Beyond Defaults: Hyperparameters and AI Model Accuracy

Adib Md Alim Chowdhury	Caden Quiring	Matthew Muscedere	Tayo Alalade
chowdh52@uwindsor.ca	quiringc@uwindsor.ca	muscede2@uwindsor.ca	alaladej@uwindsor.ca
110036540	110045557	110063159	110038930

Abstract - The Spaceship Titanic competition on Kaggle comprises close to 13,000 instances, each characterized by 13 features. Within the training set, every instance includes a 'Transported' value, indicating whether the passenger was transported to an alternate dimension. Our primary objective in conducting experiments is to assess the impact of hyperparameter tuning on model performance. For this purpose, we employed both default and hyperparameter-tuned models. The metric used to evaluate model performance in this context is accuracy. However, it's essential to note that the evaluation of model performance solely based on accuracy might not capture the entire picture. Depending on the nature of the problem, considering additional metrics like precision, recall, or F1-score could provide a more comprehensive understanding of the model's effectiveness.

I. INTRODUCTION AND PRESENTATION OF RESEARCH QUESTION

The Spaceship Titanic Kaggle competition stands as a popular classification Machine Learning challenge, engaging 2,814 competitors, 2,675 teams, and accumulating a total of 15,813 entries. The dataset revolves around passengers aboard an interstellar passenger line who were transported to an alternate dimension following the *Spaceship Titanic's* encounter with a spacetime anomaly. The competition's goal is to leverage Machine Learning techniques to predict whether each passenger was transported to this alternate dimension based on provided input features. The dataset is presented in tabular form, and notably, the 'Transported' column for the testing dataset remains undisclosed.

In order to streamline model training and evaluation, we restructured the dataset, expanding it to 16 features. This restructuring encompassed the conversion of the 'name' column into 'FirstName' and 'LastName' columns, and the transformation of cabin identification into 'deck', 'num', and 'side'. This adaptation enabled us to conduct model training and testing solely with numerical values.

While this dataset might not have immediate practical applications, as it is based on a fictional story, it still provides many researchers with a dataset that can be used to test their machine learning models. Their research can then provide useful insights on the effectiveness of the models on a variety of datasets which may have real life applications. For us this dataset has provided a valuable opportunity to assess the impact of hyperparameter tuning on model performance and compare performance disparities among various Machine Learning models. Initially, we employed four distinct Machine Learning models using default Scikit-Learn hyperparameters, training and testing them on our dataset. The metrics derived from this phase served as baseline measurements for model performance. Subsequently, we conducted hyperparameter tuning using diverse methods on these models, retraining and retesting them on the dataset. The resulting accuracy served as a comparative metric against our default baseline measurements.

This strategic approach allowed us to gauge the influence of hyperparameter tuning on model

performance and its differential effects across various model types. For instance, it enabled a comparison between the impact of hyperparameter tuning on a deep neural network model versus a random forest classifier model. This exploration raised intriguing questions and led to significant conclusions regarding the varying impacts of hyperparameter tuning on deeper to shallower models.

Leveraging insights drawn from this study will enable evidence-based decision-making concerning hyperparameter tuning and model selection. These insights hold relevance in real-world scenarios where decisions regarding the utilization and extent of hyperparameter tuning, as well as the choice of an appropriate model for a specific problem, need to be made.

Classification, a supervised learning technique, aims to categorize categorical data. The primary objective of a classification model is to establish a function that maps input features to two or more predefined categories. In this scenario, we employ binary classification, where the output categories are 'transported' or 'not transported.' While the features within this context can be both numerical and categorical, during data cleaning, they are converted into numeric values to facilitate the training and testing of models. The Spaceship Titanic dataset comprises a mix of numeric and categorical features. For instance, the 'Destination' feature is represented as a string of characters denoting the passengers' destinations, while 'RoomService' is a numeric value signifying the amount spent by a passenger on room service during the voyage.

To initiate the training of our classification models, we utilized the labeled training dataset, covering approximately two-thirds of the total dataset. This deliberate allocation aimed to furnish an adequate volume of training examples, crucial for optimizing the accuracy of our models. During this phase, these models underwent training using the labeled dataset, learning to establish correlations between input features and one of the categorical outputs. Once the training phase was completed, we employed the provided testing dataset, excluding the 'transported' column (the output we aimed to predict). Leveraging the functions derived from the training phase, the models endeavored to predict the output value based on the input features available in the testing dataset.

These models demonstrate proficiency in identifying both simple and intricate relationships existing within the dataset's input features. However, it's imperative to approach hyperparameter tuning cautiously. While this tuning aids in detecting complex relationships, an excessive adjustment closely aligned with the training data might lead to a phenomenon known as overfitting. Overfitting occurs when the model is excessively tailored to the intricacies of the training data, hindering its ability to accurately predict unseen data. Conversely, there's the risk of underfitting, where the model hasn't been sufficiently trained to grasp the nuances of the provided data, leading to suboptimal predictive performance. Striking the right balance between model complexity and generalization is pivotal for ensuring the model's ability to make accurate predictions on new, unseen data.

There are a plethora of classification Machine Learning models to use. To choose models for this problem, we used a Machine learning library called Pycaret. We utilized Pycaret's ability to test a plethora of base models for us and chose some of the best performing among them. We settled on using a neural network, random forest classifier, decision tree classifier, and Logistic Regression. Using these models on this classification problem, we aimed to uncover the true impact of hyperparameter tuning.

II. RELEVANT LITERATURE OVERVIEW

Competition platforms like Kaggle.com provide many researchers with datasets that they can use to train and test their machine learning models, which means that there are many articles and reports on the internet posted by these researchers. One of these research articles is "A Comparative Analysis of Machine Learning Algorithms to Predict Alzheimer's Disease". While their dataset is on a different topic, they have researched into a similar concept where they compare machine learning algorithms. This article focused on the following models: Support vector machine (SVM), Logistic Regression (LR), Decision tree algorithm (DT) and Random Forest (RF). They used a dataset from the Open Access Series of Imaging Studies (OASIS), which they mention to be small but

contain significant values. Like our algorithms they have also investigated the difference between the default and tuned models. Through their research it was found that the SVM model provided the best results with the most accuracy in detecting Dementia.

Machine learning techniques have become essential tools in predictive modeling, offering a range of algorithms each suited to different types of data and analysis. A notable study in this field is the research paper that performs an exploratory data analysis and applies machine learning techniques to the Titanic disaster dataset. This comprehensive study delves into algorithms like Logistic Regression, K-nearest neighbors, Support Vector Machines, and Decision Trees, comparing their accuracies based on the specific features provided in the dataset. The Titanic dataset, detailing the survival of passengers on the ill-fated voyage, presents a unique challenge for predictive modeling. This research paper stands out for its thorough exploration of various machine learning techniques, providing a detailed comparison of their effectiveness. The findings of this paper are significant as they reveal how different algorithms perform under the constraints of a limited yet complex dataset, offering insights into the nuances of machine learning applications in real-world scenarios. Such comparative analyses are invaluable for researchers and practitioners in the field, as they highlight the strengths and weaknesses of various algorithms, guiding the selection of the most appropriate technique for specific tasks (Semantic Scholar: Exploratory Data Analysis and Machine Learning on Titanic Disaster Dataset)

III. TOOLS USED

For this research, we implemented a variety of Machine Learning algorithms, including a neural network, random forest classifier, decision tree classifier, and Logistic Regression. Our implementations were carried out using Python, chosen for its extensive array of high-level machine learning libraries. Scikit-Learn served as our primary library for this task. Its high-level API supports a wide range of Machine Learning algorithms and offers robust functionality for hyperparameter tuning via grid search and random search. Leveraging these tuning capabilities was crucial in evaluating their impact. Scikit-Learn's compatibility with supporting libraries such as Pandas and Matplotlib proved immensely beneficial.

Pandas proved invaluable for importing data from the provided CSV file, performing data cleaning, and formatting, streamlining the data preparation process. Matplotlib was instrumental in visualizing and comparing the performance of our algorithms.

While Scikit-Learn sufficed for the base neural network model, we turned to Tensorflow, Keras, and Keras-tuner for optimizing the neural network. The performance of neural networks is highly sensitive to their architecture; a simplistic architecture might lead to underfitting, while an overly complex one might cause overfitting. Leveraging the high-level Tensorflow API, Keras, alongside Keras-tuner, enabled us to merge a robust feature set with the ability to conduct Bayesian optimization. This method allowed us to explore hyperparameter combinations efficiently without relying on exhaustive techniques like grid search, which can be resource-intensive.

In fine-tuning our Scikit-Learn models, we adopted a grid search approach. This method facilitated an exploration of various hyperparameter combinations, particularly effective for the more straightforward Machine Learning methods. It empowered us to define and test specific hyperparameter combinations, identifying the best-suited combination from our predefined selection.

Using the Pycaret library was integral in our initial model selection process. It assisted in identifying the models to be tested for this research by training and evaluating a set of models on the dataset. Pycaret's functionality allowed us to determine the best-performing models among the batch, providing valuable insight into which models are most suitable for addressing this specific problem.

IV. CHOSEN METHODS TO TACKLE RESEARCH QUESTION

IV.A. RANDOM FOREST CLASSIFIER (RFC)

In the evaluation of various algorithms through the utilization of Pycaret, the Random Forest Classifier

emerged as one of the most proficient algorithms without undergoing any hyperparameter tuning initially.

The selected hyperparameters for tuning encompassed n_estimators, criterion, max_features, bootstrap, warm_start, and class_weight. The scikit-learn library offered robust hyperparameter optimization techniques, with the ultimate choice being Grid Search with Cross Validation for refining the Random Forest Classifier. During the execution of Grid Search, the scoring metric employed was "accuracy," reflecting the correct or incorrect classification, with a Cross Validation fold count of K = 5.

IV.B. NEURAL NETWORK ALGORITHM (NN)

To identify the most optimal neural network for this dataset, a Bayesian optimization method was employed running 1000 trials. The parameters tested during this optimization included:

- Num_hidden_layers: Ranging from 1 to 10
- Layer_num_units: Varied from 2 to 512 with a step of 2
- Dropout_rate: Ranged from 0 to 0.5 with a step of 0.01

Certain parameters remained constant throughout the optimization:

- Activation: relu
- Optimizer: Adam

The focus of tuning primarily honed in on the key architectural parameters, which significantly influence model performance and complexity. The decision to keep the activation function and optimizer parameters constant stemmed from the belief that attempting to optimize them might not yield substantial benefits and could substantially increase computational costs and time.

Num_hidden_layers plays a pivotal role in defining the neural network's architecture by determining the number of hidden layers of neurons. This parameter strongly influences model complexity, potentially differentiating between an underfit, well-fit, or overfit model. The selected range aimed to accommodate more complex models towards the upper limit while managing computational costs associated with complex networks. Similarly, it allowed for simpler networks with a minimum of one hidden layer. While extending this range might have been beneficial, it would have significantly escalated computation costs and time due to the increased complexity of neural networks.

The number of neurons within each layer is equally vital to model architecture and complexity, hence the use of a broad range to ascertain the optimal point for the model.

Lastly, the dropout rate is crucial for mitigating overfitting. A wide range for dropout rate accommodates lower rates for less complex neural network architectures, minimizing the risk of overfitting. Simultaneously, it allows for higher rates in more complex architectures to prevent overfitting while capturing intricate data relationships.

Maintaining Relu as the activation function and Adam as the optimizer was a deliberate choice. Both are widely acknowledged as efficient options for neural networks, striking a balance between performance and computational time. Altering these parameters could potentially increase computational time, limiting the number of trials and potentially missing out on identifying a more optimal set of hyperparameters.

IV.C. DECISION TREE REGRESSOR ALGORITHM (DTR)

Decision Tree models use trees to learn by splitting the dataset from the root to create leaf nodes until there is no need to split anymore. When the model makes predictions, it uses these splits. This model has both classification and regression variety, where it can provide discrete outputs or continuous output respectively. For this report I have decided to use the Decision Tree Classifier to compare it to the other three learning models. I have used the python library named scikit-learn to import and call the DecisionTreeClassifier() model, so I am able to use their website to check all the parameters that I can tune to adjust the performance of the model. There are 12 parameters in total that we can tune, however I took 8

of these parameters that I found to be important and tuned them, while letting the other parameters be their default setting. To find the best combination of parameters I used grid search and outputted a file which is sorted by best to worst ranking. This allowed me to see the performance difference between the best and worst combination of the selected parameters.

The combination of parameters I have use in grid search are:

- Criterion: gini, entropy, log_loss
- Splitter: best, random,
- Max_depth: 2, 6, 10, 14, 18
- Min_samples_split: 2, 6, 10, 14, 18
- Min_samples_leaf: 2, 6, 10, 14, 18
- Min_weight_fraction_leaf: 0.1, 0.3, 0.5
- Max_features: sqrt, log2
- Random_state: 2, 6, 10, 14, 18

The ‘criterion’ parameter is used to set the quality of the splits for the decision tree, where ‘gini’ is used for Gini impurity and ‘entropy’/‘log_less’ for Shannon information gain. The model uses this to identify the optimal splits of the decision tree, impacting the model accuracy.

The ‘Splitter’ parameter is used to set the strategy for choosing the split at each node in the decision tree created by the model, where ‘best’ is to pick the best splits while ‘random’ picks the splits randomly. This is important as it affects the accuracy and computational efficiency.

‘Max_depth’ is used to set the maximum depth of the decision tree hence limiting the number of levels that the model is allowed to have, which helps prevent overfitting.

‘Min_samples_split’ tunes the minimum number of samples needed to split an internal node for the decision tree created by the model.

‘Min_samples_leaf’ sets the minimum number of samples needed to be at a lead node of the decision tree, which can impact the granul;airty of decision making in the final branches of the decision tree.

‘Min_weight_fraction_leaf’ is used to choose the minimum weighted fraction of the sum total of weights needed to be at a lead node.

The parameter ‘Max_features’ sets the number of features that is considered when looking for the best split in a decision tree. Since this restricts the number of features, it decreases the risk of overfitting.

‘Random_state’ controls the randomness of the decision tree estimator. This is important to make experiments with the same parameters yield consistent results.

IV.D. Logistic Regression Algorithm (LogR)

Logistic Regression, a widely used classification algorithm, was employed to model the data and predict binary outcomes. To enhance its predictive capabilities, a systematic hyperparameter tuning process was executed through a grid search. The choices of hyperparameters for tuning were:

- Penalty: l1, l2, none
- C: 0.001, 0.01, 0.1, 1, 10, 100, 1000]
- max_iter: 100, 200, 300, 400, 500

After exhaustive evaluation across the parameter grid, the selected hyperparameters that yielded the best performance were:

- Penalty: l2,
- C: 100
- max_iter: 100

The 'C' parameter denotes the inverse of regularization strength, 'max_iter' defines the maximum number of iterations for convergence, and 'penalty' specifies the type of regularization applied ('l2' in this case). These optimized hyperparameters contribute to the model's accuracy and robustness in capturing underlying patterns within the data, showcasing the effectiveness of the fine-tuning process in Logistic Regression.

V. FINDINGS

Our experiment has provided us with different scores and insights that are analyzed in this section.

V.A. SCORES BEFORE OPTIMIZATION

The scikit-learn library has default parameters which are set for each of their models. This makes the overall usability easier for beginners as they don't have to add all the parameters and optimize them. Before optimizing our chosen models we have decided to use these default settings to get scores that we can use to compare the effectiveness of the optimization.

Using kaggle.com we were able to upload our output file to the chosen competition and get an accuracy rate for each of the chosen models. The results were:

Model	Kaggle Result
DTC	74.02%
RFC	79.35%
NN	73.55%
LR	79.26%

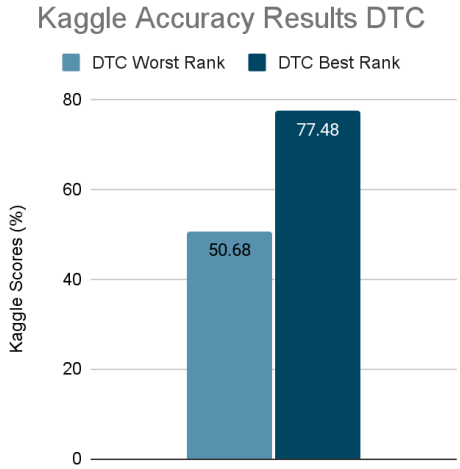
Table 1: Default Model Results

Looking at Table 1 above we can see that the default parameters of scikit-learn help our chosen models score in the range of 73.55% to 79.35% which is fairly good. Their default settings may be optimized to work with a variety of datasets which could leave room for improvement when focusing on a particular dataset instead. To find out we optimized our models by tuning the parameters to the dataset we are using.

V.B. SCORES AFTER OPTIMIZATION

Using grid search Decision Tree Classifier (DTC) we get a peak accuracy score of 77.48%. This means that we have gained 3.46% accuracy after optimizing the parameters from the default settings.

As mentioned in IV.C., I had outputted a file which ranks the combination of selected parameters from best to worst. I have tested both of those combinations to see the performance difference between them. In the graph before, titled 'Kaggle Accuracy Results DTC', we can see the scores provided by kaggle.com.



According to grid search, the best combination of the selected parameters on this dataset, which gives the peak accuracy score, would be:

- Criterion: gini
- Splitter: best
- Max_depth: 14
- Min_samples_split: 18
- Min_samples_leaf: 18
- Min_weight_fraction_leaf: 0.1
- Max_features: log2
- Random_state: 6

According to grid search, the worst combination of the selected parameters on this dataset would be:

- Criterion: entropy
- Splitter: best
- Max_depth: 2
- Min_samples_split: 6
- Min_samples_leaf: 14
- Min_weight_fraction_leaf: 0.5
- Max_features: sqrt
- Random_state: 2

This gives an accuracy score of only 50.68% which is poor. We can notice that Max_depth is set to 2

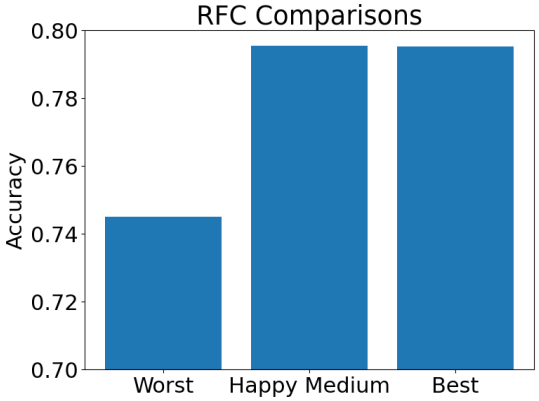
here, which may be one of the bigger reasons why it scored so low combined with others.

We can have a deeper understanding of the factors that cause a lower rank (hence, a lower score), if we look through the output from the grid search of DTC. In the appendix you can find a link to the ‘Simplified table of DTC grid search result’, where you can see the combination of the parameters and how they rank. There we can see that the parameters, ‘random_state’, ‘splitter’ and ‘min_weight_fraction_leaf’ don’t change for a big portion of the top ranked combinations, which may mean that the values ‘6’, ‘best’ and ‘0.1’ respectively, work to produce the best accuracy scores even if the other parameters are changing. The ‘criterion’ parameter also holds its value as ‘gini’ in the top rankings but not as much as the ones mentioned. The other parameters seem to be changing their values a lot comparatively as we go down the ranks. Also going down the ranks it can be noticed that the parameter ‘max_depth’ is not of the value ‘2’ until we go down the ranks significantly, supporting how it could be a big factor of a lower accuracy score as we noticed it for the lowest ranking combination of parameter which scored the lowest in terms of accuracy in kaggle.com. Also from the table we can notice that a significant amount of bottom ranks have ‘0.5’ as the ‘min_weight_fraction_leaf’ parameter, which may mean that it causes a big decrease in the accuracy score of the DTC model.

In the pursuit of refining the Logistic Regression model's performance, an extensive exploration of hyperparameters was conducted through a meticulous grid search process. This investigation led to the identification of a set of hyperparameters that significantly enhanced the model's predictive capabilities. The trio of {'C': 100, 'max_iter': 100, 'penalty': 'l2'} emerged as the optimal configuration. The hyperparameter 'C', governing the regularization strength, was fine-tuned to strike an optimal balance between fitting the training data precisely and preventing overfitting. Simultaneously, the 'max_iter' parameter, specifying the maximum number of iterations for model convergence, was carefully calibrated to ensure computational efficiency while refining model coefficients. The 'penalty' parameter, set to 'l2', underscored the adoption of ridge regularization, a choice that contributed to the model's stability and generalization capacity.

Notably, the pursuit of hyperparameter optimization was not merely an academic exercise but yielded tangible improvements in model performance. The default Logistic Regression model, when applied to the dataset, achieved a Kaggle score of 79.26%. However, with the integration of the optimized hyperparameters, the model exhibited a commendable boost, resulting in a Kaggle score of 79.47%. This marginal yet impactful increase underscores the significance of hyperparameter fine-tuning in enhancing the model's accuracy and effectiveness in making predictions. It also serves as empirical evidence that the intricacies of regularization strength, convergence iterations, and regularization type can collectively influence the model's predictive prowess. These findings, coupled with the Kaggle scores, not only validate the efficacy of the hyperparameter tuning process but also underscore the practical relevance of such optimizations in real-world applications.

The Random Forest Classifier (RFC) demonstrated exceptional performance, achieving a peak score of 79.54%. Subsequent analysis involved the utilization of the worst-performing algorithm, the best-performing algorithm, and the algorithm securing the 9th position, which was deemed the "Happy Medium." The selection of the 9th place algorithm as the compromise stemmed from a tie among the top eight algorithms for the first position. Additionally, the Happy Medium algorithm, distinguished by 100 n_estimators compared to the best algorithm's 500, exhibited greater efficiency in resource utilization.



A noteworthy observation was the Happy Medium algorithm's marginally superior performance

on the test dataset compared to the Best algorithm (79.54% vs. 79.52%). A detailed examination of the top 50 results obtained from the Grid Search highlighted the most effective hyperparameters for this dataset. Notably, all top 50 results (100%) featured enabled bootstrap. Among these, 60% had class_weight set as balanced, criterion exhibited an even split between log_loss and entropy at 50%, while for max_features, 36% were set as None, and 32% each were attributed to sqrt and log2. Regarding n_estimators, 48% were at 500, followed by 28% at 100, and 24% at 200. Lastly, warm_start featured 52% as True and 48% as False.

These outcomes underscore the significant impact of the bootstrap hyperparameter, as evidenced by its prevalence across all top 50 results. This observation aligns with the fact that the worst algorithm produced by Grid Search had bootstrap set to False.

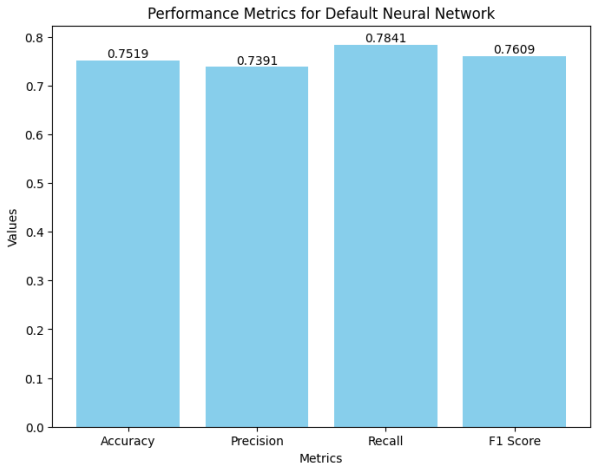
Following the optimization process, the neural network emerged as the highest-performing model, achieving an accuracy score of 80.14%. This marked a significant improvement from the unoptimized model, which stood at 73.55% accuracy. Notably, this transformation elevated the neural network from being the worst performer in its unoptimized state to the best among the models we tested. This outcome underscores the sensitivity of the neural network's architecture concerning model performance.

The optimized configuration for this dataset comprised:

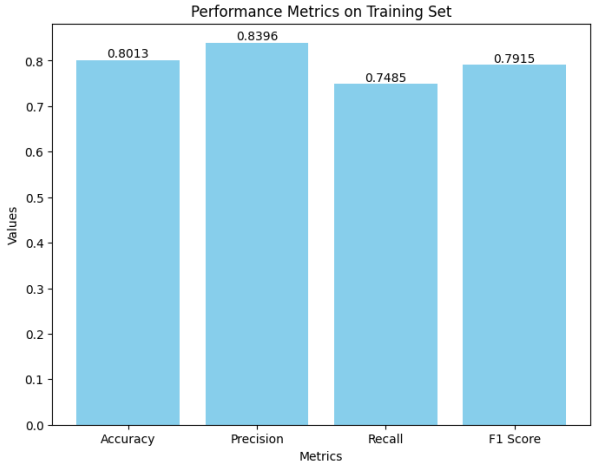
- Hidden Layers: 1
- Neurons in hidden layer: 450
- Dropout rate: 0.42

These findings are intriguing since conventional wisdom often leans toward deeper and more narrow architectures for neural networks. However, the optimized model contradicts this by employing a wider and shallower architecture, coupled with a relatively high dropout rate of 42%, meaning 42% of the neurons are deactivated during each training iteration. The key takeaway from this experiment is the substantial impact of the optimization process on the neural network's performance. The optimized network outperformed its unoptimized counterpart by approximately 8.97%. Such an increase could hold significant value in real-world scenarios where even marginal improvements in accuracy are critical, especially in fields like medical diagnoses. Although due to the constraints of the Kaggle competition, we were unable to evaluate Recall, Precision, and F1 score on the testing set, the metrics on the training set are provided below:

Default Network

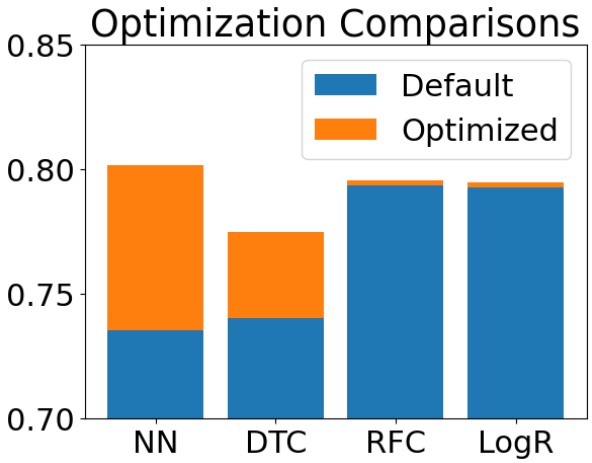


Optimized Network



Upon analysis, it's evident that the optimized network exhibits notably improved accuracy, precision, and F1 score. Surprisingly, however, it falls slightly behind in recall. This signifies that while the optimized model excelled in positive predictions, it also missed more instances compared to the unoptimized network. Overall, the optimized network emerges as the superior performer, boasting higher accuracy and F1 score. This trial allows us to deduce that the neural network architecture yielded a significantly greater performance increase than the shallower Machine Learning methods

post-optimization. Consequently, the neural network proved to be the most accurate among the tested models for this dataset.



Looking above at the graph, 'Optimization Comparisons', and table 2 from the appendix, we can see that NN got the biggest boost in performance between default and optimized models with a gain of 6.59% accuracy score, DTC came second with a gain of 3.46% , LR coming third was only a gain of 0.21% and RFC coming last with only a gain of 0.19%.

VI. CONCLUDING REMARKS

After the experiment was completed, we noticed that some of the models stand out more than the others as shown previously. In terms of peak accuracy scores Neural Network scored the best with 80.14% and Decision Tree Classifier scored the lowest with 77.48%, however the primary goal of this report wasn't focusing on which model is the best, but instead looking at how big of a difference can optimizing the hyperparameters can make to the models. As discussed previously we saw that NN saw the biggest difference after optimization with a gain of 6.59% while RFC only saw a gain of 0.19%. Although we were able to improve two of the four models by a significant amount, we weren't able to reach closer to 100% accuracy score meaning that there is further work to be done through optimizing the models and getting better results. In conclusion, optimizing the hyperparameters can yield a significant performance gain depending on the model, and some models work great even with the default scikit-learn parameters which is great for beginners to get into machine learning models.

VII. FUTURE WORK

Exploring the effects of hyperparameter tuning on model performance offers a vast landscape for further investigation, considering the significant impact observed in the enhancement of neural network performance. Delving deeper into this area holds substantial potential for refining the accuracy of Machine Learning models.

One avenue for exploration involves broadening the spectrum of model types and conducting tests across various scenarios. Scikit-Learn alone offers a diverse array of classification models to explore. Expanding into regression analysis could unlock a different level of insight, offering additional metrics and enabling more intricate conclusions based on these metrics. Furthermore, venturing into domains beyond tabular data, such as image recognition, presents an intriguing avenue for exploration.

Extending the duration and scope of our optimization processes by exploring a wider range of hyperparameters would provide a more comprehensive understanding of hyperparameter optimization. However, resource limitations hinder us from running optimizations for extended periods. Yet, broadening the range of hyperparameters could unveil new strategies for fine-tuning models, potentially elevating their performance even further.

Experimenting with various optimization methods or conducting this study on different datasets could yield fascinating results and shed light on which methods excel in diverse scenarios. Assessing the effectiveness of different optimization methods across various models and datasets would be an engaging area to explore. Diversifying datasets would enable a holistic understanding of hyperparameter tuning's efficacy in varying scenarios and its specific impacts on individual models within those scenarios.

VIII. REFERENCES

[1] Bari Antor, Morshedul, et al. “A Comparative Analysis of Machine Learning Algorithms to Predict Alzheimer’s Disease.” Journal of Healthcare Engineering, Hindawi, 3 July 2021, www.hindawi.com/journals/jhe/2021/9917919/

[2]Singh, Karman, et al. “Exploratory Data Analysis and Machine Learning on Titanic Disaster ...” Exploratory Data Analysis and Machine Learning on Titanic Disaster Dataset, IEEE, 9 Apr. 2020, ieeexplore.ieee.org/document/9057955/.

[3]Cukierski, Will. “Spaceship Titanic.” Titanic - Machine Learning from Disaster, Kaggle, 2012, www.kaggle.com/competitions/spaceship-titanic/overview.

[4]“Sklearn.Ensemble.Randomforestclassifier.” Sklearn.Ensemble.RandomForestClassifier, scikit, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed 3 Dec. 2023.

[5]“Sklearn.Preprocessing.StandardScaler.” Sklearn.Preprocessing.StandardScaler, scikit, scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed 3 Dec. 2023.

[6]“Sklearn.Linear_model.Logisticregression.” Sklearn.Linear_model.LogisticRegression, scikit, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 3 Dec. 2023.

[7]“Sklearn.Tree.Decisiontreeclassifier.” Sklearn.Tree.DecisionTreeClassifier, scikit, scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Accessed 3 Dec. 2023.

[8]“Sklearn.Model_selection.GRIDSEARCHCV.” Sklearn.Model_selection.GridSearchCV, scikit, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed 3 Dec. 2023.

[9]“The Sequential Model : Tensorflow Core.” TensorFlow, www.tensorflow.org/guide/keras/sequential_model. Accessed 3 Dec. 2023.

[10]“Tf.Keras.Layers.Dense : Tensorflow V2.14.0.” TensorFlow, www.tensorflow.org/api_docs/python/tf/keras/layers/Dense. Accessed 3 Dec. 2023.

[11]Team, Keras. “Keras Documentation: Bayesianoptimization Tuner.” Keras, keras.io/api/keras_tuner/tuners/bayesian/. Accessed 3 Dec. 2023.

IX. APPENDIX

Unchanged table of DTC grid search result:https://docs.google.com/spreadsheets/d/1FLb16j1w6ACWGJ-G7Xg-nhI_vYcBYQ3XGz-RDYBGpDM/edit?usp=sharing

Simplified table of DTC grid search result:https://docs.google.com/spreadsheets/d/1QOYUbRXUx42iUyivKnUTyRABJELmx8Nc2OeIqSspvEg/edit?usp=sharing

Models	Kaggle Scores Before (%)	Kaggle Scores After (%)
NN	73.55	80.14
DTC	74.02	77.48
RFC	79.35	79.54
LR	79.26	79.47

Table 2: Accuracy Scores Before and After

X. GROUP CONTRIBUTIONS

Matthew Muscedere

- Wrote introduction and Abstract
- Wrote Tools Used
- Wrote IV.B
- Wrote neural network part of V
- Wrote future work
- Wrote Pycaert.ipynb
- Wrote neural_network.ipynb

Adib Md Alim Chowdhury

- Wrote DTC.ipynb
- Wrote first part of II
- Wrote IV.C
- Wrote V.A
- Wrote V.B DTC part
- Wrote V.I
- Made Table 1 and 2
- Setup DTC grid search tables for appendix
- Made report document and worked on format

Tayo Alalade:

- Wrote Relevant literature paragraph
- Wrote Logistic Regression paragraph
- Wrote references section
- Wrote code for logistic_regression.ipynb
- Worked on the formatting of doc

Caden Quiring:

- All data preparation (data_prep.ipynb + data folder)
- Wrote README.md
- Created the algorithm template to follow for peers (submission_example.ipynb)
- Wrote all code for the Random Forest Classifier (quiring_algo.ipynb)
- Designed some graphs (RFC Comparisons, Optimization Comparisons)
- Wrote IV.A
- Wrote RFC section of V.B