Optimizing Sanity Test Performance through Machine Learning-Based Risk Assessment

INTRODUCTION

The second problem statement is selected for the project which is the Optimizing Sanity Test Performance through Machine Learning-Based Risk Assessment.

The reason for selecting this problem statement is being an active part of the open source community it is really beneficial for contributors to get to know if the sanity check is failing or not with optimal accuracy

IMPORTING LIBRARIES AND READING THE DATA

Here, all the required libraries are imported and read the data in a data frame.

DATA TRANSFORMATION

In order to visualize the data or create a model, first data needs to be transformed and pre-processed in a suitable fashion.

Transformation starts with the column of File Changed. As is, it has too many unique values and huge file paths. Hence, the file path is split into a super file and a specific file.

Now moving toward components, there are certain components that don't have their super components mentioned.

Components are split into super components and specific components and then, later on, dropped the rows without a super component.

Null values have been dropped in the Tester column.

For sanity names and nets tested, since there are several comma-separated values, duplicate rows are created for each of the comma-separated values, this not only reduces the number of unique values but also increases the size of the data.

A new data frame is created then with all the intended columns and then all the categorical columns are label encoded to make them ready for the machine learning model.

The next step is to oversample the data so that pass, fail, and aborted all get equal importance and then split the data into training and testing.

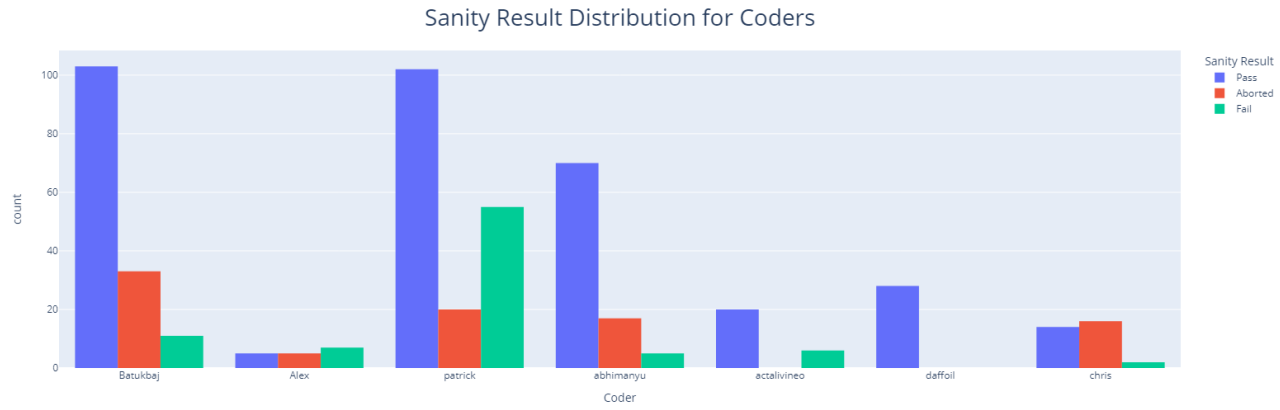
DATA VISUALIZATION

Here, a combination of ipywidgets as well as plotly has been used to create a highly interactive visualization.

iPywidgets allows condensing multiple visualizations into one which can be summoned as per requirement through a dropdown menu.

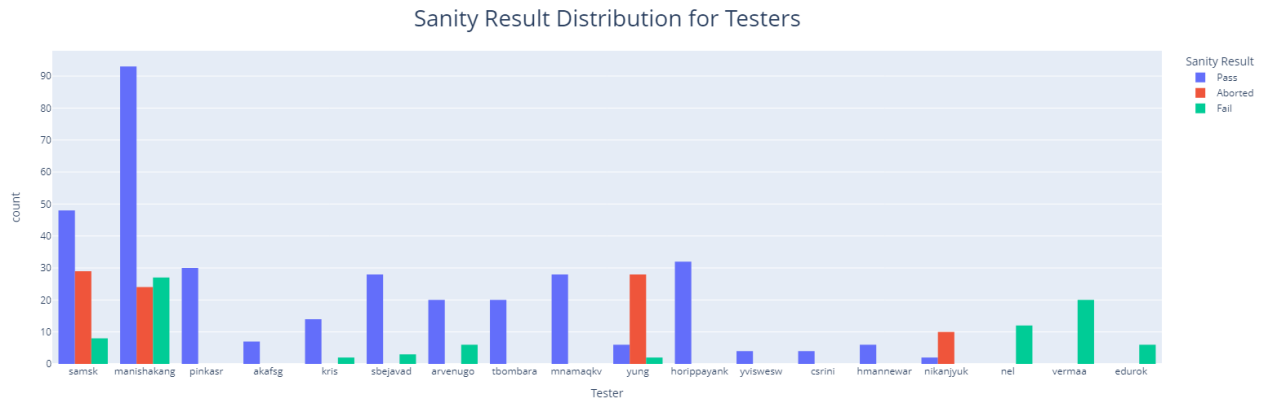
The visualizations themselves are created using plotly so as to allow interactivity and to make relevant data visible on hover

The first visualization is of the Sanity Result Distribution among the coders.



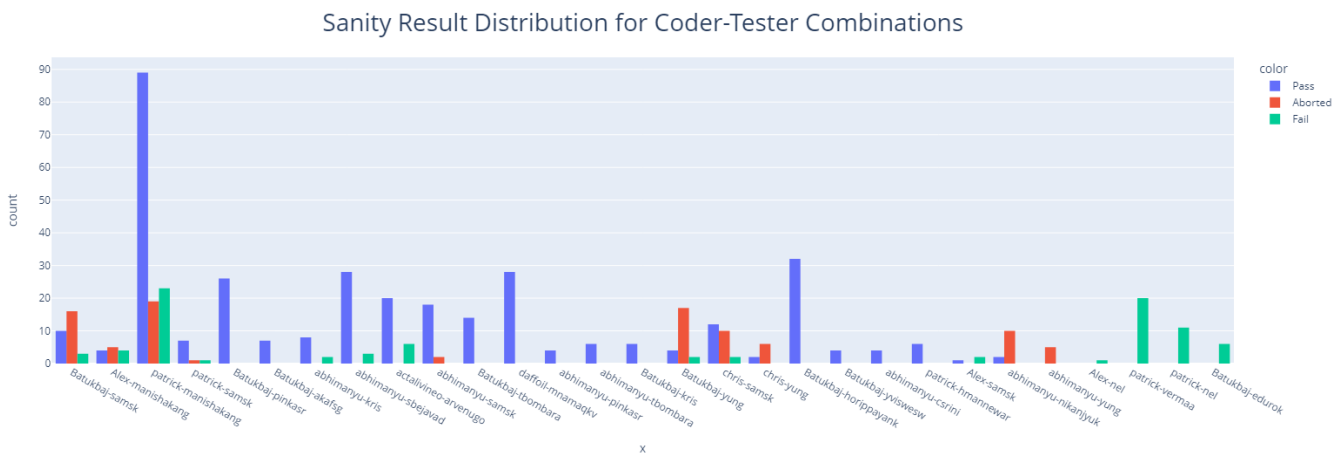
Coders like Batukbaj and Abhimanyu have a high passing rate while coders like Alex and Patrick have a comparatively higher failure rate

The next visualization is of the Sanity Result Distribution for Testers.



Testers like samsk, pinkasr, sbejavad and mnamaqkv have a higher passing rate while testers like manishakang have very high passing count. Also, testers like nel, vermaa and edurok have near 100% failure rate for the given data.

Next up is a visualization of the Sanity Result Distribution for Coder-Tester Combinations

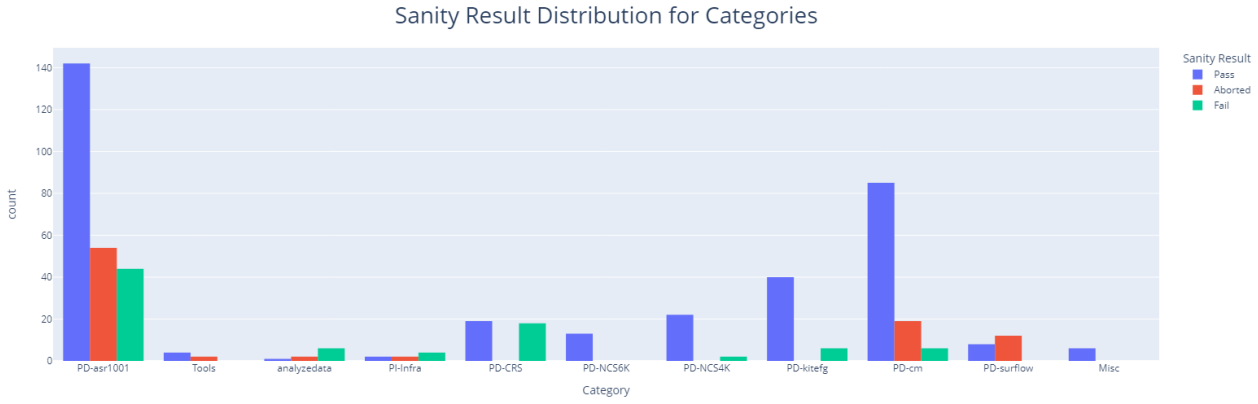


Here the inference is that combinations like that of Patrick-Manishakang, Batukbaj-pinkasr and Batukbaj-horippayank have very high passing rates while that of Patrick-vermaa, Patrick-nel and Batukbaj-edurok have near 100% failure rates.

From the above visualizations, the ultimate inference is that Batukbaj, Patrick, and Abhimanyu are some of the best coders in the company. But sanity test results also seem to be affected heavily by the Tester. Strict Testers even when paired with good coders tend to have a very high failure rate and vice versa.

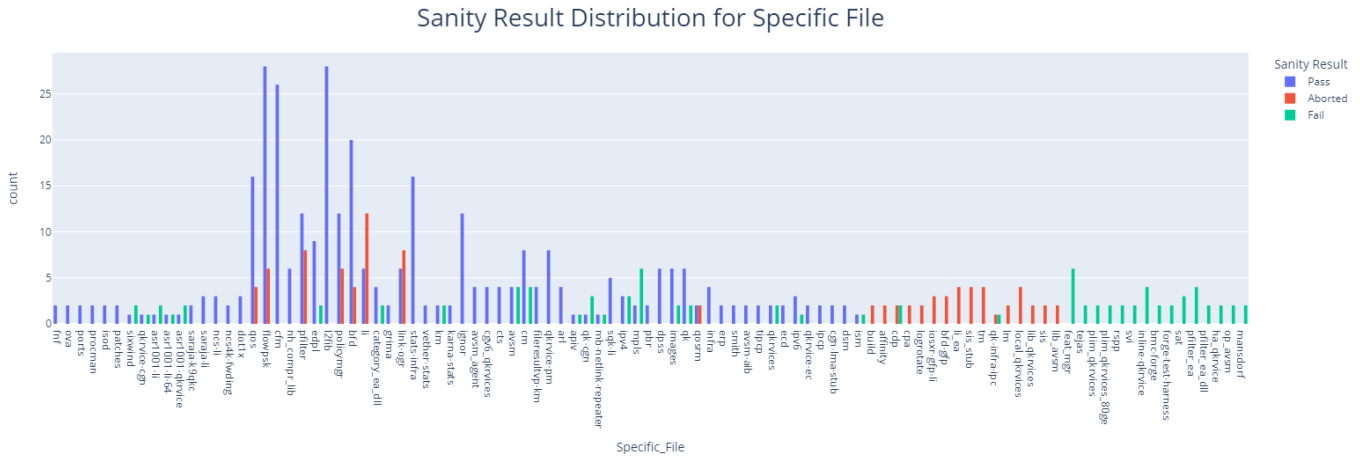
So a low failure probability is only when a good coder is paired with a lenient tester.

Next in the queue is a Sanity Result Distribution for Categories



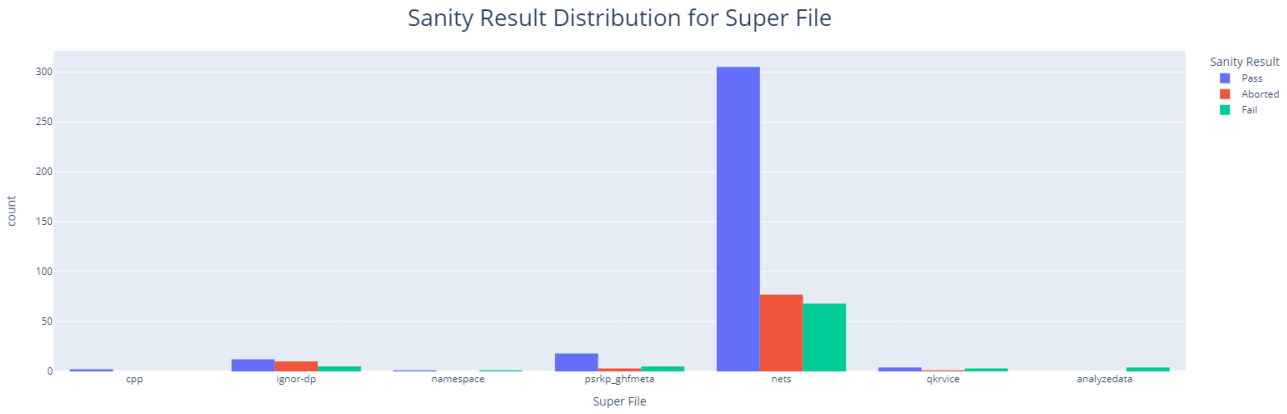
Categories like PD-asr1001 and PD-cm have high passing rates while categories like PD_CRS and analyze data have relatively high failure rates.

Now, next is a Sanity Result Distribution for Specific File

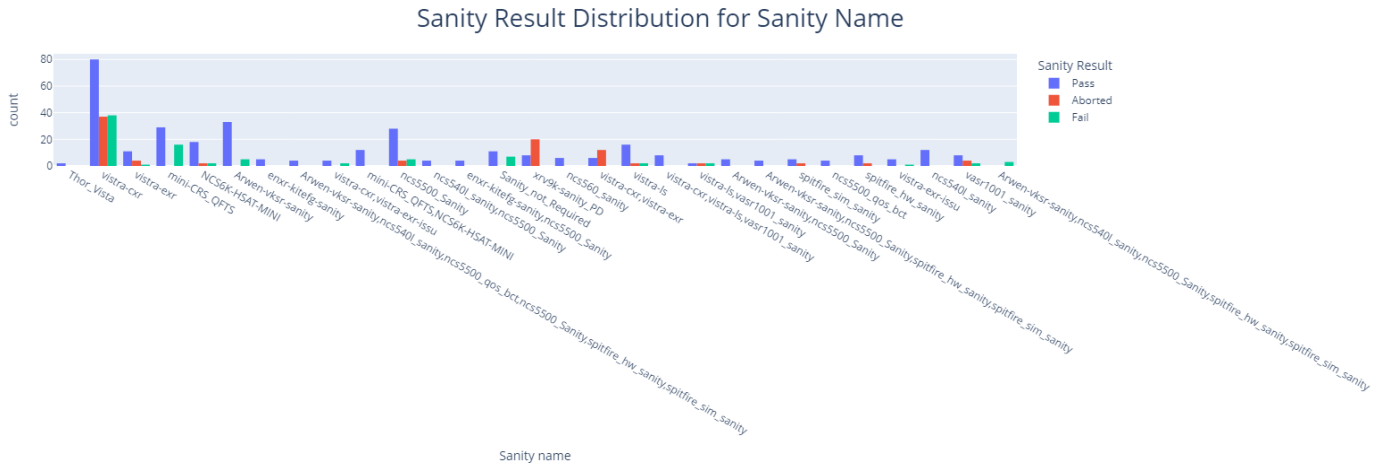


Here, specific files like flowpsk, cfm and l2fib seem to have higher passing rates as compared to specific files like feat_mgr and inline-qkrvice.

Visualization for Super File

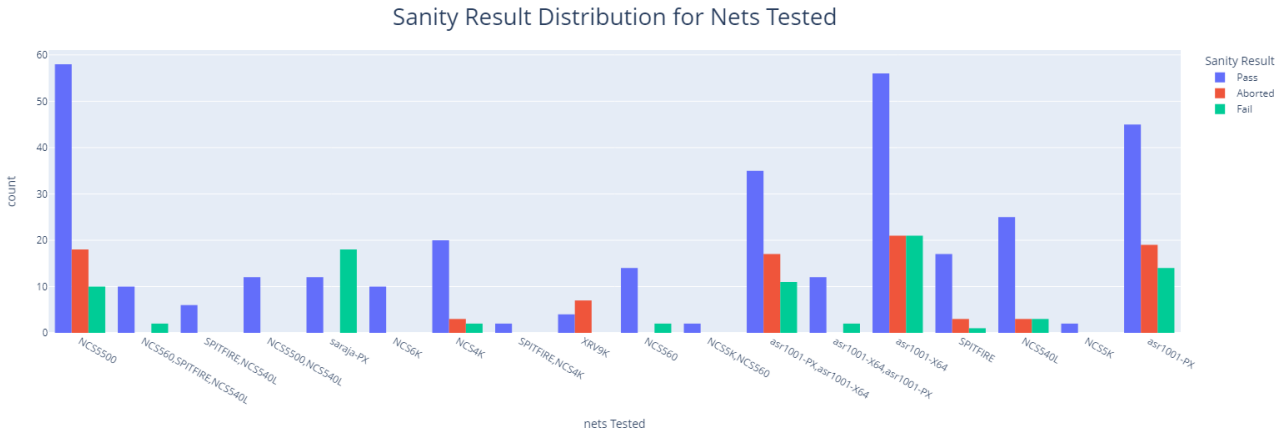


Next is the Sanity Result Distribution for Sanity Name



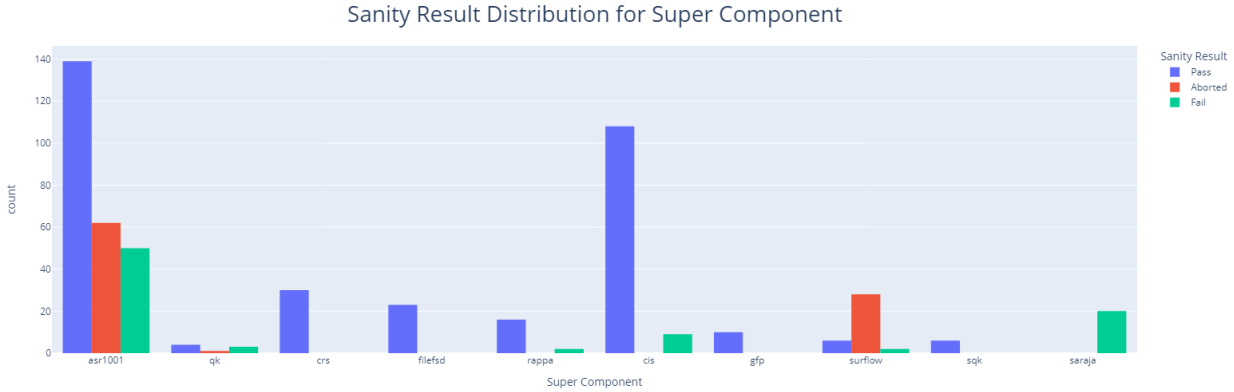
Sanity tests like vistra-cxr and Arwen-vksr-sanity

Next up is the Sanity Distribution for Nets Tested.

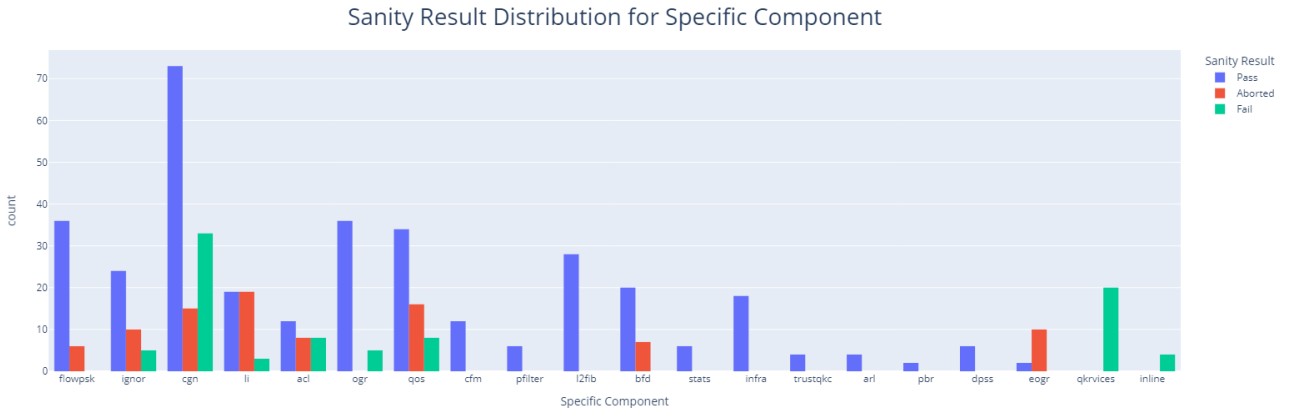


Nets Tested like NCS5500 and SPITFIRE have a high passing rate while Nets like saraja-px have a higher failure rate.

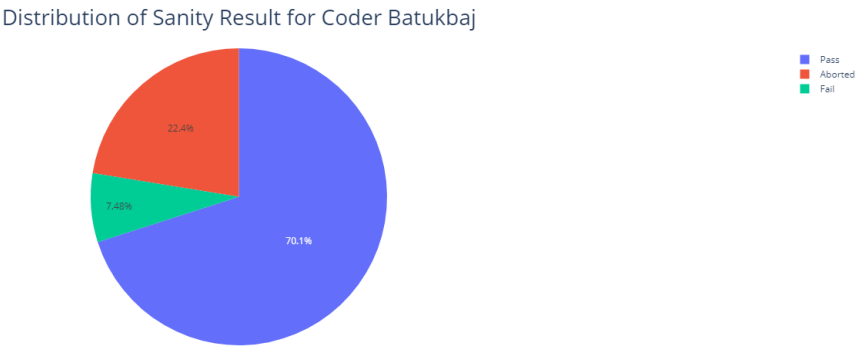
Super Components like asr1001 and cis have high passing rates while super components like saraja have high failure rates.



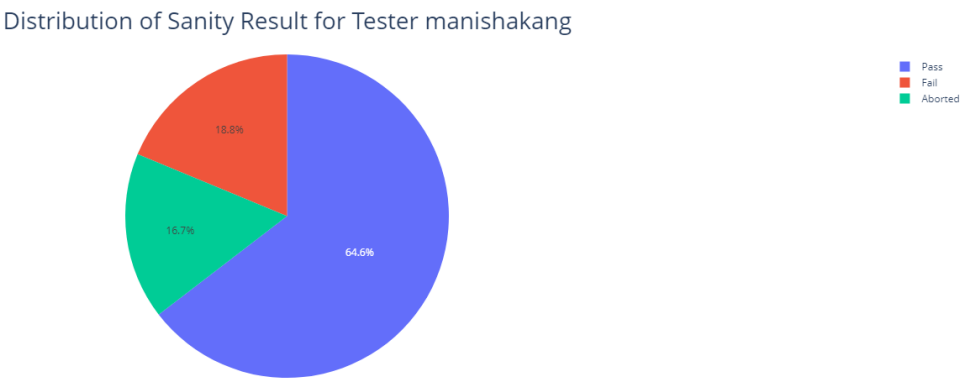
Specific Components like flowpsk and ogr have high passing rates while specific components like qkrvices and inline have high failure rates



In the next interactive menu one can see in the form of a pie graph, the performance of individual coders with blue representing the percentage of tests that passed, green being the failure percentage and red being the aborted percentage



Next up is a similar menu but this time for testers. So one can see and estimate how strict or lenient a particular tester is.



Now that the data has been transformed, pre-processed, and analyzed in depth, the next step is to move towards making a Machine Learning Model to predict the Sanity test results given other parameters.

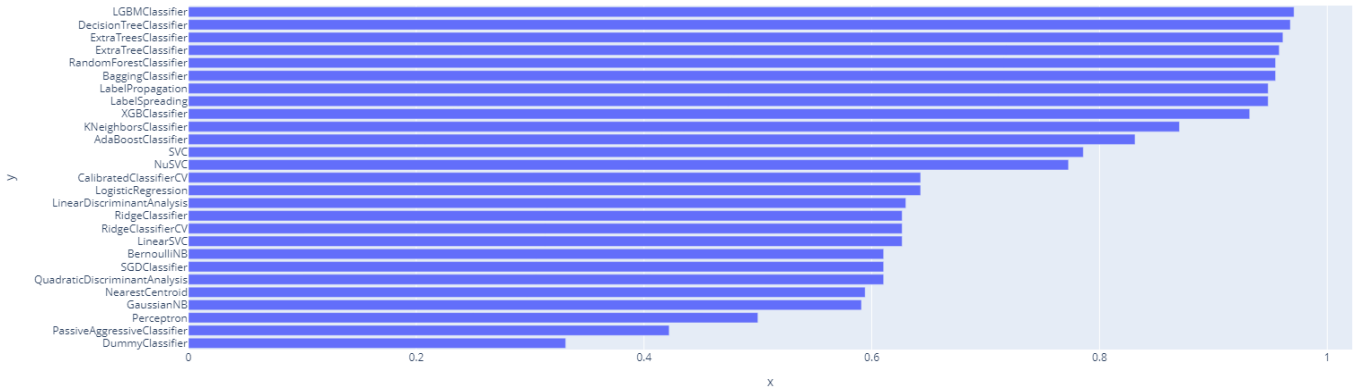
MACHINE LEARNING MODEL

Moving on to the Machine Learning Model, first a decision tree model is used as a base just to be able to plot the feature importance.

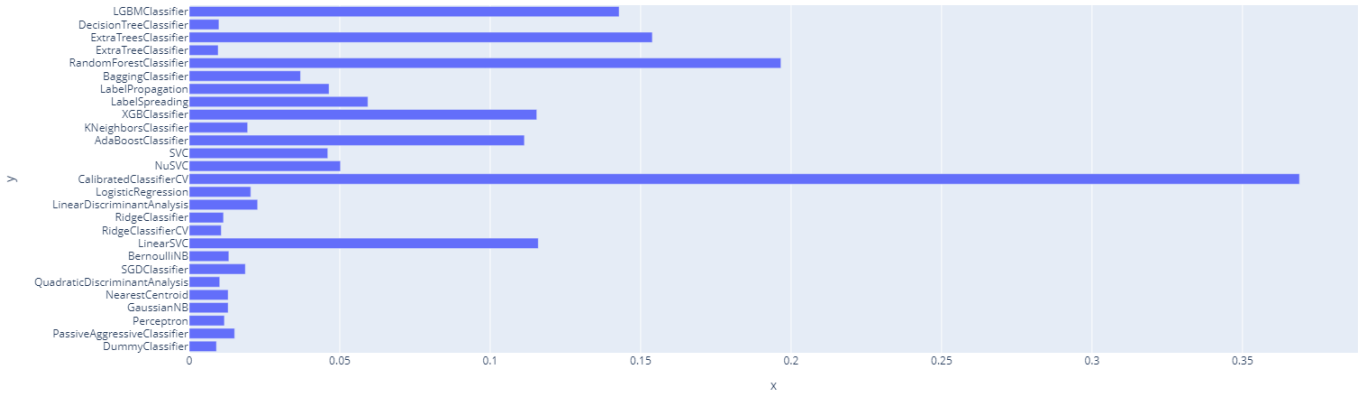
Plotting the same it is observed that Specific files, Sanity names, Tester, and coder are some of the most important features or in other words major players in deciding the sanity test result.

Then the classic Lazy Predict Classifier is used to get an estimate of how each model performs for this particular dataset.

Plotting this new found information, the inference is that LightGBM and Decision Tree classifiers are some of the best-performing models



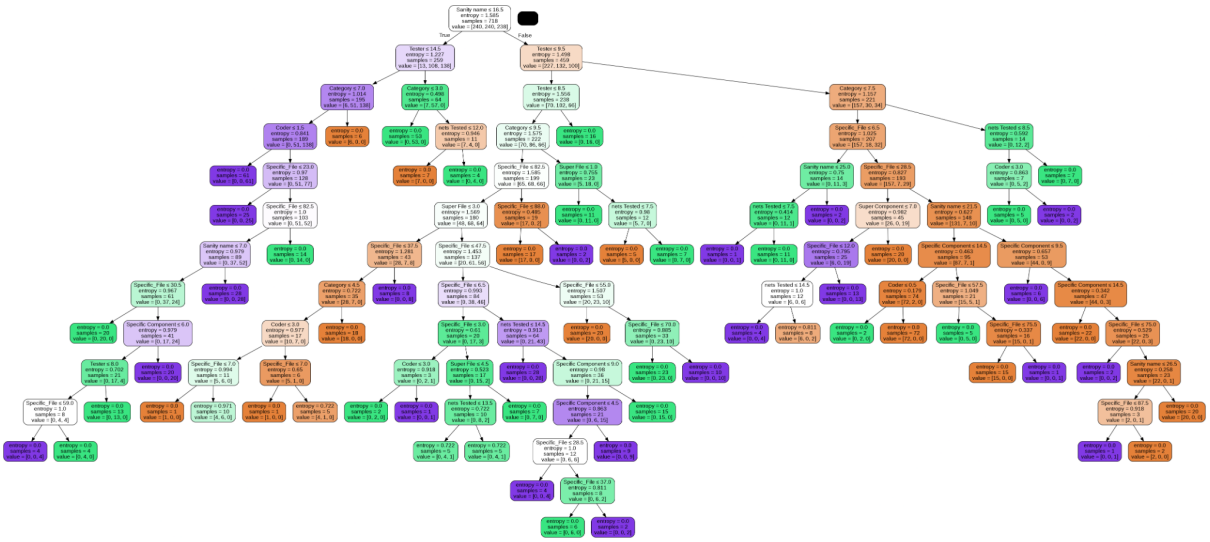
A similar plot to analyze the time taken by each classifier, is also plotted.



In practice, the decision tree and LightGBM both give similar results but Decision Tree is chosen as our final model as it takes much less time for computation as compared to LightGBM as shown by lazy prediction.

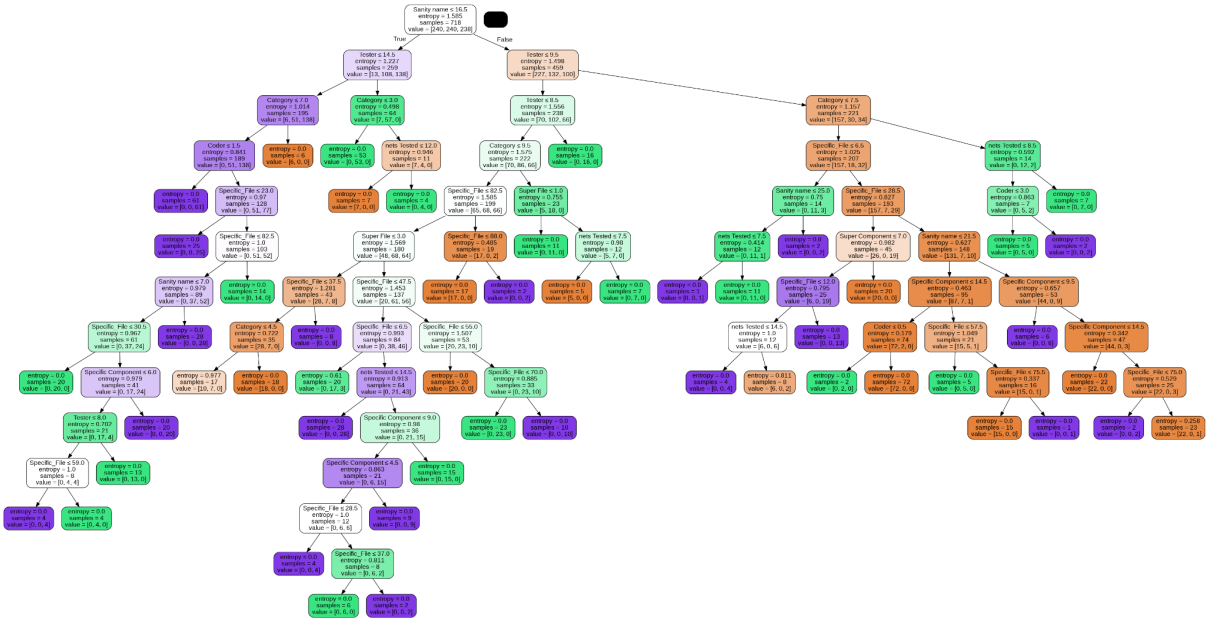
Following is a Decision Tree Plot showing the decision tree structure behind our model.

This visualization reinstates that the Sanity name and Tester are some of the most important features in our dataset.



POST PRUNING COMPARISON

Decision Tree Classifier is implemented initially without pruning it to see how it fits the data and then performed Post Pruning Alpha Cut to compare it with the original non pruned model. After pruning our testing accuracy drops to 96.56 which is less compared to our non pruned tree which gives test accuracy of 97.07.



MODEL PERFORMANCE

Moving on to the model performance, it is observed that the Accuracy score, Recall Score have minimum false positives and the F1 score for the model is at an impressive 97.08% on the testing data and 98.75% on the training data.

A confusion matrix is a table that is used to define the performance of a classification algorithm. The confusion matrix visualizes and summarizes the performance of a classification algorithm.

SHAP

SHAP helps to understand how each feature contributes to the prediction made by the model and can be used to identify biases in the model and to improve its interpretability.

The force plot is another way to see the effect each feature has on the prediction, for a given observation. In this plot, the positive SHAP values are displayed in red color and the negative in blue color, as if competing against each other.

While hovering over the target label of failure it can be seen how different features affect the probability of a decision.

The effect of specific files affects the decision like sis sis_stub for 0, sat for 1, cfm for 2

Summary plot is then used to find out the feature importance of all columns

Hence, a well-working model has been created to classify whether a sanity test will pass, fail or abort. Also, the model has been enhanced to identify the scenarios which have a very low probability of failure for the change in a particular set of files.