Vemund Brynjulfsen
HIOF
ITI41720

Trees and GA for hyperparameter optimization

Abstract

In this paper a Genetic algorithm (GA) for hyperparameter tuning was developed. This paper shows that the made GA was capapble of creating good hyperparameter tuning results.

The machine learning models used were the tree models random forest and xgboost from scikit-learn.

The following dataset has been used:
The dataset is called **Loan Defaulter Classification**.

Introcution

The purpose of writing the algorithm is to learn about how to make a genetic algorithm. The baysian optimization algorithm is used for comparison.

Related work

This paper takes inspiration from Ghatasheh et al. (2022) to produce the GA. The GA produced in that paper represents individuals partly as a combination of parameters. Crossvalidation is done by exchanging the parameter values between individuals, and mutation is done by sliglhty tweaking the parameters values. This paper produces a less complex version of that algorithm, by only representing the parameter values.

Methology

Dataset

The goal of the dataset is to predict wether a person will default on a loan.
The Bank Indessa has not done well. Many of their NPA was from loan defaulters. They have collected data over the years, and can use machine learning models to analyze their data. They want to find the defaulters using the data.

There is a row for each individual person. Each person has an id called **member_id**. The target feature is called **loan_status**. It has a value of 0 and 1. 0 means that the person has not defaulted on the loan, 1 means that the person has defaulted.
It has a train set: train-indessa.csv, and a test dataset: test_indessa.csv. Only the train_indessa.csv was used. The test_indessa.csv dosen't include a target column, which makes it difficuilt to get insightfull accuracy results. Therefore the test_indessa.csv file wasn't used. Rather the train_indessa.csv set was split into a train and test set.

Data preprocessing

The steps for cleaning the data, and making it suitble for the tree models, are shown in Cleaning dataset.

Genetic algorithm (GA)

An individual is represented as a list. Each index is a holds a parameter value for a specific parameter. The 1st index holds the parameter value for parameter x, 2nd index holds the parameter value for parameter y.

Crossover is done by switching the parameter values between individuals.

Mutation uses Gaussian mutation. The parameters to be mutatet are randomly selected. If 10 parameters are specified, then the maximum mutation rate is 10, meaning that 10 parameter values will be mutated.

Parent selection is done by using roulette wheel. Each pair of parents has to be unique. Two different kinds roulette wheels are tested. One is a the traditional roulette wheel that selects individuals in accordance to the fitness. The other more heavily favours the best individuals. the likelyhood of selecting the best individual is 50%.

Image GA Algorithm explained gives insight into how to algorithm works. Table GA structure gives a overview of the algorithm.

The genetic algorithm has parameters that can be tuned. These are:
The population size, the mutation rate, the kind of roulette wheel, amount of iterations.

Baysian optimization (BA)
This algorithm is used to get something close to the optimal parameter values. This is used to compare against the GA, to see if the GA produces good results. It should produce close to the optimal parameter values. The best parameter value is found one by one for each parameter. When finding the best value for a parameter, the model uses all the previous best values found for the other parameters.

A good algorithm to be used for this could've been Grid Search. But runtime is tied to the amount parameters used. Grid Search scales badly with the amount of parameters. This means that only so few parameters can be used, that it can't be proparly compared to GA.

Experiments

Parameters
The tried parameters for each algorithm are shown in tables Parameters tried. I've tried to make the parameters similair for the GA and BA.

Parameter optimalization of GA
The GA has parameters that can be varied. These are the mutation rate, population size, and iteration size. It's intresting to see what effect population size and iteration size has. These GA parameter combinations will be tried: Large population size with small iteration size, small population size with large iteration rate. Both with a medium mutation rate.
I have tried using these combinations on the XGB. If they create a large difference, then I will also conduct such an experiment on RF. If no remarble difference occurs, I will use the best combination for RF, without testing the other combination.

Metrics
For GA and BA, accuracy is used for fitness.
T-test will be used to see if there is a significant difference between the best fitnesses when varying the population size and iteration size.
Each GA parameter combination will only be run 10 times, so the t_test may be faulty, as it's recommended to have 30 samples.
The BA is only run once, as it outputs the same results every time.

Data diminishing
The original dataset consists of ~500.000 rows. I've found that I can use a lower amount of rows, and get similair results on the test set. The test dataset will stay consisten at 20% of the full dataset.
The results are shown in table Varying dataset training size RF
At 100.000 rows, the model complains about unseen data in the test set, ~150.000 rows of the dataset can be

used without being concerned about the performance. This reduces training time, which means that the solution space can be increased by providing more parameter values.

Results

GA XGB

Table XGB GA varying population size and iteration size shows the results of the GA parameters. In the experiment, varying population size and iteration size gave no different results. This might be because of the dataset. It might be that the accuracy on the dataset can't be improved more than a certain amount on for both three models, and so using different GA parameter combinations have no effect.

XGB accuracies also shows the accuracy gotten using BA and the default parameters (DEF).

The BA produced the best accuracy (better than the GA mean), which is expected. But the GA has a close accuracy for the mean accuracies.

GA RF

The table in Random Forest accuracies shows the results for random forest using GA, BA, and DEF. The GA RF uses a high population size and small iteration size.
The parameters aren't varied here, as the experiment with XGBoost didn't give evidence for a difference when varying the GA parameters on the three, on this dataset.

GA gave the lowest accuracy, for the mean and the best accuracy. But later it will be seen that the BA and default model has a lot more overfitting.

Generalization ability

Overfitting is when a model is too specilized on it's training data, so it's ability to handle more general cases not in the training data might be worse. When a model has a lot better metrics on a train dataset, than on a test dataset, then this can indicate overfitting.

A lot of parameter configurations have been produced with GA for both trees. The configuration that had an accuracy close to the mean is shown in the tables in Varying dataset size for train accuracies on GA, BA and DEF. The accuracies on test data and training data is listed for the GA (for XGB, one with a large population and few iterations), BA, and DEF. The accuracies are shown as more data is added to the training set.

For all instances, the accuracy on the test dataset goes up a bit,
All the instances has a bit higher accuracy on the train dataset than on the test dataset.
But on BA and DEF for RF, the accuracy is a lot higher on the train dataset, than on the test dataset. This makes it seem that these models have a lot of overfitting.

Costs

In the graphs in TN, TP, FN, TP the TN, FP, FN, TP values are shown as a percentage of all predicted values on the test dataset, as the weights of the classes change, for all the model variants with the full amount of rows in in the dataset.
In this dataset's case, FN means that a person is indicated as not defaulting on a loan, when the person actually did. Making a wrong rediction like this can mean giving out loans to people that will default on them, which can be costly for the bank.

Weight 0.0 for label 0 gives the lowest FN, but then we also see that TN becomes so low, and FP becomes so high that the model becomes useless, it can ONLY predict FN cases. It can be hard to lower the FN value using weights, because the TN value is lowered sharply when doing so.

For the GA model, TN, FP, FN, TP values don't stagnate as fast when changing the weights. The GA RF model also had the lowest overfitting, while the other two RF models had high overfitting, the GA RF model is a bit different from the other RF models.

The XGB variants have less difference between them in the cost graphs, as none of them seemed to have high overfitting or very different accuracies. For the XGB models, the different values don't grow as sharply, so it's more possible to lower the FN, without drastically lowering the TN.

Parameters of models

The tables in Produced parameter values with GA and BA shows the parameters for the models discussed,

For the GA algorithms, the chosen parameter values are the same as those used to compare against the BA and DEF. When a parameter value is not included, it is because the BA didn't find a better value than the default value.
Eta is a parameter in XGBoost. Eta is how much each tree contributes to the prediction. A low eta value means each tree has a smaller impact, and that more trees are needed for good predictions. The GA had a eta value of 0.6, and the sequential model a eta of 0.3.

The GA model had a lot higher value for n_estimators in both tree models. n_estimators is how many trees that are in the model. Having more trees can give better generalization. This might have contributed to the GA RF's better generalization.

Gamma is a parameter in XGBoost. Gamma is minimum loss reduction for further partition on leaf node of a tree. A larger value means that fewer splits are allowed, and so the trees become simpler. Both GA and BA parameters had higher values.

Lambda is a parameter in XGBoost. It applied L2 regularization on weights. It penelizes larger weights. A lower value means more complex trees, but also a higher risk of overfitting.

Alpha is a XGBoost parameter. It applies L1 regularization.

three_method is a parameter in XGBoost. tree_method can be either exact, approx, or hist. Hist uses histograms to determine best split points. It is faster than the other methods. Exact uses exact greedy algorithm to find best splits. It genereally gives better results but is also slower. Approx uses approximate greede algorithm. It gives a balance between speed and accuracy.

The GA RF model had a lot higher max_depth than the BA RF model. max_depth is the manimum depth each tree can have. A shallower tree, means that the tree becomes less comples, and only capture essential patterns. This can be better for generalization. The GA RF model had a higher max_depth, even if the model had less overfitting. It might be that even if the max_depth it set to something, it isn't nessecary that the trees reasches that maximum. It can also be that the higher n_estimators in RF GA compensates for the higher max_depth, in preventing overfitting.

min_samples_splitwas higher for the RF GA model. min_samples split is the minimum number of sampltes required to split a a node.

The criterion can be gini, entropy or log_loss. Criterion evaluates splits. Gini uses gini impurity. Entropy is based on information gain. log_loss is based on logarithmic loss. GA RF used log_loss, while BA RF used entropy.

Evolution of parameter values
In graphs XGBoost evolution of parameters, it can be seen how different parameter values in the GA evolve

over time. For the XGBoost model, I have showcased the graphs for the model with a small population, and more iterations, as they show the evolution more clearly, since it goes on for more iterations.

Graphs in Random Forest evolution of parameters also shows the evolution of parameters for GA RF parameters. This used a large population, with few iterations.

It can be seen that the population in general gradually gets better parameter values, and that the fitness climbs over iterations. The lower amount of iterations make this a bit less clear for the GA RF, but it can still be seen. For example more trees with n_estimators is generally considered better for both algorithms. It can be seen that both these values climb over the iterations.

Discussion

The GA produced good results. The parameter values are evolved over iterations to become better, and comparable to the parameter values produced by BA.

The default RF and BA RF parameters had some overfitting. The BA RF overfitting might have come from that it uses default parameters if all tried values gives less acuuracy. Since many of the parameters were the same as the default parameter values, it might be that the BA RF had the same traits that gave overfitting, as the default parameters.

One problem with the GA is that it there can at times be gaps in parameter values of the initial population. This can potentially be solved by smarter initilization, with making sure that the initial parameter values cover all possible values. It can potentially also be solved by a hiher mutation rate and more iterations, which can make the parameters evolve to values that aren't in the initial population. This can be seen in graphs XGBoost evolution of parameters for lambda and max_depth, where the found parameter values were not in the initial population.

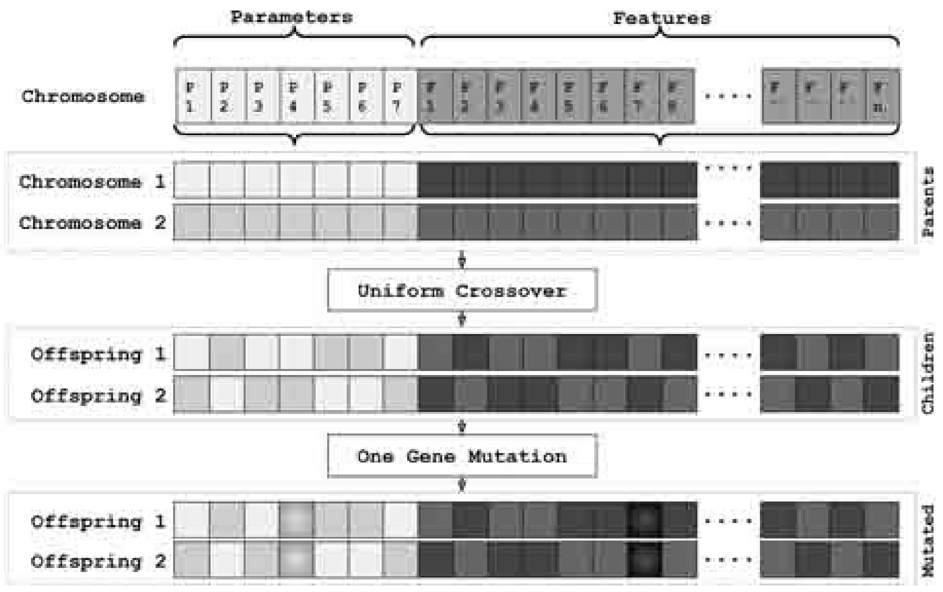
Conclusion

The GA achived good results. After the creation of the GA, I feel like I got a better understanding of how to make a GA, by having to implement all the components of a GA, and also by comparing it with BA and DEF. Further work that can be done on the GA could be to implement a parameter for the crossover rate, which currently can be varied. Smarter initilization can also be tried, to make sure the GA can search across the whole search space.

Appendix

GA Algorithm explained

(Ghatasheh et al., 2022, p. 8)



GA Structure

Representation	Real-valued representation
Recombination	Uniform crossover
Mutation	Gaussian mutation
Parent selection	Roulette wheel
Survivor selection	Elitism

Varying dataset training size RF

~Rows in dataset	Accuracy% on train	Accuracy% on test	TP%	FP%	TN%	FN%	Runtime
400.000	84	85	9.56	1.42	74.96	14.05	95 sec
350.000	84	85	9.53	1.40	74.99	14.09	79 sec
300.000	84	84	9.31	1.34	75.05	14.30	66 sec
250.000	84	84	9.15	1.31	75.97	14.46	53 sec
200.000	84	84	9.00	1.33	75.06	14.61	39 sec
150.000	84	84	8.79	1.32	75.07	14.82	28 sec

XGB accuracy

XGB GA varying population size and iteration size

Large population, few iterations	Small population, many iterations
0.8924098657471057	0.8951069440375037
0.8927554523804158	0.8857836176910305
0.8907570601095359	0.8941077479020637
0.8889990759313935	0.8962789334895986
0.8913881313529717	0.8918764602913446
0.8950393292614213	0.8892695350357231
0.892958296708663	0.8939274418325107
0.8963240100069869	0.8933940363767495
0.8923197127123291	0.8926051973224549
0.8911402105073362	0.8922370724304507

T-statistic	-0.04255633167259053
P-value	0.9665237253929065
Conclusion	Fail to reject the null hypothesis: there is no significant difference between the two models

Best: 0.8963240100069869	Best: 0.8962789334895986
Mean: 0.8924091144718159	Mean: 0.8924586986409431

XGB BA accuracy
Accuracy: 0.8943706942534954

XGB DEF accuracy
Accuracy: 0.8851300081889006

Random Forest accuracies

RF GA

Large population, few iterations
0.8176880254231558
0.8179058952571991
0.8182514818905091
0.8178533059869127
0.817733101940544
0.817718076434748
0.8174551300833164
0.8173649770485399
0.8177706657050343
0.8177781784579323

Best: 0.8182514818905091
Mean: 0.8177518838227892

RF BA:
Accuracy: 0.8394722665224218

RF DEF:
Accuracy: 0.839369830286912

Varying dataset size fo train accuracies on GA, BA and DEF

XGBoost

variant	Test score	Train score	rows
GA	0.8950393292614213	0.90692	150000
GA	0.8950393292614213	0.90692	200000
GA	0.8950393292614213	0.90692	250000
GA	0.8950393292614213	0.90692	300000
GA	0.8950393292614213	0.90692	350000
GA	0.8988558077336278	0.9056573533573241	399321
BA	0.896015987138167	0.91907	150000
BA	0.896015987138167	0.91907	200000
BA	0.896015987138167	0.91907	250000
BA	0.896015987138167	0.91907	300000
BA	0.896015987138167	0.91907	350000
BA	0.9019210109160299	0.9148955351709527	399321
def	0.886549918486631	0.898085	150000
def	0.886549918486631	0.898085	200000
def	0.886549918486631	0.898085	250000
def	0.886549918486631	0.898085	300000
def	0.886549918486631	0.898085	350000
def	0.8900057848197315	0.8957254940261093	399321

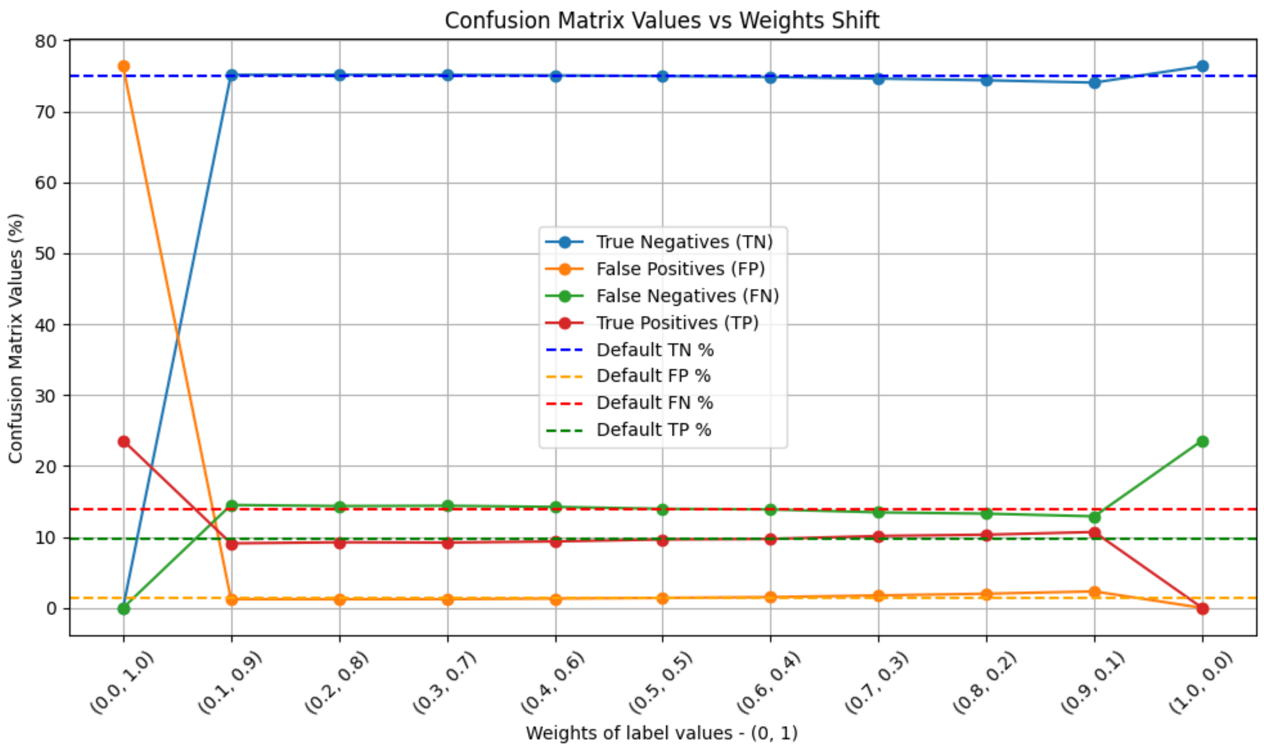
Random Forest

variant	Test score	Train score	rows
GA	0.8177932039637285	0.8171	150000
GA	0.8177781784579323	0.81712	200000
GA	0.8177481274463402	0.81708	250000
GA	0.8178533059869127	0.817145	300000
GA	0.8177030509289519	0.817015	350000
GA	0.8177706657050343	0.818073680071922	399321
BA	0.8411728909824427	0.999925	150000
BA	0.841390760816486	0.999915	200000
BA	0.8412630440172192	0.999895	250000
BA	0.8413006077817095	0.99989	300000
BA	0.8419767555425335	0.999915	350000
BA	0.8469126341965486	0.9999148554671555	399321

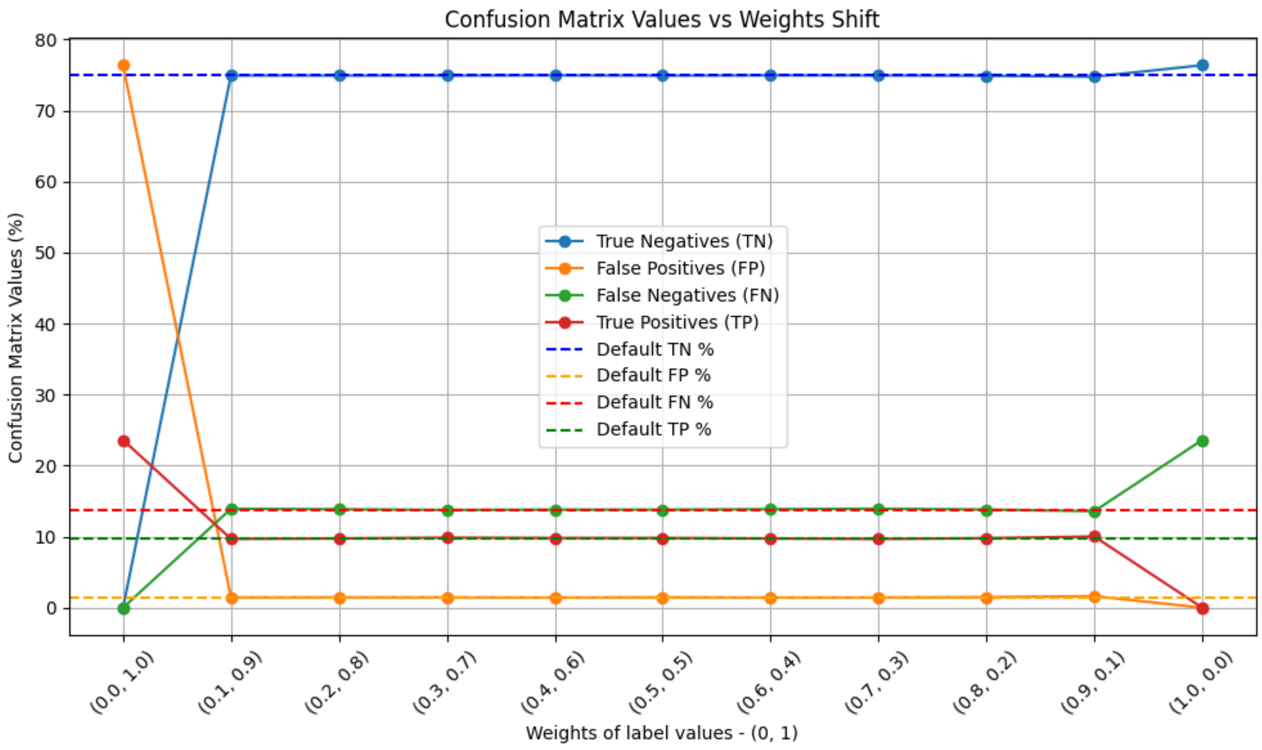
def	0.8415184776157527	0.999995	150000
def	0.8400459780477361	0.99999	200000
def	0.8402337968701872	1.0	250000
def	0.8409925849128896	0.99998	300000
def	0.8417588857084901	0.999985	350000
def	0.8456054151922889	0.999989982996136	399321

TN, FP, FN, TP

DEF RF

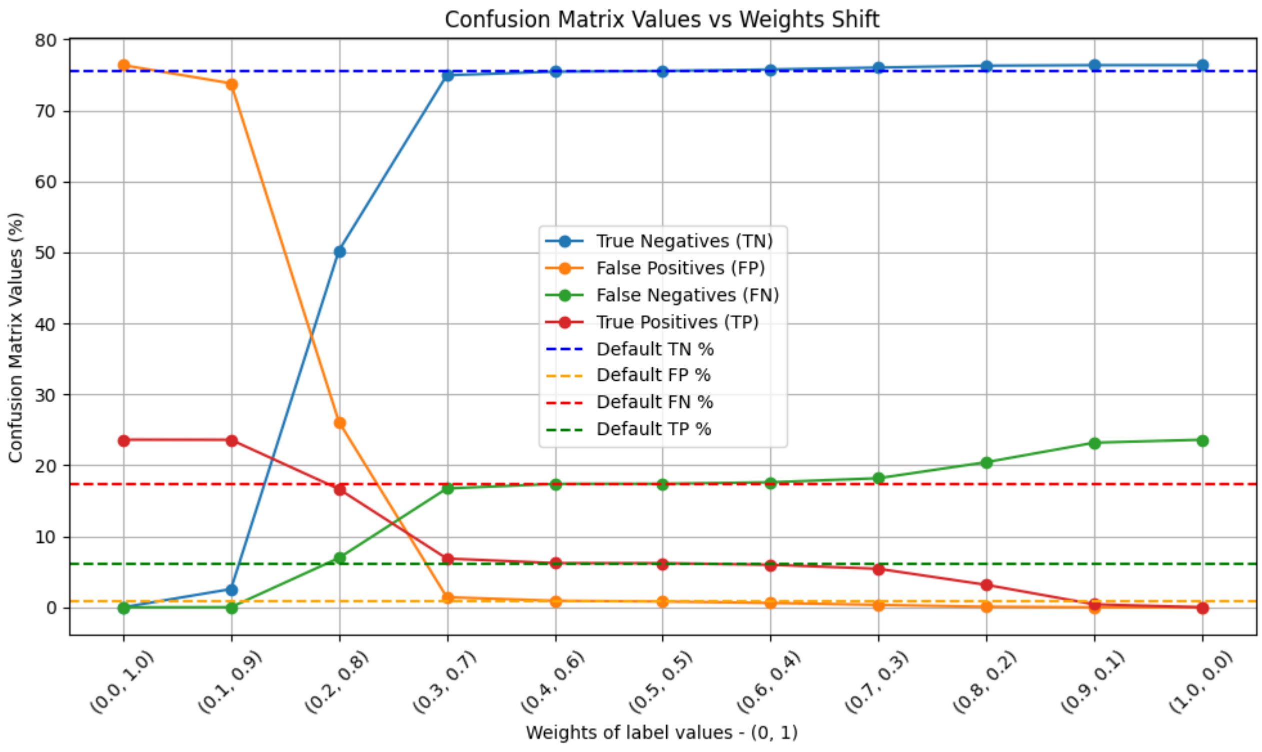


BA RF

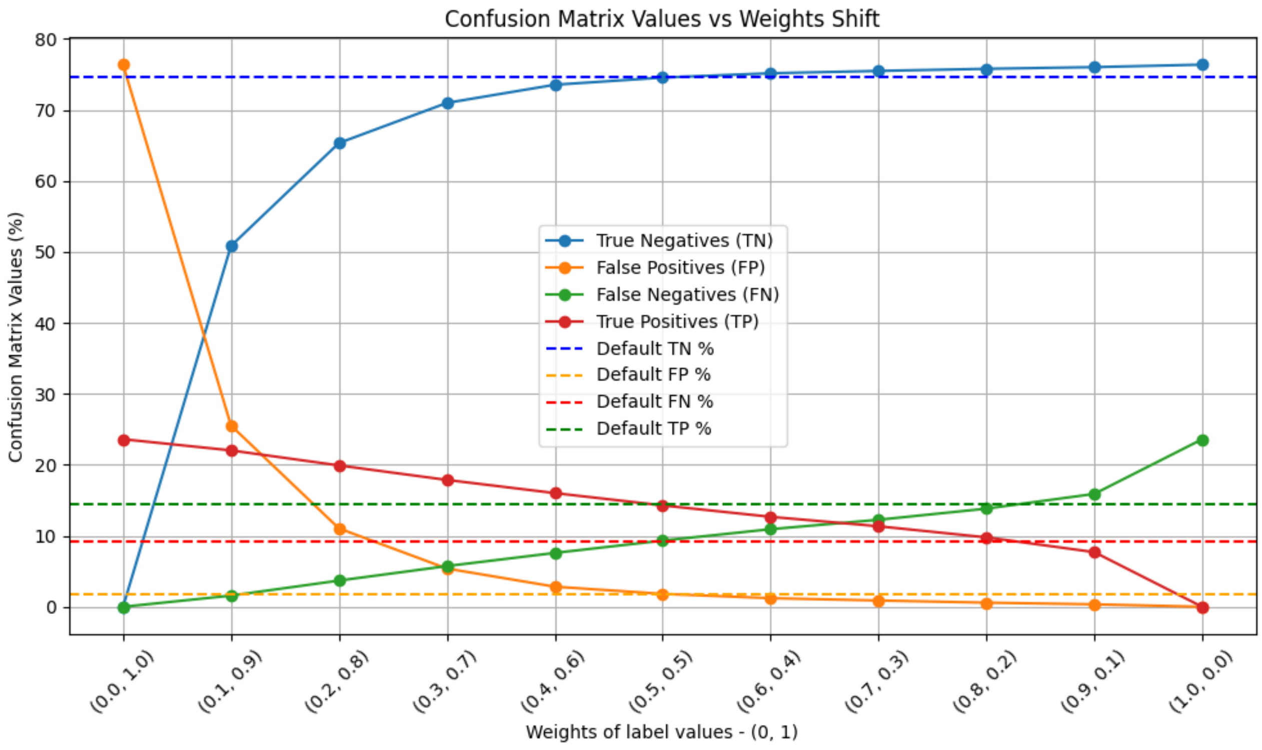

Confusion Matrix Values vs Weights Shift
Confusion Matrix Values (%)
Weights of label values - (0, 1)
(0.0, 1.0)
(0.1, 0.9)
(0.2, 0.8)
(0.3, 0.7)
(0.4, 0.6)
(0.5, 0.5)
(0.6, 0.4)
(0.7, 0.3)
(0.8, 0.2)
(0.9, 0.1)
(1.0, 0.0)
True Negatives (TN)
False Positives (FP)
False Negatives (FN)
True Positives (TP)
Default TN %
Default FP %
Default FN %
Default TP %

,

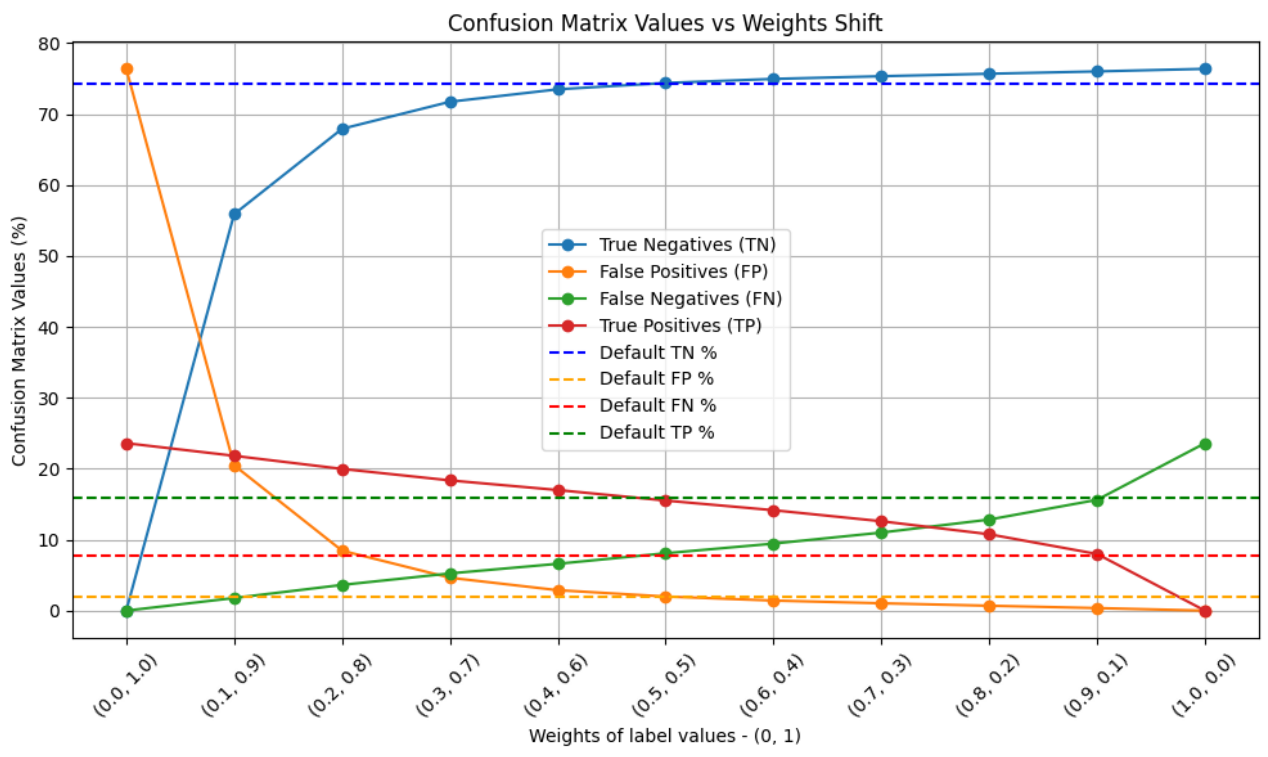
GA RF


Confusion Matrix Values vs Weights Shift
Confusion Matrix Values (%)
Weights of label values - (0, 1)
True Negatives (TN)
False Positives (FP)
False Negatives (FN)
True Positives (TP)
Default TN %
Default FP %
Default FN %
Default TP %
(0.0, 1.0)
(0.1, 0.9)
(0.2, 0.8)
(0.3, 0.7)
(0.4, 0.6)
(0.5, 0.5)
(0.6, 0.4)
(0.7, 0.3)
(0.8, 0.2)
(0.9, 0.1)
(1.0, 0.0)

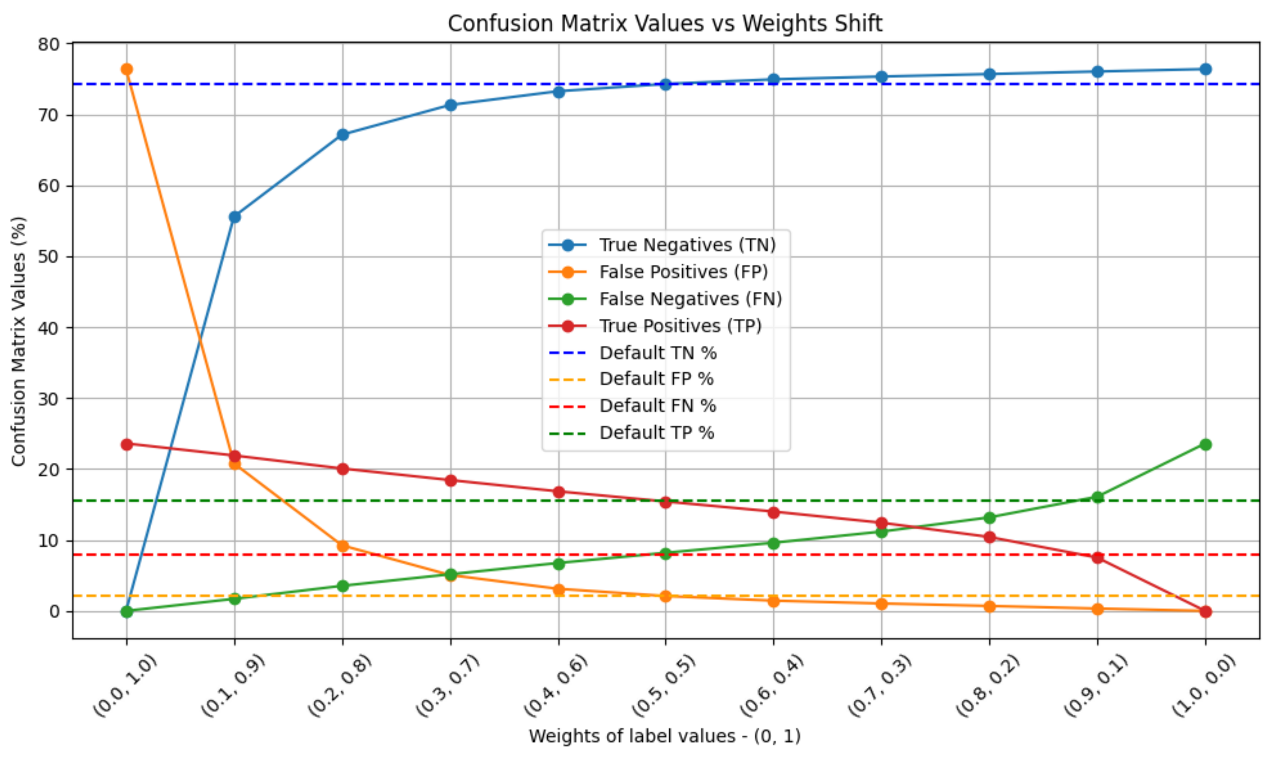
XGB DEF


Confusion Matrix Values vs Weights Shift
Confusion Matrix Values (%)
Weights of label values - (0, 1)
True Negatives (TN)
False Positives (FP)
False Negatives (FN)
True Positives (TP)
Default TN %
Default FP %
Default FN %
Default TP %
(0.0, 1.0)
(0.1, 0.9)
(0.2, 0.8)
(0.3, 0.7)
(0.4, 0.6)
(0.5, 0.5)
(0.6, 0.4)
(0.7, 0.3)
(0.8, 0.2)
(0.9, 0.1)
(1.0, 0.0)

XGB BA


Confusion Matrix Values vs Weights Shift
Confusion Matrix Values (%)
Weights of label values - (0, 1)
True Negatives (TN)
False Positives (FP)
False Negatives (FN)
True Positives (TP)
Default TN %
Default FP %
Default FN %
Default TP %
(0.0, 1.0)
(0.1, 0.9)
(0.2, 0.8)
(0.3, 0.7)
(0.4, 0.6)
(0.5, 0.5)
(0.6, 0.4)
(0.7, 0.3)
(0.8, 0.2)
(0.9, 0.1)
(1.0, 0.0)
0
10
20
30
40
50
60
70
80

XGB GA



Produced parameter values with GA and BA

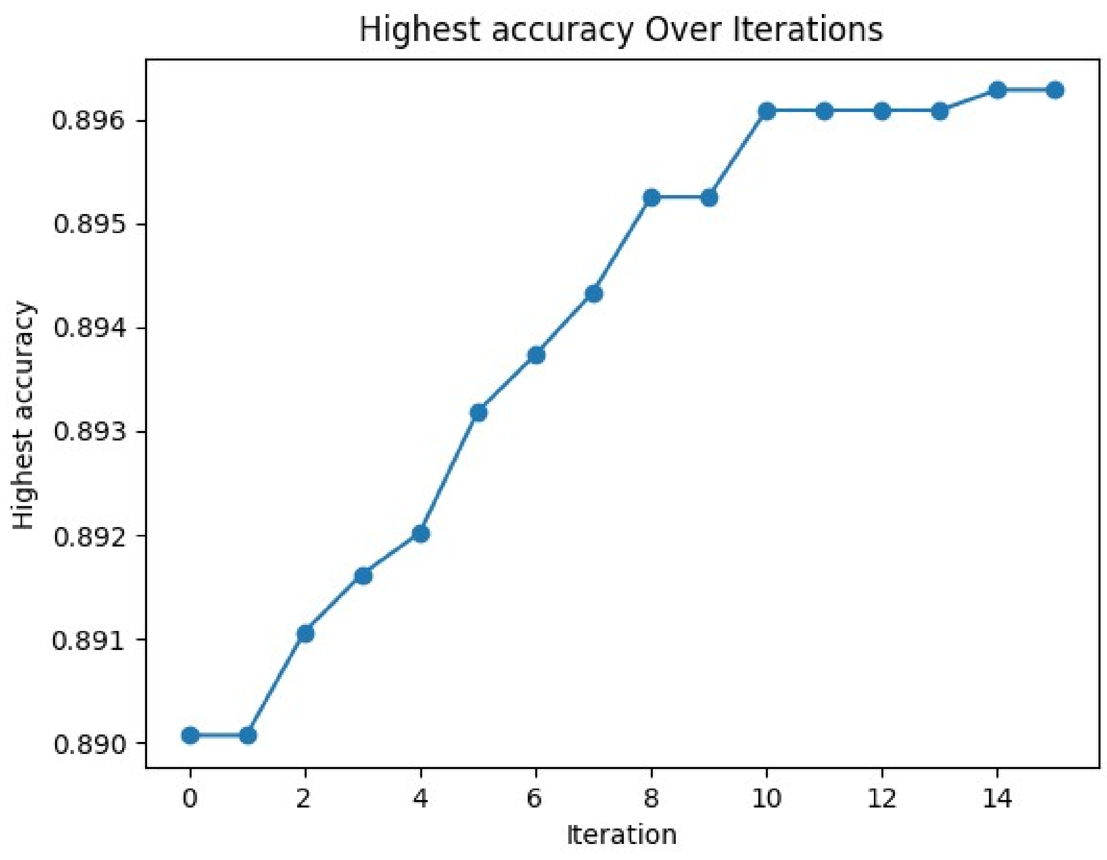
XGBoost

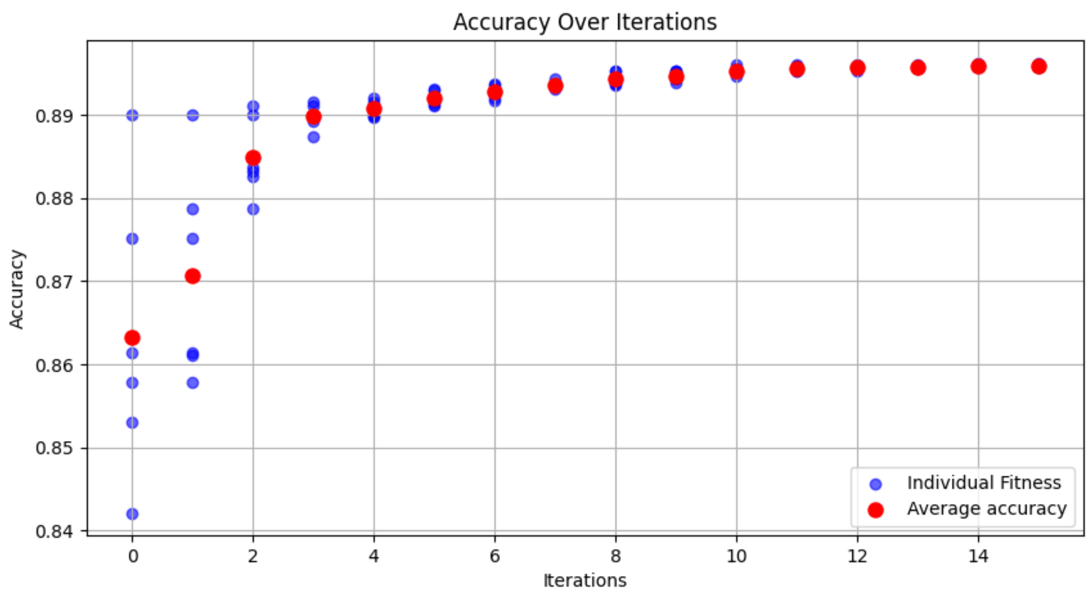
Parameter	GA Value	SEQ Value
eta	0.650526310496016	0.3
gamma	0.700417268555284	1.0
n_estimators	1056	250
max_depth	3	
min_child_weight	1	
max_delta_step	4	
subsample	0.836867407543178	
sampling_method	uniform	
colsample_bytree	0.9698688251745518	
colsample_bylevel	0.8438854659636649	
colsample_bynode	0.500193527056886	
lambda	12	
alpha	7	
tree_method	hist	
refresh_leaf	1	
max_leaves	382	

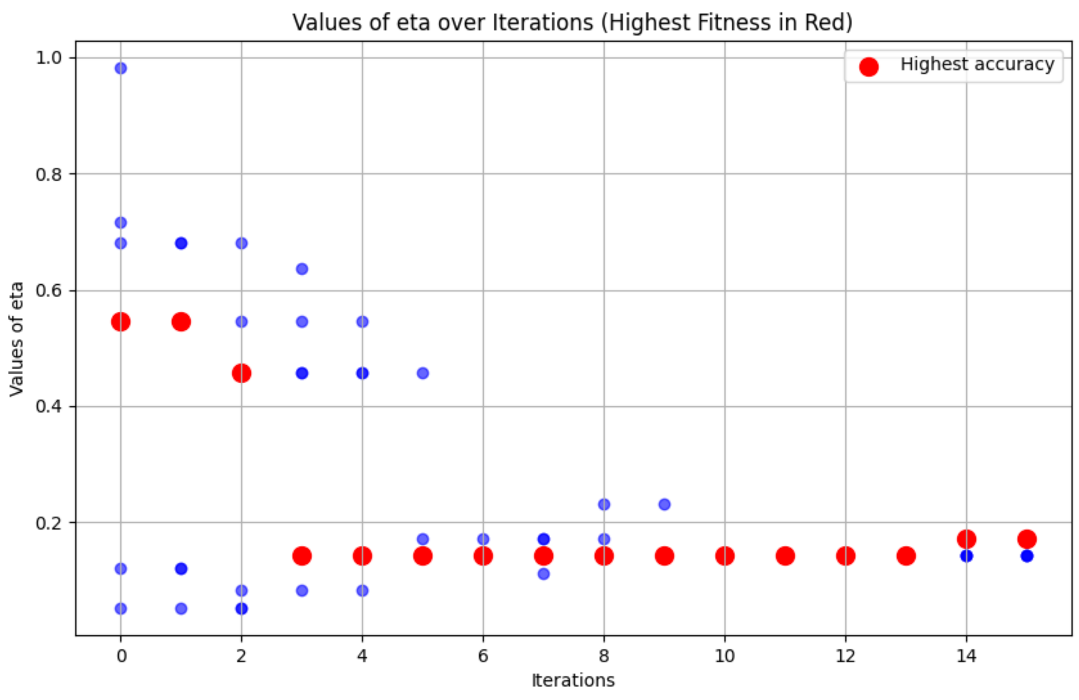
Random Forest

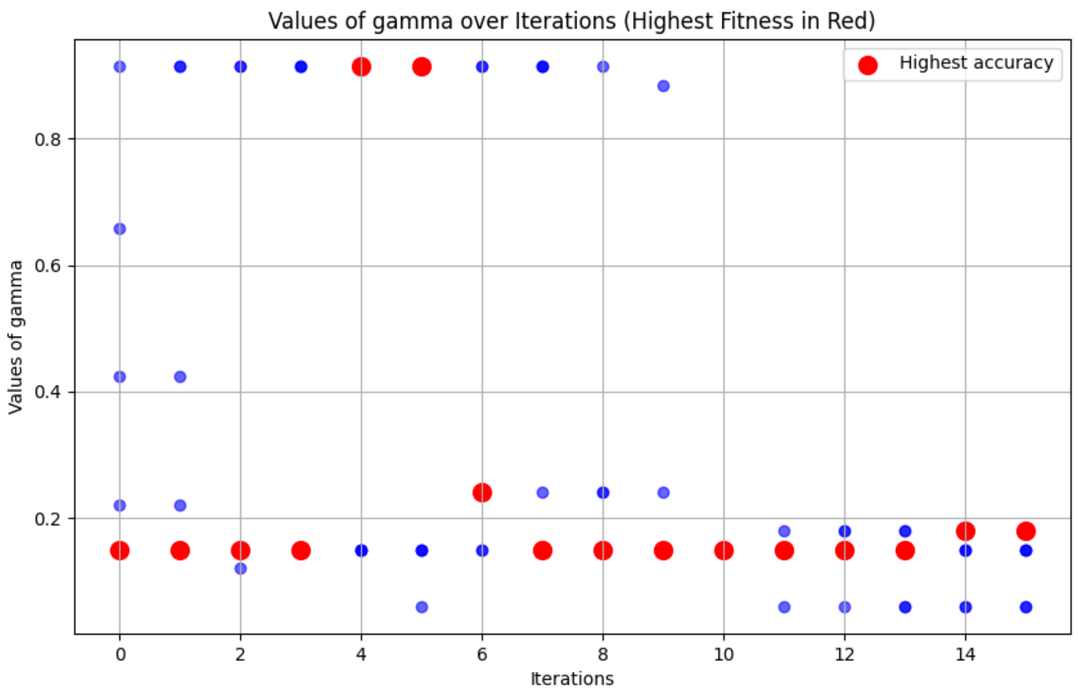
Parameter	GA Value	SEQ Value
n_estimators	1923	100
criterion	log_loss	entropy
max_depth	150	50
min_samples_split	10	3
min_samples_leaf	5	
max_leaf_nodes	6	
max_samples	982	

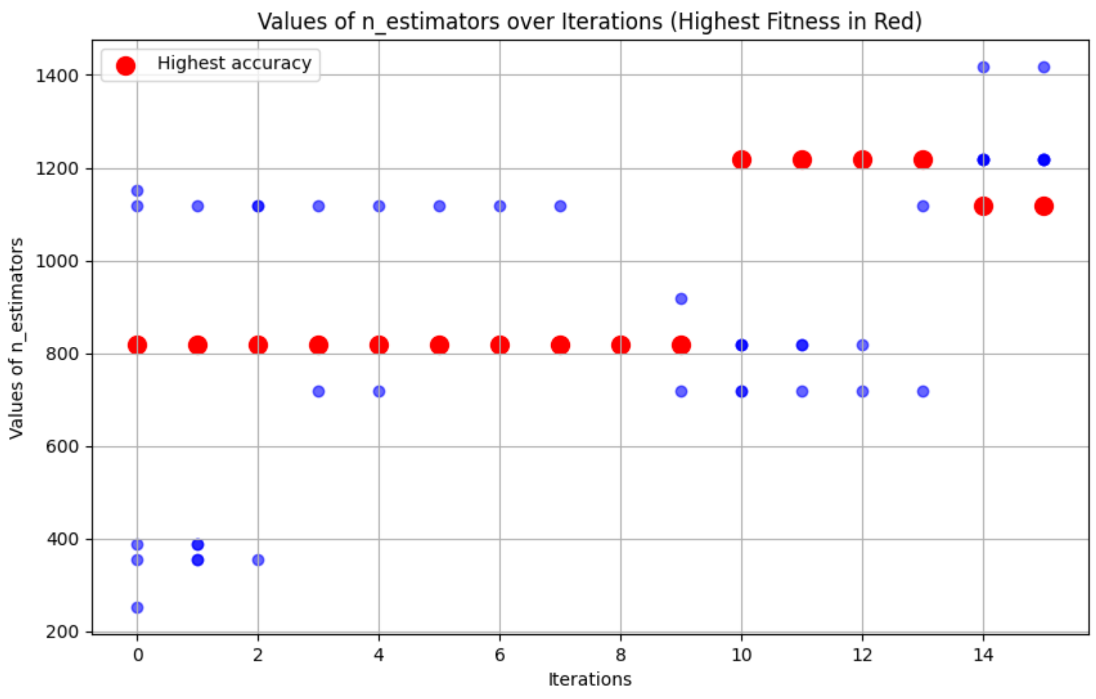
XGBoost evolution of parameters – Many iterations, low population.

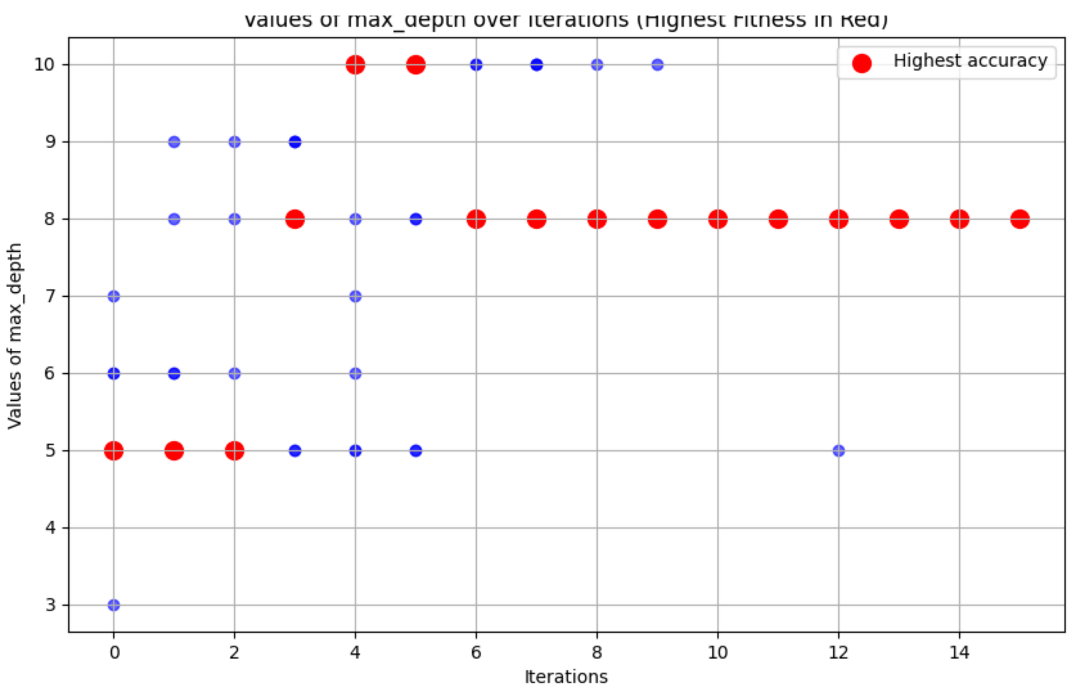


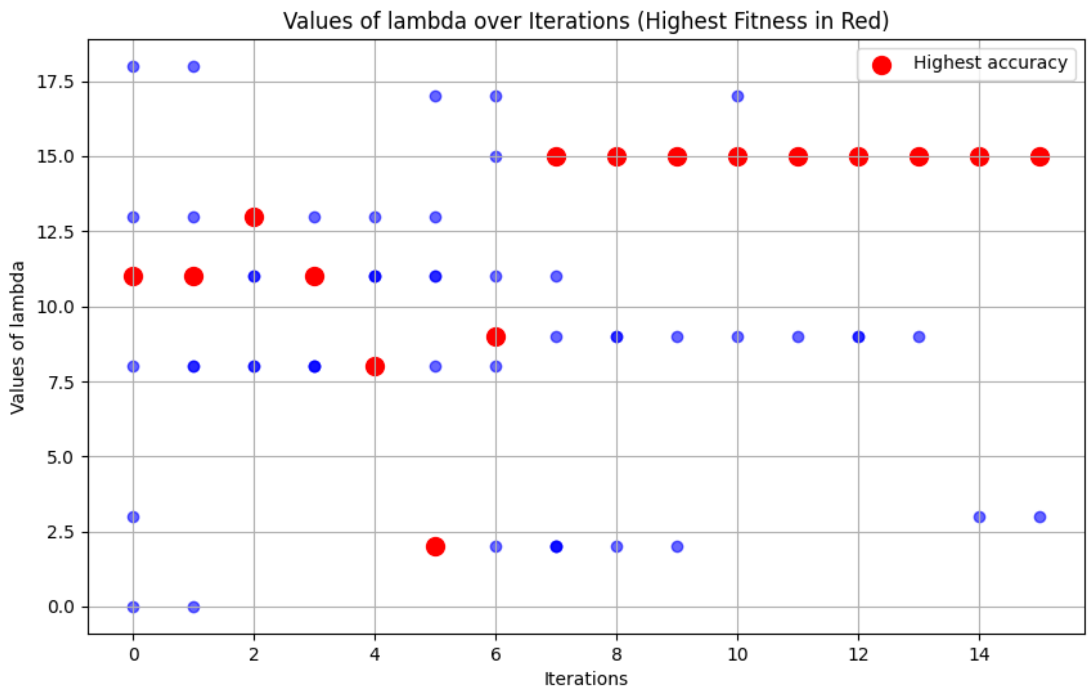



Values of eta over Iterations (Highest Fitness in Red)
Highest accuracy
Values of eta
Iterations
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14


Values of gamma over Iterations (Highest Fitness in Red)
Highest accuracy
Values of gamma
Iterations
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14

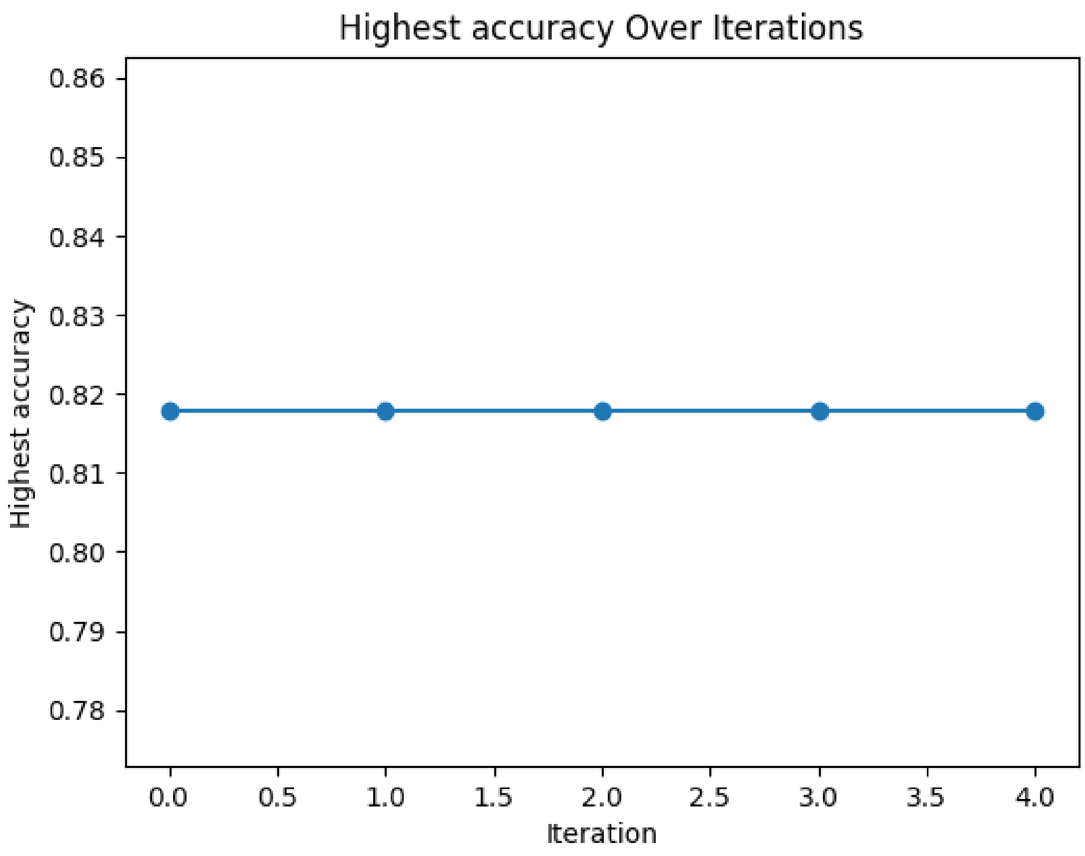

Values of n_estimators over Iterations (Highest Fitness in Red)
Highest accuracy
Values of n_estimators
Iterations
1400
1200
1000
800
600
400
200
0
2
4
6
8
10
12
14

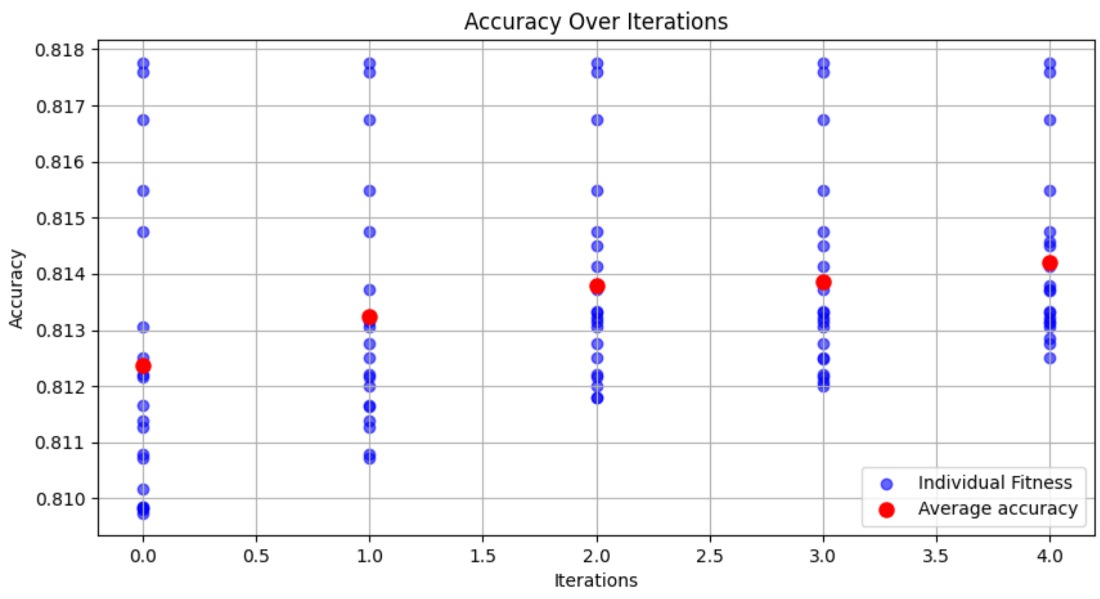

Values of max_depth over Iterations (Highest Fitness in Red)
Highest accuracy
Values of max_depth
Iterations
10
9
8
7
6
5
4
3
0
2
4
6
8
10
12
14

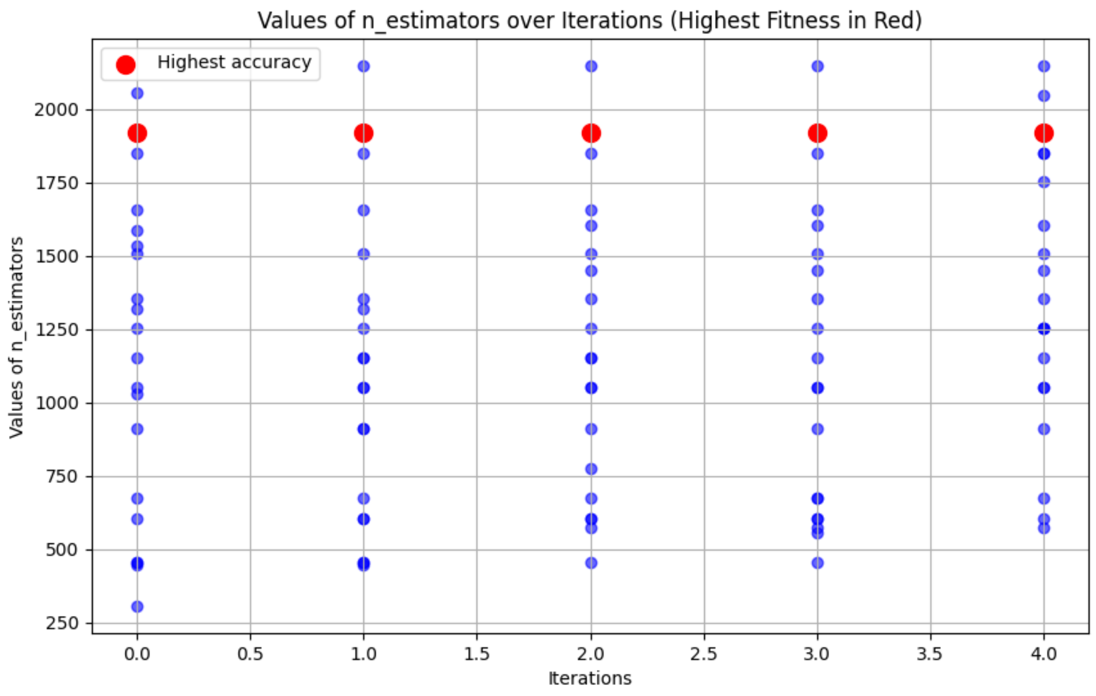


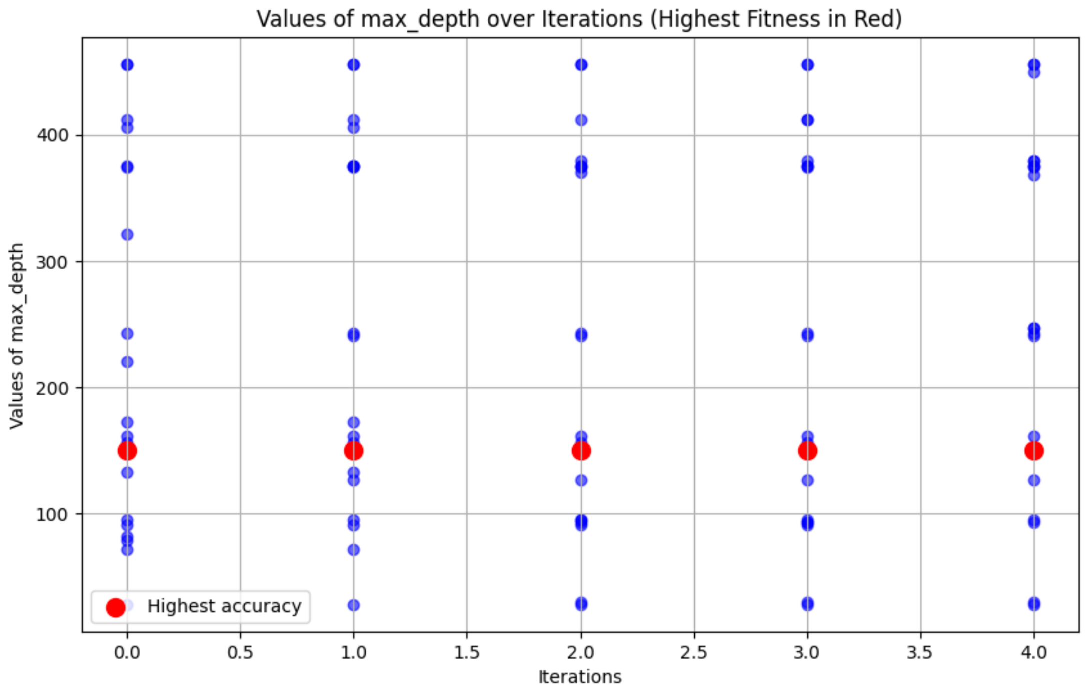
Values of lambda over Iterations (Highest Fitness in Red)
● Highest accuracy
Values of lambda
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8
10
12
14
Iterations

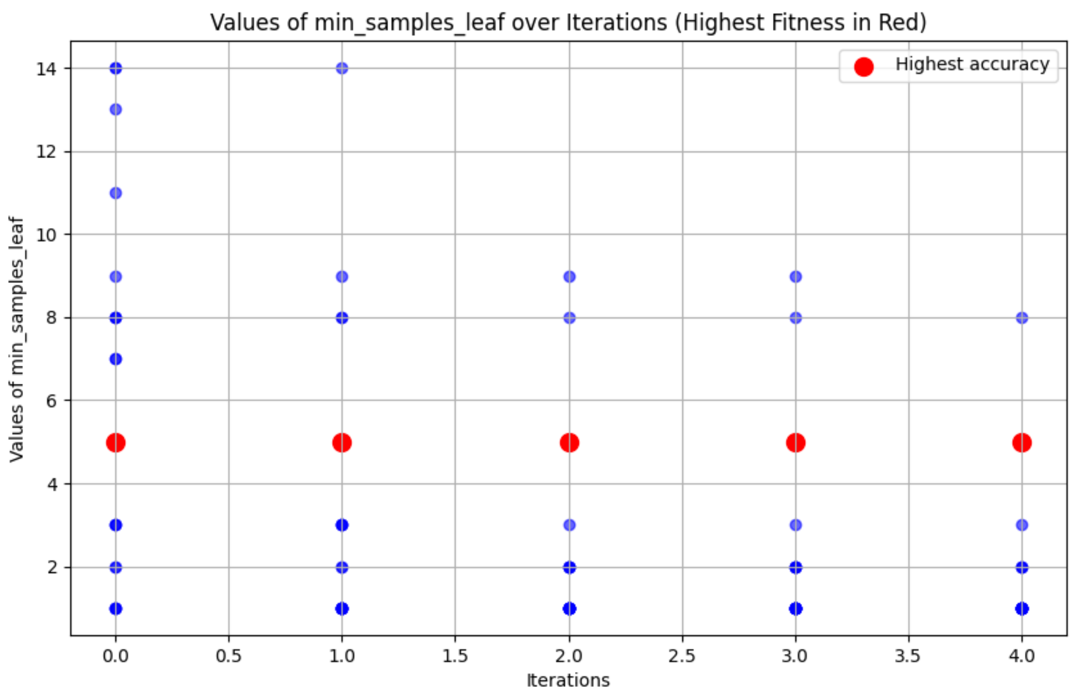
Random Forest evolution of parameters






Values of n_estimators over Iterations (Highest Fitness in Red)
● Highest accuracy
Values of n_estimators
2000
1750
1500
1250
1000
750
500
250
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Iterations


Values of max_depth over Iterations (Highest Fitness in Red)
● Highest accuracy
Values of max_depth
400
300
200
100
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Iterations


Values of min_samples_leaf over Iterations (Highest Fitness in Red)
Highest accuracy
Values of min_samples_leaf
Iterations
14
12
10
8
6
4
2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Parameters tried

Parameters XGB GA

Parameter	Min Value	Max Value	Step
eta	0.01	1	0.03
gamma	0.01	1.0	0.03
n_estimators	10	1500	100
max_depth	1	10	1
min_child_weight	1	5	1
max_delta_step	1	5	1
subsample	0.01	1.0	0.03
sampling_method	uniform		
colsample_bytree	0.01	1.0	0.03
colsample_bylevel	0.01	1.0	0.03
colsample_bynode	0.01	1.0	0.03
lambda	0	20	2
alpha	0	20	2
tree_method	Hist, exact, approx		
refresh_leaf	0	1	1
max_leaves	0	500	50
max_bin	100	500	50

Parameters RF GA

Parameter	Min Value	Max Value	Step
n_estimators	100	3000	100
criterion	Gini, entropy, log_loss		
max_depth	1	500	2
min_samples_split	2	15	1
min_samples_leaf	1	14	2
max_leaf_nodes	2	30	2
max_samples	150	1000	150

Parameters XGB BA

Parameter	Values
eta	0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
gamma	0.0, 0.01, 1.0, 5.0, 10.0, 20.0
n_estimators	100, 250, 500, 1000, 1500
max_depth	6, 7, 8, 9, 10
min_child_weight	1, 2, 3, 4, 5
max_delta_step	0, 1, 2, 3, 4, 5
subsample	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
sampling_method	uniform
colsample_bytree	1.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
colsample_bylevel	1.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
colsample_bynode	1.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
lambda	1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
alpha	0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20
tree_method	Hist, exact, approx
refresh_leaf	1, 0
max_leaves	0, 50, 100, 150, 200, 250, 300, 400, 500
max_bin	256, 100, 150, 200, 250, 300, 400, 500

Parameters RF BA

Parameter	Values
n_estimators	100, 300, 350, 750, 850, 900, 1100, 1200, 1700, 1900, 1900, 2400, 2700, 3000
criterion	gini, entropy, log_loss
max_depth	None, 25, 50, 75, 100, 125, 150, 175, 200, 250, 300, 350, 400, 450, 500
min_samples_split	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15
min_samples_leaf	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
min_weight_fraction_leaf	0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5
max_leaf_nodes	None, 2, 3, 5, 6, 8, 9, 12, 15, 18, 21, 24, 27, 30
max_samples	None, 67, 150, 245, 312, 475, 550, 620, 738, 812, 875, 920, 990, 1000

Cleaning dataset

The **target feature** is passed directly into the models.
Some features have too various categories to give any predictive value. The values in **member_id** and **zipcode** are all unique. These features are removed,.

Some other features has so **many null** values that they provide little predictive value. These are removed if they have more than 50% missing values.

The **ordinal numerical features** can be filled with 0 where they have null values, and then normelized to minimize outliers.

There is a feature called **desc**, which has a textual description about the loan. There isn't any good way to analyze these texts without neural networks, so the feature was removed.

There are some **categorical features that represents an ordinal variable.** Features like these were be encoded into ordinal numerical values.

Then there are some **categorical features that represents a nominal variable**.
These features were be encoded using one-hot encoding.

Features with high correlation can be merged together using PCA. I merged features with >= 90% correlation. Random Forest has been used to see how PCA and removal of highly correlated features affects performance. The 2nd configuration gave the best performance, and so is used further.

Explenation	Accuracy on train	Accuracy on test	TP%	FP%	TN%	FN%	Runtime
Including all features	0.84	0.83	7.36	0.60	75.79	16.25	95.66 sec
Removing highly correlated features	0.84	0.83	7.35	0.64	75.74	16.26	94.07 sec
Removing highly correlated features, using PCA on more than 90% for correlation.	0.80	0.80	7.21	3.73	72.66	16.40	61.01 sec

Refrences

Lipadhyay, P. (2021). *Loan defaulter classification* [Dataset]. Kaggle. https://www.kaggle.com/datasets/prateek146/loan-defaulter-classification

Ghatasheh, N., Altaharwa, I., & Aldebei, K. (2022). *Modified genetic algorithm for feature selection and hyperparameter optimization: Case of XGBoost in spam prediction*. *IEEE Access, 10*, 9851666. Institute of Electrical and Electronics Engineers. https://ieeexplore.ieee.org/abstract/document/9851666