CS 178: Final Project

FRONT COVER

CS 178 Final Project

Names	Amy Lee, Rebecca Tio, Varshini Vijay
IDs	17974032, 38625325, 22508415
Dataset	Diabetes 130-US Hospitals
Classification Methods	kNN, Logistic Regression, Neural Network, Decision Tree

Dataset: Diabetes 130-US Hospitals
Classification Methods: kNN, Logistic Regression, Neural Network, Decision Tree

Authors: Amy Lee, Rebecca Tio, Varshini Vijay
Team Name: 31 (Diabetes)

June 8 2025
Collaborators: None

Summary

The supervised learning project in machine learning implements a variety of classifiers, such as K-Nearest Neighbors, Logistic Regression, Neural Networks, and Decision Trees, on the UCI Diabetes 130-US Hospitals dataset. Several hyperparameter values, including number of nearest neighbors, learning rate, hidden layer sizes, and minimum number of leaf nodes, were evaluated respectively, and exploratory data analysis was performed to achieve the highest possible accuracy. The neural network classifier had the highest test accuracy of 59.42% and captured the complexity of data patterns.

Data Description

The Diabetes 130-US Hospitals dataset contains records of patients diagnosed with diabetes, some data recorded during their stay, and if they were readmitted later on, and if it was within 30 days using the labels: >30 for greater than 30, <30 for within 30, or NO for no readmission. There are over 100,000 patients in the dataset and ~50 features detailing attributes such as drug prescriptions, lab results, and demographics.

For dataset exploration, we looked at a generation of summary statistics (seen in **Figure 1.1** and **1.2** in Appendix A) and a histogram for numerical features (seen in **Figure 1.3** in Appendix A). With variables such as outpatient number, emergency number, and inpatient number having medians of 0 over 100,000 rows of patients, this can demonstrate that most patients had 0 visits and few had many, while the number of lab procedures has a rather broad spread with a normal distribution, with outliers at 0 lab procedures.

The research paper we examined in relation to the diabetes dataset was *Variance-Based Feature Importance* which detailed a method to determine the importance of features in a neural network by analysis when features are changed. Specifically, it produced a list of features based on weight fluctuations during training, allowing for a global interpretation of feature interpretation. We approached a smaller subset of this analysis by looking into LIME, which does a local comparison of features, and single feature removal against kNN and examining those results.

Classifiers

Our team looked over four different classifiers using the Diabetes dataset:

Decision Tree classifier

This classifier creates a binary tree of decisions that lead to different labels. For example, if there are only three features where a is a boolean, b is a float, and c is an int, a decision tree may label it as ‘dog’ if a is false, b is >0.5 and c is <-4. Other values of a, b, or c may lead to a different “decision” and thus a different label. The hyperparameter settings investigated for decision trees were a range of 1-20 for max depth of the tree, a range of 90-100 for min leaf samples, the minimum number of samples each branch/leaf must have before making a decision, and a range of 1-100 for maximum leaf nodes, which is

the maximum leaf nodes that were allowed in the tree. We used scikit-learn's implementation of DecisionTreeClassifier.

kNN classifier

This classifier looks into a given k-value's nearest neighbor of a chosen point. For example, if k is 50, it would take all 50 nearest neighbors and the majority label would be the label of this point. The hyperparameter settings investigated for kNN were 1,10, 25, 50, 100, 200, 500 neighbors, uniform or distance weight, that determined how much influence a neighbor had in evaluating the result, Euclidean or Manhattan or Minkowski distance metric, that determined the equation to calculate the distances between points. We used scikit-learn's implementation of KNeighborsClassifier.

Neural Network classifier

For this classifier we examined the Multilayer Perceptron model which uses layers of neurons to learn patterns in the data using layers. There is an input layer, a hidden layer, and an output layer, where layers are connected to the next, multiplying the input against weights and biases. The hyperparameter settings investigated were the shapes of hidden layer sizes, ((100,), (50,), (100,100), (64,)), activation functions (ReLU, tanh, logistic) that introduces non-linearity, solvers (sgd, adam), to best optimize the log-loss function and minimize loss, alphas (0.1, 0.01, 0.001, 0.0001), used for regularization, and initial learning rates (0.05, 0.01, 0.001), which determine how much models change between iterations. We used scikit-learn's implementation of MLPClassifier.

Logistic classifier

This classifier looks at probabilities of a point having a label using a logistic function, such as sigmoid. The hyperparameter settings investigated for logistic regression were L1 or L2 penalties, which are methods of regularizing the model to stop overfitting, liblinear and lbfgs solvers, which determined the most efficient optimization method, true and false for the fit intercept, which would just move the height from the origin, and 0.1 or 1.0 for C, which is the inverse of the regularization strength, making it so that small values creates more regularization. We used scikit-learn's implementation of LogisticRegression.

Experimental Setup

We began our experiments by partitioning data into a 20% testing split, 80% for training and validation, where 75% was for testing and 25% for validation. We then replace missing values in each partition with the mode of each column and scale using StandardScaler given by scikit-learn. The experiments are reproducible with the seed 1234 and the steps above.

For classifier evaluation, training and validation accuracy plots were created for different hyperparameter configurations:

Decision Tree classifier (general process)

We first used default parameters, then looked at scaling different hyperparameters (with 'gini' used as criterion) and used a range of max_depths, min_leaf_samples, and maximum leaf nodes, where we could then evaluate the best validation accuracy and pick our model..

kNN classifier (general process)

We used GridSearchCV with different values, done on the pipeline used to preprocess variables and apply the KNeighborsClassifier where parameters could be automated. It would generate a graph

showing the Mean CV Accuracy against K Neighbors using the different weights. The best hyperparameters could then be selected based on the highest accuracy found.

Logistic and Neural Networks classifier (general process)

Logistic regression and neural networks used the same process: we first tried different combinations of the parameters of both classifiers using a dictionary to output the results in a table, allowing for ease of readability. We then evaluated the training and validation accuracy, selecting the best performing model.

Experimental Results

For a bar chart on classifiers with their best-performing hyperparameters and test accuracies consult ***Figure I-II*** in Appendix D*. For respective accuracy charts, see* ***Figure V-X*** *in Appendix D.* The feedforward neural network displayed the lowest testing error.

Generally, more training data is believed to be better until diminishing returns on accuracy are received. With small training sets, the model may perform well on training data (low training error) but poorly on new unseen data (high validation error), formally defined as overfitting. With more training examples, the model is less likely to overfit to a small subset of the data. The model does not memorize or capture every individual data point, hence, the training error may increase. However, generalization is improved and validation error decreases. This is demonstrated in our results, as the number of training samples varied with error.

For the MLP, Logistic Regression, and kNN, validation error steadily decreases as the training size increases, reaching its lowest point at 20,000 samples. For logistic regression and MLP, the training error seems to increase. The Decision Tree reaches its minimum error early, around 2,500 to 5,000 training examples, beyond which the error curve flattens, implying that more data does not significantly improve performance **(Figure III** in Appendix D**)**.

In the bias-variance tradeoff, while a higher bias may lead to underfitting, a higher variance can cause overfitting. Regularization prevents overfitting, and introduces a penalty for model complexity, creating simpler models that are able to better generalize and improve performance on new testing data. Different models perform regularization differently:

Logistic Regression Model

Inverse regularization strength C was varied wherein a smaller C indicated stronger regularization by penalizing larger feature weights. At C = 100, the model achieved a low training error (~0.337) but suffered from a high test error (~0.45). An optimal range of C=1-10 lowered the test error to ~0.41.

Feedforward Neural Network/Multi-layered Perceptron Model

The alpha parameter was varied where a higher alpha means stronger regularization, as it incorporates a penalty to feature weights in the cross-entropy loss function. With alpha = 0.0001, the model overfit (train error ~0.373, test error ~0.403), and underfit with alpha = 1.0 (train ~ test error ~ 0.38). Alpha= 0.01–0.02 reduced the test error to ~0.373–0.374.

kNN Model

Regularization is controlled by the number of neighbors (k). At k = 1, the model memorized the training data (train error = 0.0) but generalized poorly (test error ~0.52). Increasing k to 3–5 yielded the best generalization (test error ~0.43–0.44).

Decision Tree Model

The minimum samples per leaf were varied in an attempt to reduce the sensitivity of every single data sample. As it increased from 90 to 99, training error rose slightly (~0.381 to ~0.384), while validation error remained nearly constant around 0.413-0.414.

Overall, the most substantial accuracy improvements came from regularization in MLP, Logistic Regression, and kNN, while Decision Tree remained largely unaffected **(Figure IV** in Appendix D**)**.

In an attempt to examine how specific features affect accuracy in terms of kNN, we implemented a case where we would remove a single feature and find the accuracy across all features. (See **Figure 2** in the Appendix B). The resulting box plot had low deviation in terms of the best and worst performing feature removal and was relatively uniform across all feature removals.

We found that when removing time_in_hospital, accuracy was 57.65%, which was the best performing model. This may signify that length of stay does not hold much merit in terms of readmission probability. We also found some global minima where removing number_inpatient had the worst accuracy at 56.26%, with local minima of 57.09% for number_outpatient, and 56.95% for discharge_disposition. This may signify that the number of outpatient and inpatient visits have some effect on the patient getting readmitted, where chances for past readmission often has an influence on chances for future readmission. Discharge disposition refers to how the diabetes patient left the hospital they were treated at, for example if they were transferred to another facility, undergo hospice care, or were discharged home, and thus less likely to readmit, while patients that were transferred to another facility are more likely to readmit in the case that they have future issues.

Insights

One aspect that surprised our team is that logistic, kNN, and decision trees ended up having an accuracy of around 57% every time at the maximum, however if we turn it into a binary classifier of <30 or either no readmission or >30, the accuracy increases to around 80% and around 60-65% for other binary classifications.

Additionally, the results we obtained had errors made by the classifier which are also hard for humans to identify. One such way the errors were explained was using LIME. As depicted in **Figures 5-8** (Appendix C), features that positively contribute to the chosen classification are green and those negatively contributing are red. If the green and red bars seem close in quantity, it depicts that the classifier is relatively uncertain. **Figure 3** (Appendix C) is an example of a correct classification of '>30' readmittance and the other (**Figures 4** in Appendix C) is a misclassification of 'NO' readmittance as '<30' readmittance. **Figures 3-8** (Appendix C) are LIME representations of the logistic, kNN, and neural network classifiers each with examples that correctly classify the data and those that do not. Decision trees do not need LIME representations because it is already inherently explainable. Therefore, it is seen in the LIME representations that the kNN classifier is relatively indecisive, while the neural network and logistic regression are decisive as they have more green quantities (positive contributors to classification) than red (negative contributors).

Conclusively, we can speculate that the strengths of using machine learning is that it can weigh a large number of features to make a classification that would be hard for a human to discern. The weaknesses of machine learning is that it only works with the data it's given, therefore if it is given faulty or biased data, its classifications would be biased as well. One way we can improve our classifiers is by giving it more features to analyze and ensure that data is not biased. These could include graphical data or other features that were not accounted for.

Contributions

Amy worked on the data description paragraph, experimental setup in terms of both code and paragraph, the classifiers paragraph, kNN code and analysis, and varying the number of features against kNN.

Varshini worked on the initial dataset setup, the summary paragraph, the experimental results paragraph, logistic regression and neural network code and analysis, learning curves, and the effects of regularization code and analysis.

Rebecca worked on summary statistics, the classifiers paragraph, the insights paragraph, decision trees classifier code and analysis, LIME exploration, and the final edit of the report.

Appendix A: Summary Statistics

Numerical

	feature	range	std	var	mean	mode
0	admission_type_id	7	1.445403	2.089189	2.024006	1
1	discharge_disposition_id	27	5.280166	27.880148	3.715642	1
2	admission_source_id	24	4.064081	16.516753	5.754437	7
3	time_in_hospital	13	2.985108	8.910868	4.395987	3
4	num_lab_procedures	131	19.674362	387.080530	43.095641	1
5	num_procedures	6	1.705807	2.909777	1.339730	0
6	num_medications	80	8.127566	66.057332	16.021844	13
7	number_outpatient	42	1.267265	1.605961	0.369357	0
8	number_emergency	76	0.930472	0.865779	0.197836	0
9	number_inpatient	21	1.262863	1.594824	0.635566	0
10	number_diagnoses	15	1.933600	3.738810	7.422607	9

Figure 1.1. Summary statistics for numerical data (Diabetes dataset)

Categorical

	feature	mode	num_unique
0	race	Caucasian	5
1	gender	Female	3
2	age	[70-80)	10
3	weight	[75-100)	9
4	payer_code	MC	17
5	medical_specialty	InternalMedicine	72
6	diag_1	428	716
7	diag_2	276	748
8	diag_3	250	789
9	max_glu_serum	Norm	3
10	A1Cresult	>8	3
11	metformin	No	4
12	repaglinide	No	4
13	nateglinide	No	4
14	chlorpropamide	No	4
15	glimepiride	No	4
16	acetohexamide	No	2
17	glipizide	No	4
18	glyburide	No	4
19	tolbutamide	No	2
20	pioglitazone	No	4
21	rosiglitazone	No	4
22	acarbose	No	4
...			
32	metformin-rosiglitazone	No	2
33	metformin-pioglitazone	No	2
34	change	No	2
35	diabetesMed	Yes	2

Figure 1.2. Summary statistics for categorical data (Diabetes dataset)

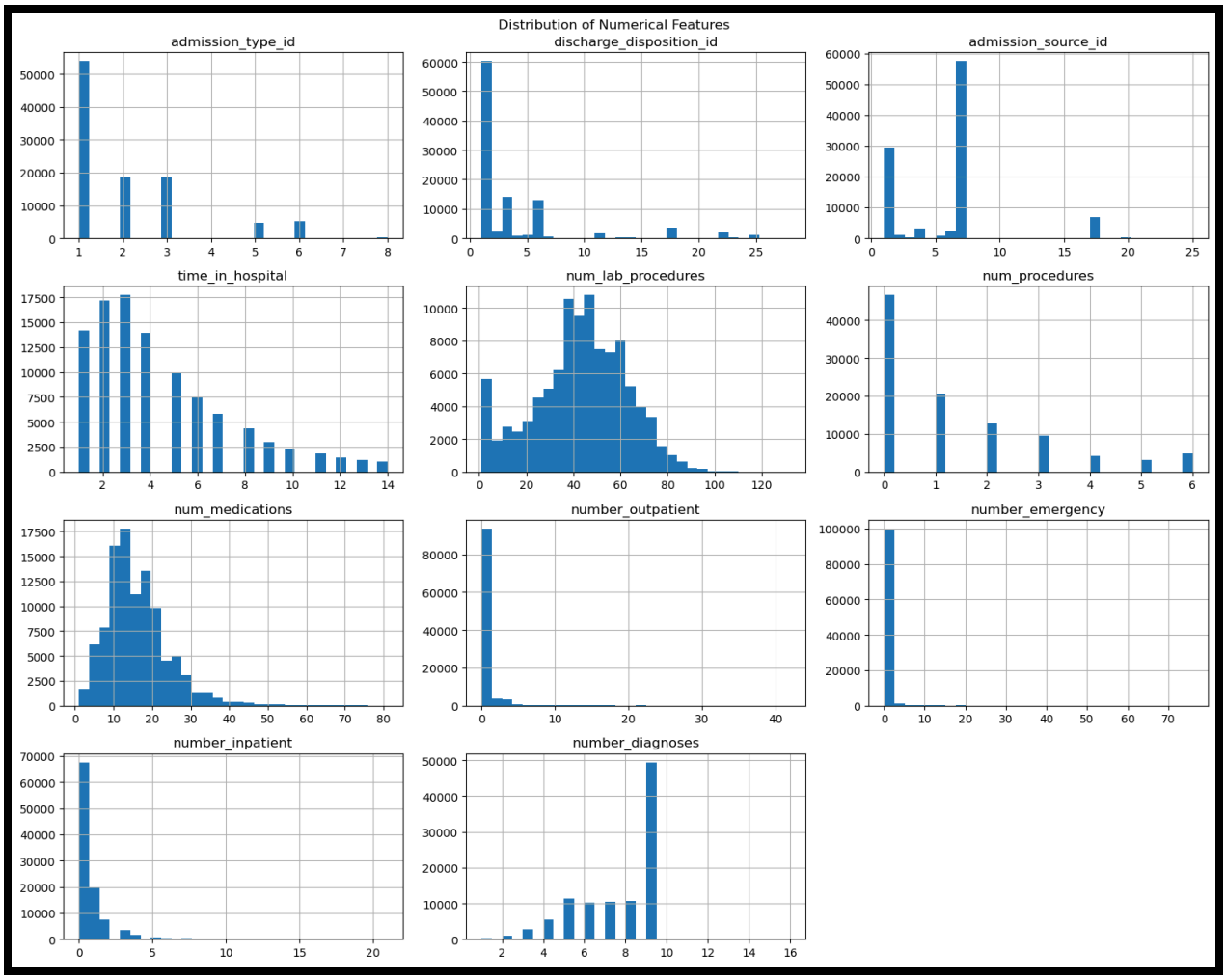

Figure 1.3. Histogram generated for numerical data (Diabetes dataset)

Appendix B: Removing Features

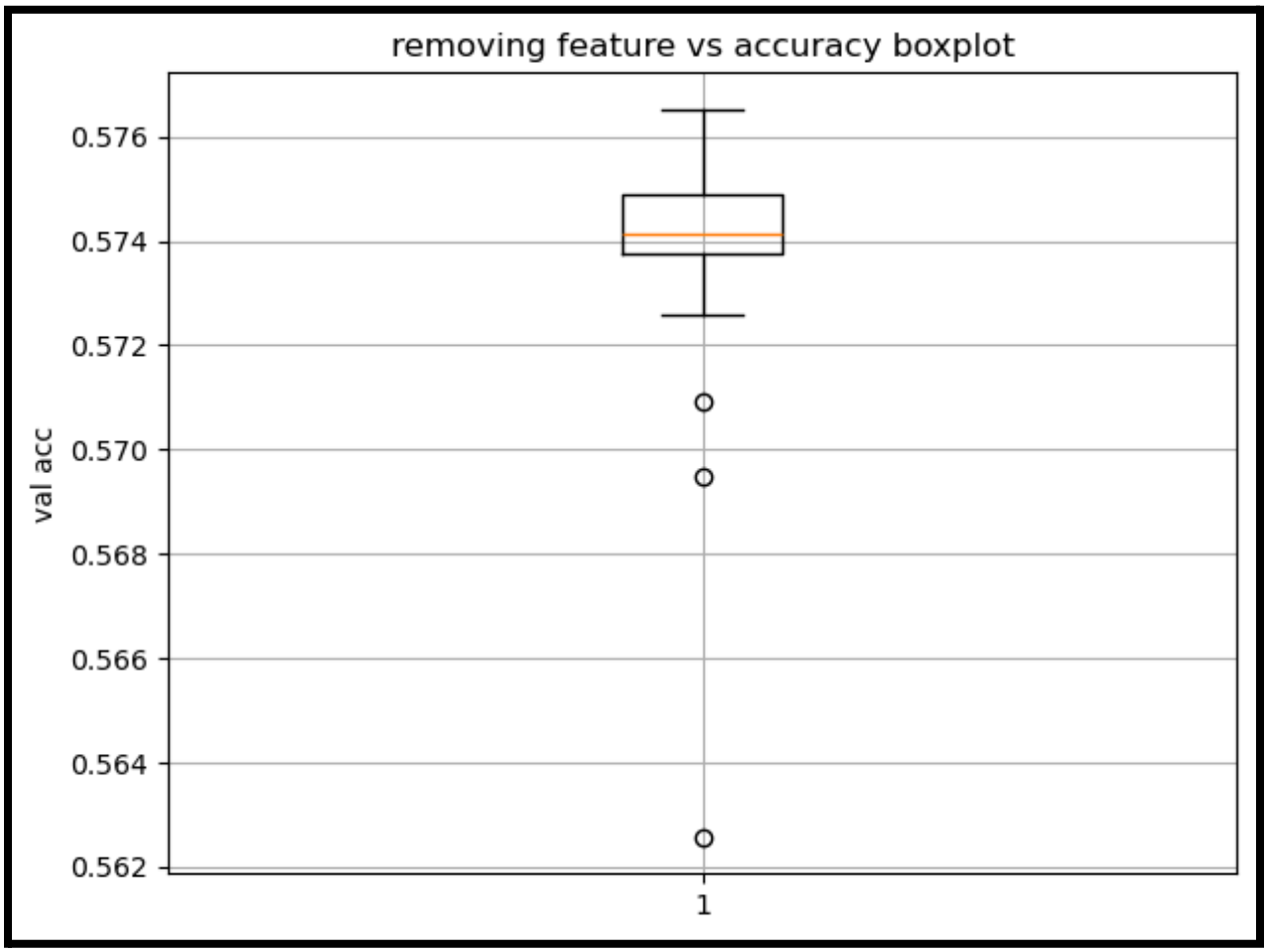

Figure 2. Boxplot detailing effects of removing features against accuracy

Appendix C: LIME Representations

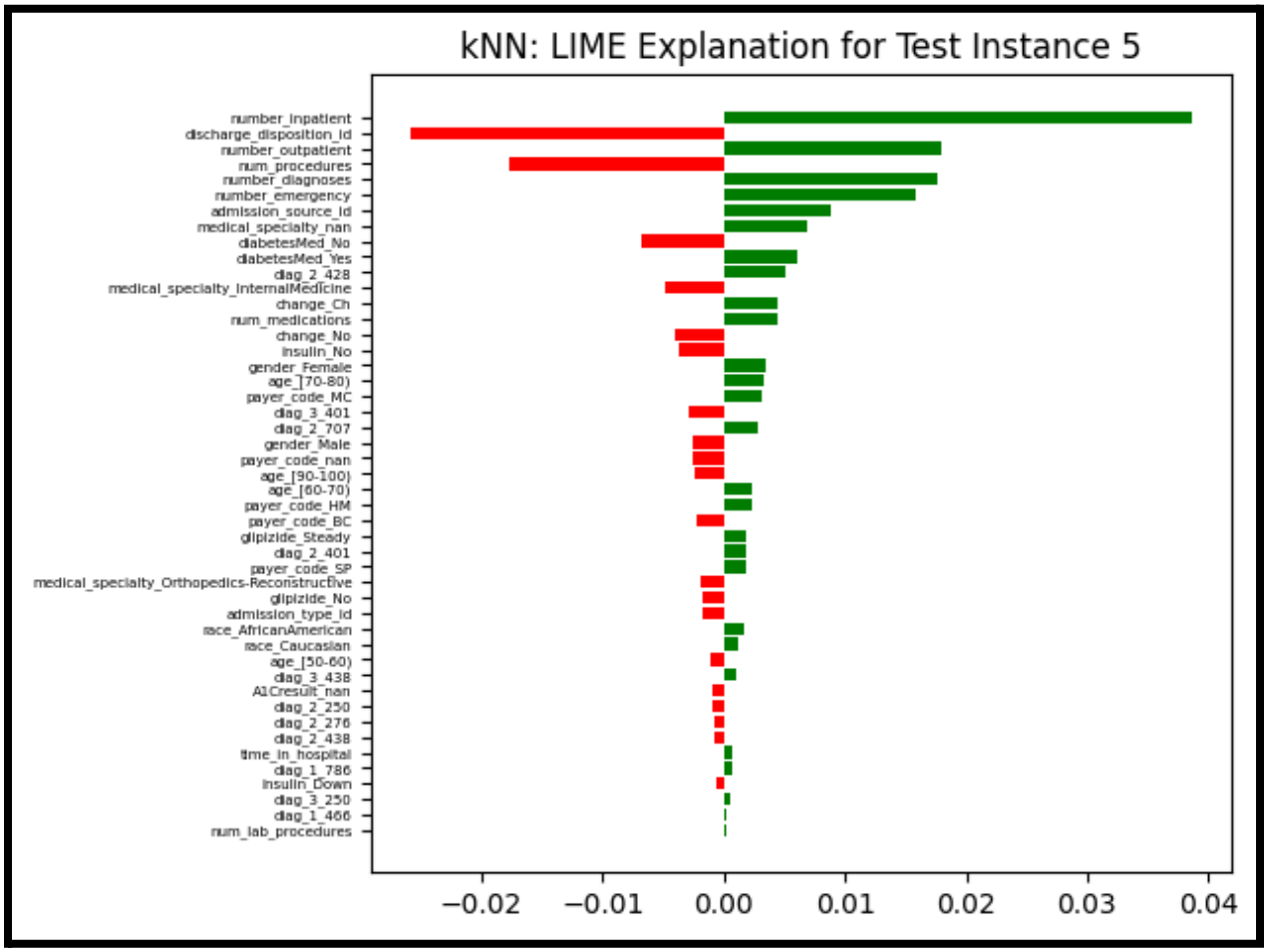

Figure 3. kNN: Boxplot using LIME where prediction matches actual (instance 5)

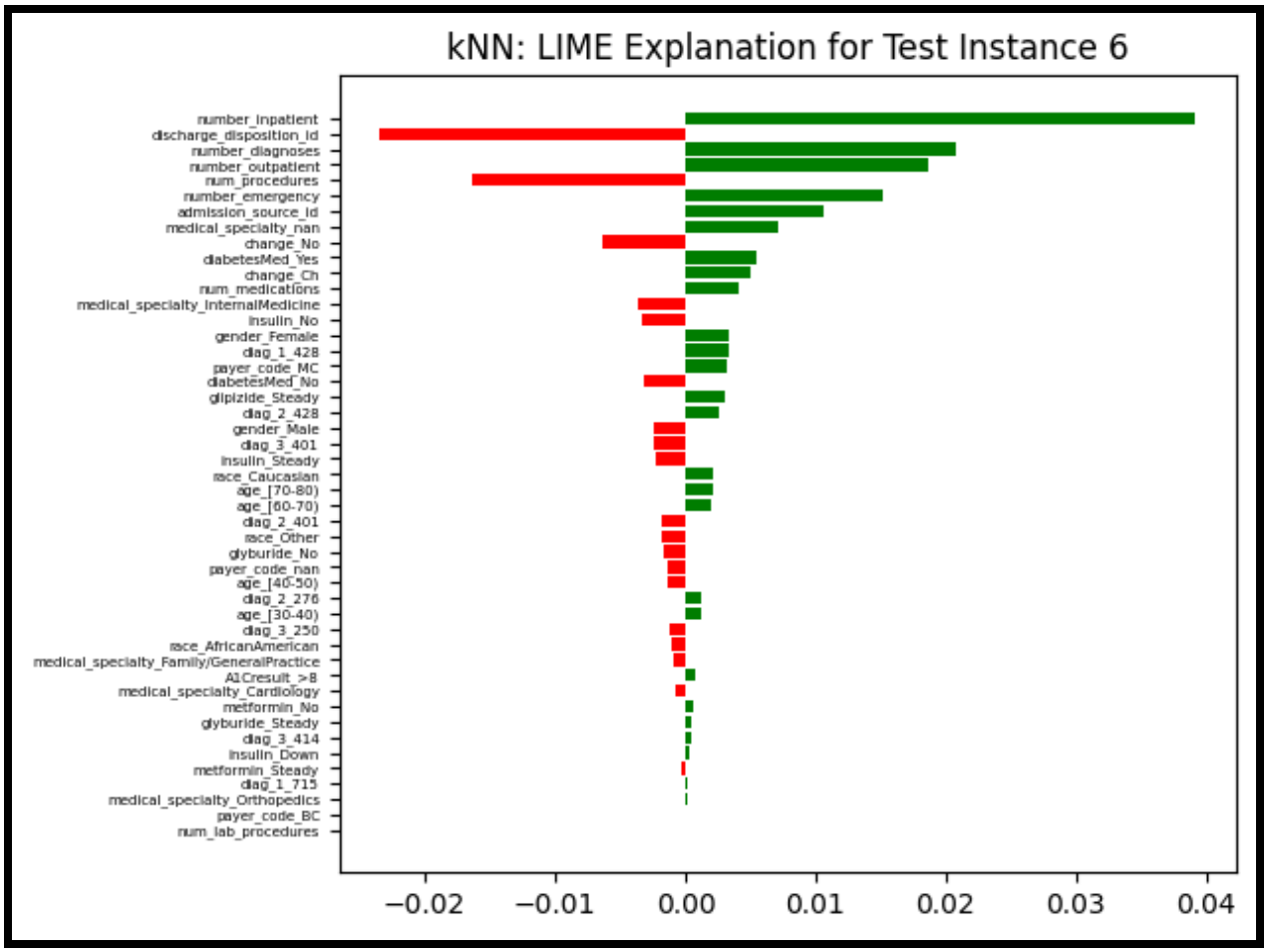

Figure 4. kNN: Boxplot using LIME where prediction does not match actual (instance 6)

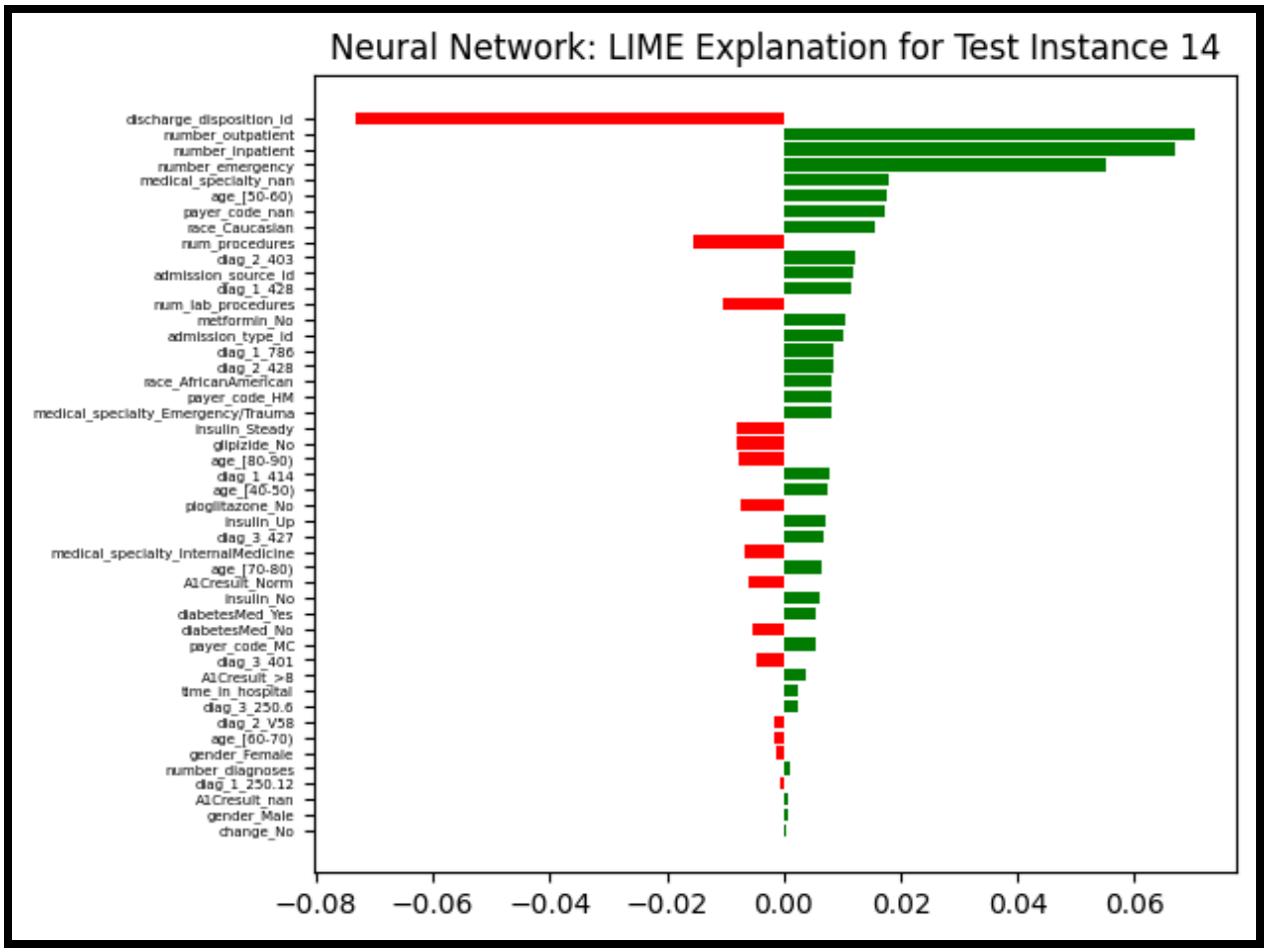

Figure 5. MLP: Boxplot using LIME where prediction matches actual (instance 14)

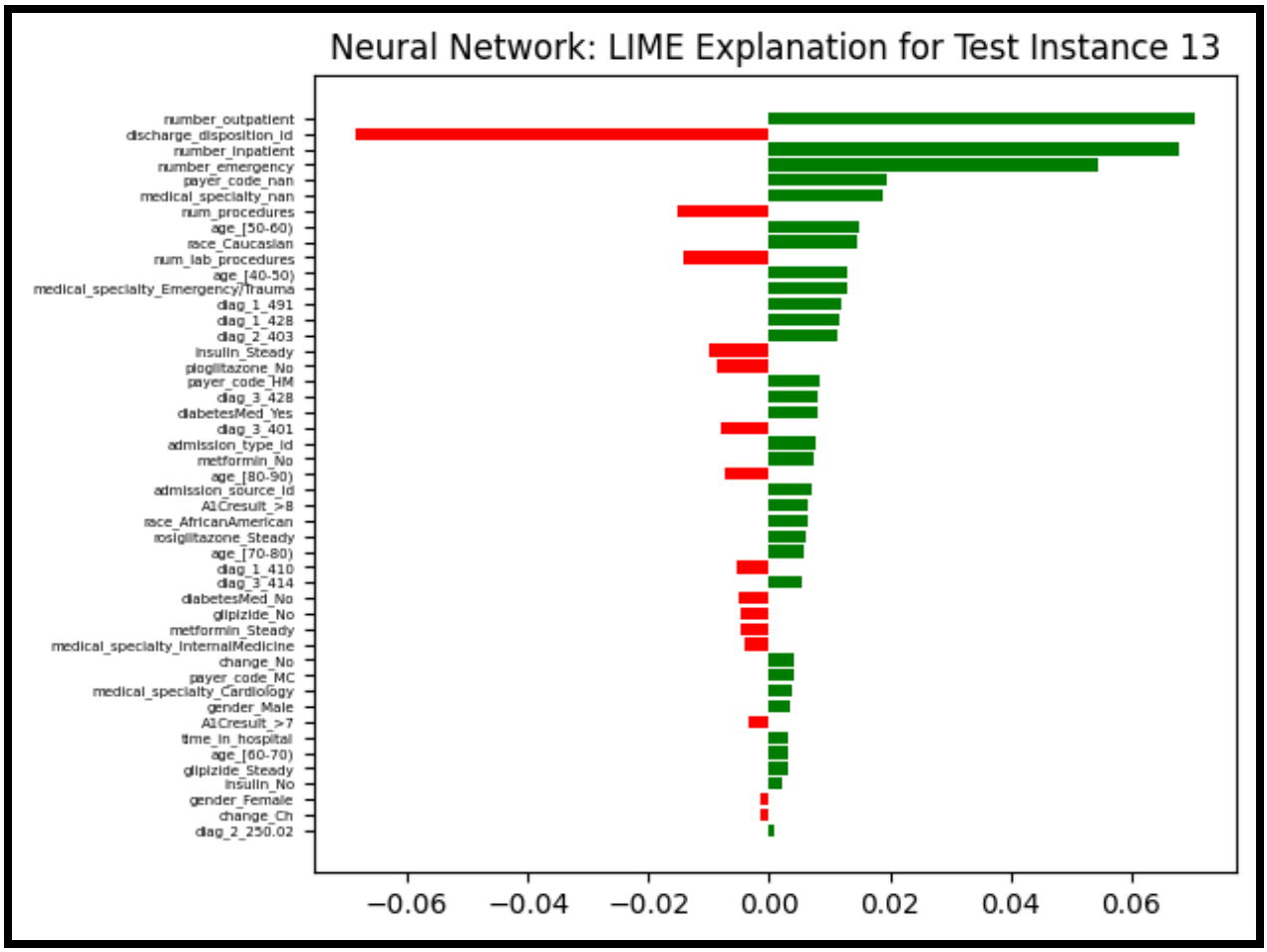

Figure 6. MLP: Boxplot using LIME where prediction does not match actual (instance 13)

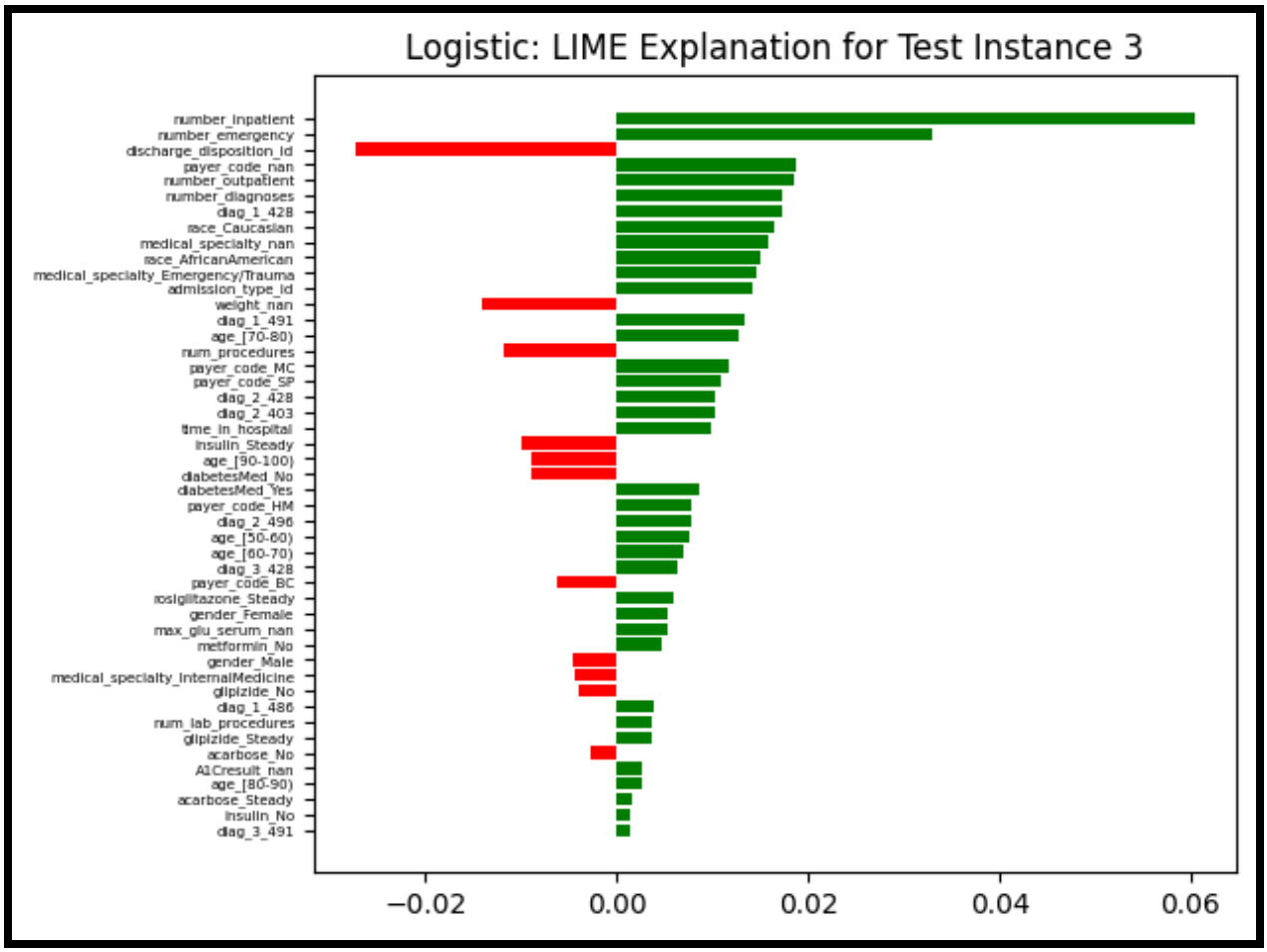

Figure 7. Logistic: Boxplot using LIME where prediction matches actual (instance 3)

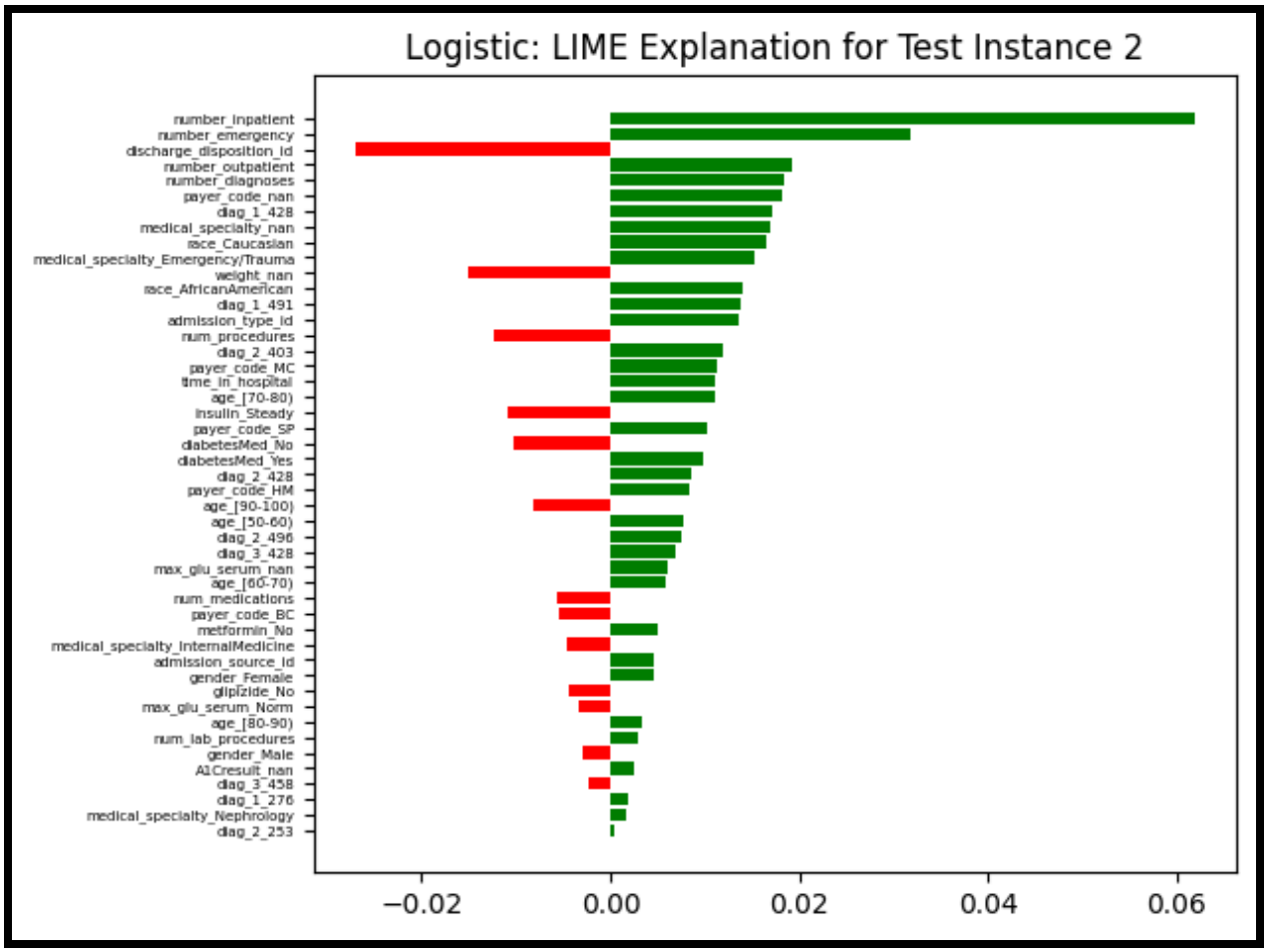

Figure 8. Logistic: Boxplot using LIME where prediction mismatches label (instance 2)

Appendix D: Other

Classifier	Configuration	Test Accuracy (%)
MLP Classifier	hidden_layer_sizes=(100,), activation='relu', solver='sgd', alpha=0.1, learning_rate_init=0.05	59.42
Logistic Regression	penalty='l2', solver='lbfgs', C=0.01	58.30
KNN Classifier	n_neighbors=75, weights='distance', metric=''minkowski'	56.97
Decision Tree	criterion='gini', max_depth=7, min_samples_leaf=100, min_samples_split=2	58.08

Figure I. all best accuracies

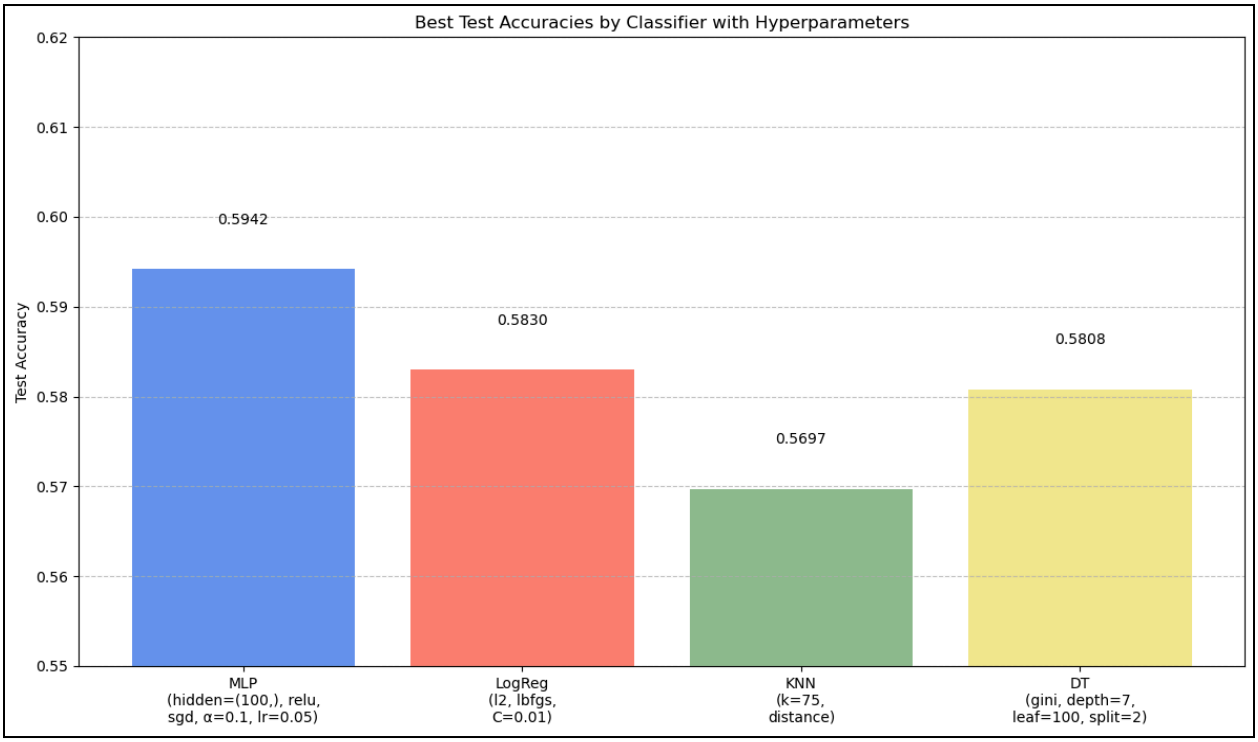

Figure II. all best accuracies (table form)

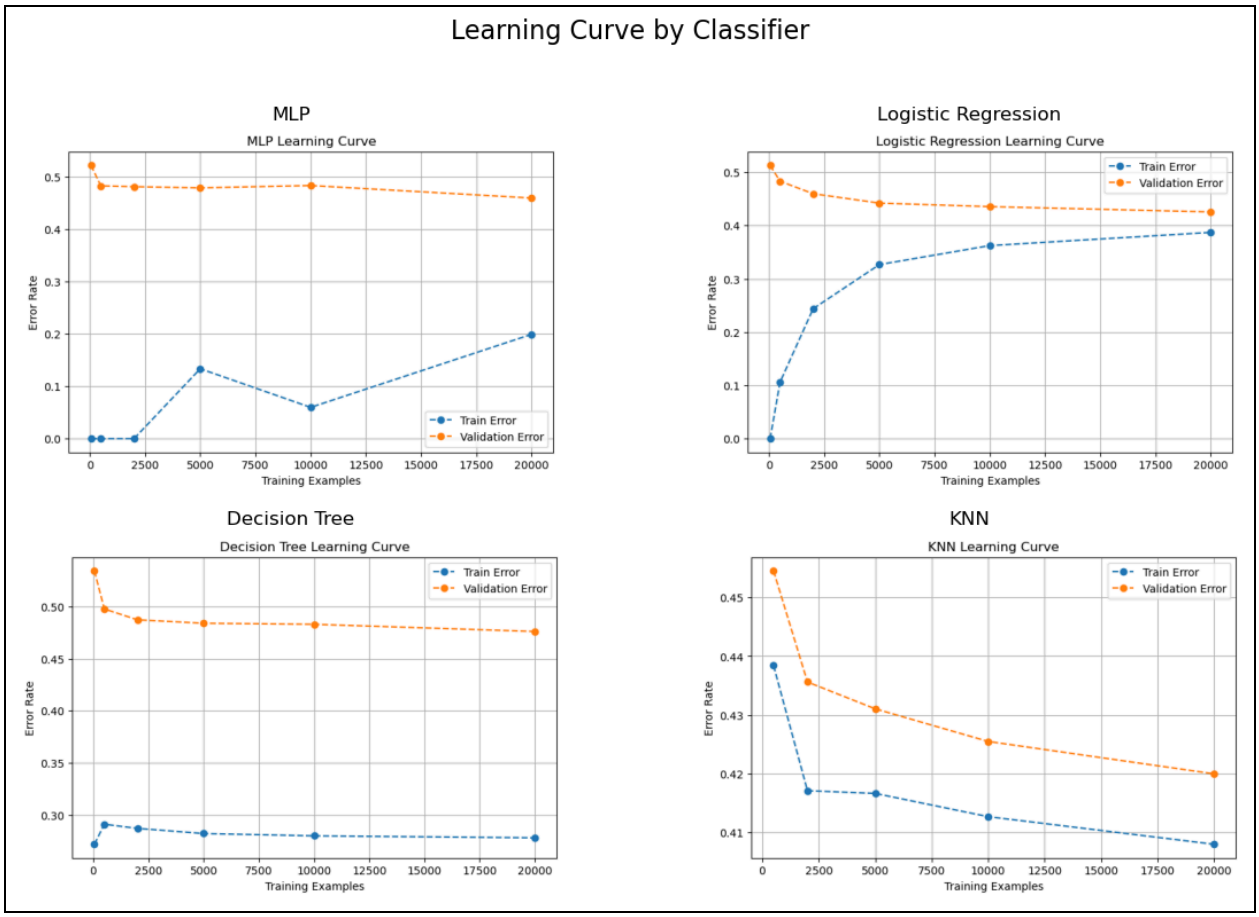

Figure III. Learning Curve by Classifier Graphs

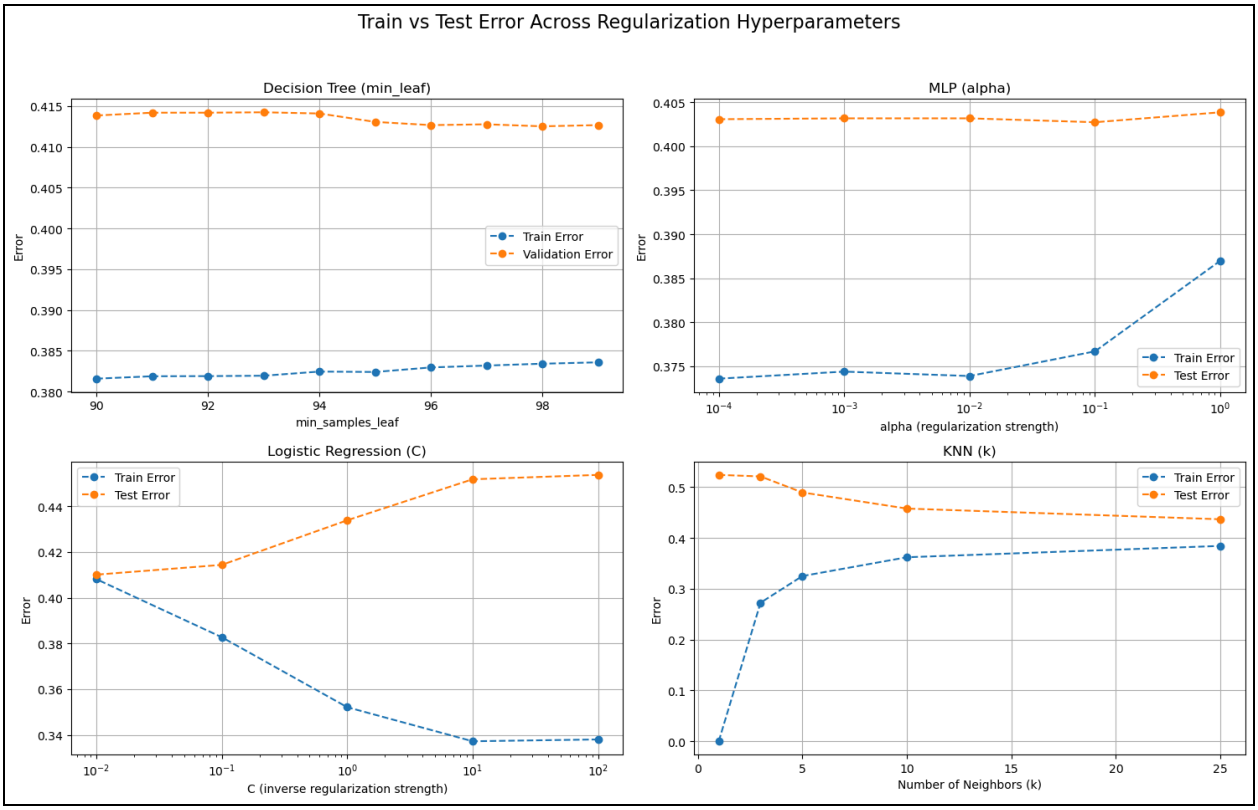

Figure IV. Train vs Test Error Across Regularization Hyperparameters

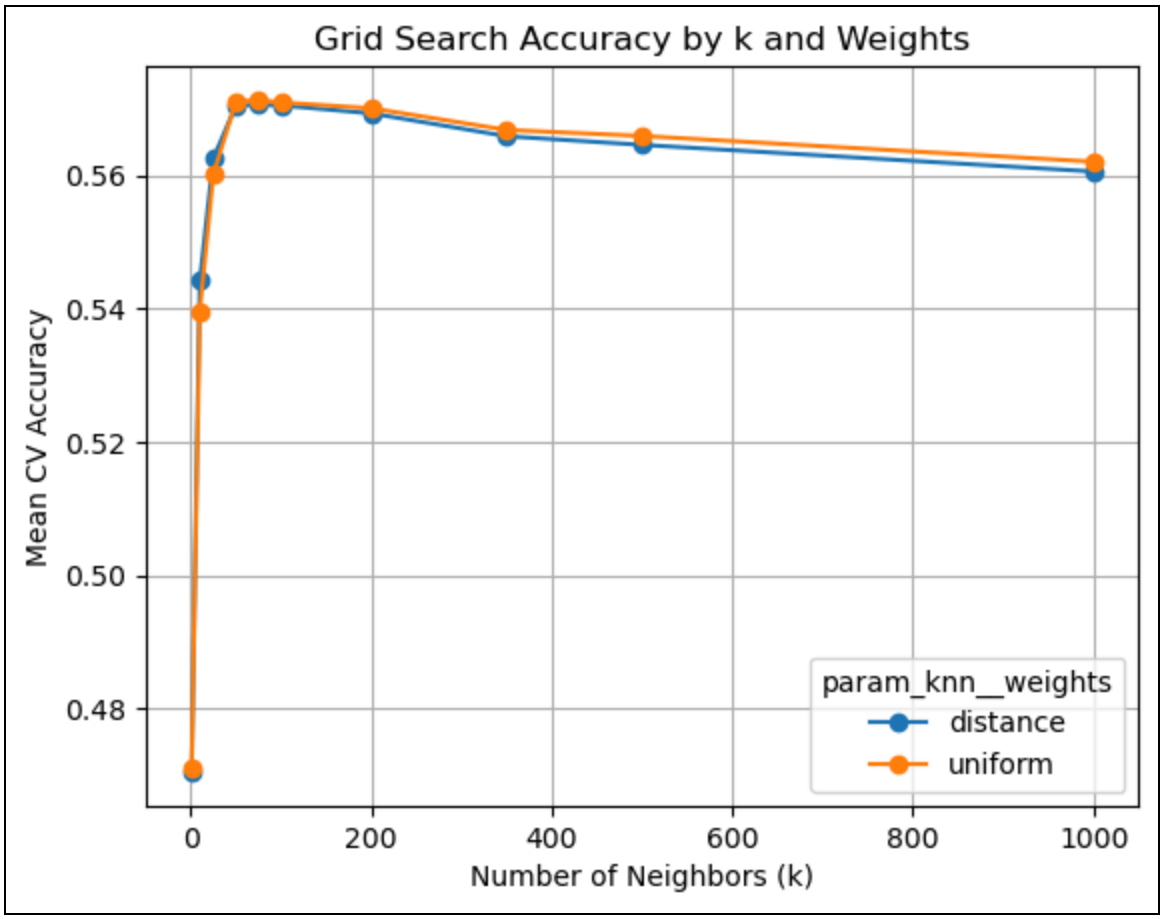

Figure V. kNN Classifier accuracy graph for different increasing numbers of k

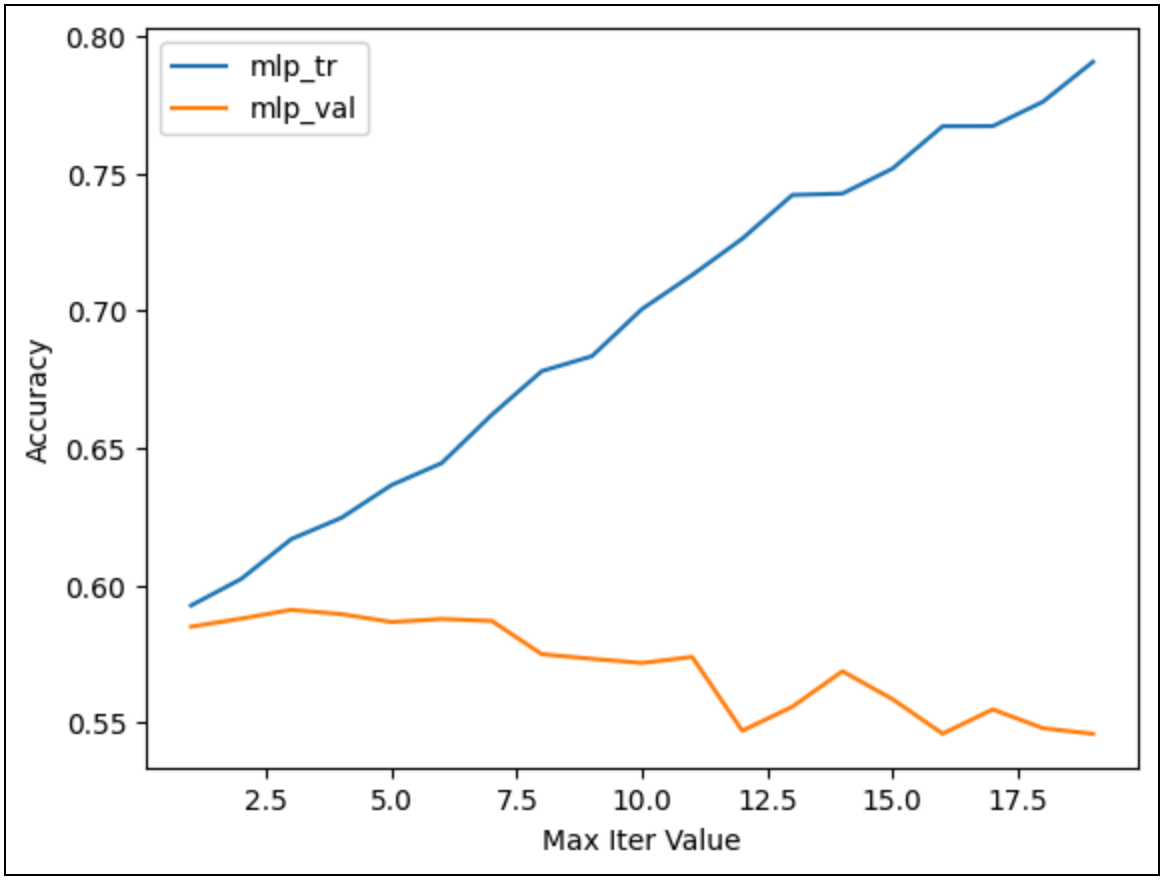

Figure VI. MLP Classifier accuracy graph for different values of max_iter

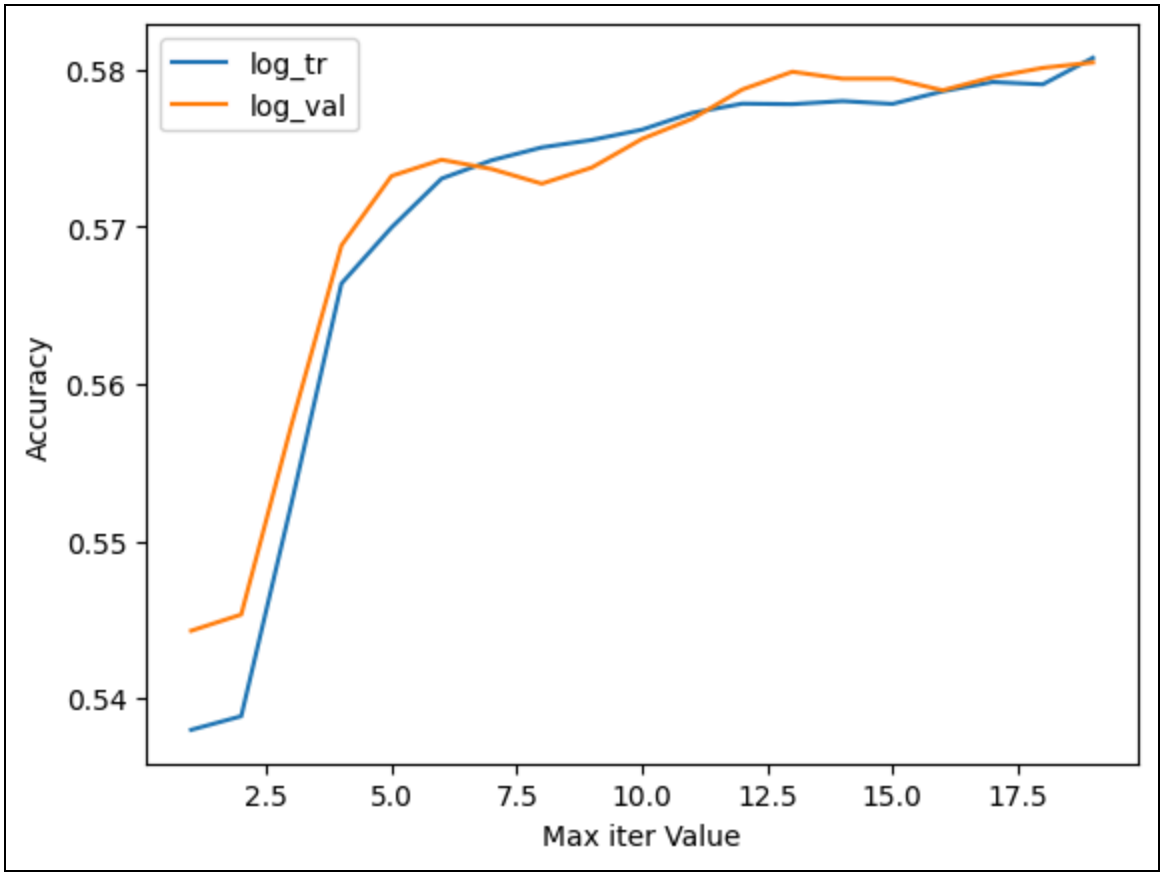

Figure VII. Logistic regression training vs validation accuracy across different max_iter values

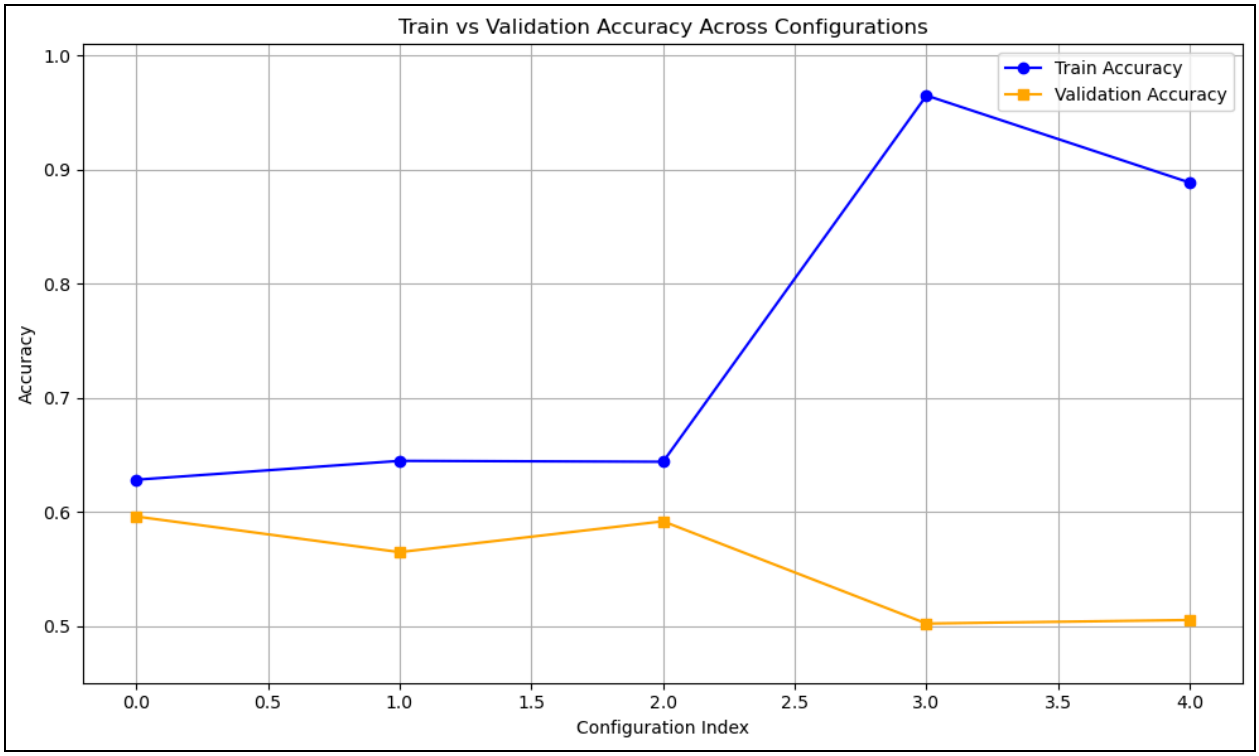

Figure VIII. Neural Network training vs Validation accuracy for different configurations of hyperparameters

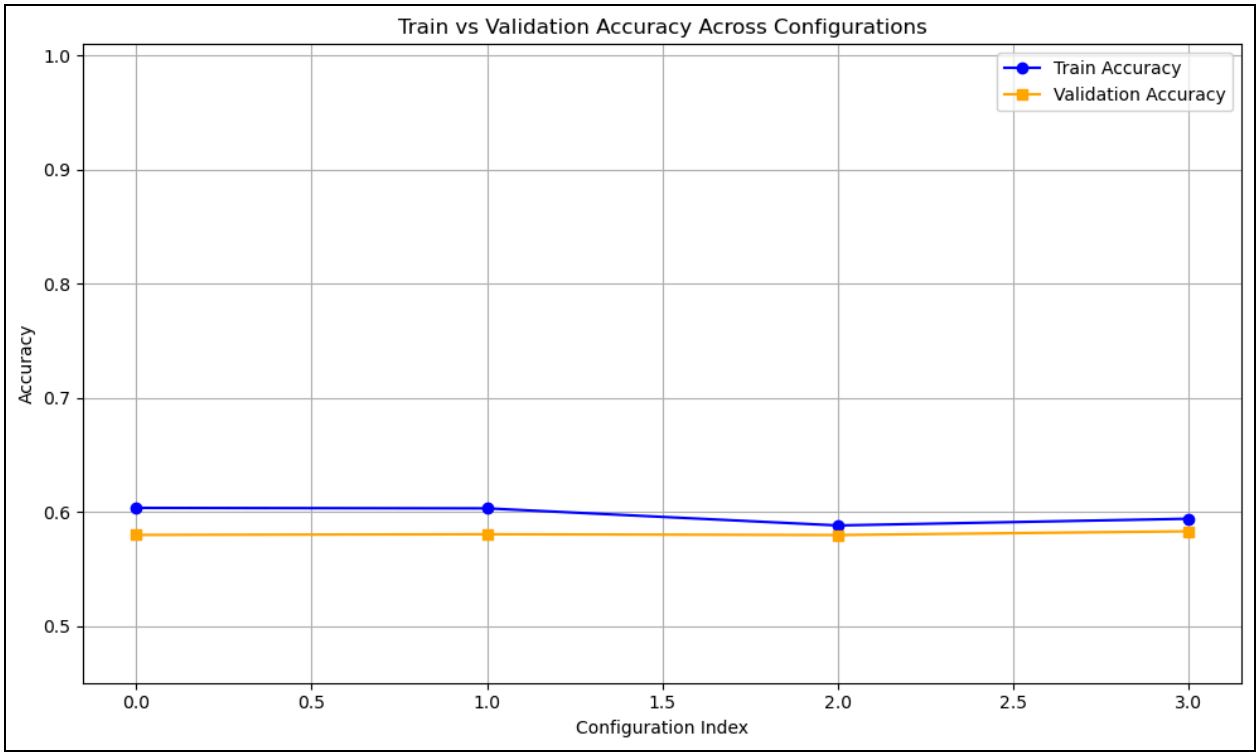

Figure IX. Logistic regression training vs validation accuracy for different configurations of hyperparameters

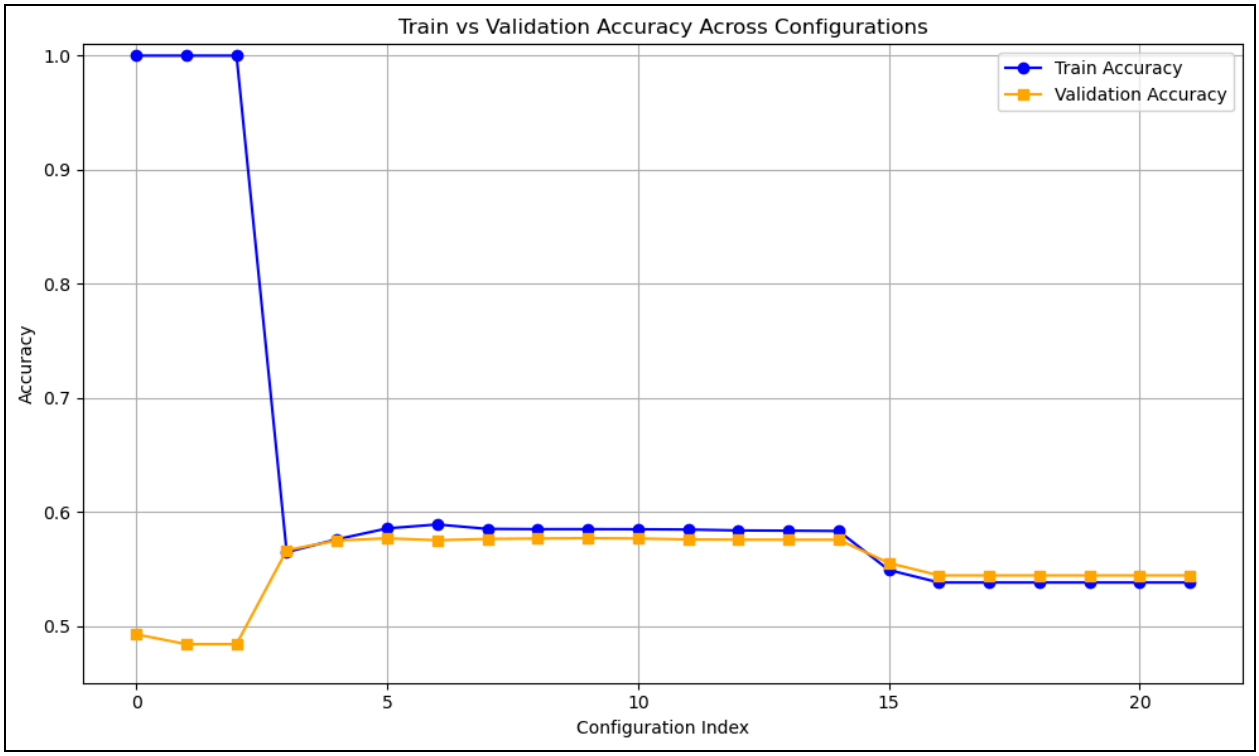

Figure X. Decision trees training vs validation accuracy for different configurations of hyperparameters