Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization

Hussain Alibrahim
Department of Computer Science
North Dakota State University
Fargo, USA
hussain.alibrahim@ndsu.edu

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, USA
simone.ludwig@ndsu.edu

***Abstract*—The performance of machine learning algorithms are affected by several factors, some of these factors are related to data quantity, quality, or its features. Another element is the choice of an appropriate algorithm to solve the problem and one major influence is the parameter configuration based on the problem specification. Parameters in machine learning can be classified in two types: (1) model parameters that are internal, configurable, and its value can be estimated from data such as weights of a deep neural network; and (2) hyperparameters, which are external and its values can not be estimated from data such as the learning rate for the training of a neural network. Hyperparameter values may be specified by a practitioner or using a heuristic, or parameter values obtained from other problems can be used etc., however, the best values of these parameters are identified when the algorithm has the highest accuracy, and these could be achieved by tuning the parameters. The main goal of this paper is to conduct a comparison study between different algorithms that are used in the optimization process in order to find the best hyperparameter values for the neural network. The algorithms applied are grid search algorithm, bayesian algorithm, and genetic algorithm. Different evaluation measures are used to conduct this comparison such as accuracy and running time.**

***Index Terms*—Hyperparmeter optimization, Grid Search, Bayesian, Genetic Algorithm**

I. Introduction

Classification is one of the important techniques in data mining; one reason is since classification aids humans in the understanding of the labels being classified. This results in a great deal of interest in classification from the academic community from statistics to machine learning to data mining and so on. Different domains have applied classification, including bioinformatics [1] image processing, [2], text classification [3], [4], and language processing [5].

A machine learning model has two types of parameters [6]:

- **Model Parameters:** consist of configurable variables used inside the model; their values can be estimated or learned from the data but are usually not manually provided and are incorporated as a part of the learning process. Some examples of model parameters include weights in an artificial neural network, support vectors in support vector machines, and coefficients in linear regression or logistic regression.
- **Hyperparameters:** are external parameters that are not part of the model and thus can not be predicted from the data set but can be configured by subject matter experts or by trial and error until an acceptable accuracy is achieved. Neural networks have many of these parameters such as the learning rate, optimizer, number of hidden layers, and size of each layer, etc.

For classification problems, to achieve a high degree of accuracy, a user has to successfully manage the hyperparameter configuration process, which is different for different algorithms and data sets. This process can be performed by using default values for the algorithm, manually setting them using previous examples or experiments, or asking experts to help in defining these parameters [6].

An alternative solution [7] involves the use of hyperparameter-tuning strategies that are data dependent, which try to minimize the expected generalization error of the algorithm over a hyperparameter search space of considered candidate configurations, usually by evaluating predictions on an independent test set or running a resampling scheme such as cross-validation. These search strategies range from easy, such as random search or grid search, to more complex ones like bayesian optimization or iterated forcing.

The choice of tuning strategy represents only one step in the hyperparameter optimization process; the user also has to define the hyperparameter lists and search range for each parameter, as well as to identify that parameter that may use default values. Both are important steps in the optimization process and any mistake in defining the parameters and setting the values may lead to wasting of computational time and resources or an unsuccessful optimization process.

II. Related Work

In [8], the authors highlight the importance of the concept of early termination, which is a good way to optimize hy-

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

perparameters since the normal optimization process requires large computational power and time due to the number of hyperparameters necessary, each of which could assume a range of values or even multiple values. The core idea behind the proposed approach instead involves training each model for only a short time and then extracting the information available to predict the final score of the model. The model was tested through experiments with a bank marketing data set that contained 45,211 records and 17 attributes. The results were compared to the random search and the Predictive Hyperparameter Optimization (PHO) algorithm (their proposed model). All results showed that PHO performed better than the random search.

In [9], the grid search and manual search were compared; these are most often used algorithms in hyperparameter optimization. The finding of the paper was that random search achieves better results than grid search for the hyperparameter tuning process, both empirically and theoretically. Empirical evidence stems from a comparison with a previous large study that used grid search, and manual search to configure neural networks and deep belief networks. Compared with neural networks configured by a pure grid search, the result was that the random search over the same domain was able to find models that were as good or better within a small fraction for the same computation time. Granting random search the same computational budget, the random search finds better models by effectively searching a larger configuration space.

In [10], random search, gaussian process approach (GP), and the tree-structured parzen estimator approach (TPE) were used in a hyperparameters optimization comparison study. Random search has been shown to be sufficient and efficient for learning neural networks on several data sets, but have been unreliable for training Deep Belief Networks (DBN). This test used a convex data set and Market Research Bureau of Ireland (MRBI) data set. TPE performed the best with test errors of 14.13% and 44.55%, respectively, while the random search resulted in test errors of 18.97% and 50.52%.

This paper investigates the performance of three algorithms for hyperparameter optimization, grid search, bayesian and genetic algorithm. These were chosen since these three approaches have not been compared with each other as of now. In particular, we are interested to see how the genetic algorithm optimization of the hyperparameters fares against the two other algorithms.

III. Optimization Approaches

This section introduces and describes the algorithms used in this study on hyperparameter optimization namely grid search, bayesian, and genetic algorithm.

A. Grid Search Algorithm

Grid search [8] is a systematic way to search over the search space for hyperparameters, and it will create all possible combinations regardless of the effects of the elements in the optimization process. All parameters have the same probability of impacting that process. While this method provides certain guarantees, it also has some major drawbacks. For instance, for an optimization with a large number of parameters and each parameter consisting of several values, this leads to a large number of combinations, that will result in an extensive amount of computations and time spent.

B. Bayesian Algorithm

The bayesian algorithm [11] is more dynamic than the grid search; it consists of two key components, which are the probabilistic surrogate model and the acquisition function. Bayesian is an interactive algorithm, in each iteration the surrogate model is fitted to all observations of the target function made so far. Then, the acquisition function searches for most parameters that improve the search and focuses on them to find the best set of hyperparameters. The Bayesian-based model attempts to predict how untested combinations will perform. Although many acquisition functions exist, the expected improvement is a common choice since it can be computed in closed form if the model prediction and configuration set follows a normal distribution.

C. Genetic Algorithm

Genetic algorithms [10] are metaheuristic optimization algorithms that resemble natural evolution. By relying on the evolutionary theory of the survival of the fittest as well as the ideas of selection and mutation, genetic algorithms aim to simulate the evolution of solutions over different generations so as to eventually identify an optimal or near-optimal solution for an optimization problem.

The algorithm evaluates the fitness of each solution set of parameters and selects the best individuals according to a fitness function, or a function defining the suitability of the solution to the problem. A set of genetic operators, such as crossover or mutation, is used to evolve from the best individuals until a new population is generated. The new population, in turn, evolves and is evaluated with the same mechanism until the termination condition is reached. In this way, the fittest or best individuals in the population are therefore identified and returned as the solution to the optimization problem.

IV. Data Set and Evaluation Metric

The Keras sequential class [12] was used to build the neural networks that can be used for binary classification over a data set. The aim of the experiment was to find the best combination of parameters that give best performance given the appropriate measure. The input layer dimension is fixed to 200, which is the same as the number of features, and the activation function is set to “relu”, while the output layer size was one with the “sigmoid” activation function. All other hyperparameters were passed during the training phase.

A. Data Set

This paper used the Santander Customer Transaction Prediction data set [13] to compare the three hyperparameter based algorithms. The data set has 200,000 records with 200 numeric features to predict whether the customer will make a specific transaction or not, regardless of the amount.

B. Evaluation Metric

In order to conduct a good comparison among the different algorithms for the hyperparameters optimization process, the following criteria have been chosen to be included in the evaluation; more explanation can be found in [14]:

1) **Accuracy:** The accuracy of a classifier is one metric for assessing classification models. Accuracy is the ratio of estimating the correct class compared to the overall number of classifications as defined:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{1}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives. Accuracy gives an estimate of the probability of a correct prediction; thus, the higher the accuracy, the better the model performance.

2) **Loss Value:** Opposite to accuracy, the loss value shows how bad the predication process is. This value is calculated based on a loss function, which is detailed in the following section on Hyperparameters search space. In general, it shows the difference between the predicted value of a model and the true value.

3) **Mean Squared Error (MSE):** MSE is the mean of the squared prediction errors over all instances. The prediction error is the difference between the true y_i and the predicted value $\lambda(x_i)$ for an instance n. MSE is calculated as:

$$mse = \frac{\sum_{i=1}^{n}(y_i - \lambda(x_i))^2}{n} \tag{3}$$

4) **Mean Squared Log Error (MSLE):** MSLE is a variation of the Mean Squared Error, where the focus lies in the percentual difference between the true y and the predicted $\hat{y}$ value. MSLE is calculated as:

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(\log(y+1) - \log(\hat{y}+1))^2 \tag{4}$$

5) **Area Under the Curve (AUC) [15]:** AUC is an experimental computation of the classification process performance based on the area under an Receiver Operating Characteristics (ROC) curve. ROC is a probability curve that is plotted with the true positive rate against the false positive rate. AUC assess the performance of a scoring classifier on a test set, but ignores the magnitude of the scores and only takes their rank order into account. The AUC value ranges between 0 to 1, where 0 means that all negatives are ranked before all positives, and 1 means that all positives are ranked before all negatives.

6) **Confusion matrix:** The confusion matrix summarizes the classification performance of a classifier with respect to test data. It is a two-dimensional matrix for binary classification, indexed in one dimension by the true class of an object and in the other by the class that the classifier assigns. Table I shows an example of this matrix. This matrix is used to evaluate the model prediction process.

TABLE I: Confusion Matrix

	True Class	
Predicted Class	Positive (c1)	Negative (c2)
Positive (c1)	True Positive (TP)	False Positive (FP)
Negative (c2)	False Negative (FN)	True Negative (TN)

7) **Wall time:** The wall time measures the total time to execute a program in a computer or basically the program running time only, which is different from the CPU or execution time that measures the time the CPU spends on running the program, which includes preparation and closing time for the CPU to run the job.

C. Search Space

Search space has been defined to cover different combinations of neural network hyperparameters. The search space covered are hidden layer number, hidden layer size, optimizers, loss functions, activation, dropout and validation split. The value range is different for each parameter, and in total there are 28,800 combinations. The details of each hyperparameter and its value are:

1) **Hidden layer number and size:** The major parameters in a neural network are the number of layers. As stated earlier, the input layer and output layer size was fixed to 200 and one layer, respectively. The input size is the number of features of the data set, and the output layer size is one since the network will decide whether the customer will make a specific transaction (1) or not (0). Hidden layers are the layers in the middle between the input and output layer. These layers perform nonlinear transformations of the input provided to the network. Based on the problem to be solved, these are the different layer numbers and sizes used in this paper:
   a) 1 hidden layer with 100 neurons;
   b) 1 hidden layer with 150 neurons;
   c) 2 hidden layers with 150 neurons and 75 neurons;
   d) 3 hidden layers with 150, 100, 50 neurons.

2) **Optimizers:** In this experiment, different optimizers are widely used by the deep learning community. The following optimizers are listed below; more details are available in [16].
   - **Stochastic Gradient Descent (SGD):** Gradient descent is a way to minimize an objective function J(θ) parameterized by a model's parameters $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate η determines the

size of the steps we take to reach a (local) minimum. Stochastic Gradient Descent (SGD) is an iterative algorithm that starts with random values and moves downside the slope in each iteration until it finds the lowest point of the function. SGD updates the parameter θ for each single training sample $x^{(i)}$ and labeled $y^{(i)}$:

$$\theta = \theta - \eta . \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{5}$$

Because of the multiple updates after each iteration, SGD is a fast algorithm compared to other gradient descent algorithms such as batch gradient descent, which perform the parameter update only after the training is completed for the whole training data set, and at the same time it has high variance that causes the objective function to fluctuate heavily.

- **Root Mean Square Propagation (RMSprop):** This is a gradient based optimization technique used for training neural networks. It was proposed by Geoffrey Hinton. Gradients of very complex problems have an inclination to either vanish or explode. Rmsprop was developed as a stochastic technique for mini-batch learning. It deals with this issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size, decreasing the step for large gradients to avoid exploding, and increasing the step for small gradients to avoid vanishing. Rmsprop updates parameter θ_t at time t as:

  $$\theta_{(t+1)} = \theta_t - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} g_{(t)} \tag{6}$$

  where η represents the learning rate, $E[g^2]$ is the running average for the squared gradients, and ϵ is the smoothing term used to avoid the divide-by-zero error.
- **Adaptive Gradient (Adagrad):** This is a gradient-based optimizer that adapts the learning rate to the parameters. It performs the adaption based on the frequency of the parameter occurring features. It will assign a small learning rate value for parameters associated with regularly occurring features, and a larger value for parameters associated with rarely occurring features. The main advantage of this optimizer is that there is no manual update of the learning rate. The update action can be represented as:

  $$\theta_{(t+1)} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_{(t)} \tag{7}$$

  where G_t is the diagonal matrix that contains the sum of gradients for all parameters up to time t, and $g_{(t)}$ represents the gradient at this time.
- **Adaptive Moment Estimation (Adam):** This algorithm is an extension of stochastic gradient descent, and combines the advantages of the RMSprop and Adagrad algorithms described earlier. Adam, instead of adapting the parameter learning rates based on the average past squared gradients only υ_t, it also makes use of the average of the exponentially decaying average of past gradients m_t. Adam's update rule is the following:

  $$\theta_(t+1) = \theta - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} \hat{m}_t \tag{8}$$

- **Adamax:** It is a variant of the Adam optimization algorithm, but instead of using two norms to adapt the parameter learning rate, Adamax generalizes this limitation to ∞. The update rule for Adamax is:

  $$\theta_(t+1) = \theta - \frac{\eta}{u_t} \hat{m}_t \tag{9}$$

  while u_t is the max value:

  $$u_t = max(\beta_2 . u_{t-1}, |g_t|) \tag{10}$$

  with initial value $u_0 = 0$. Note that, conveniently enough, Adamax does not need to correct for initialization bias. Also note that the magnitude of parameter updates has a simpler bound with AdaMax than Adam, namely: $|\triangle_t| \leq \alpha$.
- **Nesterov-accelerated Adaptive Moment Estimation (Nadam):** As Adam can be viewed as a combination of RMSprop and Adagrad, Nadam can be seen as a combination of Adam and Nesterov accelerated gradient (NAG). NAG is a way to calculate the next position of parameter $t + 1$ in addition to the current position t so then NAG can decide to stop or the move ahead to the next position. After incorporation of NAG into Adam, this is the update rule:

  $$\theta_t + 1 = \theta - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} (\beta_2 \hat{m}_t + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t} \tag{11}$$

  where β represents the decay rate.

3) **Loss Function:** The loss function is the method that evaluates how well an algorithm models a data set. If the predictions are mostly wrong then the loss function will return a large number, which means the model has to be improved. On the other hand, if the return value was small then this means the model is predicting very well. There are many different functions and for this experiment the following loss functions were used:
   - **Mean Squared Error (MSE):** as previously stated it is the mean of the squared prediction errors over all set instances. The prediction error is the difference between the true (y_i) and the predicted value $\lambda(x_i)$ for an instance (n):

     $$mse = \frac{\sum_{i=1}^{n} (y_i - \lambda(x_i))^2}{n} \tag{12}$$

   - **Mean Absolute Error (MAE):** is the mean of the absolute values of the individual prediction errors over all instances in the test set. Each prediction error is the difference between the true value (y_i) and the predicted value $lambda(x_i)$ for instance n.

$$mea = \frac{\sum_{i=1}^{n} abs(y_i - \lambda(x_i))}{n} \tag{13}$$

- **Mean Absolute Percentage Error (MAPE):** as given in [17], MAPE is the most common measure used to predict errors, and works perfectly if there are no extreme values in the data. MAPE is a statistical measure of how accurate a prediction model is. It measures the accuracy as a percentage between the true value y_i and the predicted value $lambda(x_i)$ for the instance n:

$$mape = \frac{\sum_{i=1}^{n} |\frac{y_i - \lambda(x_i)}{y_i}|}{n} \tag{14}$$

- **Mean Squared Logarithmic:** is a variation of the Mean Squared Error, where the focus lies on the percentual difference between the true y and the predicted $\hat{y}$ value:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (\log(y+1) - \log(\hat{y}+1))^2 \tag{15}$$

- **Squared Hinge [18]:** This loss function is used for "maximum margin" binary classification problems. The hinge loss guarantees that during training the classifier will find the classification boundary, which is the furthest apart from each of the different classes of data points. In other words, it finds the classification boundary that guarantees the maximum margin between the data points of the different classes. Mathematically it is defined as:

$$L(y, \hat{y}) = \sum_{i=0}^{n} (max(0, 1 - y_i.\hat{y}_i)^2) \tag{16}$$

- **Log Cosh:** As per tensor flow documentation, Log-cosh is the logarithm of the hyperbolic cosine of the prediction error, and is another function used in regression tasks, which is smoother than the mean square error. Log-cosh works mostly like the mean squared error, but will not be so strongly affected by an occasional incorrect prediction. Mathematically it is defined as:

$$L(y, \hat{y}) = \sum_{i=0}^{n} \log(\cosh(\hat{y}_i - y_i)) \tag{17}$$

- **Binary Cross entropy:** Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

$$\mathrm{L} = -\frac{1}{n} \sum_{i=1}^{\mathrm{n}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log\ (1 - \hat{y}_i) \tag{18}$$

- **Kullback-Leible Divergence class [19]:** The KL divergence measures the distance between two density distributions. This divergence is also known as information divergence and relative entropy. If the densities P and Q exist with respect to a Lebesgue measure, the Kullback-Leibler divergence is given by:

$$L = \sum_{i}^{n} y_i \log \frac{y_i}{\hat{y}_i} \tag{19}$$

4) **Activation Function:** Is a function provided to an artificial neural network in order to assist the network in learning a complex model from data. The activation function helps to restrict the output value of data within the required data range. Furthermore, the activation function is used to remove linearity from the neural network. There are many different functions [12]:
   - **Rectified Linear Units (ReLu):** is a non-linear activation function, which is widely used in neural networks. The function is defined as: $f(x) = max(0, x)$.
   - **Exponential Linear Unit (ELU):** is a variant of the Rectified Linear Unit. ELU introduces a parameter slope for the negative values of x. It uses a log curve for defining the negative values:

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \tag{20}$$

   - **Exponential:** The equation of the exponential function is $f(x) = 2^x$.
   - **Scaled Exponential Linear Units (selu):** is similar to ELU with minor changes. The function is modified using α and λ as fixed value parameters derived from the input as per [20]:

$$f(x) = \lambda \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \tag{21}$$

   - **Sigmoid** is the most widely used activation function and is a non-linear function. The sigmoid function transforms the values in the range 0 to 1. It can be defined as: $f(x) = \frac{1}{e - x}$. The sigmoid function is continuously differentiable and has a smooth S-shaped function.
   - **Hyperbolic Tangent function (Tanh):** is similar to the sigmoid function but it is symmetric around the origin. Symmetric results in different signs of the output from previous layers, which will be fed as input to the next layer. It can be defined as: $f(x) = 2sigmoid(2x) - 1$, the output values lie in the range -1 to 1. As compared to the sigmoid function, the gradient of the tanh function is more steep. Tanh is preferred over the sigmoid function as it has gradients which are not restricted to vary in a certain direction and also is zero centered.

5) **Dropout:** is at technique that removes or ignores some units in a neural network. Dropout is dropping connections in the network by dropping incoming and outgoing edges of certain neurons. This is a random process and the number specified means the probability of each neuron to be removed from the network. The main purpose of this technique is to prevent overfitting and thus improving the performance. Overfitting means the neural network too closely models the input data. Model combination typically improves the performance of machine learning models. Averaging the predictions of several models, which are the result of dropout, is most helpful when the individual models are different from each other and each model is fast to train and is used at test time. The dropout value used in this paper ranged between 0.0 to 0.4.
6) **Validation Split:** As per keras official documentation, it is a float value between 0 and 1. The fraction of the training data is to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x and y data provided before shuffling. The validation split value used in this paper was ranging between 0.2 and 0.4.

V. Results

Table II shows the best parameters found by each algorithm, as well as the wall time and the number of runs. These results are based on the validation of the training data. Table III shows the results in terms of accuracy, MSE, MSLE, and AUC based on the test data.

A. Grid Search Algorithm

Grid search tries all possible combinations of parameters to identify the best parameters to achieve the best accuracy on the binary classification problem. As shown in Table II, Grid search finds the best number of hidden layers to be 3, which is the highest number in the search space for this parameter. The optimizer is Adam and the activation function is sigmoid.

Figure 1 shows the loss value at the first run was higher than 0.9, and then at the end after all training cycles were completed the best value for this data set was 0.092. The best model found by grid search was tested and the accuracy was 0.8976 as shown in Table III, which has a higher loss value in testing than the one achieved during validation.

From the confusion matrix in Figure 2, the model was tested on 50,000 records, which around 45,000 were correctly predicted; both true positives and true negatives. TPR is 0.91, FPR is 0.43, sensitivity is 0.91, and specificity is 0.02.

Figure 3 shows the AUC curve, which as discussed before measures the relation between TPR and FPR using different thresholds. The AUC value is 0.792. This value will be compared to other model AUC values in order to determine the best model.

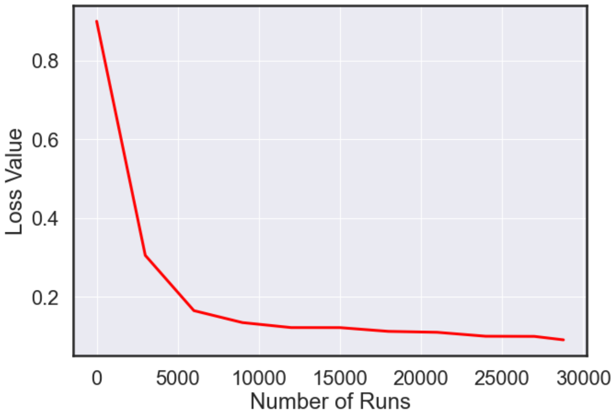

Fig. 1: Loss of Grid Search

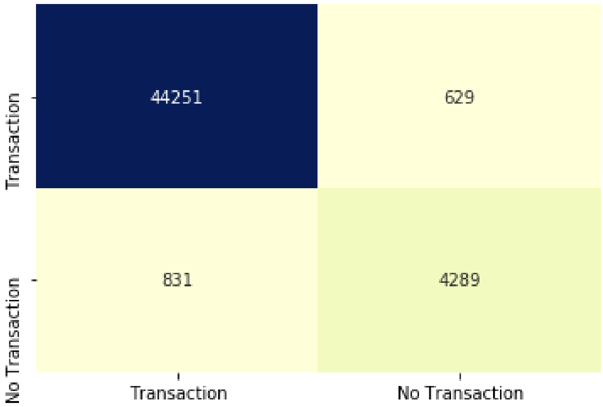

Fig. 2: Confusion Matrix of Grid Search

B. Bayesian Algorithm

The values of the best model achieved by the Bayesian algorithm are given in Table II. It consists of one hidden layer with 100 neurons, the optimizer is RMSP, and the activation function is elu.

The Bayesian algorithm consumed more time than the genetic algorithm to find the best parameters. It stopped earlier than the grid search; after 28,000 runs.

The loss value as given in Figure 4 dropped from above 0.9 to 0.0998, whereby the best model found achieved good accuracy on the testing data set with a value of 0.8959; the MSE and the MSLE values were 0.1041 and 0.0502, respectively (Table III).

The confusion matrix for the Bayesian approach given in Figure 5 shows the same results on the test set as the grid search. The Bayesian algorithm had lower positive and negative true values around 500 records. The algorithm has the highest false positive rate compared to the other two algorithms with a value of 0.45. The specificity is 0.02, and sensitivity is 0.92.

Figure 6 shows the AUC for the Bayesian algorithm and the area is 0.789, which is lower than the AUC of the grid search algorithm.

TABLE II: Summary of Best Parameters

	Grid Search	Bayesian Algorithm	Genetic Algorithm
Number of hidden Layer	3	1	3
Size of hidden Layers	150-100-50	100	150-100-50
Optimizer	Adam	Root Mean Square Propagation	Adam
Loss Function	mean squared error	Kullback-Leible Divergence class	Binary cross entropy
Activation Function	sigmoid	elu	relu
Drop out	0.0	0.1	0.0
Validation Split	0.15	0.4	0.25
Time [h:m:s]	16:30:40	23:10:40	11:58:40
Number Of runs	28,800	28,000	20,000

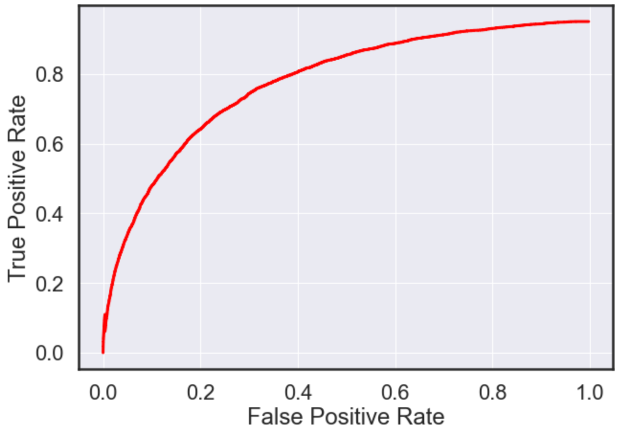

Fig. 3: AUC of Grid Search

TABLE III: Accuracy, MSE, MSLE and AUC of all Algorithms

	Grid Search	Bayesian Algorithm	Genetic Algorithm
Accuracy	0.8976	0.8959	0.9059
MSE	0.0924	0.1041	0.0744
MSLE	0.0455	0.0502	0.0352
Precision	0.81	0.79	0.89
Recall	0.90	0.84	0.91
F1-score	0.85	0.81	0.87
AUC	0.792	0.789	0.826

C. Genetic Algorithm

The genetic algorithm's best parameters, as given in Table II, are 3 hidden layers with 150, 100 and 50 neurons in each layer, with Adam as optimizer, and relu as the activation function. This algorithm stopped earlier compared to the other algorithms. It stopped after around 20,000 cycles and thus was the fastest algorithm.

The genetic algorithm performed very well in solving this problem and achieved the lowest loss value of 0.092 as given in Figure 7. The best model tested achieved values for accuracy, MSE, and MSLE with 0.9059, 0.0744 and 0.0352, respectively (Table III). Precision, recall, and F1 score were measured as well and provided in the table.

The confusion matrix in Figure 8 shows a lower false prediction rate of 0.27, and a lower specificity value of 0.01 while the other algorithms' value was 0.02.

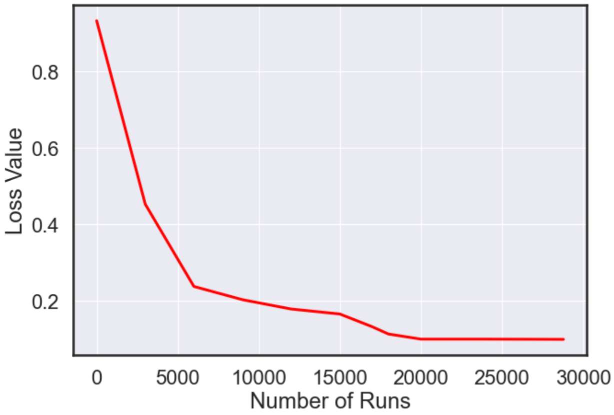

Fig. 4: Loss of Bayesian Algorithm

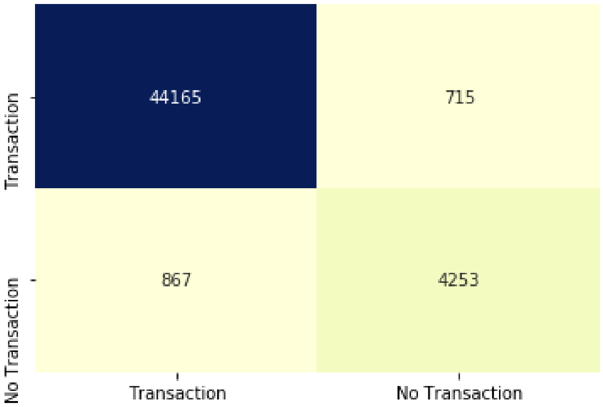

Fig. 5: Confusion Matrix of Bayesian Algorithm

The AUC in Figure 9 shows the best curve among the algorithms since it has a higher area value with 0.826.

Unfortunately a direct comparison with past research results cannot be made, however, putting it into perspective in [21] the same data set was investigated. The authors used different machine learning algorithms and applied them to the Santander Customer Transaction Prediction data set. The differences compared to our work are that they used 16,000 instances for the test set, while we used 50,000 instances. Furthermore, the authors used undersampling and oversampling before applying the different classifiers. The undersampling technique can be defined as balancing the target distribution by random

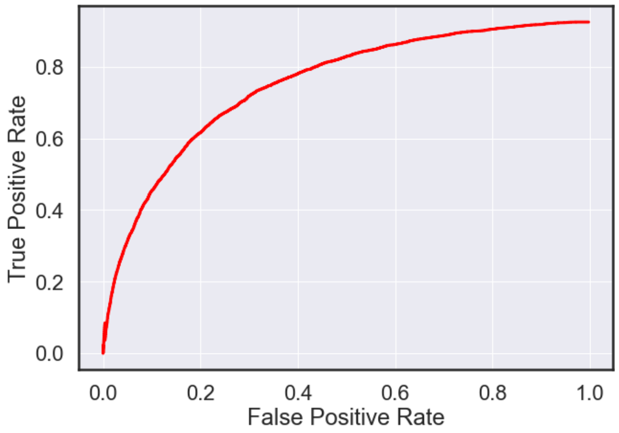

Fig. 6: AUC of Bayesian Algorithm

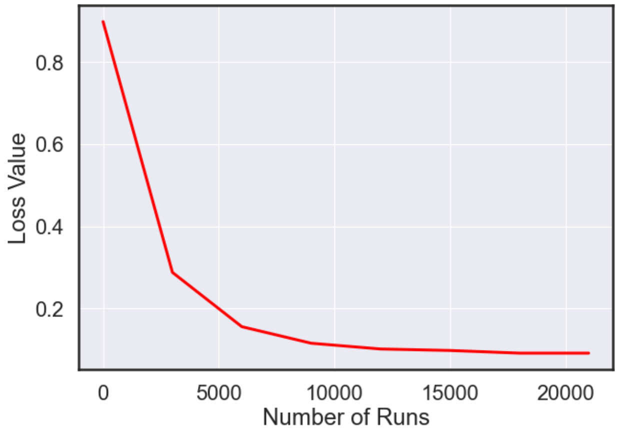

Fig. 7: Loss of Genetic Algorithm

repetition of minority target instances, on the other hand oversampling balances the target distribution eliminating random samples from the majority class instances. Oversampling and undersampling help with class imbalances in the data set and is usually the preferred technique used as a pre-processing method in the data mining life-cycle and has shown to help in achieving higher classification accuracy.

TABLE IV: Results after under sampling [21]

Models	Accuracy	Precision	Recall	F1	AUC
SVM (Linear Kernel)	0.74	0.74	0.73	0.74	0.74
SVM (Poly Kernel)	0.74	0.78	0.67	0.72	0.74
SVM (RBF)	0.50	0.00	0.00	0.00	0.50
Naıve Bayes	0.77	0.77	0.75	0.76	0.77
Logistic Regression	0.74	0.74	0.73	0.74	0.74
Decision Tree	0.59	0.59	0.59	0.59	0.59
Random Forest	0.75	0.75	0.76	0.75	0.75
Gradient Boosting	0.73	0.75	0.70	0.73	0.73
Ada Boosting	0.68	0.70	0.63	0.66	0.68

Table IV summarizes the results of the algorithms used in [21] with the undersampling technique, and Table V shows the results of the same algorithms after applying the oversampling technique. The classification results after undersampling was applied whereby the Naıve Bayes algorithm achieved the highest accuracy, Precision, F1 score and AUC with values of 0.77, 0.77, 0.76, 0.77, respectively. Random Forest obtained the highest recall value of 0.76, and the oversampling has better results with the random forest algorithm achieving around 0.99 for all classification results. Gradient Boosting and Decision Tree achieved values that are in the same range as our approaches while all other algorithms had lower results.

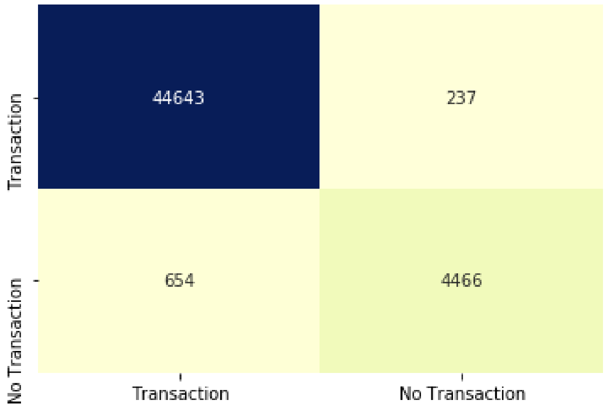

Fig. 8: Confusion Matrix of Genetic Algorithm

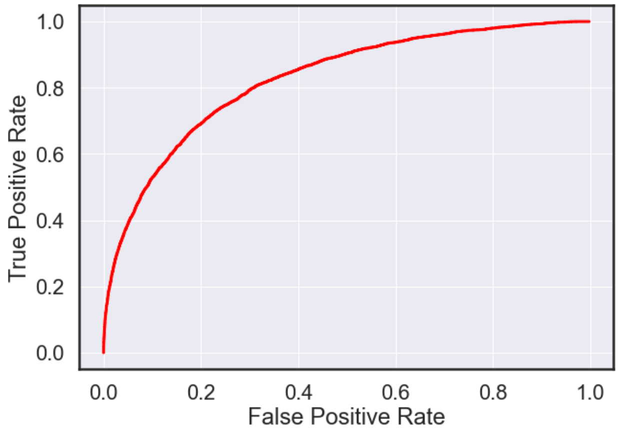

Fig. 9: AUC of Genetic Algorithm

TABLE V: Results after oversampling [21]

Models	Accuracy	Precision	Recall	F1	AUC
SVM (Linear Kernel)	0.73	0.74	0.73	0.73	0.73
SVM (Poly Kernel)	0.73	0.74	0.72	0.73	0.73
SVM (RBF)	0.80	1.00	0.60	0.75	0.80
Naıve Bayes	0.76	0.77	0.75	0.76	0.76
Logistic Regression	0.73	0.74	0.73	0.73	0.73
Decision Tree	0.94	0.90	0.99	0.94	0.94
Random Forest	0.99	0.99	0.99	0.99	0.99
Gradient Boosting	0.93	0.92	0.94	0.93	0.93
Ada Boosting	0.68	0.70	0.64	0.67	0.68

VI. Conclusion

The objective of this paper was to compare three different algorithms for finding the best hyperparameters for a neural network. The algorithms used were Grid Search, Bayesian

Algorithm and Genetic Algorithm. The Santander Customer Transaction Prediction data set was chosen for the experiments in order to determine if a customer will do a transaction or not. The evaluation process focused on loss and accuracy, MSE, MSLE, AUC, confusion matrix, and wall time. The hyperparameters used were hidden layer number and size, optimizers, loss function, activation function, drop out, and validation split with different values each. The results of the experiments were similar, however, the genetic algorithm performed better than the grid search followed by the Bayesian algorithm. For the best hyperparameters the first two algorithms (Grid search and Bayesian) had the same number of hidden layers as well as the same optimizer, while all other parameters were different.

REFERENCES

[1] N. Chu, J. Li, Methodology study of classification algorithm in TCM Zheng diagnosis, IEEE International Conference on Bioinformatics and Biomedicine, pp. 22-25, 2014.

[2] S. Buluswar, B. A. Draper, Nonparametric classification of pixels under varying outdoor illumination, Proc. SPIE – Int. Soc. Opt. Eng., pp. 1619-1626, 1994.

[3] T. Joachims, Text Categorization with Support Vector Machines: Learning with many Relevant Features, Springer, 1998.

[4] R. M. Silva, T. A. Almeida, A. Yamakami, Mdltext: an efficient and lightweight text classifier, Know. Based Syst., pp. 152-164, 2017.

[5] W. G. Lehnert, S. Soderland, D.B. Aronow, F. Feng, A. Shmueli, Inductive text classification for medical applications, J. Exp. Theor. Artif. Intell., pp. 49-80,1995.

[6] G. Luo, A review of automatic selection methods for machine learning algorithms and hyper-parameter values, 2016.

[7] O. Mersmann, H. Trautmann, C. Weihs, Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation, pp. 249-275, 2012.

[8] D. Marinov, D. Karapetyan, Hyperparameter Optimisation with Early Termination of Poor Performers, 2019.

[9] J. Bergstra, Y. Bengio, Random Search for Hyper-Parameter Optimization, Journal of Machine Learning Research 13, pp. 281-305, 2012.

[10] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzia, L. Simonetto, Genetic Algorithms for Hyperparameter Optimization in Predictive Business Process Monitoring, Publication: Information Systems, January 20, 2018.

[11] J. Bergstra, R. Bardenet, Y. Bengio, B. Kegl, Algorithms for Hyper-Parameter Optimization, Neural Information Processing Systems Conference, 2014.

[12] F. Chollet, and others, Keras, 2015, keras web page: https://keras.io.

[13] Santander Bank, Santander Customer Transaction Prediction Version 1, Retrieved from https://www.kaggle.com/c/santander-customer-transaction-prediction/data, 2018.

[14] W. Meira, M. J. Zaki, Data Mining and Analysis: Fundamental Concepts and Algorithms, ISBN: 9780521766333, 2014.

[15] C. Sammut, G. I. Webb, Encyclopedia of Machine Learning and Data Mining, publisher: Springer US, 2017, isbn:978-1-4899-7687-1.

[16] S. Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747, 2017.

[17] A. de Myttenaere, B. Golden, B. Le Grand, F. Rossic, Mean Absolute Percentage Error for Regression Models, Neurocomputing, Elsevier, Advances in artificial neural networks, machine learning and computational intelligence, 2016.

[18] C. Lee, C. Lin, A Study on L2-Loss (Squared Hinge-Loss) Multiclass SVM, Neural Computation, pp. 1302-1323, 2013.

[19] F. Pérez-cruz, K. Leibler, Divergence Estimation of Continuous Distributions, Proceedings of IEEE International Symposium on Information Theory, pp.1666-1670, 2018.

[20] Z. Huang, T. Ng, L. Liu, H. Mason, X. Zhuang, D. Liu, SNDCNN: Self-normalizing deep CNNs with scaled exponential linear units for speech recognition, 2020.

[21] R. Mohammed, J. Rawashdeh , M. Abdullah, Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results, 2020 11th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2020, pp. 243-248, doi: 10.1109/ICICS49469.2020.239556.