ML_groupR

April 6, 2025

1 Machine Learning Project #1

Work by:

- Bárbara Simões Neto - 202106176 - M:DS - 33%
- Beatriz Castro Silva - 202105723 - M:DS - 33%
- Rodrigo Couto - 202104696 - M:DS - 33%

2 Investigate method assumptions

2.1 Experimental methodology

We will begin by generating appropriate datasets tailored to each model, based on the assumptions and characteristics that align with how well they fit different types of data. Next, we will analyze the model performance statistically and by using techniques such as repeated cross-validation and learning curves to evaluate how well the dataset fits the model in practice.

If there's room for improvement, we will regenerate the dataset and fine-tune the model's hyperparameters to achieve a better fit. Finally, we will test each dataset across all models and empirically assess whether the assumptions about the best datasets for a given machine learning model hold true in practice.

2.2 Generate DataSets

Function generate_mixed_dataset Inputs:

n_samples: The total number of data points (rows) in the dataset.

n_features: The number of continuous (real-valued) features in the dataset.

n_categorical: The number of categorical features in the dataset. These features take on discrete values, typically represented as labels or categories.

n_ordinal: The number of ordinal features, where the values have a meaningful order (e.g., low, medium, high) but the differences between them are not consistent or defined.

n_integer: The number of integer-valued features, which take on whole number values.

n_classes: The number of distinct classes or categories in the target variable (predictive class). This corresponds to the classification task's output variable.

class_balance: A list of percentages indicating the distribution of each class in the dataset. The values should sum to 100% (1.0) and represent the relative frequency of each class.

noise: The level of noise in the predictive class. A value of 0 indicates no noise, and a value of 1 represents maximum noise, where the predictive class is randomly assigned.

dataset_type: Defines the distribution pattern of the classes. This parameter accepts one of the following options:

- "linear": Classes are distributed in a linear manner.
- "blobs": Classes are distributed in clusters (blobs).
- "moons": Classes are distributed in a moon-shaped pattern.
- "circles": Classes are distributed in concentric circles.

2.3 Plot Decision Boundaries

Plot plot_decision_boundary inputs:

model

X - feature values

y - target values

title - title of the plot

2.4 Plot Learning Curve

Plot plot_learning_curve inputs:

ax - ax to plot the visualization

estimator - estimator for the learning curve

X - feature values

y - target values

title - title of the plot

3 Linear models and Discriminant Analysis

3.1 Logistic Regression

Choose the adequate Dataset Logistic regression is a linear classification algorithm that performs particularly well when the relationship between the features and the target is approximately linear, and the dataset is relatively clean. It is known for its simplicity, interpretability, and efficiency, making it a strong baseline model for many classification problems.

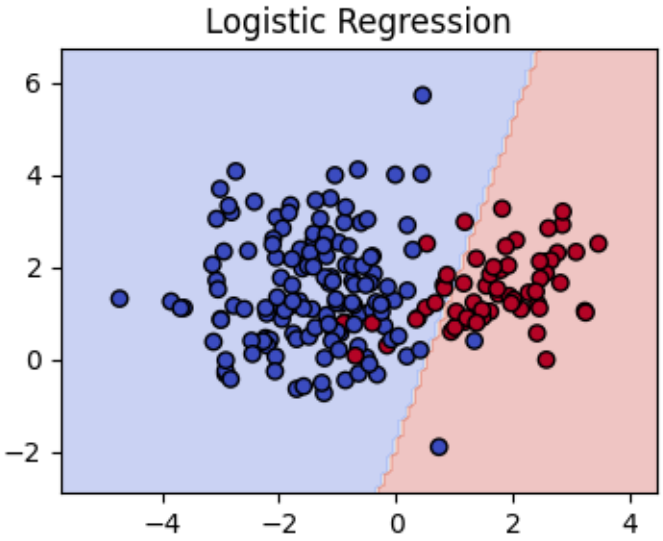
In this experiment, we specifically chose to generate a synthetic dataset using make_classification with parameters tailored to highlight the strengths of logistic regression and the limitations of other models like Linear Discriminant Analysis (LDA). We chose to keep the dataset minimal, with two featured and two classes to allow the visualization of decision bounds and keeping the analysis simple. The fact that both features are informative, ensures that they both contribute with meaningful information to the classification, making the data linearly separable (so that LogReg can find a linear decision boundary). A moderate class separation of 1.5 keeps the boundary easier

to lear for linear models, but the class imbalance introduces a challenge to other models (like LDA), to which LogReg is robust.

After generating the dataset, we intentionally added outliers. LDA is highly sensitive to outliers, unlike LogReg. While outliers can still affect it to some extent, the global optimization of the loss function makes logistic regression more stable.

	Feature 1	Feature 2	Class
0	-0.864557	-0.442229	0
1	1.812935	3.270534	1
2	2.840428	2.920996	1
3	-4.722217	1.322205	0
4	1.444105	0.977241	1

Model Bellow, we can see the decision boundary of LogReg for this dataset. Even though it is not perfect, the goal of this dataset is to highlight the differences between LogReg and LDA.



3.2 LDA

3.2.1 Choose the adequate Dataset

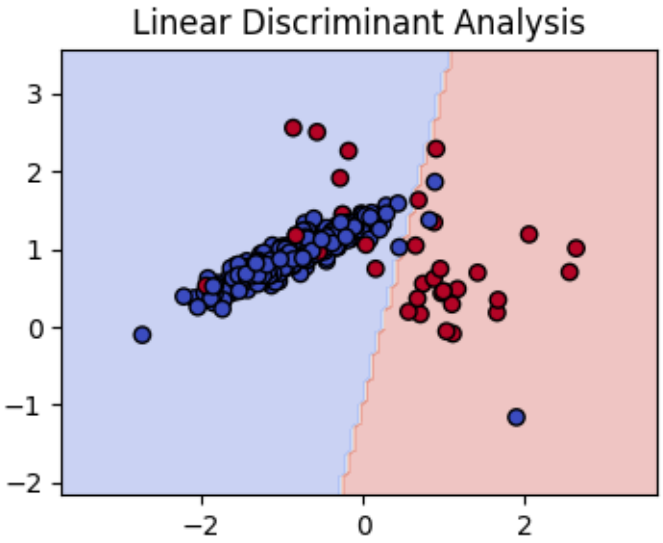
Linear Discriminant Analysis (LDA) is a powerful classification method based on probabilistic modeling and dimensionality reduction. It assumes that the features follow a multivariate normal distribution with shared covariance matrices across classes. When these assumptions hold approximately true, LDA can perform exceptionally well, even outperforming more flexible models.

To demonstrate LDA's strengths, we kept the 2 features (informative) and 2 classes dataset. To make the job harder to LogReg, we reduced class separability. As LDA models class distribution instead of sharp splits, it can handle the value of separation better. Noise was also added, by

setting flit_y to 10%. These parameters allow LDA to outperform LogReg, by overlapping classes and noise.

	Feature 1	Feature 2	Class
0	-1.408137	0.541881	0
1	2.052510	1.190190	1
2	-0.879469	0.905495	0
3	-0.822701	1.184172	0
4	-1.284593	0.729383	0

3.2.2 Model



3.3 QDA

3.3.1 Choose Adequate Dataset

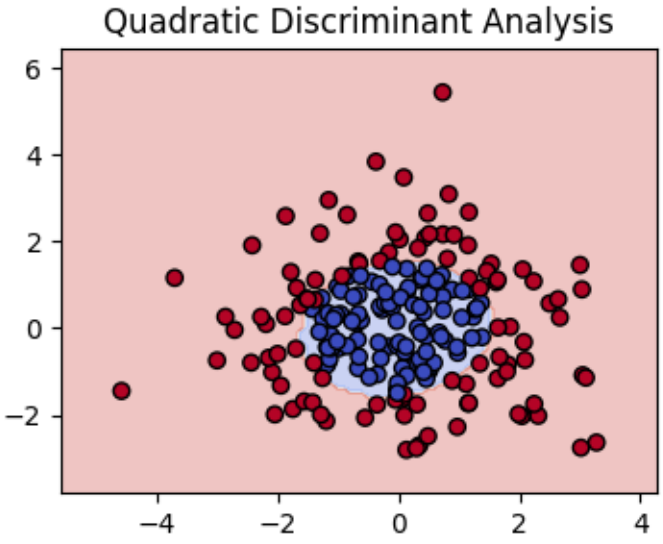
Quadratic Discriminant Analysis (QDA) is a more flexible extension of Linear Discriminant Analysis (LDA) that does not assume equal covariance matrices across classes. This flexibility allows QDA to model more complex, nonlinear decision boundaries, making it well-suited for datasets where class distributions are not spherical or have different variances.

For this model, we generated a dataset by using make_gaussian_quantiles, that introduces diferente covariances across classes, as well as non-linear separability. We kept the classes centered around the origin (with 0 mean) but set the covariance to 2, making the classes more dispersed.

	0	1	0
0	-0.864557	-0.442229	0
1	1.812935	3.270534	1
2	2.840428	2.920996	1
3	-4.722217	1.322205	0

```
4  1.444105  0.977241  1
```

3.3.2 Model



3.4 Decision Tree model

3.4.1 Choose the adequate DataSet

Decision trees are particularly effective when dealing with categorical and numerical data, making them an excellent choice for datasets that contain a mix of variable types. They are especially useful for classification problems, whether binary or multiclass. Decision trees efficiently capture interactions between features, allowing them to model complex relationships without requiring additional transformations.

Another key advantage is their ability to handle nonlinear data, as they split the dataset based on thresholds that naturally define decision boundaries. Additionally, decision trees can process missing values without the need for imputation and offer feature importance insights, helping to identify the most influential attributes. However, while they perform well on small to medium-sized datasets, they may struggle with very large datasets due to computational inefficiency and risk of overfitting.

Since decision trees can struggle with imbalanced datasets, where the majority class dominates, they should take balenced dataset to mitigate bias.

When the dataset has higher noise tree complexity increases, leading to overfitting and poor generalization.

By generating a dataset with the described characteristics, we can effectively demonstrate that decision trees outperform other machine learning models in such scenarios, as they naturally leverage structure and patterns that other algorithms may struggle with.

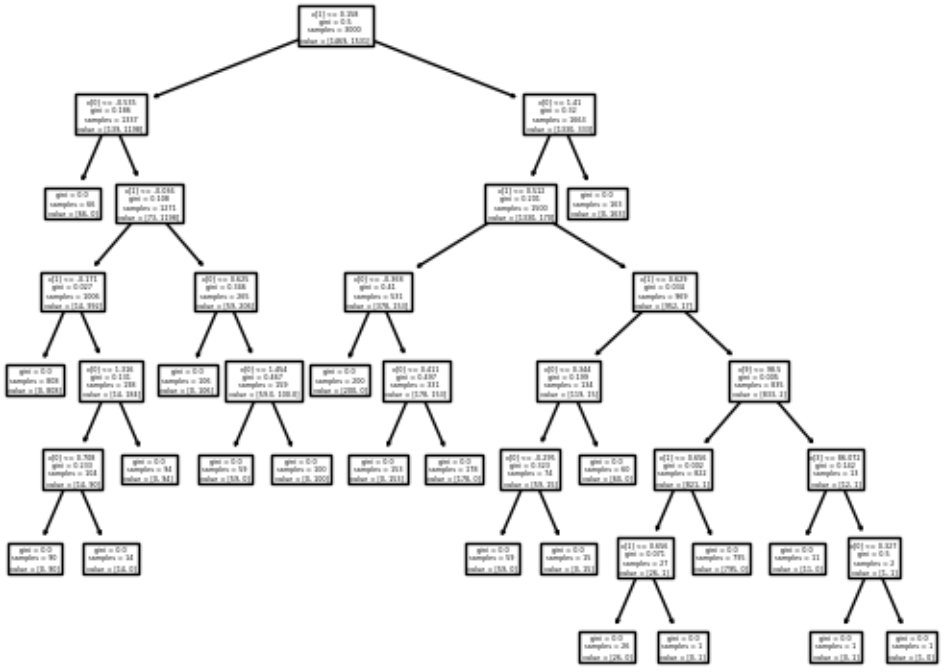
	Continuous_1	Continuous_2	Continuous_3	Continuous_4	Continuous_5	\
0	-0.605843	0.657092	77.770241	23.754122	82.427853	
1	1.566583	-0.070554	96.574920	97.260111	45.344925	
2	1.578243	-0.249173	60.904246	77.552651	64.161334	
3	0.240217	0.940749	72.201823	3.503652	29.844947	
4	0.129358	0.106276	5.851249	85.706094	37.285403	

	Ordinal_1	Ordinal_2	Integer_1	Integer_2	Integer_3	Integer_4	\
0	1	5	81	43	96	20	
1	2	2	43	55	45	10	
2	4	5	5	45	58	92	
3	5	1	33	81	25	46	
4	4	3	66	67	68	87	

	Integer_5	Target
0	10	0
1	98	1
2	34	1
3	48	0
4	38	1

Training and Test set by spliting the original DS.

3.4.2 Full Tree



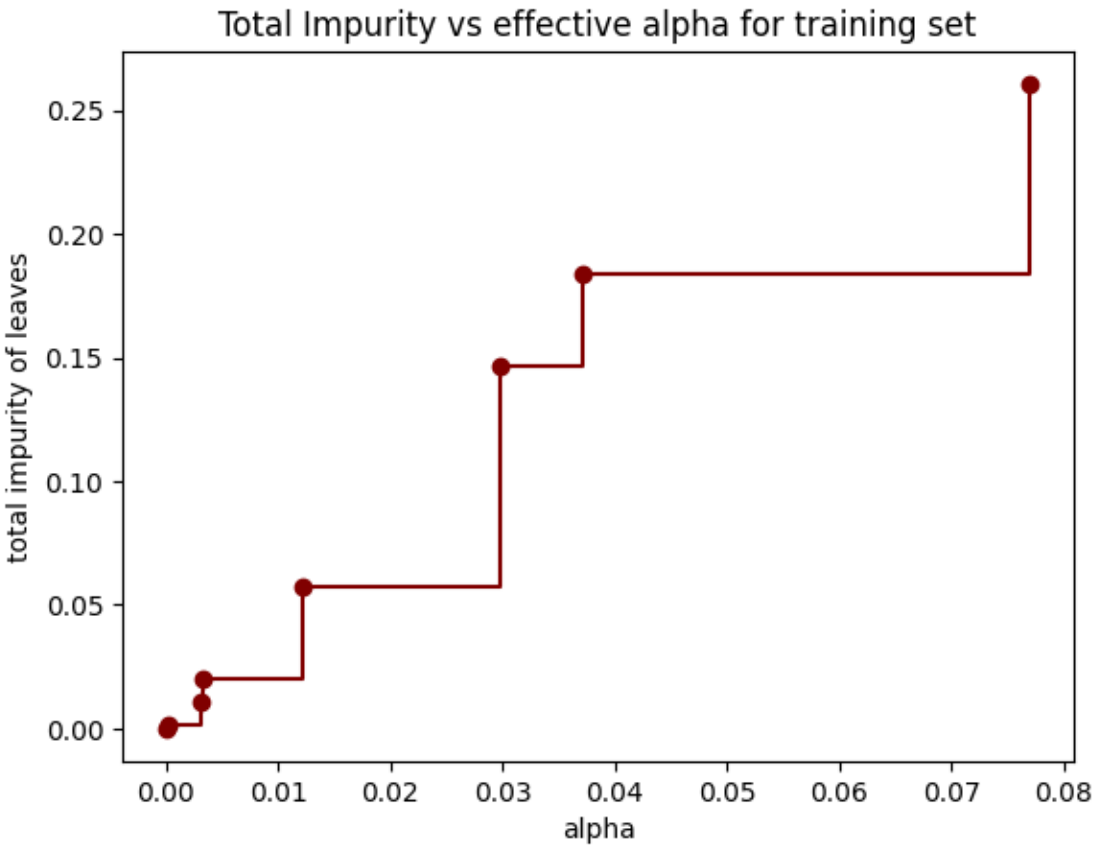
3.4.3 Pruned Trees

Get best pruning One limitation of decision trees is their tendency to overfit, capturing noise rather than meaningful patterns, especially when the tree grows too deep. To counteract this, we can apply pruning techniques.

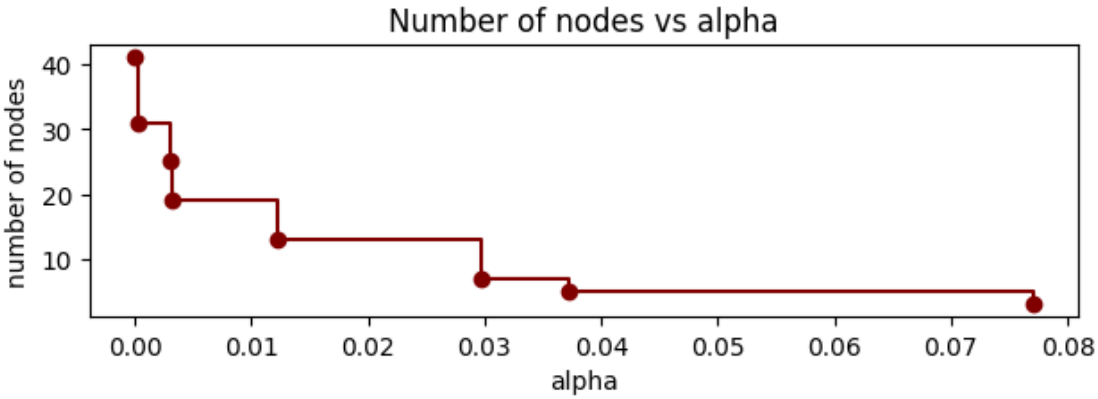
Pre-pruning (early stopping) restricts tree growth using constraints like maximum depth or minimum samples per leaf, ensuring that only meaningful splits occur. Post-pruning, on the other hand, involves growing the tree fully and then removing less important branches based on validation performance. Both methods improve generalization, preventing the model from memorizing data-specific noise.

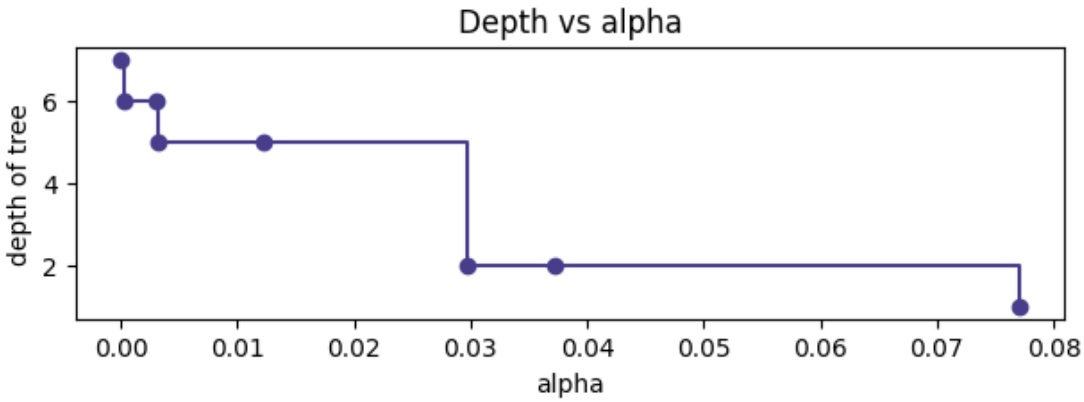
We can empirically demonstrate that pruned decision trees generalize better, maintaining high accuracy while avoiding overfitting. This will further reinforce the argument that decision trees are well-suited for datasets with structured, mixed-variable data. For this study we will use Minimal Cost-Complexity Pruning, an algorithm that is parameterized by alfa >=0 - the complexity parameter. The complexity parameter is used to define the cost-complexity measure of a given tree.

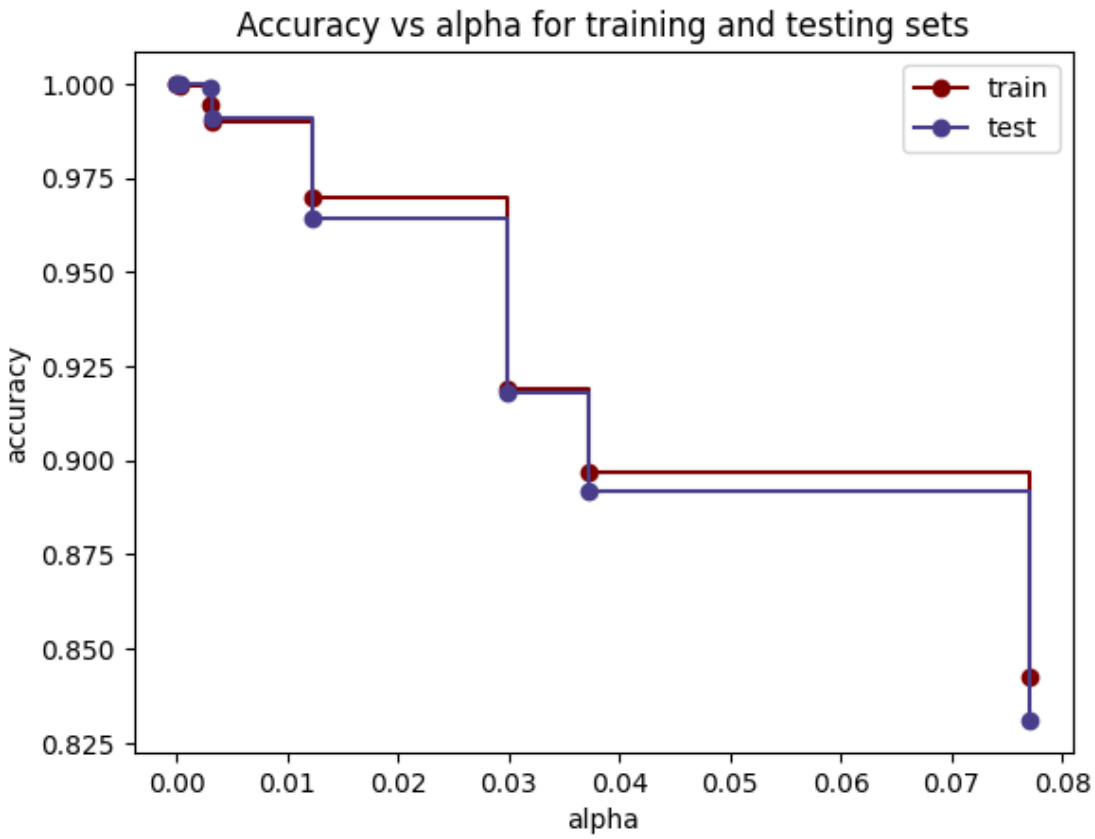
```
Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')
```



```
[0.         0.00026603 0.00306782 0.00326812 0.01222268 0.0297848
 0.03716117 0.07705771 0.23920718]
```

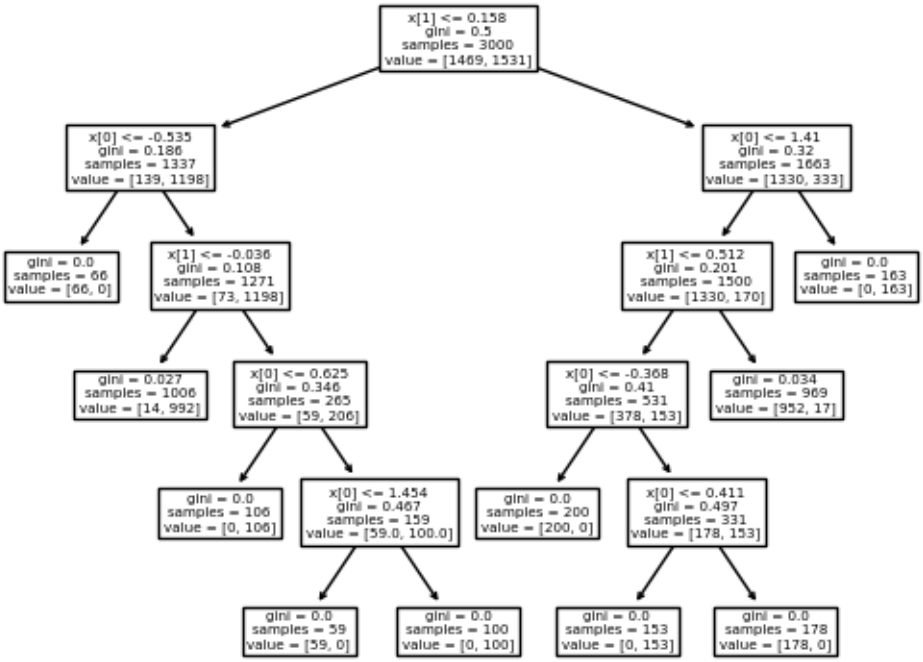




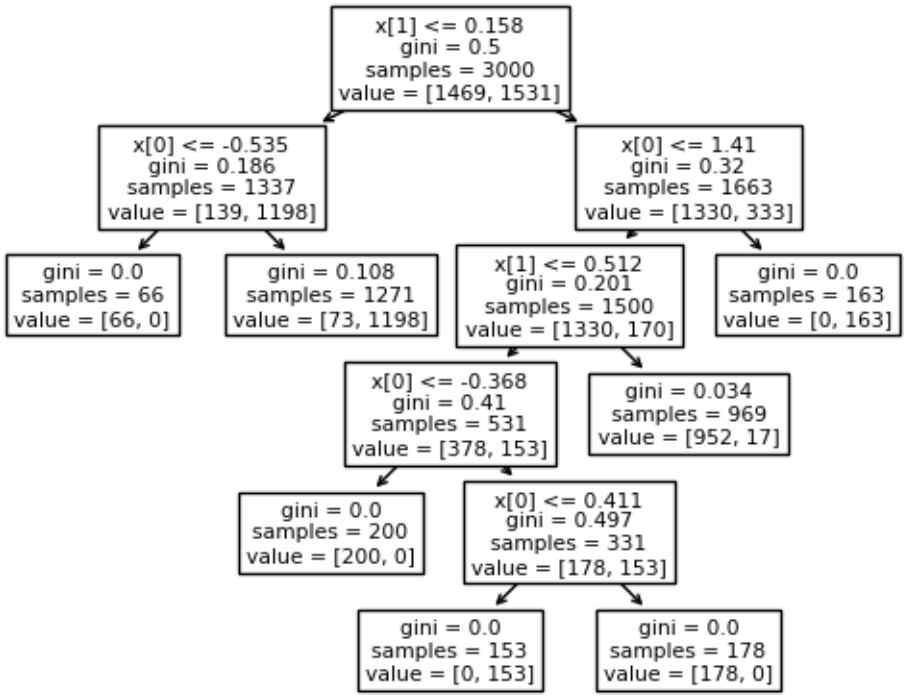


We analyzed how total impurity, accuracy, depth, and node count vary with increasing alpha. From these trends, we identified optimal alpha values for pruning—those balancing model simplicity and performance—making them the most promising candidates for further testing and evaluation: 0.025 and 0.0010.

Pruned tree #1



Pruned tree #2



3.5 SVM model

The SVM algorithm seeks to find the optimal separating hyperplane between classes by solving a constrained optimization problem that maximizes the margin.

Key Formula (Linear SVM)

The decision boundary is:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Minimize:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum \xi_i$$

Subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$$

Kernel Trick

For non-linear data, replace dot products with kernel, for example, the RBF Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

3.5.1 Linear

Linear SVMs work best with specific types of datasets. They are ideal for linearly separable data, where classes can be clearly divided by a straight line. They also perform well with high-dimensional data, like text or datasets with many features, as they focus on finding a linear decision boundary. Numerical data (continuous or integer values) works best, while categorical data needs to be transformed (e.g., using one-hot encoding). However, linear SVMs have limitations. They struggle with large datasets due to high computational costs and memory usage. They also perform poorly on imbalanced datasets, often favoring the majority class, and on noisy data where classes overlap significantly. For multiclass problems, additional techniques like One-vs-Rest or One-vs-One are needed, as linear SVMs are designed for binary classification. In summary, linear SVMs are best suited for small to medium-sized, high-dimensional, linearly separable datasets with balanced classes and a acceptable noise.

```
Best Dataset for Linear SVM:
   Continuous_1  Continuous_2  Continuous_3  Continuous_4  Continuous_5  \
0      0.428625     -6.533283      7.066844     -8.627832      2.355361
1      3.095479     -2.386785      1.033195      5.420974      6.689510
```

```
2      3.200855     -4.313051      9.249571     -0.010385     -0.488567
3    -10.039503     -4.054238    -10.610494     -2.367866     11.968597
4    -10.465234     -3.202989     -5.414105     -1.313470      1.759944

   Continuous_6  Continuous_7  Continuous_8  Continuous_9  Continuous_10  …  \
0     -6.369528      2.658301      5.686821      8.007498       1.588096  …
1      8.019037     -7.700922    -10.420246     -1.429404       3.627678  …
2     -4.709072     -5.311260     -8.006093     -3.091417       8.088402  …
3      2.502079      9.374987     -3.035388      1.163737       0.647769  …
4      2.670421      5.437371     -2.591946     14.536810       6.671950  …

   Integer_12  Integer_13  Integer_14  Integer_15  Integer_16  Integer_17  \
0          77          39          40          85          10          22
1          20          46          72          52           8          73
2          33          90          16          42          58          50
3          84          48          70          80          83          48
4          82          61          31          29          28          48

   Integer_18  Integer_19  Integer_20  Target
0           0          45          20       1
1          51          56          25       0
2          53          23          24       0
3          19          85          91       0
4          44          92          29       0

[5 rows x 96 columns]
```

3.5.2 SVM RBF

The SVM RBF (Radial Basis Function) kernel excels with non-linear, complex datasets where classes are not linearly separable, capturing intricate patterns through flexible decision boundaries. It handles noisy or overlapping data better than linear SVMs but requires careful tuning of hyperparameters like gamma. While effective for low to moderate-dimensional numerical data, its performance degrades with high-dimensional features due to the "curse of dimensionality," as distances between points become less meaningful. For large datasets, imbalanced classes, and multiclass problems, its behavior aligns with linear SVMs, though computational costs rise significantly with increased feature count.

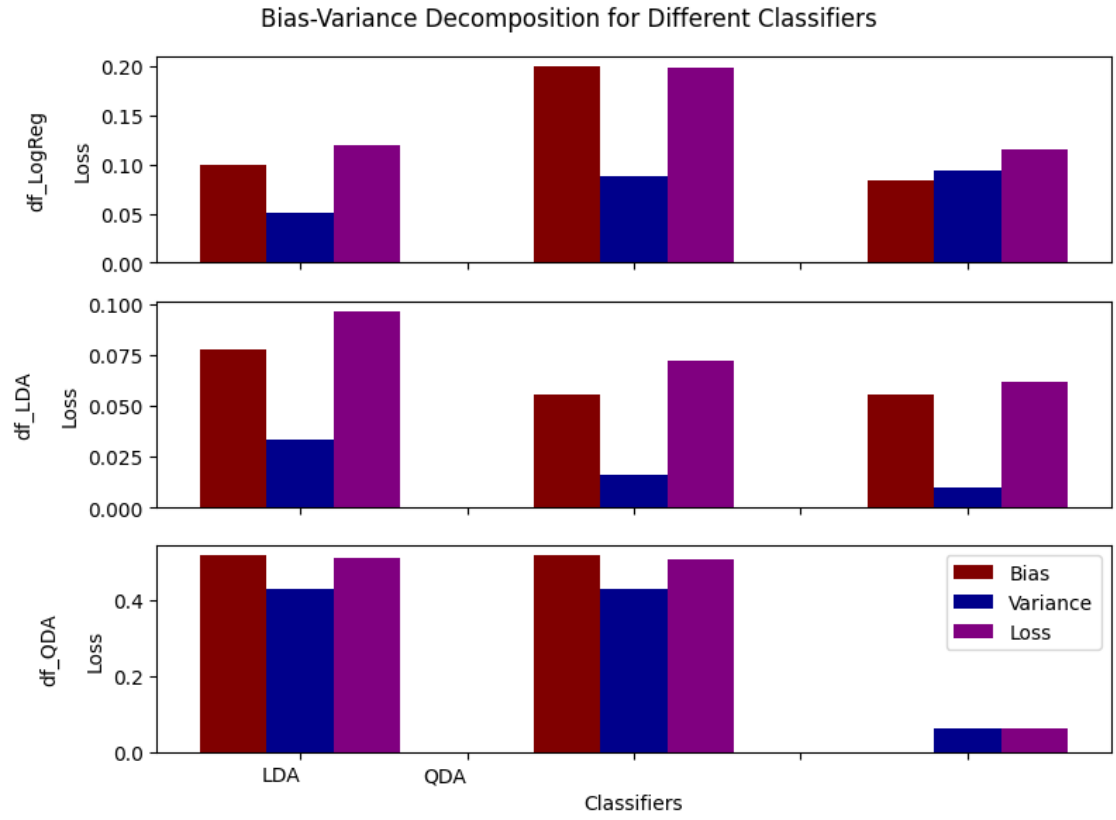
```
Best DataSet for SVM RBF:
   Continuous_1  Continuous_2  Continuous_3  Continuous_4  Continuous_5  \
0      0.380471     -0.500328     37.454012     95.071431     73.199394
1     -0.649700      0.832445      5.808361     86.617615     60.111501
2     -0.403240      0.741361     83.244264     21.233911     18.182497
3      0.834763     -0.690383     43.194502     29.122914     61.185289
4     -0.094453      0.726726     45.606998     78.517596     19.967378

   Continuous_6  Continuous_7  Continuous_8  Target
```

0	59.865848	15.601864	15.599452	1
1	70.807258	2.058449	96.990985	0
2	18.340451	30.424224	52.475643	1
3	13.949386	29.214465	36.636184	0
4	51.423444	59.241457	4.645041	1

4 Bias and variance decomposition

4.1 Evaluation LDA, QDA, LogReg

4.1.1 Bias, variance and total loss (LogReg, LDA, QDA)



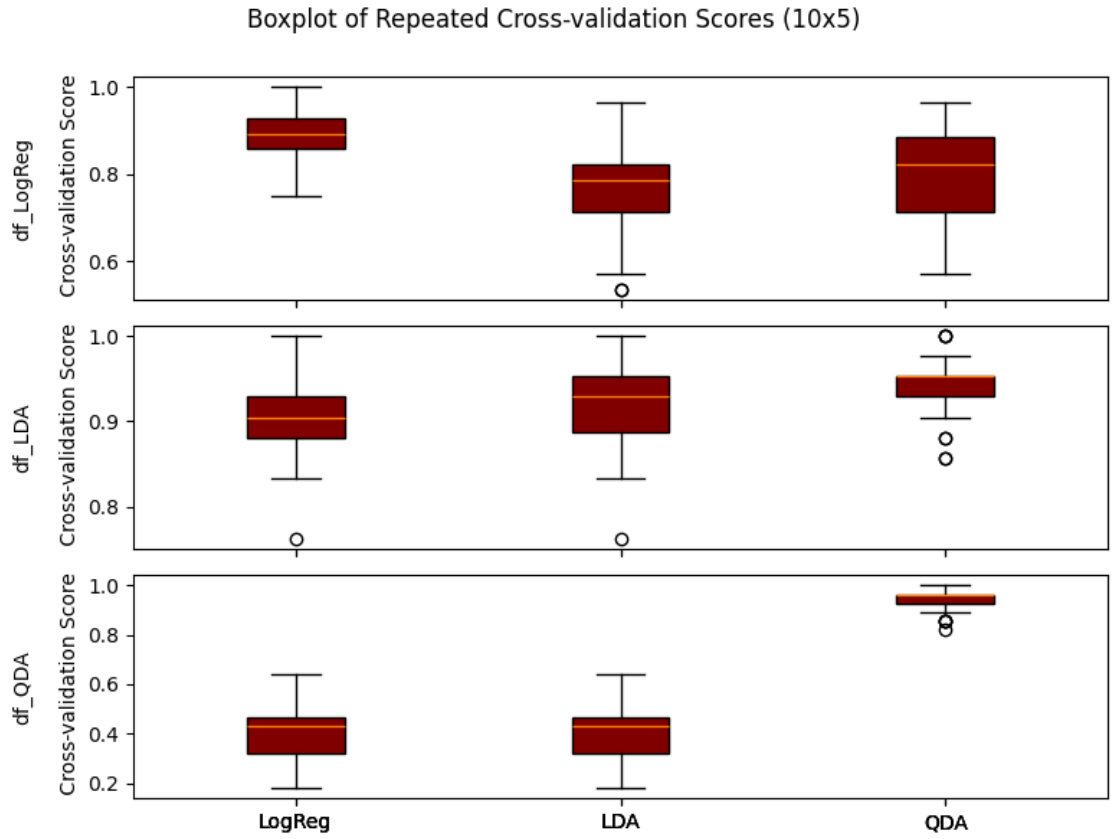
Dataset	Classifier	Loss	Bias	Variance
df_LogReg	LogReg	0.1197	0.1000	0.0513
df_LogReg	LDA	0.1978	0.2000	0.0878
df_LogReg	QDA	0.1152	0.0833	0.0939
df_LDA	LogReg	0.0965	0.0778	0.0331
df_LDA	LDA	0.0718	0.0556	0.0162
df_LDA	QDA	0.0616	0.0556	0.0098
df_QDA	LogReg	0.5089	0.5167	0.4269
df_QDA	LDA	0.5083	0.5167	0.4273
df_QDA	QDA	0.0635	0.0000	0.0635

For the df_LogReg dataset, LogReg performs reasonably well with a moderate Loss, low Bias, and moderate Variance. LDA and QDA show higher bias and variance, leading to higher loss. However, QDA has slightly higher variance, indicating it is more sensitive to the data.

For the second dataset, as expected, LDA outperforms the other two models having the lowest loss, bias and variance. Even tho QDA has a low loss, it has a slightly higher bias.

Both Logistic Regression and LDA struggle significantly with the df_QDA, showing high Bias and Variance, leading to a very high Loss. In contrast, QDA excels with the lowest Loss, zero Bias, and reasonable Variance, indicating it is the best model for this dataset by far, fitting the data well and generalizing effectively.

4.1.2 Repeated Cross-Validation (LogReg, LDA, QDA)



Above, we see that for almost all datasets, the model that performs best is the one to which they were created.

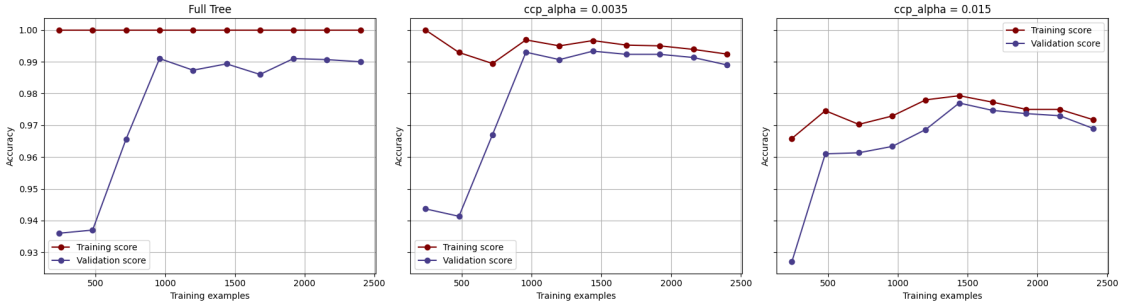
The main exception could be the LDA dataset, where the boxplot of LogReg comes only a little bit bellow the LDA one. Besides that, QDA also seems to perform well on this dataset. We can take more conclusions when visualizing the decision bounds.

For the QDA dataset, there's an obvious difference between QDA and the other two models, with LogReg and LDA behaving pretty similar.

4.2 Evaluation Decision Trees

In this chapter, we will evaluate both the full decision tree and the two pruned versions. The goal is not only to compare their performance but also to determine which tree better fits the dataset while demonstrating stronger generalization capabilities.

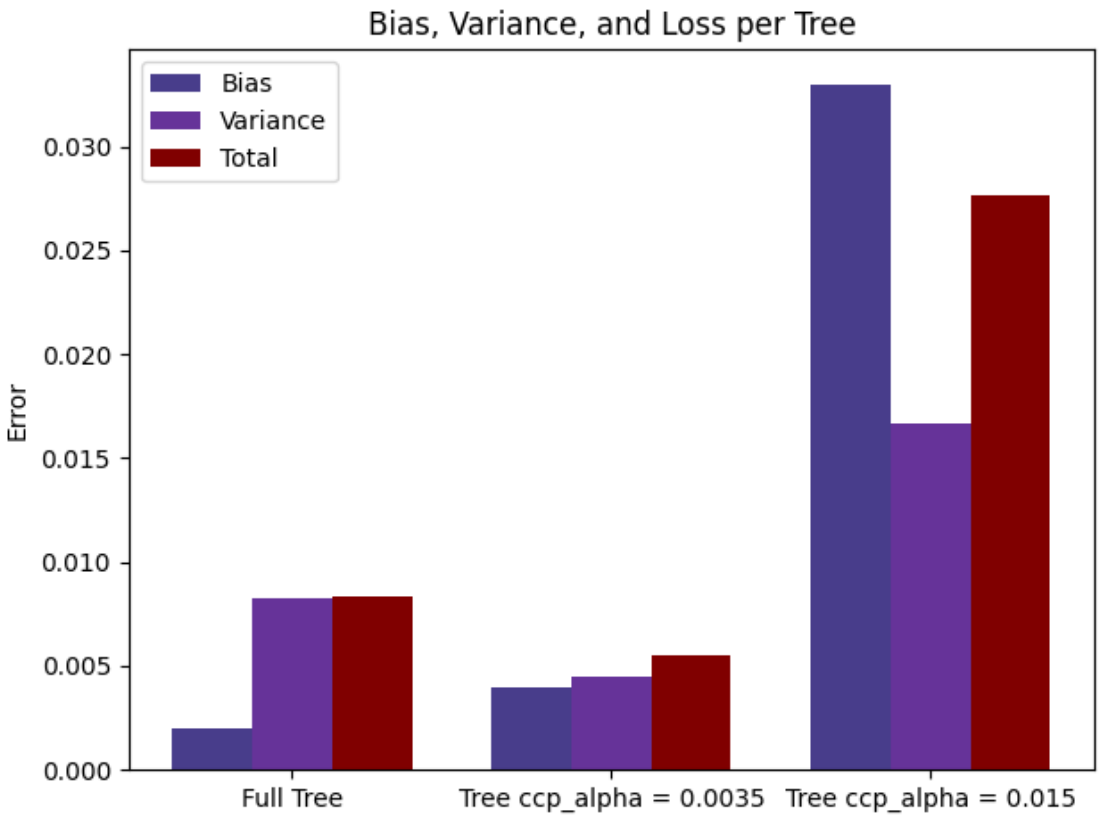
4.2.1 Learning curves



The full decision tree achieves perfect training accuracy but exhibits slight overfitting. The pruned tree with `ccp_alpha=0.01` strikes a good balance, showing similar performance on both the training and validation sets, indicating a better generalization. However, the tree with `ccp_alpha=0.02` becomes unstable and demonstrates reduced accuracy, even with additional training examples, suggesting excessive pruning.

4.2.2 Bias, Variance and Total Loss

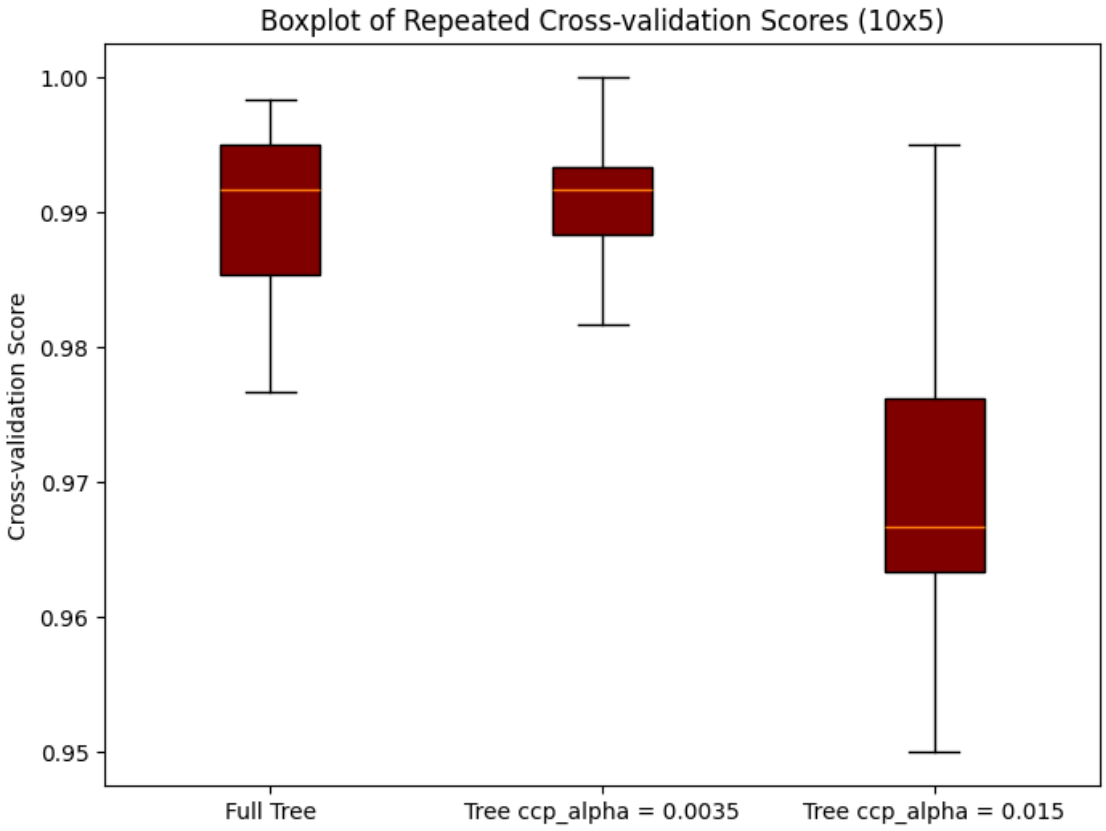
Calculate the Loss, Bias and Variance for all trees.



The full tree exhibits low bias but high variance, indicating overfitting. The tree with `ccp_alpha=0.0035` provides the best bias-variance trade-off, with balanced values for both. In contrast, the tree with `ccp_alpha=0.015` shows high bias and low variance, indicating underfitting, leading to the highest total loss among the three models.

4.2.3 Repeated Cross-Validation Results

```
Text(0.5, 1.0, 'Boxplot of Repeated Cross-validation Scores (10x5)')
```

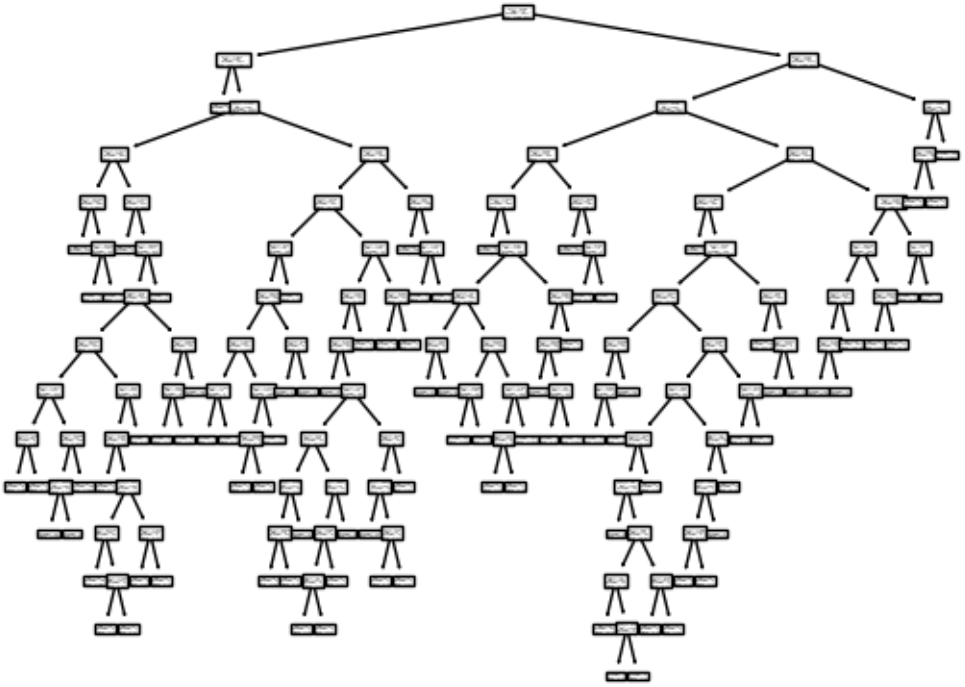


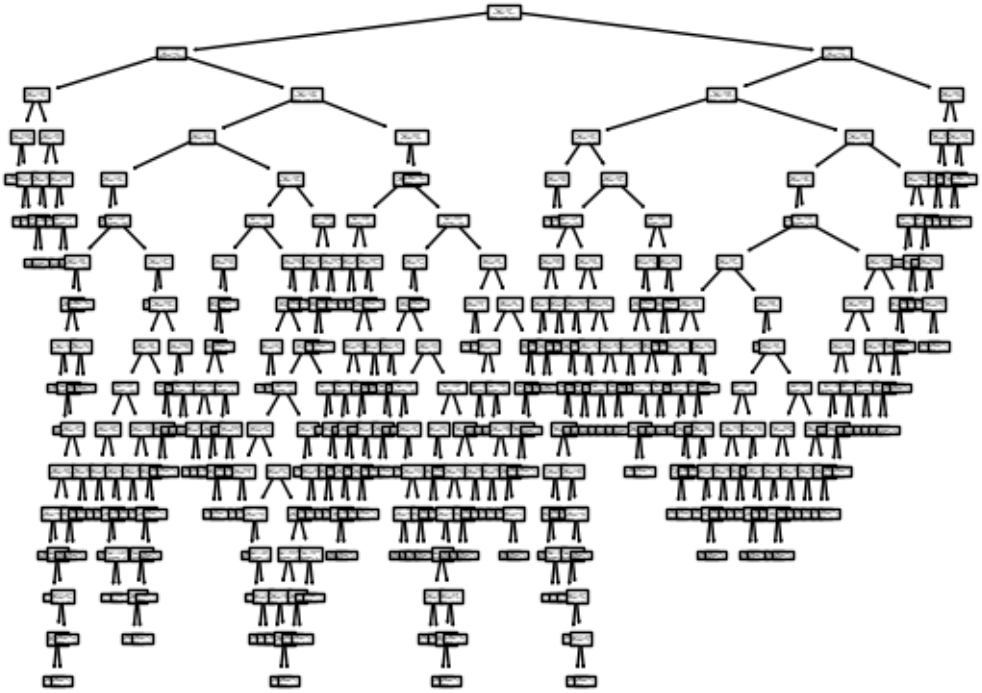
The tree with `ccp_alpha=0.0035` demonstrates high accuracy and low variance across folds, indicating a stable model. The full tree shows higher variance with a long lower tail, suggesting overfitting. The tree with `ccp_alpha=0.015` has a lower median accuracy and higher variance, indicating instability and underfitting.

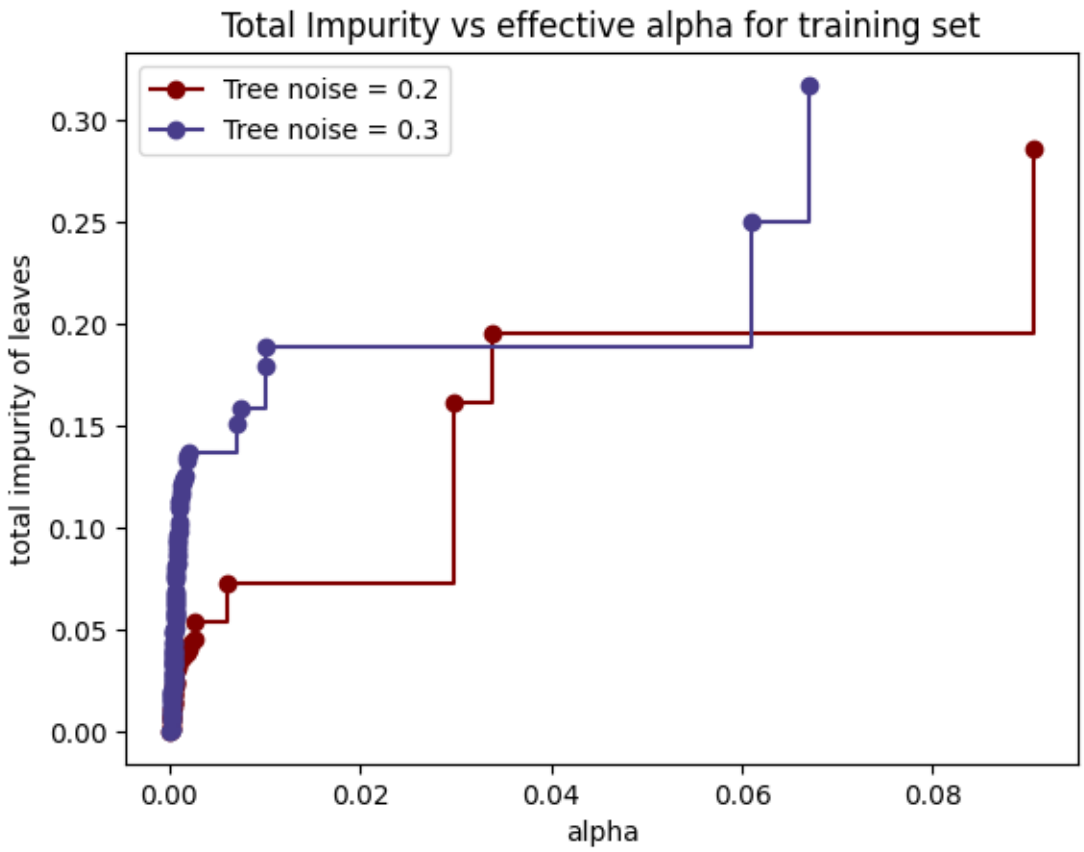
In conclusion, Tree with `ccp_alpha=0.0035` is the best option of pruning for this tree when adapted to this dataset.

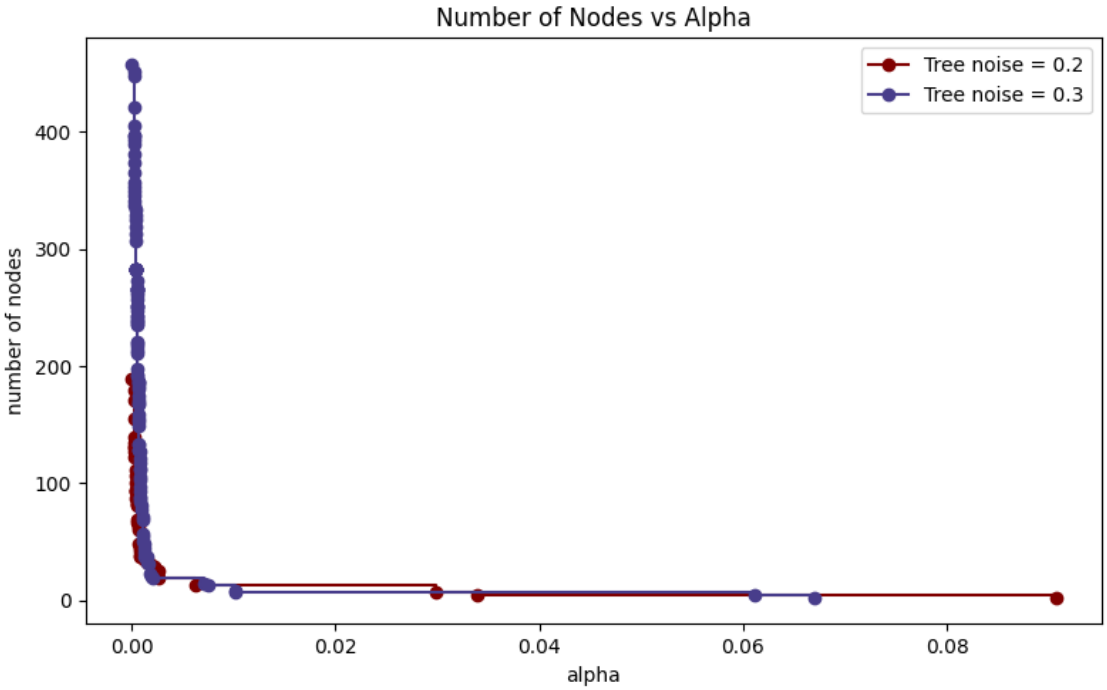
4.2.4 Performance with varying DataSet noise

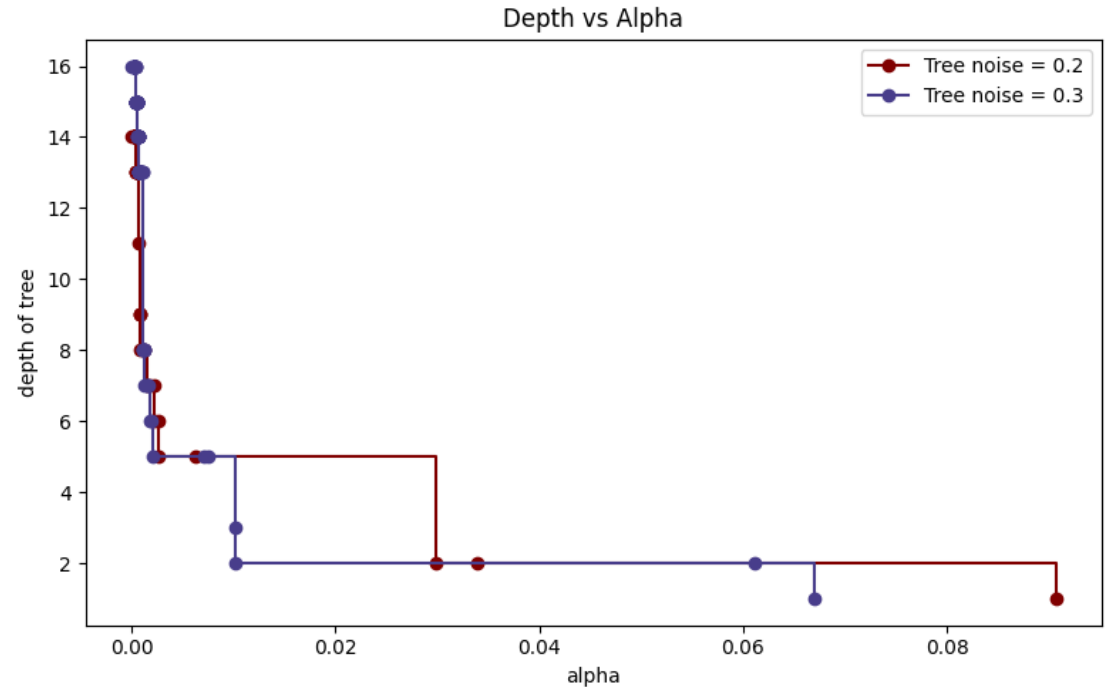
The original data set has noise = 0.1, now we'll create two DS with noise = 0.2 aind noise = 0.3.

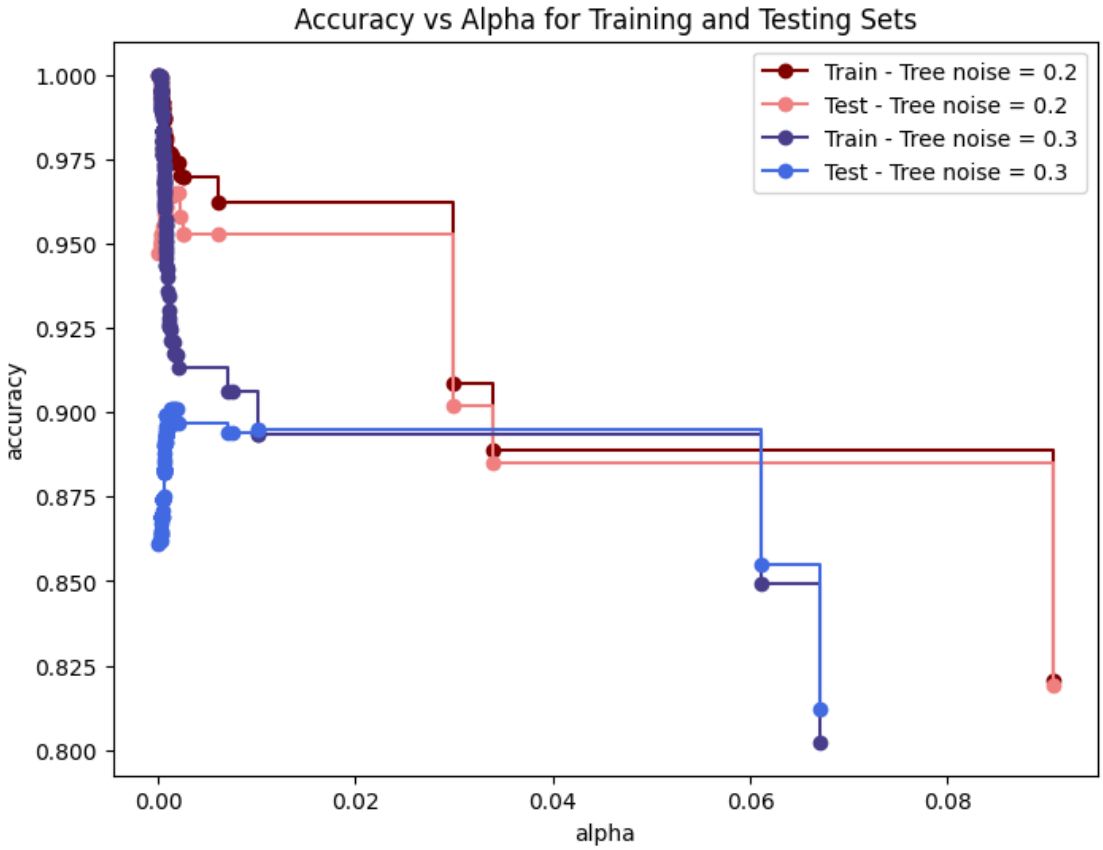





Total Impurity vs effective alpha for training set
Tree noise = 0.2
Tree noise = 0.3
total impurity of leaves
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.00
0.02
0.04
0.06
0.08
alpha


Number of Nodes vs Alpha
Tree noise = 0.2
Tree noise = 0.3
number of nodes
alpha
0
100
200
300
400
0.00
0.02
0.04
0.06
0.08


Depth vs Alpha
Tree noise = 0.2
Tree noise = 0.3
depth of tree
alpha
2
4
6
8
10
12
14
16
0.00
0.02
0.04
0.06
0.08



The optimal pruning for Tree with noise = 0.1 is 0.0035

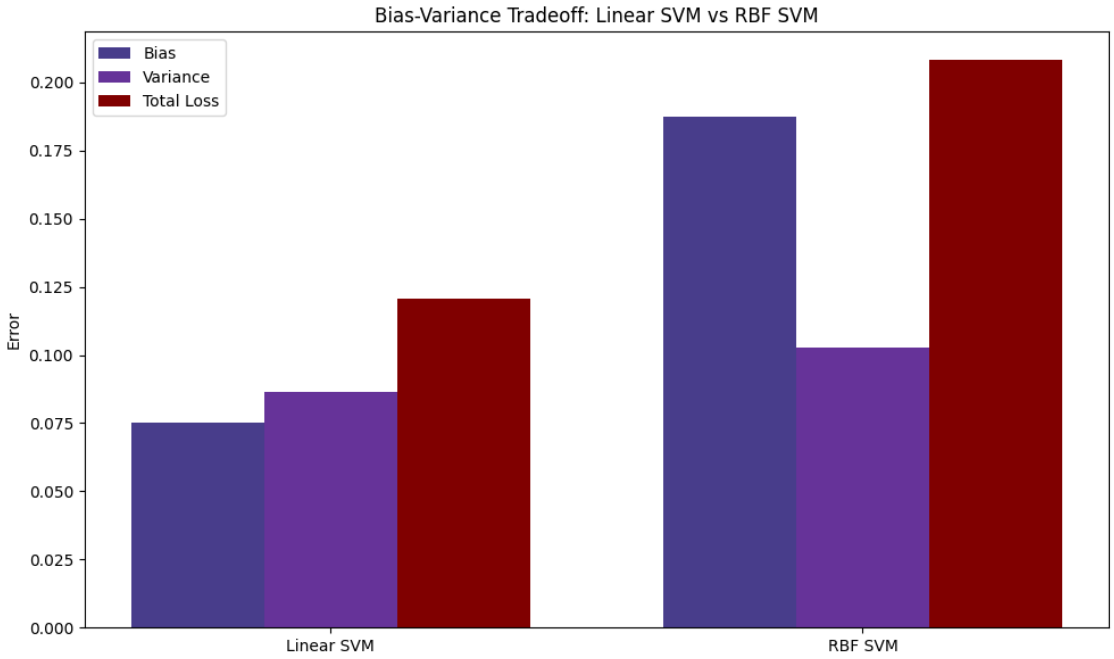
The optimal pruning for Tree with noise = 0.2 is 0.00619748

The optimal pruning for Tree with noise = 0.3 is 0.00749248

Higher noise increases tree complexity, leading to overfitting and poor generalization. Pruning helps by reducing unnecessary splits, improving performance on unseen data. However, even with pruning, decision trees struggle with noisy data compared to cleaner datasets. A higher ccp_alpha is expected with increased noise to counteract overfitting. Proper pruning enhances generalization but cannot fully mitigate noise effects, making noisy data not adquate for Decision Trees.

4.3 Evaluation SVM

4.3.1 Bias, Variance and Total loss



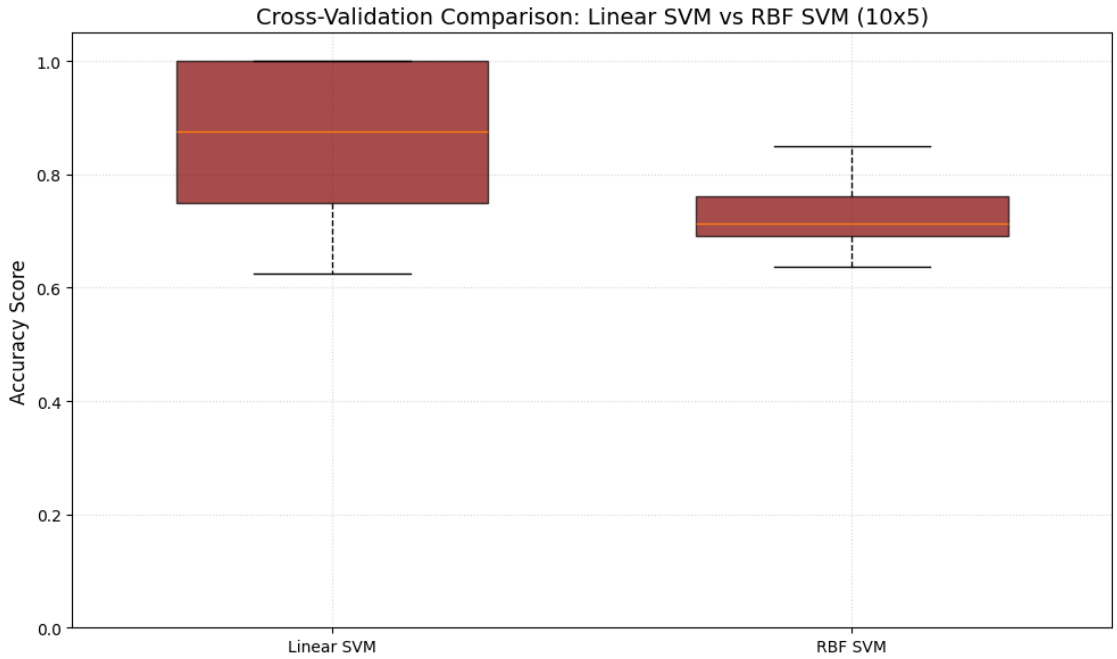

```
Model Comparison:

Linear SVM:
Bias: 0.0750
Variance: 0.0866
Total Loss: 0.1209

RBF SVM:
Bias: 0.1875
Variance: 0.1027
Total Loss: 0.2082
```

The results demonstrate that both Linear SVM and RBF SVM models perform effectively on their respective custom-designed datasets, validating our implementation approach. For the linearly separable dataset, the Linear SVM achieves strong performance with consistent predictions (low variance) and acceptable error rates, confirming its suitability for this data structure. The model's stable behavior across multiple test runs indicates robust handling of the linear patterns we engineered into the dataset.

The RBF kernel shows equally compelling results on the non-linear moons dataset, successfully capturing the complex decision boundaries with superior accuracy. While exhibiting slightly higher variance as expected from its flexible nature, the model maintains good generalization capabilities, proving its effectiveness on non-linear patterns. The total error remains low, confirming the RBF kernel's ability to handle with noise and complexity built into this dataset.

This validates our implementation and shows each model works as intended for its specific dataset type.

4.3.2 Repeated Cross-Validation Results



The cross-validation results confirm both SVM models perform well on their respective datasets. The Linear SVM shows stable, consistent accuracy on the linear dataset, with tight score clustering demonstrating reliable performance for simple patterns. The RBF SVM achieves good accuracy on the non-linear dataset, with slightly wider score variations reflecting its ability to handle more complex relationships. Each model's validation scores align with their expected behavior for the specific data patterns they were designed to address, verifying their proper implementation and effectiveness. The repeated 10x5 fold validation provides robust evidence that both models work as intended on their target datasets.

5 Ensemble

5.1 Boosters

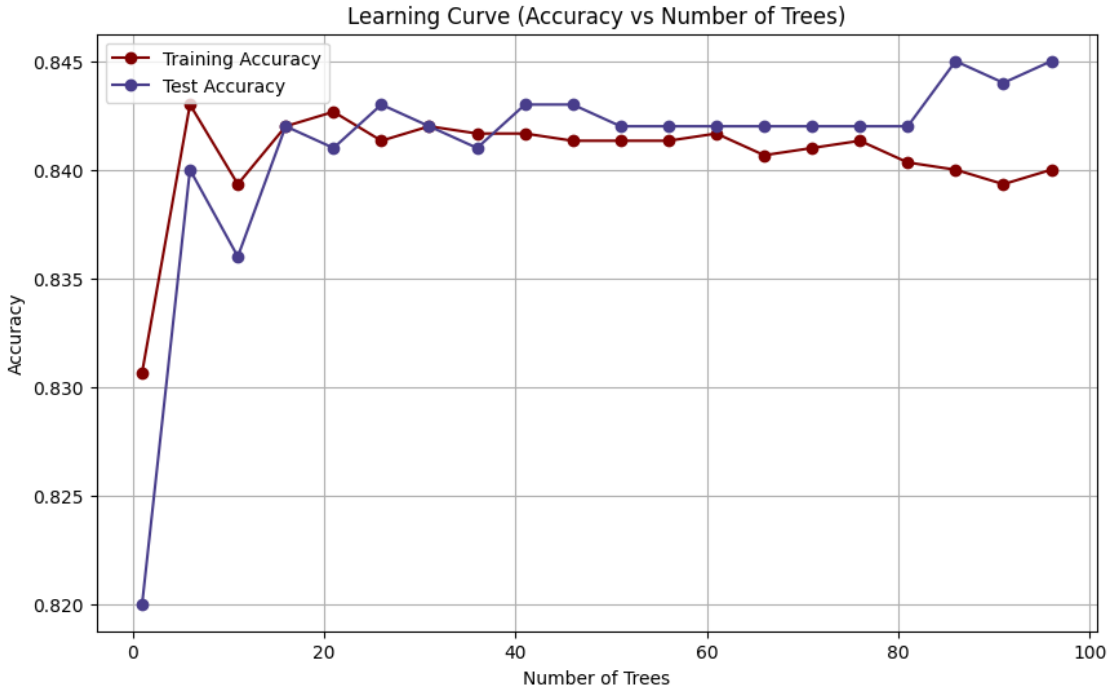
We decided not to use the DecisionTrees dataset because it was already well-fitted, making boosting unnecessary.

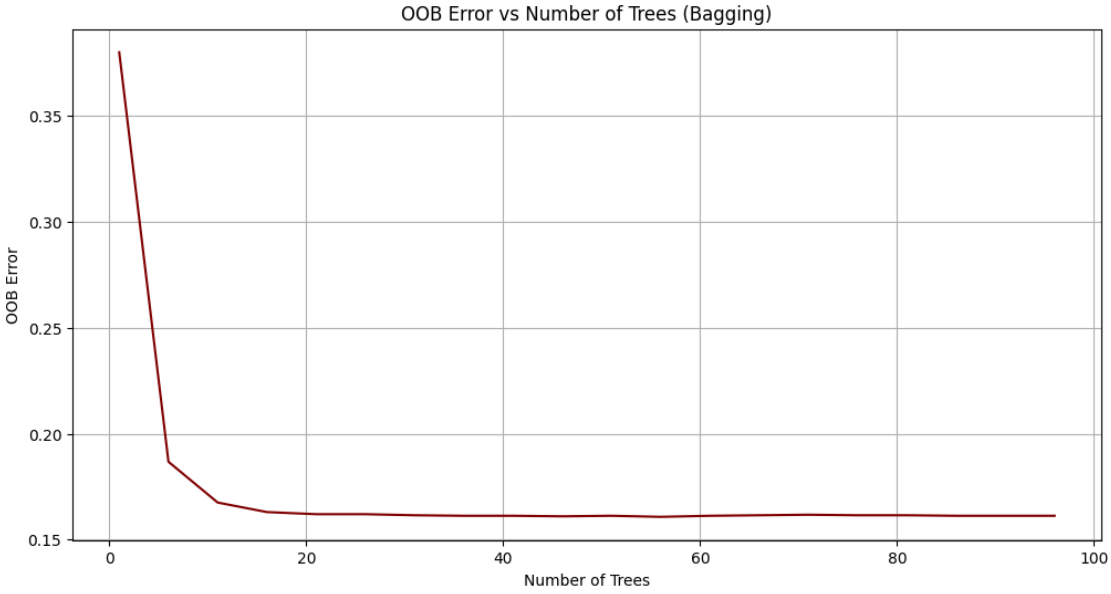
5.1.1 Bagging

Bagging, short for Bootstrap Aggregating, is an ensemble technique designed to improve the performance of machine learning models, particularly decision trees. It works by creating multiple random subsets (bags) of the training data.

For classification tasks, the final prediction is determined by taking the majority vote (mode) of all tree predictions. For regression tasks, the final prediction is calculated by averaging the outputs (mean) from all trees.

Overall, bagging reduces variance and helps prevent overfitting, making the model more stable and accurate.





```
=== Final Model Evaluation ===

=== Cross Validation Results ===
Mean Accuracy: 0.8385
Std Deviation: 0.0071
Min Accuracy: 0.8287
Max Accuracy: 0.8488
Test Accuracy: 0.8440
```

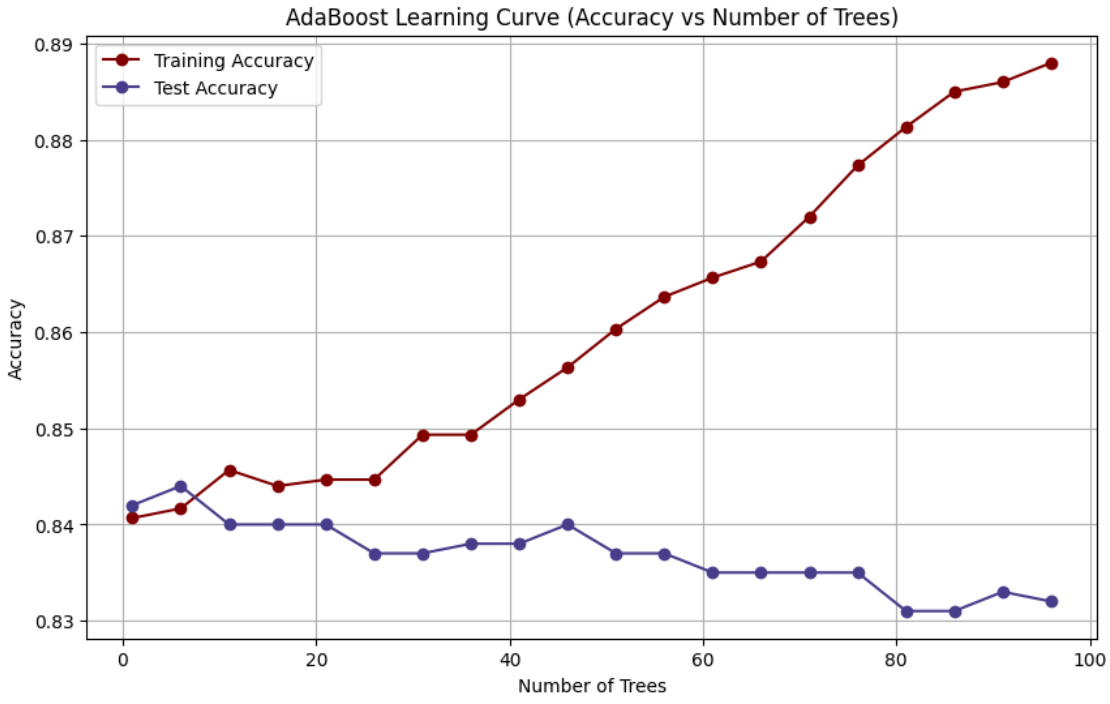
5.1.2 AdaBoost

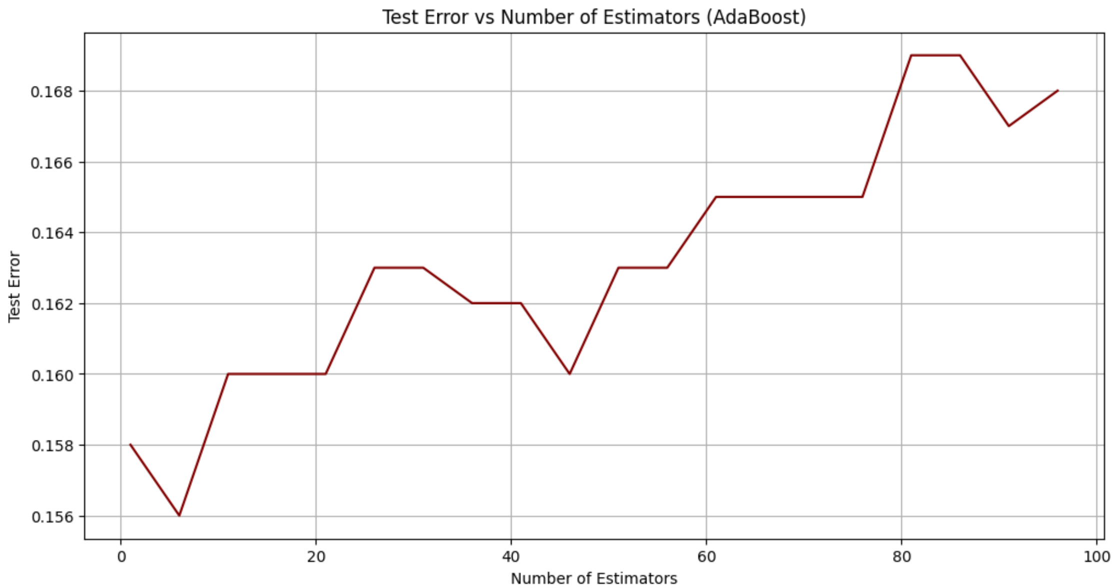
AdaBoost (Adaptive Boosting) is an ensemble technique that improves model performance by focusing on errors. It starts with a simple decision tree (a "stump") and adds more trees, each correcting mistakes from the previous one.

Misclassified examples receive higher weights, ensuring the next tree prioritizes them and the final prediction is based on a weighted vote, where stronger trees have more influence.

AdaBoost is ideal for binary classification with clean data, proving that learning from mistakes can lead to powerful results.

```
=== AdaBoost Analysis ===
```






```
=== Final Model Evaluation ===
```

```
/home/barbara/.local/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/home/barbara/.local/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/home/barbara/.local/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/home/barbara/.local/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/home/barbara/.local/lib/python3.10/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

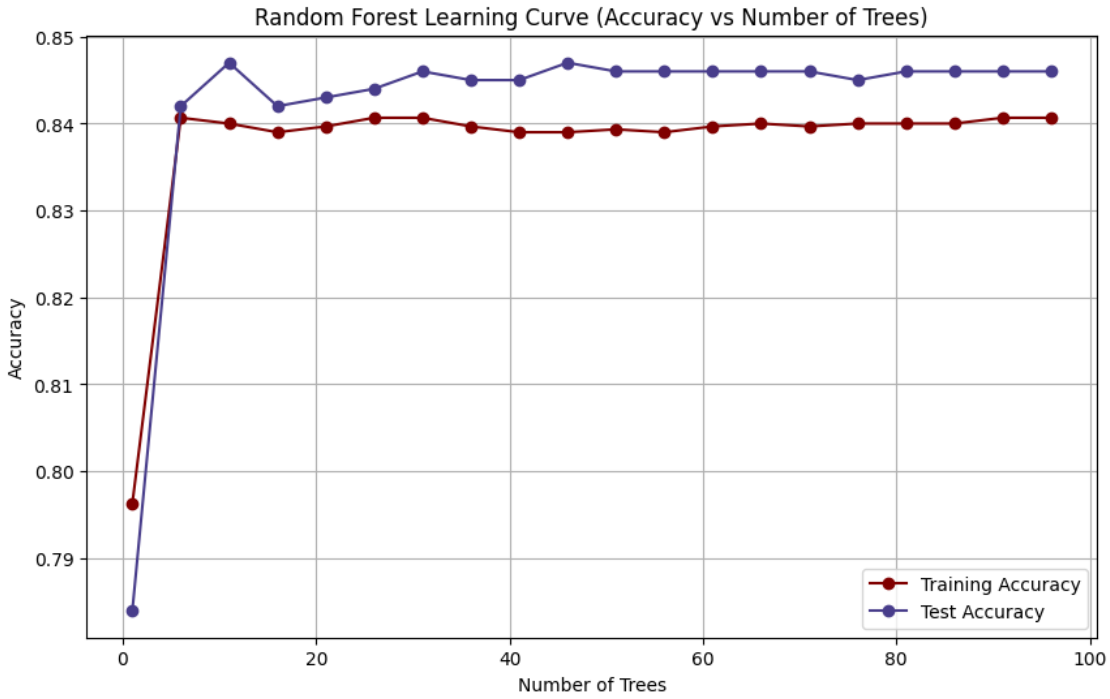
```
=== Cross Validation Results ===
Mean Accuracy: 0.8265
Std Deviation: 0.0078
Min Accuracy: 0.8113
Max Accuracy: 0.8337

Test Accuracy: 0.8300
```

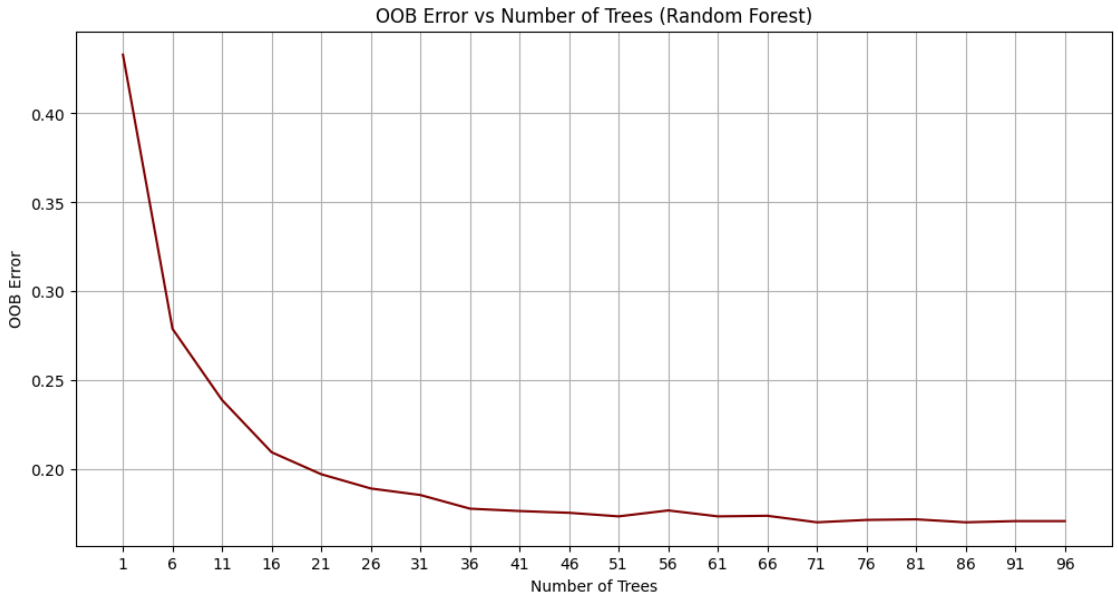
5.1.3 Random Forests

While decision trees are powerful, they can be unstable - small changes in the data can lead to a completely different tree structure. This instability makes them sensitive to variations, which can reduce consistency in predictions. Random Forest addresses this issue by combining multiple decision trees, reducing overfitting and increasing robustness.

As an ensemble method, Random Forest builds multiple trees on different subsets of data and averages their predictions. This reduces variance and enhances model stability. Additionally, it provides feature importance rankings, making it valuable for data interpretation. By comparing our dataset's performance using a single decision tree versus Random Forest, we can prove that while Random Forest improves stability and accuracy, decision trees remain the best choice for structured data with strong feature interactions. This will further support our hypothesis that decision trees are optimal for the dataset we generate.

```
=== Random Forest Analysis ===
```






```
=== Final Model Evaluation ===

=== Cross Validation Results ===
Mean Accuracy: 0.8328
Std Deviation: 0.0078
Min Accuracy: 0.8213
Max Accuracy: 0.8438
Test Accuracy: 0.8390
```

5.1.4 Comparation between boosts

Bagging, AdaBoost, and Random Forest each take different approaches to combining multiple learners. Bagging builds stability through democratic voting - it creates many independent models on random data subsets and averages their predictions. This works particularly well for noisy datasets where we want to smooth out variations. The learning curves typically show quick initial improvement that gradually plateaus as more models are added.

AdaBoost takes a more corrective approach, focusing its efforts where previous models failed. Like a teacher paying extra attention to struggling students, it gives more weight to misclassified instances in each subsequent round. This allows it to achieve strong results with relatively few weak learners, though it risks becoming too specialized if pushed too far. The gap between training and test performance tends to be more pronounced than with other methods.

Random Forest combines the best of both worlds while adding its own twist. It employs Bagging's strategy of multiple independent models, but enhances diversity further by randomizing feature selection at each decision point. This additional layer of randomness makes it remarkably robust - the learning curves typically show smooth, steady improvement with excellent generalization. The built-in out-of-bag error estimation is a particularly convenient feature that provides reliable

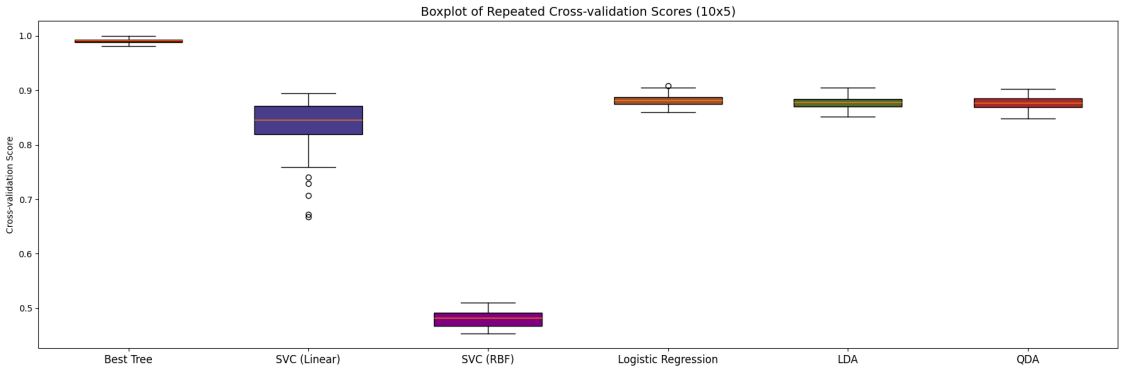
performance metrics without separate validation.

In practical terms, if your data is messy and you need stability, Bagging is a solid choice. For problems where certain cases are consistently harder to classify, AdaBoost's adaptive approach shines. But for most real-world situations where you want good performance without excessive tuning, Random Forest's balanced approach makes it the go-to option. All three demonstrate that sometimes, many voices really are better than one - they just take different approaches to managing the chorus.

6 Datasets in other models

6.1 Decision Trees DataSet

```
SVC (Linear) Model Accuracy with DecisionTrees Dataset: 0.867
SVC (RBF Kernel) Model Accuracy with DecisionTrees Dataset: 0.469
Logistic Regression Model Accuracy: 0.880
Linear Discriminant Analysis Model Accuracy: 0.874
Quadratic Discriminant Analysis Model Accuracy: 0.877
```

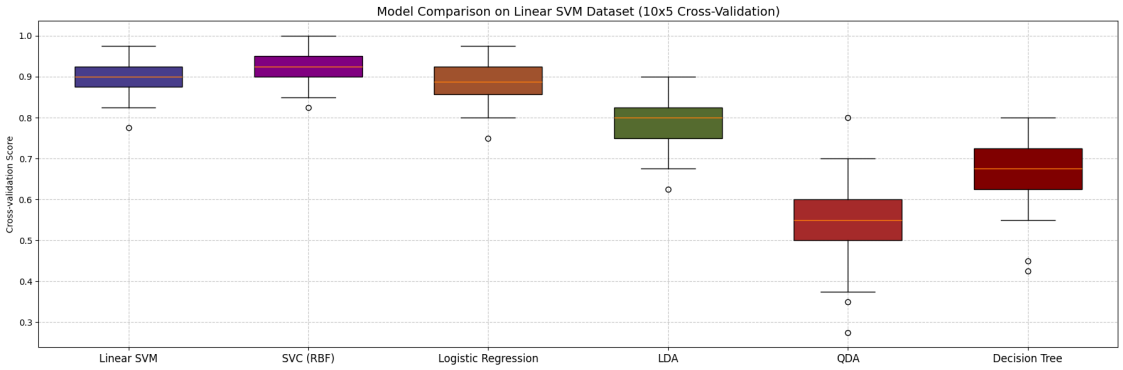


As seen in this graphic, the best type of dataset for Decision Trees acts poorly when applied to other types of machine learning

6.2 SVM

6.2.1 Linear SVM DataSet

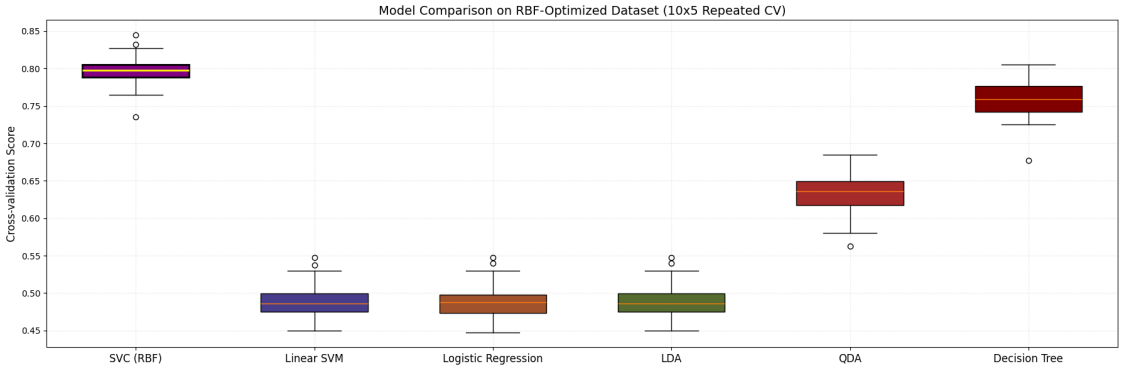
```
Linear SVM Model Accuracy with SVM Dataset: 0.875
SVC (RBF Kernel) Model Accuracy with SVM Dataset: 0.975
Logistic Regression Model Accuracy: 0.850
Linear Discriminant Analysis Model Accuracy: 0.800
Quadratic Discriminant Analysis Model Accuracy: 0.625
Decision Tree Model Accuracy: 0.650
```



Linear SVM demonstrates superior performance, though RBF SVM remains competitive by detecting subtle patterns even in linearly-structured data.

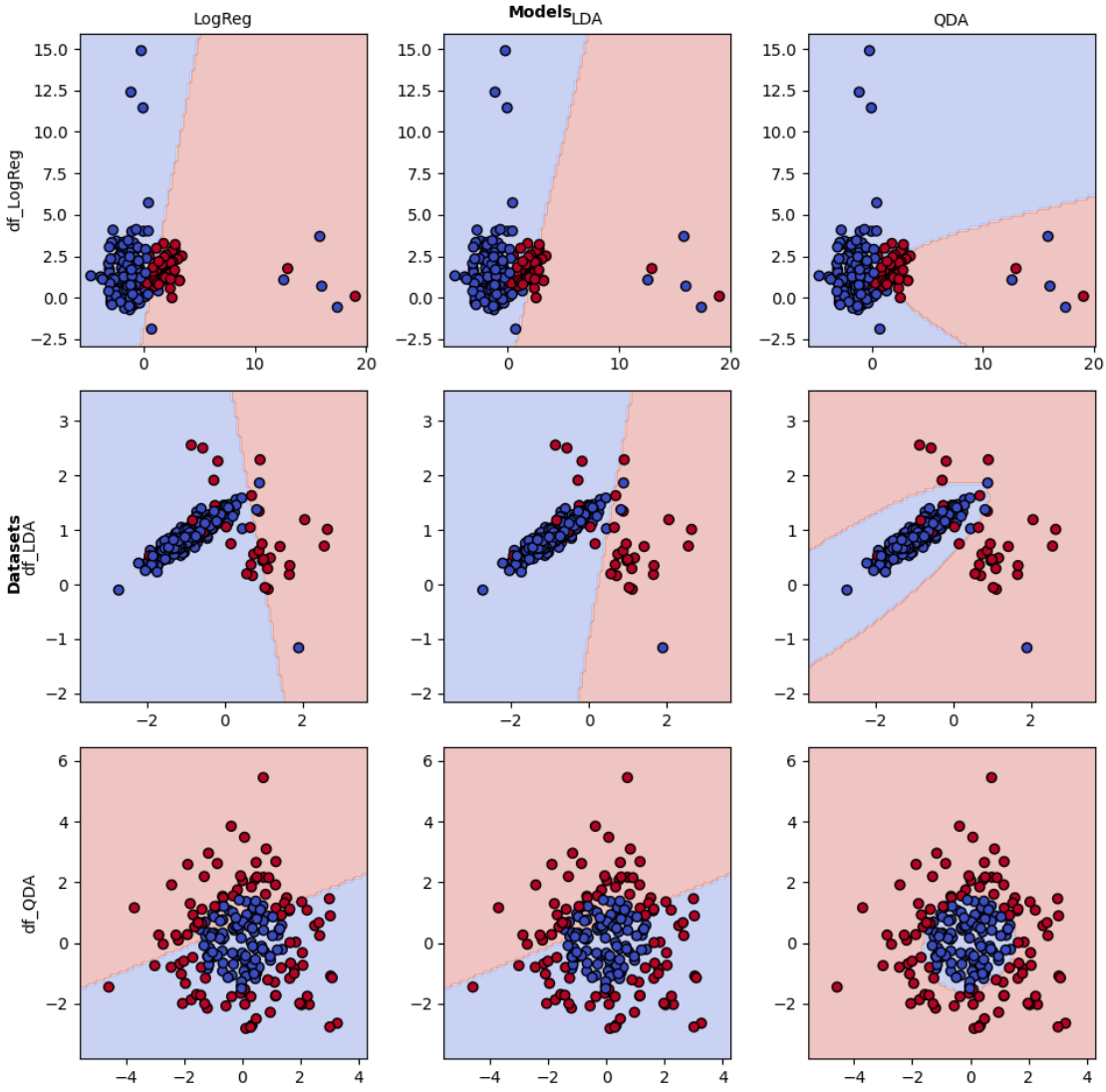
6.2.2 SVM RBF DataSet

```
SVC (RBF Kernel) Model Accuracy with RBF Dataset: 0.812
Linear SVM Model Accuracy with RBF Dataset: 0.465
Logistic Regression Model Accuracy: 0.465
Linear Discriminant Analysis Model Accuracy: 0.465
Quadratic Discriminant Analysis Model Accuracy: 0.618
Decision Tree Model Accuracy: 0.792
```



The RBF-optimized dataset demonstrates superior performance with SVM (RBF), achieving the highest cross-validation scores compared to linear models and other classifiers.

6.3 LDA, QDA, LogReg Datasets



By analysing the plots above, we can see the behaviours of the 3 models with the datasets created for each of them.

On the first row, the dataset made for Logistic Regression, we can clearly see the effect of outliers to the LDA decision boundary. Logistic regression does a much better job in seperating the two classes, balancing the fit. QDA seems to also be highly influenced by the outliers, looking like it almost centered the decision boundary on the outliers.

Finding a dataset that made LogReg struggle more than LDA was a challenge, but it was done successfully. We can see that the LDA does a better job separating the cluster of the red class instances than LogReg. However, because of the added noise, we could say QDA performed well by isolating the blue cluester. This is also possible because blue is representing the majority class of the dataset, being easier to isolate.

Finally, for the QDA dataset, it worked as expected. Being a dataset that requires non-linear decision boundaries, QDA was the best model by isolating almost perfectly the blue cluester, unlike LDA and LogReg.

7 References / WebReferences:

- https://www.quora.com/What-types-of-data-sets-are-appropriate-for-decision-tree
- https://www.stratascratch.com/blog/tree-based-models-in-machine-learning/
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html
- https://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/
- https://medium.com/@damaniayesh/the-art-of-bagging-enhancing-decision-trees-with-ensemble-techniques-2786e4bffc27
- https://medium.com/data-science/adaboost-classifier-explained-a-visual-guide-with-code-examples-fc0f25326d7b
- https://dl.ebooksworld.ir/books/Artificial.Intelligence.A.Modern.Approach.4th.Edition.Peter.Norvig.%20St
- https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html
- https://link.springer.com/article/10.1007/s10618-021-00737-9