Development of a machine learning model for classifying steel plate defects

Samuel O'Neill
University of Portsmouth
UP879282

***Abstract*—This project aims to investigate machine learning algorithms to assist the development of a predictive model capable of classifying surface defects in steel plates. The intention is for this classification model to be used in a condition-based maintenance (CBM) system to improve the efficiency of maintenance decisions.**

***Keywords*—condition-based maintenance, predictive maintenance, machine learning, artificial intelligence, data science**

I. Introduction

Condition-based maintenance (CBM) is a strategy used to ensure that maintenance of a system is only performed when needed, with the intention of reducing maintenance costs and increasing efficiency. Typically, CBM uses artificial intelligence (AI) to monitor a system and determine the best strategy by classifying faults. This report documents the development of an AI-based algorithm for classifying surface defects in steel plates.

II. Literature Review

This literature review aims to analyse existing literature on the topics of AI and CBM in order to assist the development of a CBM system. It will investigate previously used techniques to solve the task of using AI in classifying steel plate defects.

CBM is a predictive maintenance (PdM) strategy. According to Ellis [1], the main objective of CBM is to ensure that an asset can fulfil its purpose in the most cost-effective manner possible. CBM allows just-in-time replacement of components to maximise their lifetime [2], while also minimising inspection/repair costs by analysing data representing the condition of a system's critical assets [3]. The strategy is particularly useful in environments where downtime is extremely costly, such as manufacturing plants. An effective approach to classification is to develop an ML model capable of detecting and classifying faults based on previous training data. According to DataRobot [4], the three primary steps in ML classification are:

1. Apply an algorithm which identifies shared characteristics of classes
2. Compare characteristics to data to be classified
3. Estimate likelihood of data belonging to a certain class

. Susto et al. [5] proposed a multiple classifier (MC) methodology for PdM, suited for high-dimensional problems where there is a large amount of variables in play. The core idea of this method is to train multiple models to provide different performance trade-offs. This methodology can deal with unbalanced, high-dimension classification problems. The concept can be applied using '*k*' different classifiers in parallel, where each faces a different problem, resulting in different maintenance management outcomes.

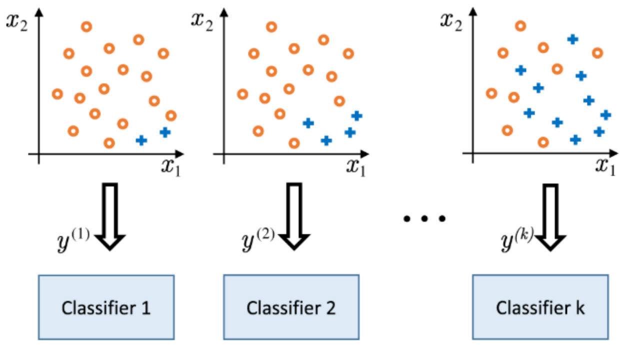


Figure 1: an example of a two-dimensional application of the MC methodology

The authors note that the performance of the methodology increases with the number of classifiers, '*k*', however '*k*' is constrained by available computational capabilities and the chosen algorithm. They concluded that the methodology guarantees improved maintenance decisions, while allowing users to dynamically change policies based on current needs.

The task of classifying steel plate defects has been previously attempted by several other researchers, each applying their own methodologies. Wu [6], approached the problem using SVM, and was able to obtain an initial prediction accuracy of 79%. He applied a non-linear Gaussian RBF kernel SVM model (see fig. 3) and tuned the hyperparameters C and Gamma, where C is the cost of misclassification, and Gamma is the kernel parameter. See table 1 for a breakdown of these parameters [7]. After tuning, Wu was able to achieve significantly improved prediction accuracy, up to 92

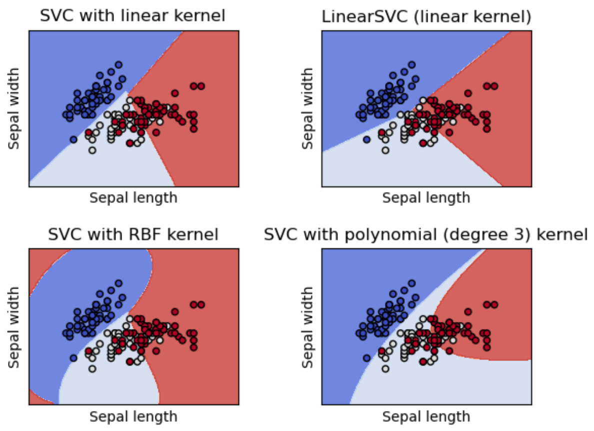
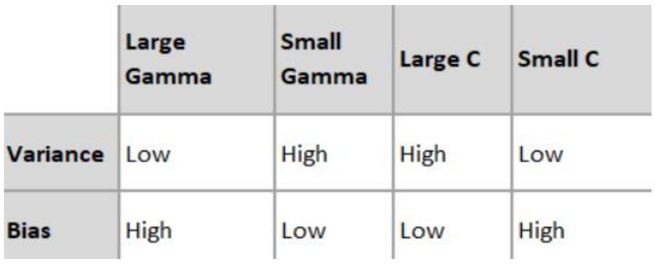


Figure 2: SVC boundaries using different kernel types



	Large Gamma	Small Gamma	Large C	Small C
Variance	Low	High	High	Low
Bias	High	Low	Low	High

Table 1: Effect of hyperparameters on SVM. (The aim of tuning SVM is to find a model with optimum variance and bias)

Lowe [8] addressed the same problem with a more investigative approach, comparing 7 ML models. From initial modelling, he achieved an average accuracy of 73.99%, with the gradient boosting algorithm (GBM) scoring highest.

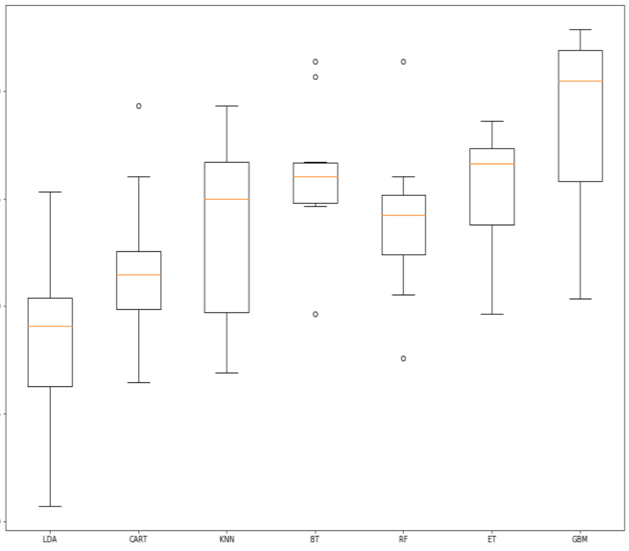


Figure 3: accuracy score comparison of 7 common models

Lowe then tuned the two best performing algorithms (GBM and bagged decision trees (BDT)), but still found GBM to be the optimal model. His finalised model (tuned GBM) achieved 78.8% accuracy.

Some say that accuracy score should not be used as a measure of comparing models as it can be misleading. Brownlee attributes this to the fact that differences in mean performance may be caused by a statistical fluke [9]. Instead, model performance should be assessed using statistical hypothesis tests, such as Dietterich's '5x2-fold cross-validation test' [10]. Brownlee also proposes a series of other statistical hypothesis tests, detailed in table 2 [11].

Type of test	Test name	What it checks
Normality	Shapiro-Wilk	Does a sample have Gaussian distribution?
	D'Agostino K^2	
	Anderson-Darling	
Correlation	Pearson's coefficient	Do two samples have linear relationship?
	Spearman's rank	Do two samples have monotonic relationship?
	Kendall's rank	
	Chi-squared	Are two variables related or independent?
Parametric	Student's t-test	Are the means of two independent samples significantly different
	Paired STT	Are the means of two paired samples significantly different
	Analysis of Variance (ANOVA)	Same as STT for 2+ samples
	Repeated measures ANOVA	Same as paired STT for 2+ samples
Nonparametric	Mann-Whitney U	Are the distributions of two independent samples equal
	Wilcoxon Signed-rank	Are the distributions of two paired samples equal
	Kruskal-Wallis H	Same as MWU for 2+ samples
	Friedman	Same as WSR for 2+ samples

Table 2:15 statistical hypothesis tests

Finally, Brownlee also provides a rough guide to increasing a model's performance [12]. Contrary to Wu and Lowe's approaches, Brownlee states that hyperparameter tuning is not actually the best method. He states 4 ways that performance can be improved (in order of effectiveness). Improve using…

1) …data (cleaner/transformed data)

2) …algorithms (test multiple algorithms/configurations)

3) …hyperparameter tuning

4) …ensembles (combine well-performing models)

This concludes a brief investigation into existing literature around the classification problem and how it should be approached. Work by Wu and Lowe demonstrates some provenly effective solutions, while Brownlee provides useful information to assist in selecting a final model.

III. Model Development

This section will detail the development of the classification model. This framework was used for development (based on similar frameworks by Lowe [8] and MathWorks [13]):

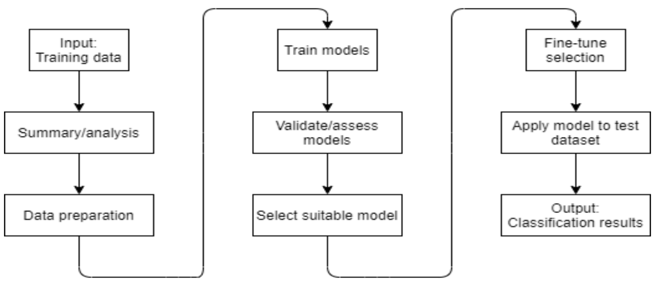


Figure 4: model development framework

A. Data summary/analysis

Visualisation of the data can help to provide some insight into which algorithms should be applied to the problem. *Figure* 5 shows histograms of each predictor.

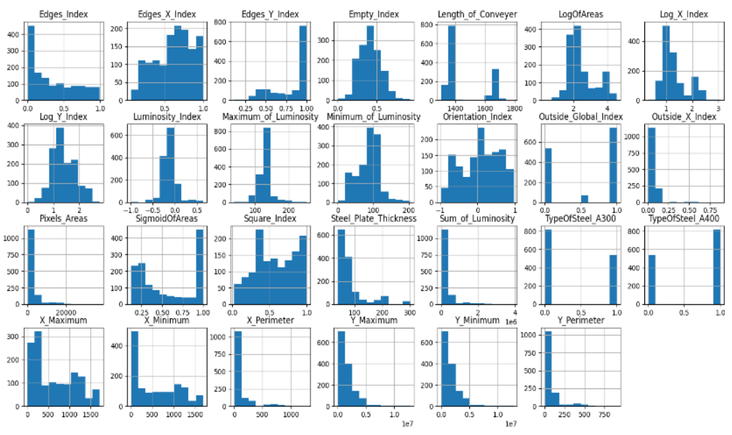


Figure 5: histograms of the dataset's 27 predictors

Most of the predictors are multimodal and normally distributed, though some are skewed or contain outliers. These outliers are shown more clearly in the box-plots in fig. 6.

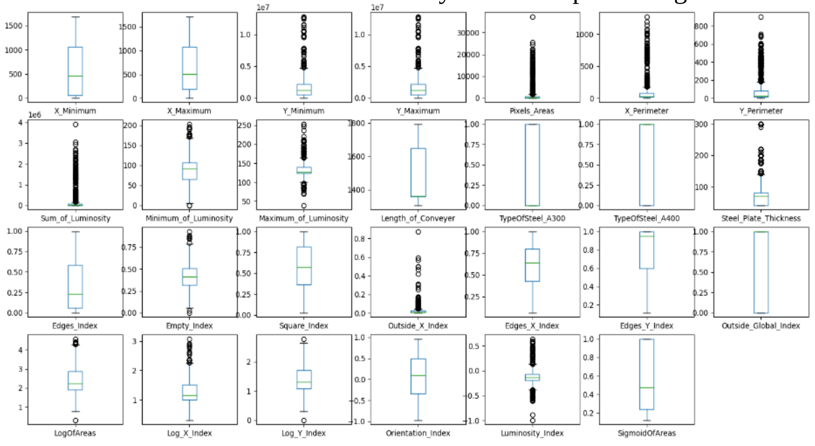


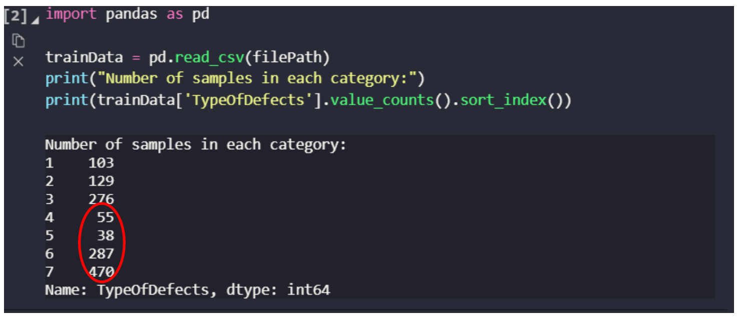
Figure 6: box-plots of the dataset's 27 predictors

B. *Models and predictions*

The models that will be tested in this project are: k-Nearest Neighbour (kNN), Support Vector Machines (SVM), decision trees/forests, Naïve Bayes (NB) and Gradient Boosting (GBM). Each model is expected to perform differently due to the nature of the predictors. For example, the presence of outliers can affect the kNN classifier, and NB usually only performs well if conditional independence is satisfied. From this, we can predict that these classifiers will not perform well with this dataset. Meanwhile, decision trees and forests are robust to outliers and can handle skewed, multi-modal data that does not meet normality assumptions [14], so can be expected to perform better. Also, Wu [6] and Lowe [8] have already demonstrated the effectiveness of the SVM and GBM classifiers when applied to a similar dataset.

C. *Data preparation*

The dataset requires some additional preparation before models can be trained. The number of samples in each category is imbalanced (fig. 7), known as 'naïve behaviour.' This means that the classifier accuracy is maximal in the more common categories.



```
[2] import pandas as pd

    trainData = pd.read_csv(filePath)
    print("Number of samples in each category:")
    print(trainData['TypeOfDefects'].value_counts().sort_index())

    Number of samples in each category:
    1    103
    2    129
    3    276
    4     55
    5     38
    6    287
    7    470
    Name: TypeOfDefects, dtype: int64
```

Figure 7: naïve behaviour in the dataset

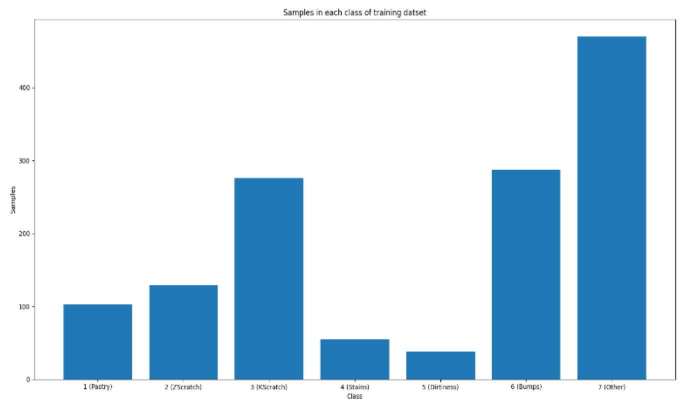


Figure 8: bar plot of naive behaviour in dataset

One technique for dealing with naïve behaviour is resampling – either removing samples from the majority class (under-sampling) or adding more examples from the minority class (over-sampling) [15].

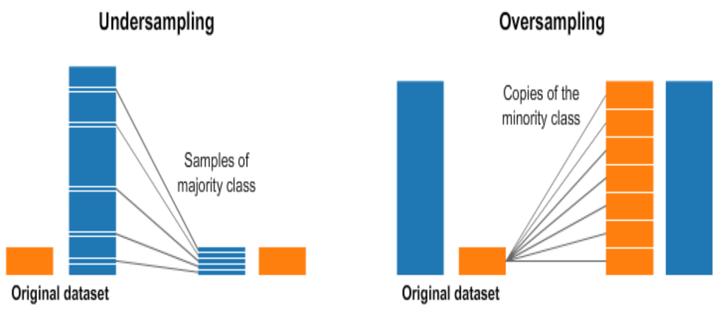
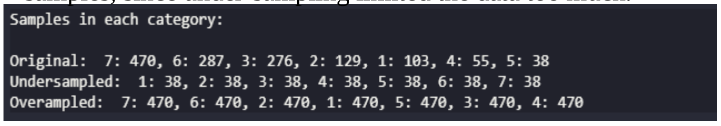


Figure 9: over- and under-sampling

Random over-sampling was chosen to balance the samples, since under-sampling limited the data too much.

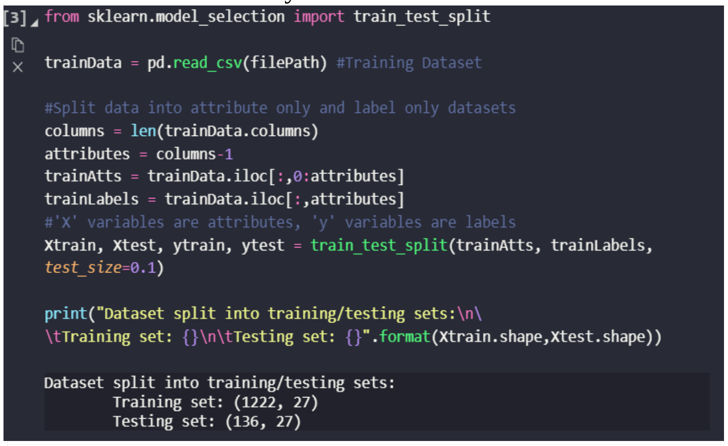


```
Samples in each category:

Original:  7: 470, 6: 287, 3: 276, 2: 129, 1: 103, 4: 55, 5: 38
Undersampled:  1: 38, 2: 38, 3: 38, 4: 38, 5: 38, 6: 38, 7: 38
Overampled:  7: 470, 6: 470, 2: 470, 1: 470, 5: 470, 3: 470, 4: 470
```

Figure 10: results of under- and over-sampling

The dataset is then split into a training set (90% of original) and a test set (10%). The test set will be used with the trained models to validate accuracy.



```
[3] from sklearn.model_selection import train_test_split

    trainData = pd.read_csv(filePath) #Training Dataset

    #Split data into attribute only and label only datasets
    columns = len(trainData.columns)
    attributes = columns-1
    trainAtts = trainData.iloc[:,0:attributes]
    trainLabels = trainData.iloc[:,attributes]
    #'X' variables are attributes, 'y' variables are labels
    Xtrain, Xtest, ytrain, ytest = train_test_split(trainAtts, trainLabels,
    test_size=0.1)

    print("Dataset split into training/testing sets:\n\
    \tTraining set: {}\n\tTesting set: {}".format(Xtrain.shape,Xtest.shape))

    Dataset split into training/testing sets:
            Training set: (1222, 27)
            Testing set: (136, 27)
```

Figure 11: splitting the dataset for accuracy testing

D. *Initial model training and evaluation*

The results of initial training of the 6 models are shown in table 3, also demonstrating the effectiveness of resampling.

Model	10-fold Cross-validation Accuracy Score			Train time (m:s)
	Before resampling	After resampling	Difference	
Random forest	77.41%	93.55%	+ 16.14	00:00.47
Gradient boost	78.23%	92.94%	+ 14.71	00:01.12
Decision tree	68.49%	92.27%	+ 23.78	01:53.41
kNN	43.21%	71.56%	+ 28.35	00:09.10
Naïve Bayes	45.83%	58.32%	+ 12.49	00:00.06
SVM	50.24%	41.88%	- 8.36	00:14.73

Table 3: initial model training results (sorted by best accuracy after resampling)

As predicted, RF and DT initially performed among the best, while the performance of kNN and NB suffered due to the data not meeting the correct requirements. Interestingly, SVM performed the worst, contrasting Wu's research [6].

E. *Fine-tuning*

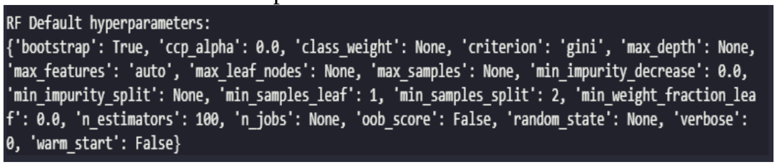
The RF model was selected to be used based on its performance. To maximise performance, fine-tuning was carried out using Brownlee's guide [12]:

1) Oversampling carried out, clear improvements seen in table 3

2) Several algorithms tested and the highest performing selected

3) Hyperparameters tuned using random grid search which samples a wide range of values:

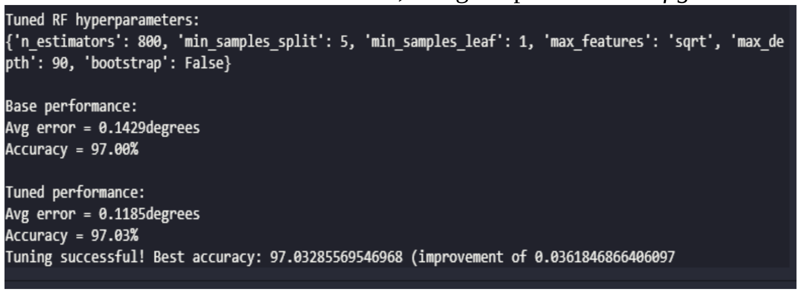
- default parameters:



```
RF Default hyperparameters:
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None,
'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0,
'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_lea
f': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose':
0, 'warm_start': False}
```

Figure 12: default random forest hyperparameters

- tuning resulted in an improved accuracy of up to 97% on the test dataset, using the parameters in *figure* 13



```
Tuned RF hyperparameters:
{'n_estimators': 800, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_de
pth': 90, 'bootstrap': False}

Base performance:
Avg error = 0.1429degrees
Accuracy = 97.00%

Tuned performance:
Avg error = 0.1185degrees
Accuracy = 97.03%
Tuning successful! Best accuracy: 97.03285569546968 (improvement of 0.0361846866406097
```

Figure 13: result of hyperparameter tuning

Hyperparameter	**Base**	**Tuned**
n_estimators (№ trees)	100	800
min_samples_split (min samples required to split a node	2	5
min_samples_leaf (min samples at each leaf node)	1	1
max_features (number of features considered at splits)	auto	sqrt
max_depth (max levels in tree)	none	90
bootstrap (method of selecting samples to train each tree)	true	false

Table 4: tuned vs base hyperparameters

IV. Conclusion

A machine learning model was successfully developed. After investigations, the random forest model achieved the best overall results, and the fine-tuned model was used to make predictions using the testing dataset 'test.csv'. This model achieved high accuracy initially, though in future, it may be best to try multiple configurations of hyperparameters and compare performance, instead of just one configuration.

V. Bibliography

[1] B. A. Ellis, "Condition Based Maintenance," Nov. 2008. Accessed: Mar. 09, 2021. [Online]. Available: www.jethroproject.com;

[2] B. Wood, "Intelligent building care," *Facilities*, vol. 17, no. 5, pp. 189–194, May 1999, doi: 10.1108/02632779910259288.

[3] G. M. Knapp and H. P. Wang, "Machine fault classification: A neural network approach," *Int. J. Prod. Res.*, vol. 30, no. 4, pp. 811–823, 1992, doi: 10.1080/00207543.1992.9728458.

[4] "Classification | DataRobot Artificial Intelligence Wiki." https://www.datarobot.com/wiki/classification/ (accessed Apr. 16, 2021).

[5] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine learning for predictive maintenance: A multiple classifier approach," *IEEE Trans. Ind. Informatics*, vol. 11, no. 3, pp. 812–820, Jun. 2015, doi: 10.1109/TII.2014.2349359.

[6] J. Wu, "Faulty Steel Plates Classification." Accessed: May 13, 2021. [Online]. Available: https://rstudio-pubs-static.s3.amazonaws.com/378820_f2d0a64cbe044e31b95aba4fb9eac037.html.

[7] B. Tripathi, "What are C and gamma with regards to a support vector machine?" Oct. 17, 2019, Accessed: May 13, 2021. [Online]. Available: https://qr.ae/pGvIcf.

[8] D. Lowe, "Multi-Class Classification Model for Faulty Steel Plates Using Python," *GitHub*, Nov. 19, 2019. https://github.com/daines-analytics/tabular-data-projects/blob/master/py-classification-faulty-steel-plates-take2/py-classification-faulty-steel-plates-take2.ipynb (accessed Apr. 20, 2021).

[9] J. Brownlee, "Hypothesis Test for Comparing Machine Learning Algorithms," Aug. 21, 2020. https://machinelearningmastery.com/hypothesis-test-for-comparing-machine-learning-algorithms/ (accessed May 13, 2021).

[10] T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," *Neural Comput.*, vol. 10, no. 7, pp. 1895–1923, Oct. 1998, doi: 10.1162/089976698300017197.

[11] J. Brownlee, "17 Statistical Hypothesis Tests in Python," Aug. 15, 2018. https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/ (accessed May 13, 2021).

[12] J. Brownlee, *Machine Learning Performance Improvement Cheat Sheet*, 1.1. 2017.

[13] "Supervised Learning Workflow and Algorithms." https://uk.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html (accessed May 13, 2021).

[14] V. Mendekar, "It's All About Assumptions, Pros & Cons," Jan. 2017. https://medium.com/swlh/its-all-about-assumptions-pros-cons-497783cfed2d (accessed May 14, 2021).

[15] B. Kumar, "10 Techniques to deal with Imbalanced Classes in Machine Learning," Jul. 23, 2020. https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/ (accessed May 14, 2021).