UNIVERSITY OF PRINCE EDWARD ISLAND

Controlling Exploration in EMNA-TC Through Machine Learning

by

Jordan Luke

Supervised by: Dr. Antonio Bolufé-Röhler

A thesis submitted in partial fulfillment for the
Honours degree in Computer Science

in the
Faculty of Science
School of Mathematical and Computational Science

March 2022

Declaration of Authorship

I, Jordan Luke, declare that this thesis titled, 'Controlling Exploration in EMNA-TC Through Machine Learning' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 22-03-2022

UNIVERSITY OF PRINCE EDWARD ISLAND

Abstract

Faculty of Science
School of Mathematical and Computational Science

Honours in Computer Science

by Jordan Luke

Exploration plays a key role in the performance of metaheuristics. An algorithm should perform more exploration when reaching the "ideal search scale"; this happens when solutions are regularly sampled from different attraction basins. The moment this search scale is reached depends on the topological features of the objective function and the inherent randomness of the heuristic optimization process. Previous work on adjusting exploration have mostly used fixed rules based on fitness improvement. In this project, we model it as a supervised machine learning problem. We apply a data-centric approach to understand whether variations in the data are more relevant than variations in the classification models. For our study we use the Estimation Multivariate Normal Algorithm with Thresheld Convergence, which provides an ideal framework as it allows us to directly control exploration through the gamma parameter. Optimization results show that certain machine learning hybrids significantly outperform the baseline algorithm.

Acknowledgements

I would like to thank Dr. Antonio Bolufé-Röhler for agreeing to supervise me for this project. I had no experience in Machine Learning, and yet he agreed to allow me to work on this interesting project. More, his patience with me as we worked through this project was impressive. I am grateful for the guidance and support provided throughout the project, as well as the thoughtful feedback. The experience gained through this opportunity was wonderful, and I am thankful that I had the opportunity to work on this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A heuristic is based on experience and practicality rather than directly from mathematics. It provides rule(s) for algorithms that allow them to narrow the problem space down to a more accessible level; but heuristics are not very interchangeable, as they are problem-specific [1]. Metaheuristics, on the other hand, are more general and thus more versatile. This allows them to be used in a wider variety of problems, without the work that would be required in creating new heuristics for every new problem. While in some cases they might not have the success of a specific heuristic given the problem said heuristic was designed for, they are still time-savers, and very useful based on their wider applicability [2].

The hybridization of metaheuristics with machine learning techniques is an emerging research field that is frequently referred to as learnheuristics. These algorithms are "hybrids", this is, a mix of machine learning and heuristics, and so the algorithms themselves are altered as the system learns more about the problem being solved [3]. While heuristics have static values for some design decisions and rules, the learnheuristics can modify these rules as "the situation changes". A change in situation may be considered as the change in topological properties of the function being optimized, as the metaheuristics transits from one region of the search space to another.

An example of this is the use of an adaptive rule for changing the minimum step of Thresheld Convergence (TC) when applied to an Estimation Multivariate Normal Algorithm [4]. In this metaheuristic a simple transition rule based on if-then-else condition was used to adapt the γ parameter of TC. The γ parameter controls the convergence rate of the metaheuristic and thus indirectly its exploration. Despite its simplicity, this approach leads to a significant improvement of the algorithm.

1.1 Research Goals

Specifically, our aim is to train machine learning models and use those to replace the if-then-else rule currently guiding the changes in the γ values. Sequence prediction is a problem that involves using historical sequence information to predict a given output. In our case, the sequence will contain information from different iterations of EMNA-TC.

In the course of this research, we will test the hypothesis that: "provided with enough information a machine learning model will perform better than a hard-coded heuristic rule". In order to effectively test our hypothesis, we will need to answer these two questions:

- What information is relevant for a adjusting the γ parameter; and how to build a data set of training examples?
- Which machine learning techniques are the most effective for adjusting the γ parameter?

We will address the first question by gathering different subsets of attributes characterizing the objective function and the current state of EMNA-TC. We will then create different datasets for training the networks and perform an experimental analysis to determine which dataset yields the best results.

To answer the second question, we will test different machine learning architectures and algorithms. A good starting point will be using the existing algorithms that are known to work well with sequential data.

1.2 Overview

This report begins with some background information on the relevant aspects of the experiments; such as information on the various model types, EMNA-TC, and the benchmark functions used. This will be found in Chapter 2.

After that, there will be a discussion on creating the datasets used to train the models. There were five overall dataset types (only four of these were used in the hybridization phase) with three of them each having six subtypes. How these were created will be discussed in Chapter 3.

After collecting and preparing the data, the models need to be trained and tested based on a portion of the available data. This will be discussed in Chapter 4.

Once the models have been trained and evaluated, a subset of the available models will be evaluated by placing them into use; this is, replacing the if-then-else rule in EMNA-TC. This portion, which includes an analysis of the optimization results of the EMNA-TC learnheuristic, will be discussed in Chapter 5.

Chapter 6 presents a discussion and conclusion with promising future research lines arising from this project.

Please note that the writing of this Thesis and the paper (A Data-Centric Machine Learning Approach for Controlling Exploration in Estimation of Distribution Algorithms) [5] overlapped, and so there will be significant overlap in the material discussed and the interpretations. If there are shared images, tables, or algorithms, they will be cited as originating from that paper.

Code and some extra information (both for [5] and this Thesis) can be found at a specified GitHub repository (https://github.com/jmpluke/EMNA-TC-ML-Hybrid) [6].

Chapter 2

Background Information

2.1 Heuristics

According to Levitin a heuristic is based on experience and practicality rather than directly from mathematics. It provides rule(s) for algorithms that allow them to narrow the problem space down to a more accessible level, but that heuristics are not very interchangeable, as they are problem-specific [7].

From Fox's article we can see that an algorithm may do complete searches to find solutions to problems when this might not be necessary or practical. Such as when the search space is too expensive to completely search. Heuristics provide limits for the algorithm saving time, processing power, and memory (in some cases). Heuristics can help to guide algorithms in how to find potential solutions. This comes at the cost of complete searches. An algorithm utilizing a specific heuristic will be in the position where it is a possible (often likely) that not all solutions may be represented, but only those closest to the heuristic function's best case (or better than best case) scenario, depending upon the heuristic applied, or only those that it was able to find, given the heuristic guidelines, within a time limit (if a time limit was present) [8].

In chapter 3.5 of Artificial Intelligence, we can see that an idea behind having an algorithm guided by a heuristic, can include having a cost comparison. The heuristic function can provide a cost that indicates an optimistic target from the current position to the end. This cost can be measured in numerous ways, but depends upon the problem. In seeking the shortest distance between two points (say between two buildings in different parts of the city) the cost can be measured in time travelling, distance travelled, or possibly even monetary cost for means of transportation. It is important to note, though, that the heuristic function should be providing an optimal (or even

better-than-optimal, given realistic conditions) cost, when measuring the cost to get to the goal, from the goal, the cost should be 0. If the heuristic is not providing an optimal (or even an under-estimated) cost, the algorithm may fail [9].

2.2 Learnheuristics

Calvet et al. take time to describe learnheuristic as a combination of techniques. They allow for dealing with problems that have dynamic inputs. These algorithms will actually cause changes in the inputs over time. As the solution is being formed, the changes in the solution cause changes to the inputs. These updates are not random, but are based on the logic in the heuristic(s) in place in the learnheuristic algorithm. The reason that a learnheuristic is able to do this is because it is (what they call) a "hybrid". This is a mix of machine learning and heuristics, and so the inputs are altered as the system learns more about the situation. Many problems in the real world are dynamic ones. As a method is being implemented, the situation changes. Learnheuristics help in simulating that. While the heuristic estimates mentioned earlier have static values, the learnheuristic's values change as the situation changes. With that, the estimates are able to come closer to the reality of the situation. If these are used in a model, it helps to get more accurate models from the situation/solution [10].

2.3 Metaheuristics

Memeti et al. show that many problems that require heuristics (or that are solved using heuristics) are computationally-expensive problems. The other important note is that a particular heuristic is designed based on a given problem. One can't take a heuristic and apply it to a wide variety of problems. Meta-heuristics, on the other hand, are useful in a much broader sense, application-wise [11].

Memeti then explains that the meta-heuristics are more versatile, as they are more general. This allows them to be used in a wider variety of problems, without the work that would be required in creating new heuristics for every new problem. They are less specialized, and so have less specific information about the problem they are applied to work on. While they might not have the success of a specific heuristic given the problem said heuristic was designed for, they are still time-savers, and very useful based on their wider applicability [11].

Calvert et al. explain that while metaheuristics have some short-comings (such as that they may not find the most optimal solution), they are useful tools in that they find solutions, and they do so quicker than is typical without using metaheuristics (or heuristics). Like heuristics, they are algorithms used to help solve problems, but unlike heuristics, they are not as specialized and are more widely-applicable. Of course, one still needs to use an appropriate metaheuristic for a given problem, as they are more general than heuristics but still not universally applicable [10].

2.4 Classification Machine Learning Algorithms

Within this section there will be a discussion of the various general (classification) machine learning algorithms that were used in this project. The hyperparameters used will be detailed when discussing model training.

2.4.1 Logistic Regression

This is a linear classification model. It determines the probability of various outcomes using a logistic curve. It is suitable for problems such as multinomial logistic regression, binary classification, and one-vs-rest classification problems (which is one way to view our problem). The probabilities for each of the potential outcomes is calculated by the model with a logistic function. [12, 13].

Like the name implies, this is a regression algorithm. Unlike what the name implies, that regression algorithm is used to solve classification problems. It does a series of probability calculations in order to determine how probable each class is given the vector (input information). A calculation takes place (given an input) and the log of the probability of a single class over the probability of a set class is calculated for each class. Each potential class will receive a probability of it being the correct solution, and those probabilities are compared and the highest is selected. [14]

2.4.2 Linear Discriminant Analysis

LDA is a classifier with a distinct decision boundary. Unlike QDA, for LDA the decision boundary is linear. The formation of this boundary (through Bayes' Rule) determines the location of the classes based on the Guassian densities of the learning values. The intention behind LDA is to transform the data into its linear representation, where the boundaries between the class densities become clear. The dimensionality of the data is reduced (in a supervised fashion) in the process. The process is meant for multi-class

data, which is where our data falls in, two different types of solvers (SVD and LSQR) may be used. SVD (Single Value Decomposition) is the default solver for LDA. Unlike other solvers, with SVD there is no covariance matrix calculation, which makes this a more efficient option for datasets with many features (such as ours). LSQR (least squares solution) is specific for classification problems, and requires covariance matrix calculations [15, 16].

In LDA, the boundaries have (as mentioned above) linear aspects, but aspects are important here, as the actual boundaries may be made of a number of linear segments (piecewise linear), given multiple classes. There may be a single linear divide between two classes, but looking at the whole, one will get the image of multiple linear segments making up boundaries. As before, probabilities are calculated by the model. Here they use Gaussian probabilities with an assumption of a shared covariance matrix among all classes [14].

2.4.3 Quadratic Discriminant Analysis

QDA is very similar to LDA. The main difference is in the decision boundaries. While in LDA, the boundaries are linear, in QDA those boundaries are quadratic. Also, while LDA has multiple available solvers, QDA only has SVD available (And therefore no covariance matrix calculations). QDA is considered to have greater flexibility than LDA, due to the ability to learn quadratic boundaries, rather than being limited to linear boundaries [16, 17].

As in LDA, the probability calculations are based on Gaussian densities. This may be seen as a more reasonable fit to many datasets (compared to LDA) simply given the greater flexibility in the boundaries. Having piecewise quadratic boundaries rather than linear boundaries allows for more complex relationships [14].

2.4.4 Stochastic Gradient Descent

SGDs are methods for modeling linear classifiers/regressors using loss functions. SGD itself, is a fairly general classification, the distinction between the types of SGD learning comes with the chosen loss function. Luckily, SGD, is itself, easy and efficient to use. Unfortunately, there have been known scaling issues, and there are a fair number of hyperparameters to consider. Five different loss functions were tested: hinge, log, modified Huber, squared hinge, and perceptron. The hinge variant is similar to a support vector classification. Log is a logistic regression. Modified Huber is useful for outliers

as well as providing probabilities. Squared hinge is similar to hinge, save that it utilizes quadratic penalization. Perceptron utilizes a specific algorithm for calculating loss, called a perceptron algorithm [18, 19].

There are times where the algorithm may encounter regions where the slope is small, but there is no zero nearby. This causes slowed convergence. This can be aided through SGD, which will draw from a smaller sample of points [14].

2.4.5 Linear Support Vector Classifier

Linear SVC is a multi-class classifier typically using the one-vs-rest strategy. LinearSVC does not calculate probabilities in the same fashion as many other methods. Instead, LinearSVC uses cross-validation, which proves to be a fairly expensive option. These methods are typically well-regarded in situations with a large number of features, but relatively few observations (though this requires utilizing carefully chosen model options). It limits the accessed memory through training subsets. The Linear SVC uses a linear kernel, acting faster than other SVC types [20, 21].

There are interesting differences between support vector machines (SVMs) and a number of other classifiers. For instance, the way that SVMs have a very different setup in their loss functions, including the potential for penalty for classification relative to decision boundaries. These penalties have even been applied to examples that are correctly identified, if said examples have vectors that are within a certain distance from the classification boundary. The principle involved is placing hyperplanes between classes. Each separating hyperplane can actually be seen as three different, parallel, hyperplanes (a separating hyperplane, and an equidistant hyperplane on each side marking class boundaries). The margin (distance between the two exterior hyperplanes) shows the margin distance. There will be cases where points will be on the wrong side of a hyperplane, but that is included in the calculations as slack values. This is not desirable, but allows the algorithm to be utilized in cases where perfect separation is not feasible [14].

2.4.6 Radius Neighbors Classifier

RNN is a classifier that bases the classification on the nearest neighbours to the point in question. The neighbour classification with the highest population claims the point in question. The neighbours are chosen by the points within a given radius from the selected point. In this case, 2600.0 was used, as this was the smallest number (starting

at 40, the radius kept increasing the number until a number was found that worked, in other words it didn't cause errors, for each of the attempted datasets) [22, 23].

2.4.7 K Nearest Neighbors

KNN utilizes the same basic principle as is used in RNN, the only difference is that with KNN there are a fixed number of neighbours to be queried. The closest K neighbours to the query point are examined, and the one class that has the highest representation claims the query point. We also added a distance weight (in our models), so that the closer points had a larger weight assigned to their position than those furthest away (so those closer counted for more than those more distant) [14, 23, 24].

2.4.8 Decision Trees

This type of classifier (also usable for regression purposes) creates supervised learning models through the establishment of rules based on the training data. These models are generally user-friendly (both in ease of use and being able to understand the foundational logic), relatively inexpensive, and they conform well to testing. Unfortunately they can be prone to overly complex (overfitted) models, prone to errors with small variations, issues in trying to test outside of the reference data ranges, and prone to bias issues when the training data has not been created with balance in mind. These models are able to utilize multiple classes, (when using leaf classification) they issue a probability to a requested data sample as to where it should be classified, while in the event that there are multiple classes in which the sample may fit (equally well) this classifier will provide the index of the class that (first) has the highest probability, but the lowest index of those with that probability. For instance, if index positions 2, 5, and 7 all have the highest scoring probability, it would select index position 2 as the option [25, 26].

A tree is built up of a number of nodes. These nodes may be internal nodes, leaf nodes, or the root node. The root node is the start of the tree where any data will start on the path to being classified. Each internal, thereafter, will serve as a branching point, where some characteristic of the example will be tested, and how the data is represented relative to the branching will indicate which branch the example will move down to. This will continue until the example reaches a leaf node, wherein it has been classified. Trees go through a training process until (hopefully) there are no errors found in how a tree classifies the supplied training data. Unfortunately, this is not always possible. There are other stopping criteria present, and there is also the understanding that there may be training examples present with identical (or near-identical) feature vectors, but different classifications. One can also utilize a tuning parameter. When this is utilized,

it will prune the tree by finding a subtree within the tree that has the best possible value in a cost-complexity calculation [14].

2.4.9 Random Forest

This algorithm was a late addition to the project, added in after most of the work was completed.

Random Forests utilize the Decision Tree algorithm as part of the overall classification process. It does this by having a number of trees present. Each tree will only look at a subset of the data's features (one of the hyperparameters to be decided beforehand or tuned) [14].

In some cases, each tree will indicate which classification it indicates the example should fall under, and each tree is given a single vote. The class with the highest votes ends up being set as the class for that input. In other cases, each tree will give probabilities (such as the leaf variety from the Decision Tree section), and those probabilities are weighed in determining which class is used as the output for a given input [27, 28].

2.4.10 Gaussian Naïve Bayes Classifier

Gaussian Naive Bayes is one of a set of algorithms that falls under Naive Bayes. These fall under supervised learning. When it receives a set of features, it functions by assuming that no single feature has an appearance that is dependent on the appearance of another single feature. Each feature receives a Gaussian (normal) distribution probability (for the GNB variant). These models are fairly simple due to the assumption of independence between the features, and so run fairly quickly. They have also been shown to work relatively well as classifiers, if not well as probability estimators [29, 30].

This classifier has a somewhat different look at how it deals with probabilities. Each class receives a score based on a product of m terms (m being the number of features in the dataset). Each feature is examined and receives a probability (the probability of that feature type having that value given a specific class). The class that has the largest probability product can be selected (in simple models) to be the class for that input [14].

2.4.11 Neural Networks

Neural networks are a network of units (neurons). These neurons have an activation function (such as Sigmoid, Symmetric Sigmoid, and Rectified Linear Unit) and receive inputs with weights assigned to them. A network may have many layers (an input layer that receives the features, interior/hidden layers that receive inputs from the previous layer, and an output layer which in classification problems will have the same number of output neurons as potential classes. When considering feed-forward operations; neurons (other than inputs) receive as inputs the outputs of the previous neuron layer and their associated weights [9, 14].

The neural network trains in such a way as to attempt to limit its loss. This not only involves feeding information forward, but also utilizing back-propagation. The loss is calculated. That loss calculation is used in the algorithm's gradient descent. Iterations of feeding forward through the network and back-propagation continue for a set period (too long, especially with too few training examples may cause over-fitting) allowing the weights to be adjusted [9, 14].

2.5 Regression Machine Learning Algorithms

While only used in one phase of the project, and not used in the final portions of the experiment (and so not involved in the conclusions section); we also used a small number of regression models.

2.5.1 Linear Model

In a given dataset, not all of the features may be relevant. Some models may train in such a way as to determine that certain features aren't important to the end result and so are provided with a coefficient of 0. The Lasso Linear Model is one that utilizes this strategy. It will often shift toward solutions with multiple 0-valued coefficients, making the end result dependent on fewer features. The function (for an input) turns into the product of the feature values vector with the coefficient matrix plus a constant [31].

2.5.2 Ridge

Ridge regression applies a penalty based on the values of the weights. This is done through an alternative to the sum of squares. The end result is the function (for an input) turns into the product of the feature values vector with the coefficient matrix

plus a constant. Like the other regression examples mentioned so far, your end equation will look something like: y = a_0 + a_1x_1 + ... a_nx_n Where a_i is the coefficient for the i-th x term and a_0 is a constant [32].

2.5.3 RidgeCV

RidgeCV is simply Ridge Regression which automatically also utilizes cross-validation [33].

2.5.4 Bayesian Ridge

Bayesian Ridge is an adaptive regression model. It can include a regularization procedure as well as calculating certain parameters for use in training the model. This is important as it is a probabilistic model that works by assuming a normal distribution. Instead of a given output being a certain point, the output is seen to exist within a spherical probabilistic space. It is adaptive but also very time-consuming [34–36].

2.5.5 ARDRegression

Automatic Relevance Determination (ARD) Regression is very similar to Bayesian Ridge. They assume different probabilistic spaces (elliptical for ARD). The end result can have very small coefficients. These weights are calculated in an iterative process by calculating each weight's associated precision distribution as well as the noise's precision distribution [37, 38].

2.5.6 HuberRegressor

The Huber Regressor can be a reasonable choice when dealing with datasets that may have outliers. While it does not do any feature scaling (this would need to be done before training if it is going to be done). There are thresholds given over the samples. If the sample's error is outside of said threshold, it can be deemed to be an outlier. The loss function that it utilizes varies based on whether the example is deemed to be an outlier. Within the threshold, the value from the loss function is more heavily dependent on the error, if an outlier, that weight from the error is reduced by placing more weight on an introduced parameter (epsilon, which is given by the user, though SKLearn suggests using 1.35) [39, 40].

2.6 The Optimization Benchmark

In an effort to advance the development of optimization algorithms, it helps to improve the understanding and utility of simpler optimization algorithms (in this case those that have just one objective). In service to that, there have been benchmark functions published to test algorithms upon. The CEC'13 version of those benchmark functions were used in the last stage of testing for this project, as well as in the initial data collection. This suite contains 28 functions (5 unimodal functions, 15 multimodal functions, and 8 composition functions). [41] This encourages blackbox thinking in algorithm preparation, rather than specializing on single problem types. These functions were released in the CEC 2013 Special Session as a Test Suite [41].

2.7 EMNA-TC

EMNA-TC or "Estimation Multivariate Normal Algorithm with Threshold Convergence" [42] is an Estimation of Distribution Algorithm (EDA) that utilizes Threshold Convergence. While EDAs will determine potential future solutions through the use of a distribution function (which is designed by the algorithm based on a portion of the solutions currently available for the current generation), they may have a tendency to converge too quickly, missing more desirable solutions [41].

The TC (Threshold Convergence) portion allows this to restrain the algorithm from converging too quickly (delaying exploitation by controlling the search distances, and their continual decay, to allow for more, hopefully fuller, exploration), forcing a threshold to be met before convergence is allowed. Unfortunately, a typical method of directly enforcing a distance between solutions showing a descendent relationship is not a method viable with EDAs, as such a relationship does not exist with EDAs. So instead, the TC is utilized within the parameter space, where the norm of the covariance matrix will be set. That, in turn, allows for the desired distance between solutions [41].

With this, the search distance is controlled by the parameter space changes, and so does not decay too rapidly, and won't reach the endpoint early. The γ parameter (what our project manipulates through machine learning) is adjusted within this algorithm based on the quality of the solutions found. In the event that more promising solutions are being identified, γ is decreased so as to allow more exploration, to ensure that desirable areas aren't missed. In the event that the solutions aren't representing an improvement, γ can be increased allowing for convergence [41].

2.7.1 Thresheld Convergence

Part of the purpose behind Thresheld Convergence involves a separation between exploration and exploitation. Exploration involves gathering samples from different attraction basins. The ability to do that requires having a sufficient threshold size to allow broader sampling. The next phase (exploitation - where there is convergence on local optima) does not have the same threshold requirement [43, 44].

Within Thresheld Convergence is the idea of minimum distance. This minimum distance is applied between potential solutions and their reference solution. With that in play, there is a more limited chance of oversampling a local basin rather than wider sampling. This distance is controlled by the algorithm's decay rate (the output generated by the models in our experiment), and is decreased as the algorithm progresses through exploration toward exploitation [45].

From [44, 46] we can see that the distance is important because there are times when local optima are relatively close to a reference point. When an algorithm is allowed to move to exploitation without sufficient exploration, there may be bias introduced based on those proximate optima. When a local attraction basin is over-sampled, due to exploitation, this may skew the results. If other basins have fewer solutions being sampled than the initial basin, that initial basin has a higher chance of having a better solution sampled (for example, if you have two equal attraction basins with several random samples from one, but only one from the second, you are more likely to find a better solution from the heavily-sampled basin). The heavier-sampled basin may not have the best solutions, but the fact that it is so heavily sampled (comparatively) means that the probability of one of its better solutions being sampled increases over that of other basins with similar proportions that are less-heavily sampled. With wider exploration, if the algorithm randomly samples all of the attraction basins equally, it is more likely that the collected samples will be more representative of the quality of the basins. Having that minimum distance between reference and sample allows for fuller exploration. This causes a delay in exploitation until there is more confidence in exploration results, and so it is more likely that exploration has found basins with better solutions. When it is time to shift into exploitation, it is seen as more likely that the intended solutions will be found [44, 46].

Chapter 3

Creating the Datasets

3.1 From Previous Work on the Algorithm

EMNA-TC utilizes Thresheld Convergence (TC) within the parameter space. The threshold function provides a value that, in turn, is utilized in order to control the covariance matrix Σ within the distribution function. EMNA-TC is an iterative algorithm. It starts with (and evaluates) an initial population, and then moves into iteration mode (bounded by a limited number of function evaluations). Within a cycle of an iteration it will select the best solutions that it has found from within the population. In the first iteration, it will set the threshold by normalizing the initial Σ found in the initial population. If it is not the first iteration, the process will check to see if the current best solution has improved. If it has, γ can be decreased (replaced with the product of the current γ multiplied by 0.95) else it is increased (replaced with the product of the current γ multiplied by 1.05). At that point, the threshold is updated using a calculation including γ, the covariance matrix is updated, and a new population is gathered. Assuming that the iterations haven't been exhausted by exceeding the number of function evaluations, the cycle starts over again [42]. See Algorithm 1 from [5].

In this project, we have modified how γ changes. Instead of relying on a fixed calculation to determine what the next γ will be, we use a machine learning model to predict (update) the best value for γ as the optimization proceeds. For that we create a vector based on the current state of EMNA-TC, and then the set up allows the algorithm to use that as an input for a machine learning model, and have that model indicate what the next γ should be. In order to do that, we had to build and train models to determine which γ would be most ideal under a variety of situations.

Algorithm 1 EMNA-TC

Randomly initialize population P
while $FEs \leq maxFEs$ **do**
  $best\_solutions \leftarrow select\_best(P)$
  $[\mu, \Sigma] \leftarrow estimate\_parameters(best\_solutions)$
  if first iteration **then**
    $\Sigma_{norm} \leftarrow init\_\Sigma_{norm} \leftarrow norm(\Sigma)$
  else
    if best solution has improved **then**
      $\gamma \leftarrow 0.95 \times \gamma$
    else
      $\gamma \leftarrow 1.05 \times \gamma$
    end if
    $\Sigma_{norm} \leftarrow init\_\Sigma_{norm} \times \left(\frac{maxFEs - FEs}{maxFEs}\right)^{\gamma}$
  end if
  $\Sigma \leftarrow \Sigma_{norm} \times \frac{\Sigma}{norm(\Sigma)}$
  $P \leftarrow sample\_normal\_multivariate(\mu, \Sigma)$
end while
return $best\_solution$

3.2 Collecting the Data and Building Examples

Building training examples for this problem is a very expensive process. It was determined that γ values would range from 0.0-4.0, and these were placed in increments of 0.5, thus we had 9 output classes (when using categorical models). These classes could be considered to have the whole number values 0-8, and the corresponding γ values being the class index divided by 2.0 (eg. $0.0/2.0 = 0.0$ and so class 0 corresponds to $\gamma = 0.0$; $1.0/2.0 = 0.5$ and so class 1 corresponds to $\gamma = 0.5$; up to class 8 corresponds to $\gamma = 4.0$).

The training examples came from running a slight variant of EMNA-TC. A training sample has a set γ of 3. There is a randomized value selected (said value corresponding to a point somewhere between 25% and 95% of the available EMNA-TC iterations (calculated by function evaluations). Once that value has been reached, the example will branch out into 9 branches (one branch for every potential γ value. At that point, the algorithm will continue to run on each branch, and the end value will be returned, and all of the values from the branches will be collected. The features collected with such an example include values from the last 200 iteration leading to a branch point (only every second iteration includes data collection for features, and so instead of 200 entries for each feature, each feature only has 100 entries). These features include the number of function evaluation used up to the collection point, the number of evaluations still available, the current γ, the current threshold value, the fitness standard deviation of the population (at that point), the fitness average of the population (at that point), the fitness percentiles (0%, 10%, ..., 100%) of the population (at that point).

3.3 Initial Data Preparation

3.3.1 Variant Type V0

When the training examples were collected, they were put through an elimination process. If a training example had more than one branch with the same 'best' value, the entire training example would be discarded. All other training examples were retained. This sets up the base for Variants 1, 2, and 3, while being kept as-is for Variant 0 (V0).

After finding the base, each of the three non-0 variants were also issued percent subtypes. These subtypes started at 1% and incremented by 2% up to 11% (by which point we had relatively few training examples remaining in V1 and V3). With that you would, for instance, have V1p1, V1p3, ..., V1p11 from the variant 1 type. That percentage indicated the minimum relative distance between values. What that means changes based on the basic variant type.

3.3.2 Variant Type V1

V1, like V0, utilizes an elimination strategy. Beyond the eliminations already dealt with at the base (V0) stage, it also eliminates any training example where the relative difference found between the best fitness branch and the worst fitness branch is less than the given percentage. For example, if a training example for V1p1 had the best and worst fitness values within 1% of each other, then that example would be discarded.

3.3.3 Variant Type V2

V2 is the sole variant type that does not discard examples (beyond the base discards explained in V0). The median index value was 5, and so the median γ value was 2.5. With that in mind, whenever we had a sample that would have been eliminated by its V1 counterpart, we replaced the γ that, up to that point, was considered the best for that example with 2.5 and kept the example. This kept more examples (one can see that there was significant loss of examples in V1 and V3 as the percentages increased).

3.3.4 Variant Type V3

V3 is similar to V1. This variant type employed an elimination strategy, but it also utilized a reduction in classes. While V0, V1, and V2 all had nine classes, V3 only had three (0.5, 2.0, 3.5). Instead of selecting the best γ from an individual branch, it

averaged the fitness values from three branches for every remaining γ value (so γ=0.5 had branch values averaged from $\gamma = 0$, $\gamma = 0.5$, and $\gamma = 1.0$). The remaining class that had the best average fitness would be selected as the class for the example, in the event that the example survived the comparison and elimination procedure. Once the example had been reduced to three classes, their values were compared, and if the best and worst were within the percentage of one another, the sample would be discarded. Due to this, V3 had significant example loss.

3.4 Distribution of Classes in Datasets

With our data preparation procedures, we did not have the same number of training examples (see Figure 3.1) or even distributions of either γ type (see Figure 3.2, more detailed version of figure found in [5]), nor in CEC '13 Function examples (more on that last part later).

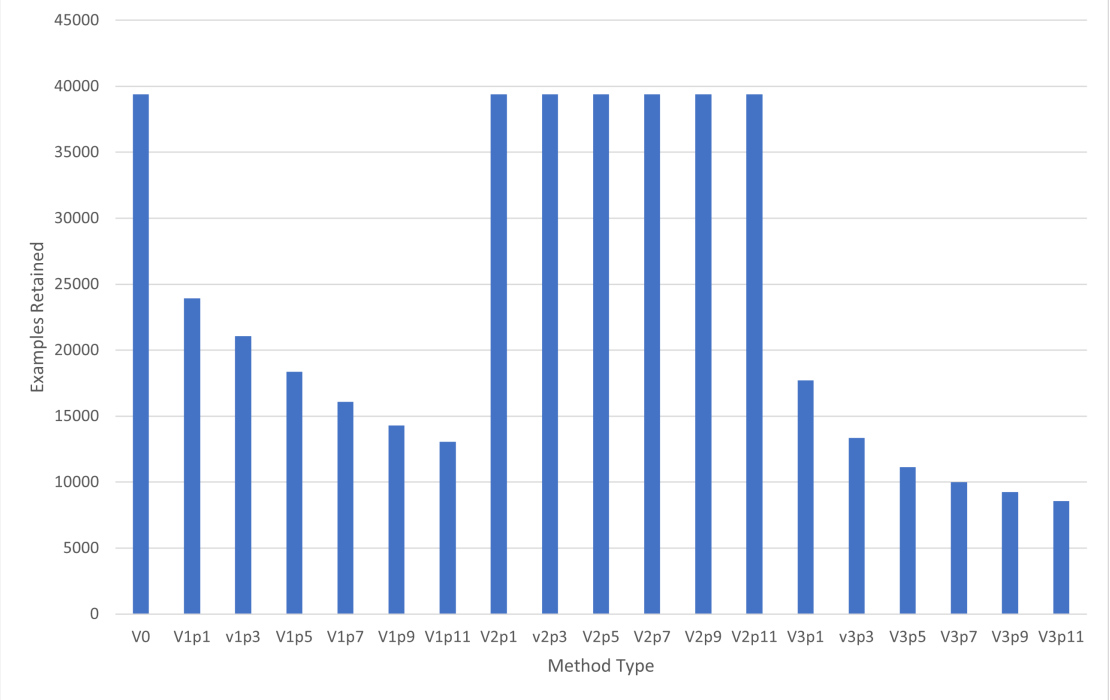

FIGURE 3.1: The Number of Examples Retained for Model Training/Testing for the Preparation Methods

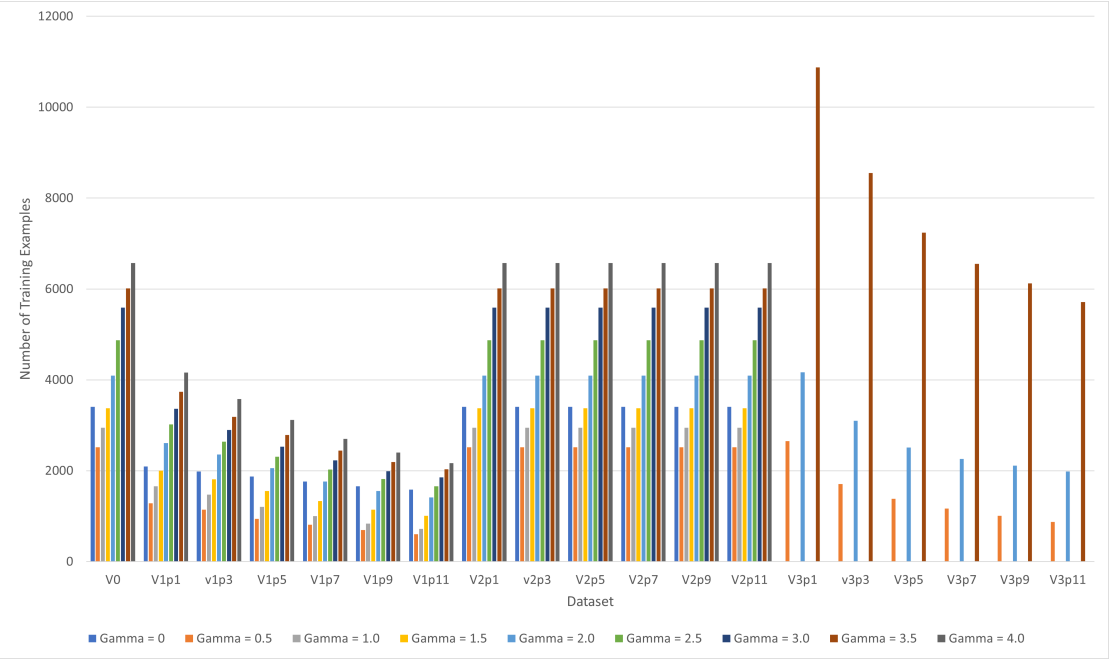

FIGURE 3.2: Displayed use of γ with Preparation Methods

3.5 Example and CEC'13 Function Distribution Throughout Variant Subtypes

It was quite noticeable that (for V1 and V3), as the percentage increased, the number of examples decreased (see Figure 3.1). V0 and the V2 type, included all of the examples (39382). V1 ranged from 23930 examples (with V1p1) down to 13041 examples (with V1p11). V3 ranged from 17698 examples (with V3p1) down to 8565 examples (with V3p11). This also carried through to the training examples for particular functions. Here, we can see the number of training examples varied significantly.

V0 and V2 had a static set (as far as which functions were represented), but not evenly distributed (See Figure 3.3). F3 and f15 both had 1900 training examples for V0 and V2. Some functions such as f6, and f14 were also very close to that mark. Others, such as f25 (978) and f27 (999) still had significant showings, but not at the same level as f3. At the low end, f1 only had 272 examples present in V0/V2.

F3 and f15 also had a strong start with V1, both starting with the same 1900 training examples in V1p1 (See Figure 3.4). F27 remained with a decent-sized 835 (compared to V0's 999) for V1p1. Unfortunately, things changed significantly after that. F25 (978 with V0) only showed 28 examples for V1p1. F8 and f20 both had 0 training examples throughout V1, while f16 had just 6 for V1p1, and 0 for the rest of V1. By V1p11; f3, f14, and f15 had close to 1900 each, f22 and f23 had over 1400 examples, while 11 other functions had less than ten examples (nine of which had no examples present).

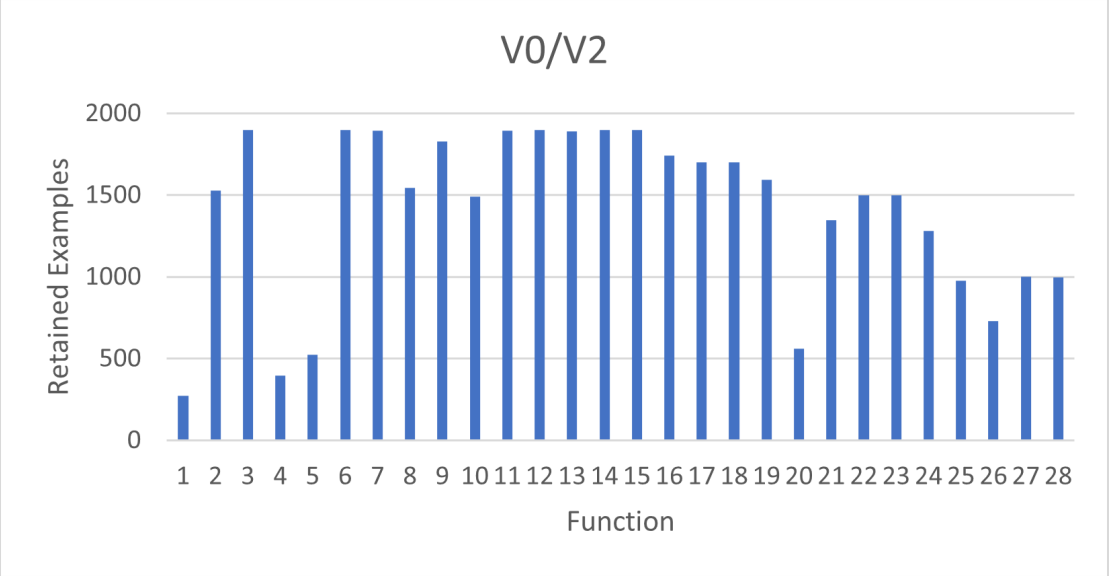

FIGURE 3.3: Quantity Retained for Each Function in V0 and V2 Variant Types

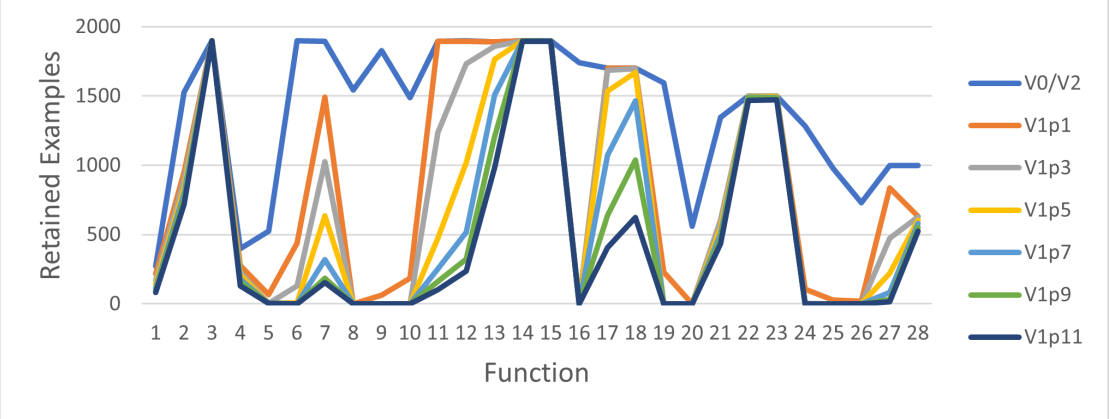

FIGURE 3.4: Quantity Retained for Each Function in V1 Variant Types

F14 and f15 had relatively strong starts for V3 (1591 and 1579 examples respectively for V3p1), while f2 and f11 (867 and 859) showed a fair number of examples (See Figure 3.5). F8, f16, and f20 each started V3 with no examples, and were joined with f9, f10, f19, f25, and f26 where in V3p3 they also had no examples remaining. When we reached V3p11 thirteen functions had fewer than 10 examples, 9 of which had no examples. This while f3, f14, and f15 still had over 1000 examples.

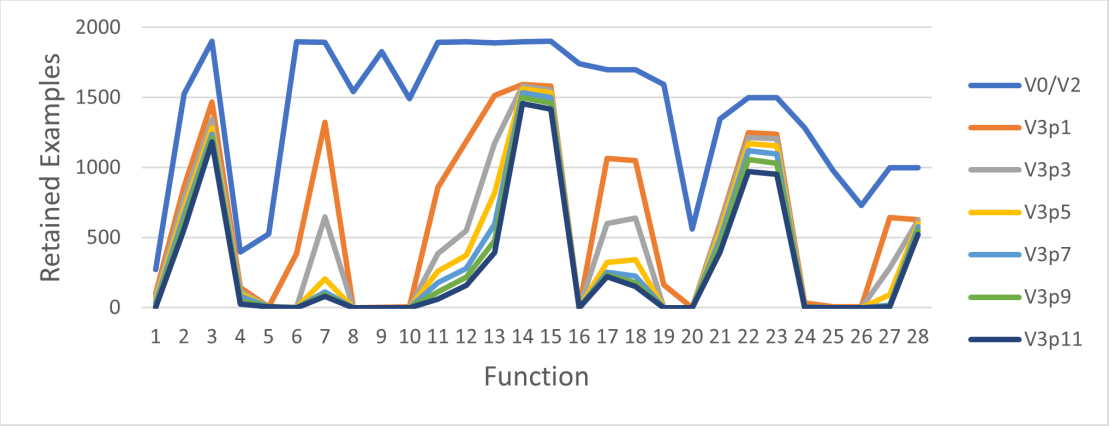

FIGURE 3.5: Quantity Retained for Each Function in V3 Variant Types

Chapter 4

Training and Initial Testing of Models

4.1 Basic Model Information

When referring to preparing models in this section, there will be a reference to the basic model's name (detailed in the background section), and some parameters. The rest of the model's parameters should be considered to be the default parameters for that model type. During the course of the experiment, there were a number of machine learning algorithms employed. Many Scikit-Learn (SKLearn) model types were utilized [47], as well as a number of TensorFlow Keras models [48, 49]. This is a list of said algorithms along with the hyperparameters used:

Discrete SKLearn models:

- Logistic Regression Classifier using the 'sag' (stochastic gradient descent) solver and a maximum of 10000 iterations
- Logistic Regression Classifier using the 'sag' solver and a maximum of 100000 iterations
- Logistic Regression Classifier using the 'sag' solver and a maximum of 1000000 iterations
- Logistic Regression Classifier using the 'sag' solver
- Linear Discriminant Analysis classifier (default)
- Linear Discriminant Analysis classifier with 'lsqr' (least squares solution) as the solver

- Quadratic Discriminant Analysis classifier (default)
- Stochastic Gradient Descent classifier with a maximum of 10000 iterations
- Stochastic Gradient Descent classifier with a maximum of 10000 iterations and a logarithmic loss option
- Stochastic Gradient Descent classifier with a maximum of 10000 iterations and a modified huber loss option
- Stochastic Gradient Descent classifier with a maximum of 10000 iterations and a squared hinge loss option
- Stochastic Gradient Descent classifier with a maximum of 10000 iterations and a perceptron loss option
- Linear Support Vector Classifier with a tolerance of 1 e -3 and a maximum of 10000 iterations
- Nearest Neighbors (Radius) classifier using a radius of 2600.0 (This number was chosen to be consistent throughout the entire study. Some of the datasets worked with a much smaller radius, but others had examples wouldn't work until the radius was increased to this point), and an outlier classifier of most frequent
- Decision Tree Classifier (default) Gaussian Naïve Bayes classifier with priors given None
- Linear Support Vector Classifier with a tolerance of 1 e -3, a Crammer Singer multi class option, and a maximum of 10000 iterations
- Nearest Neighbors (K) with weights set to 'distance' and the number of neighbors set to 20

Continuous SKLearn models:

- Linear Model with Lasso regularizer, alpha set at 0.1, and a maximum of 100000 iterations
- Linear Model with Lasso regularizer, alpha set at 0.01, and a maximum of 100000 iterations
- Linear Model with Lasso regularizer, alpha set at 0.5, with fit intercept set to false, and a maximum of 100000 iterations
- Linear Least Squares Model with Ridge regularizer, alpha set at 1, with fit intercept set to false, and a maximum of 100000 iterations

- Linear Least Squares Model with Ridge regularizer, alpha set at 0.1, with fit intercept set to false, and a maximum of 100000 iterations
- Linear Least Squares Model with Ridge regularizer, alpha set at 2, with fit intercept set to false, and a maximum of 100000 iterations
- Linear Least Squares Model with Ridge regularizer (with cross validation), alphas set at [0.01, 0.01, 0.1, 1, 3], with fit intercept set to false
- Bayesian Ridge regression with fit intercept set to false and a maximum of 100000 iterations
- Bayesian ARD regression with fit intercept set to false and a maximum of 100000 iterations
- Linear Regression model with a Huber Regressor with fit intercept set to false and a maximum of 100000 iterations

Continuous Keras models:

- Sequential model with three dense layers with outputs of 64, 32, and 1; an SGD optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)
- Sequential model with three dense layers with outputs of 64 ('sigmoid' activation), 32, and 1; an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)
- Sequential model with three dense layers with outputs of 64 ('relu' activation), 32, and 1 ('sigmoid' activation), sandwiching two 0.1 Dropout layers; an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)
- Sequential model with three dense layers with outputs of 256 ('relu' activation), 128, and 1 ('softmax' activation); an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)

Discrete Keras models:

- Sequential model with three dense layers with outputs of 64, 32, and 9; an SGD optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)

- Sequential model with three dense layers with outputs of 64 ('sigmoid' activation), 32, and 9; an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)
- Sequential model with three dense layers with outputs of 64 ('relu' activation), 32, and 9 ('sigmoid' activation), sandwiching two 0.1 Dropout layers; an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)
- Sequential model with three dense layers with outputs of 256 ('relu' activation), 128, and 9 ('softmax' activation); an Adam optimizer with a learning rate of 0.01; a Categorical Cross Entropy loss function; an accuracy metric (1000 epochs)

The models used were from Scikit-Learn and from TensorFlow Keras. The same twenty-three model varieties (which were made up of the 11 general types discussed in Chapter 2) were used for each of the different data preparations except for the continuous preparations, which required different models. The mentioned continuous models were only used in a single run (using V0 set to train continuous models), after that we focused solely on classification models.

Many of the models from this point forward will be referred to in abbreviated form. These models (models that went on to the hybridization stage) are as follows (with details for those model types that had multiple versions of that basic type used in the classification stage):

- Decision Tree (DT)
- K Nearest Neighbours (KNN)
- Radius Nearest Neighbours (RNN)
- Linear Discriminant Analysis with 'svd' solver (LDA)
- Quadratic Discriminant Analysis (QDA)
- Logistic Regression 'sag' solver and a maximum of 100000 iterations (LOG)
- Neural Network: Sequential Keras model, 5 layers (in order Output Dimensions or Dropout (OD or DO): OD 64, DO 0.1, OD 32, DO 0.1, OD 9), with a 'relu' first activation layer and 'sigmoid' last layer (TF3)
- Neural Network: Sequential Keras model, 3 layers (outputs of 256, 128, and 9) with a 'relu' first layer and 'softmax' last layer (TF4)
- Random Forest (RF)

4.2 Training

The training process for the models was kept fairly simple. The available dataset was split into two parts (training and testing) with 20% of the available examples being set aside for accuracy testing. Each dataset (for instance V1p1) had this split take place once. Once that occurred, each model type had a model built and trained with the dataset's indicated training set. Then each of those models was tested with the dataset's testing set.

4.3 First Model Evaluation

After the models were trained, they were tested (data was collected when they were tested against the test set, the training set, and the entire dataset), with the accuracy, precision, recall score, and number of classes used as evidenced by a confusion matrix all being collected. Some of the collected accuracies are available (see Figure 4.1 and Table 4.1), while more information is available in the Appendices and more still is available in the GitHub repository [6].

During this initial evaluation, one of the differences between the datasets started to show. There was a steep increase in all averages as we looked at the variant 3 models. This was expected as there were only three classes available to the variant 3 models. While there were variations between the accuracy scores of a single model type between datasets belonging to a single Variant type, there was also a fair bit of consistency. Some models did well (in a comparative sense) and others did not. QDA was something of an exception. Most models had some fluctuations in accuracy within a variant, but the fluctuations with QDA were significant. As you can see from (Figure 4.1), most models were remained close to an initial accuracy within a variant type (V1 and V2) and many saw increases in accuracy as P increased in V3, QDA was very inconsistent, especially in V1 (where accuracy climbed) and V3 (where accuracy rose, fell, and rose again with a range of 0.1425, the next highest V3 range was found in TF3 and TF4 at 0.069096). While the neural network models were selected to undergo hybridization testing, they did not score as well in accuracy testing as models such as RNN and DF (which was a late addition).

Throughout the training and testing procedure, there were a number of the classification models that failed to meet the null accuracy (see Table 4.2, and Figure 4.2). In some cases, those selected for use in potential hybrids found themselves among those that failed to reach that mark. For instance, in V0 QDA, KNN, and TF4 failed to reach the null accuracy. Looking at the accuracies from V0 through to V3p11, there ranged from

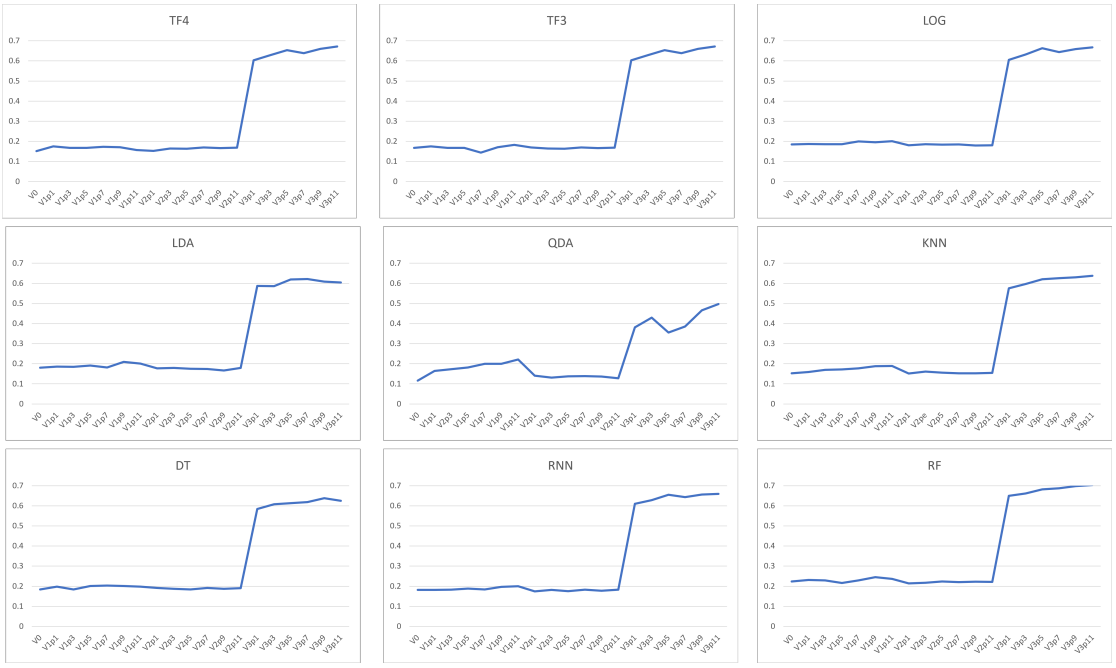

FIGURE 4.1: Averages of the Model Types Selected For Hybridization

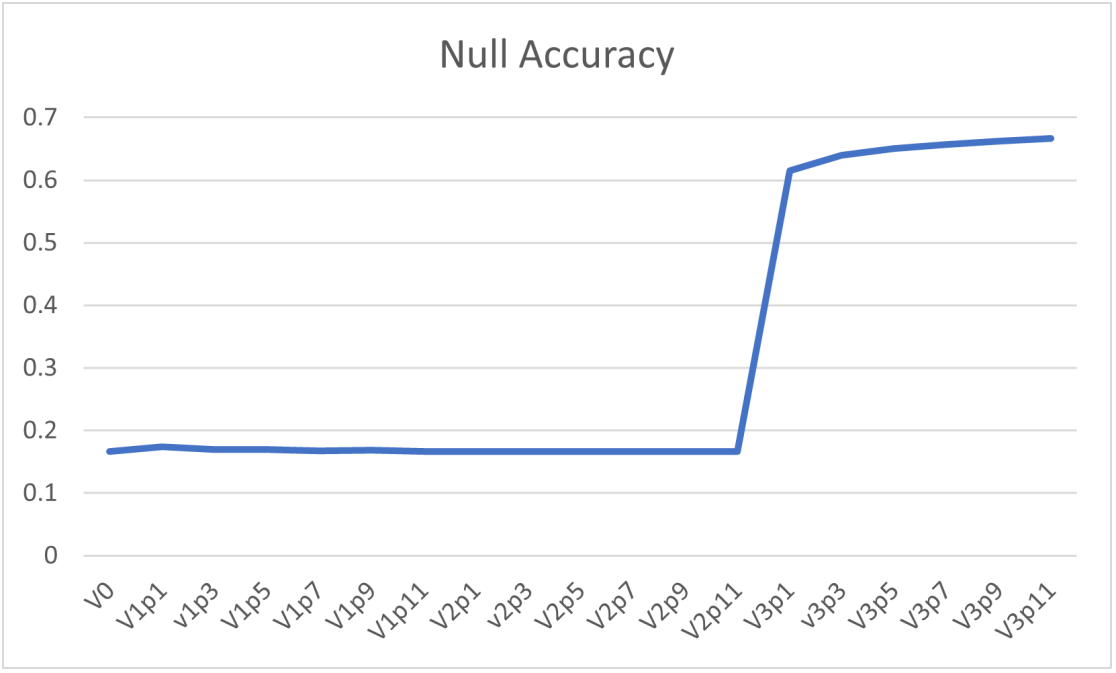

FIGURE 4.2: Null Accuracy by Dataset

7 to 22 (average of 13.4 per dataset) classification models that failed to match the null accuracy. Of those, between 0 and 8 of them (average of 3.8 per dataset). It is notable that TF4 (importance of this model type discussed later) found itself failing to meet that null threshold 13 times. It is also notable that the only model that always had an accuracy above the null average was the Random Forest model.

TABLE 4.1: Variant Type Average Accuracies By Model Type

Model	VO	V1	V2	V3
RNN	0.182	0.189	0.179	0.642
TF4	0.151	0.168	0.164	0.643
KNN	0.153	0.176	0.155	0.615
TF3	0.167	0.168	0.167	0.643
LOG	0.185	0.192	0.182	0.645
DT	0.184	0.198	0.189	0.615
RF	0.224	0.231	0.220	0.680
LDA	0.180	0.192	0.175	0.605
QDA	0.116	0.190	0.136	0.419
Average	0.168	0.179	0.169	0.637

TABLE 4.2: Number of Models Within Each Dataset Training Group that Failed to Meet Null Accuracy

Dataset	Classification Models Below Null	Hybrid Models Below Null
V0	10	3
V1p3	13	4
V1p1	12	2
V1p5	12	2
V1p7	7	1
V1p9	8	0
V1p11	11	1
V2p1	12	3
V2p3	12	4
V2p5	13	4
V2p7	11	2
V2p9	13	4
V2p11	11	2
V3p1	22	8
V3p3	22	8
V3p5	12	4
V3p7	22	8
V3p9	21	8
V3p11	11	5

Chapter 5

Hybridization

5.1 Machine Learning Hybrids based on EMNA-TC

The Nine model types mentioned in Chapter 4 were selected for hybridization: RNN, KNN, DT, QDA, LOG, LDA, TF4, TF3, and RF. Many of these were chosen because they had relatively strong showings in accuracy, precision, and confusion matrices. TF4 and TF3 were the two strongest of the Tensorflow models, LOG showed strong accuracy but inconsistencies elsewhere, QDA was of interest because of its very irregular showing in initial testing, and RF was selected last with the strongest accuracy showing of any model tested (across all datasets).

In order to test how well the hybrids function, a new base code was written. To start out with, the EMNA-TC-ML hybrids begin in the same way that normal EMNA-TC would, the only difference is that it regularly collects the data required to create vectors to use as inputs for the machine learning models. The hybrid algorithm will proceed as a normal EMNA-TC algorithm until it has used up 10% of the available number of function evaluations. At this point, it stops altering γ through use of the normal EMNA-TC method, and instead relies on the γ produced by the linked machine learning model. Every 10 iterations, γ is updated by feeding an updated vector into the model and replacing the current γ with the new one. In the interests of ensuring convergence, this process is halted when there are only 30% of the function evaluations left. At that point, γ is changed to 4 (the highest γ value available from the models), and it remains there throughout the rest of the process. This higher γ is a means to promote faster convergence.

5.2 Hybrid Testing

In order to effectively test these hybrids, we needed a setup that would allow us to do this. Luckily, such a test set was already available. We were able to test these hybrids using the IEEE CEC '13 Benchmark Suite [41]. This is a set of 28 functions varying from unimodal functions (in functions 1-5), multi-modal basic functions (functions 6-20), and composite multi-modal functions (functions 21-28) [41]. Parameters for the experiments involved each hybrid having 30 independent runs of each of the 28 functions with 30 dimensions, and a set maximum of 300000 function evaluations available. Each hybrid had its results written to file (including EMNA-TC which was also run, but as itself rather than as a hybrid, during the course of this project). Each hybrid (and EMNA-TC) had their average scores for each function calculated and stored. In order to determine the performance of a given hybrid, their function averages were compared to that of EMNA-TC [100(b-a)/max(a,b)] (with the a-value representing the hybrid's and the b-value representing EMNA-TC's average score). With a positive score, the hybrid is outperforming EMNA-TC, a negative score indicates that EMNA-TC is outperforming the hybrid.

Note: relative difference = $$\frac{(EMNA - TC\_score) - (hybridScore)}{(Maximum[EMNA - TC\_score, hybridScore])} X100\%$$

5.3 Hybrid Results

5.3.1 General Results

If one were to set a cut-off of a minimum of 10% improvement (over results found using EMNA-TC) we would indicate that we have had some success in this project. Each model type had 19 models trained (one for each of the classification datasets), for a total of 171 hybrids. Of those, 32 hybrids met that minimum cut-off (1 DT hybrid, 2 KNN hybrids, 4 TF3 hybrids, 8 TF4 hybrids, and 17 RNN hybrids). It is also notable that the two RNN hybrids that missed the cut-off were both above 9%, and so were quite close. Figure 5.1 shows this comparison, but using 0% as the cut-off.

Figure 5.2 shows each model type's Hybrid Performance (relative to EMNA-TC) and Classification Accuracy. In a somewhat surprising turn of events, RNN had the strongest overall showing. It had an average percent difference (compared to EMNA-TC) higher than any other model type. In fact, it significantly outperformed the other Nearest Neighbours classifier (KNN). The two included Tensorflow models also had surprising showings. Neither of them had strong showings in initial testing (in fact their results were nearly identical at that stage), but they did have some strong (if inconsistent) showings

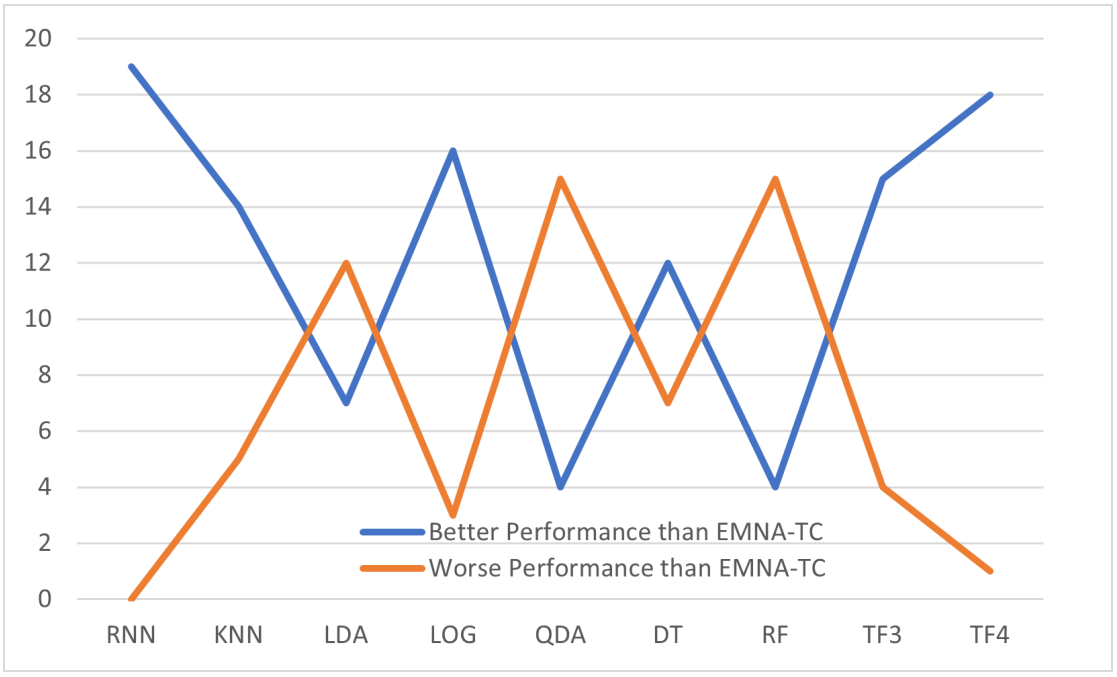

FIGURE 5.1: Number of Models (For Each Model Type) That Exceeded EMNA-TC Average Performance

FIGURE 5.2: Overview of All Models VS EMNA-TC Performance and their Accuracy in Classification Testing

here. While both were among the five model types that had at least one hybrid that had a performance above 10%, they also had at least one hybrid below EMNA-TC performance. This is seen in that TF4 had an overall range of 27.68% (from -4.98% to 22.70%) and TF3 had a range of 32.06% (from -18.87% to 13.19%). TF4 had the overall best hybrid, as the TF4 hybrid trained with the V3p9 dataset scored 22.70% which was better than RNN's highest of 21.67% with the V0 dataset. In another surprising showing, the Random Forest type had a very poor showing at this stage. Random Forest showed the highest accuracy for every dataset, but it showed very poor performance. Not one of its hybrids appeared above 10%, and most of them showed negative values.

In looking at final results, we have already mentioned that there are a number of models that can be used in hybrids that significantly improve upon the performance of EMNA-TC, but let's look at some specifics. Here we will be looking at the two best hybrids RNN (with dataset V0) and TF4 (with dataset V3p9).

5.3.2 RNN Results

In looking at RNN (Table 5.1, results for other algorithms from [42]), it performed better than EMNA-TC in all but five of the CEC '13 functions (f3, f12, f19, f23, f24). There were significant improvements in f1-f18. In a comparison to CMA-ES, DE, and PSO the RNN hybrid also showed significantly better performance. This was very noticeable in the multi-modal ranges (f6-f20 and f21-f28) where RNN showed 62.7% (f6-f20) and 37.5% (f21-f28) better performance than CMA-ES, 67.6% (f6-f20) and 29.7% (f21-f28) better performance than DE, and 63.4% (f6-f20) and 30.2% (f21-f28) better performance than PSO. In all three cases, RNN also had better performance with f4 and better than DE and PSO in f2 within the unimodal functions.

5.3.3 TF4 Results

In looking at TF4 (hybrid using dataset V3p9) (Table 5.2, also found in [5], results for other algorithms from [42]), it performed better than EMNA-TC in all but eight of the CEC '13 functions (f6, f9, f12, f15, f19, f23, f24, and f25). In all eight of those cases, there was less than a 10% difference between the two. In fact the difference (in favour of EMNA-TC) was less than 1% in four of the cases. There were significant improvements in f1-f18. In a comparison to CMA-ES, DE, and PSO the TF4 hybrid also showed significantly better performance. This was very noticeable in the multi-modal ranges (f6-f20 and f21-f28) where TF4 showed 61.31% (f6-f20) and 37.65% (f21-f28) better performance than CMA-ES, 66.09% (f6-f20) and 29.86% (f21-f28) better performance than DE, and 61.54% (f6-f20) and 30.38% (f21-f28) better performance than PSO. In

TABLE 5.1: Comparison of EMNA-TC-ML(RNN) vs. EMNA-TC, CMA-ES, DE, and PSO .

Function	RNN	EMNA-TC		CMA-ES		DE		PSO	
No.	Mean	Mean	%-diff	Mean	%-diff	Mean	%-diff	Mean	%-diff
1	8.6E-05	2.0E-03	-95.6%	0.0E+00	100.0%	4.2E-07	99.5%	0.0E+00	100.0%
2	7.8E+05	2.4E+06	-67.6%	0.0E+00	100.0%	3.9E+06	-80.1%	1.9E+06	-58.4%
3	2.2E+10	5.1E+09	77.0%	2.1E+00	100.0%	2.1E+06	100.0%	9.0E+07	99.6%
4	1.3E-04	2.4E-03	-94.6%	2.6E+03	-100.0%	2.4E+04	-100.0%	1.7E+04	-100.0%
5	4.2E-04	4.0E-03	-89.5%	0.0E+00	100.0%	7.8E-05	81.3%	0.0E+00	100.0%
F1-F5			-54.1%		60.0%		20.1%		28.2%
6	2.3E+01	2.3E+01	-0.8%	7.7E+00	66.5%	1.4E+01	39.9%	1.5E+01	33.3%
7	5.1E-01	4.7E+00	-89.3%	3.1E+01	-98.4%	2.1E+01	-97.6%	6.5E+01	-99.2%
8	2.1E+01	2.1E+01	0.0%	2.2E+01	-2.7%	2.1E+01	-0.4%	2.1E+01	0.1%
9	1.7E+00	1.8E+00	-6.0%	2.0E+01	-91.4%	3.8E+01	-95.5%	2.8E+01	-93.9%
10	1.0E-03	1.8E-02	-94.2%	1.4E-02	-92.5%	5.9E-01	-99.8%	1.2E-01	-99.1%
11	2.2E+00	2.7E+00	-16.2%	5.4E+01	-95.9%	6.1E+01	-96.4%	6.3E+01	-96.5%
12	1.7E+00	1.5E+00	9.5%	5.3E+01	-96.8%	2.2E+02	-99.2%	8.0E+01	-97.9%
13	1.7E+00	1.8E+00	-9.2%	1.2E+02	-98.7%	2.2E+02	-99.2%	1.4E+02	-98.8%
14	3.9E+02	4.4E+02	-10.8%	3.8E+03	-89.8%	2.7E+03	-85.5%	2.6E+03	-85.1%
15	2.5E+02	2.8E+02	-10.9%	4.4E+03	-94.4%	7.3E+03	-96.6%	4.1E+03	-94.0%
16	1.9E-01	3.6E-01	-45.7%	8.9E+00	-97.8%	2.5E+00	-92.2%	1.6E+00	-88.2%
17	4.7E+01	6.4E+01	-26.2%	1.7E+02	-71.6%	1.1E+02	-58.1%	9.8E+01	-51.8%
18	7.5E+01	1.1E+02	-33.6%	2.9E+02	-73.8%	2.5E+02	-69.6%	1.7E+02	-56.5%
19	3.2E+00	3.1E+00	0.5%	3.3E+00	-2.7%	1.4E+01	-76.8%	5.7E+00	-44.7%
20	1.5E+01	1.5E+01	-0.9%	1.5E+01	-0.9%	1.3E+01	13.2%	1.2E+01	21.3%
F6-F20			-22.3%		-62.7%		-67.6%		-63.4%
21	3.0E+02	3.1E+02	-0.2%	3.2E+02	-4.4%	3.0E+02	3.3%	2.3E+02	24.2%
22	3.4E+02	3.7E+02	-6.3%	4.2E+03	-91.7%	2.3E+03	-84.8%	3.1E+03	-88.7%
23	5.0E+02	4.6E+02	7.3%	4.8E+03	-89.6%	7.3E+03	-93.1%	4.7E+03	-89.2%
24	2.1E+02	2.1E+02	0.2%	2.4E+02	-14.1%	2.4E+02	-13.0%	2.8E+02	-25.0%
25	2.0E+02	2.0E+02	-0.1%	2.7E+02	-23.3%	2.5E+02	-18.4%	2.9E+02	-29.9%
26	2.9E+02	3.0E+02	-1.7%	3.1E+02	-5.2%	2.0E+02	31.2%	2.1E+02	28.8%
27	3.7E+02	3.7E+02	-1.4%	7.4E+02	-50.1%	9.8E+02	-62.4%	1.0E+03	-64.7%
28	3.0E+02	3.0E+02	-0.5%	3.8E+02	-21.2%	3.0E+02	0.1%	2.9E+02	2.8%
F21-F28			-0.3%		-37.5%		-29.7%		-30.2%
F1-F28			-21.7%		-33.6%		-41.1%		-37.6%

all three cases, TF4 also had better performance with f4 and better than DE and PSO in f2 within the unimodal functions.

5.3.4 Threshold Curves

We also collected threshold data during the runs. Some of this data has been included in figures for a combination of each hybrid-model-type's highest scoring dataset variant paired with its best function (see Figure 5.4) and its lowest scoring dataset variant paired with its worst function (see Figure 5.3). Each of the individual graphs includes an EMNA-TC run of that function, a RNN (V0) run with that function, a TF4 (V3p9) run with that function, and the examined hybrid. It is noticeable that the RNN examples converge just a bit faster than EMNA-TC, while TF4 runs very similarly to EMNA-TC. In the figure showing the graphs of the most successful pairs (Figure 5.4), it is clear that some of the hybrids stick fairly close to EMNA-TC, while others (such as QDA, TF3, and LOG) don't converge quite so quickly, and have noticeable periods where convergence is slowed and then sped up. On the other hand, the other figure (Figure 5.3) shows a very different story. While RNN stays quite close to EMNA-TC's example, many of the others show difficulties. TF3, for instance, appears to stall for quite a while and then

TABLE 5.2: Comparison of EMNA-TC-ML(TF4) vs. EMNA-TC, CMA-ES, DE, and PSO .

Function	TF4	EMNA-TC		CMA-ES		DE		PSO	
No.	Mean	Mean	%-diff	Mean	%-diff	Mean	%-diff	Mean	%-diff
1	1.8E-04	2.0E-03	-90.9%	0.0E+00	100.0%	4.2E-07	99.8%	0.0E+00	100.0%
2	1.3E+06	2.4E+06	-47.1%	0.0E+00	100.0%	3.9E+06	-67.4%	1.9E+06	-31.9%
3	1.6E+09	5.1E+09	-67.8%	2.1E+00	100.0%	2.1E+06	99.9%	9.0E+07	94.5%
4	2.3E-04	2.4E-03	-90.5%	2.6E+03	-100.0%	2.4E+04	-100.0%	1.7E+04	-100.0%
5	8.1E-04	4.0E-03	-79.7%	0.0E+00	100.0%	7.8E-05	90.4%	0.0E+00	100.0%
F1-F5			-75.2%		60.0%		24.5%		32.5%
6	2.4E+01	2.3E+01	3.5%	7.7E+00	68.0%	1.4E+01	42.5%	1.5E+01	36.2%
7	1.1E+00	4.7E+00	-75.9%	3.1E+01	-96.4%	2.1E+01	-94.6%	6.5E+01	-98.2%
8	2.1E+01	2.1E+01	-0.1%	2.2E+01	-2.8%	2.1E+01	-0.4%	2.1E+01	0.0%
9	2.0E+00	1.8E+00	9.6%	2.0E+01	-89.9%	3.8E+01	-94.7%	2.8E+01	-92.9%
10	1.9E-03	1.8E-02	-89.2%	1.4E-02	-86.1%	5.9E-01	-99.7%	1.2E-01	-98.4%
11	2.1E+00	2.7E+00	-21.2%	5.4E+01	-96.2%	6.1E+01	-96.6%	6.3E+01	-96.7%
12	1.6E+00	1.5E+00	5.8%	5.3E+01	-96.9%	2.2E+02	-99.3%	8.0E+01	-98.0%
13	1.4E+00	1.8E+00	-24.8%	1.2E+02	-98.9%	2.2E+02	-99.4%	1.4E+02	-99.0%
14	3.9E+02	4.4E+02	-10.7%	3.8E+03	-89.8%	2.7E+03	-85.5%	2.6E+03	-85.0%
15	2.8E+02	2.8E+02	0.5%	4.4E+03	-93.6%	7.3E+03	-96.2%	4.1E+03	-93.2%
16	3.1E-01	3.6E-01	-13.1%	8.9E+00	-96.5%	2.5E+00	-87.4%	1.6E+00	-81.1%
17	5.2E+01	6.4E+01	-18.9%	1.7E+02	-68.8%	1.1E+02	-53.9%	9.8E+01	-47.0%
18	9.3E+01	1.1E+02	-17.8%	2.9E+02	-67.5%	2.5E+02	-62.4%	1.7E+02	-46.2%
19	3.1E+00	3.1E+00	0.1%	3.3E+00	-3.2%	1.4E+01	-76.9%	5.7E+00	-45.0%
20	1.5E+01	1.5E+01	-1.1%	1.5E+01	-1.1%	1.3E+01	13.1%	1.2E+01	21.2%
F6-F20			-16.9%		-61.3%		-66.1%		-61.5%
21	3.0E+02	3.1E+02	-2.8%	3.2E+02	-6.9%	3.0E+02	0.6%	2.3E+02	22.2%
22	3.5E+02	3.7E+02	-5.2%	4.2E+03	-91.6%	2.3E+03	-84.7%	3.1E+03	-88.6%
23	4.8E+02	4.6E+02	3.8%	4.8E+03	-90.0%	7.3E+03	-93.4%	4.7E+03	-89.6%
24	2.1E+02	2.1E+02	0.7%	2.4E+02	-13.7%	2.4E+02	-12.7%	2.8E+02	-24.7%
25	2.0E+02	2.0E+02	0.3%	2.7E+02	-23.0%	2.5E+02	-18.1%	2.9E+02	-29.7%
26	3.0E+02	3.0E+02	-1.2%	3.1E+02	-4.6%	2.0E+02	31.6%	2.1E+02	29.2%
27	3.7E+02	3.7E+02	-1.5%	7.4E+02	-50.2%	9.8E+02	-62.5%	1.0E+03	-64.7%
28	3.0E+02	3.0E+02	-0.4%	3.8E+02	-21.1%	3.0E+02	0.2%	2.9E+02	2.9%
F21-F28			-0.8%		-37.7%		-29.9%		-30.4%
F1-F28			-22.7%		-32.9%		-39.6%		-35.8%

is forced to converge late. RF goes through periods of slow convergence and areas of steeper slopes. QDA and LDA show somewhat similar problems as TF3, but not quite as severe and all three show that forced late convergence.

5.4 Selected Function-Specific Results

As was mentioned in Chapter 3, there was not an even distribution for functions within the examples. This distribution also changed significantly as percentages increased in the V1 and V3 variant types.

We revisit this now because it is interesting to note that f3, f7, f15, f16, f22, and f23 were shared problem functions (See Figure 5.5) within many of the hybrids (RNN often something of an exception). Of these, only f16 typically had no training examples (only present for V0/V2 and V1p1, though only 6 examples for V1p1). F3, f15, f22, and f23 all remained throughout the datasets, with fairly strong numbers.

F1, f4, f5, and f10 all had strong showings (positive relative differences) throughout the hybrids. F10 was the only one of these that had a strong (number of examples)

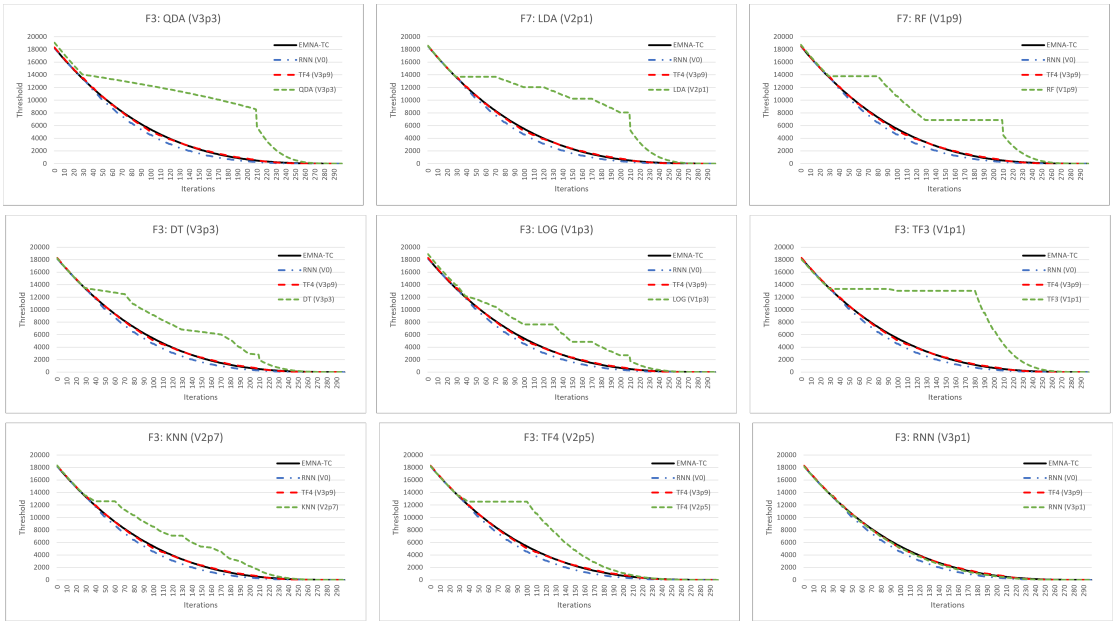

FIGURE 5.3: Threshold Curves For Each Model Type's Lowest-Scoring Hybrid on Its Comparatively Lowest-Scoring CEC '13 Function

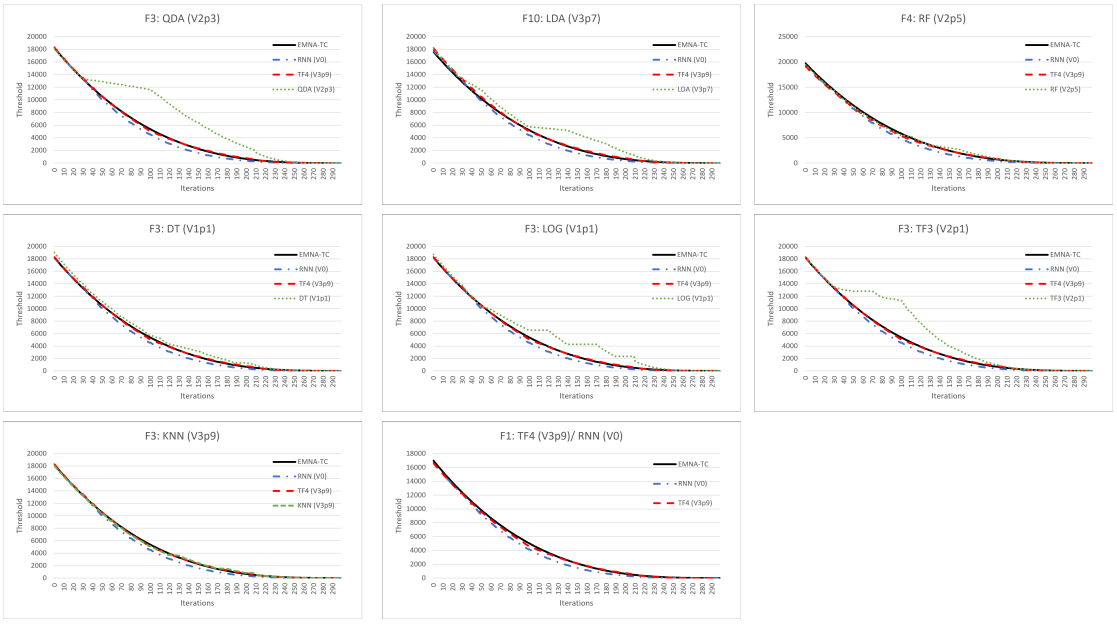

FIGURE 5.4: Threshold Curves For Each Model Type's Highest-Scoring Hybrid on Its Comparatively Highest-Scoring CEC '13 Function

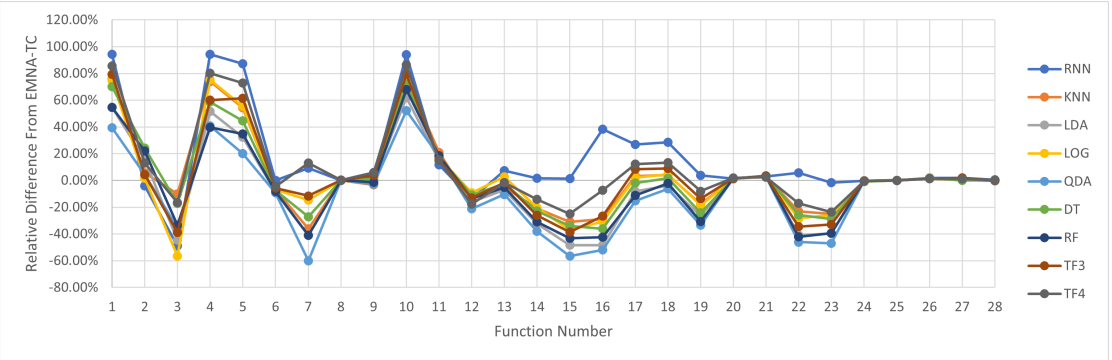

FIGURE 5.5: Hybrid-Type Average Relative Difference from EMNA-TC by CEC Function

start, and all save f1 and f4 diminished very quickly (those two diminished as well, but somewhat slower than the others).

Chapter 6

Discussion and Conclusions

6.1 Discussion

While it may be somewhat difficult to determine how much success we had in our project, it is clear that we did have some success. For instance, we had 32 hybrids that had average performances that were at least 10% better than our guide (EMNA-TC). We also had two different model types that had hybrids that exceeded EMNA-TC performance by more than 20%. One of our general model types (RNN) had a very strong showing, only failing to have two of its models outperform EMNA-TC by at least 10%, and those two still bested EMNA-TC by more than 9%. Ignoring thresholds other than 0; 109 hybrids scored better performances than EMNA-TC (even if only marginally), while 62 hybrids scored worse performances than EMNA-TC (even if only marginally). The worst performance was turned in by QDA (V3p3) where EMNA-TC had a 22.34% better average performance than that QDA hybrid.

With all of that in mind, it was not wholly surprising that QDA and LOG did not do well in hybrid testing. Both of them had several instances where the models suffered (comparatively) in terms of precision and instances of limited utilization of the available classes.

There was also the question of which is more important in hybrid performance: dataset or model type. This is a difficult question. When we look at our models (see Figure 5.2 and Figure 5.1), we do have one model that was relatively good across all datasets (RNN) and a second model (TF4) which almost met the same bar (notably falling short with V2p5 at -4.98%), the same cannot be said for datasets. Every dataset had models that showed performance below that of EMNA-TC (See Figure 6.1 and Figure 6.2). It should be noted, though, that the V0 dataset only had two models (KNN and RF) that showed

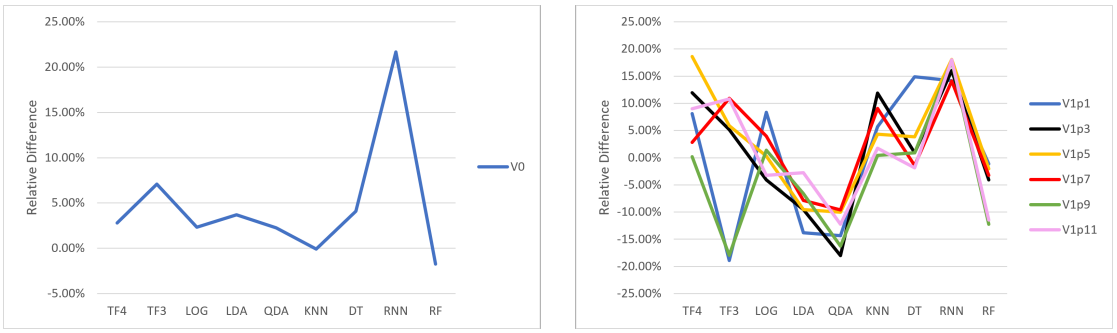

FIGURE 6.1: Relative (to EMNA-TC) Performance of Hybrids Within The V0 (left) and V1 (right) Variant Type2

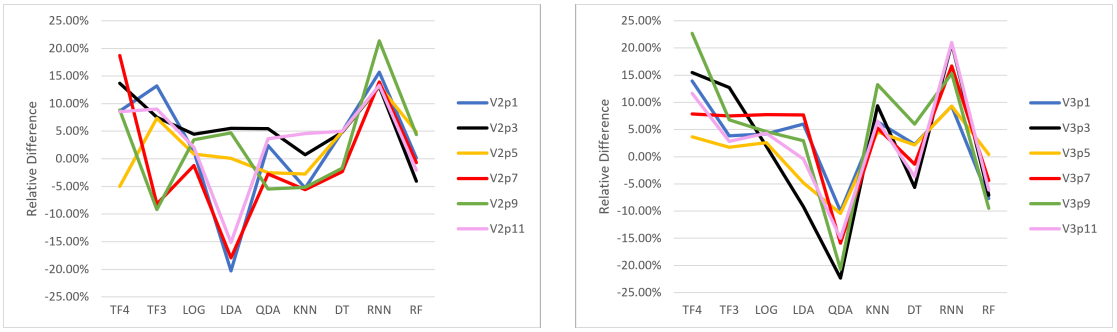

FIGURE 6.2: Relative (to EMNA-TC) Performance of Hybrids Within The V2 (left) and V3 (right) Variant Types

performance below EMNA-TC, and both of those were within 2%. Of course, it is also important to note that RNN was the only model that showed a strong performance with that dataset (21.67%) with the next highest being TF3 at 7.07%. V2p3 only had one model showing worse performance than EMNA-TC (Rf with -4.02%) and two models above 10% (RNN with 13.06% and TF4 at 13.66%). When looking at the averages (see Table 6.1 and Table 6.2), the highest datasets averages are owned by V2p3 (5.69%) and V0 (4.68%), with the lowest from V1p9 (-3.60%). These pale in comparison to RNN (15.94%), TF4 (9.59%), and QDA (-9.04%). I think an argument could definitely be made that dataset-model type pairs are important to consider, but it looks as if model type may be the more important of the two.

Another interesting note is the poor correlation between accuracy and performance (see Figure 6.3). It is possible that this may be due to how low the accuracy was to begin with, but it was expected that there would be some correlation between classification ability and hybrid performance. This simply did not present itself well. In looking at (Figure 6.3), we can see that there is very poor correlation between the two metrics. Instead we have tall groupings around two accuracy ranges (approximately 0.13-0.22 and 0.58-0.68), with some scattered points between and drifting away from the denser cluster regions. Expecting good correlation between the two, we would expect our

Table 6.1: Variant Type Average Relative Differences From EMNA-TC

Variant	Average Relative Difference
V0	4.68%
V1p1	0.34%
V1p3	1.14%
V1p5	3.26%
V1p7	2.08%
V1p9	-3.60%
V1p11	0.89%
V2p1	2.32%
V2p3	5.69%
V2p5	2.34%
V2p7	-0.67%
V2p9	2.36%
V2p11	3.22%
V3p1	3.14%
V3p3	1.79%
V3p5	1.01%
V3p7	3.46%
V3p9	4.62%
V3p11	2.35%

Table 6.2: Hybrid's Model Type Average Relative Differences From EMNA-TC

Model Type	Average Relative Difference
TF4	9.59%
TF3	3.06%
LOG	2.39%
LDA	-4.59%
QDA	-9.04%
KNN	3.42%
DT	1.93%
RNN	15.94%
RF	-3.55%

Random Forest hybrids (these models had the highest accuracy for each dataset) to be among the best of the hybrids. Instead, the RF hybrids fared very poorly throughout.

6.2 Conclusions

Within this thesis, we have shown that it is possible to improve upon EMNA-TC by using Machine Learning. In this project, we were able to influence Thresheld Convergence by directly setting the γ parameter by having the hybrid request the parameter from a machine learning model rather than a static calculation. That influence translated into an increase in performance relative to the original algorithm (EMNA-TC). The method

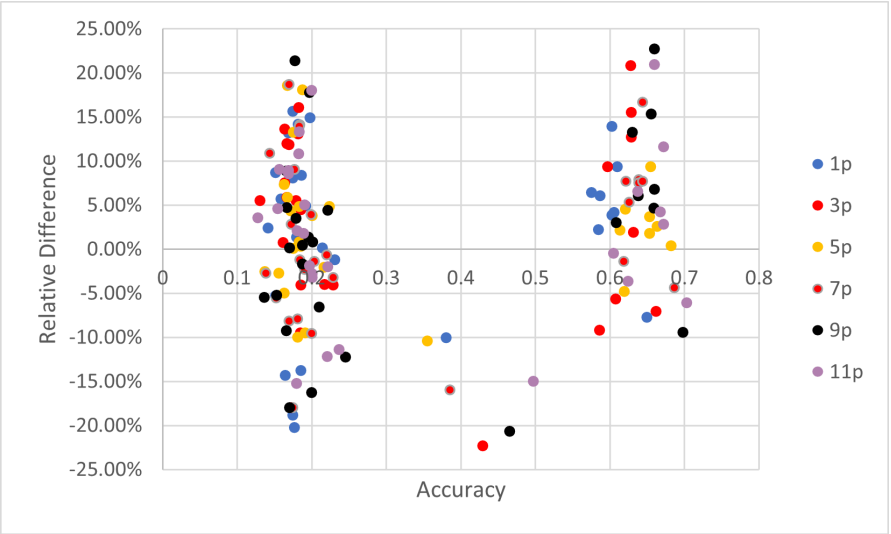

FIGURE 6.3: Accuracy vs Performance Difference (relative to EMNA-TC) for All Datasets (Coded by Dataset Percentage [p] Value)

employed should be transferable to other algorithms, specifically those that also use the same Thresheld Convergence strategy.

It was also seen that, while both have influence on the result, selecting an appropriate model type is more important to performance for this particular problem. The evidence shown within this thesis indicates that varying the dataset will influence the performance of a model. Unfortunately, that influence alone will not guarantee that the model utilized in the hybrid will be able to increase the performance of the algorithm. On the other hand, we have seen that there was one model choice that was always a reasonable choice (at least for the datasets attempted). While there were variations in how much improvement was present, evidence suggests that an RNN-hybrid will improve upon the algorithm's performance (at least in terms of this problem and strategy), regardless of the dataset (assuming that the dataset is reasonably-populated and appropriate to the problem).

We finished the project with two notable model types. First, TF4 was notable because it gave us a hybrid (TF4-V3p9) with the highest average relative difference compared to EMNA-TC (22.70%). RNN was notable because it also presented a hybrid (interestingly, four above 20%) that had a high average relative difference compared to EMNA-TC (RNN-V0 with a relative difference of 21.67%). Moreover, the RNN hybrids also had a very successful run, and their overall average difference was 15.94%. That model-type performance average was significantly higher than both the next most successful average model (TF4 at 9.59%) and the average of the dataset with the highest overall hybrid average (V2p3 with 5.69%).

An interesting note is that many of the models we used to create hybrids would not be considered to be good classifiers. Most of RNN's hybrids in V3 had accuracies less than the Null Accuracy, Yet Hybrids RNN-V3p3 and RNN-V3p11 both had relative differences (compared to EMNA-TC) greater than 20%. 12 of TF4's models failed to match the Null Accuracy, including TF4-V3p9 (the hybrid with the best overall performance) and TF4-V1p5 (TF4's second best performing Hybrid). It seems that, in attempting to create classifiers that would improve performance in an EMNA-TC, we have trained models that were often limited as classifiers, but (for certain models) still worked well in improving EMNA-TC performance.

6.3 Recommendations and Limitations

Unfortunately, this project was not without limitations.

The set of examples prepared for the project was not very large, especially with the original aim of utilizing Deep Learning. The number of examples was already limited based on the time required to produce them. After that, examples were very heavily removed through the initial stages of the project as many of the examples had results where multiple classes would have been applicable to the example, making them difficult to include and use to train models. After that, many more examples were removed (from various datasets) as their results were simply too close and part of the project was looking at how variations in the dataset would influence the final result. With that in mind, we were sharply limited in terms of examples (see Figure 3.1).

As was shown in Chapters 3 and 5, we also had limitations in the distribution of specific functions. Some of these limitations did not seem to influence the results, while others might have (as f16, for example, had limited examples and showed relatively poor performance among the hybrids).

Between those two items, what it may mean is that there are specific functions that it might have been better to focus on when building examples. We eliminated examples because of how close those examples came to having multiple acceptable classes (or when they started with multiple acceptable classes), and many of the problem areas were among those that still had a significant number of examples. It might be that we simply needed more examples of the functions that we know are problem areas. Some of the other functions have hybrids that are seemingly indifferent to their absence. F1 and f10, were both dealt with relatively well by the hybrids despite having scant examples. This may be evidence that we may not need any further examples of that type (or at

least no particular excess of them) as the current crop of hybrids weren't overly hindered without them.

The original intention of the project was intended on focusing on improving EMNA-TC performance through controlling exploration with Deep Learning models. This quickly changed to more of a focus of more conventional models. Part of this was due to the limited number of training examples. We did have some success with neural networks, though our strongest (as far as performance as gauged by relative difference from EMNA-TC) only had three layers.

6.4 Future Work

In terms of the future, it might be worthwhile, if the resources are available, to build up the examples (possibly with a focus on known problem areas shown by the current work). This would then allow for more work on building hybrids using supervised models. Again, the initial aim was to focus on Deep Learning models, and they typically rely on a significant amount of data. If future projects had more data available, they might be able to utilize that route more effectively.

In looking at our current work, an investigation into the correlation issues between accuracy and hybrid performance may prove of interest. The poor correlation between the two came as something of a surprise, and more information (and hopefully a good explanation) would be of interest.

There are also significant avenues of expansion within the presented work. We did a limited amount of work with regression models. A single set of regression models was trained for the V0 training set, but after that, all focus was on classification. Looking at this as a regression problem may yield interesting results.

We did some work in reducing features, but this was also somewhat limited. More focus on reduced dimensions may also yield better results.

There are other avenues of approach. There is the possibility that, given the information that we have collected, this problem could shift into a reinforcement project. This would change the structure of the problem. There is already some data available to compare progress on new techniques, through our findings in this project.

Bibliography

[1] Anna Levitin. *Introduction to the design and analysis of algorithms*. Pearson, 3rd edition, 2012.

[2] S. Memeti, S. Pllana, A. Binotto, Kolodziej. J., and I. Brandic. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. *Computing*, 101:893–936, 2019.

[3] L. Calvet, J. de Armas, D. Masip, and AA Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280, 2017.

[4] Dania Tamayo-Vera, Antonio Bolufé-Röhler, and Stephen Chen. Estimation multivariate normal algorithm with thresheld convergence. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 3425–3432, 2016. doi: 10.1109/CEC.2016.7744223.

[5] A. Bolufé-Röhler and J. Luke. A data-centric machine learning approach for controlling exploration in estimation of distribution algorithms. 2022 IEEE Congress on Evolutionary Computation (CEC) - Under Review, 2022.

[6] Bolufe-Rohler-A. Luke, J. Emna-tc-ml-hybrid. https://github.com/jmpluke/EMNA-TC-ML-Hybrid, 2022.

[7] A. Levitin. *Introduction to the design and analysis of algorithms*. Pearson, 2012. 3rd Edition.

[8] P Fox. Using heuristics. URL https://www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/solving-hard-problems/a/using-heuristics.

[9] Stewart Russel and Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson, 2020. 4th Edition.

[10] de Armas-J. Masip D. Juan AA Calvet, L. Learnheuristics : hybridizing meta-heuristics with machine learning for optimization with dynamic inputs. *De Gruyter Open*, 2017.

[11] Pilana S.-Binotto A. Kolodziej. J. Brandic-I. Memeti, S. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. *Computing*, 101(8), 2019.

[12] Scikit-Learn. sklearn.linear_model.logisticregression, . URL https://scikit-learn.org/stable//modules/generated/sklearn.linear_model.LogisticRegression.html.

[13] Scikit-Learn. 1.1. linear models, . URL https://scikit-learn.org/stable//modules/linear_model.html.

[14] SW Knox. *Machine Learning: A Concise Introduction*. Wiley-Blackwell, 2018.

[15] R. A. Fisher. The use of multiple measurements in taxonomic problems. In *Annals of Eugenics*, page 7 (2): 179–188, 1936.

[16] Scikit-Learn. 1.2. linear and quadratic discriminant analysis, . URL https://scikit-learn.org/stable//modules/lda_qda.html.

[17] Scikit-Learn. sklearn.discriminant_analysis.quadraticdiscriminantanalysis, . URL https://scikit-learn.org/stable//modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html#sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.

[18] Scikit-Learn. sklearn.linear_model.sgdclassifier, . URL https://scikit-learn.org/stable//modules/generated/sklearn.linear_model.SGDClassifier.html?highlight=sgd#sklearn.linear_model.SGDClassifier.

[19] Scikit-Learn. 1.5. stochastic gradient descent, . URL https://scikit-learn.org/stable//modules/sgd.html#sgd.

[20] Scikit-Learn. sklearn.svm.linearsvc, . URL https://scikit-learn.org/stable//modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC.

[21] Scikit-Learn. 1.4. support vector machines, . URL https://scikit-learn.org/stable//modules/svm.html#svm-classification.

[22] Scikit-Learn. sklearn.neighbors.radiusneighborsclassifier, . URL https://scikit-learn.org/stable//modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html#sklearn.neighbors.RadiusNeighborsClassifier.

[23] Scikit-Learn. 1.6. nearest neighbors, . URL https://scikit-learn.org/stable/
/modules/neighbors.html#classification.

[24] Peter E Hart. Thomas M Cover. In *Nearest neighbor pattern classification*. IEEE, 1967.

[25] Scikit-Learn. sklearn.tree.decisiontreeclassifier, . URL https://scikit-learn.
org/stable//modules/generated/sklearn.tree.DecisionTreeClassifier.
html.

[26] Scikit-Learn. 1.10. decision trees, . URL https://scikit-learn.org/stable/
/modules/tree.html#tree.

[27] Scikit-Learn. sklearn.ensemble.randomforestclassifier, . URL https:
//scikit-learn.org/stable/modules/generated/sklearn.ensemble.
RandomForestClassifier.html.

[28] Scikit-Learn. 1.11.2. forests of randomized trees, . URL https://scikit-learn.
org/stable/modules/ensemble.html#forest.

[29] Scikit-Learn. sklearn.naive_bayes.gaussiannb, . URL https://scikit-learn.org/
stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

[30] Scikit-Learn. 1.9. naïve bayes, . URL https://scikit-learn.org/stable/
modules/naive_bayes.html#gaussian-naive-bayes.

[31] Scikit-Learn. 1.1.3. lasso, . URL https://scikit-learn.org/stable/modules/
linear_model.html#lasso.

[32] Scikit-Learn. 1.1.2. ridge regression and classification, . URL https://
scikit-learn.org/stable/modules/linear_model.html#ridge-regression.

[33] Scikit-Learn. sklearn.linear_model.ridgecv, . URL https://scikit-learn.org/
stable/modules/generated/sklearn.linear_model.RidgeCV.html#sklearn.
linear_model.RidgeCV.

[34] Scikit-Learn. 1.1.10.1. bayesian ridge regression, . URL https://scikit-learn.
org/stable/modules/linear_model.html#bayesian-regression.

[35] Scikit-Learn. Bayesian ridge regression, . URL https://scikit-learn.org/
stable/auto_examples/linear_model/plot_bayesian_ridge.html.

[36] Scikit-Learn. 1.1.10. bayesian regressions, . URL https://scikit-learn.org/
stable/modules/linear_model.html#bayesian-ridge-regression.

[37] Scikit-Learn. Automatic relevance determination regression (ard), . URL https://scikit-learn.org/stable/auto_examples/linear_model/plot_ard.html.

[38] Scikit-Learn. sklearn.linear_model.ardregression, . URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ARDRegression.html.

[39] Scikit-Learn. sklearn.linear_model.huberregressor, . URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html.

[40] Scikit-Learn. 1.1.16.4. huber regression, . URL https://scikit-learn.org/stable/modules/linear_model.html#huber-regression.

[41] P.N. Suganthan Alfredo G. Hernández-Díaz J.J. Liang, B.Y. Qu. Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. 2013.

[42] Bolufé-Röhler A. Tamayo-Vera, D. and Chen S. Estimation multivariate normal algorithm with thresheld convergence, comparison, and synergy test. evol. comput. pages 3425–3432. 2016 IEEE Congress on Evolutionary Computation (CEC), 2012.

[43] Stephen Chen, James Montgomery, Antonio Bolufé-Röhler, and Yasser Gonzalez-Fernandez. A review of thresheld convergence. *GECONTEC: Revista Internacional de Gestión del Conocimiento y la Tecnología*, 3:1–13, 05 2015.

[44] Dania Tamayo-Vera, Antonio Bolufé-Röhler, and Stephen Chen. Estimation multivariate normal algorithm with thresheld convergence. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3425–3432, 2016.

[45] Antonio Bolufé-Röhler and Dania Tamayo-Vera. An exploration-only exploitation-only hybrid for large scale global optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1062–1069. IEEE, 2021.

[46] Antonio Bolufé-Röhler, Stephen Chen, and Dania Tamayo-Vera. An analysis of minimum population search on large scale global optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 1205–1212. IEEE, 2019.

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

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing

Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[49] François Chollet et al. Keras. https://keras.io, 2015.

[50] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[51] Janani Ravi. Building classification models with scikit-learn, 2019. URL https://app.pluralsight.com/library/courses/building-classification-models-scikit-learn/table-of-contents.

[52] Janani Ravi. Getting started with tensorflow 2.0, 2020. URL https://app.pluralsight.com/library/courses/getting-started-tensorflow-20/table-of-contents.

Appendix A

Python Code

In this section, we have presented some of the code used for this project. The code present will be similar to that used for the variant 1 setup at 1% (V1p1).

All code used available at the following GitHub repository: [6].

For this code, the models included involve Scikit-Learn model types [50] and TensorFlow Keras models [48, 49]. Strategies for building these were learned in the process of this project through Pluralsight courses [51, 52].

A.1 Create Excel files

```
'''
    This is an example of the code used to take the data,
    remove selected examples based on Variant type, and
    write it to an Excel file. This is the code used for
    Variant 1, at 1%
'''

import pickle
import xlsxwriter
import pandas as pd

start_file = 1
# the first number for the part between end_file and ending_file in the file name

end_file = 28 # the last number for the files I have.
init_file = '../../abr data/emna_g_exp'
second_file = '/fun'
mid_file = '_exp'
end_file_name = '.p'
exp_dir_nums = []
functions = []
```

```
for i in range(1,11):
    exp_dir_nums.append(str(i))
    functions.append(28)
for i in range(11,20):
    s = str(i) + '-'
    exp_dir_nums.append(s)
functions.append(23)
functions.append(15)
functions.append(18)
functions.append(19)
functions.append(16)
functions.append(23)
functions.append(24)
functions.append(24)
functions.append(24)

titles = ['count', 'exp', 'function', 'j', 'evaluations consumed', 'evaluations available',
        'gamma', 'threshold-0']
for i in range(1,100):
    x = 'threshold-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = 'stdev-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = 'avfit-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '0%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '10%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '20%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '30%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '40%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '50%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '60%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '70%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '80%-' + str(i)
    titles.append(x)
for i in range(0,100):
```

```
    x = '90%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '100%-' + str(i)
    titles.append(x)
#titles
titles2 = [ 'best_f(0)', 'best_f(0.5)', 'best_f(1)',
          'best_f(1.5)', 'best_f(2)', 'best_f(2.5)',
          'best_f(3)', 'best_f(3.5)', 'best_f(4)',
          'best_f', 'best_gamma', 'best_gammas']
for i in range(0, len(titles2)):
    titles.append(titles2[i])

titles3 = ['exp', 'func', 'number used', 'number excluded', 'extra excluded in variant 1']

import os
import sys
import inspect

current_dir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parent_dir = os.path.dirname(current_dir)
sys.path.insert(0,parent_dir)

from linked_node import LinkedNode

#used for x% difference between high/low
difference = 0.01

workbook = xlsxwriter.Workbook('data_col_v1.xlsx') # First Excel file
workbook.use_zip64()
worksheet = workbook.add_worksheet('main data')
workbook2 = xlsxwriter.Workbook('excel_files/data_info_v1.xlsx') # Info Excel file
worksheet2 = workbook2.add_worksheet('data info')
count  = 0 # Will be incremented to make up the count attribute
row = 0
column = 0
num_gammas = 9
num_multiples = 0
multiples_array = []
total_used = 0
full_set = 0
partial_set = 0
var1ex = 0
total_used_v1 = 0

# write column headers
for i in range(0, len(titles)):
    worksheet.write(row,column,titles[i])
    column += 1

# Write info headers
rowI = 0
columnI = 0
for i in range(0, len(titles3)):
    worksheet2.write(rowI,columnI,titles3[i])
```

```
    columnI += 1
columnI = 0
rowI = 1

# Increment Row
column = 0
row = 1

for e in range(1,len(exp_dir_nums) + 1):
    #for i in range(start_file, end_file + 1):
    for i in range(start_file, functions[e - 1] + 1):
        included = 0
        excluded = 0
        extra_v1_ex = 0

        # Get LinkedNode from file
        expString = ''
        if e < 9:
            expString = exp_dir_nums[e-1][0:1]
        else:
            expString = exp_dir_nums[e-1][0:2]

        filename = init_file + exp_dir_nums[e-1] + second_file + str(i) + mid_file + expString +
                   end_file_name
        example = open(filename, 'rb')
        exampleContents = pickle.load(example)
        example.close()

        for j in range(0, len(exampleContents)):
            max_f = exampleContents[j].max_f
            worksheet.write(row,column,count)
            count += 1
            column += 1
            worksheet.write(row,column,e)
            column += 1
            worksheet.write(row,column,i)
            column += 1
            worksheet.write(row,column,j)
            column += 1

            for z in range(0,3):
                worksheet.write(row,column,exampleContents[j].data[z][0])
                column += 1

            # Threshold done separately
            for z in range(0,len(exampleContents[j].data[3])):
                worksheet.write(row,column,exampleContents[j].data[3][z])
                column += 1

            # starting at 4 to exclude threshold
            for k in range(4,len(exampleContents[j].data)):
                for z in range(0,len(exampleContents[j].data[4])):
                    worksheet.write(row,column,exampleContents[j].data[k][z] / max_f)
                    column += 1
```

```
best_g = 0
best_gs = '0'
best_f = 1000000000
worst_f = -1000000000
range_min = 1000000000
range_max = 1000000000
range_avg = 1000000000

for w in range(0, len(exampleContents[j].results)):
    f = exampleContents[j].results[w].best_f * 2
    if f > worst_f:
        worst_f = f

    if abs(f - range_avg) < 0.001:
        best_gs = best_gs + ' ' + str(float(exampleContents[j].results[w].gamma))
        if f < best_f:
            best_f = f
            best_g = float(exampleContents[j].results[w].gamma)
        if f < range_min:
            range_min = f
            range_avg = (range_max + range_min) / 2
        elif f > range_max:
            range_max = f
            range_avg = (range_max + range_min) / 2
    elif f < best_f:
        best_f = f
        range_min = f
        range_max = f
        range_avg = f
        #w_float = float(exampleContents[j].results[w].gamma)
        best_g = int(exampleContents[j].results[w].gamma * 2)
        best_gs = str(best_g)

    worksheet.write(row,column,f)
    column += 1
worksheet.write(row,column,best_f)
column += 1
worksheet.write(row,column,best_g)
column += 1
worksheet.write(row,column,best_gs)
column = 0

# This if/else collects information on if the example has equal fitness for all
# gammas, if so that training set gets overwritten
if len(best_gs) == 17:
    num_multiples += 1
    multiples_array.append(best_gs)

    # for info
    excluded += 1
    full_set += 1
elif len(best_gs) > 1:
    num_multiples += 1
    multiples_array.append(best_gs)
```

```
                # for info
                excluded += 1
                partial_set += 1
            else:
                if (abs((best_f - worst_f) / worst_f) > difference):
                    total_used_v1 += 1
                    row += 1
                else:
                    extra_v1_ex += 1
                    var1ex += 1

                total_used += 1

                # for info
                included += 1

        worksheet2.write(rowI,columnI,e)
        columnI += 1
        worksheet2.write(rowI,columnI,i)
        columnI += 1
        worksheet2.write(rowI,columnI,included)
        columnI += 1
        worksheet2.write(rowI,columnI,excluded)
        columnI += 1
        worksheet2.write(rowI,columnI,extra_v1_ex)
        columnI += 1

        rowI += 1
        columnI = 0

workbook.close()
workbook2.close()

# Write a csv file from the xlsx file produced above
switchFile = pd.read_excel(r'data_col_v1.xlsx')
switchFile.to_csv(r'excel_files/data_col_v1.csv', index=None, header=True)

os.remove('data_col_v1.xlsx')
```

A.2 Data Prep

```
'''
    An example of data prep. This was used for variant 1, 1%
'''

import pandas as pd

data = pd.read_csv('excel_files/data_col_v1.csv')

titles = ['count', 'exp', 'function', 'j', 'evaluations consumed', 'evaluations available',
        'gamma', 'threshold-0']
```

```
for i in range(1,100):
    x = 'threshold-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = 'stdev-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = 'avfit-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '0%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '10%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '20%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '30%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '40%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '50%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '60%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '70%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '80%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '90%-' + str(i)
    titles.append(x)
for i in range(0,100):
    x = '100%-' + str(i)
    titles.append(x)
#titles
titles2 = [ 'best_f(0)', 'best_f(0.5)', 'best_f(1)',
          'best_f(1.5)', 'best_f(2)', 'best_f(2.5)', 'best_f(3)', 'best_f(3.5)', 'best_f(4)',
          'best_f', 'best_gamma', 'best_gammas']
for i in range(0, len(titles2)):
    titles.append(titles2[i])

# Remove unnecessary attributes

data_post = data.drop(['exp', 'function','j','count'], axis=1)
data_post = data_post.drop(['gamma','best_f(0)', 'best_f(0.5)', 'best_f(1)',
          'best_f(1.5)', 'best_f(2)', 'best_f(2.5)', 'best_f(3)', 'best_f(3.5)', 'best_f(4)',
          'best_f', 'best_gammas'], axis=1)
```

```
data_post.to_csv('excel_files/data_col_cleaned_v1.csv', index=False)
```

A.3 Train and Test One SKLearn (RF) Model

```
'''
    Training, scoring, and saving RF model for V1p1 models
'''

# Imports
import pickle
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
import pandas as pd

# set hyperparameters
crit = 'gini'
mf = 'log2'
n_est = 282

#An example of One model

model_name = 'v1p1'
filename = 'models/x_train_' + model_name + '.p'
file = open(filename, 'rb')
x_train = pickle.load(file)
file.close()
filename = 'models/y_train_' + model_name + '.p'
file = open(filename, 'rb')
y_train = pickle.load(file)
file.close()
filename = 'models/x_test_' + model_name + '.p'
file = open(filename, 'rb')
x_test = pickle.load(file)
file.close()
filename = 'models/y_test_' + model_name + '.p'
file = open(filename, 'rb')
y_test = pickle.load(file)
file.close()

model = RandomForestClassifier(criterion = crit, max_features=mf, n_estimators=n_est, random_sta
model.fit(x_train, y_train)

model_file = 'models/model_' + model_name + '.p'
file = open(model_file, 'wb')
pickle.dump(model, file)
file.close()

y_pred = model.predict(x_test)
```

```
acc = accuracy_score(y_test, y_pred, normalize=True)
pre = precision_score(y_test, y_pred, average='weighted')
cross_results_test = pd.DataFrame({'y_test': y_test, 'y_pred': y_pred})
result_cross_test = pd.crosstab(cross_results_test.y_pred, cross_results_test.y_test)

writeFile = open('text/ReadMe.txt', 'a')
temp = model_name + ': acc:' + str(acc) + ' pre: ' + str(pre)
writeFile.write(temp)
writeFile.write('\n\n')
writeFile.write(result_cross_test.to_string())
writeFile.write('\n\n')
writeFile.close()
```

A.4 Tensorflow Models

```
'''
    Training, scoring, and saving Keras models
'''
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
import pickle

filename = 'v1_models/x_train_v1.p'
file = open(filename, 'rb')
x_train = pickle.load(file)
file.close()
filename = 'v1_models/y_train_v1.p'
file = open(filename, 'rb')
y_train = pickle.load(file)
file.close()
filename = 'v1_models/x_test_v1.p'
file = open(filename, 'rb')
x_test = pickle.load(file)
file.close()
filename = 'v1_models/y_test_v1.p'
file = open(filename, 'rb')
y_test = pickle.load(file)
file.close()

y_test = to_categorical(y_test, 9)
y_train = to_categorical(y_train, 9)

def create_model1():
    model1 = keras.Sequential()
    model1.add(layers.Dense(64, input_shape=(1402,)))
    model1.add(layers.Dense(32))
    model1.add(layers.Dense(9))
    optimizer = keras.optimizers.SGD(lr = 0.01)
```

```
    loss = tf.keras.losses.CategoricalCrossentropy()
    metrics = ['accuracy']
    model1.compile(loss=loss, metrics=metrics, optimizer=optimizer)
    return model1
model1 = create_model1()

model1.fit(x_train, y_train, epochs = 1000)

evaluation = model1.evaluate(x_test, y_test)
ev = pd.Series(evaluation, index = model1.metrics_names)
print(ev)

model2 = keras.Sequential()
model2.add(layers.Dense(64, input_shape=(1402,), activation='sigmoid'))
model2.add(layers.Dense(32))
model2.add(layers.Dense(9))
optimizer = keras.optimizers.Adam(lr = 0.01)
loss = tf.keras.losses.CategoricalCrossentropy()
metrics = ['accuracy']
model2.compile(loss=loss, metrics=metrics, optimizer=optimizer)

model2.fit(x_train, y_train, epochs = 1000)

evaluation = model2.evaluate(x_test, y_test)
ev = pd.Series(evaluation, index = model2.metrics_names)
print(ev)

model3 = keras.Sequential()
model3.add(layers.Dense(64, input_shape=(x_train.shape[1],), activation='relu'))
model3.add(layers.Dropout(0.1))
model3.add(layers.Dense(32))
model3.add(layers.Dropout(0.1))
model3.add(layers.Dense(9, activation='sigmoid'))
optimizer = keras.optimizers.Adam(lr = 0.01)
loss = tf.keras.losses.CategoricalCrossentropy()
metrics = ['accuracy']
model3.compile(loss=loss, metrics=metrics, optimizer=optimizer)

model3.fit(x_train, y_train, epochs = 1000)
evaluation = model3.evaluate(x_test, y_test)
ev = pd.Series(evaluation, index = model3.metrics_names)
print(ev)

model4 = keras.Sequential()
model4.add(layers.Dense(256, input_shape=(x_train.shape[1],), activation='relu'))
model4.add(layers.Dense(128))
model4.add(layers.Dense(9, activation='softmax'))
optimizer = keras.optimizers.Adam(lr = 0.01)
loss = tf.keras.losses.CategoricalCrossentropy()
metrics = ['accuracy']
model4.compile(loss=loss, metrics=metrics, optimizer=optimizer)
model4.fit(x_train, y_train, epochs = 1000)

evaluation = model4.evaluate(x_test, y_test)
ev = pd.Series(evaluation, index = model4.metrics_names)
```

```
print(ev)

model1.save('v1_models/model19')
model2.save('v1_models/model20')
model3.save('v1_models/model21')
model4.save('v1_models/model22')
```

A.5 Function Testing Main File for DT V1 P1

```
from emna_tc_ml import emna_tc_ml
import functions
import pickle


class OptResults(object):
    def __init__(self):
        self.fitness = []
        self.threshold = []

def obj_function(X):
    return cec_benchmark.Y_matrix(X, fun_num)


fDeltas = [-1400, -1300, -1200, -1100, -1000, -900, -800, -700, -600,
           -500, -400, -300, -200, -100, 100, 200, 300, 400, 500, 600,
           700, 800, 900, 1000, 1100, 1200, 1300, 1400]

dim = 30
max_evals = 10_000 * dim
cec_benchmark = functions.CEC_functions(dim)
popsize = 1000
model = "models/model_DT_v1_1p.p"

runs = 30
thresholds = []
fits = []

for run in range(runs):
    for fun_num in range(1, 29):
        threshold, best_f = emna_tc_ml(obj_function, dim, max_evals, popsize, 100, 3, model)
        if run == 0:
            fits.append([best_f - fDeltas[fun_num-1]])
            thresholds.append([threshold])
        else:
            fits[fun_num - 1].append(best_f - fDeltas[fun_num-1])
            thresholds[fun_num - 1].append(threshold)
        print(f"Run: {run}, EMNA-TC-ML Function {fun_num}, pop
             {popsize}, result: {(best_f - fDeltas[fun_num - 1]):.2E}")

pickle.dump(fits, open("pickles/results_DT_v1_1p_fits.p", "wb"), protocol=4)
```

```
pickle.dump(thresholds, open("pickles/results_DT_v1_1p_thresholds.p", "wb"), protocol=4)
print('fitnesses:')
print(fits)
```

Appendix B

SciKitLearn Models Tables

Table B.1: Model Scores for Original Model Set (V0)

Model	Accuracy	Precision	Recall-Score	Used Classes (n/9)
LogisticRegression-iter-10000	0.184588041	0.078821212	0.078821212	0.333333333
LogisticRegression-iter-100000	0.184588041	0.078821212	0.078821212	0.333333333
LogisticRegression-iter-1000000	0.184588041	0.078821212	0.078821212	0.333333333
LogisticRegression w/ sag	0.184080234	0.101874762	0.101874762	0.555555556
LinearDiscriminantAnalysis-svd	0.180144725	0.164781204	0.164781204	1
LinearDiscriminantAnalysis-lsqr	0.167068681	0.027911944	0.027911944	1
QuadraticDiscriminantAnalysis	0.115907071	0.144499014	0.144499014	1
SGDClassifier-hinge	0.128475308	0.044236422	0.044236422	0.222222222
SGDClassifier-log	0.139266218	0.040628093	0.040628093	0.222222222
SGDClassifier-modified-huber	0.133299479	0.066120371	0.066120371	0.444444444
SGDClassifier-squared-hinge	0.098133807	0.017247601	0.017247601	0.222222222
SGDClassifier-perceptron	0.174304938	0.043829037	0.043829037	0.222222222
LinearSVC-ovr	0.180017773	0.076900352	0.076900352	0.333333333
RadiusNeighborsClassifier-2600	0.181541196	0.157352577	0.157352577	1
DecisionTreeClassifier	0.184080234	0.18253641	0.18253641	1
GaussianNB	0.176336169	0.093766752	0.093766752	0.555555556
LinearSVC-crammer-singer	0.132791672	0.04331649	0.04331649	0.222222222
KNeighborsClassifier-20	0.152977022	0.149850381	0.149850381	1

Table B.2: Model Scores for Continuous Model Set

Model	Accuracy	Precision	Recall-Score
lasso-linear-alpha-0.1	0.133172528	0.141820825	0.133172528
lasso-linear-alpha-0.01	0.129110067	0.133159632	0.129110067
lasso-linear-alpha-0.5	0.130125682	0.137714576	0.130125682
ridge-linear-a-1	0.1110829	0.239435397	0.1110829
ridge-linear-a-0.1	0.100926749	0.110956972	0.100926749
ridge-linear-a-2	0.113621937	0.212344622	0.113621937
ridge-cv	0.128475308	0.173849061	0.128475308
BayesianRidge	0.138123651	0.171495333	0.138123651
ARDRegression	0.122254665	0.142083385	0.122254665
HuberRegressor	0.142947823	0.143975157	0.142947823

Table B.3: Model Scores for Variant 1 Model with 1 percent (V1p1) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/9)
LogisticRegression-iter-10000	0.186376933	0.172660042	0.172660042	0.777777778
LogisticRegression-iter-100000	0.186376933	0.172660042	0.172660042	0.777777778
LogisticRegression-iter-1000000	0.186376933	0.172660042	0.172660042	0.777777778
LogisticRegression w/ sag	0.184287505	0.118636273	0.118636273	0.666666667
LinearDiscriminantAnalysis-svd	0.185750104	0.167434471	0.167434471	1
LinearDiscriminantAnalysis-lsqr	0.079607188	0.006337304	0.006337304	0.111111111
QuadraticDiscriminantAnalysis	0.164646887	0.115729495	0.115729495	0.777777778
SGDClassifier-hinge	0.055787714	0.003112269	0.003112269	0.111111111
SGDClassifier-log	0.149603009	0.078180131	0.078180131	0.444444444
SGDClassifier-modified-huber	0.135185959	0.083998522	0.083998522	0.555555556
SGDClassifier-squared-hinge	0.14814041	0.056303532	0.056303532	0.444444444
SGDClassifier-perceptron	0.178437108	0.047512597	0.047512597	0.222222222
LinearSVC-ovr	0.182824906	0.071041574	0.071041574	0.333333333
RadiusNeighborsClassifier-2600	0.181780192	0.162231672	0.162231672	1
DecisionTreeClassifier	0.198077727	0.197245074	0.197245074	1
GaussianNB	0.165273715	0.064029566	0.064029566	0.444444444
LinearSVC-crammer-singer	0.125574593	0.015768978	0.015768978	0.111111111
KNeighborsClassifier-20	0.15879649	0.155844376	0.155844376	1

Table B.4: Model Scores for Variant 1 Model with 11 percent (V1p11) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/9)
LogisticRegression-iter-10000	0.200843235	0.121309229	0.121309229	0.444444444
LogisticRegression-iter-100000	0.200843235	0.121309229	0.121309229	0.444444444
LogisticRegression-iter-1000000	0.200843235	0.121309229	0.121309229	0.444444444
LogisticRegression w/ sag	0.195093906	0.083233175	0.083233175	0.333333333
LinearDiscriminantAnalysis-svd	0.200459946	0.188487479	0.188487479	1
LinearDiscriminantAnalysis-lsqr	0.040245305	0.001619685	0.001619685	0.111111111
QuadraticDiscriminantAnalysis	0.221157532	0.069301515	0.069301515	0.555555556
SGDClassifier-hinge	0.151782292	0.045981625	0.045981625	0.222222222
SGDClassifier-log	0.122269069	0.149631292	0.149631292	0.777777778
SGDClassifier-modified-huber	0.114220008	0.040264329	0.040264329	0.333333333
SGDClassifier-squared-hinge	0.091605979	0.062602715	0.062602715	0.555555556
SGDClassifier-perceptron	0.119969337	0.057716053	0.057716053	0.444444444
LinearSVC-ovr	0.195093906	0.083460993	0.083460993	0.333333333
RadiusNeighborsClassifier-2600	0.199693369	0.185067311	0.185067311	1
DecisionTreeClassifier	0.197393637	0.197129884	0.197129884	1
GaussianNB	0.145649674	0.098450533	0.098450533	0.555555556
LinearSVC-crammer-singer	0.145266386	0.035433227	0.035433227	0.222222222
KNeighborsClassifier-20	0.189344576	0.181731	0.181731	1

Table B.5: Model Scores for Variant 2 Model with 1 percent (V2p1) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/9)
LogisticRegression-iter-10000	0.179763869	0.116528453	0.116528453	0.555555556
LogisticRegression-iter-100000	0.179763869	0.116528453	0.116528453	0.555555556
LogisticRegression-iter-1000000	0.179763869	0.116528453	0.116528453	0.555555556
LogisticRegression w/ sag	0.178494351	0.09115776	0.09115776	0.555555556
LinearDiscriminantAnalysis-svd	0.176970928	0.156880738	0.156880738	1
LinearDiscriminantAnalysis-lsqr	0.092294021	0.008518186	0.008518186	0.111111111
QuadraticDiscriminantAnalysis	0.141297448	0.123041277	0.123041277	1
SGDClassifier-hinge	0.142440015	0.049992211	0.049992211	0.333333333
SGDClassifier-log	0.09559477	0.049921361	0.049921361	0.555555556
SGDClassifier-modified-huber	0.09902247	0.025132195	0.025132195	0.444444444
SGDClassifier-squared-hinge	0.100545893	0.029405926	0.029405926	0.333333333
SGDClassifier-perceptron	0.13469595	0.0345327	0.0345327	0.222222222
LinearSVC-ovr	0.172019804	0.073079198	0.073079198	0.333333333
RadiusNeighborsClassifier-2600	0.174685794	0.145334238	0.145334238	1
DecisionTreeClassifier	0.191824299	0.193190739	0.193190739	1
GaussianNB	0.171638949	0.09954689	0.09954689	0.666666667
LinearSVC-crammer-singer	0.151834455	0.023053702	0.023053702	0.111111111
KNeighborsClassifier-20	0.151326647	0.14982293	0.14982293	1

Table B.6: Model Scores for Variant 2 Model with 11 percent (V2p11) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/9)
LogisticRegression-iter-10000	0.180525581	0.087338048	0.087338048	0.555555556
LogisticRegression-iter-100000	0.180525581	0.087338048	0.087338048	0.555555556
LogisticRegression-iter-1000000	0.180525581	0.087338048	0.087338048	0.555555556
LogisticRegression w/ sag	0.177986543	0.106109403	0.106109403	0.555555556
LinearDiscriminantAnalysis-svd	0.179890821	0.161384877	0.161384877	1
LinearDiscriminantAnalysis-lsqr	0.107274343	0.011507785	0.011507785	0.111111111
QuadraticDiscriminantAnalysis	0.1279675	0.144798459	0.144798459	0.888888889
SGDClassifier-hinge	0.086962041	0.013646465	0.013646465	0.222222222
SGDClassifier-log	0.128221404	0.057679796	0.057679796	0.444444444
SGDClassifier-modified-huber	0.107274343	0.011507785	0.011507785	0.111111111
SGDClassifier-squared-hinge	0.109432525	0.020959528	0.020959528	0.222222222
SGDClassifier-perceptron	0.165291355	0.050681164	0.050681164	0.333333333
LinearSVC-ovr	0.171511997	0.073205596	0.073205596	0.333333333
RadiusNeighborsClassifier-2600	0.183318522	0.151682593	0.151682593	1
DecisionTreeClassifier	0.190681732	0.191820497	0.191820497	1
GaussianNB	0.168592104	0.090533495	0.090533495	0.555555556
LinearSVC-crammer-singer	0.141043544	0.019893281	0.019893281	0.111111111
KNeighborsClassifier-20	0.154373492	0.15226972	0.15226972	1

Table B.7: Model Scores for Variant 3 Model with 1 percent (V3p1) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/3)
LogisticRegression-iter-10000	0.605649718	0.435253426	0.435253426	0.666666667
LogisticRegression-iter-100000	0.605649718	0.435253426	0.435253426	0.666666667
LogisticRegression-iter-1000000	0.605649718	0.435253426	0.435253426	0.666666667
LogisticRegression w/ sag	0.605649718	0.434836532	0.434836532	0.666666667
LinearDiscriminantAnalysis-svd	0.587288136	0.5389517	0.5389517	1
LinearDiscriminantAnalysis-lsqr	0.152542373	0.023269176	0.023269176	0.333333333
QuadraticDiscriminantAnalysis	0.38079096	0.480606258	0.480606258	1
SGDClassifier-hinge	0.276553672	0.499225496	0.499225496	1
SGDClassifier-log	0.602824859	0.36339781	0.36339781	0.333333333
SGDClassifier-modified-huber	0.602824859	0.36339781	0.36339781	0.333333333
SGDClassifier-squared-hinge	0.602824859	0.36339781	0.36339781	0.333333333
SGDClassifier-perceptron	0.578813559	0.468489783	0.468489783	0.666666667
LinearSVC-ovr	0.607062147	0.435544576	0.435544576	0.666666667
RadiusNeighborsClassifier-2600	0.610169492	0.546909681	0.546909681	1
DecisionTreeClassifier	0.584745763	0.585909449	0.585909449	1
GaussianNB	0.58559322	0.528119933	0.528119933	1
LinearSVC-crammer-singer	0.602824859	0.36339781	0.36339781	0.333333333
KNeighborsClassifier-20	0.575423729	0.538376711	0.538376711	1

Table B.8: Model Scores for Variant 3 Model with 11 percent (V3p11) Set

Model	Accuracy	Precision	Recall-Score	Used Classes (n/3)
LogisticRegression-iter-10000	0.667834209	0.49742522	0.49742522	0.666666667
LogisticRegression-iter-100000	0.667834209	0.49742522	0.49742522	0.666666667
LogisticRegression-iter-1000000	0.667834209	0.49742522	0.49742522	0.666666667
LogisticRegression w/ sag	0.677174548	0.589250044	0.589250044	1
LinearDiscriminantAnalysis-svd	0.604786924	0.59331725	0.59331725	1
LinearDiscriminantAnalysis-lsqr	0.209573847	0.043921197	0.043921197	0.333333333
QuadraticDiscriminantAnalysis	0.49737303	0.460070063	0.460070063	0.666666667
SGDClassifier-hinge	0.671920607	0.451477302	0.451477302	0.333333333
SGDClassifier-log	0.671920607	0.451477302	0.451477302	0.333333333
SGDClassifier-modified-huber	0.671336836	0.525732501	0.525732501	0.666666667
SGDClassifier-squared-hinge	0.674255692	0.504854443	0.504854443	0.666666667
SGDClassifier-perceptron	0.611792177	0.547573733	0.547573733	0.666666667
LinearSVC-ovr	0.677174548	0.512894289	0.512894289	0.666666667
RadiusNeighborsClassifier-2600	0.659661413	0.552663683	0.552663683	1
DecisionTreeClassifier	0.624635143	0.63379474	0.63379474	1
GaussianNB	0.642148278	0.604917157	0.604917157	1
LinearSVC-crammer-singer	0.118505546	0.014043564	0.014043564	0.333333333
KNeighborsClassifier-20	0.637478109	0.594910907	0.594910907	1

Appendix C

Keras Models Tables

Table C.1: Keras Model Scores for Original (V0) Model Set

Model	Accuracy	loss
TF1	0.0849	9.3963
TF2	0.1367	7.8677
TF3	0.1671	2.1521
TF4	0.1513	2.1522

Table C.2: Keras Model Scores for Continuous Model Set

Model	Accuracy	loss
TF1	8.94E-02	2.84E-07
TF2	7.25E-02	2.84E-07
TF3	0.07249	0
TF4	0.07249	0

Table C.3: Keras Model Scores for Variant 1-1 (V1p1) Model Set

Model	Accuracy	loss
TF1	0.075846	11.116969
TF2	0.079607	8.746067
TF3	0.175094	2.141035
TF4	0.175094	2.142853

Table C.4: Keras Model Scores for Variant 1-11 (V1p11) Model Set

Model	Accuracy	loss
TF1	0.111154	10.440622
TF2	0.05481	7.79031
TF3	0.182829	2.116632
TF4	0.156765	2.1215

Table C.5: Keras Model Scores for Variant 2-1 (V2p1) Model Set

Model	Accuracy	loss
TF1	0.151834	7.36026
TF2	0.116796	6.805738
TF3	0.169227	2.150721
TF4	0.151834	2.17503

Table C.6: Keras Model Scores for Variant 2-11 (V2p11) Model Set

Model	Accuracy	loss
TF1	0.117811	8.532745
TF2	0.086581	8.072342
TF3	0.168719	2.151458
TF4	0.168719	2.148602

Table C.7: Keras Model Scores for Variant 3-1 (V3p1) Model Set

Model	Accuracy	loss
TF1	0.152542	11.032246
TF2	0.244633	6.456345
TF3	0.602825	0.941153
TF4	0.602825	0.937909

Table C.8: Keras Model Scores for Variant 3-11 (V3p11) Model Set

Model	Accuracy	loss
TF1	0.209574	10.83008
TF2	0.209574	16.118101
TF3	0.671921	0.852262
TF4	0.671921	0.85208

Appendix D

SciKitLearn Models Graphs

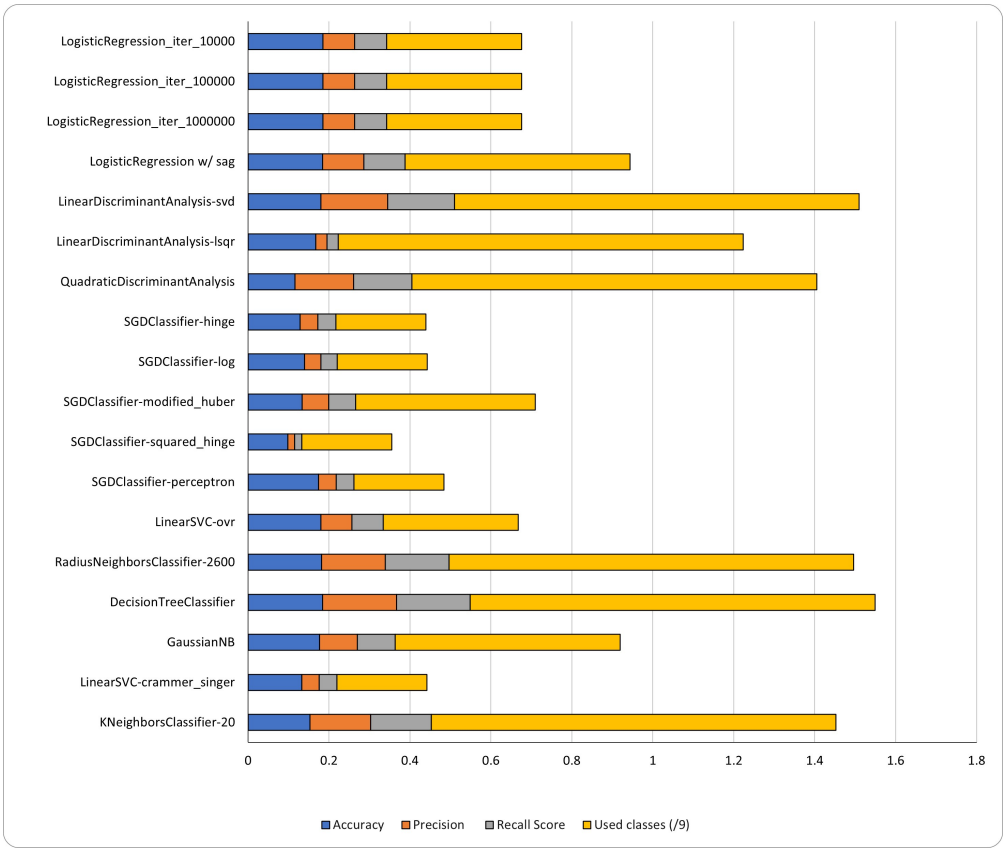

FIGURE D.1: First Pass (V0) Model Comparison

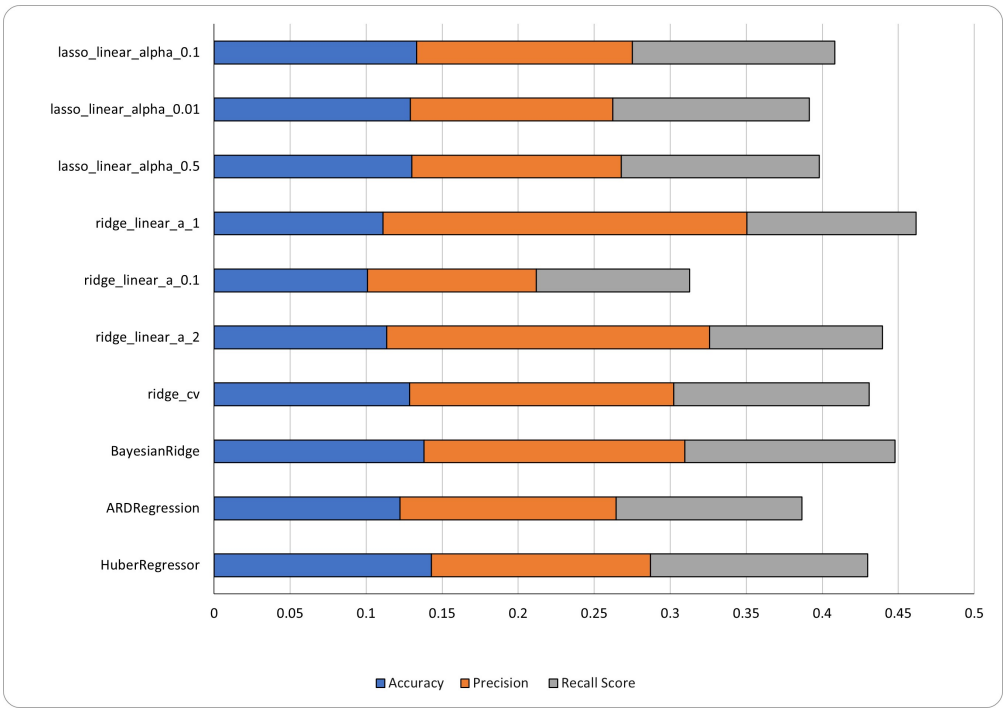

FIGURE D.2: Continuous Model Comparison

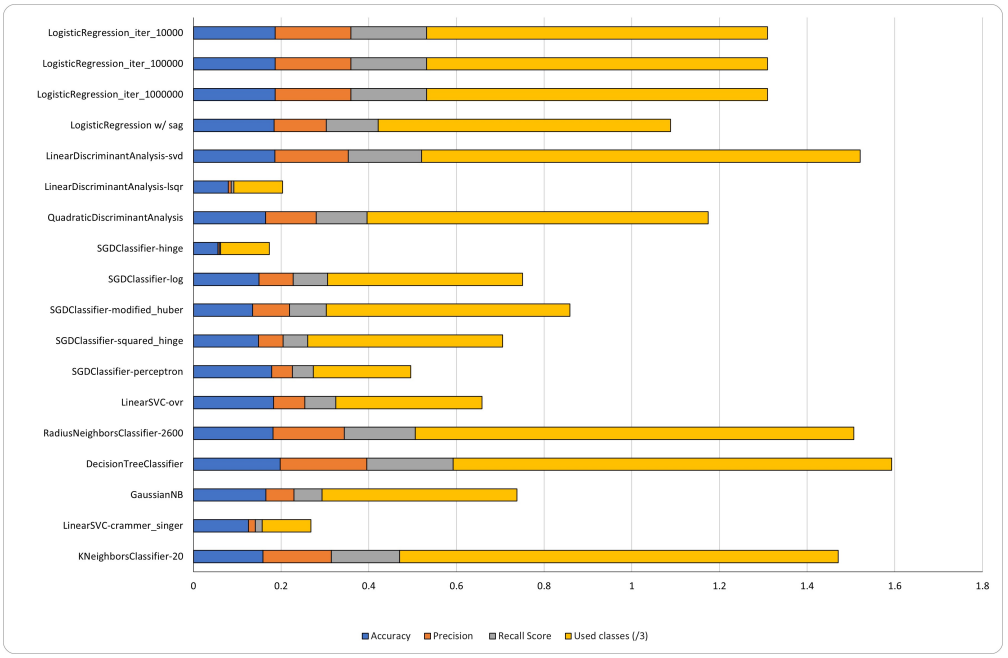

FIGURE D.3: Variant 1-1 (V1p1) Model Comparison

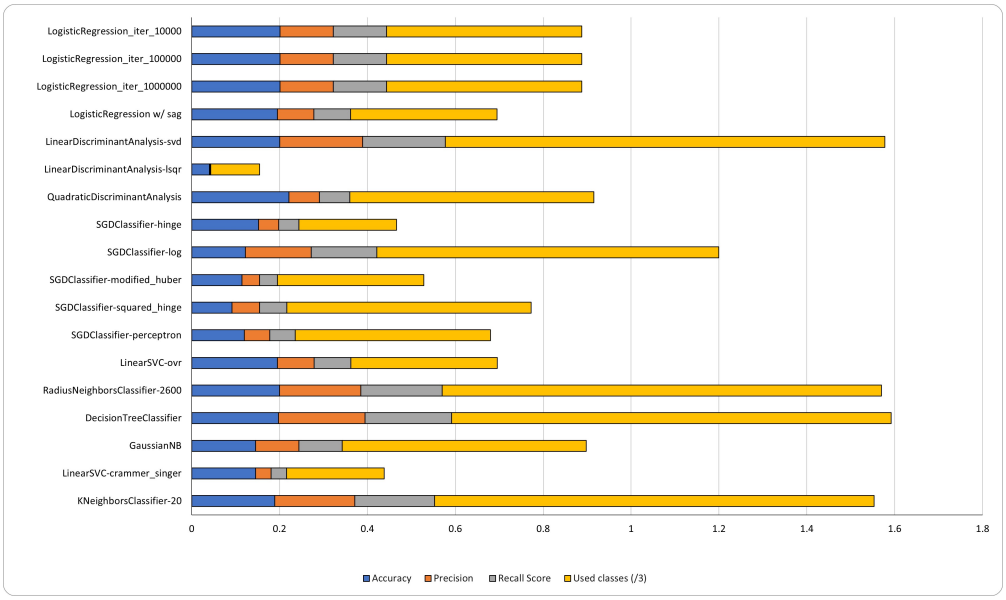

FIGURE D.4: Variant 1-11 (V1p11) Model Comparison

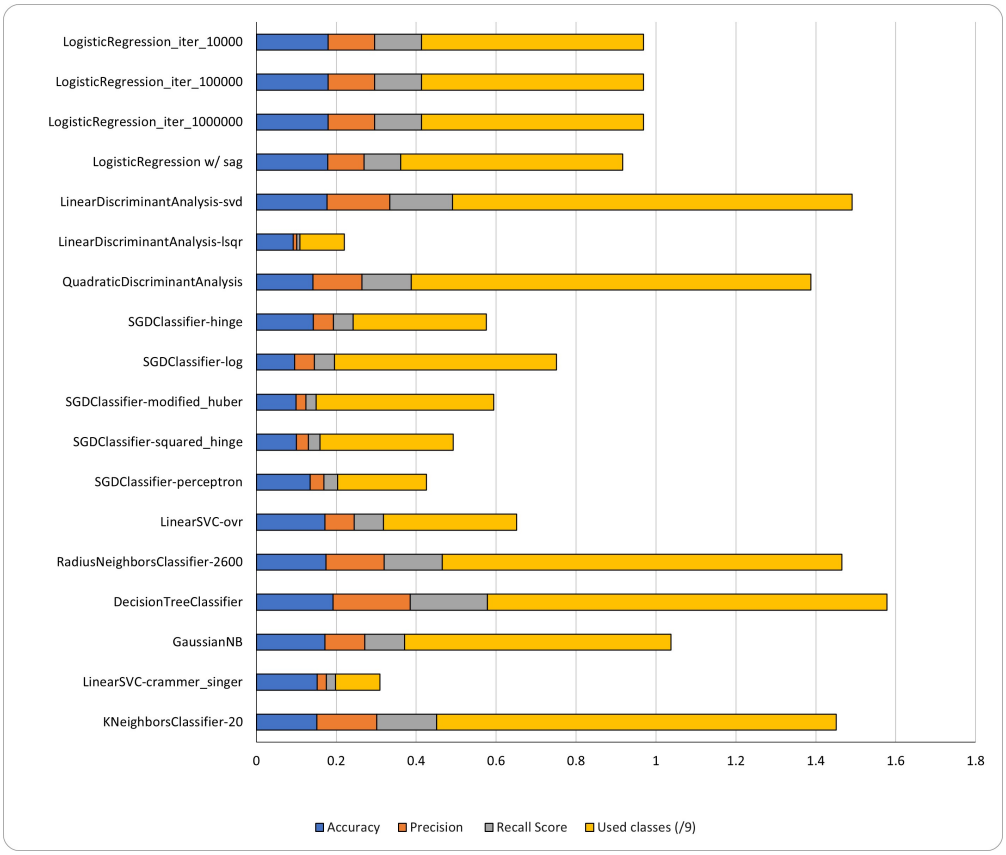

FIGURE D.5: Variant 2-1 (V2p1) Model Comparison

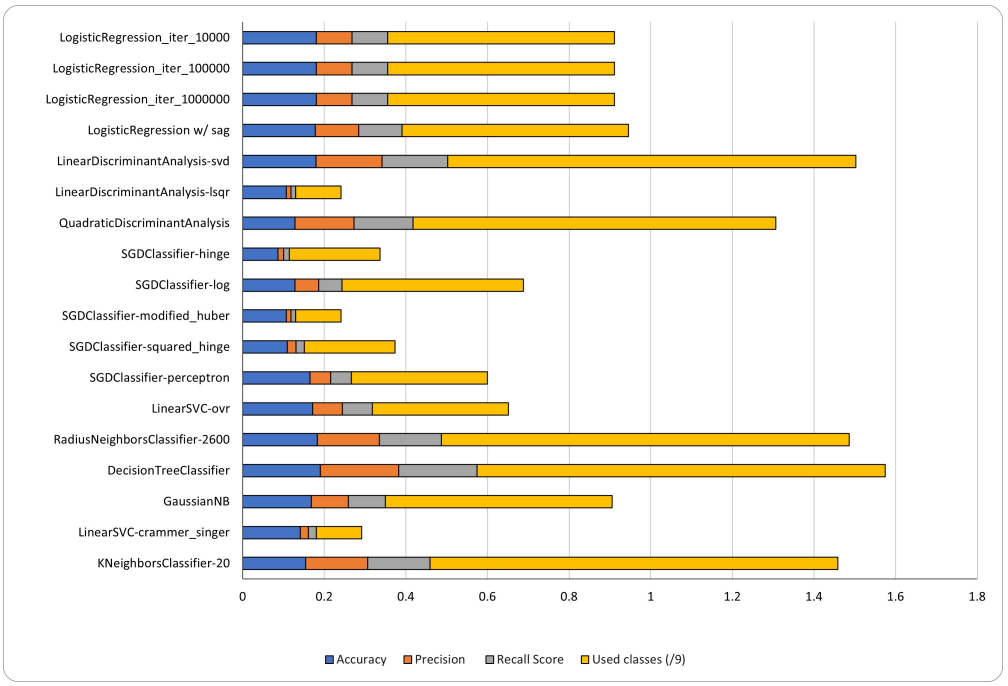

FIGURE D.6: Variant 2-11 (V2p11) Model Comparison

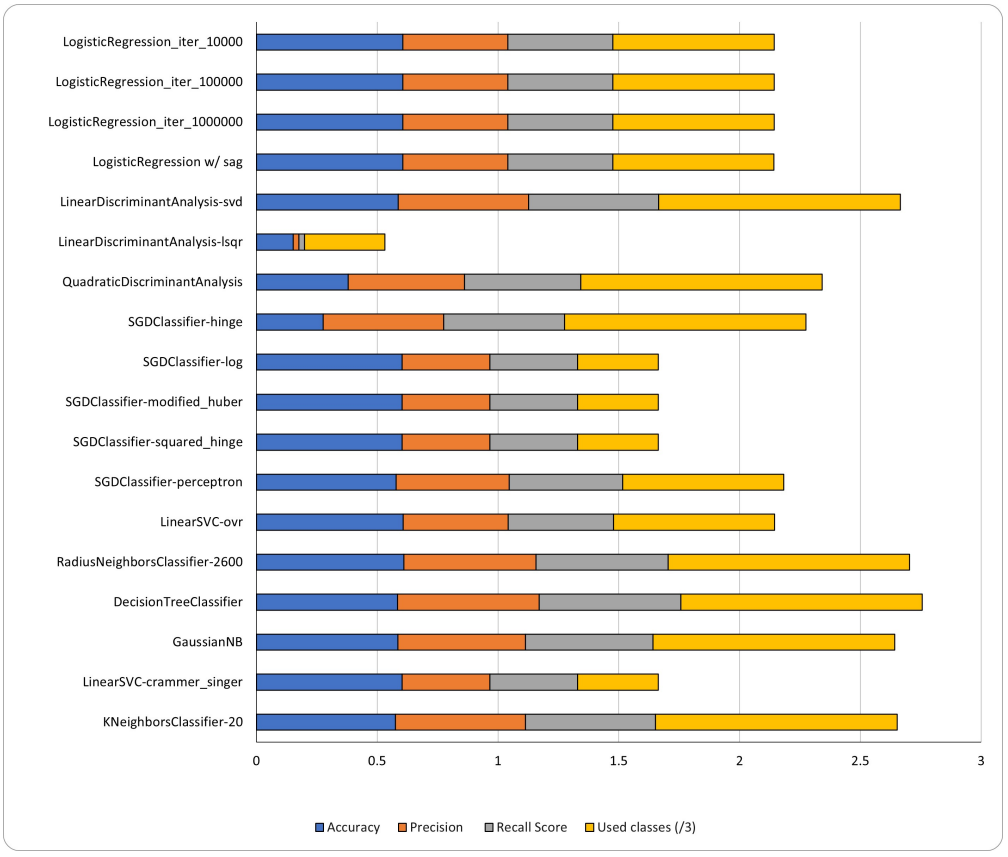

FIGURE D.7: Variant 3-1 (V3p1) Model Comparison

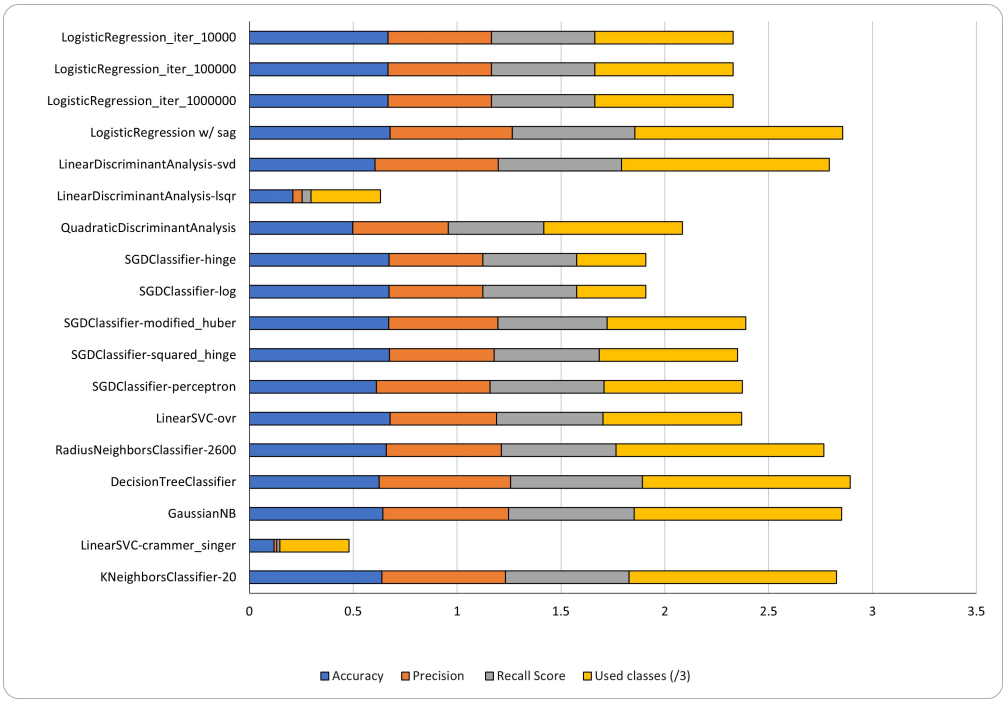

FIGURE D.8: Variant 3-11 (V3p11) Model Comparison