

**Hyperparameter Selection:**

**An Exploration of Uninformed vs Probabilistic Methods for Model Tuning**

# **Abstract**

**Student ID:** 1981101

**Student Name**: Miles Boyle

**Supervisor Name**: Dr Ian Batten

**Project Category/Topic:** Artificial Intelligence & Theoretical Machine Learning

­­­­­­­­­­­­­­­­

***A dissertation submitted in partial fulfilment for the degree of Master of Science in Computer Science***

***At***

***The University of Birmingham***

***School of Engineering, Physical Sciences & Computing***

Hyperparameter optimisation is the process of choosing a set of optimal hyperparameters for a machine learning algorithm. Hyperparameters are parameters given that control the learning process during the training phase. This paper examines and visualises the tuning process across 3 major optimisation regimes; Grid Search, Random Search & Bayesian Sequential Model Based Optimisation methods. Experimental results show that there are significant computational advantages to using a probabilistic approach to cover high dimensional Euclidean search spaces as opposed to uninformed strategies. Mathematical primers for all relevant statistical material are included. The full Jupyter Notebook code can be found at <https://github.com/MBStudent1/FinalProject/blob/master/SMBO%20Parameter%20Optimisation%20Python%20Notebook.ipynb>

# **Technical Notes**

Please note that in order to run the code on your own machine and see the visualisations you will require the following packages and API’s;

* Python 3 NOT Python 2
* TensorFlow Core v.2.3.0 with Keras Wrapper
* Matplotlib & Plotly for graphing

However, if you would prefer to run the code in the browser, copy and paste this link into your browser to access NB Viewer and press ‘Go’

<https://nbviewer.jupyter.org/github/MBStudent1/FinalProject/blob/master/SMBO%20Parameter%20Optimisation%20Python%20Notebook.ipynb>

Table of Contents

[**Abstract** 1](#_Toc49857638)

[**Technical Notes** 2](#_Toc49857639)

[**Introduction & Aims** 4](#_Toc49857640)

[**Theoretical Foundations & Previous Work** 5](#_Toc49857641)

[**The Nature of Tuning Machine Learning Models** 5](#_Toc49857642)

[**Grid Search & Cross Validation** 5](#_Toc49857643)

[**Random Search** 6](#_Toc49857644)

[**Bayesian Search** 7](#_Toc49857645)

[**Project Aims** 10](#_Toc49857646)

[**Methodology, Process & Execution** 11](#_Toc49857647)

[**Experiment 1: Grid Search with Support Vector Classifier (SVC)** 12](#_Toc49857648)

[**Experiment 2: Random Search with Random Forest Classifier** 13](#_Toc49857649)

[**Experiment 3: Probabilistic Reasoning with K-Nearest Neighbours & Tree-Parzen Estimator** 14](#_Toc49857650)

[**Results** 15](#_Toc49857651)

[**Experiment 1: Grid Search** 15](#_Toc49857652)

[**Experiment 2: Random Search** 16](#_Toc49857653)

[17](#_Toc49857654)

[**Implications: Bias-Variance Decomposition** 18](#_Toc49857655)

[**Experiment 3: Bayesian Approaches with Hyperopt** 20](#_Toc49857656)

[**Project Management & Limitations** 21](#_Toc49857657)

[**Successes** 21](#_Toc49857658)

[**Limitations** 22](#_Toc49857659)

[**References & Bibliography** 23](#_Toc49857660)

# **Introduction & Aims**

The major aims of this project were to model and visualise the hyperparameter tuning process during the training and testing phases when creating a machine model used for classification tasks. The target audience are novice machine learning practitioners in both academia and commercial organisations. Hyperparameter tuning is currently one of the most challenging areas within applied machine learning and often relies on the programmer possessing a significant amount of domain knowledge as to which tuning regime is most optimal. Through colourful and fully interactive 4D visualisations this opaque and mathematically dense topic can be brought to a wider audience.

Other work exists in this area however this project is unique in the sense that it directly compares the efficiencies of both random search and grid search but also explores computationally superior approaches by adopting a probabilistic approach to finding the optimum set of parameters and also visualises these results so hopefully individuals who are unfamiliar with machine learning tuning processes can achieve a more intuitive understanding.

Experimental results show that Bayesian optimization algorithm outperforms other global optimization algorithms. The results also show that there are significant computational savings with respect to the algorithm runtime from using probabilistic approach. These results could be used in research or engineering so individuals can identify the optimal tuning regime prior to building their models.

# **Theoretical Foundations & Previous Work**

## **The Nature of Tuning Machine Learning Models**

Schratz et al (2018) writes that whilst machine learning models have increased significantly in recent years due to their promising ability in prediction and classification problems, there remains significant challenges with regards to unbiased performance estimation. Tuning a machine learning models is the process of maximising a model’s performance by selecting appropriate hyperparameters whilst working without overfitting or creating a high model variance. Overfitting is the phenomenon that occurs when a model cannot generalise to future datasets since a high-density polynomial structure causes high model specificity to that particular feature set.

Hyperparameters can be conceptualised as ‘dials’ that are adjusted in order to achieve a fit between the model and the dataset. This project focusses on the process of choosing hyperparameters in an educational way through visualisations of this tuning process. Examples of hyperparameters are the ‘C’ regularisation parameter for Support Vector Machine algorithms or the ‘k’ value in K-Nearest Neighbours.

## **Grid Search & Cross Validation**

Syarif, Prugel-Bennet & Wills (2016) define Grid Search as an ‘*uninformed search technique based on defined subset of the hyper-parameter space. The hyper-parameters are specified using minimal value (lower bound), maximal value (upper bound) and number of steps’* (pp. 3). Essentially, Grid Search is an exhaustive approach to searching a defined space for the optimal set of hyperparameters. For example, let us imagine we had a neural network model that took the below parameters;

1. Number of hidden layers [2,5]
2. Number of neurons in each layer [5,10]
3. Number of epochs (iterations) [10,50]

If, for example, we needed to compare every single combination of parameter sequence we would require 2 ^ 3 = 8 attempts. Given that this is an extremely simple example that would be much too simple for any applied model, it is clear that this can be an extremely time consuming process to achieve by hand (if attempts > 1000 for example), hence an automated Grid Search.

Dufour & Neves (2019) describe Grid Search more formally, here we are given a search space of vectors over which we want to maximise the p-value (the model fit value). Conceptually, the algorithm is passed a parameter of lower bounds to search from

and a vector of upper bounds for each component of *V*. Grid Search takes equally spaced points in each interval of the form which creates a total of different combinations of values. The authors highlight here a crucial problem with this type of method since the number of evaluations increases exponentially a *n* and *m* increase.

Cross-validation (CV) is the process of reserving a part of the dataset to use in the evaluation of the model. This is required in the Grid Search process in order for the algorithm since, once it has identified a set of candidate parameters, the remaining data (<30%) is used to evaluate the model and prevent overfitting.

## **Random Search**

Random Search, as the name suggests, searches for candidate hyperparameters through randomly searching through the vector space. In their seminal paper, Bergstra & Bengio (2012) show empirically that random search is more efficient for hyper-parameter selection than Grid Search. Mathematically, this algorithm works as follows; Let be defined as the cost function which must be minimized and let ℝ*n* designate a candidate solution in the search space, the random search algorithm can be described as the following;

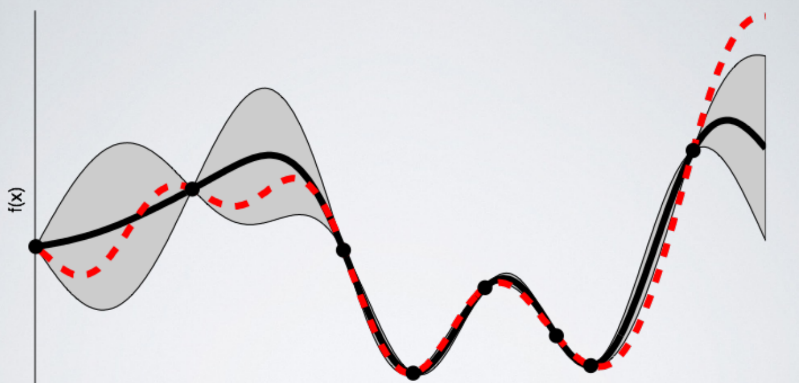
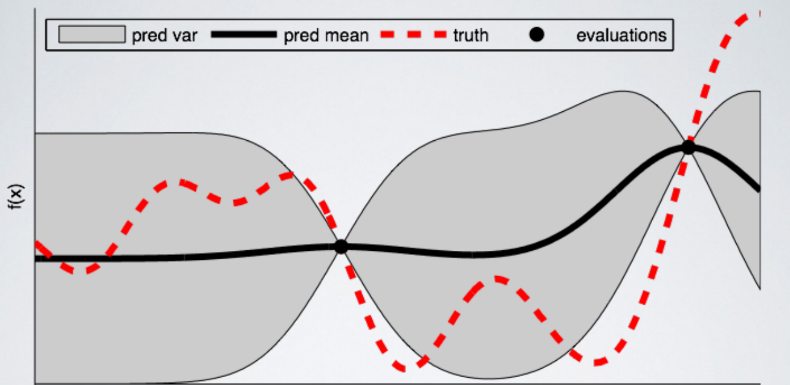
1. Initialize **x** with a random position in the search-space.
2. Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
   1. Sample a new position **y** from the radius surrounding the current position
   2. If *f*(**y**) < *f*(**x**) then move to the new position by setting **x** = **y**

## **Bayesian Search**

In comparison to uninformed search methods as per the two approaches above, Bayesian Search instead keeps track of past evaluation results and incorporates these data points into choosing the next sequence of vectors to investigate. The algorithm performs this by creating a probabilistic model mapping hyperparameters to a probability score on the objective function. (Joy et al, 2016)

)

The model, also known as the ‘surrogate’ since it ‘carries’ the probability to the next iteration, is represented as *P (y | x)*. The aim is for the surrogate function to build a probability distribution of the objective function, find the hyperparameters that perform optimally on the surrogate, given the Bayes Rule equation;

And these hyperparameters returned are applied to the surrogate function which is updated with the new results. The aim of Bayesian Optimisation is for the algorithm to incorporate prior states into the criteria from which it can search the next sequence of parameters. The reason why Bayesian tuning is superior to uninformed methods (from a theoretical standpoint) is because the algorithm allocates more computational resources to identifying the next hyperparameters by integrating prior beliefs and hence making fewer function calls to the objective function saving time.

*(Below)* **Fig 1**. *Initial estimate of the surrogate function, notice poor fit score*

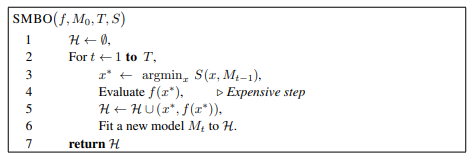
Source: Koehrson, 2018

*(Below)* **Fig 2**. *Surrogate function accuracy metrics after 8 evaluations, notice the exact matching*

Source: Koehrson, 2018

**Sequential Model-Based Decision Making & Optimisation (SMBO)**

SMBO methods are a family of Bayesian optimisation techniques that refer to running trials one after another and with each iteration the algorithm is updating the probability model by trying different combinations of hyperparameters based on Bayesian reasoning. Bergstra et al (2013) formalise the execution of SMBO algorithms as;

Briefly, the pseudocode given on the right describes the process of the surrogate model attempting to approximate *f*. The point that maximises the surrogate becomes the proposal for where the ‘true’ function should be evaluated.

*(Above)* **Fig 3**. *Pseudocode for a generic implementation of Sequential Model Based Optimisation*

Source: Bergstra et al, 2013

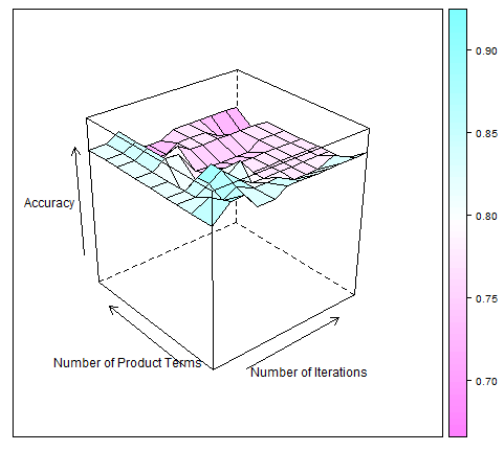
There are essentially five elements of a model-based optimisation search; (Bergstra et al, 2013)

1. A domain of hyperparameters over which to search
2. An objective function which takes in hyperparameters and outputs a score that we want to minimize (or maximize)
3. The surrogate model of the objective function
4. A criterion, called a selection function, for evaluating which hyperparameters to choose next from the surrogate model
5. A history consisting of (score, hyperparameter) pairs used by the algorithm to update the surrogate model

With Random Search & Grid Search as discussed previously, the domain of hyperparameters is essentially a grid which can be instantiated trivially in most programming languages. For a Bayesian approach to work however the domain must consist of probability distributions which can allow us to embed our own domain knowledge (if applicable) to guide the algorithm to search within high probability regions.

The objective function accepts hyperparameters as input and outputs a real-valued score that we want to optimise (). This step is usually the most computationally expensive part of the tuning process due to the extremely high number of candidate values to search through, Bayesian optimisation reduces this training time by choosing to run only the most promising set of hyperparameters given by previous calls to the objective/ surrogate function.

The surrogate function, to provide more detail, is the probability representation of the objective function established using previous calculations. The surrogate function is also referred to in the literature as the ‘response surface’ since is reflects a high-dimensional mapping of non-linear hyperparameters to a probability score on the objective function. The diagram (right) constructs a visualisation of this;



*(Above)* **Fig 4** The response surface for Adaboost algorithm

Source: Bergstra et al, 2013

There are several types of surrogate functions including Gaussian Processes however for this project we will focus on the Tree-Structured Parzen Estimator which will be discussed on the next page.

The selection function is the criteria that the algorithm accepts as an internal parameter which is used to select the next optimal set of hyperparameters. Bergstra et al (2018) note that the criteria of Expected Improvement (EI) is a robust choice since we can simply maximise EI with respect to *x*

where *y\** is the threshold value of the objective function, *x* is the proposed set of parameters which is taken from the last set of valid parameters that maximised the objective score, and *p (y | x)* represents the surrogate probability model. For example, if *p (y | x)* is equal to zero for all parameter searches then technically this would yield no improvement, likewise if the integral is positive then this will return a superior set of optimal hyperparameters.

For the experiments in this project, we have used the Tree Parzen Estimator algorithm (TPE). TPE constructs a model *p (y | x)* (the surrogate model) using Bayes rule given by *p (x | y) = p (x | y) \* p(y)/ p(x)* which in turn is expressed as;

where *y < y\** represents the lower bound of the objective function threshold, i.e. we make tow different distributions for the hyperparameters, one for when the value of the objective function < *l(x)* and one where the objective function *>g(x)*. Therefore, with Bayes rule and substitutions, the Expected Improvement equation is mathematically given by;

Note the right hand term *g(x) / l (x)*. This says that the Expected Improvement is proportional to the ratio *g(x) / l (x)* therefore to maximise the EI we should aim to maximise this ratio. TPE works by drawing sample hyperparameters from *l(x)* and evaluating them in terms of *l(x)/ g(x).* Each time the algorithm performs an iteration of EI calculations this is stored as a training history object which is then visualised to see how this optimisation process works on various models.

# **Project Aims**

The overall aim of this project was to provide an interesting and educational visualisation of the hyperparameter tuning process using three modes of search, random, grid & Bayesian. The target audience is not advanced machine learning practitioners but instead students and those with beginner level AI knowledge. Visualisation can provide much more clarity when attempting to learn a complex topic like hyperparameter selection as opposed to lengthy and opaque mathematical formulas. Initial research conducted led to the conclusion that whilst work existed on all three of these search methods, there was no single project that visualised all 3 approaches.

At the start of the project, the aims of the project were to initially visualise the results from just the inbuilt Reuters News dataset. Upon exploratory data analysis however, it was decided that in order to make this project more relevant to a wider audience it would perhaps be more helpful to use a range of datasets with the intent however of focussing on exclusively classification problems (not prediction/ diagnostic etc.)

The overall aims of the project were as follows;

1. Build and test at least 3 machine learning models for classification purposes
2. Study the effects of Grid Search and Random Search for model tuning
3. Visualise these results using the Plotly external API
4. Continue by extending the tuning regimes to Bayesian Optimisation methods
5. Visualise these results
6. Perform all necessary data pre-processing and feature scaling
7. Ensure the system/ notebook is reliable and functional
8. Analyse results from model tuning and make judgements around superior methodology

# **Methodology, Process & Execution**

As mentioned, the decision was taken early on to use separate datasets for each visualisation. The reasons behind this is that visualisations of high dimensional data, particularly the tuning process itself, are difficult to represent with one dataset. Feature scaling and data-pre-processing for separate datasets mean the transferability of the visualisations is sacrificed for higher interpretability i.e. this is **not** intended to be a decision making application due to different datasets however users can interact with a wide range of datasets to learn how they respond to different tuning regimes.

The decision was taken at the start of the project to use the Python programming language since it is the primary language as of 2020 for machine learning due to its powerful syntax and host of libraries and API’s which help programmers build models fasters. In addition to this, the code was written in a Jupyter Notebook, a web hosted program that allows easy and presentable coding specifically for data science and machine learning projects.

## **Experiment 1: Grid Search with Support Vector Classifier (SVC)**

Support Vector Machines, in supervised machine learning, are learning models with associated algorithms. An SVM constructs a hyperplane, a geometric space similar to a ‘barrier’, in high-dimensional Euclidean space that can be used for classification & regression. Support Vectors are essentially data points that lie close to the hyperplane that non-linearly separates two classes of data, we are looking to maximise the margin between the hyperplane and the data points by maximising the loss function. This gives us at least two classes of data that can form classifications.

For our first experiment we imported all the relevant libraries, these were;

* Matplotlib – a popular graphing library
* Sklearn – featuring various classification, regression and clustering algorithms
* Keras – an open source neural network library

Following this, we imported and framed the built in dataset MNIST. This is a popular dataset used in machine learning practice and research which consists of handwritten digits from 1 – 10 used for classification tasks. Before looking for which combination of parameter values produces the most accurate model, we specified the different candidate values we wanted to try. In the code below we have a number of candidate parameter values, including four different values for C (1, 10, 100, 1000), two values for gamma (0.001, 0.0001), and two kernels (linear, RBF). The grid search tried all combinations of parameter values and select the set of parameters which provides the most accurate model. Using the chosen algorithm (Support Vector Machine), a classifier object was constructed with the classifier and parameter candidates. The call to the fit function was made and the Grid Search object returned the associated values.

The values that are returned by the Gird Search CV object are represented as JSON-like structure. The author of this paper is fairly unfamiliar with retrieving data from this type of structure however the issue was resolved by declaring separate variables e.g. fit\_mean which corresponds to the nth key-pairing in the JSON object.

Using the Plotly external API these results were visualised and animated in a 4 dimensional model where the user can interact with the animation bar at the bottom of the visualisation in order to see how the training process changes with time.

## **Experiment 2: Random Search with Random Forest Classifier**

Random Forest Classifier is an ensemble learning technique for classification & regression tasks. It works in essentially 4 steps however the full algorithmic pseudocode is too advanced for this paper;

1. Select random samples from a given dataset
2. Construct a binary decision tree for each sample and return a prediction output
3. Perform a consensus vote between each tree
4. Select optimal tree which returns the highest prediction output score

Random Forest is comparatively slow to other machine learning algorithms due to the fact that there are multiple decision trees that must be processed in batch. Each single iteration takes steps 1 – 4 above and steps 3 involve a 1 – n mapping of all trees to the current tree when comparing the predictive function.

For this experiment, we used a randomised search approach to find the optimal sequence of hyperparameters. As given in the theoretical foundations, there was an expectation here that this would outperform the Grid Search algorithm however due to different datasets it is difficult to verify this empirically. The inbuilt Temperature & Rainfall dataset was used for this experiment and the Random Forest Classifier for training.

The initial state of the Random Forest Classifier was set to 50 and research conducted prior to the experiment noted that the following parameters have the most impact on the algorithm’s performance;

* n\_estimators = number of trees in the forest
* max\_features = max number of features considered for splitting a node
* max\_depth = max number of levels in each decision tree
* min\_samples\_split = min number of data points placed in a node before the node is split
* min\_samples\_leaf = min number of data points allowed in a leaf node
* bootstrap = method for sampling data points (with or without replacement)

Pre-processing steps for the temperature dataset were taken from <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74> since the project time constraints did not permit a focus on this part of the tuning process, instead most of the efforts were placed on producing visualisation of the results.

Following pre-processing, Random Forest Regressor was instantiated, and the search was executed by passing the feature scaled dataset using 3 fold cross-validation searching across 100 different combinations. This process took 8.3 min using all available core processors. Results were visualised using a similar technique to Grid Search.

## **Experiment 3: Probabilistic Reasoning with K-Nearest Neighbours & Tree-Parzen Estimator**

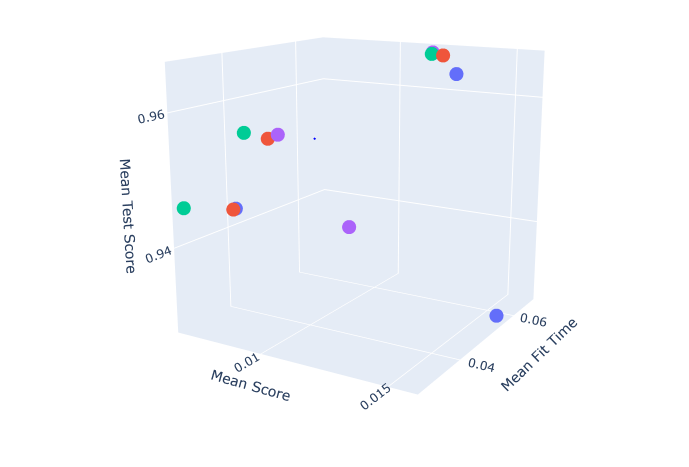
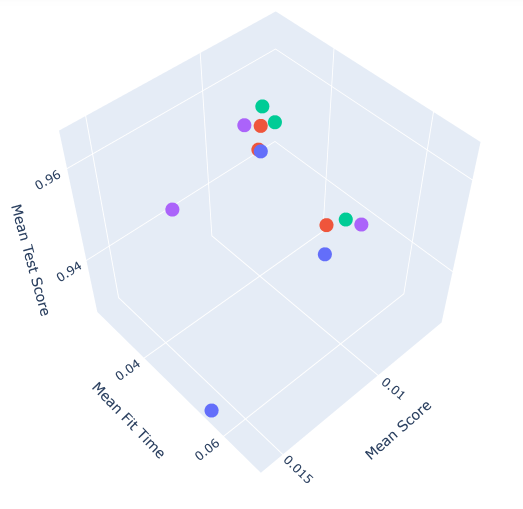
For our final experiment and visualisation, we used the inbuilt Iris dataset which stores 3 classes of 50 instances where each class corresponds to petal colour. Applications of this search process can be implemented in subsequent research in computer vision and object detection. The Tree Parzen Estimator algorithm was implemented through the Hyperopt API which provides implementation functions for Bayesian sequential model based optimisation regimes.

The function ‘hyperopt\_train\_test(params)’ accepts an arbitrary params argument which then instantiates the K-NN classifier (clf) and return the cross validation score for this particular iteration. We defined a space for K-NN as a search region with a 1, 100 range to avoid an unacceptably long training time. There is an internal call to ‘hyperopt\_train\_test(params)’ from the second function f(params) which declares a separate variable ‘acc’ which is assigned to ‘hyperopt\_train\_test(params)’. Lastly, the history object referred to in the theoretical basis section is called declared as a Trials object (an internal class of Hyperopt) which stores the optimal sequence of surrogate parameters.

There were some minor difficulties in extracting the data from the Trials object and visualising these results since instead of a JSON like object the data is returned as a sequence of lists which are partly stored as 1 dimensional arrays and partly n – dimensional arrays so the NumPy library was used here to recast all the required data structures.

# **Results**

## **Experiment 1: Grid Search**



*(Above)* **Fig 6**. *Visualisation of hyperparameter selection & tuning process - longitudinal view*

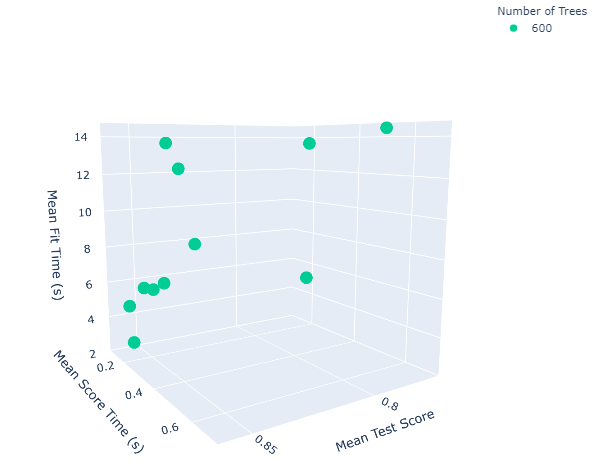
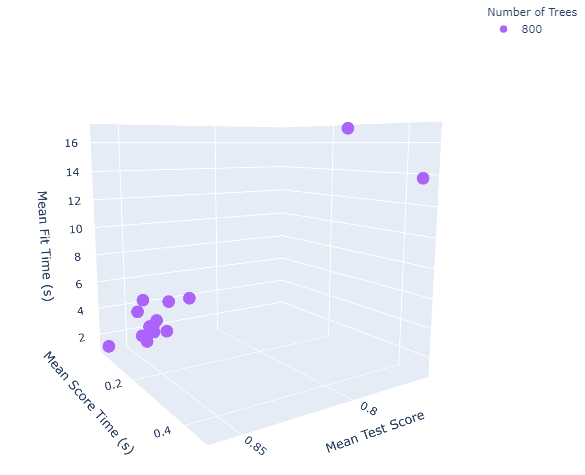
Source: GitHub Repository

*(Above)* **Fig 5**. *Visualisation of hyperparameter selection & tuning process - lateral view*

Source: GitHub Repository

In the two visualisations above, we can make some initial observations. Across the Y axis we have ‘mean test score’, across the Z we have Mean Fit time and across the X axis we have ‘mean score’. You can see from the image that as the Grid Search object increases ‘C’ which is our regularisation parameter we have an outlier where C=1 where the mean test score is a full 2% lower than its counterpart parameter which offer 96% test scores and a gradual tightening of all three metrics as C approaches one thousand given by the fact that the coloured dots are grouped in tighter groups as the tuning process continues most likely due to computational trade-offs occurring within the optimum finding process. These relationships do become clearer as we experiment with different search methodologies in subsequent models.

## **Experiment 2: Random Search**



*(Above)* **Fig 8**. *Visualisation of hyperparameter selection & tuning process – 400 trees*

Source: GitHub Repository

*(Above)* **Fig 7**. *Visualisation of hyperparameter selection & tuning process – 200 trees*

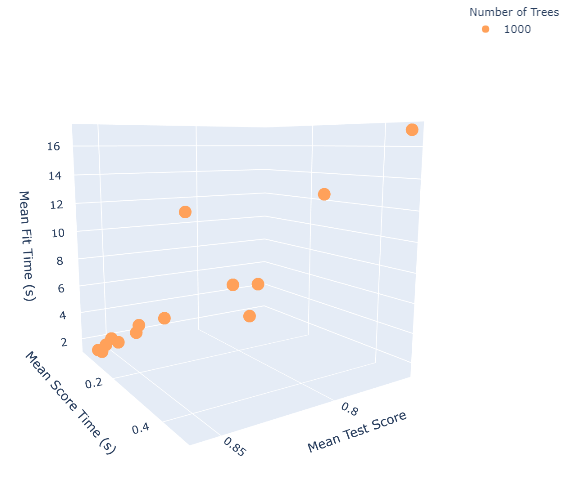
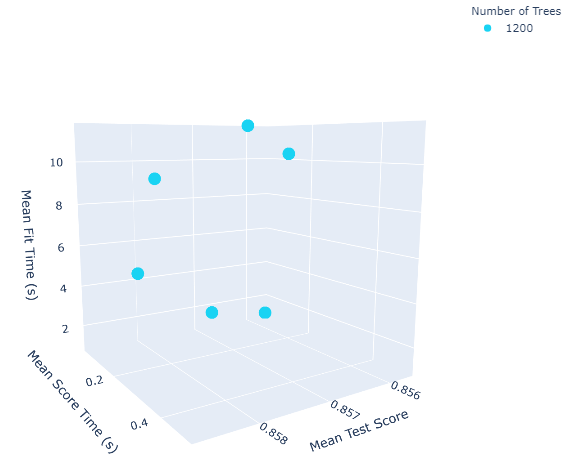
Source: GitHub Repository

*(Previous)* **Fig 10**. *Visualisation of hyperparameter selection & tuning process – 800 trees*

Source: GitHub Repository

*(Previous)* **Fig 9**. *Visualisation of hyperparameter selection & tuning process – 600 trees*

Source: GitHub Repository

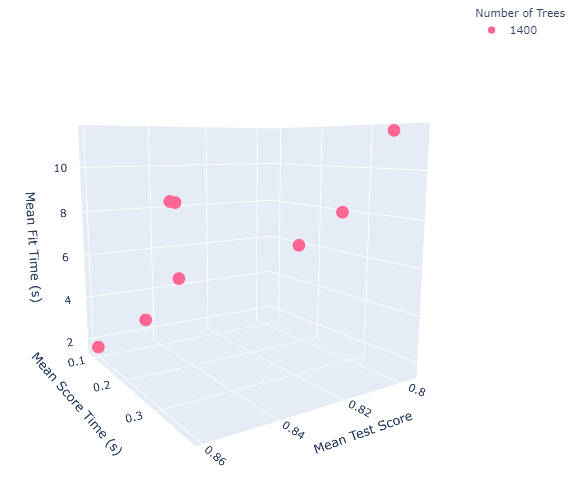
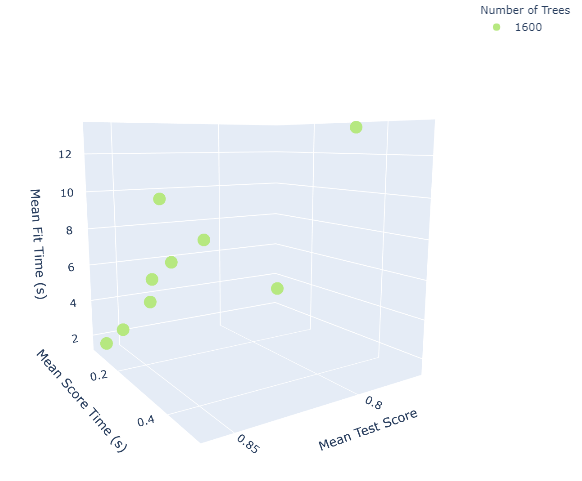


*(Above)* **Fig 12**. *Visualisation of hyperparameter selection & tuning process – 1200 trees*

Source: GitHub Repository

*(Above)* **Fig 11**. *Visualisation of hyperparameter selection & tuning process – 1000 trees*

Source: GitHub Repository

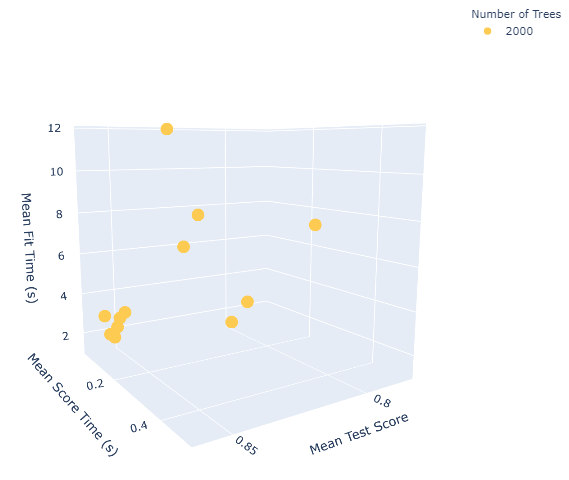
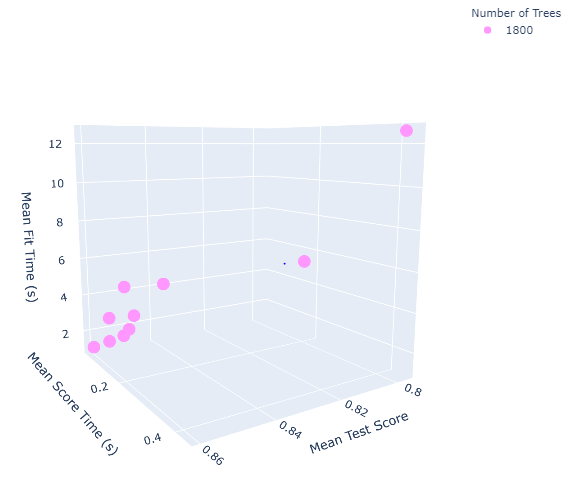


*(Above)* **Fig 13**. *Visualisation of hyperparameter selection & tuning process – 1400 trees*

Source: GitHub Repository

*(Above)* **Fig 14**. *Visualisation of hyperparameter selection & tuning process – 1600 trees*

Source: GitHub Repository



*(Above)* **Fig 15**. *Visualisation of hyperparameter selection & tuning process – 1800 trees*

Source: GitHub Repository

*(Above)* **Fig 16**. *Visualisation of hyperparameter selection & tuning process – 2000 trees*

Source: GitHub Repository

In the above diagrams, there are some initial observations that can be made.

Firstly, we can see from the coloured dots (which represent elements from the Trials object mentioned in the previous section) that as the number of trees approaches 400 there is an increasingly small variance in the data. This shows users that as n\_estimators increase, which represents the number of trees in the Random Forest, the model returns higher scores for mean score time & mean fit time. However, as the number of trees increases past 400, we see that this does not reliably improve the relevant scoring metrics.

## **Implications: Bias-Variance Decomposition**

One of the key lesson’s users can extract from this data is a deeper understanding around how models overfit to data and what the implications are for model accuracy and future generalisation. Bias-Variance decomposition is a feature of statistical learning theory where models with a lower bias in parameter estimation have a higher variance and vice versa. The bias error, also knowns as underfitting, occurs when a learning algorithm such as Random Forest misses the relevant feature mappings of train to test data – in other words it has not recognised the pattern in the data due to a high difference between the expected value and the true value of the parameter being estimated. Conversely, variance error is the error that results from small fluctuations in the training data set, this means that the algorithm models not the true pattern we are interested in but instead due to an overly complex polynomial structure models the noise within the data preventing future generalisation to different training datasets. (Friedman, Hastie & Tibshirani, 2013)

Mathematically, if we denote the variable we are trying to predict as *Y* and our covariates as *X* the aim is to derive the model that relates these variables to one another in the form *Y = f(x) +* where represents the error term that is minimised (Friedman, Hastie & Tibshirani, 2013). For all datapoints in our Random Forest example, the expected squared prediction error at any point *X* is;

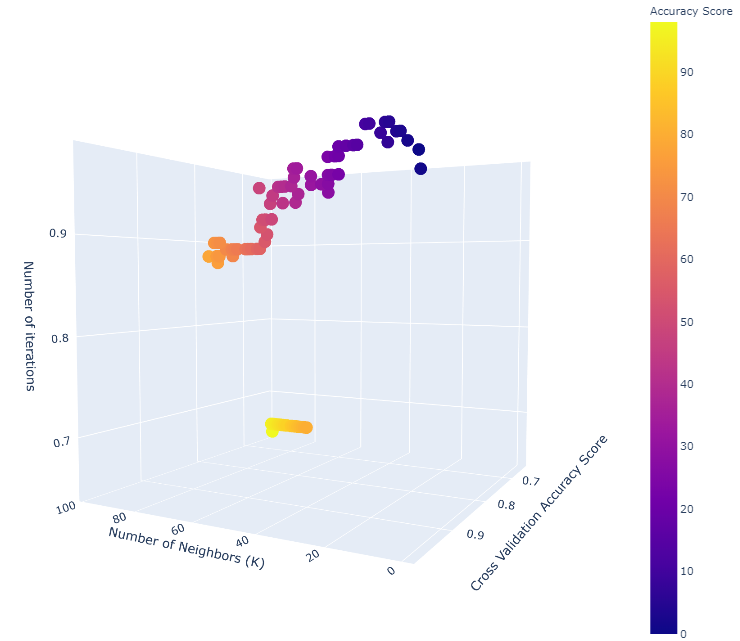
This error can be decomposed into its separate bias and variance components;

This can be conceptualised as;

*Err(x) = Bias ^ 2 + variance + irreducible error*

Where the third term, ‘irreducible error’, corresponds to the noise in the data that cannot be fundamentally reduced through more training. Technically, given the true model and infinite data to model it is possible to reduce both bias and variance to zero however given imperfect datasets this is rarely possible.

## **Experiment 3: Bayesian Approaches with Hyperopt**



*(Above)* **Fig 17**. *Visualisation of hyperparameter selection & tuning process – Bayesian Optimisation*

Source: GitHub Repository

In Figure 17 above we can make some initial observations regarding the distribution and variance of the testing data;

We notice severe overfitting/ high variance as K approaches 70 and a sharp, almost step-function from the high 60’s to the low 70’s. We can also observe that the data points are much closer together than our last 2 visualisations. This indicates that the algorithm is effectively incorporating prior information about the optimality of its previous state into how it will choose the next set of parameters.

Additionally, the cross-validation score is the lowest during the first 20 iterations of the fitting process and increases as the iteration count increases. If we combine this observation with also the fact that most of the data points are clustered very close together compared to our Gird Search and Random Search example we can conclude that this is strong evidence that the Bayesian element of parameter selection has worked, since that the Hyperopt function has successfully searched the parameter space efficiently by incorporating the prior probability distributions discovered in the preliminary training phase.

# **Project Management & Limitations**

There were several successes and limitations with respect to the management of the project which will be discussed within this section. Below is a risk and contingency table that received supervisor approval prior to starting the project.

|  |  |
| --- | --- |
| **Risk** | **Contingency** |
| *Inadequate findings* | It is a possibility that due to a combination of factors the research may not produce the results that were expected. This can be avoided by carefully selecting which variables to change and maintaining a ‘control group’ set of variables which may be formed of algorithm parameters or pre-processing steps. |
| *Computational runtime* | It is unlikely, but possible, that due to hardware requirements (laptop) the deep learning algorithms may not run. This can be avoided through a highly complex task where the algorithms can be run on sperate clusters. This would be complicated to implement therefore it is likely that dropout and batch normalisation would aid this problem. |
| *System errors and bugs* | Since the notebook is self-contained is unlikely that any persistent bugs will emerge. The usual debugging process will take place prior to demonstration to avoid this. |

## **Successes**

* The models were built and are stable using all relevant algorithms discussed in the aims
* All visualisations were built and fully functional
* The project was finished one week ahead of the presentation

## **Limitations**

* In future iterations, the author would prefer to use one dataset and explore multiple training regimes on this dataset since it could have more of a decision making purpose as opposed to a primarily educational one.
* Additionally, future recommendations are for developers to expand the visualisations and include a Graphical User Interface that would allow users to interact with the models without needing to run the entire Jupyter Notebook
* Finally, the visualisations would be clearer if they showed a slowed version of real-time model fitting however due to time constraints this was far beyond the scope of this project

# **References & Bibliography**

Barber, D., 2012. *Bayesian Reasoning and Machine Learning*. Cambridge: Cambridge University Press.

Barbero Jiménez, Á., López Lázaro, J. and Dorronsoro, J., 2009. Finding optimal model parameters by deterministic and annealed focused grid search. *Neurocomputing*, 72(13-15), pp.2824-2832.

Bergstra, J. and Bengio, Y., 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(6).

Bergstra, J., Bardenet, R., Bengio, Y. and Kegl, B., 2011. Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, 24(7).

Bishop, C., 2006. *Pattern Recognition and Machine Learning*.

Dufour, J. and Neves, J., 2019. Conceptual Econometrics Using R. *The Handbook of Statistics*, [online] 14(5), pp. 12 - 18. Available at: <https://www.sciencedirect.com/topics/mathematics/grid-search> [Accessed 2 August 2020].

Feurer, M. and Springberg, T., 2015. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. *Association for the Advancement of Artificial Intelligence*, 12(5).

Géron, A., 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*.

Hastie, T., Tibshirani, R. and Friedman, J., 2013. *The Elements of Statistical Learning*.

Hutter, F. and Kothoff, L., 2019. *Automated Machine Learning: Methods, Systems & Challenges*. Springer International Publishing.

T. T. Joy, S. Rana, S. Gupta and S. Venkatesh, "Hyperparameter tuning for big data using Bayesian optimisation," 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, 2016, pp. 2574-2579, doi: 10.1109/ICPR.2016.7900023.

Lambert, B., 2018. *A Student's Guide to Bayesian Statistics*.

Russell, S. and Norvig, P., 1994. *Artificial Intelligence: A Modern Approach*.

Snoek, J. and LaRochelle, H., 2012. Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 25.

Syarif, I., Prugel-Bennett, A. and Wills, G., 2016. SVM Parameter Optimization using Grid Search and Genetic Algorithm to Improve Classification Performance. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 14(4), p.1502.

Yang, L. and Shami, A., 2020. On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing*, 415, pp.295-316.