Introduction:

This report will summarize and compare the results of using 3 different types of classification models. Using a dataset with 9 features pertaining to a binary class (class 0 or class 1). For all classifiers, a 80/20 train/test data split was used. Hyper parameter tuning (using sklearn's CV grid search class) was used to find the best set of hyper parameters along with K Fold cross validation using K=5. The choice of models used isn't based on a preference metric and therefore is completely random since the nature of the classification problem is not explicitly defined, the report merely offers a summary on how the different models performed based on the given classification problem. Numpy and Pandas were used to create and alter datasets and dataframes. Seaborn was used along with matplotlib to provide improved UI in viewing heatmaps and metric evaluations.

Data Descriptives:

Total data instances = 700

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9
count	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000	700.000000
mean	4.415714	3.131429	3.204286	2.804286	3.214286	3.557143	3.437143	2.864286	1.588571
std	2.814236	3.050343	2.970958	2.854153	2.213193	3.613026	2.436676	3.052265	1.713995
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	6.000000	5.000000	4.000000	1.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

Fig 1

Class 1

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9
count	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000
mean	7.195021	6.572614	6.560166	5.547718	5.298755	7.593361	5.979253	5.863071	2.589212
std	2.428849	2.719512	2.562045	3.210465	2.451606	3.129263	2.273852	3.350672	2.557939
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	5.000000	4.000000	4.000000	3.000000	3.000000	5.000000	4.000000	3.000000	1.000000
50%	8.000000	6.000000	6.000000	5.000000	5.000000	10.000000	7.000000	6.000000	1.000000
75%	10.000000	10.000000	9.000000	8.000000	6.000000	10.000000	7.000000	10.000000	3.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000

Fig 2

Class 0

	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9
count	459.000000	459.000000	459.000000	459.000000	459.000000	459.000000	459.000000	459.000000	459.000000
mean	2.956427	1.324619	1.442266	1.363834	2.119826	1.437908	2.102397	1.289760	1.063181
std	1.672490	0.906830	0.996960	0.995886	0.916145	1.310839	1.079976	1.057787	0.501456
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000	1.000000	1.000000
50%	3.000000	1.000000	1.000000	1.000000	2.000000	1.000000	2.000000	1.000000	1.000000
75%	4.000000	1.000000	1.000000	1.000000	2.000000	1.000000	3.000000	1.000000	1.000000
max	8.000000	9.000000	8.000000	10.000000	10.000000	10.000000	7.000000	9.000000	8.000000

Fig 3

Means of Class Features

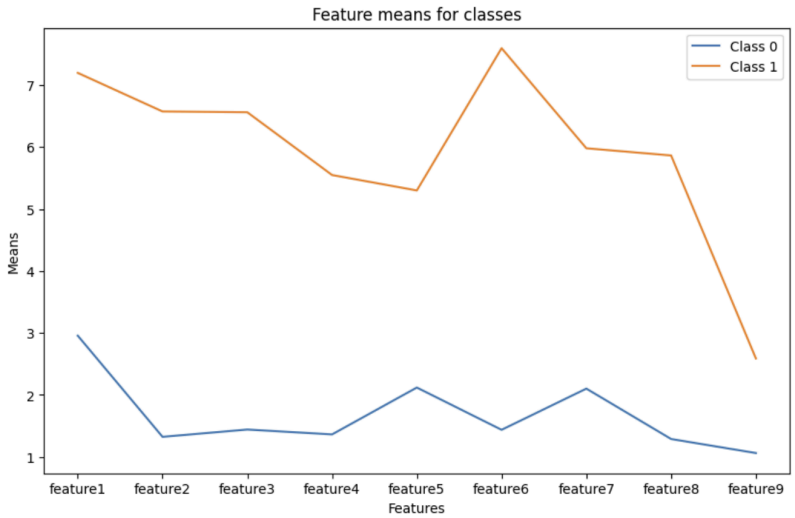

Fig 4

Analysis:

Taking a look at the overall data descriptives (*Fig 1*) we can see that there are a total of 700 data instances available to us to train our model(s). However, if we take a closer look at the data

descriptives as per each class (*Fig 2 & Fig 3*), we begin to identify the discrepancy that exists in the distribution of the data. Class 0 has 90.45% more data instances which means that the training data is biased towards class 0 which will negatively affect our model predictions. *Fig 4* illustrates the means of each feature for both classes. We can visually identify that the means for class 0 oscillate between the range of 1-3 (lower quartile) whereas for class 1 the values range mostly between 5-7 which in the upper quartile of means.

Since the features of the training data are more than 2, it is not possible to visualize them in a 2 dimensional space therefore I have included a section in my code which creates a scatter plot for all pairs of features as a way to visualize given features in 2 dimensional space.

Classifiers:

Decision Tree Classifier:

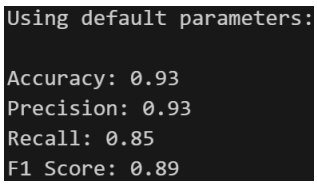

Fig 5

After the training data was successfully split into train/test sets, the decision tree classifier was trained using default parameters. The results for the performance of this model can be seen using *Fig 5.*

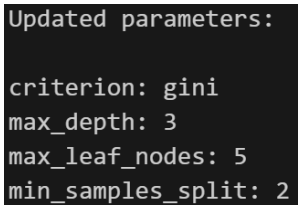

Fig 6

The hyper parameters were then tuned using sklearn's grid search and the model was retrained using the updated parameters *(Fig 6).*

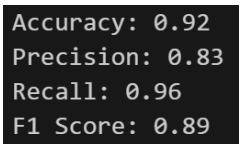

Fig 7

The updated model performed significantly well identifying actual positives correctly leading to an improved recall score. However, the precision scores fell which means the model is now more likely to predict a false positive. Using cross validation and hyper parameter tuning seemed less useful for a decision tree classifier as the model score for a shuffled data set performed the same (if not better) for default parameter values.

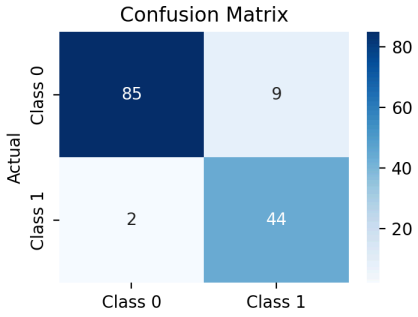

Fig 8

The confusion matrix in *Fig 8* summarizes the results for the updated model (using a data set with random state = 42).

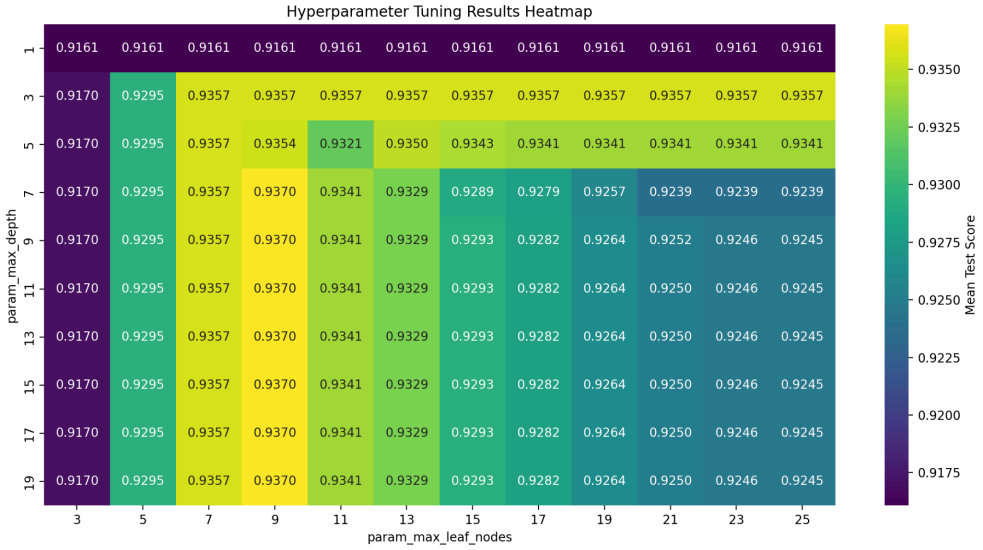

Fig 9

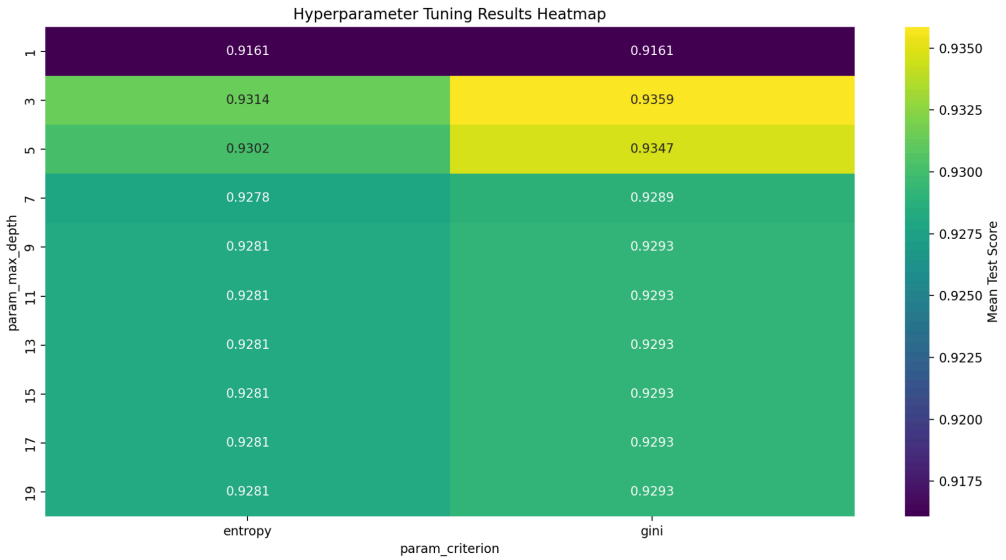

Fig 10

Fig 9 and *Fig 10* illustrate the process of the hyper parameter tuning as we can see the mean test score for the model maximizing around the updated parameters chosen by the grid search.

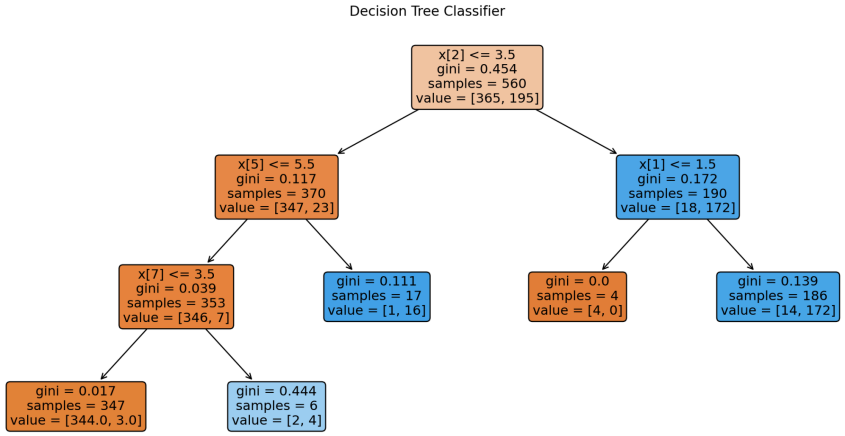

Fig 10

The final updated decision tree can be seen in *Fig 10.*

Random Forest Classifier:

The same process was used for the random forest classifier as the model is basically a collection of a number of individual decision trees. This also means that we have more hyper parameters to tune now e.g number of decision trees.

```
Using default parameters:

Accuracy: 0.97
Precision: 0.98
Recall: 0.93
F1 Score: 0.96
```

Fig 11

After the model was run using the default parameters, the following results were achieved *(Fig 11)*.

```
Updated parameters:

criterion: entropy
max_depth: 5
max_leaf_nodes: 15
min_samples_split: 2
n_estimators: 70
```

Fig 12

The parameters were then updated using the grid search and the resulting best set of parameters were achieved (*Fig 12*).

```
Accuracy: 0.96
Precision: 0.96
Recall: 0.93
F1 Score: 0.95
```

Fig 13

The results in *Fig 13* show that the updated model performed poorer as compared to the default parameters. This can be due to a number of reasons. Firstly, unlike decision trees, the hyper tuning process took significantly longer (400 seconds on average) and therefore only a selected number of hyper parameters were allowed to be tuned. Due to this computational limitation the steps for the hyper parameters might have been set too large allowing the model to miss the optimal state of certain hyperparameters.

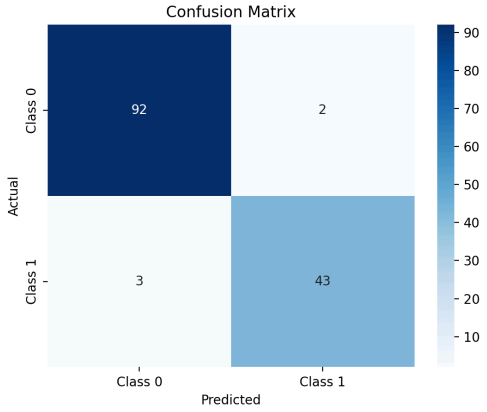

Fig 14

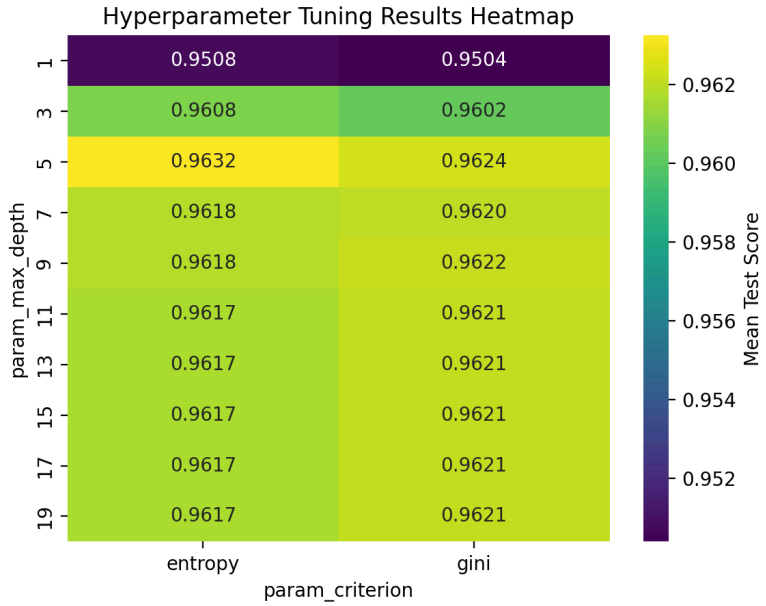

Fig 15

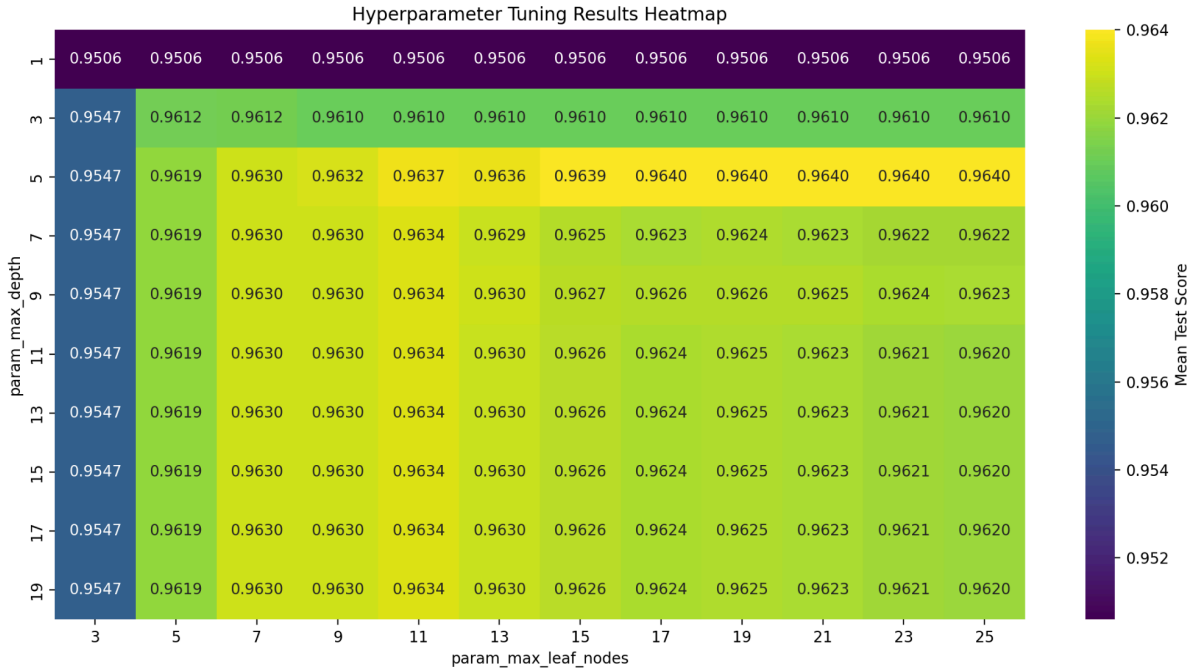

Fig 16

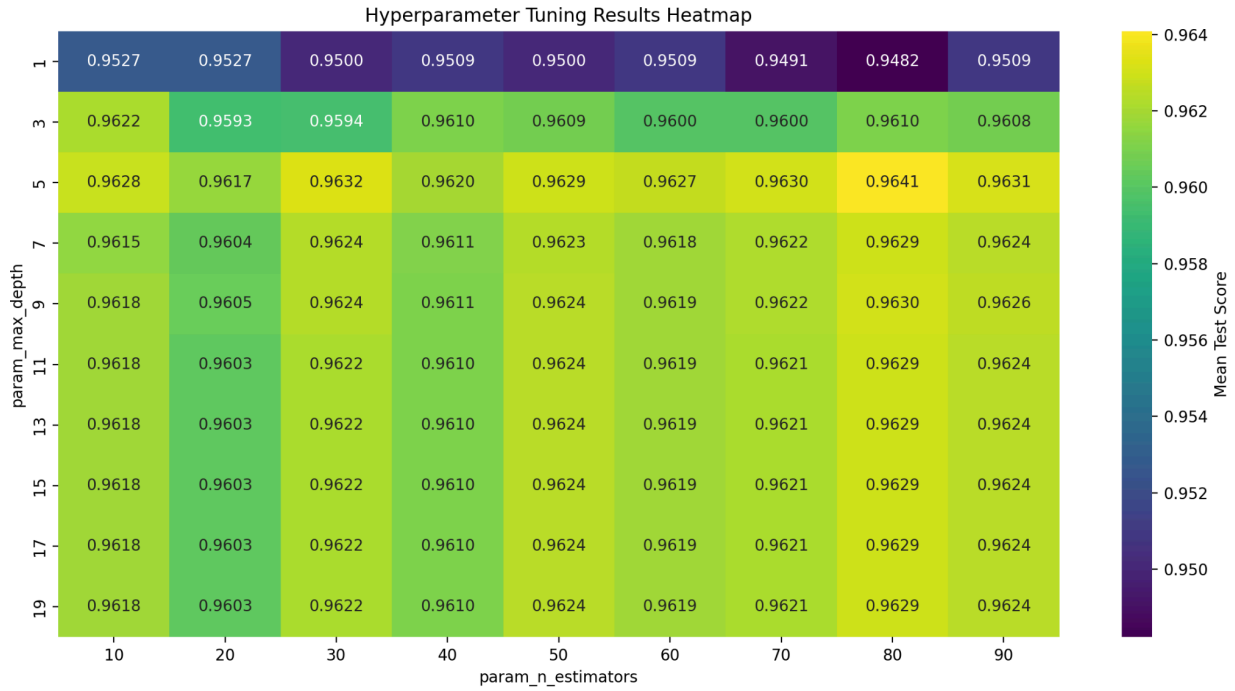

Fig 17

The confusion matrix in Fig 14 summarizes the results using the updated hyperparameters and *Fig 15*, *Fig 16,* and *Fig 17* show how the hyperparameter tuning converges to obtain the optimal hyperparameters.

Support Vector Machine:

The support vector machine classifier acts differently than the aforementioned classifiers by assigning support vectors in the classification dataset.

```
Using default parameters:

Accuracy: 0.96
Precision: 0.94
Recall: 0.96
F1 Score: 0.95
```

Fig 18

```
Accuracy: 0.96
Precision: 0.94
Recall: 0.96
F1 Score: 0.95
```

Fig 19

```
Updated parameters:

C: 1.0
degree: 0
gamma: scale
kernel: rbf
```

Fig 20

The model performed fairly well with the default parameters and therefore even with the updated parameters the performance did not seem to budge (*Fig 19*) as the default parameters were already close to optimal (*Fig 20*).

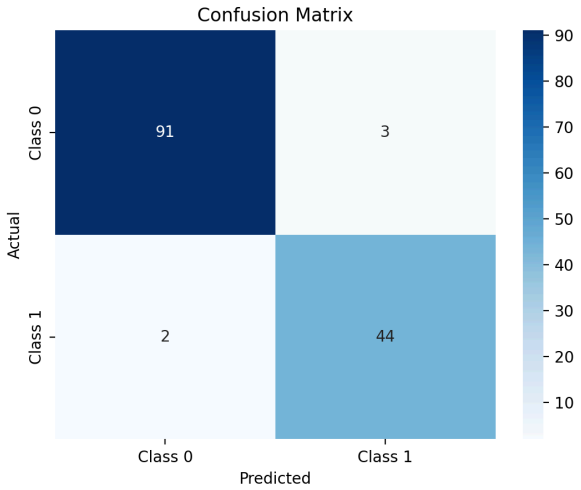

Fig 21

The confusion matrix summarizes the evaluation metrics in *Fig 2* . The model performed significantly better than the random forest classifier in identifying true positives yielding a high recall score. The Random forest classifier proved to be an inferior classifier when trying to

maximize model recall.

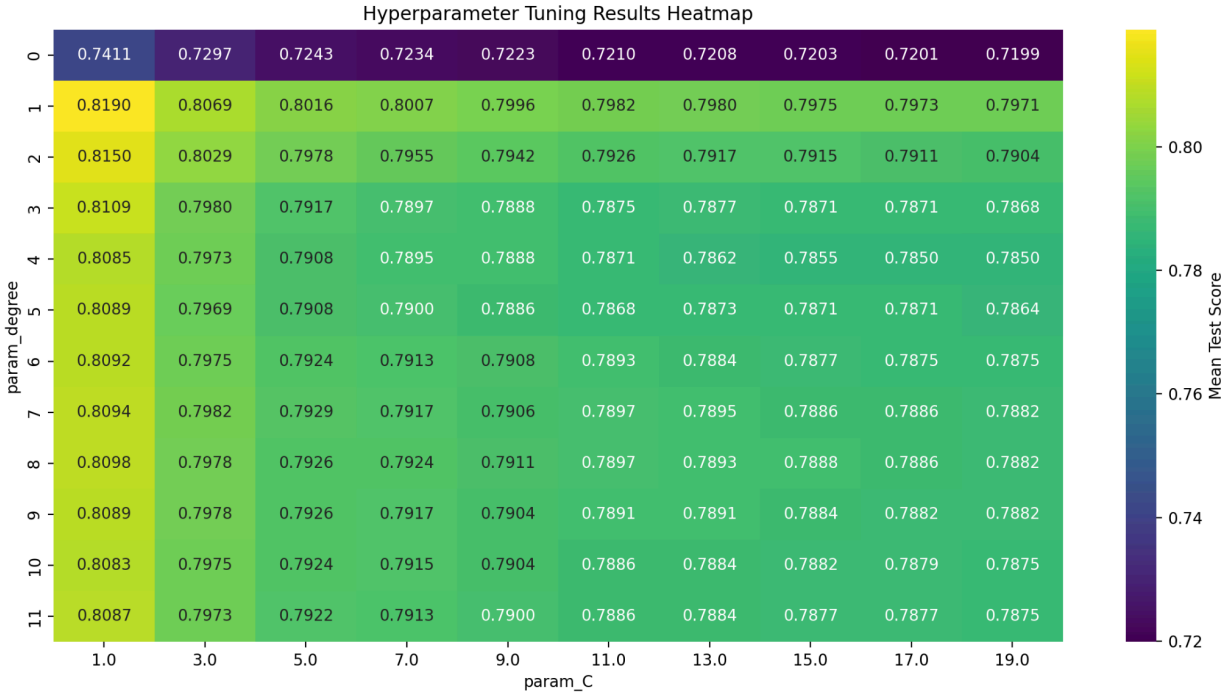

Fig 22

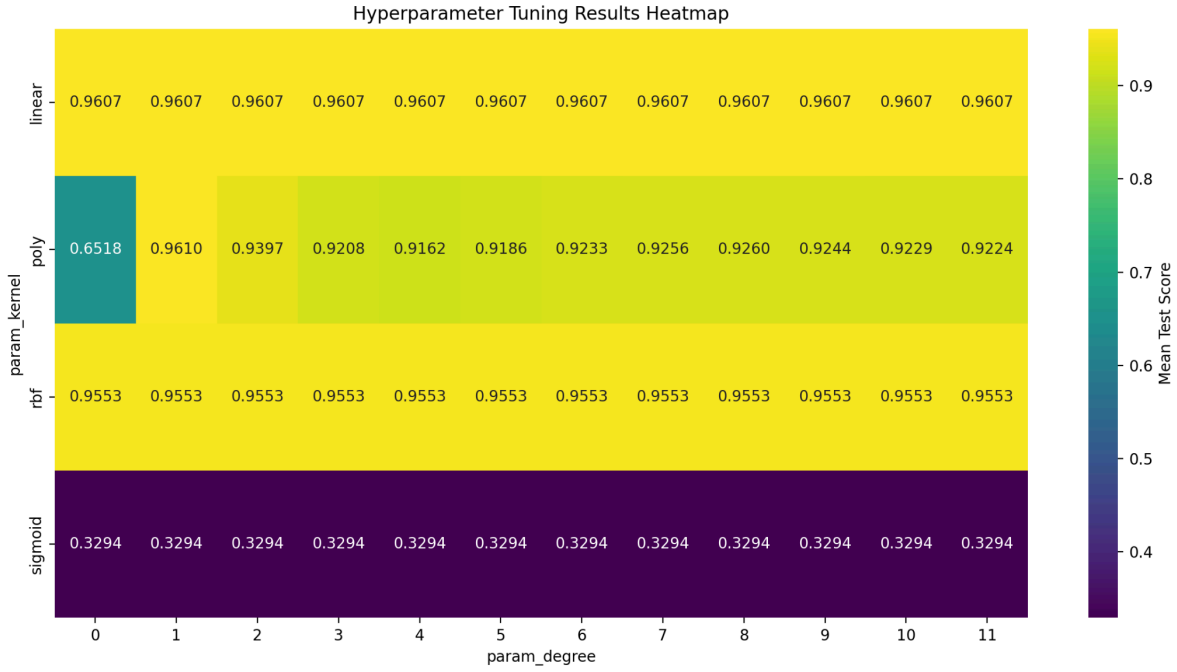

Fig 23

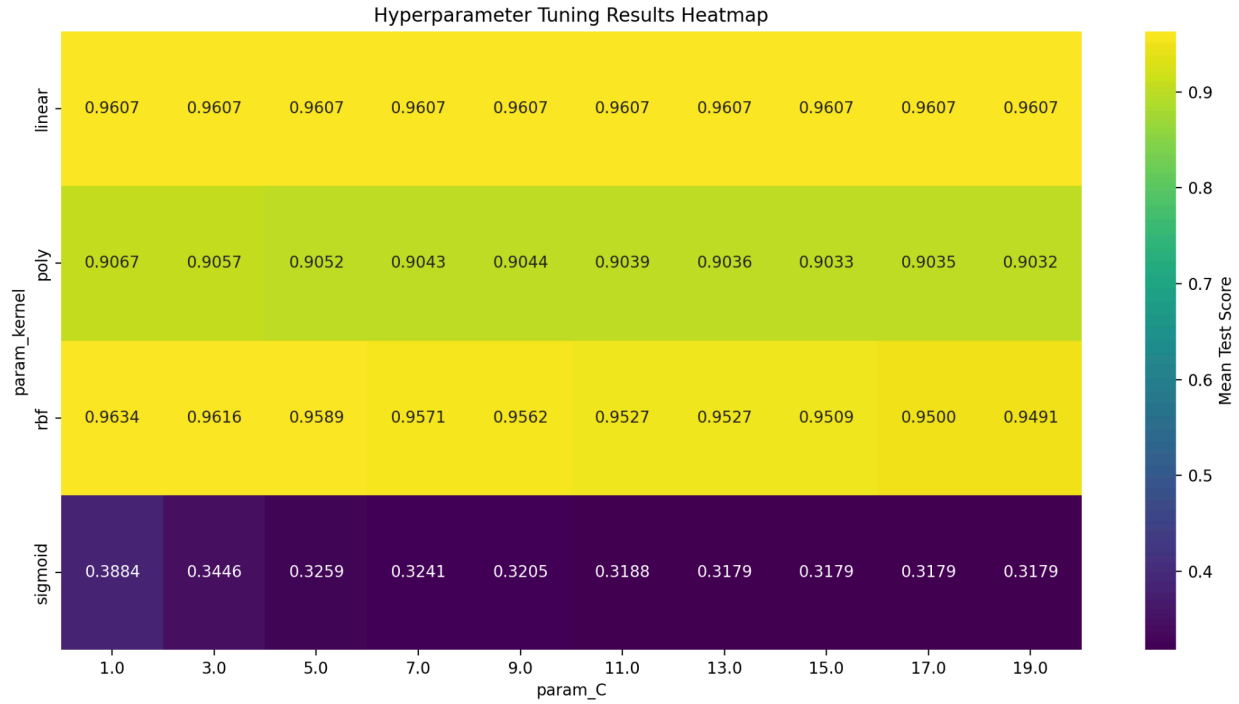

Fig 24

Results in Fig 22, Fig 23, and Fig 24 exhibit the performance of the model using different hyper parameters. We can see that RBF performed as the superior kernel when paired with a lower C value. A low C value means that the model is trying to avoid overfitting the training data set.

Conclusion:

All three models performed well in differing evaluation metrics. The decision tree classifier performed well with updated parameters and had the highest recall metric of 0.94 which is a big jump when compared to the same model using default parameters (recall = 0.84) therefore hyperparameter tuning worked best for the decision tree classifier. The random forest classifier performed the best when it came to accuracy and precision which makes intuitive sense as random forest comprises a number of decision trees working together using a rating system hence yielding more optimal results. The support vector machine functions a little differently then the other two classifiers and the results depict this as the model performed well enough across all metrics achieving the highest values for accuracy and recall proving itself to be the best of both worlds whilst not having to sacrifice on its precision score unlike the decision tree classifier.