]

To calculate feature importances using the `RandomForestClassifier`, a model was trained using the training data and the feature importance scores were then extracted from the model. The results are shown in Figure 3.9. By analyzing these scores, it's possible to determine which features have the greatest impact on the classifier's decision-making process. [30]

Compared to mutual information and mean absolute SHAP score, feature importances from `RandomForestClassifier` provide a more direct measure of how each feature affects the classifier's predictions. It's important to note that feature importances can be sensitive to the specific algorithm and parameters used, and may not always provide a complete picture of the relationships between features and the target variable. Therefore, it's important to use multiple methods for evaluating feature importance and to interpret the results in the context of the specific problem being addressed. In this case, these results are very similar to the SHAP results.

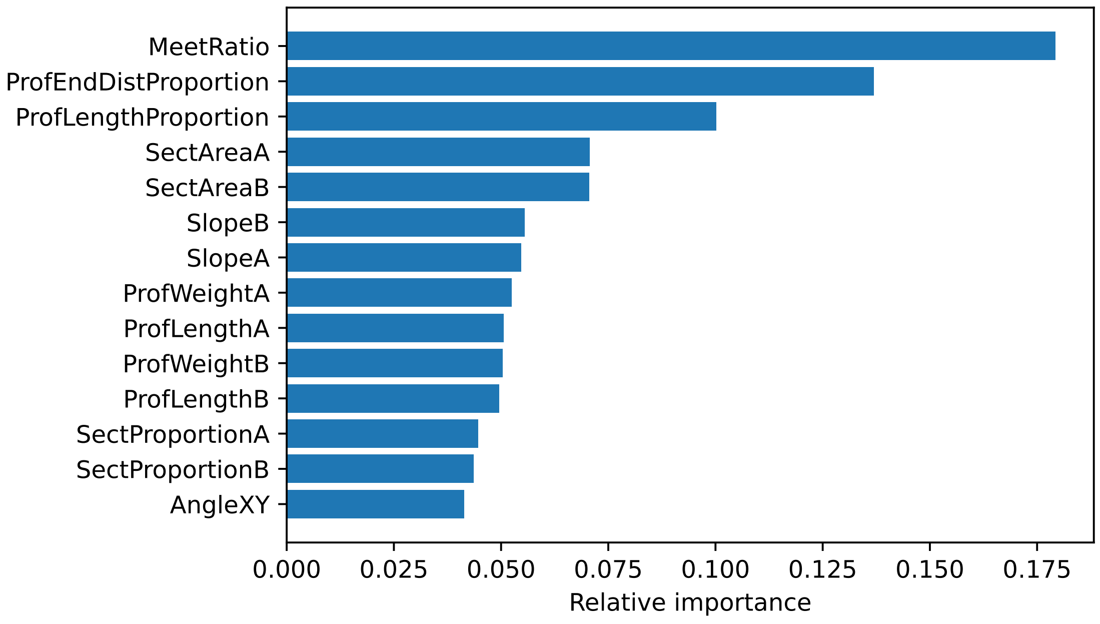


Figure 3.9. *Random Forest feature importances*

The results of the feature evaluation suggest that certain features are more informative than others for predicting the type of profile joint. Although these results alone can't always tell which features are useful and which are not, they can provide information that can then be verified by additional testing. By selecting the most informative features, it's possible to improve the performance of the classifiers and obtain more accurate predictions. [30]

3.3 Data preprocessing

The full dataset consists of both numerical and categorical features. Because the categorical features were deemed less useful according to the feature evaluation, they were excluded from the training. The subset utilized in the thesis contains both dimensionless numerical features, which are already distributed between 0 and 1, and other numerical features, such as profile lengths and weights, that have a range that is not known in advance. To improve the distribution of the second group of features and mitigate the impact of outliers, a two-step transformation was applied. First, they were logarithmically transformed, and then scaled to a range of 0 to 1. The scaling is based on the extremes of the training set, which means that new samples might not fall within the same range. None of the classifiers used in this thesis require strict ranges for the input values which means this should, at worst, lower the classification accuracy for those samples [30, 32, 33].

3.3.1 Dimensionality reduction

The preprocessed dataset samples were projected onto a 2D space using the Uniform Manifold Approximation and Projection (UMAP) method after reducing the original 14 dimensions. UMAP is a state-of-the-art non-linear dimensionality reduction technique that aims to preserve both global and local structure in the data. The `UMAP` Python library by McInnes et al. was used for this purpose. [34] Figure 3.10 shows the resulting visualization of the data after applying UMAP. Although there are clear clusters of samples, some overlap can also be observed, indicating that some samples from different classes may be similar. To address the stochastic nature of the UMAP method, different random seeds were used and compared. By visual inspection, the results were found to be consistent. The UMAP parameters used were `n_neighbors=50`, `min_dist=1`, and the algorithm was run for 1000 epochs. Finally, the results were visualized using Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to cluster the samples for each class and Kernel Density Estimate (KDE) plots were used to represent the clusters [30].

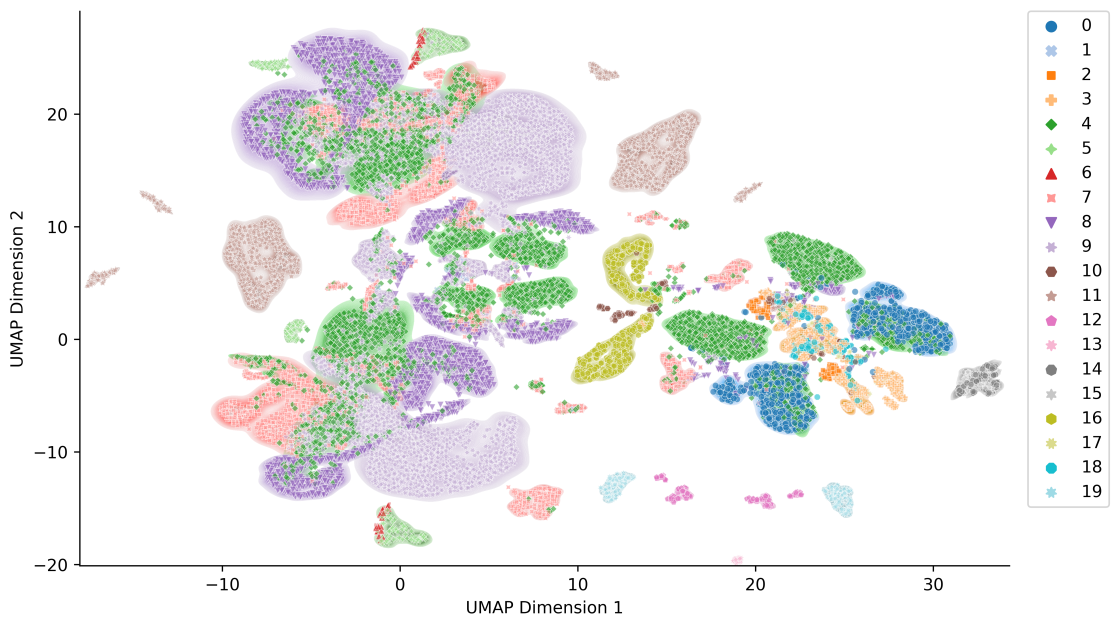


Figure 3.10. *UMAP results*

In the UMAP figure some classes clearly are spread out much more when others are packed very tightly. Most classes form multiple separate clusters, and while this could mean many things, one theory is that many of the profile joints can be used in multiple ways. Also, as two versions of each profile joint sample exist in the dataset to account for selecting the profiles in either order, this is a likely reason for forming at least the two separate clusters that can be seen for nearly every class. It raises the question, whether it would've been better to link the order to some other parameters, such as profile length to avoid the need to add each joint twice to the dataset.

Figure 3.11 displays the connectivity plot of the UMAP result, which was created using the `plot.connectivity()` method from the UMAP library. The parameter `edge_bundling` was set to `"hammer"` to create a less busy view of the graph. Each line represents an edge or a nearest neighbor in the weighted graph that UMAP creates when reducing the dimensionality of the data. It can help in distinguishing the relations between the clusters more easily. Missing connections between two clusters indicate that they are likely well-separated in the original high-dimensional space. It was created on the same UMAP run as Figure 3.10 to allow direct comparisons. [34]

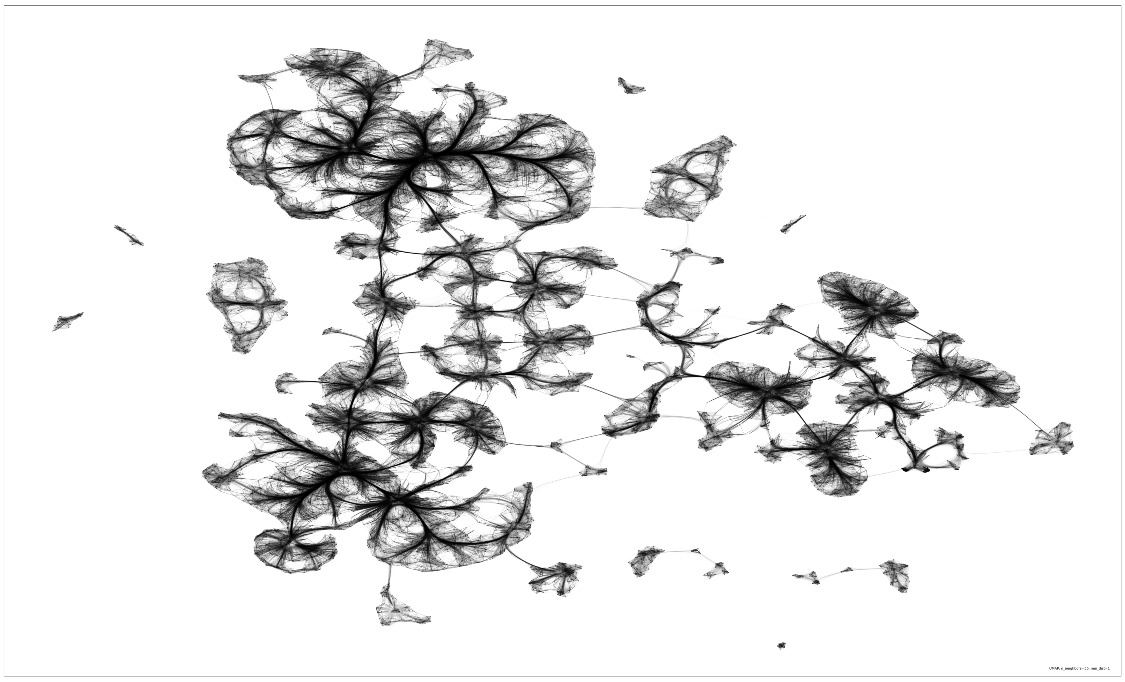


Figure 3.11. *UMAP connectivity plot*

For comparison purposes, t-Distributed Stochastic Neighbor Embedding (t-SNE) was used to create a similar visualization of the data. The algorithm from scikit-learn's manifold module based on the research by Maaten and Hinton was used for this purpose [30, 35]. The resulting t-SNE plot is presented in Figure 3.12. As the t-SNE is a stochastic algorithm as well, the results obtained using different random seeds were also compared. By visual inspection they too converged to similar end results. Clustering and KDE plots were applied to t-SNE for visualization similarly as in Figure 3.10.

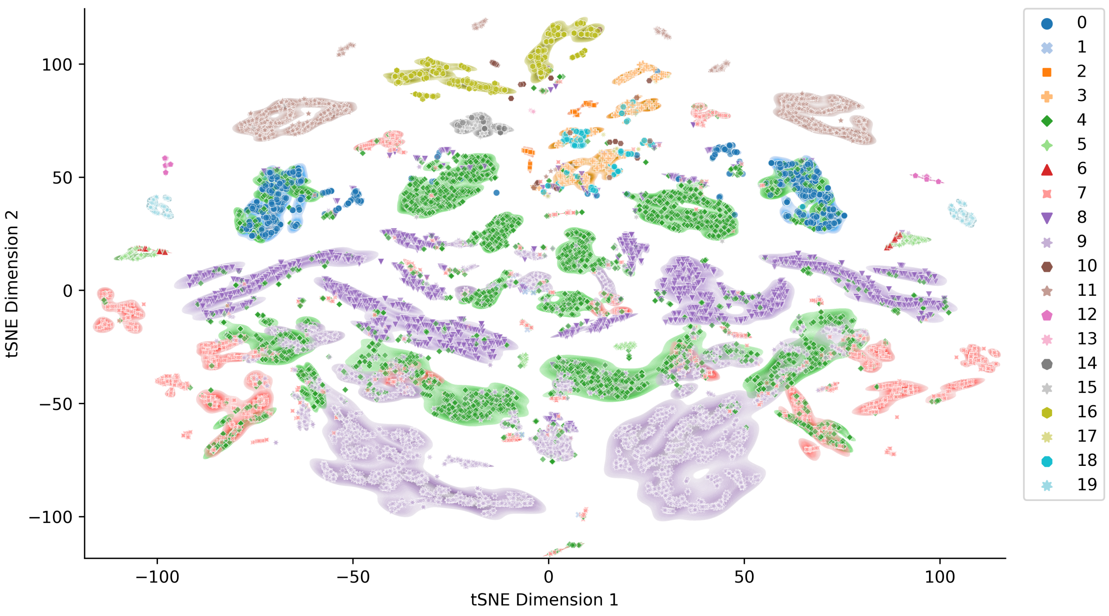


Figure 3.12. *t-SNE results*

Compared to UMAP, t-SNE can sometimes be better at preserving the local structure of the data, while UMAP is generally considered to be better at preserving the global structure [36]. The t-SNE plot definitely has differences compared to the UMAP plot, but similarities can be observed too. For example, class 16 has two distinct main clusters of similar shapes in both plots.

3.3.2 Dataset splitting and balancing

The dataset had two significant aspects to examine. First, the dataset contained duplicated samples and it poses a risk of data leakage. Second, the number of samples in each class was not evenly distributed, which can impact the classification performance negatively. In this subchapter both these aspects are discussed. The evaluation of these methods is conducted by training a neural network with the same hyperparameters each time. It has four dense layers with dropout layers in between. The details of the neural network are explained in Subchapter 4.1.2.

The dataset contained natural repetition, necessitating an exploration of various methods to prevent duplicate data from impacting the validation sets. One considered method was the removal of all duplicate samples from the dataset. This approach might alter the distribution of different classes, potentially leading to a negative impact on the classifier's performance. Due to limited data availability, confirming this effect was challenging. Tests on a dataset without duplicates yielded promising results, but only seven classes retained more than 100 samples. As a result, further analysis using this method was discontinued.

Another method involved allowing duplicate data in the training and validation sets, provided that it originated from separate building projects. Unfortunately, this approach also encountered difficulties due to limited data availability. With 80 building projects accessible, splitting them into training, validation, and test projects while ensuring enough samples for each class was unattainable. Experiments using a limited test set revealed performance similar to the other methods, but some classes had very few or no samples.

A third method was using k-fold cross-validation. This approach partitioned the data into k-folds and trained and validated the classifier k times, using a different fold as the validation set each time. This method facilitated a more comprehensive evaluation of classifier performance while reducing the possible effect of duplicate samples. However, it required more computational resources and time. Following testing, it was determined that using 5 folds ($k = 5$) offered a suitable balance between resources and performance. [13, p. 99].

Finally, a method aimed to split the data in a manner that would prevent duplicates from appearing in both sets was experimented with. Two variations of this method were tested: one in which data present in both sets was eliminated from the validation sets, and another where the same data was moved to the training set to preserve more of the data. The first variation yielded on average 2% lower accuracy compared to not deleting any data. The second variation did not have a meaningful difference on the accuracy over the first variation.

Ultimately, all these methods proved that the repeated samples can have an influence on the evaluation metrics. Since the difference was only around 2%, the comparisons between the classifiers were done without utilizing the methods here. In the future dataset iterations, the duplicate samples are less likely since the number of different features is increased.

The class imbalance was the other significant aspect to examine. The full dataset has a little over million samples, of which almost 70% belongs to a single class. However, the subset of data that was actually utilized is not as extreme. To slightly balance the dataset and improve the training speed of the classifiers, classes with more than 7500 samples were undersampled and minority classes with less than 250 samples were dropped. The training data split has 32706 samples and the distribution of classes is displayed in Figure 3.13.

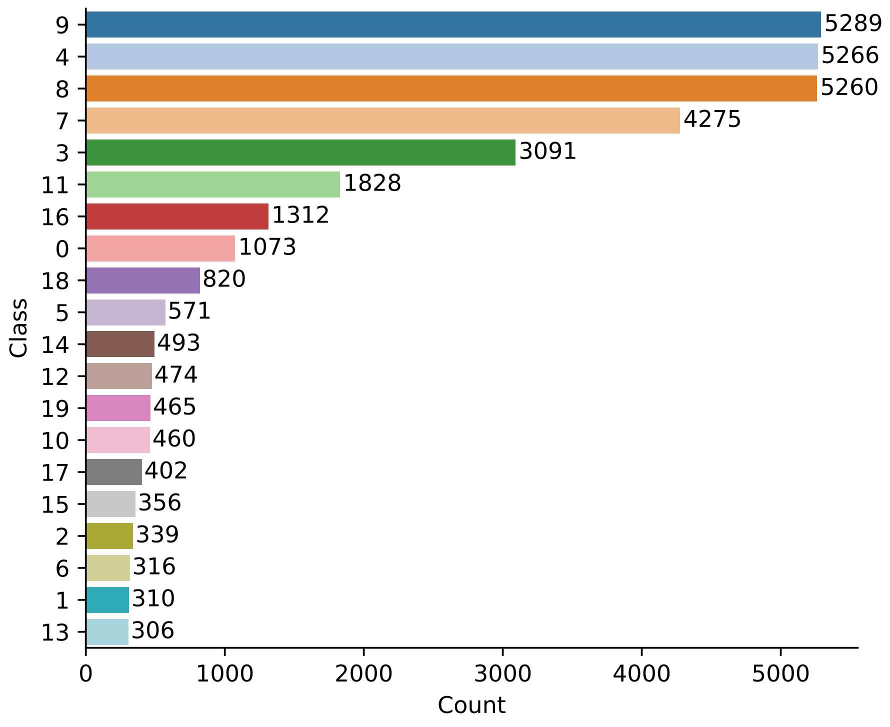


Figure 3.13. *Distribution of the classes in the training subset*

As the class imbalance might have a negative impact on the classifier performance, some methods to mitigate this were experimented with. First method was to give the classes with fewer samples a higher weight during the training. For a Keras model this was done with the `class_weight` parameter. This improved the classifier performance slightly with

0.2% higher accuracy on average. [32]

Second method was to use oversampling using the Imbalanced-learn (Imblearn) Python library to generate new samples using the existing data as a template. Imbalanced-learn is an open source library that relies on scikit-learn, and it provides tools to deal with imbalanced datasets. [37]. Two strategies to oversample the data were experimented with.

First strategy was to generate samples until a uniform distribution is reached, which increases the number of samples in the training split from 32706 to 105780. The number of synthetic samples is 73074, which is almost 70% of the total number of samples. For the previously smallest class the proportion of synthetic samples is over 94% of the total number of samples for that class. Using this strategy approximately doubled the time required to train the model.

The second strategy was to only partially oversample the minority classes resulting in 42028 samples in total. The number of synthetic samples using this strategy is 9322, which is a little over 22% of the total number of samples. Figure 3.14 displays the resulting class distribution. When using the k-fold cross validation method, the number of samples in each class can slightly differ between folds, but the distribution is generally similar. The second strategy increased the training time by 20% on average.

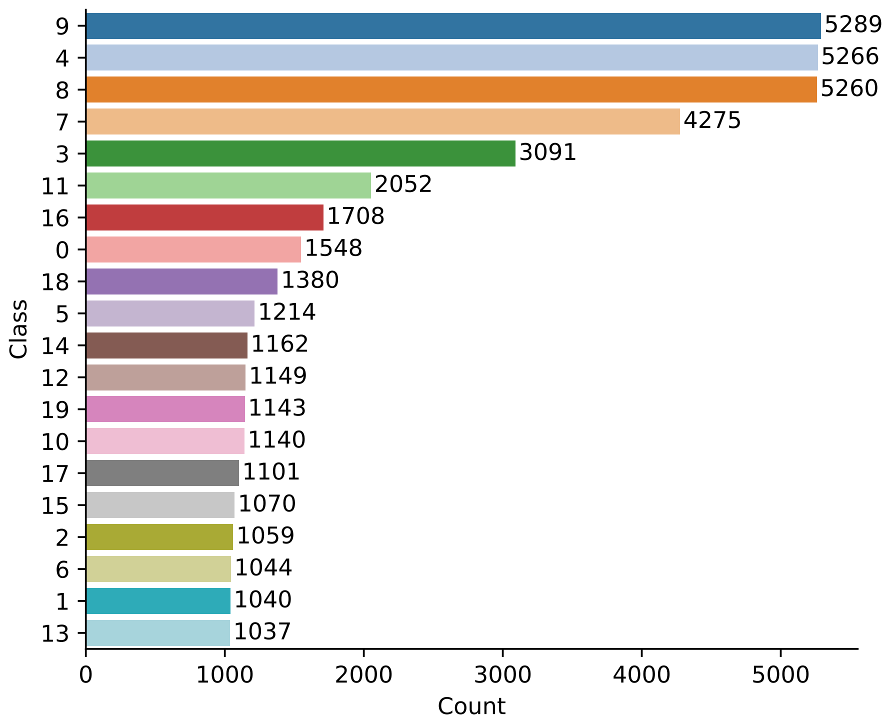


Figure 3.14. *Distribution of the classes in the training subset after oversampling*

Both strategies were applied for three different oversampling methods from Imblearn. The methods were SMOTE, KMeansSMOTE, and RandomOverSampler. SMOTE is Imbalanced-learn's implementation of the Synthetic Minority Oversampling Technique (SMOTE) proposed by Chawla et al. [28]. KMeansSMOTE is a variant of SMOTE that applies k-means clustering before oversampling the data. RandomOverSampler simply oversamples the minority classes by picking samples at random with replacement, creating duplicate samples instead of new artificial ones. Default hyperparameters were used for these algorithms. [37, 38] Table 3.2 presents the accuracies and the f1-scores achieved with each method. F1-score is a metric used to evaluate the performance of classification models, particularly in situations where class imbalance is present. A more detailed definition is presented in Subchapter 5.1. For the f1-scores, both the weighted and the macro-averaged results are presented. The weighted result takes the class sizes into account favoring the larger classes. The macro-averaged result isn't affected by the class sizes, which makes all the classes have an equal effect. Also, the standard deviation for each result in the folds is displayed. [30]

Table 3.2. *Average oversampling results using k-fold cross validation with* $k = 5$

Method	**Accuracy**	**Weighted / macro-averaged f1-score**
No oversampling	97.20% (± 0.19%)	97.18% (± 0.19%) 97.16% (± 0.35%)
No oversampling (class_weight)	97.44% (± 0.19%)	97.43% (± 0.19%) 97.61% (± 0.14%)
SMOTE	97.47% (± 0.16%)	97.47% (± 0.16%) 98.00% (± 0.19%)
SMOTE (limited strategy)	97.47% (± 0.20%)	97.46% (± 0.20%) 97.70% (± 0.17%)
KMeansSMOTE	97.67% (± 0.12%)	97.66% (± 0.12%) 98.19% (± 0.20%)
KMeansSMOTE (limited strategy)	97.37% (± 0.11%)	97.36% (± 0.11%) 97.66% (± 0.16%)
RandomOverSampler	97.44% (± 0.18%)	97.43% (± 0.18%) 97.99% (± 0.21%)
RandomOverSampler (limited strategy)	97.41% (± 0.21%)	97.41% (± 0.21%) 97.73% (± 0.19%)

There was some deviation in the results and running all tests again could potentially have a significant effect. For example, on a few test runs the highest accuracy was achieved by SMOTE instead of KMeansSMOTE. The macro-averaged f1-scores appear to have increased more than the weighted f1-scores, which suggests that the different methods improved the model's ability to classify the minority classes, but the overall performance was less affected. The conclusion is that since the overall improvements were not very substantial, class imbalance does not affect the results too much for the purposes of the profile joint classifier. Consequently, oversampling was not used in the training of the final models.

4. CLASSIFIER IMPLEMENTATIONS

4.1 Machine learning models

The original plan for this thesis was to implement a neural network to classify the profile joints. TensorFlow software library was used for the implementation due to previous experience and pre-existing support in Vertex software [32]. For comparison purposes, a Support Vector Classifier (SVC), naive bayes classifier, and a few decision tree-based classifiers were implemented. In the following subchapters, these implementations are presented and discussed.

4.1.1 Traditional classifiers

In this subchapter, the implementations are presented for six different classifiers, including `RandomForestClassifier`, `ExtraTreesClassifier`, `VotingClassifier` (of the previous two), `SVC`, `GaussianNB`, and `XGBoost`. Notably, all the classifiers except `XGBoost` are based on the scikit-learn library. `XGBoost` uses the XGBoost Python library proposed by Chen and Guestrin. [30, 33]

`RandomForestClassifier` was trained using the default hyperparameters. The default parameters include setting the number of decision trees in the forest to 100, the maximum tree depth to unlimited, and using the Gini impurity as the measure of split quality. Furthermore, the default parameters set the minimum number of samples required to split an internal node to two and the minimum number of samples required to be at a leaf node to one. Using these default parameters, the algorithm builds a collection of decision trees, each one based on a random subset of the input features, and then combines their outputs to generate a prediction. `ExtraTreesClassifier` was also trained with its default hyperparameters. It is very similar to the `RandomForestClassifier`, but it has more randomness in the way input feature splits are computed. Scikit-learn claims this usually reduces the variance more, but might introduce a greater increase in bias. Both these classifiers combine the outputs of the single trees by averaging their probabilities. [30]

`VotingClassifier` was used to create a combination of the `RandomForestClassifier` and `ExtraTreesClassifier`. `VotingClassifier` can be used to create an ensemble of virtually any ML classifier. Originally, it was used to create an ensemble model out of all the scikit-learn classifiers presented in this subchapter. That version had lower accuracy than the best models individually and took much longer to train. Combining only the two random forest variants by averaging their predictions slightly outperforms the individual models in accuracy. [30]

`XGBoost` is a decision tree-based ensemble method that uses a different approach for combining models compared to random forest, called gradient boosting. It works by adding decision tree classifiers iteratively to the ensemble until the number of models equal the `n_estimators` parameter. Alternatively, `n_estimators` can be set automatically by instead specifying `early_stopping_rounds`. For each iteration, the predictions of the current ensemble model are used to calculate a loss function and use the results to find good parameters for the next model to be added to the ensemble. The `XGBoost` model was trained with the `early_stopping_rounds` set to 25 and otherwise the default parameters. [33]

Support Vector Classifier (SVC) implementation was done by using the `SVC` algorithm from the scikit-learn's `svm` module. It was trained using the default hyperparameters, notably $C = 1$, and the Radial Basis Function (RBF) kernel type. Also the `probability` parameter was enabled to get the probability estimates for the predictions. [30]

Final classifier was utilizing the probabilistic Naive Bayes algorithm. This was implemented using the scikit-learns `GaussianNB` algorithm from the `naive_bayes` module. From the same module, also the `MultinomialNB`, and the `ComplementNB` were tested using their default parameters. They are more typically used for text classification tasks where the input features are the frequencies of certain words. Those two had lower classification accuracies than `GaussianNB` and were not included in further comparisons. For the `GaussianNB`, the default parameters were also used, which include not specifying the `priors` parameter. Having it unspecified makes the classifier adjust the prior probabilities for each class according to the data. [30]

4.1.2 Neural network

The implemented TensorFlow model is a Multilayer Perceptron (MLP), which is a fully connected neural network similar to the example presented in Chapter 2 Figure 2.6 [25, p. 5]. The final model was created using the Keras Tuner library, and that process is explained later in this subchapter. [39] The input layer of the model consists of 14 neurons, one for each feature in the dataset. The model then uses four hidden dense layers with different neuron sizes (1024, 1024, 256 and 256) to extract and learn relevant features from the input data. Dropout layers between the dense layers are used to avoid overfitting the model to the training data. The dropout probabilities in the model are all 0.3. Figure 4.1 displays the full model architecture with the locations where dropout was applied. The number of trainable parameters in this model is 1 398 292.

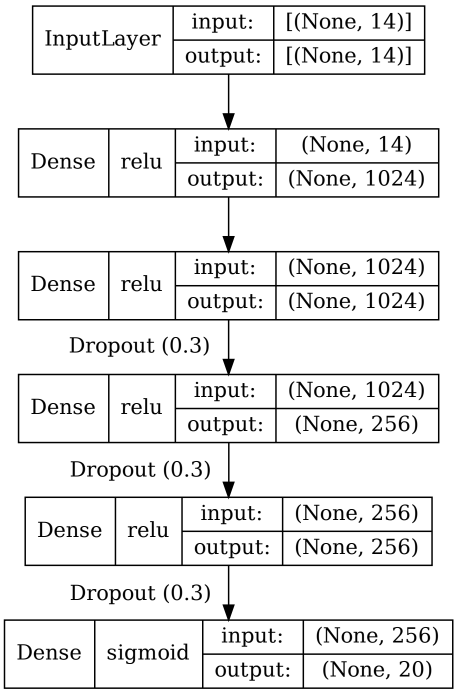


Figure 4.1. *Graph of the implemented neural network*

The output layer of this model consists of 20 neurons, with each neuron representing a different profile joint in the dataset. In some of the earlier iterations of the model, softmax was used as the activation function in the output layer. It was later discovered that the profile joints are not always mutually exclusive, and sometimes can be used interchangeably. To predict multiple possible profile joints, sigmoid was found to be a more appropriate activation function as it produces individual probabilities for each class. By using sigmoid, the software integration can suggest the most likely profile joints based on the input features, even if there is some ambiguity between them. For the same reasons, the loss function `binary_crossentropy` was then used instead of the previous `categorical_crossentropy`.

The model architecture evolved a lot during the development, and the used models were discovered by iteratively adjusting the structure and retraining the model until sufficient performance was reached. In addition to modifying the model, the dataset including the number of features changed too. For better repeatability, another attempt on recreating the model was made. This time the dataset and all factors other than the model itself were kept constant. The model architecture was selected using a manual Neural Architecture Search (NAS), where the number of neurons and the number of layers were adjusted between training runs and the performance was documented for each run. The performance metrics evaluated were the test accuracy and loss. Adam optimizer and binary cross entropy loss with the default hyperparameters were used for the whole search. The search started by training a model with two dense layers. Then the number of neurons in both layers were increased, and the model was trained again. This was repeated until the performance stopped improving. For each training iteration, the model was trained until the validation loss did not decrease further. The validation loss was evaluated after each epoch with separate validation data. Stopping the training was done automatically using the early stopping callback, and the best model weights so far were restored. After reaching the optimal width for the two layers, another dense layer was added and then the numbers of the neurons in each layer were adjusted again. Although this manual search for the optimal model architecture resulted in a model with a decent test accuracy of over 96%, the process was very time-consuming.

Since the manual search required so much active work NAS was also performed using Keras Tuner Bayesian Optimization. Keras Tuner optimizes models automatically within a certain user defined search space. [39] The number of layers was varied between 1 and 5, while the width of each layer was tested with powers of two values ranging from 32 to 1024. Simultaneously, dropout after each layer was being optimized. The dropout probability for each layer was optimized independently, with values ranging from 0.0 to 0.5, in increments of 0.1. Having a dropout probability of 0.0 effectively disables dropout for that layer.

The tuning process was allowed to run for approximately 200 trials. Early stopping was applied with a patience value of 50 and default arguments otherwise. Figure 4.2 displays the effect of the early stopping, where the training is stopped soon after the validation loss stops improving. To reduce the effect of random initialization of the models, each trial was executed 3 times. In addition to Bayesian Optimization, Random Search and Hyperband tuner proposed by Li et al. were also tested, but Bayesian Optimization was found to work the best for this application [40].

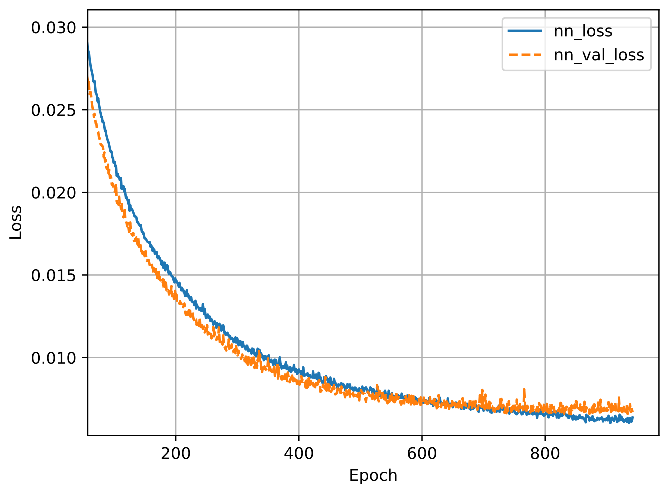
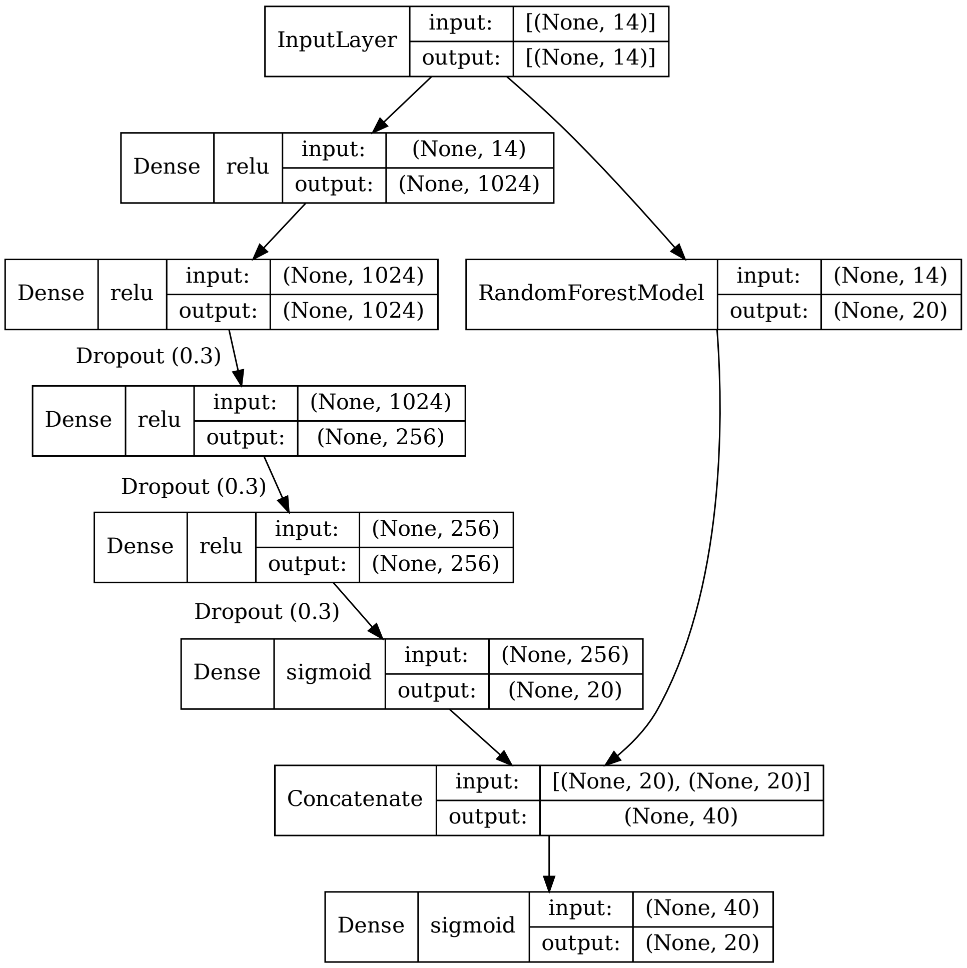


Figure 4.2. *Neural network loss plot*

Overall, the NAS process allowed for the optimization of the neural architecture of the model, resulting in improved performance. The process was computationally intensive and required a significant amount of time. On the other hand, the results should be repeatable and require minimal manual work.

The neural network implementation was also utilized in an ensemble of itself and additionally a random forest classifier implemented using TensorFlow Decision Forests module [32]. They are combined in a metamodel which concatenates the outputs of the individual models. Figure 4.3 displays the resulting classifier.



InputLayer input: [(None, 14)] output: [(None, 14)]
Dense relu input: (None, 14) output: (None, 1024)
Dense relu input: (None, 1024) output: (None, 1024)
RandomForestModel input: (None, 14) output: (None, 20)
Dropout (0.3)
Dense relu input: (None, 1024) output: (None, 256)
Dropout (0.3)
Dense relu input: (None, 256) output: (None, 256)
Dropout (0.3)
Dense sigmoid input: (None, 256) output: (None, 20)
Concatenate input: [(None, 20), (None, 20)] output: (None, 40)
Dense sigmoid input: (None, 40) output: (None, 20)

Figure 4.3. *Ensemble metamodel graph*

In the scope of this thesis, further experiments using this ensemble of a neural network and a random forest classifier were left out. The results were better than either of the classifiers individually.

During the implementation process a notable change to the model architecture was made. Originally the neural network implementation had two inputs. One version of this network is presented in Figure 4.4. In this version the first input includes the cross-section shape of both profiles represented with a fixed size binary image. This is then fed through several convolutional layers. The second input contains all the rest of the features and is essentially identical to the current network's only input. The second input is concatenated with the output of the convolutional layers. The result of the concatenation is fed through a few dense layers and finally the output layer.

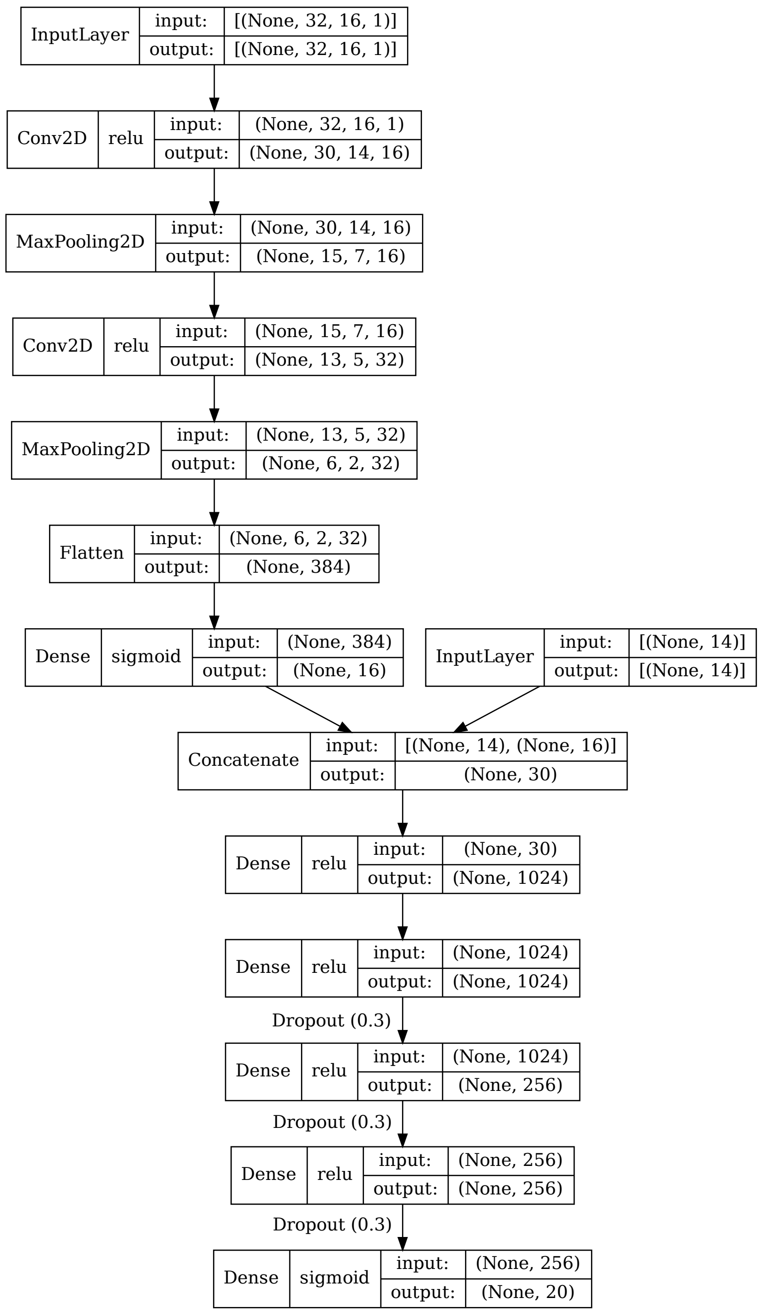

InputLayer | input: [(None, 32, 16, 1)] | output: [(None, 32, 16, 1)]
Conv2D | relu | input: (None, 32, 16, 1) | output: (None, 30, 14, 16)
MaxPooling2D | input: (None, 30, 14, 16) | output: (None, 15, 7, 16)
Conv2D | relu | input: (None, 15, 7, 16) | output: (None, 13, 5, 32)
MaxPooling2D | input: (None, 13, 5, 32) | output: (None, 6, 2, 32)
Flatten | input: (None, 6, 2, 32) | output: (None, 384)
Dense | sigmoid | input: (None, 384) | output: (None, 16)
InputLayer | input: [(None, 14)] | output: [(None, 14)]
Concatenate | input: [(None, 14), (None, 16)] | output: (None, 30)
Dense | relu | input: (None, 30) | output: (None, 1024)
Dense | relu | input: (None, 1024) | output: (None, 1024)
Dropout (0.3)
Dense | relu | input: (None, 1024) | output: (None, 256)
Dropout (0.3)
Dense | relu | input: (None, 256) | output: (None, 256)
Dropout (0.3)
Dense | sigmoid | input: (None, 256) | output: (None, 20)

Figure 4.4. *Neural network with a convolutional branch*

During the development phase, more features related to the cross-section geometry were added. With these features introduced, disabling the first input did not seem to result in any degradation in the model performance. Because of this, the first input was removed from the model and the following experimentation was done without it. It is possible that with a dataset that includes a more comprehensive collection of different types of cross-sections, the results would be different. With the data available at the time of writing, a more thorough testing proved difficult and was discontinued.

4.2 Software integration

A simple experimental user interface for using the implemented TensorFlow model in Vertex BD was constructed. In this initial version, the user can freely choose any two profiles and then use a keyboard shortcut. The program will then extract the features from this pair of profiles and use the implemented model to generate the predictions. These predictions are then used to open a browser similar to the one presented in Figure 2.1 of Subchapter 2.1, but filtered to show only the 5 joints with the highest probabilities. Also, the one with the highest probability is preselected. Figure 4.5 displays an example with selecting two C-profiles.

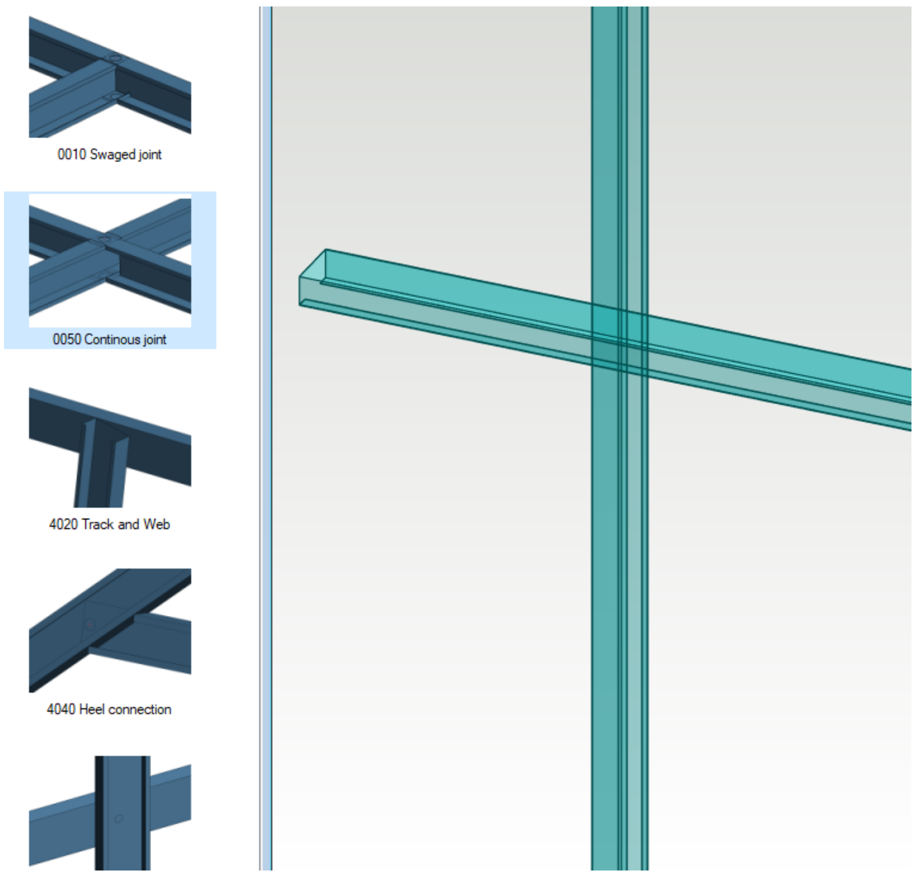


Figure 4.5. *Experimental UI for selecting a profile joint*

As the current ML model does not support the complete library of joints due to the limited training data, the user interface was not yet developed further. There are plans to extend the user interface to allow, for instance, mass operations, but the details are out of the scope of this thesis. Collecting feedback of the predictions is another idea that could be very useful. The feedback could be used to either directly influence the following predictions or at least improve the future iterations of the classifier model. The details of the feedback collection are yet to be decided.

5. RESULTS AND COMPARISONS

5.1 Evaluation metrics

For the final comparisons, the dataset was divided into three splits using scikit-learn library's `train_test_split` function [30]. The training split included 60% of the samples and the validation and test splits each had 20% of the samples.

To examine the predictions a classifier makes, the confusion matrix can be used. It provides a summary of the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class in a table format. True positives are the number of samples that are correctly predicted as belonging to that class, while false positives are the number of samples that are incorrectly predicted as belonging to that class. True negatives are the number of samples that are correctly predicted as not belonging to that class, while false negatives are the number of samples that are incorrectly predicted as not belonging to that class. [13, p. 100]

Using the information from the confusion matrix, several metrics can be calculated to evaluate the performance of the model for each class. These metrics include accuracy, precision, recall and f1-score. [13, p. 101] Table 5.1 presents all the equations for the evaluation metrics.

Accuracy is an intuitive performance measure, representing the proportion of correctly predicted observations relative to the total number of observations. For a balanced dataset, accuracy might be all that's needed to evaluate the model effectively. [13, p. 101]

Precision is the proportion of accurately predicted positive observations compared to the total number of predicted positive observations. Precision addresses the question: "Of all predictions labeled as class X, how many are genuinely class X?" High precision is associated with a low false positive rate. [13, p. 101]

Recall, or sensitivity, is the proportion of accurately predicted positive samples relative to the total samples in the actual class. Recall answers the question: "For all class X samples, how many are predicted as class X?" High recall corresponds to a low false negative rate. [13, p. 101]

Some models display a high precision with low recall, or vice versa. The f1-score is a metric that combines precision and recall into a single value, representing the weighted average of the two. As a result, the f1-score accounts for both false positives and false negatives. While not as intuitive as accuracy, the f1-score is generally more valuable, particularly when dealing with imbalanced class distributions. Accuracy is most suitable for balanced datasets; for imbalanced datasets, it is advisable to consider both precision and recall. [13, p. 102]

Table 5.1. *Equations for the evaluation metrics used [13]*

Metric	**Equation**
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1-score	$2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}}$

In addition to the individual metrics for each class, overall metrics can be calculated for the entire model. These metrics include overall accuracy, and macro-averaged, micro-averaged, and weighted versions of precision, recall and f1-score. Macro-average is simply the mean of the individual metrics, which gives equal weight to each class regardless of the number of samples. Micro-average aggregates the contributions of all classes to compute a single precision, recall, and f1-score value, focusing on the performance of the model on individual instances. Weighted average accounts for possible class imbalance by computing the average of individual metrics in which each class's score is weighted according to the sample distribution. These common metrics are helping to understand how well the model is performing for each class and identify areas for improvement. By analyzing them, it's possible to fine-tune the models to improve their performance and accuracy. [13, 30]

5.2 Performances of the classifiers

This chapter presents the performances of the implemented classifiers. Although the primary interest in the model lies in the top-k accuracy metrics, comparing the results was challenging since many of the models achieved near-perfect top-k accuracy, particularly when k is 3 or larger. Therefore, other metrics such as top-1 accuracy are used as well.

The accuracies and weighted f1-scores of the implemented classifiers are presented in Table 5.2.

Table 5.2. *Overview of the classifier results*

Classifier type	**Top-1 accuracy**	**Weighted f1-score**
XGBoost	0.9853	0.9853
RandomForest+ExtraTrees	0.9836	0.9837
RandomForestClassifier	0.9833	0.9834
ExtraTreesClassifier	0.9815	0.9812
Neural network (MLP)	0.9765	0.9764
SupportVectorClassifier	0.9165	0.9161
GaussianNaiveBayesian	0.7386	0.7034

The neural network did not have the highest top-1 accuracy, but it is easily good enough for its purpose. Figure 5.1 displays the top-k accuracies for each classifier. Even the very simple naive bayesian classifier has over 99% top-4 accuracy, which could be considered usable for the target application.

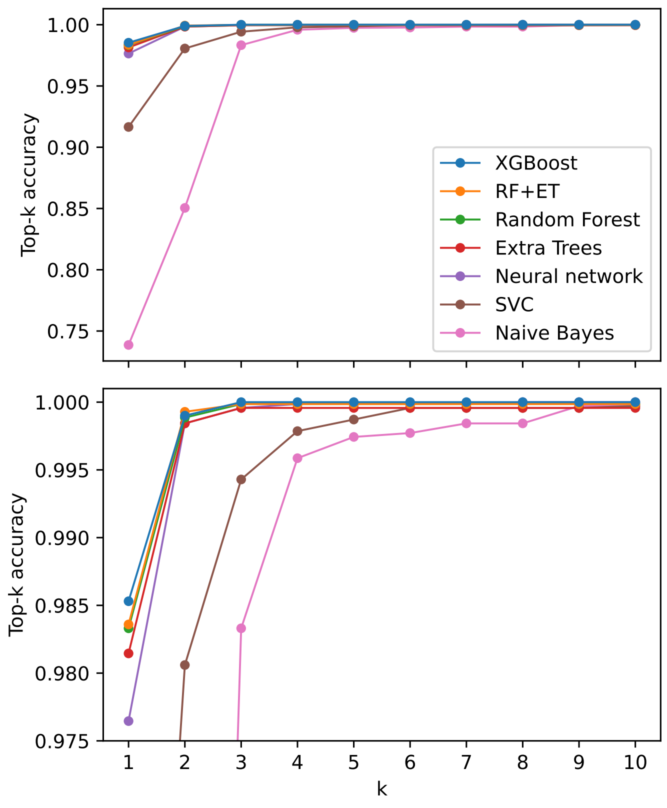


Figure 5.1. *Top-k accuracies of the classifiers*

The results of the final neural network are presented in more detail with the help of the confusion matrix, classification report, the Receiver Operating Characteristic (ROC) curve, and the precision-recall curve. Figure 5.2 displays the confusion matrix, where normalization has been applied over the true classes. It's useful for finding out which classes get mixed up the most. The model relatively often fails with classes 14 and 15, where the class 15 samples are relatively often misclassified as class 14. Those two classes are never mixed up with other classes. As the confusion matrix doesn't reveal the top-k predictions, it was separately examined for these classes. It was observed that for every single failure for classes 14 and 15, the correct class had the second-highest probability, which means they both have a top-2 accuracy of 100%. Class 4, on the other hand, has been incorrectly predicted as classes 1, 7, 8 and 9. When examined, only three predictions where the correct class was neither the first nor the second were found for class 4.

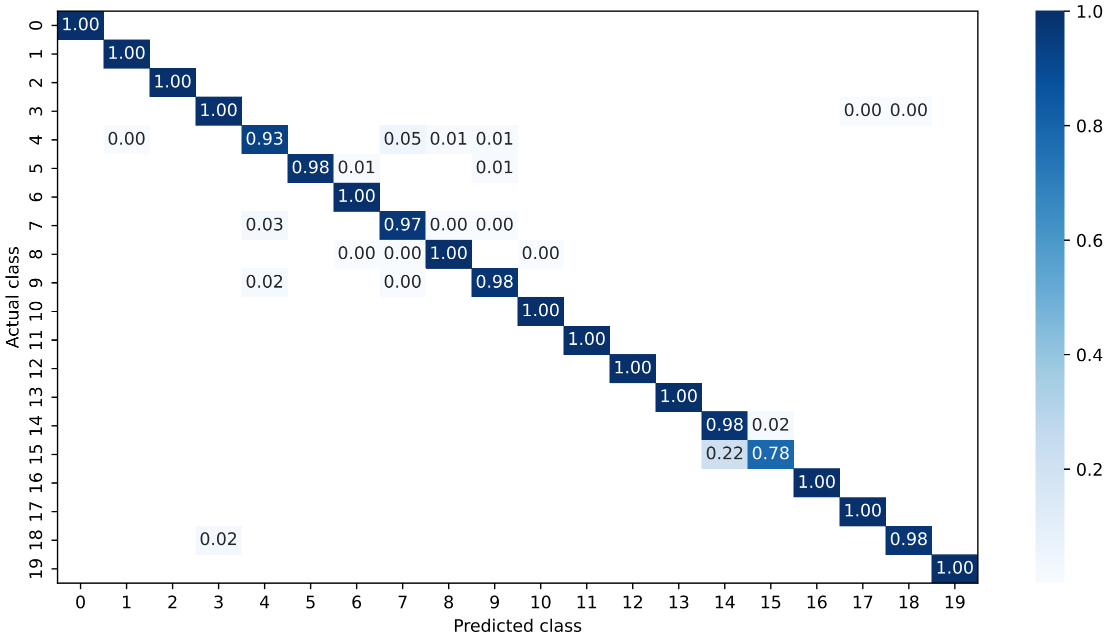
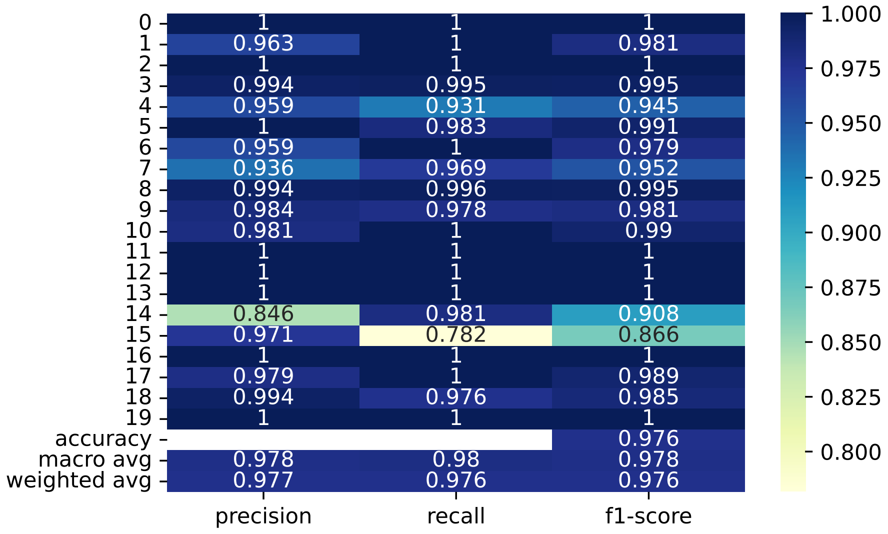


Figure 5.2. *Neural network model confusion matrix*

Figure 5.3 displays the classification report for the neural network. It presents the precision, recall and f1-scores individually for each class. These are then combined to calculate the averages for the same metrics. It also highlights the classes 14 and 15, where class 14's precision and class 15's recall are low compared to the other classes.



	precision	recall	f1-score
0	1	1	1
1	0.963	1	0.981
2	1	1	1
3	0.994	0.995	0.995
4	0.959	0.931	0.945
5	1	0.983	0.991
6	0.959	1	0.979
7	0.936	0.969	0.952
8	0.994	0.996	0.995
9	0.984	0.978	0.981
10	0.981	1	0.99
11	1	1	1
12	1	1	1
13	1	1	1
14	0.846	0.981	0.908
15	0.971	0.782	0.866
16	1	1	1
17	0.979	1	0.989
18	0.994	0.976	0.985
19	1	1	1
accuracy			0.976
macro avg	0.978	0.98	0.978
weighted avg	0.977	0.976	0.976

1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800

Figure 5.3. *Neural network model classification report*

Figure 5.4 displays the neural network's ROC curves comparing each class separately against the rest of the classes. The higher the Area under the ROC Curve (AUC-ROC) is, the better the result. The diagonal line displays AUC-ROC with a value of 0.5, which can be achieved by randomly classifying the samples. In this case, the AUC-ROC is close to 1 for all classes. Only the class 4 curve is slightly separated from the rest of the classes. [41]

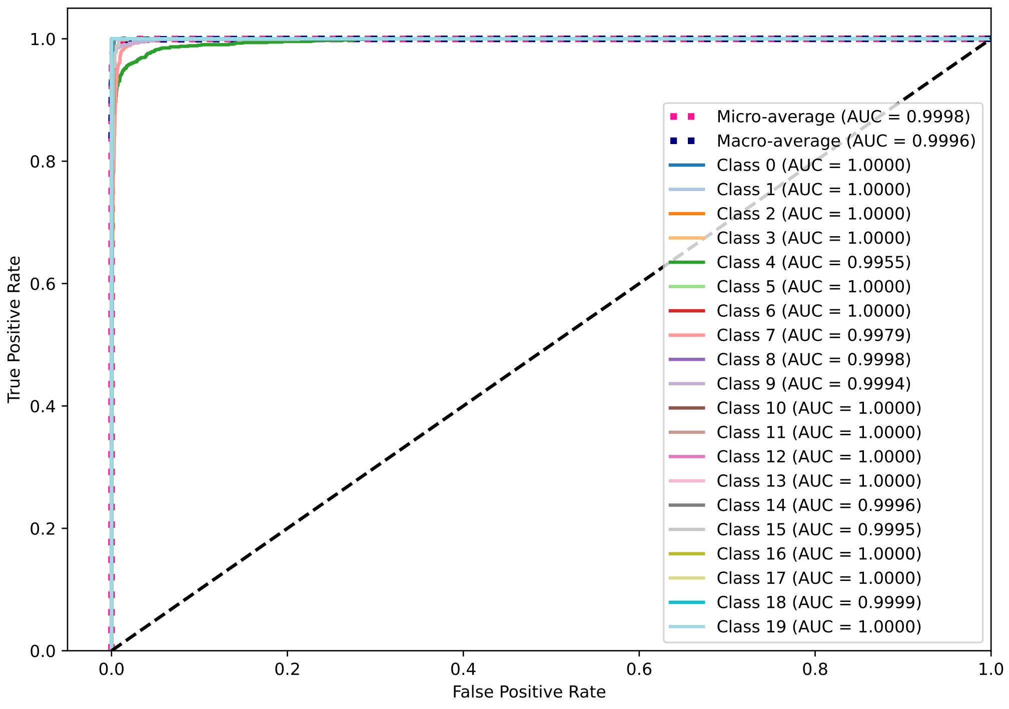


Figure 5.4. *Neural network ROC curve OvR (One-vs-Rest)*

While the ROC results are looking good, the precision-recall curves might convey more information about the classifier performance in this case. Figure 5.5 shows the precision-recall curves for each class against the rest of the classes. The Area under the Precision-Recall Curve (AUC-PR) metric also is better the higher it is. The effects of the failed predictions for the classes 14 and 15 can be observed in the precision-recall curve too. Classes 4 and 7 are the other two classes that the classifier has the most trouble with. [41, p. 105]

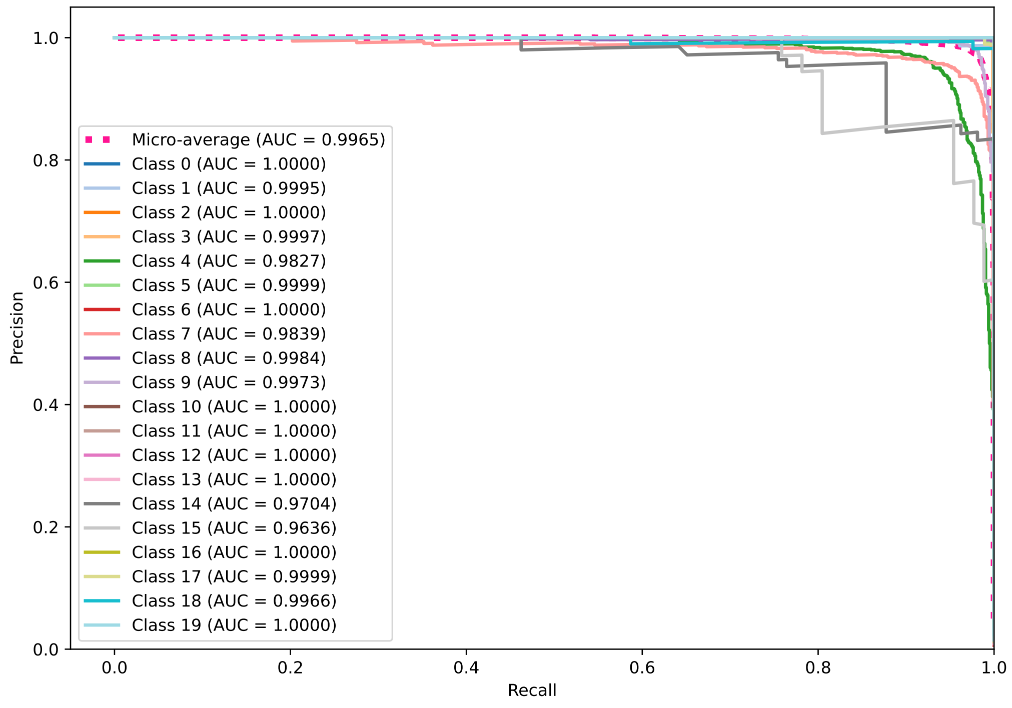


Figure 5.5. *Neural network precision-recall curve OvR (One vs Rest)*

For the other classifiers, the ROC and precision-recall curves are not displayed on a class level, but instead the averages are presented. Figure 5.6 displays the average ROC curves for the implemented classifiers. Only the SVC and naive bayes can be recognized to slightly fall short of the rest of the classifiers, which are all grouped having nearly perfect average AUC-ROC scores.

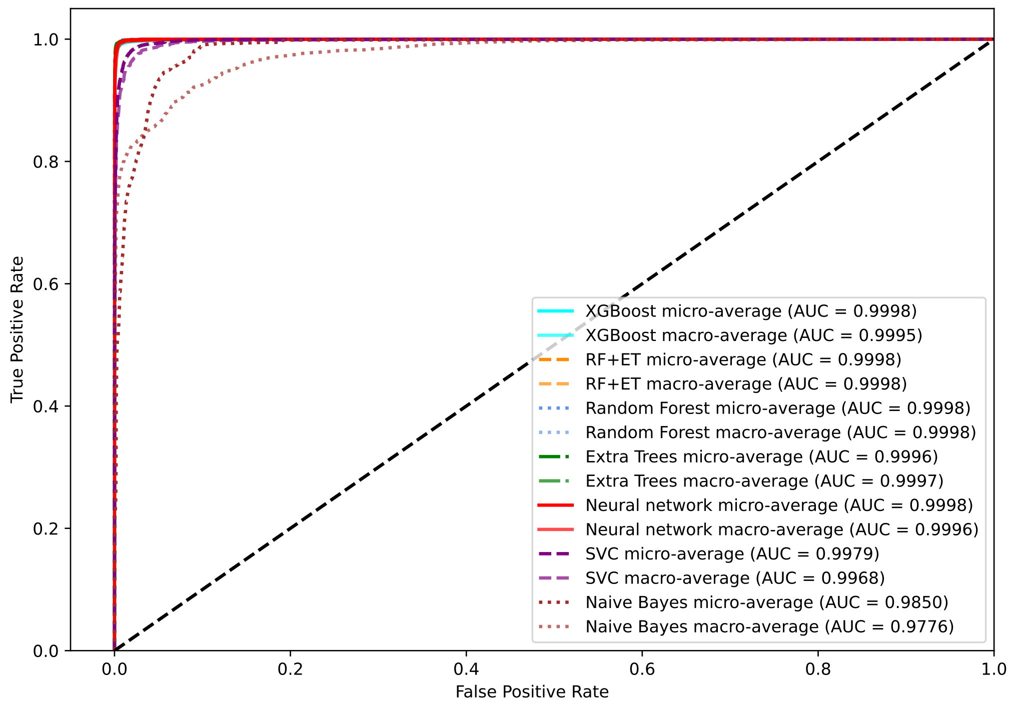


Figure 5.6. *Average ROC curves for all implemented classifiers*

Figure 5.7 displays the micro-averaged precision-recall curves for the classifiers. Here the SVC and naive bayes are even more clearly distinguished as performing worse than the other classifiers.

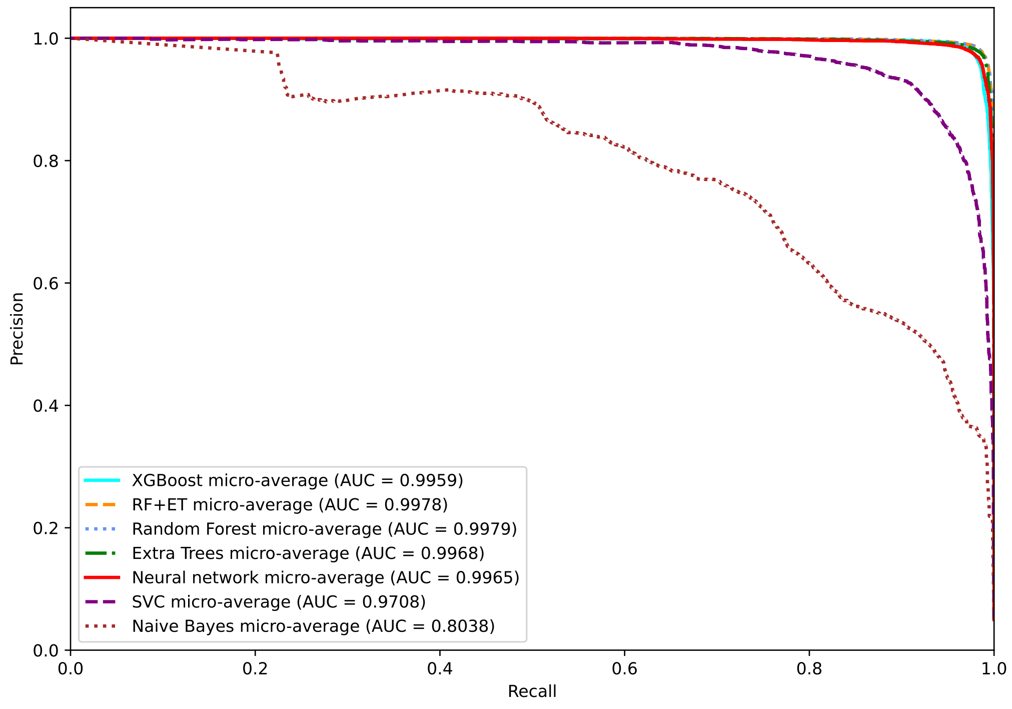


Figure 5.7. *Average precision-recall curves for all implemented classifiers*

To help interpret the predictions the implemented model makes, individual SHAP plots can be used. These plots show the contribution of each feature to the final prediction, providing insight into how the model arrived at its decision. By explaining the model's behavior in this way, SHAP plots can increase users' confidence in the model and serve as a tool for debugging possible issues. To give an example on the SHAP values, two plots for two separate predictions by the neural network are displayed. In a multi class classifier, the SHAP values can be calculated for all the possible classes for each prediction. The classification task in hand usually has 1 to 5 classes that can be considered correct. The training data only has one label for each sample, but as it was previously determined, there is some acceptable overlap among the classes. In the following examples, two different classes are examined. The features displayed in red color will push the prediction towards the currently examined class, while the blue features will do the opposite, pushing the prediction away from it.

Figure 5.8 is used to explain a successful prediction. This is done on a class 9 sample, and it was correctly predicted. The prediction is very confident giving almost 100% probability for the correct class. The force plot on the top displays the SHAP values for the correct class, which in this case also sum up to nearly 100%. The force plot on the bottom display the SHAP values for class 14, which has a near zero probability. This is reflected similarly on the SHAP values for almost all the features.

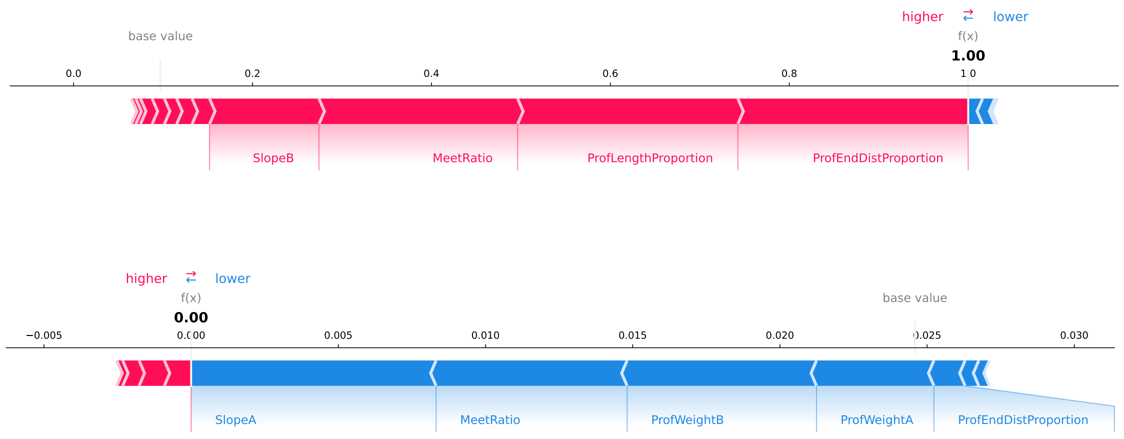
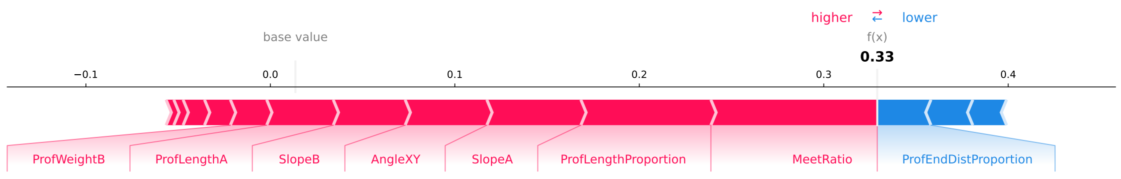


Figure 5.8. *Force plots for a successful prediction*

Figure 5.9 gives an example of a failed prediction. This is applied on a class 4 sample, which was incorrectly predicted as class 1. The prediction gave class 1 a probability of 33.10%. The correct class, 4, only has a probability of 6.24%. Class 4 had the fourth-highest probability in this prediction. The force plot on the top displays the predicted class, and the SHAP values for almost all features are contributing towards the incorrect prediction. The force plot on the bottom displays what the correct class would've been, and the SHAP values, while not exactly opposite of the other one, still mostly are having a negative contribution.



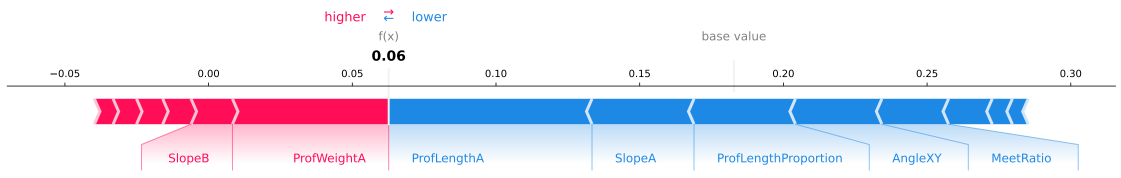


Figure 5.9. *Force plots for a failed prediction*

It should be noted that the SHAP values are calculated using k-means summarized training data. This means that the training data is first clustered using the k-means algorithm, and each cluster's centroid is used as a representative point for the entire cluster. This step reduces the calculations needed to compute the SHAP values, as the number of representative points is much smaller than the original number of training samples. However, this approach may introduce some inaccuracies in the resulting SHAP values, as the representative points may not fully capture the distribution of the original training data. Therefore, the accuracy of the SHAP values calculated using this method may not be as high as those calculated using the entire training data. Nevertheless, as the sums of the SHAP values in the presented examples are almost identical to the actual prediction probabilities, the results could be considered good enough for this purpose.

5.3 Discussion

The classifiers showed varying levels of performance in terms of top-1 accuracy, weighted f1-score, and top-k accuracy. XGBoost achieved the highest top-1 accuracy, closely followed by the other decision tree-based classifiers. The success of XGBoost was not surprising as it has been a popular choice in the Kaggle competitions, where it has currently been utilized in multiple winning solutions [13, 33, 42]. The neural network performed almost as well, while the SVC and naive bayes classifiers demonstrated lower accuracy compared to the other models.

The dataset featured in the comparisons includes only 20 different profile joints, where the total number is nearly 100 in Vertex BD. To create a classifier that could reliably work with all the profile joints, much more training data is required. The difficulties to suddenly gather enough data underscores the importance of comprehensive data collection for the development of effective machine learning models, as it directly impacts their accuracy, generalizability, and overall usefulness. With the increased amount of data and different classes, the classifier performances are likely to get worse unless more informative features are added, because the current feature space would get more crowded. Since the performances for the neural network and the decision tree-based classifiers were so high, especially when looking at the top-k accuracies, the support for more classes shouldn't cause too many problems.

Expanding the dataset likely increases the number of different types of cross-sections, which could consequently increase the useful information the binary images of the cross-sections provide. Utilizing the convolutional branch displayed in Figure 4.4 could give an edge to the neural network classifier over the decision forest-based classifiers.

A potential source for overly optimistic evaluation results was identified. The classifiers were assessed using test data separated from the full dataset, which only included samples where a profile joint had already been applied, though this was not disclosed to the classifier. In Vertex BD, applying a profile joint can lead to virtually any type of modification in the geometry of the connected profiles. In practice, the more common modifications that might affect the features utilized in the dataset are related to stretching the profile ends if necessary.

Further testing was conducted using the experimental user interface introduced in 4.2 with new data. It was observed that certain types of profile joints had a noticeable impact on the top-1 predictions even when the profiles were stretched by less than 10 mm. However, the top-5 predictions appeared to be less affected by these changes. It is important to note that these experiments were carried out with a limited sample size of fewer than 50 samples.

Using the AI-assisted experimental UI as a comparison to the existing manual method of choosing joints is not straightforward because the manual method depends heavily on the users. Those with a lot of experience working with profile joints have likely set the joints they require as favorites, and are able to select them very quickly and consistently. In this case, the AI-based method is unlikely to improve efficiency significantly. On the other hand, less experienced users may need more time to find the joints they require when searching manually. This is where the filtering and preselecting capabilities of the AI-assisted method can save a lot of time.

6. CONCLUSION

The goal of this work was to provide an AI-based solution for helping the users in selecting profile joints in a BIM software, Vertex BD. Due to limited training data, this goal couldn't be completely fulfilled. Instead, multiple methods to classify a limited number of different profile joints were created and compared with each other. A proof-of-concept method, which utilized a neural network, was also implemented and tested in Vertex BD. The neural network was created using Keras Tuner. To increase the likelihood that the proof-of-concept model could be extended to support all profile joints in Vertex BD when enough data was collected, the model was carefully analyzed. This includes comparing multiple methods to reduce the risk of overly optimistic performance metrics through data leakage. Also, multiple evaluation methods to analyze the predictions of the trained models were utilized. These include the confusion matrix, ROC curves, precision-recall curves, and top-k accuracy evaluation. Finally, a method to inspect how a single prediction is formed was created using the SHAP values. This displays how each feature contributes to the prediction and can be used to better understand the model and possibly assist in debugging.

The most important future work is to collect more data and repeat the experiments with a larger dataset. While the dataset size increases, more informative features might be needed and investigating what they are and how they can be collected is another potential future work. A more comprehensive literature review could be utilized to find ideas for the features and also consulting experts in the field could prove useful. The current pool of available features was created based on what could be extracted from the software with a limited amount of work. An experimental dataset including these features was created from 80 different professionally designed building models. Multiple feature evaluation methods were applied to find the most informative features from the data that was available. Using the information these methods revealed, the number of informative features was narrowed down to 14. Currently, for each profile joint in the project two samples are created, because the selection order of the two profiles is not linked to anything. It could be investigated how the performance is affected if the order of the two profiles was determined based on the values of the features, having each sample in the dataset just once. This halves the size of the dataset, which is great as long as the performance does not suffer, but without careful testing, it's difficult to say what happens. Also, the effect of stretching the profile ends as discussed in Subchapter 5.3 should be investigated more thoroughly.

Throughout this research, six different main types of ML classifiers were compared in order to determine the most suitable models for the application. While these classifiers provided some insights into the potential of using AI-based techniques in this context, there is still room for further exploration of alternative algorithms. As potential future work, it could be useful to investigate the performance of other machine learning models, such as more complex ensemble methods, deep learning architectures, or even custom-built algorithms tailored specifically to the problem at hand. Moreover, additional feature engineering, data augmentation, and preprocessing techniques could be explored to enhance the performance and generalizability of the models. This continuous search for improved approaches will not only contribute to a better understanding of the problem domain but also pave the way for the development of more advanced and effective AI-assisted tools in the field of building design.

REFERENCES

[1] Zabin, A., González, V. A., Zou, Y. and Amor, R. Applications of Machine Learning to BIM: A Systematic Literature Review. *Adv. Eng. Inform.* 51.C (Mar. 2023). ISSN: 1474-0346. DOI: 10.1016/j.aei.2021.101474.

[2] OpenAI. *ChatGPT*. URL: https://openai.com/blog/chatgpt (visited on 03/10/2023).

[3] Puuinfo Oy. *Teollisen puurakentamisen opetusmateriaali* (2023). URL: https://urn.fi/urn:nbn:fi:oerfi-202301_00026078_0 (visited on 02/28/2023).

[4] *Manual for the Design of Building Structures to Eurocode 1 and Basis of Structural Design*. Institution of Structural Engineers (ISTRUCTE), 2010. ISBN: 978-1-906335-07-6. URL: https://app.knovel.com/hotlink/toc/id:kpMDBSEBS1/manual-design-building/manual-design-building.

[5] Porteous, J. *Designers' guide to eurocode 5 : design of timber buildings : EN 1995-1-1*. eng. Designers' guides to the eurocodes. London: ICE Publishing, 2013. ISBN: 1-62870-390-3.

[6] Guo, T., Li, L., Cai, L. and Zhao, Y. Alternative method for identification of the dynamic properties of bolted joints. eng. *Journal of mechanical science and technology* 26.10 (2012), pp. 3017–3027. ISSN: 1738-494X.

[7] Zhu, B. *The finite element method : fundamentals and applications in civil, hydraulic, mechanical and aeronautical engineering*. eng. Hoboken, New Jersey: Wiley, 2018. ISBN: 1-119-10734-2.

[8] *Vertex BD - Automated BIM Software for Wood and Steel Framing*. 2023. URL: https://vertexcad.com/bd/ (visited on 04/14/2023).

[9] *Vertex BD Documentation*. 2023. URL: https://kb.vertex.fi/bd2023en (visited on 04/14/2023).

[10] Baduge, S. K., Thilakarathna, S., Perera, J. S., Arashpour, M., Sharafi, P., Teodosio, B., Shringi, A. and Mendis, P. Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications. *Automation in construction* 141 (2022), p. 104440. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2022.104440.

[11] Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. eng. New York, NY: Springer New York, 2017. ISBN: 0387848576.

[12] Vasilev, I. *Python deep learning : exploring deep learning techniques and neural network architectures with pytorch, keras, and tensorflow*. eng. Second edition. Birmingham: Packt Publishing Ltd., 2019. ISBN: 1-78934-970-2.

[13] Mirtaheri, S. L. *Machine learning theory to applications*. eng. First edition. Boca Raton: CRC Press, 2022. ISBN: 9780367634537.

[14] Khallaf, R. and Khallaf, M. Classification and analysis of deep learning applications in construction: A systematic literature review. eng. *Automation in construction* 129 (2021), pp. 103760–. ISSN: 0926-5805.

[15] Karypidis, E., Mouslech, S. G., Skoulariki, K. and Gazis, A. Comparison Analysis of Traditional Machine Learning and Deep Learning Techniques for Data and Image Classification. *WSEAS TRANSACTIONS ON MATHEMATICS* 21 (Mar. 2022), pp. 122–130. DOI: 10.37394/23206.2022.21.19.

[16] Patterson, J. *Deep learning : a practitioner's approach*. eng. First edition. Beijing, China: O'Reilly Media, 2017. ISBN: 1-4919-2457-8.

[17] Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*. eng. O'Reilly Media, Inc, 2022. ISBN: 9781098125967.

[18] Ganaie, M., Hu, M., Malik, A., Tanveer, M. and Suganthan, P. Ensemble deep learning: A review. eng. *Engineering applications of artificial intelligence* 115 (2022), pp. 105151–. ISSN: 0952-1976.

[19] Ali, J., Khan, R., Ahmad, N. and Maqsood, I. Random Forests and Decision Trees. eng. *International journal of computer science issues* 9.5 (2012), pp. 272–272. ISSN: 1694-0814.

[20] Breiman, L. Random forests. eng. *Machine learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125.

[21] Ho, T. K. Random decision forests. eng. *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. IEEE, 1995, 278–282 vol.1. ISBN: 0818671289.

[22] Steinwart, I. *Support Vector Machines*. eng. 1st ed. 2008. Information Science and Statistics. New York, NY: Springer New York, 2008. ISBN: 1-281-92704-X.

[23] Ding, X., Liu, J., Yang, F. and Cao, J. Random radial basis function kernel-based support vector machine. eng. *Journal of the Franklin Institute* 358.18 (2021), pp. 10121–10140. ISSN: 0016-0032.

[24] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. eng. *arXiv.org* (2017). ISSN: 2331-8422.

[25] Atienza, R. *Advanced Deep Learning with Keras*. eng. Packt Publishing, 2018. ISBN: 9781788629416.

[26] Misra, D. Mish: A Self Regularized Non-Monotonic Activation Function. eng. *arXiv.org* (2020). ISSN: 2331-8422.

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[28] Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. SMOTE: Synthetic minority over-sampling technique. eng. *The Journal of artificial intelligence research* 16 (2011), pp. 321–357. ISSN: 1076-9757.

[29] Lundberg, S. M. and Lee, S.-I. A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774. URL: `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[30] 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. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] Ross, B. C. Mutual information between discrete and continuous data sets. eng. *PloS one* 9.2 (2014), e87357–e87357. ISSN: 1932-6203.

[32] 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. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[33] Chen, T. and Guestrin, C. XGBoost: A Scalable Tree Boosting System. eng. *Proceedings of the 22nd ACM SIGKDD International Conference on knowledge discovery and data mining*. KDD '16. Software available from github.com/dmlc/xgboost. Ithaca: ACM, 2016, pp. 785–794. ISBN: 1450342329.

[34] McInnes, L., Healy, J. and Melville, J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. eng. *arXiv.org* (2020). ISSN: 2331-8422.

[35] Maaten, L. van der and Hinton, G. Visualizing High-Dimensional Data Using t-SNE. eng. *Journal of machine learning research* 9.nov (2008), pp. 2579–2605. ISSN: 1532-4435.

[36] Kobak, D. and Berens, P. The art of using t-SNE for single-cell transcriptomics. eng. *Nature communications* 10.1 (2019), pp. 5416–14. ISSN: 2041-1723.

[37] Lemaître, G., Nogueira, F. and Aridas, C. K. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine*

Learning Research 18.17 (2017), pp. 1–5. URL: `http://jmlr.org/papers/v18/16-365.html`.

[38] Last, F., Douzas, G. and Bacao, F. Oversampling for Imbalanced Learning Based on K-Means and SMOTE. eng. *arXiv.org* (2017). ISSN: 2331-8422.

[39] O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L. et al. *Keras-Tuner*. `https://github.com/keras-team/keras-tuner`. 2019.

[40] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. and Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52. URL: `http://jmlr.org/papers/v18/16-558.html`.

[41] Pietikäinen, M. and Silvén, O. *Tekoälyn haasteet : koneoppimisesta ja konenäöstä tunnetekoälyyn*. fi. Päivitetty toinen painos. Konenäön ja signaalianalyysin keskus, 2021. ISBN: 978-952-62-3202-7. URL: `http://urn.fi/urn:isbn:9789526232027`.

[42] Kaggle. *Competitions*. Online. URL: `https://www.kaggle.com/competitions` (visited on 04/25/2023).