Artificial Intelligence and Machine Learning for Robotics

Individual Coursework

Daniel Woodhall – 350272

Department: School of Aerospace

Date: 11/01/2021

AI & ML for Robotics

Individual Coursework

Contents

1. Introduction

This project investigates the use of multiple machine-learning techniques to solve a classification problem defined by the dataset "Robot Execution Failures". The analysis was done in Python using a Jupyter Notebook, where 7 different models were constructed, including an artificial neural network. In this report, the accuracy of the 5 best performing machine-learning techniques will be compared and discussed. The machine-learning techniques used will be explained with mathematical representations. The use of SMOTE and PCA techniques were also investigated and will be explained.

2. Literature Review

2.1 Fundamentals of AI&ML

Machine-learning (ML) is generally considered to be the ability of a machine to autonomously learn and improve from experience without strict programming [1]. There are many ways to categorise AI, one way is: Human or Non-human, and Thinking or Acting [2]. AI can also be split into symbolic or non-symbolic, where symbolic AI includes human-readable information and non-symbolic relies on statistical methods [3]. ML techniques can be used to predict an outcome from a set of inputs, these problems fall under regression (predicting a numerical value) or classification (predicting a class, e.g 'Cloudy' or 'Not-cloudy' weather).

2.1.1 Types of ML

There are three main categories of ML: Unsupervised, supervised, and reinforcement learning. Unsupervised learning methods attempt to find patterns in a given dataset without reference to a known outcome; this can be used to cluster or determine associations in data based on some underlying structure. With no outcome to refer to, there is no way to determine the accuracy of unsupervised learning methods [4].

Supervised learning methods attempt to find a function that maps a set of inputs to a given output; with the correct outputs given by a supervisor. The function created by the supervised learning method can then be used to predict an output when given a set of inputs [5].

Reinforcement learning methods construct a sequence of actions that form a policy, with a reward or penalty given based on the result of the actions performed. The machine will attempt to maximise the reward and minimise the penalty using trial and error across multiple policies. The machine becomes better at performing the task until it has maximised the reward for a given policy [6].

2.1.2 Hypothesis & Hypothesis Space

In ML a hypothesis refers to a specific example of a model used to approximate the target function, mapping inputs to outputs. Hypothesis space is the possible space in which these hypotheses can exist; defined by the chosen algorithm and configuration (hyper-parameters).

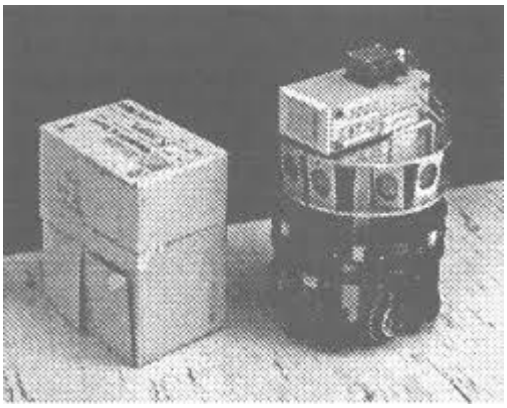

Figure 1 - OBELIX Robot (S. Mahadevan, 1991)

2.2 Applications of AI & Reinforcement Learning in Robotics

AI & reinforcement learning techniques are applied extensively in robotics, giving robots the ability to learn for themselves where manually programmed solutions are challenging [7]. One example of this is the grasping of an object by a manipulator. It would be extremely complex to manually program this interaction, especially if it was required to be performed for multiple object types. Using AI the robot could learn the best set of movements to pick up the object, being given a reward for picking up the object or a penalty for dropping/knocking over the object [8]. An early example of reinforcement learning in robotics was the OBELIX robot, **Error! Reference source not found.Error! Reference source not found.**, which was tasked with pushing a box across a room. The OBELIX robot had to find the box in the room, thus a reward was assigned when the sonar sensors were activated, encouraging the robot to move towards objects. Natural language processing is a more modern AI application in robotics, allowing a robot to interpret and understand human language [9].

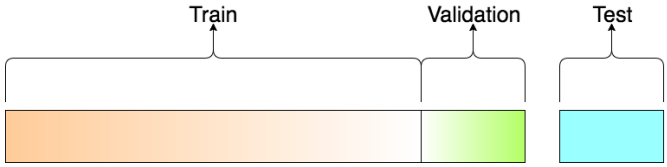

Figure 2 - Train/Validation/Test Splitting (Tarang Shah, 2017)

2.3 Data-Splitting

To assess the effectiveness of a ML model the dataset must be split into subsets, training and testing. The training subset is given to the model with both the inputs and outputs shown to allow the model to map a function between the two. The test subset is then used to assess the accuracy of the constructed function on data that is previously unseen; this ensures that the model is generalised and not fit specifically to the test subset, giving an unbiased view of model effectiveness [10]. The training subset can be split further to contain a validation subset, this validation subset is a section of the training subset that is held back during training and can be used to tune the hyper-parameters of a given model by giving an estimate of the accuracy; shown in **Error! Reference source not found.**.

Using the same validation subset to tune the hyper-parameters results in an increasingly more bias model as the accuracy on the validation subset becomes incorporated into the model [11]. To avoid bias, K-fold cross-validation can be used. This method iterates over the training subset, assigning a new validation subset for each iteration, where K equals the number of iterations [12]; illustrated in Figure 3.

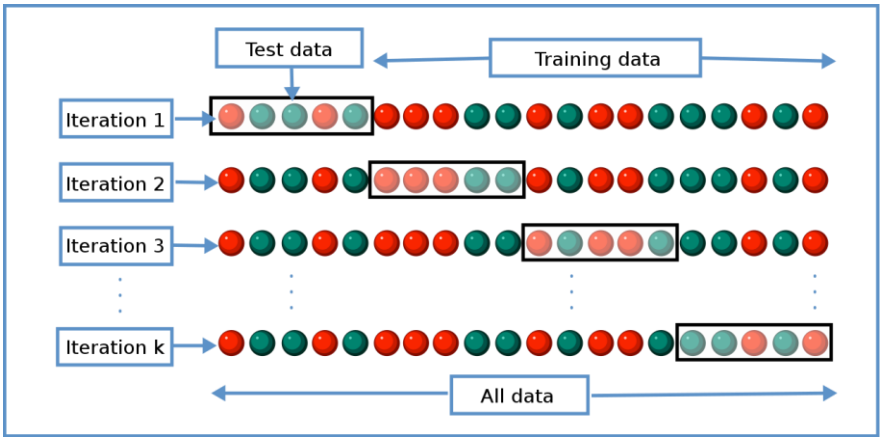

Figure 3 - K-fold cross validation (Gufosowa, 2012)

2.4 SMOTE

Refer to Appendix A.

2.5 Dimensionality Reduction

Dimensionality-reduction transforms high-dimensional data to a lower-dimension, maintaining important features from the original dataset. High-dimensional data is subject to the Curse of Dimensionality which can cause issues to arise when producing models. Increasing the number of parameters increases the number of samples needed to represent all combinations of parameter values as well as the complexity of the model. Having a high number of parameters also increases the chance of the model becoming over-fit; becoming dependent on the training data and giving poor results on real-data [13]. High-dimensional data also increases computational time, as defined by the Big O notation [14].

2.5.1 PCA

PCA (Principal Component Analysis) is a dimensionality reduction method that involves producing PCs (Principal Components) from the original dataset. Each PC explains a certain amount of variance from the original dataset [15]. The first step in PCA is the standardisation of the dataset, ensuring that each variable contributes equally to the PCs. PCA is especially sensitive to variances in variables meaning that if there are large discrepancies between the ranges of variables, the variables with the largest ranges will skew the PCs; leading to bias. The standardisation includes transforming the variables to be on the same scale. An issue with PCA is the loss of explainability of the model [16].

After standardisation a covariance matrix of the variables is constructed; with positive and negative covariance's meaning correlation or inverse correlation between variables respectively. From the covariance matrix eigenvectors and eigenvalues are computed; where eigenvectors show the direction of the PCs and eigenvalues show the amount of variance in the PCs. Eigenvalues are then sorted in descending order, selecting the corresponding eigenvectors until N eigenvectors are selected (where N is the desired dimensions in the transformed dataset).

A projection matrix is then constructed from the chosen eigenvectors, with the projection matrix describing the transformation of the data-points. The final step is to apply this projection matrix to the standardised original dataset, resulting in a transformed dataset with N dimensions [17]. An example of PCA is shown in Figure 4.

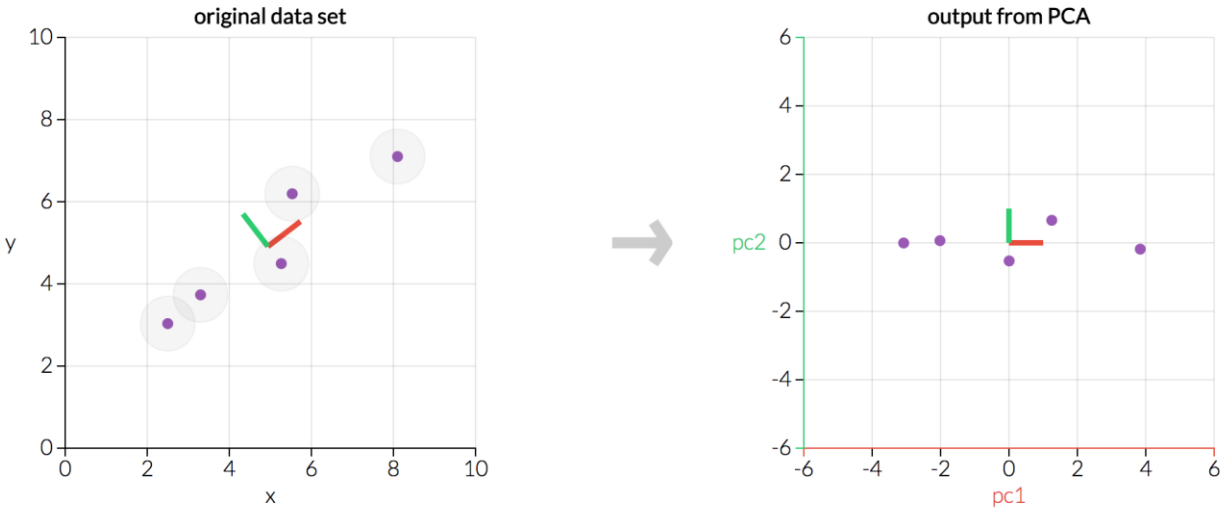

Figure 4 - PCA (Brems, 2017)

2.6 Feature Selection

Feature selection differs from dimensionality reduction as it doesn't transform the dataset, it selects specific variables and constructs a new dataset only containing these. There are three main types of feature selection algorithms: Filter, Wrapper, and Embedded methods. Applying feature selection to the entire dataset can result in a bias model, as such feature selection should be applied within the model individually to each fold in cross-validation [18].

Filter methods use a statistical algorithm, assigning a score to each variable from the chosen algorithm and ranking them based upon the score; Chi-squared test and Pearson's Correlation are two commonly used examples [19].

Wrapper methods construct different combinations of parameters which are then compared using the accuracy score from a predictive model.

Embedded methods assess which parameters contribute the most to the accuracy of a model during the training of the model. Regularisation methods are the most common type of embedded methods; with examples being Elastic Net and Ridge Regression.

3. Scope and Objectives

This project aims to produce and analyse several ML classification models to determine failure methods. Models are to be constructed in Python using open-source libraries. Model construction should be robust and variable wherever possible. A list of objectives is shown below:

- Perform initial exploratory data analysis to determine any obvious indicators of a classification.
- Produce at least 3 ML classification models, including an Artificial Neural Network.
- Perform hyper-parameter tuning for the ML models.
- Analyse metrics from the constructed models and determine the best model.
- Investigate the use of dimensionality reduction techniques for this dataset.

4. Methodology

4.1 Data Preparation

The data was checked for any null or duplicated values, with one duplicated row found in LP2-5 which was dropped. Due to the small datasets and severe class imbalances present, the multi-classification case was only performed for one model, with the remaining models being converted to a binary classification problem.

4.1.1 Feature Selection

Feature selection was not implemented within the code. If feature selection was implemented, an Elastic Net embedded algorithm would be used as described in 2.6 Feature Selection. Embedded methods are generally more accurate than Wrapper or Filter methods and reduce overfitting as they have inbuilt regularisation functions [20]. The equation for Elastic Net in classification is shown in (1), where $\emptyset$ is a margin-based loss function and $y \in \{1, -1\}$.

$$\hat{\beta} = argmin \sum_{k=1}^{n} \emptyset(ykx_k^T\beta) + \lambda_2 \left|\left|\beta\right|\right|^2 + \lambda_1 ||\beta||_1 \quad (1)$$

Applying feature selection helps to alleviate the Curse of Dimensionality, although increases computational time.

4.1.2 PCA

PCA was performed on LP1, to demonstrate how it would be applied to each dataset, PCA has not been used in the models. If there are more parameters than observations, which is the case for LP2-3, models can become over-fit, giving false confidence in their accuracy; PCA would remove this issue. PCA would also reduce the computational time of the models. PCA would however result in losing some of the variance from the data, potentially giving worse models [17]. Since the data has so many dimensions, PCA would likely provide more positives than negatives.

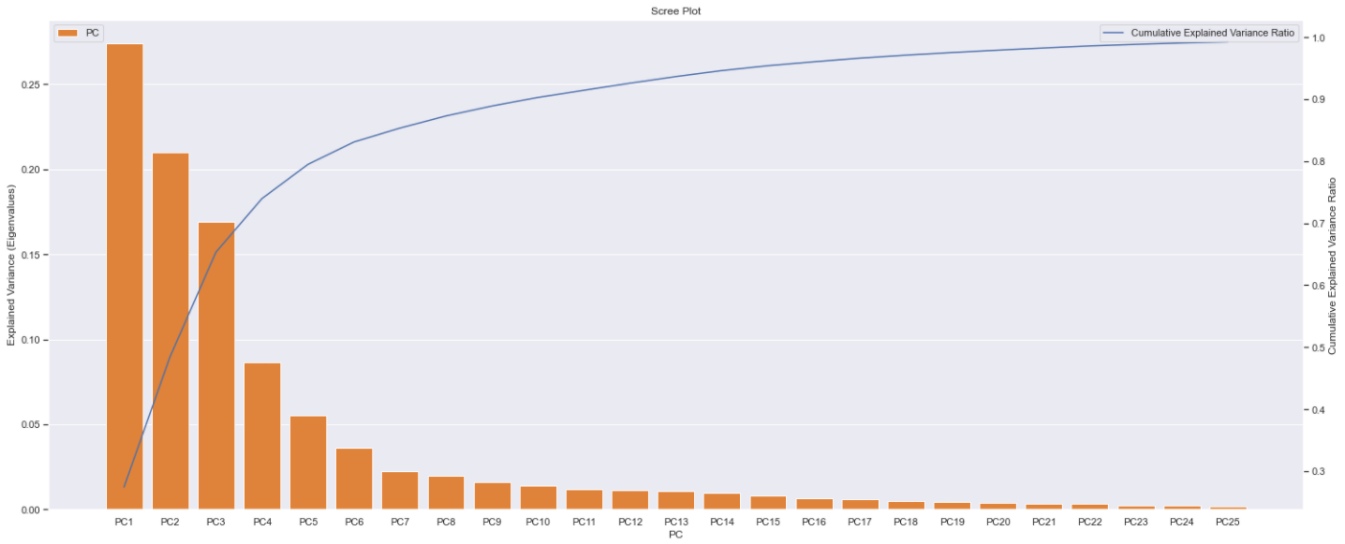

Figure 5 - Scree Plot for LP1

4.2 Cross-Validation

K-fold cross validation, K = 2, was applied to the models as shown in Figure 3. Since the datasets are small, 2 cross-folds were used to ensure that each fold properly represents the variation in the dataset. Using a lower amount of folds also reduces the computational time for the models [21].

4.3 SMOTE

Refer to Appendix A.

4.4 Models

7 different ML models were constructed for this project, due to the word-count restriction of this report the three best performing models have been basically explained, along with the Neural Network.

4.4.1 Random Forest Classifier

Random Forest Classifiers are decision-tree ensemble-learning methods that use a "bagging" (bootstrap-aggregating) technique. The model contains multiple decision trees, each giving a predicted classification; the classification predicted most is selected as the output. Decision trees consist of three node types, root nodes which occur first and often are the most important in determining output, internal nodes which represent the decision state, and leaf nodes which are the outputs; Figure 6 shows an illustration.

The Gini Impurity, (2), is used to score each decision tree, with a lower impurity value suggesting that the decision tree will give a more accurate estimate.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2 \quad (2)$$

In (2), p_i is the frequency of the class being observed and C is the number of classes in the data. Entropy can be used as a metric for evaluation of the decision trees. The decision trees can then be weighted in terms of accuracy based on their Gini values to give a better estimate of the model [22].

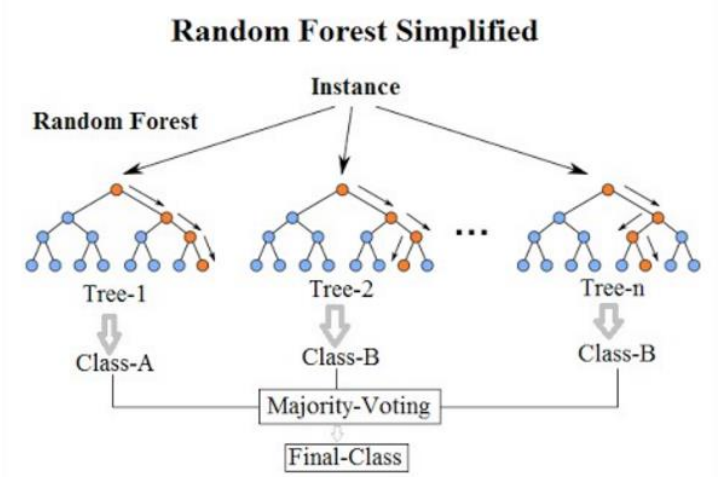

Figure 6 - Random Forest Classifier (Jagannath, 2014)

4.4.2 Naïve Bayes

Naïve Bayes is a probabilistic classification method, outputting the probability of a given classification as opposed to the classification itself. Bayes Theorem, (3), allows the probability of a classification to be calculated, given a set of inputs. $P(Y|X)$ cannot be calculated directly, but instead is calculated from $P(Y)$ and $P(X|Y)$, determined from the training data, as shown:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \quad (3)$$

In (3), X represents the inputs and Y represents the classification [23]. In Naïve Bayes, the assumption that all inputs are independent is made (which is unlikely to be true). This changes the number of parameters used from exponential to linear, resulting in much faster computation; as a result Naïve Bayes handles high-dimensional data well which is especially useful for the given datasets [24].

4.4.3 SVM – RBF

SVM algorithms fit provided data to a hyperplane, with the resulting hyperplane categorising a data-entry depending on where it lies; a simple example is shown in Figure 7. As an example, say there is a 2-D space with a hyperplane, (4):

$$\beta_0 + (\beta_1 \times x_1) + (\beta_2 \times x_2) = 0 \quad (4)$$

Any point above this line satisfies equation (5):

$$\beta_0 + (\beta_1 \times x_1) + (\beta_2 \times x_2) > 0 \quad (5)$$

And any point below the line satisfies equation (6):

$$\beta_0 + (\beta_1 \times x_1) + (\beta_2 \times x_2) < 0 \quad (6)$$

Depending on whether the point satisfies equation (5) or (6), it will be given a different classification [25].

From the constructed hyperplane, boundaries are also calculated, where the boundary is the shortest distance from the given data-entries to the hyperplane. This boundary is attempted to be maximised by the algorithm.
In the case of non-linearly separable data, soft-boundaries and Kernel functions can be introduced [26]. Soft-boundaries simply allow for a few data-entries to be misclassified when constructing the boundary

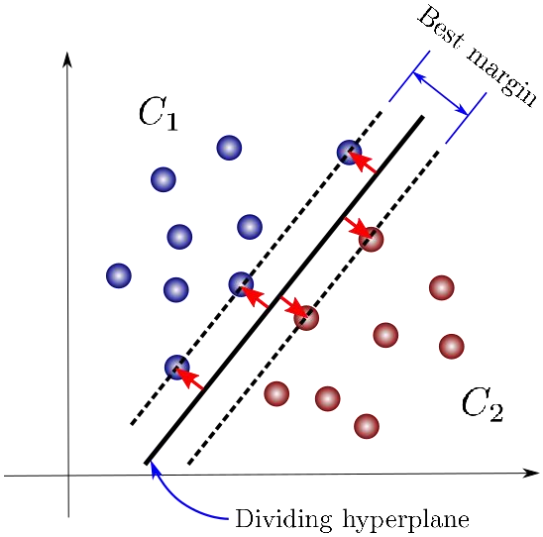

Figure 7 - SVM Illustration (Carrasco, 2019)

conditions. Kernels are more complex and involve transforming a dataset to higher-dimensional state. Both linear and Radial Basis Function (RBF) kernels were used in this project, only the RBF kernel will be explained here as it performed significantly better.

An example of the RBF calculation will be shown below, starting with (7) [27], where a and b refer to two data-entries and γ is a scalar for the relative influence:

$$e^{-\gamma(a-b)^2} \quad (7)$$

When a and b are far apart, the relative influence is close to 0; a lower influence value meaning that values have less correlation [27].

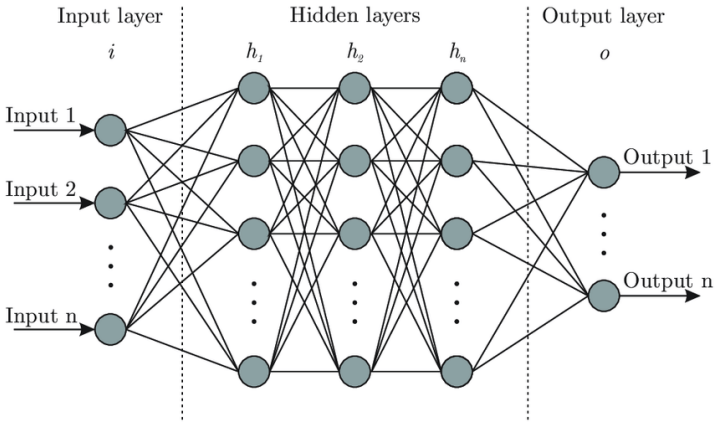

Figure 8 - Basic Neural Network Structure (Shukla, 2019)

4.4.4 ANN

Neural networks (NN) are comprised of 3 layer types: an input layer, output layer and one or several hidden layers. NNs are self-teaching, meaning they iteratively improve themselves throughout the training process by adjusting a weight and threshold value; the weight represents the amount the weight of that nodes output to the decision of the NN, the threshold value represents a value that the output of the node has to reach in order to be 'activated'. The hidden layers are comprised of interconnected nodes where data is processed and transferred from one layer of nodes down (towards the output layer). Each node in the hidden layers is initially assigned a random weighting and threshold value. During the training process, the weights and threshold values are automatically adjusted to minimise the error of the network [28].

The basic structure of a NN is shown in Figure 8. Neural networks can be tuned by altering the amount of hidden layers and nodes per layer. For simple problems the general consensus is that 1-5 hidden layers are sufficient; more hidden layers doesn't necessarily improve accuracy and can result in over-fitting.

There are several drawbacks of NNs, namely their 'black-box' (meaning they are not explainable) nature and intensive computational demands [29].

5. Results and Discussion

The results section will be split per dataset, comparing the results for the best 3 models and the ANN for each. The results for all datasets are shown below:

Classifier	LP1	LP2	LP3	LP4	LP5	Average
Random Forest	0.963	0.786	0.786	1	1	0.907
XGBoost	0.926	0.786	0.786	1	0.959	0.8914
K-Nearest Neighbors	0.963	0.429	0.429	0.971	0.816	0.7216
Naïve Bayes	1	0.929	0.929	0.971	0.816	0.929
SVM - Linear Kernel	0.815	0.5	0.5	0.857	0.673	0.669
SVM - RBF Kernel	1	0.929	0.929	0.971	0.959	0.9576
Artificial Neural Network	0.741	0.286	0.429	0.943	0.894	0.6586

Figure 9 - Table of Model Accuracies (Ratio of correct predictions)

Classifier	LP1	LP2	LP3	LP4	LP5	Average
Random Forest	0.074	0.094	0.099	0.045	1.073	0.277
XGBoost	0.209	0.024	0.025	0.116	0.16	0.1068
K-Nearest Neighbors	0.003	0.002	0.002	0.002	0.002	0.0022
Naïve Bayes	0.003	0.003	0.004	0.004	0.003	0.0034
SVM - Linear Kernel	0.005	0.003	0.003	0.004	0.014	0.0058
SVM - RBF Kernel	0.003	0.003	0.003	0.004	0.004	0.0034
Artificial Neural Network	97.163	92.407	89.679	98.632	100.472	95.6706

Figure 10 - Table of Model Computational Times (Seconds)

All models were run on an Intel I5-3570K, overclocked to 4.1GHz.

Confusion matrices and ROC curves were plotted for the models in addition and are shown in the 'robot-failure-classification.ipynb' file. Other metrics were also calculated (precision, F1, recall) and are shown in the 'robot-failure-classification.ipynb' file.

5.1 LP1

The LP1 dataset showed the best performance with the SVM-RBF model and the Naïve Bayes model, both achieving an accuracy of 100%. Naïve Bayes' result was expected as it is known to perform well on high-dimensional data (this conclusion applies to all datasets) [24]. The next best models were Random Forest and K-Nearest Neighbors, both achieving accuracies of 96.3%.

The ANN produced an accuracy of 74.1%, which is expected given the tendency of NNs to improve as the amount of data increases (with LP1 dataset containing 88 samples) [28].

As with all the datasets, since LP1 contains such a small amount of entries, there cannot be great confidence in the results that are produced [30]. The models could be significantly over-fit to the dataset, producing over-estimation of their accuracy. If this is the case, the models will produce much worse results when applied to a real world scenario. (The conclusions in this paragraph can be applied to all datasets).

Naïve Bayes and SVM-RBF showed roughly the same computational times, 0.003s. Random Forest was slower although not by any significant margin, taking 0.074s to train. The NN took 97.163s to train, showing an extreme jump in computational time required to train. This is expected due to the computational power required for NNs and could be alleviated somewhat if it was trained using a GPU instead (this conclusion applies to all datasets) [31].

5.2 LP2

For LP2, the best performing models were again SVM-RBF and Naïve Bayes, with accuracies of 92.9%. Random Forest gave an accuracy of 78.6%, a decrease in accuracy of 14.3% compared to the best performing models.

The ANN showed extremely poor results, an accuracy of only 28.6%. This is for the same reason as LP1, although is exacerbated as LP2 is only 47 samples (this is the same for LP3) [32].

Naïve Bayes and SVM-RBF again showed the same computational time, 0.003s. Random Forest was again slower although not by any significant margin, taking 0.094s to train. The NN took 92.407s

5.3 LP3

LP3 showed identical accuracy results as LP2. Computational time was not significantly different for any of the models apart from, again, the NN which had a computational time of 89.679s.

5.4 LP4

The best performing model for LP4 was Random Forest, achieving 100% accuracy, potentially due to the increased dataset size. Naïve Bayes and SVM-RBF achieved the same accuracy of 97.1%.

The ANN for LP4 gave an accuracy of 94.3%, a vast improvement over the previous datasets. This could be due to the increased dataset size, 117 samples, although it is not a significant enough increase in size to explain such a large jump in accuracy [33]. The reasoning for this jump is inconclusive, especially since the ANN is not explainable due to the 'black-box' nature (this applies to LP5).

5.5 LP5

The best performing model for LP5 was Random Forest, achieving 100% accuracy, again potentially due to the increased dataset size. LP5 was the only dataset where Naïve Bayes and SVM-RBF produced different results, with 81.6% and 95.9% accuracy respectively.

The ANN for LP5 gave an accuracy of 89.4%, performing better than Naïve Bayes for the largest dataset. This could again be due to the increased dataset size, 164 samples, however it performs worse than LP4 which would not be expected as it contains 47 more samples [33].

Computational time for the Random Forest model increased by a magnitude of 10 when compared to the next slowest dataset training. This is an unusual behaviour as the dataset did not increase by much, this may suggest that the function that maps the inputs to the outputs for LP5 is more complex; however if this was the case the other models would be expected to also take much longer to train which was not the case.

6. Conclusions

The models constructed in this project successfully achieved the scope and objectives set out. The results from the models were analysed and the best model for this dataset was, on average, found to be a SVM using a RBF Kernel. However, since the datasets are so small not a lot of confidence can be placed in the accuracy of the models as they could be severely over-fit.

It is difficult to draw any conclusive results or comparisons between such a small dataset. Some classifications had only 3 instances ('lost' in LP3), resulting in the problem having to be simplified to a binary-classification. More results and metrics are shown in the 'robot-failure-classification.ipynb' file, with more time these could be analysed in addition to the accuracy metric.

If this project were to be developed further, PCA would be applied to the models, comparing the results of the models with and without PCA to assess the effectiveness of PCA for this dataset. The variable importance for each dataset would also be assessed and feature selection applied as outlined in 2.6 Feature Selection. The multi-class case would also be analysed for all models.

If more time was available, the NN would be further tuned by adjusting the amount of layers and nodes, although it would not be expected to produce better results than the other models, again due to the small datasets (NNs generally needing 10,000's of data-entries to produce good results). The NN did show significant improvement with the larger datasets, which is to be expected due to nature of NNs to improve themselves over each iteration.

References

[1] D. S. C. C. T. D Michie, "Machine Learning," *Neural and Statistical Classification,* vol. 13, pp. 1 - 298, 1994.

[2] S. J. R. Peter Norvig, Artificial Intelligence: A Modern Approach, Saddle River: Prentice Hall, 1995/2020.

[3] D. Willshaw, "Non-symbolic approaches to artificial intelligence and the mind," *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences,* vol. 349, pp. 87 - 102, 1994.

[4] Z. Ghahramani, "Unsupervised Learning," in *Advanced Lectures on Machine Learning*, Berlin, Springer, 2003, pp. 77 - 112.

[5] R. T. J. F. Trevor Hastie, "Overview of Supervised Learning," in *The Elements of Statistical Learning*, New York, Springer, 2008, pp. 9 - 41.

[6] A. G. B. Richard S. Sutton, Reinforcement Learning: An Introduction, Cambridge: MIT Press, 2018.

[7] J. A. B. J. P. Jens Kober, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research,* vol. 32, no. 11, pp. 1238 - 1274, 2013.

[8] E. J. O. N. C. F. J. I. S. L. Deirdre Quillen, "Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 2018.

[9] K. R. Chowdhary, "Natural Language Processing," in *Fundamentals of Artificial Intelligence*, New Delhi, Springer, 2020, pp. 603 - 649.

[10] K. N. B. Richard R. Picard, "Data Splitting," *The American Statistician,* vol. 44, no. 2, pp. 140 - 147, 1990.

[11] Y. G. Yoshua Bengio, "No Unbiased Estimator of the Variance of K-Fold Cross-Validation," *Journal of Machine Learning Research,* vol. 5, pp. 1089 - 1105, 2004.

[12] Z. Reitermanova, "Data Splitting," *WDS'10 Proceedings of Contributed Papers*, pp. 31 - 36, 2010.

[13] M. Koppen, "The curse of dimensionality," in *5th Online World Conference on Soft Computing in Industrial Applications*, 2000.

[14] S. J. Chivers I., "An Introduction to Algorithms and the Big O Notation," in *Introduction to Programming with Fortran*, Cham, Springer, 2015, pp. 359 - 364.

[15] e. a. S. Karamizadeh, "An Overview of Principal Component Analysis," *Journal of Signal and Information Processing,* vol. 4, pp. 173 - 175, 2013.

[16] M. Ringner, "What is principal component analysis?," *Nature Biotechnology,* vol. 26, pp. 303 - 304, 2008.

[17] I. T. Jolliffe, "Principal component analysis," *Technometrics,* vol. 45, no. 3, p. 276, 2003.

[18] H. L. M. Dash, "Feature Selection for Classification," *Intelligent Data Analysis,* vol. 1, no. 3, pp. 131 - 156, 1997.

[19] A. A.-B. M. T.-S. Noelia Sánchez-Maroño, "Filter Methods for Feature Selection – A Comparative Study," in *International Conference on Intelligent Data Engineering and Automated Learning*, Berlin, 2007.

[20] O. C. J. W. A. E. Thomas Navin Lal, "Studies in Fuzziness and Soft Computing," in *Feature Extraction*, Berlin, Springer, 2006, pp. 137 - 165.

[21] J. A. C.-O. J. F. M.-T. Julio Hernandez, "An Empirical Study of Oversampling and Undersampling for Instance Selection Methods on Imbalance Datasets," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Berlin, 2013.

[22] M. Pal, “Random forest classifier for remote sensing classification,” *International Journal of Remote Sensing,* vol. 26, no. 1, pp. 217 - 222, 2005.

[23] I. Rish, “An empirical study of the naive Bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence,* 2001.

[24] Q. M. J. W. M. A. h. A. S.-A. Mohammed J. Islam, “Investigating the Performance of Naive- Bayes Classifiers and K- Nearest Neighbor Classifiers,” in *2007 International Conference on Convergence Information Technology,* 2007.

[25] Q. D.-X. Z. Wu, “Analysis of support vector machine classification,” *Journal of Computational Analysis & Applications,* vol. 8, no. 2, 2006.

[26] A. J. W. Ben-Hur, “A user’s guide to support vector machines,” in *Data mining techniques for the life sciences,* Humana Press, 2010, pp. 223 - 239.

[27] H.-M. Gutmann, “A radial basis function method for global optimization,” *Journal of global optimization ,* vol. 19, no. 3, pp. 201 - 227, 2001.

[28] H. B. D. M. B. Martin T. Hagan, Neural Network Design, Computer Science, 1995.

[29] R. L. S.S, “On the Computational Efficiency of Training Neural Networks,” *Advances in Neural Information Processing Systems,* 2014.

[30] E. G. E. P. A. J. C. Andrius Vabalas, “Machine learning algorithm validation with a limited sample size,” *PloS one,* vol. 14, no. 11, 2019.

[31] K. J. Kyoung-Su Oh, “GPU implementation of neural networks,” *Pattern Recognition,* vol. 37, no. 6, pp. 1311 - 1314, 2004.

[32] N. V. e. a. Chawla, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal of Artificial Intelligence Research,* vol. 16, pp. 321 - 357, 2002.

[33] S. G. F. H. N. V. C. Alberto Fernandez, “SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary,” *Journal of Artificial Intelligence Research,* vol. 61, pp. 863 - 905, 2018.

Appendices

7.1 Appendix A - SMOTE

SMOTE (Synthetic Minority Oversampling Technique) is a technique used to reduce class imbalances in a dataset. Class imbalances can result in a model that is bias towards a specific classification. Older methods of solving class imbalances include over-sampling and under-sampling where examples from the minority or majority class respectively are duplicated, however this does not introduce any additional information to the model [21]. SMOTE introduces new data by selecting entries from the minority classes that are proximal (determined by K-nearest neighbours) in the feature space, constructing a line between these entries; a new data-point is then produced randomly along this line [32]. This process is repeated until the dataset contains a specified amount of the minority classifications.

SMOTE was applied to the training subsets to alleviate the issue of class imbalances in the data. SMOTE should not be applied to the testing subsets as the models would then be using synthesised data to validate their results [33].