Using Ensemble Learning and Deep Learning to Classify Welding Defects Produced During Remote Laser Welding

Aaron Chellam

Supervised by

Prof. Darek Ceglarek

Department of Computer Science

University of Warwick

Year of Study: 3rd

Abstract

Remote laser welding is a manufacturing technique used to weld thin metal foils, required for the construction of battery packs in electric vehicles. During the welding process, it is possible for defects to form due to a lack of connection between metal foils or because of overpenetration of the laser beam. Photodiode-based sensors can be used to extract real-time signals, during the welding process, which machine learning algorithms can use to perform automatic weld defect classification. At present, classical machine learning algorithms have been unable to reliably classify defects produced during the welding of thin metal foils due to a lack of available training data and their inability to classify time-series photodiode signals. This project explores the use of ensemble learning and deep learning techniques with the aim of improving upon current best-in-class accuracies. The results indicated that soft-voting ensemble learning classifiers achieved higher classification accuracies than their best-in-class constituent classical classifiers and that deep learning models achieved higher classification accuracies when applied to generalised datasets composed from several different data distributions.

Keywords: Machine Learning, Remote Laser Welding, Ensemble Learning, Deep Learning, Classification, Zero Defect Manufacturing.

Acknowledgements

I would first like to thank my supervisor, Professor Darek Ceglarek, for his support and advice throughout the project and for providing me with the opportunity to work on a practical project in a novel field. I would also like to acknowledge Giovanni Chianese for performing the welding experiments, producing the datasets and performing the machine learning benchmark tests upon which the project is based. Lastly, I would like to thank Dan Dai for her constant support and useful insights throughout the project.

Contents

Chapter 1

Introduction

One of the primary problems that can arise in industrial processes is the appearance of defects in the final product or in product components. Such defects naturally introduce delays into the production chain and also increase production costs. As a result, Zero Defect Manufacturing (ZDM)[1] has emerged as a quality management paradigm that aims to completely eliminate defective products via defect prediction and prevention. The panoptic vision of ZDM is for every product leaving the production line to meet or exceed the desired quality, avoiding any costs associated with defect reparation and waste materials.

One key tenet of ZDM is autonomous closed-loop in-process (CLIP) quality control;[2] CLIP quality control is a system in which a manufacturing process is continuously monitored in real-time using sensors. In such a system, real-time data extracted from these sensors can be analysed by supervised machine learning models to predict whether or not the current product being manufactured will be defective once the process terminates. If the model determines that a product will contain a defect, root cause analysis[1] can be performed to identify the source of the error and changes can autonomously be made to the manufacturing parameters to prevent the defect from materialising. Under perfect circumstances, this closed-loop system succinctly captures the essence of ZDM: materials are not wasted on repairing defective products and negligible time is wasted by human operators on determining which process parameters need to be modified to prevent the defect.

However, a key component of autonomous CLIP quality control is the defect prediction step. For the system to be able to make real-time alterations to the manufacturing process, the machine learning model must be able to accurately and reliably use the sensor data to differentiate between defective and non-defective products before root cause analysis can be performed. This project in particular explores the use of supervised machine learning to classify welding defects produced via the remote laser welding (RLW) of copper-to-steel battery tab connectors, which are used in the construction of battery packs for electric vehicles.

Chapter 2

Background

2.1 Electric Vehicle Battery Packs

The transition away from fossil fuel-powered transport has influenced many auto manufacturers to produce a significantly higher share of electric vehicles (EVs)[3]. Consequently, large-scale EV battery manufacturing is quickly emerging as an industry priority.

Battery packs are typically manufactured an designed in a ‘pack-module-cell’ structure. That is, multiple cells are clustered into a module and multiple modules are connected within a single battery pack, where the number of cells is proportional to the battery pack’s power and capacity.[4]

As a point of reference, the assembly of a Tesla Model S battery pack requires more than 10,000 welds at both the cell and module level,[5] with a total weld seam length of over 30m.[6] It has been estimated that approximately 6% of cells and modules produced by UK Gigafactories are defective[7] and while it remains possible to repair defective welds, the re-working of such defects also carries the risk of introducing novel defects into the final product. Additionally, repairing defects would require auxiliary specialist equipment, further driving up production costs. Moreover, since individual battery components are too expensive to scrap and since only 70-80% of battery cells are recyclable, manufacturers have set strict quality targets with weld reliability above 99.7%.[5] These factors collectively motivate the need for ZDM principles, such as autonomous CLIP quality control, to be applied to the remote laser welding manufacturing process.

2.2 Remote Laser Welding

RLW is a technique in which a high-precision laser beam is used to supply the heat required to melt and fuse two materials together to create a weld joint. Since the laser is typically emitted from a source mounted on the arm of an industrial robot, RLW operations can be performed at a significant distance from the product workpiece.[8] Hence, a major advantage of RLW is that it is a single-sided joining technology. That is, welds can be created even if only one side of the workpiece is visible, as opposed to traditional welding techniques in which materials being welded would need to be mechanically clamped on both sides.

A second major benefit of remote laser technology is that no physical contact is required to fuse the two materials.[9] This means that welds can be performed in tight corners and complex structures in which there is only a narrow, straight-line window of access to the welding target. Additionally, RLW is significantly faster than traditional welding techniques as the robot arm enables swift repositioning of the laser and a scanning mirror, placed at the head of the robot, can be used to redirect the laser beam such that it strikes the desired target without the need to

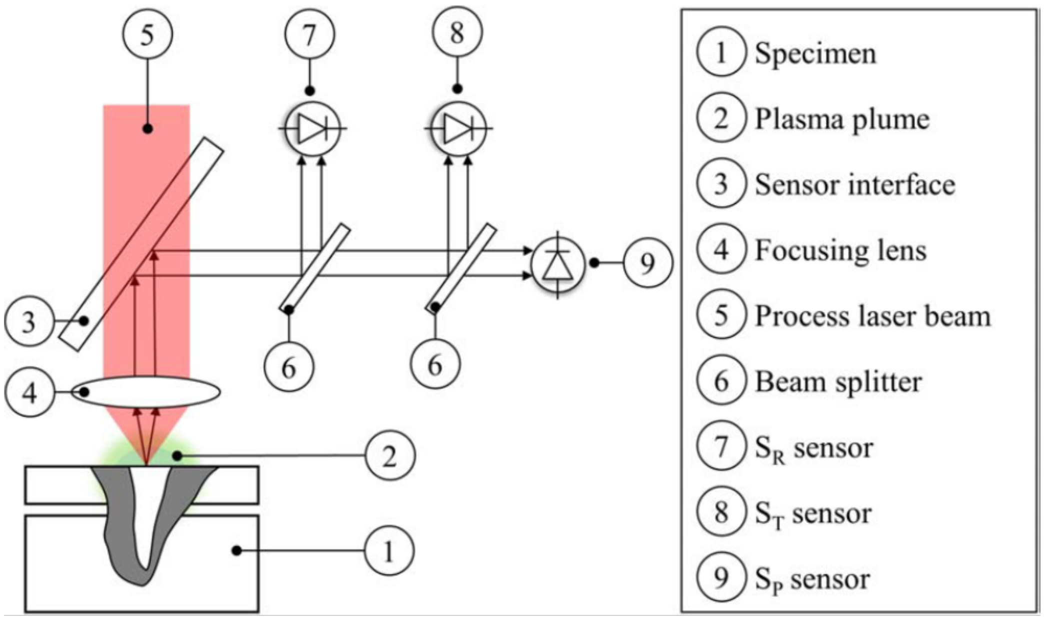


Figure 2.1. Basic setup of the RLW process with a photodiode sensor used for in-process monitoring.[5]

reposition the product.

Due to these benefits, RLW has been identified as a key technology[8] in EV manufacturing and, by extension, the wider ZDM ideology. This naturally calls for innovative solutions to improve RLW quality control, using an autonomous CLIP approach.

During the welding of battery tab connectors, it is essential that the welds maintain, with high repeatability, sufficient mechanical integrity and electrical conductivity, while also meeting specific thermal requirements.[5] For example, connections that result in unequal electrical resistances within the same battery pack can reduce the overall performance of the pack and cause different cells within the same pack to degrade at different rates.

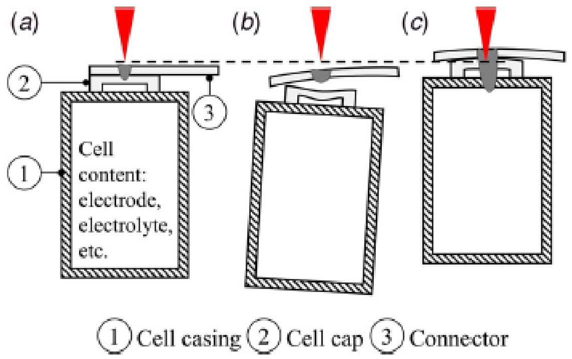


Figure 2.2. Standard design of a battery cell and tab connector; (a) ideal welding condition, (b) lack of connection (c) over-penetration.[5]

Notably, battery cell manufacturing requires the welding of thin, dissimilar metallic foils. This introduces three major challenges to the welding process:

1. **Challenge 1: Dissimilar metals**. Dissimilar metals have different thermal and mechanical properties. As a result, welds produced may have poorer joint strength in comparison to homogeneous welds, which further mandates the need for improved quality control mechanisms.

2. **Challenge 2: Lack of Connection**. When the metal foils are clamped in preparation for welding, gaps can form between the two materials. Since the metallic foils are relatively thin ($300\mu m$), part-to-part gaps can lead to a lack of joint connection, which is associated with drops in electrical connection and hence electrical conductivity.

3. **Challenge 3: Over-penetration**. During the welding process, it is possible for the laser to penetrate both of the thin metal foils, damaging the cell as well as risking fires or explosions due to leakages of harmful gases.

Challenge 1 can be resolved by implementing an established technology known as laser beam wobbling,[10] and hence it is not discussed in detail in this report. Challenges 2 and 3, however, are interdependent with each other[5] and are motivated by the fact that variations in weld penetration depth and part-to-part gap can have detrimental effects on the structural and electrical integrity of the final welds. This can be observed in Figure 2.2 which shows the typical design of a cylindrical battery cell and tab connector, as well as the two major types of defects discussed in this project. In particular 2.2(a) corresponds to a desirable weld, 2.2(b) corresponds to a defective weld in which the two metal foils are not sufficiently connected, and 2.2(c) corresponds to a defective weld in which the laser beam has penetrated through both metal foils.

2.2.1 Autonomous CLIP Quality Control in RLW

In the introduction, a general overview of autonomous closed-loop in-process quality control, in manufacturing systems, was provided. In the context of RLW, CLIP monitoring involves using sensors to collect real-time data features of the welding process. For example, surface features[11] (such as melt pool width and material concavity), as well as sub-surface features (such as weld penetration depth and weld connection), can be analysed during the welding process itself. These features from the welding process are then correlated with weld classes (defective or non-defective) via machine learning techniques. If a machine learning model predicts that a weld will be defective, changes can be made to the welding parameters; for example, the laser power or the laser focal point can be adjusted.

Within the scope of RLW for battery tab connectors, Chianese et al.[5] demonstrated that photodiode-based sensors can be used to extract key welding features which can be used to characterise variations in weld penetration and part-to-part gaps during the welding of thin metal foils. This is a crucial step in a wider autonomous CLIP system, as real-time data features relating to weld penetration and part-to-part gaps are essential for machine learning models to determine whether a weld will correspond to one of the two major types of defects: overpenetration and lack of connection.

2.2.2 Welding Features and Classes

A photodiode-based sensor can collect three types of signals from the radiation emitted during the welding process: S_P, S_T, and S_R[5]. The S_P signal corresponds to the radiation emitted from the metal vapour and plasma plume, the S_T signal corresponds to the thermal radiation of the processed zone, and the S_R signal corresponds to the reflected laser light. From these radiation signals, two key statistical features can be extracted:

- Energy intensity, μ, which is proportional to the total energy content of the emitted radiation.

- Scatter level, σ, which is proportional to the uncontrolled process variations and is measured using the standard deviation of the signal noise content.

Since the energy intensity and scatter level can be measured for all three signal types, each dataset contains a total of six features: $\{\mu_P, \sigma_P, \mu_T, \sigma_T, \mu_R, \sigma_R\}$.

As discussed, a machine learning model can use these six welding features to predict whether or not the current weld will be defective by classifying it into one of three classes[5]:

- Class (1) – Sound Weld (SW)

- Class (2) – Lack of Connection (LoC)
- Class (3) – Over Penetration (OP)

Here, Class 2 and Class 3 correspond to the two major types of defective welds; if the model predicts the current weld to be in either of these classes, then manufacturing process parameters can be adjusted to prevent the defect from occurring. In theory, after performing root cause analysis, these adjustments can be performed automatically, without human intervention, giving rise to the term autonomous closed-loop quality control.

The primary objective of this project is to construct supervised machine learning models that can use the welding features to predict the corresponding welding classes, with the aim of improving upon the accuracy of current, best-in-class models. Section 3 contains further discussion on the performance of these best-in-class classifiers as well as a more detailed review of the datasets used in the project.

2.3 Machine Learning

This section offers a summary of the machine learning concepts that appear frequently, throughout the project.

2.3.1 Supervised Machine Learning and Classification

As discussed, the project is primarily a supervised learning, classification task. In general, supervised learning is a machine learning paradigm under which algorithms attempt to learn the relationship between a set of inputs and their corresponding outputs, provided by an external human operator[12]. The inputs correspond to training data that the algorithm is trying to learn from, and the outputs correspond to the "correct" labels associated with each data point. In a classification task, the target labels are known, discrete classes.

The overall principle is that an algorithm can be trained to learn the mapping relationship between the training data points and their classes to produce a machine learning model. This model can then be used to predict the class of an entirely new, unseen data point from the same distribution as the training data.

In this project, a data point primarily corresponds to a six-dimensional feature vector, $\{\mu_P, \sigma_P, \mu_T, \sigma_T, \mu_R, \sigma_R\}$, and the target labels correspond to the three welding classes, as defined in section 2.2.2. To test a model's performance on a dataset, the dataset is first split into training and testing sets; the model is trained on the data in the training set and then used to predict the class of every data point in the testing set. Since the testing data came from the same original dataset as the training data, the class labels of the test data points are known in advance; these class values are defined as the ground truth labels. The ground truth labels can be compared with the predicted classes to compute a classification accuracy, which is simply the proportion of test data points that were correctly classified.

2.3.2 Overfitting, Underfitting and Generalisation

Overfitting occurs when a machine learning model performs well on the training data but struggles to generalise to new, unseen data.[13] That is, it accurately learns the relationship between the features and labels in the training data but struggles to reliably classify data points that it was not trained on. This can occur if a model not only learns the mapping between features and labels but also learns information about the underlying noise in the training. Additionally, overfitting can occur when data sets are too small; smaller training datasets can introduce sampling noise if the datasets themselves do not represent the general data distribution.

Overfitting occurs when the model is too complex relative to the noise of the training data. Solutions to overfitting include: simplifying the model by reducing the number of parameters, reducing the noise in the training data (by pruning outliers) or gathering more training data.

Underfitting, on the other hand, occurs when the model is too simple learn the underlying pattern of the training data. Naturally, a model that underfits the training data is also unlikely to be able to fully fit the testing data. Underfitting can be tackled by increasing the complexity of the model, by increasing the number of parameters or relaxing model constraints, or by providing improved features to the learning algorithm that better capture the full variance of the dataset.

2.3.3 Hyperparameters

A hyperparameter[14] is a parameter of a machine learning algorithm rather than of a machine learning model. That is, it must be manually set before training and cannot be modified by the learning algorithm itself. Hyperparameters govern many aspects of an algorithm's training and tuning their values can significantly improve a model's ability to generalise to new data.

Given that hyperparameters must be set manually, it can be challenging to find an optimal value. One option is to create multiple different models, each of which differ only by the value of a given hyperparameter. These models can then be compared by training them all on a common training set and evaluating their accuracy on a common testing set. However, this approach can lead to poor generalisation to new data that is not in the test set. The reason for this is that, by using the testing set to optimise a hyperparameter, the final model produced may have learned information about the underlying noise of that testing set. This is analogous to overfitting, described in section 2.3.2, however in this case the model overfits the testing data rather than the training data.

A typical solution to this problem involves the use of an additional set: a validation set. When using a validation set, a dataset is split into training and testing sets as before, but the training set is further subdivided into a smaller training set and a validation set. That is, the validation set acts as training data that is 'held out' of the training process. Models are then trained on the reduced training data, and hyperparameters are optimised using the validation data instead of the testing data. Finally, the testing set can be used to measure the model's accuracy and hence its ability to generalise as no aspect of the model was influenced by data points in the test set.

However, a major disadvantage of using a validation set is that it diminishes the size of the training set, thus increasing the risk of overfitting in smaller datasets. This problem is particularly relevant to the project and is discussed further in section 3.3.

2.3.4 Multi-Class Classification

As discussed, the project revolves around constructing machine learning classifiers which can classify welding data points into one of three classes. However, while some machine learning algorithms are capable of performing multi-class classification natively, others are strictly designed to perform binary classification. In spite of this, there are two predominant strategies that can be adopted to convert multiple binary classifiers into a single multi-class classifier[15]: one-versus-rest (OvR) and one-versus-one (OvO).

One-versus-rest, also referred to as one-versus-all, is an approach in which a binary classifier is trained for each of the classes. For example, in this project, a binary classifier will be trained to classify data points as either 'Sound Weld' (Class 1) data points or 'Not Sound Weld' data points. This is repeated for all three classes such that each class has a specialist binary classifier trying to distinguish its data points from the remaining data points not in that class. Under OvR, each binary classifier produces a decision score using a decision function that is algorithm-dependent; the final class predicted will be that which corresponds to the binary classifier that produces the highest decision score. At a glance, the decision score is a measure of a classifier's confidence that a point belongs to a specific class.

Alternatively, in the one-versus-one approach, a binary classifier is created for every combination of class pairs; in this project, there are three classes and hence only three possible pairs that can be chosen. Each binary classifier produces a predicted class, and the class with the most predictions is returned; ties can be broken by comparing decision scores as with the OvR method.

2.4 Notations and Terminology

2.4.1 Machine Learning Terminology

- **Bias and Variance**. Bias and variance[16] are machine learning concepts that influence a model's generalisation error. Bias describes the errors in a model that are caused by wrong assumptions, such as assuming that data is linear when it is actually quadratic. A model with high bias is likely to underfit the training data. On the other hand, variance describes generalisation errors caused by a model's excessive sensitivity to noise in the training data. Models with high complexity are likely to have high variance and hence overfit the training data. Reducing a model's complexity typically increases its bias and reduces its variance and vice versa, creating a natural trade-off between bias and variance.

- **Classical Machine Learning**. In this project, unless stated otherwise, classical machine learning models refer to any models that are not deep learning or voting ensemble models.

2.4.2 Machine Learning Notation

The following is a list of common machine learning notations used to discuss the mathematical theory behind machine learning models. These definitions can be universally assumed from the notation unless stated otherwise:

- **X** defines a dataset's feature matrix. The feature matrix is a (n$\times$m) matrix where n denotes the number of data instances in the dataset and m denotes the number of features contained by each data instance. Notably, when considering a single feature vector contained within the feature matrix, d may be used to denote the dimensionality of the feature vector instead of m.
- **y** defines a dataset's label vector. The label vector contains the target classes for each data instance.
- $\mathbf{X}^{(i)}$ corresponds to the i^{th} instance in dataset **X**. It is feature vector containing

all of the features values for data point i; the class label of data point i is denoted by $y^{(i)}$.

Chapter 3

Existing Literature

As discussed in section 2.2, Chianese et al.[5] used photodiode-based sensors to extract welding features $\{\mu_P, \ \sigma_P, \ \mu_T, \ \sigma_T, \ \mu_R, \ \sigma_R\}$ during the RLW of thin metal foils. Subsequent analysis suggested that these features capture significant information about the variations in weld quality, which presents opportunities for automatic weld defect classification via machine learning. In further experiments performed by Chianese et al.[17], seven supervised machine learning algorithms were used to study whether the feature data produced by the photodiodes could be used to effectively classify welds into one of the three classes defined in section 2.2.2: SW, LoC and OP. This section elucidates the structure of the datasets produced during the RLW experiments, the initial results produced by the seven machine learning models, as well as the primary motivations for this project.

3.1 Welding Configuration and Datasets

In total, three different RLW experiments were performed using three different experimental setups, each producing a distinct dataset. The general configuration for the remote laser welding process can be summarised as follows: thin copper foils of length 70mm and width 30mm were clamped above steel foils of equivalent length and width. The lower steel foils had a thickness of 300μm, while the upper copper foils had a thickness of either 200μm or 300μm depending on the experiment. For a given welding process, the laser moves linearly, delivering a constant laser power, with the direction of the laser beam perpendicular to the upper copper layer; a total welding length of 30 mm is completed for each experiment.[17]

Several process parameters were varied across the three experiments in order to simulate the welding conditions required to generate welds of each class. These parameters include the power of the laser (P_L), the part-to-part gap between the upper and lower layers, the thickness of the metal foils and the materials used to coat the metal foils. During the experiments, the part-to-part gap was controlled by placing shim packs between the upper and lower layers. Figure 3.1 shows a visual representation of the experimental setup.

Table 3.1 shows the process configurations of the three, original datasets: A, B and C. Notably, dataset B contains just 14 examples and hence was considered too small to be used in the machine learning experiments; the dataset was primarily constructed to simulate variations in part-to-part gap with a fixed laser power rather than for classification. Instead, a new data set, $A \cup B \cup C$ was created containing all examples from datasets A,B and C. Notably, there are four data points that occur in both datasets A and B, meaning dataset $A \cup B \cup C$ contains four fewer examples than the sum of examples across datasets A, B and C. The benefit of this

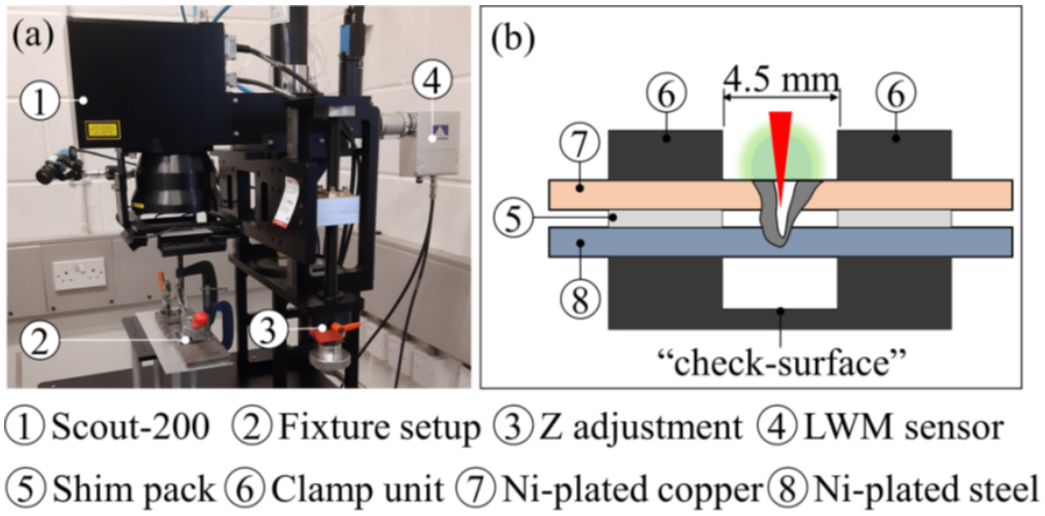

(a)
(b)
4.5 mm
"check-surface"
① Scout-200 ② Fixture setup ③ Z adjustment ④ LWM sensor
⑤ Shim pack ⑥ Clamp unit ⑦ Ni-plated copper ⑧ Ni-plated steel

Figure 3.1. (a) Photograph of the RLW apparatus and (b) 2D representation of the clamping mechanism.[17]

quasi-novel dataset is that it can be used to measure model generalisation since it consists of data examples drawn from similar data distributions, with slight variations in RLW process configurations.

Table 3.2a shows the class breakdowns for datasets A, B, and $A \cup B \cup C$. At a glance, dataset C contains a noticeable class imbalance with SW data points

Table 3.1. Structure of the three RLW datasets.

	Dataset		
	A	B	C
Laser Power (W)	600-1500	1050	390-990
Part-to-part Gap (μm)	0	0-300	0-200
Number of Data Points	46	14	86
Upper Foil	Uncoated copper, 300 μm		Nickel-plated copper, 200μm
Lower Foil	Nickel-plated steel, 300μm		

significantly underrepresented relative to LoC and OP data points; this is because dataset C contains more results from experiments performed at higher part-to-part gaps (up to 300μm) to model RLW instabilities at higher laser powers.[17]

As a result, before performing classification, dataset C was augmented by producing synthetic SW data points. Specifically, for every pair of feature vectors corresponding to SW data points, a synthetic feature vector was produced by taking the average feature values across all six features. Given that dataset C had 9 SW data points by default, this data augmentation step introduces $\binom{9}{2}$ = 36 synthetic data points, giving a total of 9 + 36 = 45 SW data examples in the final augmented dataset C. The class breakdowns after accounting for augmentation are shown in Table 3.2b. Note that the generalised dataset $A \cup B \cup C$ only considers the pre-augmented data from dataset C.

Table 3.2. Class breakdown of the classification datasets. (a) Pre-augmentation; (b) Augmented dataset C.

	A	**C**	**A ∪ B ∪ C**
Sound Weld	16	9	33
Lack of Connection	12	55	73
Overpenetration	18	22	40
Total Datapoints	46	86	142

(a)

	A	**C [Aug]**	**A ∪ B ∪ C**
Sound Weld	16	45	33
Lack of Connection	12	55	73
Overpenetration	18	22	40
Total Datapoints	46	122	142

(b)

3.2 Machine Learning Benchmark Study

As discussed, seven machine learning algorithms were benchmarked: K-Nearest Neighbours (KNN), Decision Tree, Random Forest, Naive Bayes, Support Vector

Machine (SVM), Quadratic Discriminant Analysis (QDA), and Single-Layer Neural Network. The mechanisms behind the first six of these algorithms are discussed in detail in section 4. The neural network classifier, however, requires additional explanation since it operates on time-series feature data, rather than the statistical feature data defined in section 2.2.2.

3.2.1 Neural Networks and Time-Series Data

The seventh algorithm requires further discussion since the neural network model was trained using alternative features to the remaining six models, which were trained using the statistical features, $\{\mu_P, \sigma_P, \mu_T, \sigma_T, \mu_R, \sigma_R\}$. It should be noted that these features are defined as statistical because they represent a statistical summary of the photodiode signals captured throughout the duration of the welding process.

However, since RLW is a process that takes place over time, the signals detected vary over time; by computing a statistical summary of the signal data, time-dependent information about the welding process is naturally lost. To address this, a technique known as DWT was used to represent the three photodiode signals (corresponding to temperature, plasma and back reflection) across 2500 time steps; that is, for a given data point, the time-series features are represented by 2D matrices with shape (2500 $\times 3$). Notably, while the time-series features have a different geometric representation to the statistical features, they were still produced from the same welding experiments as the statistical features. That is, for every welding experiment, there is a statistical and time-series representation for the feature data, meaning that techniques such as data augmentation remain viable for both representations.

The use of time-series data motivated the use of neural networks because they are capable of processing high-dimensional feature data, a property which is beyond the native capabilities of the remaining six benchmark models. Further discussion on the operation of neural networks is included in section 4.3.

3.2.2 Benchmark Classification Results

Table 3.3 displays the benchmark classification results. All algorithms were evaluated using the leave-one-out (LOOCV) cross-validation technique in which a single sample is held out from the dataset while the remaining examples are used as training data. Once trained, models attempt to classify the testing sample; this process is then repeated such that every data point is used as test data exactly once. Model accuracy is then computed as the percentage of samples that were correctly classified out of the whole dataset.

At a glance, it is immediately clear that models performed worst on dataset A with no model producing a higher accuracy for dataset A than their corresponding accuracies for datasets C and the generalised dataset. This can be attributed to the fact that dataset A contains the fewest samples but also because analysis of the feature data suggests that there is a relatively high overlap in the feature space across classes.[17] This is because experiments conducted for dataset A involved gradually varying the weld penetration depth (while keeping the part-to-part gap constant), resulting in a smooth transition in the signal features from lack of connection to overpenetration.

On the other hand, models unanimously performed best on dataset C with a minimum classification accuracy of 94.5%. Two contributing factors to this could be that dataset C contains more samples than dataset A and all examples were drawn

Table 3.3. Machine learning benchmark classification results.

Algorithm	**A**	**C [AUG]**	**A $\cup$ B $\cup$ C**
KNN	82.6	95.7	88.5
Decision Tree	71.7	95.1	81.6
Random Forest	**87.0**	94.5	88.0
Naive Bayes	65.2	96.9	73.8
SVM	54.3	96.3	87.4
QDA	71.7	96.3	84.4
Neural Network	84.8	**97.5**	**92.7**

from the same data distribution, unlike the generalised data set. However, it is also important to note that dataset C contains synthetic sound weld samples, produced by the pairwise average of 9 authentic signals. This form of data augmentation naturally introduces bias into the evaluation process since, when evaluating model performance on a synthetic signal, the model may already have learned something about the underlying structure of the synthetic signal, based on the authentic 'parent' signals used to generate it. This is especially the case under LOOCV in which both raw parent signals used to generate the synthetic signal will appear in the training set. Additionally, both raw signals were paired with other raw signals to produce several other synthetic signals, each time contributing half of the information used to make up these artificial signals. This could significantly increase the risk of information leakage, given that the majority of SW data points are synthetic in the augmented dataset C.

While it is challenging to estimate the effect to which augmentation bias influences the high accuracy results observed in dataset C, as opposed to the higher sample size and homogeneous distribution, one potential indicator is that those models that perform comparatively poorly on datasets A and $A \cup B \cup C$ perform on par with the best classifiers on dataset C. For example, the decision tree and naive Bayes models perform relatively poorly against the best classifiers on dataset A and the generalised dataset but no equivalent difference is observed for dataset C. An alternate hypothesis could be that these two models are particularly sensitive to small or generalised RLW datasets, meaning further examination of the class imbalance problem is required to draw strong conclusions.

A final observation of note from the results in Table 3.3 is that the neural network classifier produces the best performance on dataset C and the generalised dataset while also achieving the second-highest accuracy on dataset A, suggesting that the time-series RLW features carry significant classification potential that is not necessarily present in the statistical RLW features.

3.3 Project Motivation

The initial benchmark experiments present three outstanding challenges directly pertaining to machine learning classification and by extension zero-defect manufacturing in the RLW process:

1. **The size of RLW datasets.** Many machine learning models rely upon hundreds of thousands of samples to produce accurate predictions[18]; having access to just over a hundred data points places a severe restriction on the

training, validation and particularly the optimisation of machine learning models. However, this is a natural consequence of RLW as welding datasets are expensive to compile, since materials must be consumed to artificially create defective products, and also time-consuming as domain experts are required to label sample classes manually.

2. **The dataset classes are imbalanced.** Some classes are significantly under-represented in the datasets; this exacerbates the problem caused by the small dataset sizes since smaller training sets are less likely represent the minority class's data points.

3. **RLW data is time-series.** Signals produced by RLW photodiode sensors are time-series by nature but almost all benchmarked models are unable to operate on such high-dimensional data. While the single-layer perceptron model can classify time-series features, they are restricted in classification power by a lack of model complexity.

Challenge 1 offers the greatest level of difficulty because the datasets are fundamentally too small to make effective use of validation sets, which are typically required to optimise a machine learning algorithm's hyperparameters[19] and by extension a machine learning model's performance. The absence of validation sets motivated the first major contribution of the project: ensemble learning. In the context of classification, ensemble learning is a machine learning paradigm that involves aggregating several different machine learning classifiers into a single ensemble classifier. The key idea behind an ensemble classifier is that diverse classification algorithms have the potential to make diverse mistakes; by aggregating the results of multiple constituent classifiers, it may be possible to circumnavigate classifications produced by some models if a majority of the models can isolate the correct class. This links directly to the challenge of the absent validation set; instead of trying to perfectly optimise a single model, the objective is to minimise the errors made by individual models by adding extra layers of redundancy.

Challenge 2 is tightly linked with challenge 1 with smaller training sets aggravating the effect of the imbalanced classes. Consequently, ensemble learning offers a potential solution by distributing model sensitivity to underrepresented classes across a larger ensemble of classifiers. However, additional solutions to the imbalanced class problem can be implemented via specific model validation strategies, which are further explained in section 4.4.

Finally, Challenge 3 naturally motivates the second major contribution of the project which is deep learning. Deep learning models are intuitive successors to the single-layer perceptron model implemented in the existing research; they offer the same capability of processing high-dimensional data as well as added classification power, driven by flexible architectures and increased complexity.

Section 4.1 offers an overview of the classical, stand-alone machine learning algorithms considered in this project, section 4.2 provides an overview of various ensemble techniques used to aggregate classical models and 4.3 expands upon the primary deep learning architectures used to process the high-dimensional, time-series welding features.

Chapter 4

Methodology

4.1 Classical Machine Learning Models

This section provides a detailed overview of the classical machine learning algorithms considered in this project. Notably, the primary classical models considered were the decision tree, k-nearest neighbours and support vector machine models with the remaining models only evaluated as an extension. In particular, the Gaussian naive Bayes and quadratic discriminant analysis models were added to enable a full comparison with their counterparts in the benchmark tests. The Gaussian process model was added as a novel classifier to provide additional combinations of constituent models for the ensemble classifiers discussed in section 4.2.

4.1.1 Decision Tree

A decision tree model aims to predict the label of a new data point by applying simple decision rules, learned from the data feature during the training phase.[20]

Decision trees consist primarily of two elements: nodes and branches. During the training phase, at each node, a feature is selected from the data set for evaluation. During this evaluation process, the model splits the data points based on a binary decision rule . For example, a decision rule may separate data points whose first feature value is greater than a given threshold and data points whose first feature value is not greater than that threshold. The model is trained by recursively applying several splitting rules at every node, each time selecting the best rule found, based on a metric that measures the quality of a split, known as gini impurity.[21]

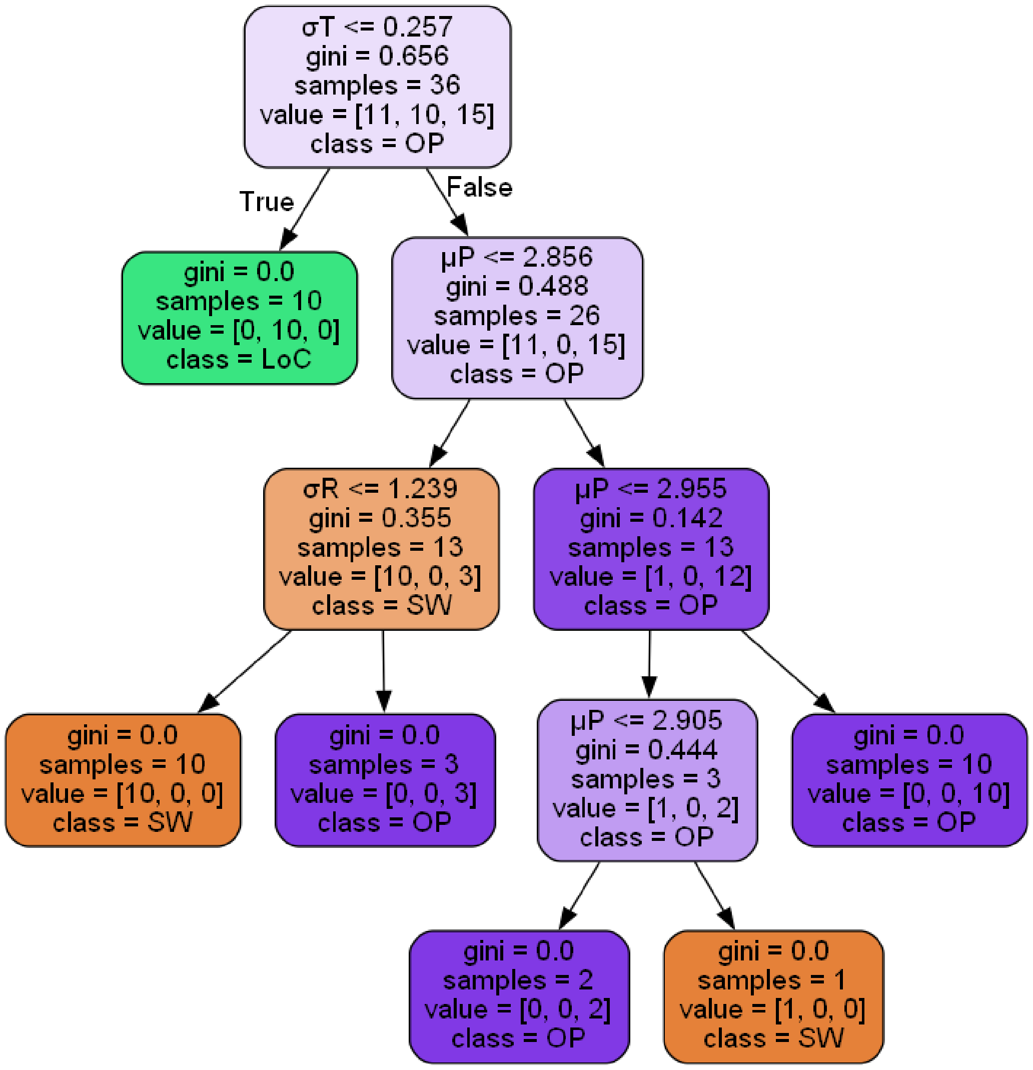


Figure 4.1. Visualisation of a decision tree model.

Figure 4.1 displays a simple visualisation of a decision tree model, trained on a subset of dataset 1. Here, the root node of the tree begins by splitting on the σ_T feature; the subset of data points with a σ_T value ≤ 0.257 are separated into the left child node and the remaining data points are moved to the right child node.

Notably, the root node has a gini impurity value of 0.656, which is a measure of the homogeneity of the data classes in the node. Equation 4.1 shows the equation used to compute gini impurity, where p_i denotes the probability of an element in the subset belonging to class i. That is, gini impurity is a value that ranges from 0 to 2/3 where an impurity of 0 corresponds to a subset of data points with entirely homogeneous class labels and an impurity of 2/3 corresponds to a subset with class labels that are uniformly distributed across all three classes.

This can be observed in Figure 4.1 by inspecting the "value" parameters of the root node and its children. The root node has a value parameter of [11, 10, 15] corresponding to 11 SW, 10 LoC and 15 OP data points respectively. The left child node however contains 10 LoC data points and zero examples from the other two classes; accordingly, this node has a gini impurity of 0, marking the node as entirely homogeneous. Once a node becomes homogeneous, the decision tree model ceases to perform any further splitting rules, creating a leaf node. The right child node, while relatively impure with 11 SW points and 15 OP points, has a lower gini impurity than the root node, due to the separation of the 10 LoC points into the left child node.

$$\text{Gini Impurity} = 1 - \sum_i (p_i)^2 \tag{4.1}$$

When making class predictions on a new, unseen data point, the decision tree model will move the point starting from the root node to a child node depending on whether or not the new data point satisfies the splitting condition. This process is repeated until the point reaches a leaf node, at which point the model will produce a class prediction based on the majority class of the data points in the leaf node.

Crucially, the decision tree model is also capable of producing predicted probabilities, where the probabilities predicted for each class correspond to the proportions of each class in the leaf node. In theory, it is possible for the decision tree depth to be restricted during the training phase, resulting in tree leaves that are not entirely homogeneous. In practice, since the datasets used in this project a relatively small, the decision tree models always produce homogeneous leaves. This property has a significant influence on the soft voting models, which aggregate models based on predicted class probabilities, described in [SECTION] because the decision tree model always predicts classes with a probability of 1.

CART Training Algorithm

In this project, decision tree models use the CART training algorithm[22] to find suitable splitting rules that separate the weld classes. Initially, a root node is created containing all of the data points in the training set. The algorithm then seeks to find a single feature k and a threshold t_k such that the splitting rule formed by the pair (k, t_k) produces the purest subsets, weighted by size.

$$J(k, t_k) = \frac{n_{left}}{n} G_{left} + \frac{n_{right}}{n} G_{right}$$
$$\text{where} \begin{cases} n_{left/right} & \text{denotes the number of instances in the left/right subset,} \\ G_{left/right} & \text{denotes the gini impurity of the left/right subset.} \end{cases} \tag{4.2}$$

Equation 4.2 demonstrates the cost function, J(k, t_k) used by the CART algorithm to find the best splitting rule; the resulting splitting rule is then used to create the subsets for the left and right child nodes. The algorithm then recurses on these child nodes until either the maximum allowable tree depth is reached or the subsets are entirely homogeneous.

4.1.2 K-Nearest Neighbours

K-Nearest Neighbours (KNN) classification is a form of non-generalising learning; it does not construct a general internal model but simply stores instances of the training data. The KNN algorithm[23] classifies a new data point based upon the data point's k nearest neighbours in the training data. The new point is assigned a class based upon a plurality vote of its nearest neighbours' classes, with the Euclidean distance metric used to calculate the closest points. In this project, the algorithm considers the 3 Nearest Neighbours, as K=3 has been observed, from previous literature[5] and experimental data, to produce the most effective model for the RLW data.

4.1.3 Support Vector Machine

The goal of a support vector machine (SVM) is to find a set of hyperplanes within the feature space that can segregate data points from different classes. In the context of machine learning, a hyperplane is a subspace with one less dimension than the total feature space.[24] For example, given a set of data with two features, the feature space would be two-dimensional and a hyperplane would be a one-dimensional straight line. In this project, the RLW datasets contain six primary features, and hence the feature space is six-dimensional, and a linear support vector machine would aim to find a set of five-dimensional hyperplanes to separate the three different welding classes.

However, initial experiments with a linear SVM model produced a poor classification accuracy, relative to the Decision Tree and Nearest Neighbour models. As a result, the linear SVM model was substituted for a non-linear SVM model, yielding improvements in classification accuracy across all three datasets. The non-linear SVM model assumes that the input data is not linearly separable - that is, there is no set of hyperplanes that can reliably separate the classes in the default feature space. Instead, non-linear SVMs utilise a technique known as the kernel trick[?] to implicitly map the data points into a higher-dimensional feature space. Subsequently, the model seeks to find a set of hyperplanes to separate the data points in this new, higher-dimensional space. These higher-dimensional hyperplanes then correspond to non-linear decision boundaries[?] in the original feature space.

Overview of Linear SVMs

Before outlining the detailed operation of non-linear SVMs, it is useful to discuss the training and classification mechanisms of linear SVM models.[21] Figure 4.2 shows the hyperplane (represented by a solid black line) found by a linear SVM classifier, trained on 2-dimensional data with binary classes.

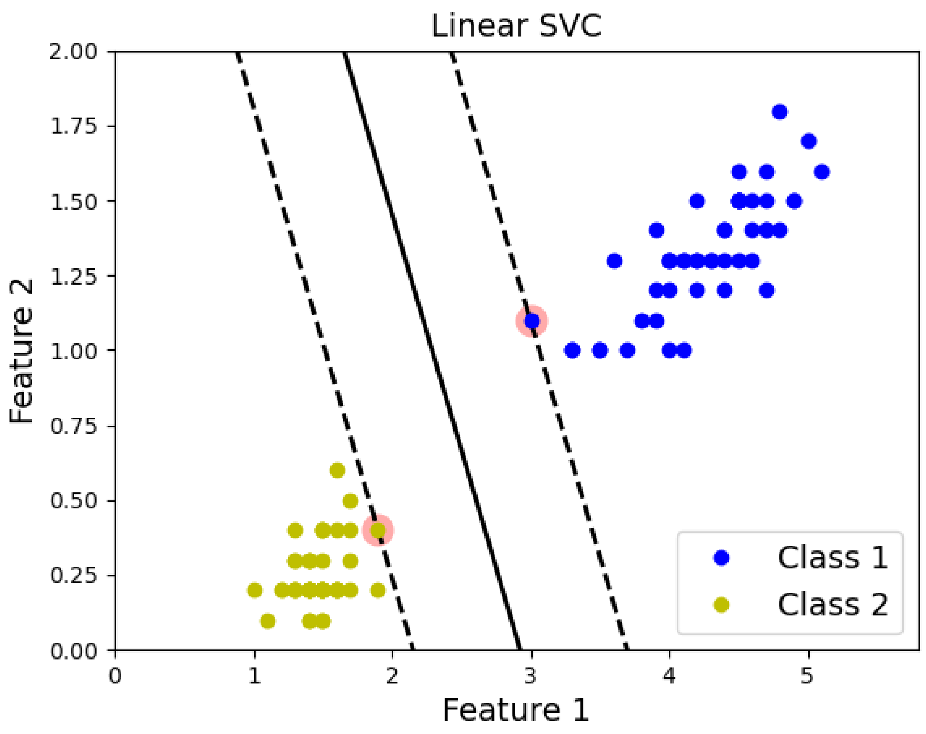


Figure 4.2. An SVM hyperplane.[21]

Intuitively, there are multiple hyperplanes that are capable of separating the two classes. Hence, the goal of the SVM model is to find an optimal hyperplane that maximises the distance between the hyperplane and the closest data points from each class; this distance is referred to as the margin, and the closest data points are known as the support vectors.

In an ideal case, the data set would be linearly separable and the SVM model would be able to find a hyperplane with a wide margin, such that data points on each side of the hyperplane are homogeneous. In reality, outliers can significantly skew the position of the hyperplane, as can be seen from Figure 4.3. On the left of the figure, a single outlier prevents the dataset from being linearly separable and hence no hyperplane can be found. On the right of the figure, the dataset remains linearly separable but the outlier significantly shifts the decision boundary away from its natural position.

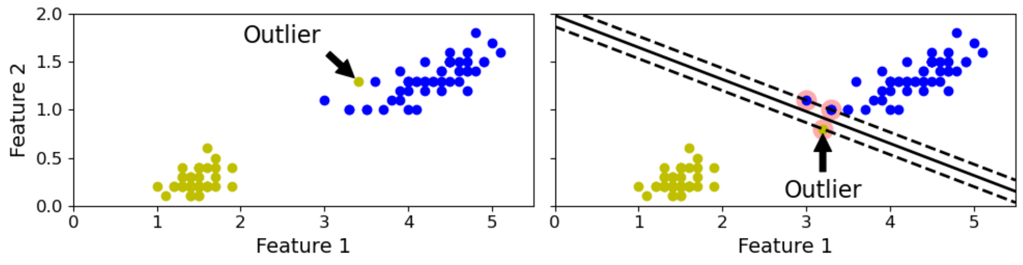


Figure 4.3. Hard-margin SVM sensitivity to outliers.[21]

A solution to this problem is to use soft-margin classification, as opposed to the hard-margin classification shown in Figure 4.3. Under soft-margin classification, the SVM model becomes more flexible, allowing some data points to appear within the margin or even on the wrong side of the margin. The objective of the soft-margin model is to find a balance between maximising the margin of the hyperplane and minimising the margin violations.

SVM Decision Function and Predictions

To find a separating hyperplane, the SVM model must learn a vector of weights,[21] **w**, and a bias term, b. Given a new data point, **x**, the classifier predicts the class of this data point by computing the decision function: h = $\mathbf{w}^T\mathbf{x} + b$ = $w_1 x_1$ + $\cdots$ + $w_n x_n$ + b. If the result of the decision function is positive, the model predicts the positive class (1); otherwise, it predicts the negative class (0). The full prediction function is shown in Equation 4.3.

$$\hat{y} = \begin{cases} 0, & \text{if } \mathbf{w}^\top \mathbf{x} + b < 0, \\ 1, & \text{if } \mathbf{w}^\top \mathbf{x} + b \geq 0. \end{cases} \tag{4.3}$$

Figure 4.4 shows an example 3D plot of the decision function of a binary SVM model; it appears as a 2D plane since the data has two features. The decision boundary, represented by the solid black line, is defined as the set of points where the decision function h is equal to zero. The dashed lines outline the margin of the SVM classifier, where the decision function equals 1 or -1. While training a linear, binary SVM model, the objective is to find weights and biases such that the margin is as wide as possible while limiting the number of margin violations.

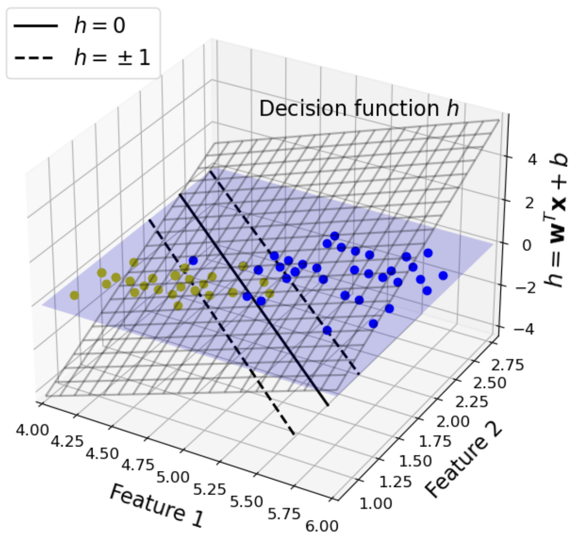


Figure 4.4. 3D SVM decision function.[21]

To understand the effect of the SVM weights on the size of the margin, it is useful to consider the relationship between the weights and the slope of the decision function. Since the weights define the projection of feature data points onto the decision plane, the slope of the decision function is equal to the norm of the weight vector, $\|\mathbf{w}\|$. If the slope of the decision plane is divided by two, the points where the decision function equals ± 1 will be twice as far from the decision boundary as before. That is, halving the slope of the decision plane will double the size of the margin.

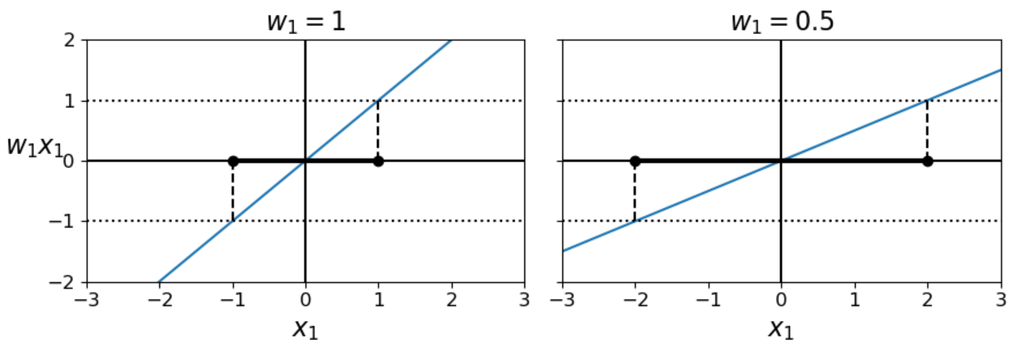


Figure 4.5. 2D SVM decision function slope.[21]

This concept can be illustrated using a 2D example, shown in Figure 4.5. In this example, only a single feature x_1 is considered, the weight vector has just one element, and the bias is set to zero. As before, the decision plane is defined as $\mathbf{w}^\top \mathbf{x} + b = w_1 x_1$, which corresponds to a straight line (shown in blue); the decision boundary now corresponds to the single point where the decision function equals zero. When $w_1 = 0.5$, the points at which the decision function equals ± 1 are twice as far away as when $w_1 = 1$; corresponding to a margin of double size, indicated by the horizontal bold lines.

Hence, $\|\mathbf{w}\|$ must be minimised in order to maximise the size of the margin. If the SVM is a hard-margin classifier, then there must also be no margin violations. This means that the decision function would need to be greater than 1 for all positive training examples and less than -1 for all negative training examples. Consider a vector **t** where t^i = 1 if data point i is in the positive class, and t^i = -1 if data point i is in the negative class. Then the margin constraints for a hard-margin classifier can be defined as $t^i(\mathbf{w}^\top \mathbf{x}^i + b) \geq 1$ for all instances i.

Hence, the objective of a hard-margin linear SVM classifier can be expressed as the constrained optimisation problem, shown in Equation 4.4.[21]

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimise}} \quad & \frac{1}{2}(\mathbf{w}^\top \mathbf{w}) \\ \text{subject to} \quad & t^i(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \quad \text{for } i = 1, 2, \ldots, n \end{aligned} \tag{4.4}$$

Note that the minimisation objective is $\frac{1}{2}(\mathbf{w}^\top \mathbf{w})$ = $\frac{1}{2}\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$ since the former has a simple derivative of **w** while the latter is not differentiable at **w** = 0. This is convenient for quadratic programming (QP) solvers[25], used to solve minimisation problems, which perform better when functions are fully differentiable.

To convert Equation 4.4 into an optimisation problem for a soft-margin classifier, it is necessary to introduce a slack variable, $\xi^{(i)} \geq 0$ for each data instance i. $\xi^{(i)}$ measures the degree to which the i^{th} instance is allowed to violate the margin; for a given instance, a higher the slack value corresponds to a greater licence to violate the margin. This introduces a new, conflicting objective to the minimisation problem: the values of the slack variables must be minimised to reduce margin violation, which conflicts with the objective of maximising the margins. The new optimisation problem for a soft-margin SVM classifier is shown in Equation 4.5.

$$\begin{aligned}
&\underset{\mathbf{w},b,\xi}{\text{minimise}} && \frac{1}{2}(\mathbf{w}^\top \mathbf{w}) + C\sum_{i=1}^{n} \xi^{(i)} \\
&\text{subject to} && t^i(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \xi^{(i)} \quad \text{and} \quad \xi^{(i)} \geq 0 \quad \text{for } i = 1, 2, \ldots, n
\end{aligned} \tag{4.5}$$

Notably, the soft-margin optimisation problem includes a hyperparameter, C, which is used to control the trade-off between the margin size and the margin violations.[21] A higher value of C produces a smaller margin since greater emphasis must be placed on minimising the slack variables; minimising the slack variables emphasises reducing margin violations and hence the margin must become smaller to compensate.

The SVM Dual Problem and the Kernel Trick

Intuitively, not all feature spaces are linearly separable and hence a linear SVM is not always sufficient to accurately separate classes. The kernel trick[26] is mathematical concept used in SVMs to simulate the mapping of features to higher dimensions such that a non-linear hyperplane can be found to separate data of differing classes.

The kernel trick operates upon an alternate form of the optimisation problem given in Equation 4.5, known as the dual form,[21] where Equation 4.5 is referred to as the primal form. In constrained optimisation problems, the solution to the dual form is typically a lower bound to the solution of the primal form and the solution to the primal form is an upper bound to the solution of the dual form. The SVM optimisation problem is a special case in which the dual and the primal forms have the same solution. The SVM dual problem is a standard result and is displayed in

Equation 4.6.

$$\underset{\alpha}{\text{minimise}} \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\top}\mathbf{x}^{(j)} - \sum_{i=1}^{n}\alpha^{(i)} \tag{4.6}$$

Here, α variables are terms of the dual form that a QP minimisation solver solves for to minimise the objective of the dual.

Once a solution vector **â** has bound found that minimises the objective of the dual, corresponding solutions for the weights, **ŵ**, and bias term, $\hat{b}$ can be found using standard mappings that are beyond the scope of the project.

The key utility of the dual SVM form relates to the $\mathbf{x}^{(i)\top}\mathbf{x}^{(j)}$ input vectors which are terms in the SVM dual objective. A kernel, K, can be applied to a pair of vectors (**a**, **b**) in order to map them to a higher, potentially infinite, number of dimensions and then compute their vector dot product. That is K(**a**, **b**) = $\phi(\mathbf{a})^\top\phi(\mathbf{b})$ where $\phi()$ defines the mapping to higher-dimensions. The kernel trick is a useful mathematical property that states that: K(**a**, **b**) = $\phi(\mathbf{a})^\top\phi(\mathbf{b})$ = $(\mathbf{a}^\top\mathbf{b})^2$.

Similarly, the kernel trick applies to feature vectors such that $\phi(\mathbf{x}^{(i)})^\top\phi(\mathbf{x}^{(j)})$ = $(\mathbf{x}^{(i)\top}\mathbf{x}^{(j)})^2$. This means that $\phi(\mathbf{x}^{(i)})^\top\phi(\mathbf{x}^{(j)})$ can be replaced in the objective of the SVM dual form with $(\mathbf{x}^{(i)\top}\mathbf{x}^{(j)})^2$, which means the QP solver attempting to minimise the SVM dual will be minimising the features as if they were mapped to higher dimensions by the kernel but the mapping is never explicitly computed due to the kernel trick.

In this project, SVM classifiers use the Gaussian RBF kernel: K(**a**, **b**) = $exp(-\gamma\|\mathbf{a} - \mathbf{b}\|)^2$; the RBF kernel in particular can simulate the mapping of features to near infinite dimensions due to the presence of the exponential term, which can be expressed in the form of an infinite Taylor series expansion.

4.1.4 Gaussian Process

Gaussian process (GP)[27]classifiers are a type of binary, probabilistic classifiers, where predictions on test data take the form of class probabilities. As with any classification model in this project, the object is to learn a function that maps welding features into welding classes. However, a GP model defines a distribution over an infinite number of functions that model this mapping, using Gaussian distributions to model the expected function output given a set of input features.

During the training of a GP model, a latent (hidden) function f is modelled as an infinite collection of random variables, any finite subset of which follows a multivariate Gaussian distribution. Each random variable corresponds to a point in the feature space; the idea is that a Gaussian process aims to model the function value at any possible input point in the feature space, $X \in \mathbb{R}^6$.

During training, to begin with, a Gaussian process prior is placed on the latent function by specifying a mean function and a covariance function. The prior's mean is set to zero by default and the prior's covariance is specified as a kernel function, such as the RBF kernel used by the SVM classifier. The primary aim of the GP prior is to model an expected function before any data has been observed; given observed training data and labels, the prior can be conditioned on the observed data to produce a GP posterior.

Notably, for classification tasks in particular, the latent function is referred to as a hidden function as its output values are not directly observed during the classification step. Instead, the intermediate output produced by the latent function is passed into a logistic link function, which squashes the real-valued output of the latent function into the range [0,1]. Specifically, when given a new data point, the model uses the GP posterior to compute the posterior distribution of the latent function value at this new point; this posterior distribution is Gaussian with a specific mean and variance. The mean value is then passed to the logistic link function which is used to produce a probability; a link function value of 0.5 or higher corresponds to a prediction of the positive class since GP classifiers are binary by nature. The variance, while not directly used to compute predicted probabilities, can be used as a measure of the model's confidence in the final predicted probability.

4.1.5 Gaussian Naive Bayes

Gaussian Naive Bayes (NB) classifiers are probabilistic classifiers based on Bayes' Theorem.[28] Given a feature vector $x_1, ..., x_m$, the objective is to estimate the probability $p(y \mid x_1, ..., x_m)$ which is the conditional probability that the feature vector corresponds to a specific class label y.[29] This conditional probability can alternatively be expressed using Bayes' rule, shown in Equation 4.7.

$$p(y \mid x_1, ..., x_m) = \frac{p(y)p(x_1, \ldots, x_m \mid y)}{p(x_1, \ldots, x_m)} \tag{4.7}$$

Crucially, Naive Bayes classifiers are referred to as "naive" because they operate using the assumption that every pair of features are conditionally independent, given the value of the class label y. For example, if the welding class is known to be SW, knowing information about the temperature signal provides no additional information about the plasma signal. While this assumption may not be accurate, it allows the NB model to make relatively swift predictions without having to model the intricate relationships between every combination of features. This naive conditional independence assumption is defined in Equation 4.8

$$P(x_i \mid y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m) = P(x_i \mid y) \tag{4.8}$$

Applying the naive assumption, for all features x_i, to Equation 4.7 yields Equation 4.9.

$$P(y \mid x_1, \ldots, x_m) = \frac{P(y) \prod_{i=1}^{m} P(x_i \mid y)}{P(x_1, \ldots, x_m)} \tag{4.9}$$

For a fixed feature vector, $P(x_1, \ldots, x_m)$ is constant for all class labels; this means that the final predicted class, $\hat{y}$, can be simplified to Equation 4.10.

$$\begin{gathered} P(y \mid x_1, \ldots, x_m) \propto P(y) \prod_{i=1}^{m} P(x_i \mid y) \\ \Downarrow \\ \hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{m} P(x_i \mid y), \end{gathered} \tag{4.10}$$

For Gaussian NB classifiers, the training phase involves using the training data to compute suitable values for P(y) and $P(x_i \mid y)$; the former can be computed by simply using the proportion of training examples with a class label of y. The latter is computed by segregating the training data by classes and then modelling each feature within a class using a Gaussian distribution, as shown in Equation 4.11.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{4.11}$$

The parameters μ_y and σ_y correspond respectively to the mean and standard deviation for the Gaussian distribution that models feature x_i in class y, where the Gaussian model can be fitted using standard maximum likelihood techniques. For example, the model will consider the subset of feature vectors that have a LoC class label; all of the data relating to the μ_T welding feature in this subset can then be fitted to a Gaussian distribution with a given mean and standard deviation.

When given a new data point to classify, the likelihood of the new data point's μ_T value, assuming the data point is in the LoC class, can be computed using the fitted distribution. This process is repeated for all of the remaining welding features and Equation 4.10 can be used to estimate a likelihood score for the LoC class. This estimation can be repeated for the SW and OP classes and the data point will be classified into the class with the highest estimated likelhood.

4.1.6 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis (QDA) classifiers are multi-class classifiers that operate by finding quadratic (as opposed to linear) decision boundaries to separate the classes.[30] They are derived from probabilistic models which model the probability $P(X|y = k)$, corresponding to the probability of observing the dataset, **X**, conditioned on a class label y=k. As with Naive Bayes classifiers, Bayes' rule can be used to produce predictions for each testing sample $x \in \mathcal{R}^d$, as shown in Equation 4.12.

$$P(y = k \mid x) = \frac{P(x \mid y = k)P(y = k)}{P(x)} \tag{4.12}$$

Intuitively, the predicted class will be the class k which maximises the posterior probability $P(y = k \mid x)$. QDA classifiers differ from Gaussian NB classifiers in that, instead of making a naive conditional independence assumption about the features, they model the distribution $P(x \mid y)$ as a multivariate Gaussian distribution, defined in Equation 4.13.

$$P(x \mid y = k) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)\right) \tag{4.13}$$

Here, μ_k corresponds to the d-dimensional mean vector, which contains the mean values for all d=6 feature variables for the class k. Σ_k denotes the d$\times$d covariance matrix, which describes the relationships and variances between every pair of feature variables in the data for class k. This is in stark contrast to the Gaussian NB model which modelled each feature variable using independent, uni-variate Gaussian distributions.

The computation of the predicted class can be simplified by first taking the natural logarithm of the posterior distribution in Equation 4.13 to produce Equation 4.14.

$$\begin{aligned}\log P(y = k \mid x) &= \log P(x \mid y = k) + \log P(y = k) + C \\ &= -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x - \mu_k)^T\Sigma_k^{-1}(x - \mu_k) + \log P(y = k) + C\end{aligned} \tag{4.14}$$

Here, $\log P(y = k)$ is the log of the prior probability of class k, which is computed as the proportion of training examples with a class label of k. The constant C simply aggregates the Bayes' denominator P(x), which is constant for all classes k, as well as the constant terms produced from the Gaussian model. The predicted class is the class that maximises the log posterior; since the natural logarithm is an increasing function, higher values of $\log P(y = k \mid x)$ correlate with higher values of $P(y = k \mid x)$.

4.2 Ensemble Classifiers

Ensemble classifiers are machine learning models whose class predictions are produced by aggregating the predictions of several constituent machine learning predictions.[21] This project considers three different types of ensemble learning techniques: bagging, boosting and voting, with the voting models.

This section discusses one bagging algorithm, random forest, as this algorithm was included in the benchmark experiments; two forms of boosting models are briefly discussed: AdaBoost and Gradient Boosting. Notably, the bagging and boosting classifiers were primarily added as project extensions, with the greatest emphasis placed on the voting models, which themselves can aggregate other ensemble models, including the bagging and boosting models.

4.2.1 Random Forests

Random forest classifiers classify data points by aggregating multiple decision tree models using the bagging ensemble technique.[21] The idea behind bagging is to train multiple base classifiers, such as decision trees, on random subsets of the training data and then combine their predictions by outputting the modal class predicted by the individual base classifiers. The random subsets are constructed by sampling the training data with replacement, meaning samples can occur multiple times in each subset. While this risks increasing the bias of the individual classifiers, the aggregation process reduces both bias and variance since generalisation errors are distributed across the ensemble. This is particularly useful when aggregating decision trees which are prone to overfitting due to their relatively high number of model parameters.

Figure 4.6 demonstrates the basic bagging framework of the random forest classifier used in the project. N=100 random training sets were sampled, each containing 90% of the number of samples as the original training set. This upper bound on the size of the random subsets was introduced in order to increase the diversity of the base decision trees. An additional source of randomness is introduced by restricting the number of features considered when splitting a node.

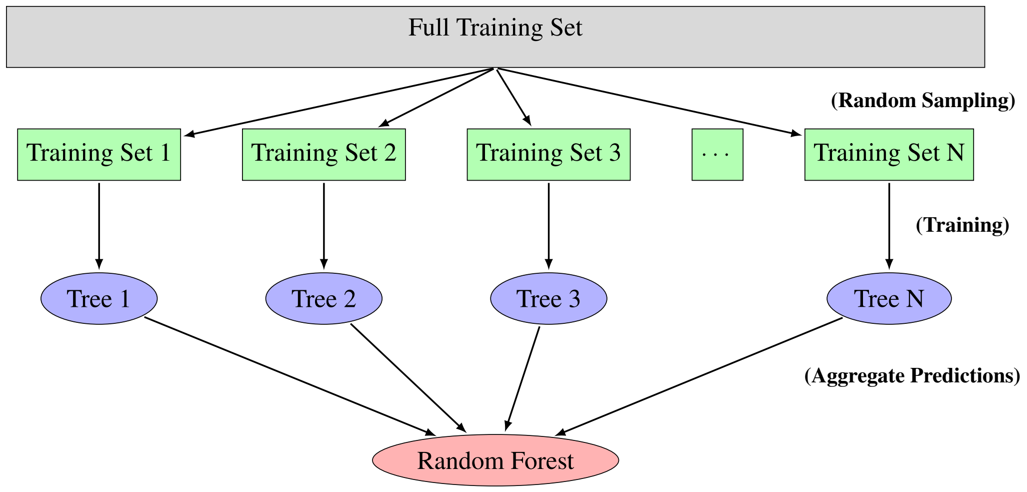


Figure 4.6. A random forest bagging framework

In a standard decision tree, the model selects the best feature and threshold split as described in section 4.1.1. However, the random forest model selects the best feature among a random subset of features, further increasing model diversity.

4.2.2 AdaBoost

Boosting refers to Ensemble methods that can combine several weak learners into a strong learner. The general idea is to train classifiers sequentially, with each successive classifier attempting to correct the mistakes made by its predecessor.[31] In this project, the AdaBoost classifier aggregates N=100 decision trees, analogously to the random forest classifier. However, unlike random forests, AdaBoost makes use of weak decision tree learners which have a restricted maximum depth of one or two; the idea is to combine several of these weak decision tree learners into a strong learner via weighted aggregation of the ensemble. A natural consequence of using weak learners is that they are generally not complex enough to overfit the data but rather tend to underfit the data. While the random forest classifier trains decision trees in parallel, AdaBoost sequentially corrects successive decision tree classifiers by paying extra attention to the training instances that the predecessor underfitted; that is, each successive learner focuses more on the data points that are harder to classify. Figure 4.7 shows the general framework for training an AdaBoost ensemble.

During training, all samples are assigned uniform weights of $\frac{1}{n}$ where n is the number of examples in the training set. The AdaBoost algorithm first trains a base decision tree as normal and uses it to produce predictions on the training set; a weighted error rate r_1 is computed using Equation 4.15

$$r_j = \frac{\sum_{i=1}^{n} w^{(i)} \cdot \left(\hat{y}_j^{(i)} \neq y^{(i)}\right)}{\sum_{i=1}^{n} w^{(i)}} \tag{4.15}$$

where $\hat{y}_j^{(i)}$ is the j^{th} classifier's prediction for the i^{th} training instance.

That is, the error rate is the weighted sum of misclassified examples divided by the total weight of all samples; samples that are correctly classified contribute nothing to the error rate.

The weighted error rate can then be used to assign a weight, α_j, to the classifier, using Equation 4.16, where η is a learning rate hyperparameter. The lower the error rate r_j, the larger the value of $(\frac{1-r_j}{r_j})$; that is, the more accurate the classifier is, the higher its assigned weight will be.

$$a_j = \eta \cdot log(\frac{1 - r_j}{r_j}) \tag{4.16}$$

After producing a weight for the classifier, the AdaBoost algorithm updates the instance weights using Equation 4.17. Only the weights of misclassified instances are boosted and the amount the weights are boosted by is exponentially

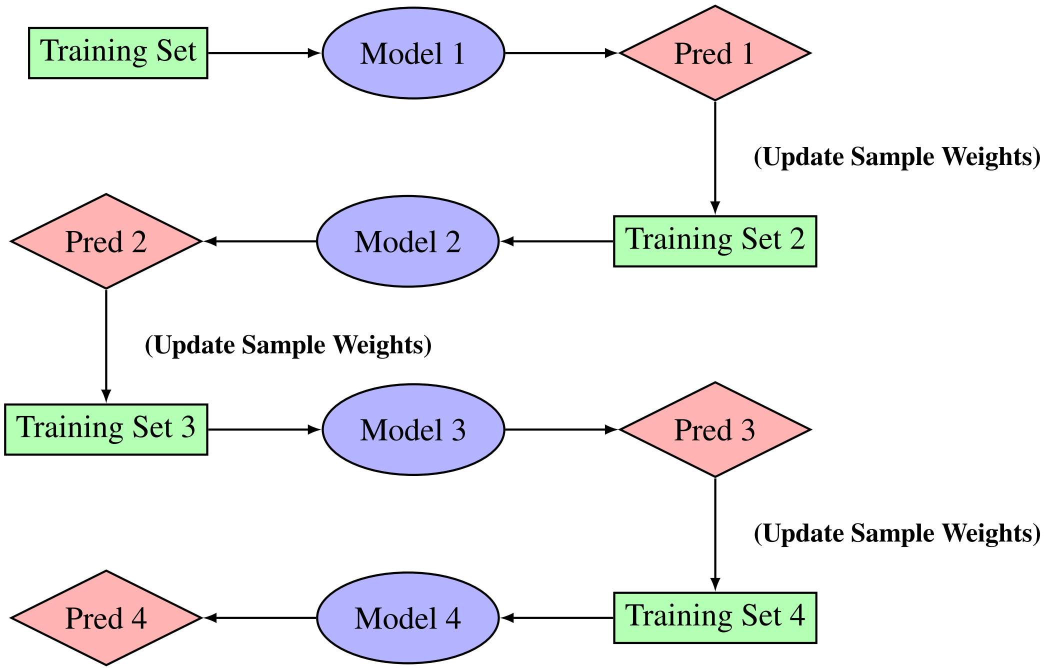


Figure 4.7. AdaBoost sequential training with sample weights updates after each prediction.

proportional to the weight of the classifier. That is, misclassifications produced by high-weight classifiers are given greater consideration than misclassifications produced by lower-weight classifiers.

$$\text{for i = 1, 2,} \cdots \text{, n} \\ w^{(i)} \leftarrow \begin{cases} w^{(i)}, & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j), & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases} \tag{4.17}$$

Once the instance weights have been updated, they are subsequently normalised by dividing them by the total sum of all instance weights. The process then repeats by training the next classifier using the newly updated weights and this continues until a fixed number of predictors is reached, or when an ideal classifier has been found.

When making predictions, AdaBoost computes the predictions of all classifiers and weights them using the classifier weights, α_j. The final class prediction is that which receives the highest weighted vote across all classifiers in the ensemble.

4.2.3 Gradient Boosting

Gradient Boosting operates in a similar manner to AdaBoost,[32] sequentially adding classifiers, typically decision trees, to an ensemble, with each successive classifier correcting its predecessor. However, the key distinction between AdaBoost and Gradient Boosting is that Gradient Boosting doe not modify any instance weights; instead, it aims to fit the new classifiers to residual errors produced by the previous classifier. In this project, it is assumed that all forms of boosting classifiers operate on constituent decision trees.

Given a training set, the Gradient Boosting algorithm begins by creating a single leaf which aims to produce an initial prediction for all training examples using log odds, which acts as an “average” prediction. The algorithm then converts these log odds into initial probabilities using a logistic function.

In a simple binary classification case, the leaf can consider which class has the highest probability, p, within the training data and simply predict that all examples in the training data have a probability p of belonging to the majority class. Subsequently, the algorithm can predict a pseudo-residual for every example, where the residual is the difference between the predicted probability of an example belonging to a given class and the actual probability that it belongs to that class. Naturally, the actual probabilities will either be 1 or 0 depending on whether or not the training example belongs to the considered class.

After computing the residuals, the algorithm uses the feature data to fit a simple decision tree to the residual probabilities. That is, in this case, the target labels are the residual probabilities rather than the original class labels, although data points with the same original class label will have the same residuals.

The general goal is to combine the initial average prediction from the log odds with the fitted tree to produce modified log odds and then probability predictions. While the initial predicted probability is likely to be significantly different from the actual probability, the decision tree aims to fit the feature data to the residual probabilities, hence learning information about how each data example deviates from the initial prediction. The Gradient Boosting algorithm makes use of a large ensemble of decision trees to iteratively improve the predicted probability, which is a linear combination of the initial prediction and the probability deviations produced by each tree in the ensemble.

4.2.4 Voting Models

Voting ensemble classifiers operate by granting each of their constituent classifiers a “vote” towards their predicted class.[21] As outlined in the project motivation, the key principle behind voting-based ensembles is that classification models that use different learning algorithms may misclassify data points in different ways. If one specific model fails to classify a data point, it may be possible that the other models in the ensemble classify the same point correctly.

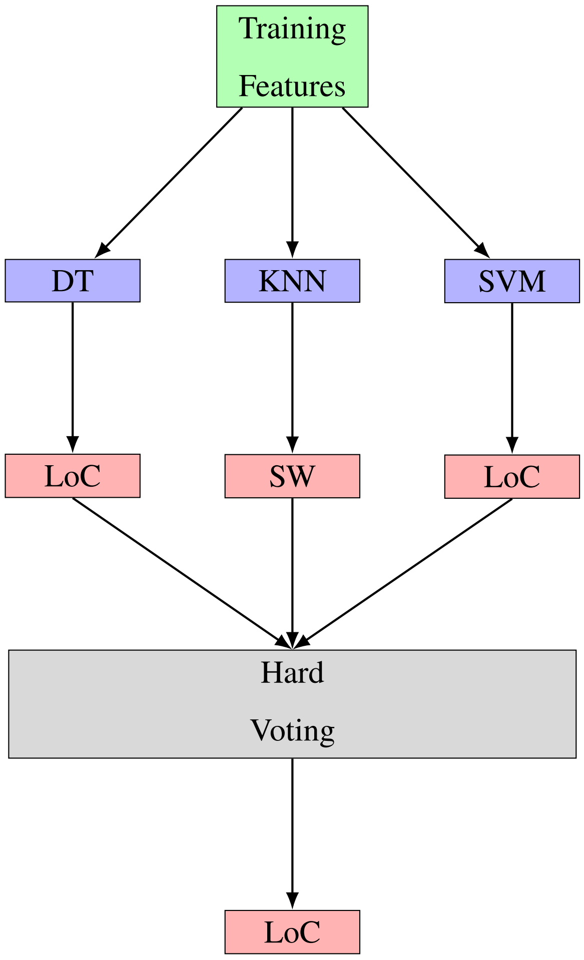


Figure 4.8. Hard voting architecture.

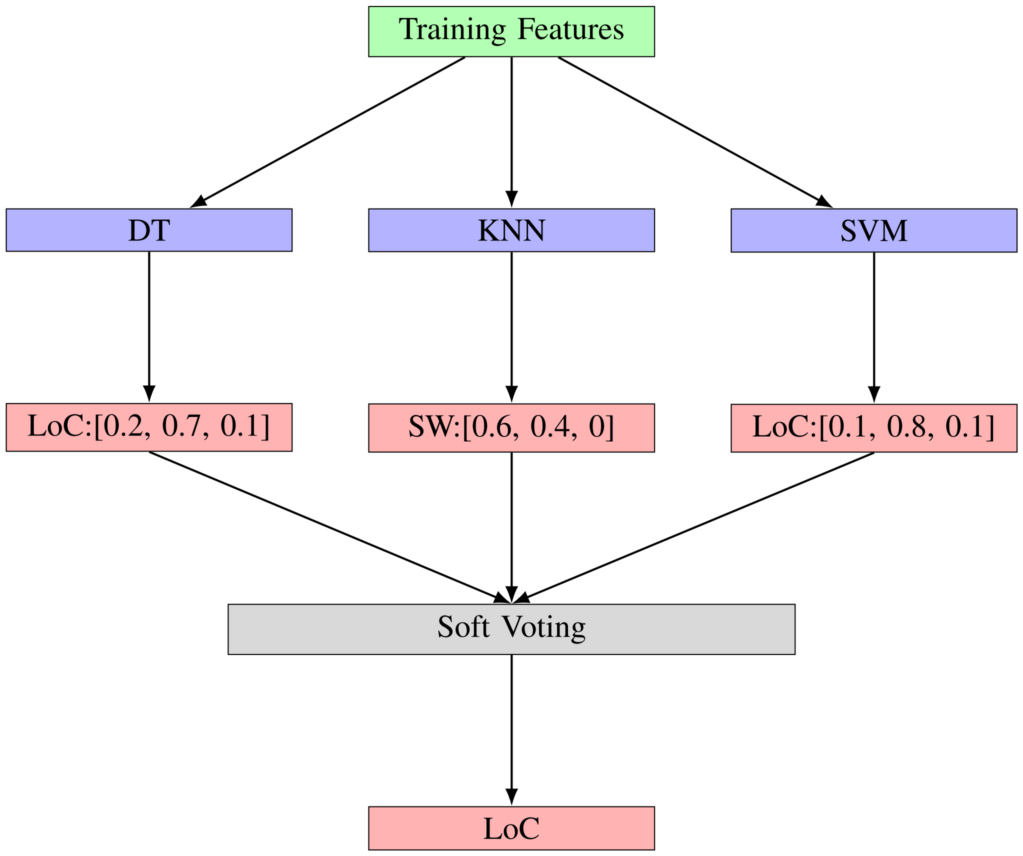


Figure 4.9. Soft voting architecture.

In total, there are two voting mechanisms used to aggregate the predictions of constituent classifiers: hard-voting (HV) and soft-voting (SV). Hard-voting classifiers operate by computing a simple majority class vote as shown in Figure 4.8. On the other hand, soft-voting classifiers aggregate the predicted probabilities of each of its constituent classifiers; the class predicted is that which has the highest predicted probability summed over all classifiers in the ensemble. Figure 4.9 demonstrates the standard architecture of soft voting models in this project.

4.3 Deep Learning and Neural Network Classifiers

In a machine learning context, neural networks (NNs) consist of layers of interconnected artificial "neurons" designed to learn underlying patterns from input data.[21] For classification tasks, neural networks receive feature data through an input layer and predict classes using the output of neurons from an output layer. Neurons in a given layer are connected to neurons in adjacent layers via weighted connections and these connection weights can be learned during the training phase to tune the predictions produced at the output layer. Crucially, additional layers of neurons, known as hidden layers, can exist between the input and output layers; neural networks typically require at least one layer of hidden neurons to be considered "deep" learning models.

As discussed in section 3.2.2, previous experiments demonstrated that neural networks have significant welding classification potential when applied to time-series signal data. However, at present, only single-layer perceptron (SLP) NNs have been considered which do not make use of hidden layers and hence are not considered to be deep learning models. In fact, deep learning models remain untested in this subfield of remote laser welding despite having greater classification potential due to the increased complexity offered by the hidden layers.

This section offers a general overview of the operation of neural networks as well as a discussion on the architectures of the three main deep learning models used in this project: fully-connected neural networks (FCNNs), long short-term memory neural networks (LSTMs) and convolutional neural networks (CNNs).

4.3.1 Fully Connected Neural Networks

Before discussing the architecture of the fully-connected neural network (FCNN), it is useful to describe the individual logic units that comprise larger neural network structures.[21] The threshold logic unit (TLU), also referred to as a “neuron”, is a type of artificial neuron that computes a weighted sum of its inputs, as shown in 4.10. The output of a TLU’s weighted sum can be notationally defined as z =

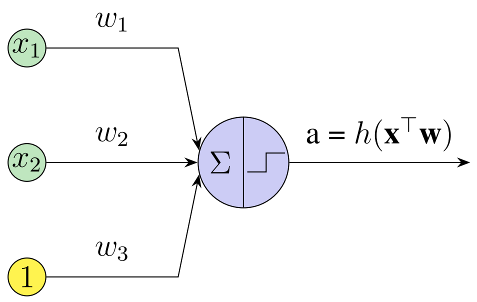


Figure 4.10. Threshold logic unit: the basic unit used to compose neural networks. The unit first computes a weighted sum, z, of its inputs and then applies a step activation function, h(z), to produce an activation output, a.

$w_1x_1 + w_2x_2 + \cdots + w_nx_n$ = $\mathbf{x}^\top\mathbf{w}$, where **x** denotes a vector containing the inputs and **w** denotes a vector containing the corresponding weights. Additionally, TLUs also make use of a weighted bias input, shown in yellow in Figure 4.10. The purpose of this bias is to allow the neuron to offset its weighted sum z when the inputs are either zero or close to zero, otherwise, all such inputs would enforce a weighted sum of approximately zero by default. Notably, every TLU includes this constant bias term, which is defined as a weighted multiple of 1. In order for the definition of the weighted sum z = $\mathbf{x}^\top\mathbf{w}$ to remain notationally consistent, a constant value of 1 can be appended to the input vector **x** and its corresponding weight term can be appended to the weight vector **w**. To reduce verbosity, further discussion of neural networks operates on the assumption that the bias terms are implicitly included in the weights and hence they are omitted from network diagrams.

After computing the pre-activation value z, the TLU applies the Heaviside step activation function, h(z), to the weighted sum, as defined in Equation 4.18.

$$h(z) = \begin{cases} 0, & \text{if } z < 0, \\ 1, & \text{if } z \geq 0. \end{cases} \tag{4.18}$$

This step function controls whether or not the neuron "fires" by outputting a 1 if z is greater than or equal to zero and outputting zero otherwise.

Single-Layer Perceptron

The binary nature of the Heaviside function allows a single TLU to be used for simple linear binary classification, however several TLU units can be combined in a layer to produce a single-layer perceptron. Figure 4.11 demonstrates the basic structure of a single-layer perception. Under the SLP architecture, n inputs are passed into the network via input layer neurons and every neuron in the input layer has a weighted connection to three output neurons. In this project, n = 2500$\times$3 = 7500, since the neural network models operate on the time-series signal data defined in section 3.2.1.

Here, the network is considered to be a "single-layer" network as the output layer is the only layer in which weighted sums and activation values are computed; the input layer acts as a pass-through layer through which the feature data is fed into the network. For clarity, neurons that output an activation value are denoted by $a_i^(j)$ which corresponds to the i^{th} neuron in the j^{th} layer. As discussed, there are three output neurons in total, one for each class prediction; class predictions

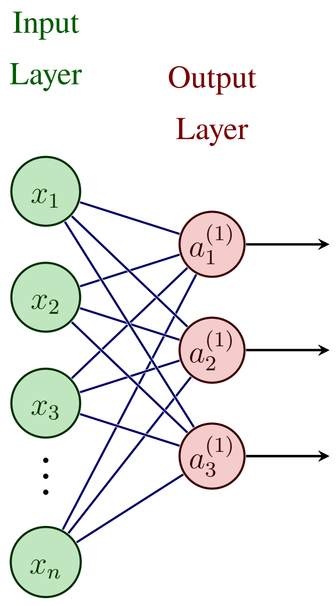


Figure 4.11. Single-layer perception architecture.

can be generated by selecting the class with the highest corresponding output or by using the standard OvO or OvR techniques discussed in section 2.3.4.

FCNN Architecture

A major disadvantage of SLP classifiers is that they are only able to identify linear decision boundaries[21] due to their limited model complexity; this problem can be addressed by adding additional, hidden layers to the network, forming a multi-layer perceptron. A fully-connected neural network is defined as a multi-layer perceptron in which neurons in a given layer are connected to all neurons in adjacent layers.

Figure 4.12 displays the primary architecture of the FCNN model used in this project.

In constrast to a SLP, the FCNN utilises a single hidden layer with 64 neurons

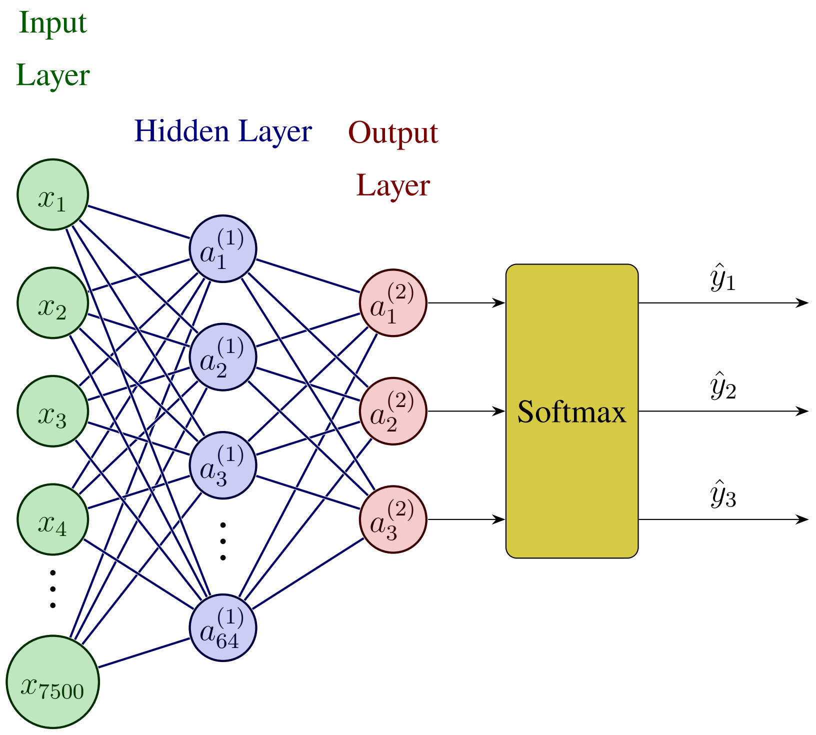


Figure 4.12. Fully-connected neural network architecture.

as well as a softmax layer[21] which is used to ensure that values produced by the output layer correspond to class probabilities (values between 0 and 1) that add up to 1. The final class prediction, $\hat{y}$ will be that which corresponds with the output neuron that produces the highest softmax probability.

Another key distinction between the FCNN and SLP architectures is that the FCNN model utilises the ReLU activation function, defined as ReLU(z) = max(0, z), in place of the Heaviside step activation function for both the hidden and output layers. The primary reason for this is due to the differentiation properties of the ReLU function compared to the step function, as can be observed in Figure 4.13.

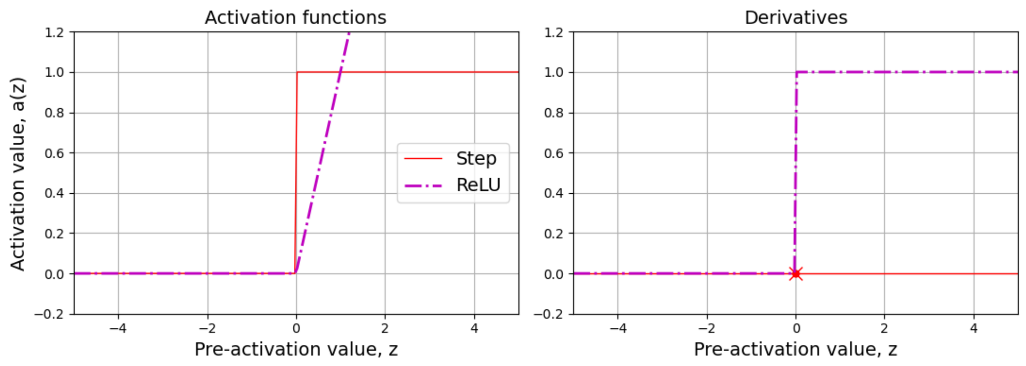


Figure 4.13. A comparison of the ReLU and step activation functions (left) and their derivatives (right).[21]

The derivative of the ReLU function is 1 when z is strictly positive and 0 when z is strictly negative, whereas the step function has a derivative of 0 at all differentiable points. The significance of these derivatives relates to the backpropagation algorithm which requires the use of non-zero activation derivatives to tweak the weight and bias parameters during training; the details of backpropagation are discussed further in section 4.3.1.

Backpropagation

As discussed, backpropagation[21] is the algorithm used by neural networks to learn suitable weight and bias values during training. Before training, the weight and bias parameters must first be initialised either randomly or using a preset initialisation strategy. Once the weights and biases have been initialised, neural network training is conducted by first performing a feedforward step in which data instances are processed in batches (for example, 16 instances at a time) and the output of every neuron at every layer is computed (for every instance), starting from the input layer and propagating forward to the three output layer neurons.

Once these output values, ($\hat{y}_1$,$\hat{y}_2$, $\hat{y}_3$) have been computed, the softmax function, shown in Equation 4.19, can be used to compute the predicted probability, p_k, for each output neuron, concluding the forward pass of the network.

$$p_k = \text{softmax}(\hat{y}_k) = \frac{\exp(\hat{y}_k)}{\sum_{j=1}^{3} \exp(\hat{y}_j)} \tag{4.19}$$

At the end of the forward pass, every instance in the batch will be associated with three predicted probabilities, (p_1, p_2, p_3), corresponding to SW, LoC and OP respectively. In order to update the network parameters, the algorithm requires a means of measuring the error in the predictions across the batch; this is done using the cross-entropy cost function, shown in Equation 4.20. Here, for instance i in the training batch, C_i denotes the cost and $y_k^{(i)}$ is the target probability that instance i belongs to class k. That is, if instance i does belong to class k, the target probability should be 1, otherwise, the target probability should be 0. Finally, $\log(p_k^{(i)}$ is the predicted probability that instance i belongs to class k.

$$C_i = -\sum_{k=1}^{3} y_k^{(i)} \log(p_k^{(i)}) \tag{4.20}$$

The cost function is intuitive since, if $p_k^{(i)}$ is 1 and k is the correct class, then the summation simplifies to -$\log(p_k^{(i)}$ = -$\log(1)$ = 0, implying there is no cost for a correct prediction. Conversely, if $p_k^{(i)}$ is 1 and k is not the correct class, then the summation simplifies to -$\log(0)$, implying an infinite error when the network assigns zero probability to the correct class. The final cost for the entire batch is computed as the average instance cost across the batch.

Once the cost has been computed, the backpropagation algorithm computes the degree to which every weight and bias parameter contributes to the cost. Given these local cost derivatives (gradients), standard cost minimisation techniques such as Gradient Descent can be used to minimise the cost. To be specific, Gradient Descent measures the local gradient of the cost with respect to a given parameter and tweaks that parameter in the direction of descending gradient; the over-arching goal is to have cost gradients close to zero, indicating that the error is close to a miniminum value.

The computation of these gradients is carried out by indirectly calculating the partial derivative of the cost with respect to every model parameter, using the chain rule. The intuition behind this use of the chain rule can be understood by considering the influence of some weight value, w1, on the error of the model. The value of weight w1 influences the pre-activation value, z1, of some neuron, n1, in the network since w1 is associated with a weighted connection in the network. The value of z1, then influences the activation value, a1, of neuron n1; these interactions are propagated forward through the network such that w1 influences all of the output neurons (since the network is fully connected)and by extension the total cost. Hence, the partial derivative of the cost with respect to w1, denoted by $\frac{\partial C}{\partial w1}$ can be computed using the chain rule by backpropagating all of the partial derivatives across all paths influenced by w1. This requires multiplying chains of derivatives which include the derivatives of activation functions; if any derivative in the chain is zero then the product of all chained derivatives will also be zero. Hence, if the partial derivative of the cost with respect to a given parameter is computed to be zero then then this means that the Gradient Descent algorithm has no means of updating that parameter, hence reinforcing the need for activation functions with non-zero gradients such as the ReLU function.

4.3.2 Long Short-Term Memory Neural Networks

While the FCNN and SLP architectures are capable of handling time-series data, the long short-term memory (LSTM) architecture was specifically designed to handle sequenced data,[21] including time-series data. In particular, LSTM cells attempt to retain information from previous time steps by maintaining two types of internal memory states: a hidden state h(t) and a cell state c(t). Here, the hidden state acts as a form of short-term memory while the cell state acts as a form of long-term memory. Both states are a function of time; the values of the hidden and cell states at time t are directly influenced by the state values at time t-1, as well as the input value at time t, x(t). Here, the input value at time t corresponds to a time-dependent feature value, such as the value of the back reflection signal at timestep t=1280. Notably, the LSTM cell also produces an output, y(t), at every timestep; this output is identical to the output of the hidden state.

Figure 4.14 displays the architecture of an individual LSTM cell. Naturally, LSTM cells need a mechanism to determine what should be stored in long and

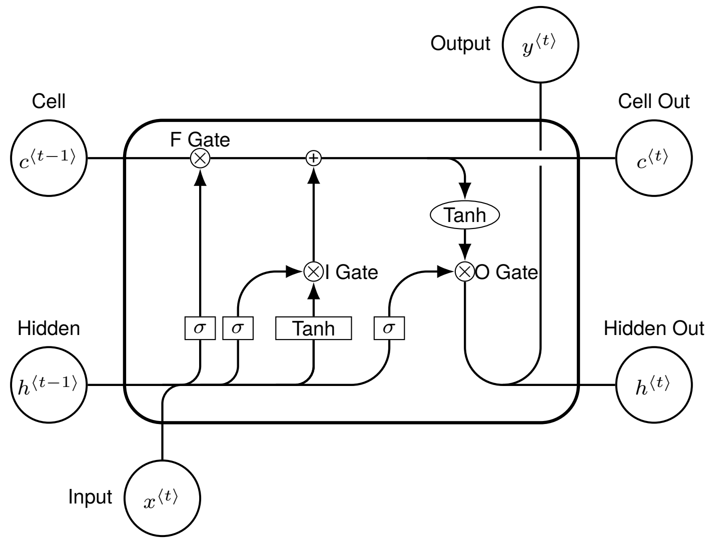


Figure 4.14. Architecture of an LSTM unit.

short-term memory; this mechanism is regulated via three essential gates: the forget (F) gate, the input (I) gate and the output (O) gate. The forget gate is responsible for controlling how much of the long-term memory information, c(t-1) is retained at each time step. The input gate uses the values of x(t) and h(t-1) to control how much new information is added to the cell state. That is, the input gate regulates how much of the information from the current input and the previous short-term memory should be added to the long-term memory. After passing through the forget gate and the input gate, the modified cell-state memory value is output as the new long-term memory, c(t).

At the same time, an alternate copy of c(t) is passed through the tanh() function, and the output from the tanh() function is filtered by the output gate to produce the output of the cell at the current time step (y(t)) as well as the hidden state value h(t). Another application of the tanh() function in the LSTM cell is to control how much information the current input value, x(t), and the previous hidden state value, h(t-1), contribute to the long-term memory via the input gate.

The remaining components of the LSTM cell are the σ operators which correspond to gate regulators; that is, they regulate the degree to which the F, I and O gates are "open", using the sigmoid function, defined in Equation 4.21.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.21}$$

Since the sigmoid function is defined between 0 and 1, the sigmoid gate regulators produce a multiplicative factor that controls the proportion of information that passes through the gate. For example, if the sigmoid functions output a 0, no information will pass through the gate while a sigmoid output of 1 means that all information passes through the gate.

Weighted connections between the input values, x(t), and the previos hidden state values, h(t-1) and the sigmoid/tanh gate regulator functions; these weights can be learned via an algorithm known as backpropogation through time, which is analogous to the backpropagation algorithm used learn the weights of an FCNN model. In this project, 64 LSTM units were trained in parallel, across the 2500 time steps. The output values produced at the final time step were fed into a dense, fully-connected layer with 3 neurons (corresponding to output neurons) from which classification was performed in an identical manner to the FCNN model.

4.3.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) process time-series data by sliding kernels across the 2500$\times$3 signal data,[21] convolving the kernel with the data to produce new feature map examples, as shown in Figure 4.15. This project considers 1D convolutional neural networks with 3×3 kernels.

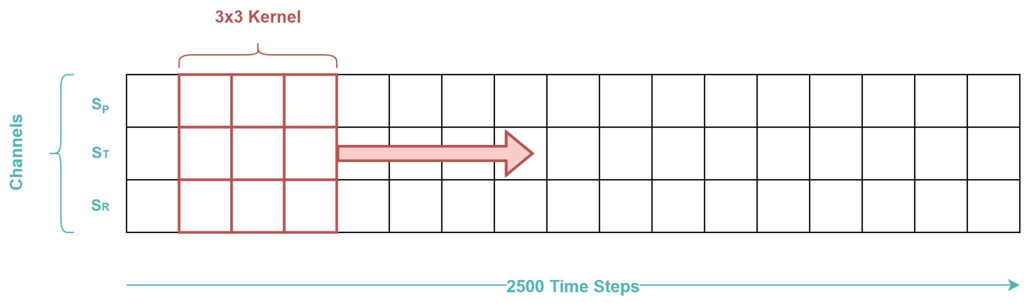


Figure 4.15. A 3×3 kernel sliding across a feature map.

To begin with, the 3×3 kernel is placed at the start of the 2D time-series data. Every square of the kernel is initialised with weight parameters using standard random initialisation techniques;[33] a convolution is performed by computing an element-wise weighted sum of the overlap between the kernel and the feature data. That is, for every square where a kernel value 'overlaps' with a feature value, the corresponding kernel weight is multiplied by the value of the feature data and the sum of all such weighted multiplications is computed to produce a final feature map value. By sliding the kernel across the full length of the feature data, a new feature map can be constructed with a new feature value for almost all of the 2500 time steps.

The kernel width of 3 is chosen such that it has the same width as the feature data to capture information from all three of the signal channels: S_P, S_T, S_R; since the kernel width matches the data width, the network architecture is considered to be a 1D CNN since the kernel can only slide in one direction. Notably, the size of the new feature map, in this case, is actually 2498 since the kernel occupies 3 time steps to begin with because of its size and hence can only slide forward 2498 times to produce a total feature map with 2498 new features.

The overall objective is to slide several different kernels with different weights across the input data, each one producing a slightly different feature map which encodes information about the structure of the time series data. These new features are then passed into a fully connected neural network which can be used to perform classification.

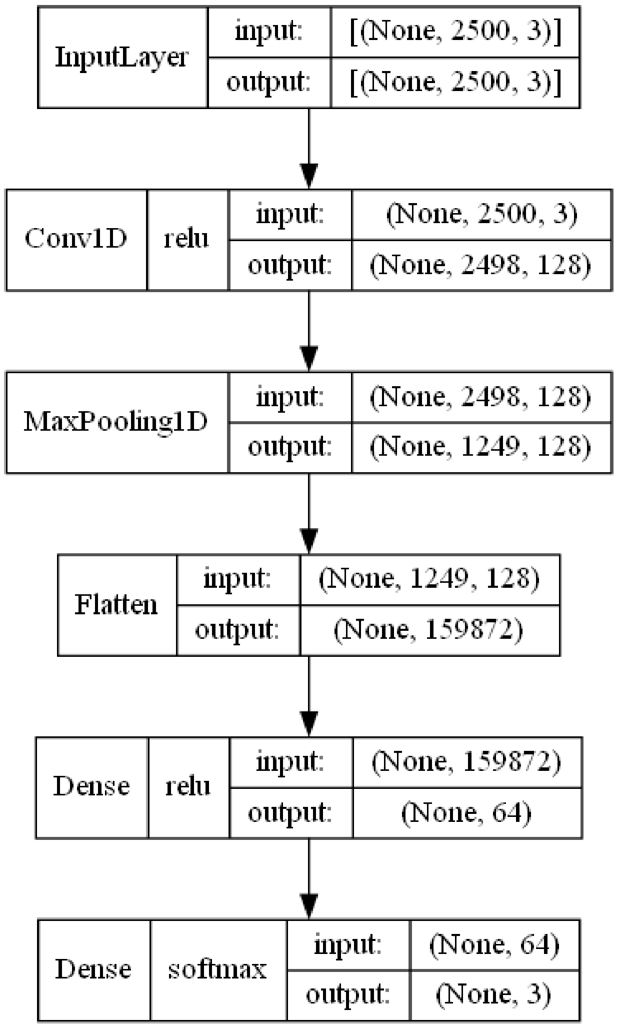


Figure 4.16. Architecture of 1D CNN model

Figure 4.16 displays the default architecture of the CNN model used in this project. As discussed, the first layer (referred to as a convolutional layer) uses kernels to produce feature maps with 2498 new features; 128 kernels are used by default resulting in 128 such feature maps of size 2498. After this, a max pooling layer is applied in which the feature maps are split into sequential pairs of values, with the higher feature value retained and the lower feature value dropped; the purpose of the max pooling layer is to reduce the size of the learned feature maps to increase model efficiency. The final output of the max pooling layer is 128 feature maps of size 1249 as the feature maps are halved in size.

Finally, the new features stored in the feature maps are flattened and passed into a fully-connected neural network for classification.

4.4 Model Validation

As discussed in section 3.2.2, leave-one-out cross-validation was used as the primary means of comparing model accuracy. However, since ensemble and deep learning classifiers required substantially more training time than the benchmark models, it was decided that 5-fold cross-validation (5CV) would be used to compare model performance.

Under 5CV, the datasets are split into five folds of roughly equal size. Four of these folds are then selected for use as training data with the fifth fold held out as testing data and this is repeated such that all five folds are used as test data exactly once. Since every example in a dataset is used as test data, this approach still enables accurate comparisons with the benchmark tests from the existing research.

Additionally, early experiments suggested that model performance was highly

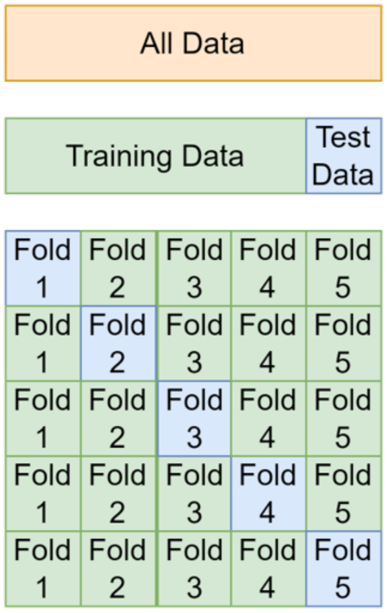


Figure 4.17. 5-fold cross validation.

sensitive to the imbalanced distribution of classes in the training folds, which was one of the three major challenges highlighted in section 3.3. As a solution, stratified cross-validation was used which meant that test folds were generated such that the class balances were preserved across all test folds as much as possible.

The impact of using stratified test folds can be seen in Figure 4.18 (without stratified folds) and Figure 4.19 (with stratified folds). In particular, the results for the smallest dataset - dataset 1 (A), shown in blue - highlight the sensitivities of models to imbalanced classes with all models performing significantly better using stratified test folds. As a result, all experiments discussed in section 5 were performed using stratified test folds.

It is important to note that the results shown in Figures 4.18 and 4.19 were produced using 10-fold cross-validation (10CV) which was originally preferred

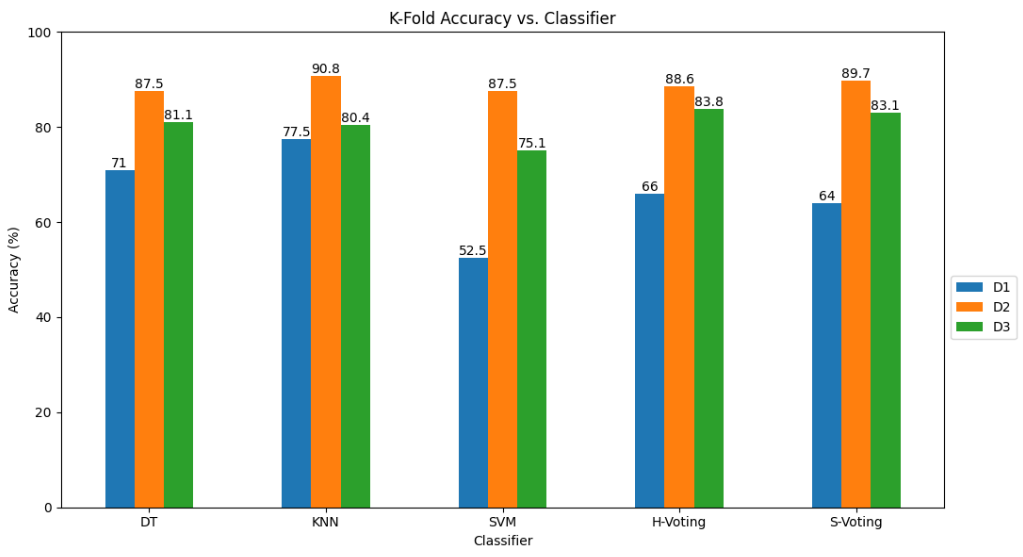


Figure 4.18. Results of 10-fold CV without stratified test fold.

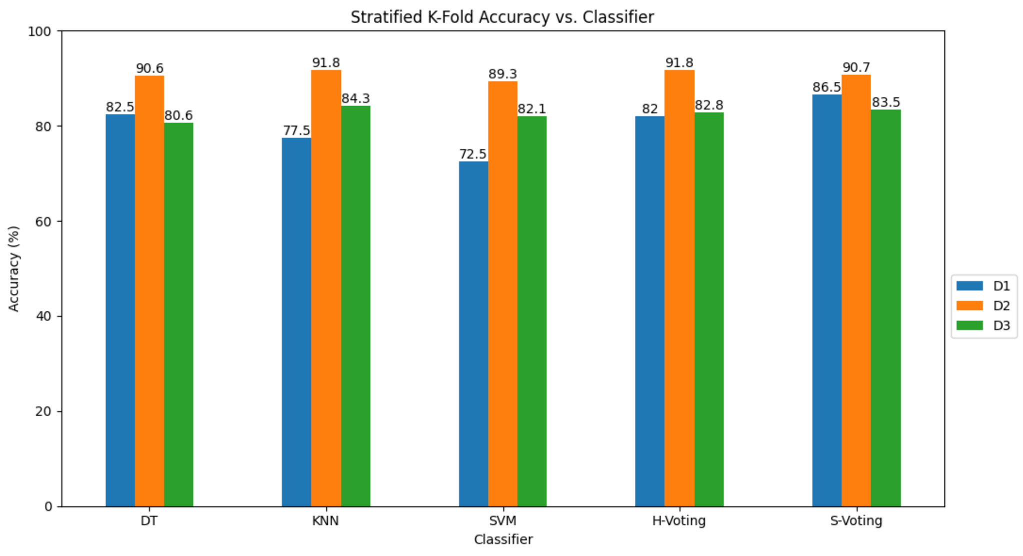


Figure 4.19. Results of 10-fold CV without stratified test fold

over 5CV since 10CV uses a greater proportion of the data for training. However, this approach proved unsustainable for training the LSTM model which was unable to be trained over 10 folds in a practical time period. Additionally, further experiments suggested that there was no noticeable drop in performance when using 5CV instead of 10CV as long as stratified test folds were used, suggesting that balancing the classes partially offset the effect of reduced training data.

Chapter 5

Results Analysis

As outlined in section 3.2.2, the primary focus of this project was to explore whether ensemble learning and deep learning techniques could be used to improve upon the performance of classical machine learning models in the context of remote laser welding. As a result, classical, ensemble and deep learning models were tested on the same datasets used in the initial benchmark study, defined in Table 3.2a.

5.1 Data Preparation

It is essential to note that the results of the benchmark classifiers, shown in Table 3.3 were produced using augmented data for dataset C, whereas experiments for this project were primarily performed on the un-augmented version of dataset C. For the purpose of simplicity, the datasets will be redefined as follows:

- Dataset 1 (D1) which corresponds to the original dataset A.

- Dataset 2 (D2) which corresponds to the original dataset C, before data augmentation.
- Dataset 3 (D3) which corresponds to the original generalised dataset $A \cup B \cup C$.

Table 5.1. Class breakdown of the primary RLW datasets.

	D1	**D2**	**D3**
Sound Weld	16	9	33
Lack of Connection	12	55	73
Overpenetration	18	22	40
Total Datapoints	46	86	142

In order to ensure tests were standardised, the five folds used to validate models were generated in advance such that all models are trained and tested on the exact same data points, in the exact same order. Additionally, for the majority of experiments, a standard scaler was applied to the data such that, for a given feature, all values across the feature have a mean of 0 and a standard deviation of 1. While this had a slight negative impact on the performance of some models, the majority of models were found to have better accuracies using scaled features and certain models such as SVMs and deep learning models performed significantly worse in the absence of scaled features. Hence, unless stated otherwise, it can be assumed results were generated using scaled features.

Table 5.2. Results of the three primary classical models and their corresponding voting classifiers using unscaled features.

Model (Unscaled)	**D1**	**D2**	**D3**
Decision Tree	82.7	89.5	83.2
KNN	74.0	**95.4**	85.9
SVM	67.3	88.4	78.2
HV	82.4	93.0	**86.0**
SV	**88.9**	91.8	84.6

5.2 Classical and Ensemble Results

During the initial planning of the project, it was determined that the Decision Tree, KNN and SVM classifiers would be the primary constituent models used to construct ensemble voting classifiers. Towards the later stages of the project, the remaining classical models from the benchmark study were included as an extension and further voting classifiers were constructed by enumerating all 3-combinations of classical machine learning, with the ten best-performing models displayed in the results.

5.2.1 Unscaled Feature Data

Table 5.2 displays the results of the three primary classical models using unscaled features which reflect the general trend across datasets observed in the existing literature: models perform worst on the smallest dataset (D1) and best on D2 which has more examples and homogeneous data. Interestingly, the three classical models still perform better on D2 than on the generalised dataset (D3) even in the absence of data augmentation. This suggests that the strong performance of benchmark models on the augmented dataset C can partly be attributed to its homogeneous dataset and hence cannot fully be attributed to information leakage from augmentation. Hence, this reinforces the idea that the generalised dataset can be used to test model generalisation since models tend to produce lower accuracy when classifying data from different distributions.

Crucially, however, the soft voting model outperforms all other models on dataset 1 with an accuracy of 88.9%, despite having no predictive power beyond aggregating predicted class probabilities produced by unoptimised KNN, SVM and decision tree classifiers. This is a significant result as it supports the key hypothesis that diverse models make diverse mistakes and the soft-voting model is hence able to mitigate mistakes made by the classical models using information collected across the ensemble. Additionally, the HV model also performs significantly better than the KNN and SVM models despite requiring at least one of the two to produce a correct prediction, suggesting that the KNN and SVM misclassifications do not completely overlap.

The strength of the SV model on dataset 1 can partially be explained by examining the predicted probabilities of the decision tree model. Since the decision tree has a high number of parameters and hence high model complexity, it unanimously predicts its class with a probability of 1, regardless of whether or not the prediction is accurate. Hence, if the decision tree model is accurate, then the KNN and SVM models must be significantly wrong to sway the output of the SV model.

This trend is further reinforced by considering datasets 2 and 3 where the decision tree model is no longer the best-performing constituent model and in correspondence, the SV model performs comparatively less well with the HV classifier producing the higher accuracy of the two ensembles. This is intuitive as the hard-voting mechanism of the HV classifier allows it to completely ignore the decision tree when it misclassifies a data point with 100% confidence as long as the KNN and SVM models “agree” on the correct class.

However, it is useful to note that the SV model outperformed the decision tree model and hence was able to correctly classify points that the decision tree failed to predict. This is significant because the decision tree predicts classes with 100% confidence, implying that there were some data points where the SV model used probabilities from the KNN and SVM classifiers were able to mitigate a major error by the decision tree.

Notably, the decision tree model performs much better on dataset 1 compared to the benchmark decision tree on dataset A; this can be attributed to the fact that the benchmark model had a restricted maximum depth of 3, whereas as the model used for this project had no depth restriction.

5.2.2 Scaled Feature Data

Table 5.3 displays the results of the same primary classifiers as well as the two voting models using the scaled feature data. By comparing these results with the unscaled feature results, it is immediately apparent that the SVM model performs significantly better using scaled features, while the KNN model is relatively insensitive to feature scaling and the decision tree performs identically, irrespective of scaling. This is further reinforced by the fact that the SVM model produces an accuracy of 76% on dataset 1 with scaled features despite producing an accuracy of just 54.3% in the benchmark experiment. With a few exceptions, results across

Table 5.3. Results of the three primary classical models and their corresponding voting classifiers using scaled features.

Model (Scaled)	**D1**	**D2**	**D3**
Decision Tree	82.7	89.5	83.2
KNN	78.0	**93.0**	**86.0**
SVM	76.0	**93.0**	79.6
HV	86.9	91.9	84.6
SV	**93.5**	91.9	83.8

all models shows either improved or equivalent results after feature scaling and hence all subsequent experiments only consider scaled features.

The boosted performance of the SVM model appears to have an indirect effect on the performance of the SV model on dataset 1, which now produces an accuracy of 93.5$\%$ despite no significant changes in the results of the decision tree model. This can be explained by inspecting the probabilities produced by the SVM model which suggest that it is more likely than the KNN model to produce higher probabilities for the correct class even when making producing a misclassification. On the other hand, the 3-nearest neighbour model will only ever produces probabilities of 0, $\frac{1}{3}$, $\frac{2}{3}$, and 1, depending on which discrete classes appear as the 3-nearest neighbours, making it difficult to analyse how its probabilities affect the SV model.

Upon comparison with the benchmark results, it is clear that the soft voting classifier outperforms the previous best-in-class classifier on dataset 1, which was the random forest model (87%) despite only aggregating the probabilities produced by models that all performed substantially worse than the random forest model. Another point of note is that D1 was defined as the dataset with a significant feature overlap which suggests that voting ensemble classifiers have a strong potential to mitigate mistakes caused by individual models by when the decision boundaries are not clearly defined since they aggregate diverse algorithms which produce different types of decision boundaries. Given that random forest classifiers, themselves are bagging ensemble classifiers, this strongly supports the idea ensemble learning can be used to mitigate the effects of the small datasets problem in RLW.

5.3 Best Ensemble Classifiers

Table 5.4 shows the performance of all classical and non-voting ensemble models, while Tables 5.5, 5.6 and 5.7 show the results of the ten best ensemble models after enumerating all 3-combinations of the classical and non-voting ensemble models. In accordance with the benchmark tests, random forest performs the best out of the individual algorithms while the Gaussian process model shows marginal improvements on dataset 3.

The results from the best voting models on datasets 2 and 3 show that there are various ensemble combinations that perform on par with the best individual models, despite no dramatic improvements in accuracy. Additionally, it is useful to note that a disproportionate number of the best ensemble classifiers for D3 are hard-voting classifiers, suggesting that probabilistic estimates are less reliable when the data does not come from a homogeneous distribution. In general, the ensemble classifiers do not perform as well as the neural networks on the generalised dataset, which further motivates the research of deep learning classifiers in conjunction

Table 5.4. Classical machine learning percentage accuracies for datasets: D1 (A); D2 (C); D3 ($A \cup B \cup C$).

Classical Model	**D1**	**D2**	**D3**
Decision Tree	82.6	89.5	83.1
KNN	78.0	**93.0**	86.0
SVM	76.1	**93.0**	79.6
Gaussian Process	87.0	88.4	**86.6**
Random Forest	**89.1**	89.5	83.1
AdaBoost	76.1	89.5	76.8
Naive Bayes	71.7	88.4	76.8
QDA	65.2	87.2	80.3
Gradient Boosting	80.4	89.5	83.8

Table 5.5. The top ten performing voting classifiers on D1 (A).

Voting Models (D1)	**Total Accuracy**
SV(Decision Tree | KNN | RBF SVM)	**93.5**
HV(Decision Tree | RBF SVM | Gaussian Process)	**93.5**
SV(Decision Tree | KNN | Gaussian Process)	89.1
HV(Decision Tree | Gaussian Process | Random Forest)	89.1
SV(KNN | RBF SVM | Gradient Boosting)	89.1
HV(KNN | Random Forest | Naive Bayes)	89.1
SV(KNN | Random Forest | Gradient Boosting)	89.1
HV(RBF SVM | Gaussian Process | Random Forest)	89.1
SV(RBF SVM | Gaussian Process | Random Forest)	89.1
HV(RBF SVM | Gaussian Process | Gradient Boosting)	89.1

Table 5.6. The top ten performing voting classifiers on D2 (C).

Voting Model (D2)	**Total Accuracy**
SV(KNN | RBF SVM | QDA)	**94.2**
SV(KNN | Naive Bayes | QDA)	**94.2**
SV(Decision Tree | KNN | Naive Bayes)	93.0
HV(Decision Tree | KNN |QDA)	93.0
SV(Decision Tree | KNN | QDA)	93.0
HV(Decision Tree | RBF SVM | Random Forest)	93.0
SV(KNN | RBF SVM | Gaussian Process)	93.0
HV(KNN | RBF SVM | QDA)	93.0
SV(KNN | RBF SVM | Gradient Boosting)	93.0
SV(KNN | AdaBoost | Naive Bayes)	93.0

Table 5.7. The top ten performing voting classifiers on D3 ($A \cup B \cup C$).

Voting Model	Total Accuracy
HV(Decision Tree | KNN | QDA)	**87.3**
HV(KNN | Gaussian Process | QDA)	**87.3**
HV(KNN | Gaussian Process |Gradient Boosting)	**87.3**
SV(KNN | Random Forest | Gradient Boosting)	**87.3**
HV(KNN | QDA | Gradient Boosting)	**87.3**
HV(Decision Tree | KNN | Gaussian Process)	86.6
HV(Decision Tree | KNN | Naive Bayes)	86.6
HV(Decision Tree | Gaussian Process | QDA)	86.6
HV(KNN | Gaussian Process | Naive Bayes)	86.6
SV(KNN | Gaussian Process | QDA)	86.6

with ensemble classifiers for RLW classification.

5.4 Deep Learning Classifiers

Table 5.8 displays the results of the deep learning classifiers; the observation here is that the FCNN and CNN models outperform all classical and ensemble models on the generalised dataset, with the CNN model producing a marginally higher accuracy than the best-in-class neural network model (92.7%).

Noticeably, the LSTM model performed relatively poorly despite being purposefully designed for classifying sequenced data. This could simply be because the LSTM model is not suited for learning patterns across thousands of time steps since it analyses every value at every step, each time deciding how much of its long

Table 5.8. Deep learning results for datasets: D1 (A); D2 (C); D3 ($A \cup B \cup C$)

Deep Model	**D1**	**D2**	**D3**
FCNN	**89.1**	**93.0**	91.5
CNN	82.6	**93.0**	**93.0**
LSTM	82.6	83.7	76.1

and short-term memory it should update, rather than analysing the general structure of the signals. Additionally, photodiode signal sequences are not necessarily similar to other types of sequenced data such as natural language; for example, when analysing a sentence, a single word can dramatically alter the meaning of the entire sentence whereas signal values may only convey useful information when analysed across a wide range of values.

This may partially explain the strong performance of the CNN model which uses the kernel to learn information not just about several time steps at a time but also several signal channels because it aggregates information from the temperature, plasma and back reflection signal whenever it performs a convolution.

5.5 Data-Augmentation Results

In order for a complete comparison with the existing literature, it was useful to enumerate the best voting models on the augmented version of dataset C. While several ensemble classifiers produced results comparable to the best-in-class neural

Table 5.9. The ten best voting ensemble classifiers on the augmented version of dataset C.

Voting Model	**Total Accuracy**
SV(Gaussian Process | Naive Bayes | QDA)	**98.4**
HV(Decision Tree | KNN | QDA)	97.5
SV(Decision Tree | KNN | QDA)	97.5
HV(Decision Tree | Gaussian Process | QDA)	97.5
SV(Decision Tree | Gaussian Process | QDA)	97.5
HV(Decision Tree | Naive Bayes | QDA)	97.5
SV(KNN | RBF SVM | QDA)	97.5
SV(KNN | Random Forest | QDA)	97.5
SV(KNN | AdaBoost | QDA)	97.5
SV(RBF SVM | Gaussian Process | QDA)	97.5

network model, a soft-voting model comprised of a Gaussian Process, Naive Bayes and QDA was able to produce a slightly higher accuracy than the 97.5% achieved by the neural network during the benchmark experiments. While further analysis is required to understand the effectiveness of this ensemble, it is interesting to note that all three of these constituents models are probabilistic by nature, whereas the best ensemble models typically feature several non-probabilistic models such as decision trees, KNN and random forests, potentially suggesting that probabilistic are more sensitive to the information leakage produced via augmentation.

One final point of note concerning augmentation is that the FCNN model was the only model that, when applied to datasets in which every class had been augmented, achieved an accuracy of 100% on every dataset, highlighting the effect of information leakage and the FCNNs ability to detect this leakage.

Chapter 6

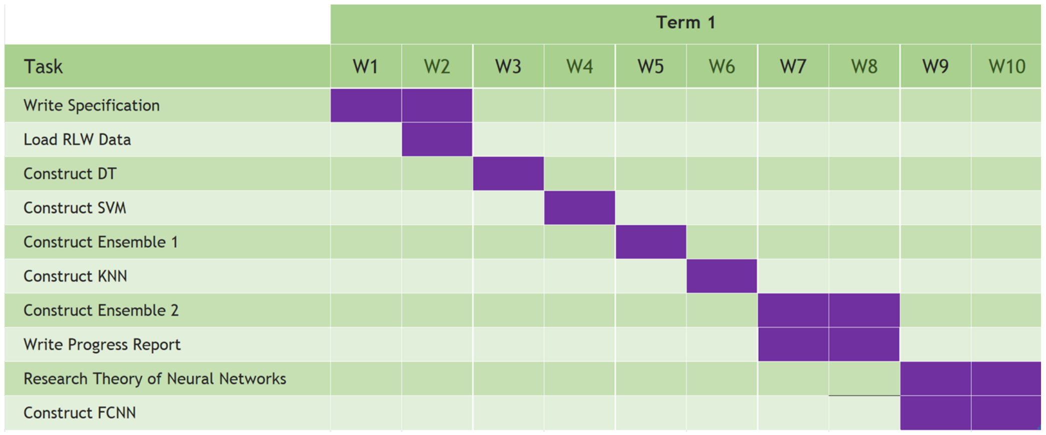
Project Evaluation

6.1 Project Management

Since the project was research-based, an agile development approach was chosen since it was not possible to estimate in advance which models, algorithms and validation strategies would be most effective for the RLW data. Additionally, the inclusion of several non-computing elements with regard to the remote-laser welding manufacturing process and the wider zero-defect ideology reinforced the need for an adaptive, incremental approach through which understanding of the problem domain could be progressively built up over time.

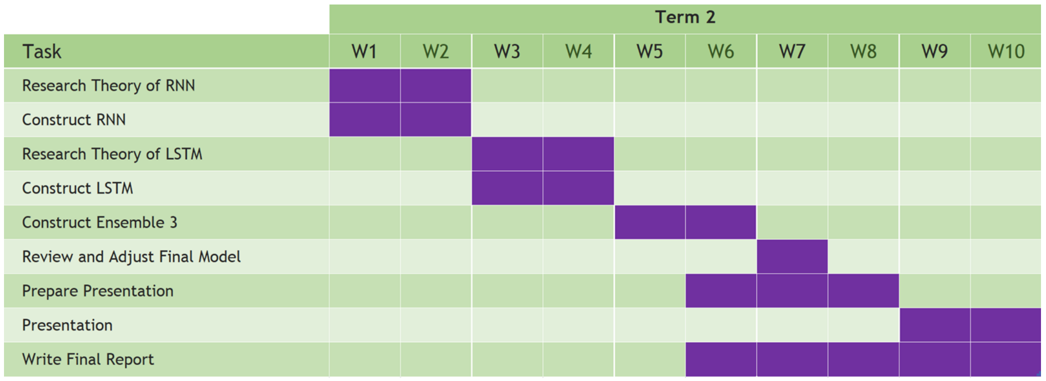
As a result, only a simple plan was outlined at the start of the project, shown in Figures 6.1 and 6.2 with the aim of using just three classical machine learning algorithms: KNN, SVM and decision tree. After this, the main idea was to create simple soft and hard-voting ensemble classifiers before finally creating the FCNN and LSTM models.

This approach proved effective as it helped lay the framework for developing



	Term 1									
Task	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Write Specification	🟪	🟪								
Load RLW Data		🟪								
Construct DT			🟪							
Construct SVM				🟪						
Construct Ensemble 1					🟪					
Construct KNN						🟪				
Construct Ensemble 2							🟪	🟪		
Write Progress Report							🟪	🟪		
Research Theory of Neural Networks									🟪	🟪
Construct FCNN									🟪	🟪

Figure 6.1. Term 1 Gantt chart.



	Term 2									
Task	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Research Theory of RNN	🟪	🟪								
Construct RNN	🟪	🟪								
Research Theory of LSTM			🟪	🟪						
Construct LSTM			🟪	🟪						
Construct Ensemble 3					🟪	🟪				
Review and Adjust Final Model							🟪			
Prepare Presentation						🟪	🟪	🟪		
Presentation									🟪	🟪
Write Final Report						🟪	🟪	🟪	🟪	🟪

Figure 6.2. Term 2 Gantt chart.

the general tools used by all models, such as the classical and deep data loggers used for data preprocessing, the cross-validation functions used for comparing models performance and the closure functions used to generate neural network architectures. Since these functions were designed as black boxes, newly created models could be immediately tested upon completion, and identifying the best ensemble classifier on a given dataset could be easily automated.

In general, a natural justification for the agile approach was that machine learning problems are naturally extendable once the primary objectives are complete, with the flexible schedule leaving time to incorporate additional features such as data augmentation and additional classifiers. This approach proved to be effective with regard to time management as all scheduled tasks were completed on or ahead of schedule, leaving plenty of time to address unexpected challenges, such as the ineffectiveness of the LSTM model. For example, once it was clear that the LSTM wasn't suited for RLW classification, the CNN model was introduced which ended up performing better on the generalised data than all ensemble, deep and best-in-class models despite it not being under consideration until the final few weeks of the project.

However, the effectiveness of the Agile approach came at the cost of poor code design and modularity. Because functions and architectures were designed when they were needed, much of the code base is largely redundant with several instances of poorly documented, duplicate code. However, upon reflection, neglecting the design of the code was not costly in the long term as the primary challenge of the project revolved around the technical understanding of various machine learning concepts, such as identifying which types of algorithms and architectures were best suited for RLW data; this proved to be a much greater bottleneck than code refactoring. In particular, for smaller code snippets, it was often more practical to partially rewrite duplicate code instead of modularising common tools and functions.

6.2 Project Reflection

Overall, the project can be considered a partial success since a classifier was identified for every dataset that improved upon the performance of existing benchmarked models. In particular, key questions were answered with regard to the primary RLW classification challenges outlined in 3: the voting ensemble classifiers demonstrated the ability to effectively aggregated weaker constituent models with unoptimised hyperparameters in order to correctly predict data points that no individual model could classify.

Additionally, the deep learning models, while not necessarily as effective as ensemble models on the homogeneous datasets, showed significant potential to generalise and learn patterns in time-series data drawn from different distributions.

One limitation of the project was the inability to produce consistent deep-learning results. While all training folds were standardised after an initial random shuffle, weights and biases were randomly initialised using standard initialisation strategies, including LeCun initialisation or He Normal initialisation, which introduced an element of randomness into the results. This did not have a significant effect on the main conclusions of the project, as the deep learning models tended to always perform best on the generalised data sets but it did make it substantially harder to identify which neural network hyperparameters to use. For example, section 4.3 discusses the general architecture of the FCNN and CNN networks but omits any significant justification on why a certain number of hidden layers, neurons or kernels were used. This was because a lack of validation data, as well as random deviations in weight initialisations, made it harder to identify which of these hyperparameters were responsible for variations in the results. Hence, the primary justification for the final neural network architectures was that these architectures were determined to be complex enough to learn the general pattern of the training, without manually selecting hyperparameters that would tune the models to the specific datasets used in this project.

6.3 Tools and Libraries

All code written for the project was written using Python 3.9; classical machine learning models were created primarily using sci-kit learn (version 1.2.0) and deep learning models were created using keras (version 2.11.0).

6.4 Ethical Concerns

There are no legal, social, ethical or professional issues. The data used for this project is freely accessible online.[34,35]

Chapter 7

Conclusions and Future Work

The primary purpose of this report was to investigate whether ensemble learning and deep learning techniques could be used to successfully classify welding defects produced via the RLW of copper-to-steel foils. In particular, the objective was to determine whether or not ensemble classifiers and deep learning classifiers could improve upon the classification accuracy produced by the best-in-class benchmark classifiers. The main findings of the project are as follows:

- Results from the ensemble voting classifiers demonstrated that classification accuracy can be improved by aggregating the predictions of several constituent classifiers. In particular, a soft-voting ensemble comprised of decision tree, KNN and SVM classifiers was able to achieve a classification accuracy of 93.5% on dataset 1 (A), outperforming the previous best random forest model which produced an accuracy of 87%. This strongly reinforced the hypothesis that diversity in classification algorithms correlates with diversity in misclassifications, allowing voting ensemble models to achieve higher classification accuracies than all of their constituent models.

The concept was further reinforced by the results of the best ensemble model on the augmented version of dataset C, where a soft-voting ensemble of Gaussian process, naive Bayes, and QDA achieved an accuracy of 98.4%, marginally higher than the SLP neural network which produced the best benchmark accuracy of 97.5%.

- While the ensemble classifiers outperformed the deep learning classifiers on datasets 1 and 2, the deep learning models performed substantially better on the generalised dataset, using time-series data. In particular, the 1D convolutional neural network achieved the highest accuracy (93.0%), marginally higher than the best benchmark model (SLP), which achieved a classification accuracy of 92.7%.

  This suggests that the problem of small RLW datasets problem could be partially mitigated by compiling larger datasets using combinations of welding data produced by different welding configurations and different manufacturing groups since the deep learning models showed a strong potential for generalisation but were unable to have their many hyperparameters optimised due to a lack of data.

- Model performance was highly sensitive to class imbalance and feature scaling; the classification results of the SVM model were substantially increased by using stratified test folds and by scaling the feature data by their means and then normalising them to have a standard deviation of 1. This result was particularly important as the SVM model was a constituent of the best-performing soft-voting model for dataset 1, with feature scaling alone producing a 4.6% increase in classification accuracy for the soft-voting ensemble.

Opportunities for future work include creating voting models that combine the predictions of classical and deep learning models; at present, no ensemble models use deep learning models as constituents. This feature was primarily omitted because separate data loggers were implemented to manually process the statistical feature data and the time-series data. Additionally, separate manual cross-validation functions were implemented for evaluating classical and deep models because the classical models use an Sklearn API to fit training data whereas the deep models use a Keras API. The primary benefit of streamlining these functions would be to test whether the deep learning models can introduce greater generalising power to the ensemble models, given that the current best ensemble models were outperformed on the generalised dataset (D3) by several of the benchmark models.

Another useful experiment would be to test the deep learning models on the statistical feature data. One of the key conclusions of the report was that the deep learning models generalise best to welding data from different distributions, but it is not immediately clear whether this was because this is an intrinsic property of the deep learning architectures or because the time-series feature data offered a significant amount of useful information about the RLW signals that the statistical data did.

Opportunities also exist with regard to data augmentation. At present, the augmentation techniques used to create synthetic sound weld data points have the potential to introduce information leakage since data is augmented before the test folds are produced, resulting in some of the test data points being generated directly from data points in the training set. An alternative approach for augmentation could involve creating the test folds first and then augmenting only the training data. This would allow models to be trained on a greater number of training examples without leaking information into the test set.

References

[1] Daryl Powell, Maria Chiara Magnanini, Marcello Colledani, and Odd Myklebust. Advancing zero defect manufacturing: A state-of-the-art perspective and future research directions. *Computers in Industry*, 136:103596, 2022. ISSN 0166-3615. doi: https://doi.org/10.1016/j.compind.2021.103596. URL https://www.sciencedirect.com/science/article/pii/S0166361521002037.

[2] Pasquale Franciosa, Mikhail Sokolov, Sumit Sinha, Tianzhu Sun, and Dariusz Ceglarek. Deep learning enhanced digital twin for closed-loop in-process quality improvement. *CIRP Annals*, 69(1):369–372, 2020. ISSN 0007-8506. doi: https://doi.org/10.1016/j.cirp.2020.04.110. URL https://www.sciencedirect.com/science/article/pii/S0007850620301323.

[3] Isabella Burch and Jock Gilchrist. Survey of global activity to phase out internal combustion engine vehicles. *Center of Climate Protection: Santa Rosa, CA, USA*, 2018.

[4] T Sun, P Franciosa, M Sokolov, and D Ceglarek. Challenges and opportunities in laser welding of 6xxx high strength aluminium extrusions in automotive battery tray construction. *Procedia CIRP*, 94:565–570, 2020.

[5] Giovanni Chianese, Pasquale Franciosa, Jonas Nolte, Darek Ceglarek, and Stanislao Patalano. Characterization of Photodiodes for Detection of Variations in Part-to-Part Gap and Weld Penetration Depth During Remote Laser Welding of Copper-to-Steel Battery Tab Connectors. *Journal of Manufacturing Science and Engineering*, 144(7), 12 2021. ISSN 1087-1357. doi: 10.1115/1.4052725. URL https://doi.org/10.1115/1.4052725. 071004.

[6] Tianzhu Sun, Pasquale Franciosa, and Mikhail Sokolov. Challenges and opportunities in laser welding of 6xxx high strength aluminium extrusions in automotive battery tray construction. *Procedia CIRP*, 94:565–570, 01 2020. doi: 10.1016/j.procir.2020.09.076.

[7] Shuai Li, Honggang Dong, Xingxing Wang, Zhongying Liu, Zhaojun Tan, Linjian Shangguan, Quanbin Lu, and Sujuan Zhong. Effect of repair welding on microstructure and mechanical properties of 7n01 aluminum alloy mig welded joint. *Journal of Manufacturing Processes*, 54:80–88, 2020.

[8] Dariusz Ceglarek, Marcello Colledani, József Váncza, Duck-Young Kim, Charles Marine, Markus Kogel-Hollacher, Anil Mistry, and Luca Bolognese. Rapid deployment of remote laser welding processes in automotive assembly systems. *CIRP Annals*, 64(1):389–394, 2015.

[9] Gábor Erdos, Zsolt Kemény, Andras Kovacs, and J. Váncza. Planning of remote laser welding processes. *Procedia CIRP*, 7:222–227, 12 2013. doi: 10.1016/j.procir.2013.05.038.

[10] Eriel Pérez Zapico, Alessandro Ascari, Vincenzo Dimatteo, and Alessandro

Fortunato. Laser dissimilar welding of copper and steel thin sheets for battery production. *Journal of Laser Applications*, 33(1), 12 2020. ISSN 1042-346X. doi: 10.2351/7.0000309. URL https://doi.org/10.2351/7.0000309. 012016.

[11] Xiangdong Gao, Zhuman Li, Lin Wang, Xiaohu Zhou, Deyong You, and Perry P. Gao. Detection of weld imperfection in high-power disk laser welding based on association analysis of multi-sensing features. *Optics Laser Technology*, 115:306–315, 2019. ISSN 0030-3992. doi: https://doi.org/10.1016/j.optlastec.2019.01.053. URL https://www.sciencedirect.com/science/article/pii/S0030399218314932.

[12] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315, 2016.

[13] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833, 1995. doi: 10.1021/ci00027a006. URL https://doi.org/10.1021/ci00027a006.

[14] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[16] Brady Neal. On the bias-variance tradeoff: Textbooks need an update, 2019.

[17] Giovanni Chianese, Pasquale Franciosa, Tianzhu Sun, Dariusz Ceglarek, and Stanislao Patalano. Using photodiodes and supervised machine learning for automatic classification of weld defects in laser welding of thin foils copper-to-steel battery tabs. *Journal of Laser Applications*, 34(4), 11 2022. ISSN 1042-346X. doi: 10.2351/7.0000800. URL https://doi.org/10.2351/7.0000800. 042040.

[18] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era, 2017.

[19] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Springer texts in statistics. Springer, New York, NY, 1 edition, June 2013.

[20] Stuart Russell, Stuart Jonathan Russell, Peter Norvig, and Ernest Davis. *Artificial Intelligence*, pages 675–683. 10 2022. ISBN 9780136042594.

[21] Aurelien Geron. *Hands-on machine learning with scikit-learn, keras, and TensorFlow*. O'Reilly Media, Sebastopol, CA, 2 edition, October 2019.

[22] L. Breiman, Jerome H. Friedman, Richard A. Olshen, and C. J. Stone. Classification and regression trees. 1984.

[23] Stuart Russell, Stuart Jonathan Russell, Peter Norvig, and Ernest Davis. *Artificial Intelligence*, pages 705–710. 10 2022. ISBN 9780136042594.

[24] Andrew Glassner. *Deep Learning*, pages 282–290. 06 2021. ISBN 9781718500723.

[25] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[26] Bernhard Schölkopf, Christopher JC Burges, Alexander J Smola, et al. *Advances in kernel methods: support vector learning*. MIT press, 1999.

[27] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.

[28] Bradley Efron. Bayes' theorem in the 21st century. *Science*, 340(6137): 1177–1178, 2013.

[29] Harry Zhang. The optimality of naive bayes. *Aa*, 1(2):3, 2004.

[30] Alaa Tharwat. Linear vs. quadratic discriminant analysis classifier: a tutorial. *International Journal of Applied Pattern Recognition*, 3(2):145–180, 2016.

[31] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49195-8.

[32] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[33] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[34] Giovanni Chianese, Pasquale Franciosa, Jonas Nolte, Darek Ceglarek, and Stanislao Patalano. Characterization of photodiodes for detection of variations in part-to-part gap and weld penetration depth during remote laser welding of copper-to-steel battery tab connectors, July 2021. URL https://doi.org/10.5281/zenodo.5115087.

[35] Franciosa P. Sun T. Ceglarek D. Patalano S Chianese, G. Using Photodiodes and Supervised Machine Learning for Automatic Classification of Weld Defects in Laser Welding of Thin Foils Copper-to-Steel Battery Tabs, June 2022. URL https://doi.org/10.5281/zenodo.6732794.