CMPE 257 - Machine Learning

Predictive Equipment Failures — Predict downhole equipment failures using sensor data

Team – 6

SAN JOSÉ STATE UNIVERSITY

Submitted to

Dr. Mahima Agumbe Suresh

Vineet Samudrala	017426253	vineet.samudrala@sjsu.edu
Satwik Upadhyayula	017423796	satwik.upadhyayula@sjsu.edu
Bala Supriya Vanaparthi	017464135	balasupriya.vanaparthi@sjsu.edu

Table of contents

Prerequisite Knowledge

Before we dive deep into the discussion, let us define a few terms for a better understanding of the business problem.
(Courtesy: Wikipedia)
Oil Wells: An **oil well** is a boring in the Earth that is designed to bring petroleum oil hydrocarbons to the surface. Usually, some natural gas is released as associated petroleum gas along with the oil. A well that is designed to produce only gas may be termed a **gas well**.
Stripper Wells: A **stripper well** or **marginal well** is an oil or gas well that is nearing the end of its economically useful life.

Surface Equipment: Equipments which are present on the surface like tubing head, Wellhead are surface equipment
Downhole Equipment: Equipment which are present below the surface like drilling pipes, tubular tools, and centralizers.

Background

80% of producing oil wells in the United States are classified as stripper wells. Stripper wells produce low volumes at the well level, but at an aggregate level, these wells are responsible for a significant percentage of domestic oil production. Stripper wells are attractive to a company due to their low operational costs and low capital intensity — ultimately providing a source of steady cash flow to fund operations that require more funds to get off the ground.

Business Problem

Imagine that you own a company that operates using oil wells. You are well aware that using an oil well is a more viable option to reduce the cost incurred to the company. If a Surface/downhole equipment fails it would cost a lot to the company to get it repaired because it is already near the end of its economically useful life. Wouldnt it be great if you could predict the equipment which is going to fail beforehand?
The goal of this challenge will be to predict surface and down-hole failures using the data set provided. This information can be used to send crews out to a well location to fix equipment on the surface or send a workover rig to the well to pull down-hole equipment and address the failure.

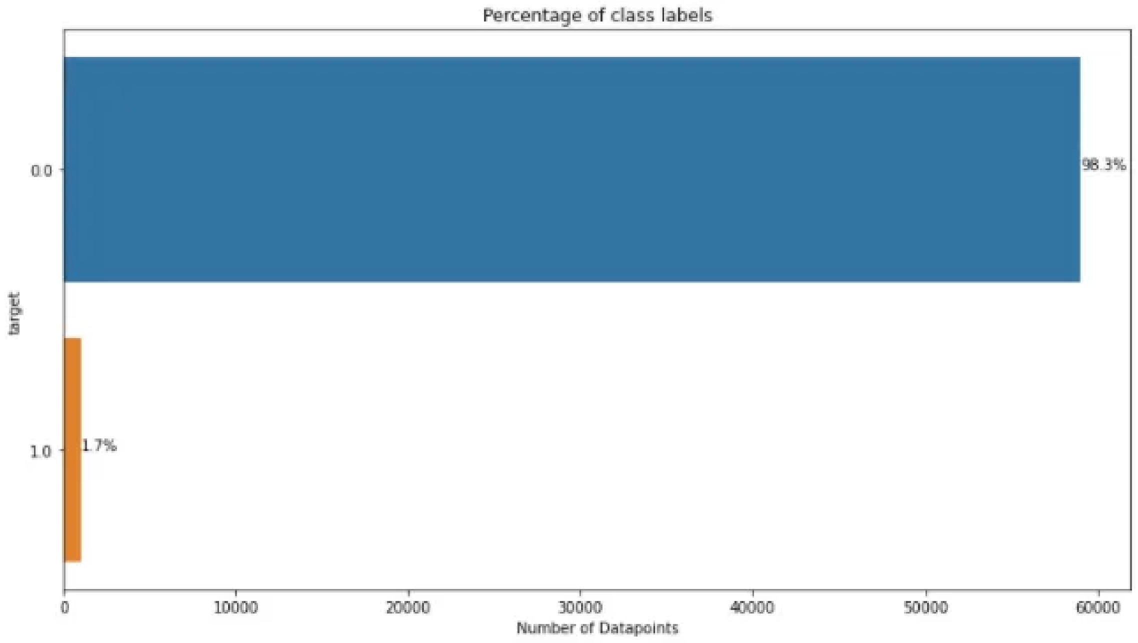
Undertsanding the data

A data set has been provided by Concophlips that has documented failure events that occurred on surface equipment and downhole equipment. For each failure event, data has been collected from over 107 sensors that collect a variety of physical information both on the surface and below the ground. You can download the dataset from here.
There are 60000 data points and 172 columns containing 107 sensors measures along with the id and target column. All of the features are numerical data.

- Iid — This is a column that is a row identifier for each row in the training and test data sets.
- Target — This column is only in the training data set to provide a label for downhole equipment failures. In the test data set, the target column is not provided to you. You must predict the target values with a predictive model of your creation. ‘1’ denotes that the equipment will fail and ‘0’ denotes that the equipment will not fail.
- There are two types of sensor columns in these data sets :
- measure columns — These columns are a single measurement for the sensor.

- histogram bin columns — These columns are set of 10 columns that are different bins of a sensor that show its distribution over time

Using the given 170 unprocessed independent numerical variables we must predict the dependent variable.



Mapping the real-world problem to a Machine Learning problem

Our task is to determine whether particular equipment is going to fail or not given the sensor values. It can have only two outcomes which are 0 or 1. So it is quite obvious that this is a binary classification problem. The performance metric used here is the F1 score.
Why choose F1 Score over other similar metrics?

- As a rule of thumb, if the cost of having a False-negative is high, we want to increase the model sensitivity and recall (In this case predicting that the equipment does not fail but it fails.
- F1 Score gives equal importance to precision and recall. We do not want the model to predict that the equipment does not fail but fails and vice versa.
- Whereas AUROC compares the True Positive Rate vs False Positive Rate. So, the bigger the AUROC, the greater the distinction between True Positives and True Negatives
- When we have a data imbalance between positive and negative samples, we should always use the F1-score because ROC averages over all possible thresholds.
- In our case, we want to increase the Precision and Recall. Hence using the F1 score is a more viable option.

Constraints

- Minimize the costs associated with failures(Maximize F1 Score)

- There are no hard latency requirements because it is fine if you do not get the prediction within a few milliseconds. Usually, there are strict latency requirements for internet applications and services.

Why use Machine Learning to solve this problem?

Given the 107 sensor reading's a small error in calculation due to confusion in the numbers can lead to a wrong prediction. The data provided by the company for the actual failure scenarios are very less. We can create synthetic data points(SMOTE) and discard the similar sensor readings to get the required result.

Exploratory Data Analysis

Exploratory data analysis *is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.*
The Dataset only consists of numeric data from different sensors. Let's first see how the target variable is distributed.

The data is highly imbalanced!!!!

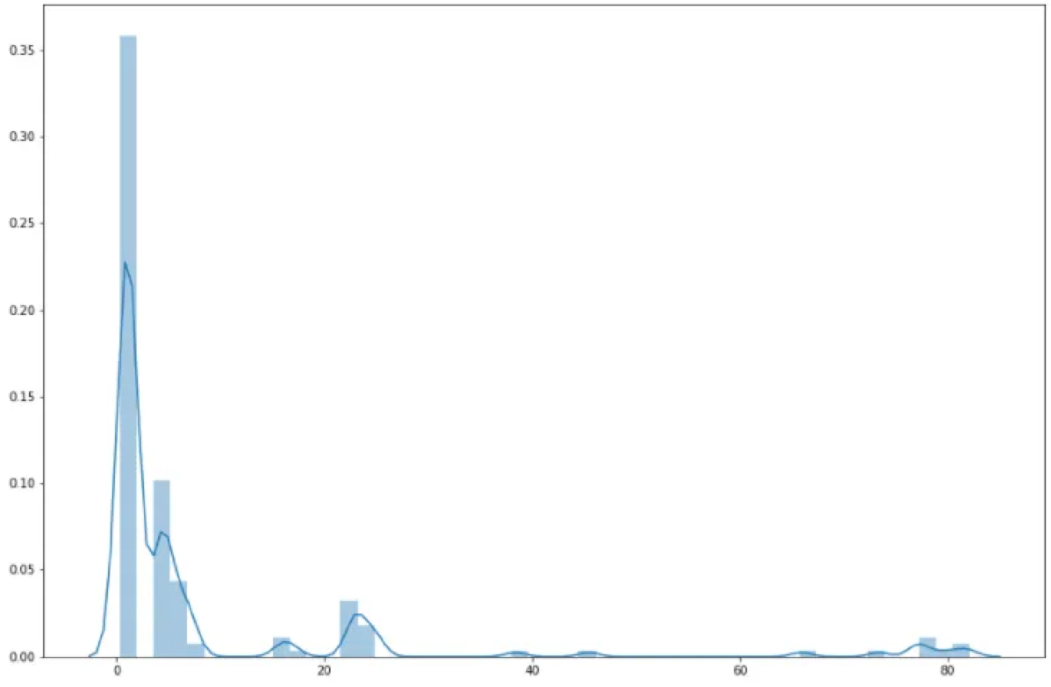
Out of the 60000 data points, only 1.7% of the points correspond to the equipment which fails.
98.3 % of the points correspond to the equipment which does not fail.
The high imbalance in the data means that we cannot use accuracy as a proper metric to determine our model's performance

Visualizing 5 rows and 7 columns from the entire data

Instead of NaN values, the dataset contains 'na' as a string so we will change the string 'na' values to numpy NaN (Null values) to perform numerical calculations and computations on it.

169 features out of 172 features are of the type "object"

Replace the 'na' values with np. Nan and convert the features to the type "Float"
The distribution of NULL values across all the features



PDF of the NULL values present in the data

- For most of the features, the percentage of NULL values are in the range of 0 to 20 percent of their data
- A very small majority of the features have NULL values in the percentage of 50 to 80 of their data.
- These features might be redundant or do not have much information which might contribute towards the training of the model.
- Those features can be dropped during the data preprocessing stage.

Univariate Analysis

- **Minimum and maximum value from all the features**

left: The plot comparing the maximum value of all the features. Right: The plot comparing the minimum value of all the features.

- Most of the minimum values are 0
- The maximum value varies from feature to feature.
- The sensor48_measure feature has a minimum value of 172.0 which is greater than 0.
- The sensor54_measure feature has a minimum value of 1209600.0 which is greater than 0

Let us dive deeper into the features with a minimum value greater than 0 to understand the significance of those features.
Box plot of the features ‘sensor48_measure’ and ‘sensor54_measure’

left: Box plot of sensor48_measure with respect to the target variables. Right: Box plot of sensor58_measure with respect to the target variables.

The box plot of the feature sensor48_measure can distinguish between the IQR of the minority class and the majority class.

- Sensor54_measure isn't able to distinguish between classes 0 and 1 for any of its data points.
- After going through its maximum and minimum value we can observe that there are only two unique values in that feature i.e 12609600 and Nan
- We can conclude that sensor54_measure does not give any valuable information as it is the same for all the data points and the remaining values are null.
- Sensor54_measure can be discarded.

Analysis of features with a maximum value beyond a given threshold

i) Box-plots of the features whose maximum values are greater than the threshold

- The features with the highest maximum values for the given threshold are able to classify most of the data points well for class 1 with minimal outliers but do not do the same for class 0
- Most of the features which are present are in the form of sensor(n) and sensor(n+1). For example — Sensor47_measure and sensor48_measure are having maximum value beyond the given threshold
- This might imply that these sensors might be placed near each other hence the readings are similar.

ii) PDF of the features whose maximum values are greater than the threshold
Most of the features are right-skewed except for sensor92_measure

Analyzing the sensor92_measure feature

The feature sensor92_measure has a lot of variability among its data points . The mean and median vary a lot which might mean that the feature might be affected by outliers or other factors.

Counting the number of Zeroes per feature

Histogram plot for the count of zeroes per feature

- Many features have a significant number of 0's.
- These 0's might imply that there aren't any notable phenomena happening around that sensor.

Multivariate Analysis

Observing the heatmaps of sensor histogram bins

Heatmaps of the random histogram bin features

- Most of the histogram bin features which are present are in the form of sensor(n) and sensor(n+1) are correlated.
- There might exist a time-based relationship among the consecutive histogram bin features

Splitting the data into train, cross-validation, and test dataset before performing pre-processing.

Perform stratify splitting to preserve the same proportion of examples in each class as observed in the original dataset.

Why split the data into train ,cross-validate, and test data? Why not preprocess the entire dataset at once?

If you do so, you are inadvertently carrying information from the train set over to the test set. This is known as data leakage. The test set should ideally not be preprocessed with the training data. This will ensure no 'peeking ahead'. Train data should be preprocessed separately and once the model is created we can apply the same preprocessing parameters used for the train set, onto the test set.

Data Pre-processing/Cleaning

Note: All the preprocessing and analysis will first be performed on the training dataset and later we will apply the same parameters on the test and cross-validation dataset.

Dropping features having more than 95% of their data points as 0

Having many zeroes implies that there is less variability among the features.

Less variability implies that there isn't much information for the model to learn from the feature.

Dropping features with less than 80 % variability

Features with less than 80 variability

We have observed that many features have most of their values as 0. Similarly, there might be features with low variance among themselves with values other than 0.

We will check each column and drop the column which has many repeating values or features with less variance as they do not contribute much towards training the model.

Dropping duplicate rows

Remove the duplicate data points

Remove the "id" feature

'ID' won't contribute towards the models performance.

Dropping features with more than 75% of their values as NULL

Handling Nan values using Median imputation

Why median imputation?

To reduce the effect of extreme values and outliers present in the data

Plotting the PDF before and after median imputation for a few features.

Features with more than 10% change in their standard deviation after Median imputation

- We can observe that the median imputation is fine for most of the features except for a few of them
- There is a significant change in the standard deviation of a few features after median imputation as shown above.
- For features with less than 10% change in their standard deviation, we shall impute the missing values with their median. For the remaining features, we shall try another approach.
- We do not want the variance to change significantly after imputation.
- Change in variance or standard deviation means that the feature is not the same as before.

Handling Nan values using Random imputation

For the features with a significant change in variance after median imputation, let us try imputing those features with Random values from the same feature

Plotting the PDF before and after Random imputation for a few features

Features with more than 10% change in their standard deviation after Random imputation

After random imputation, there are three features whose change in standard deviation is above 10%.

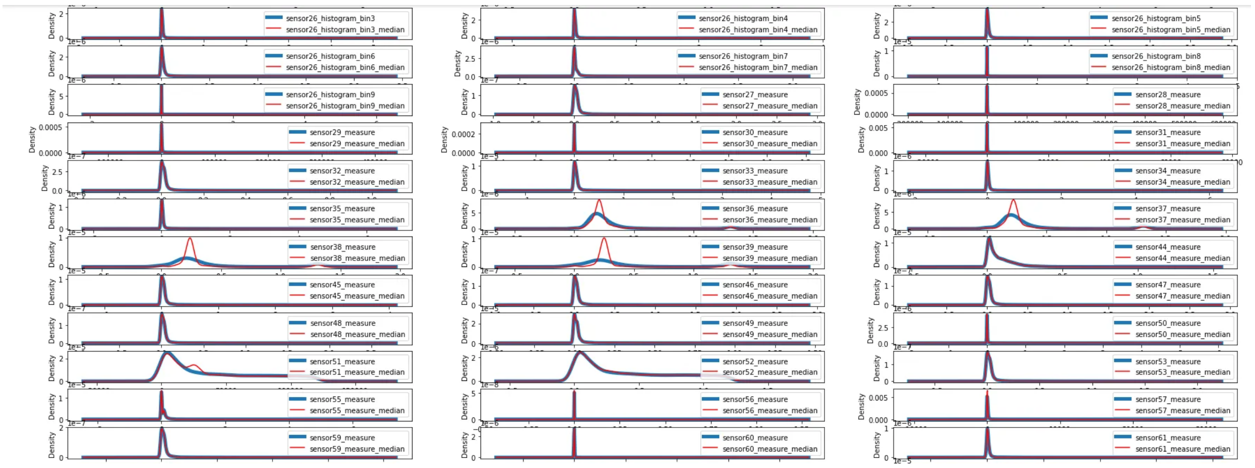
We shall try a different imputation technique for these three features

Handling Nan values using Standard deviation imputation

There isn't any significant change in standard deviation after imputing the missing values with the features standard deviation value.

We have used three imputation methods across all the features:

- Median Imputation
- Random Imputation
- Standard deviation Imputation



Dropping features with high correlation and feature selection using Random Forest

- There might be many correlated features in the dataset but dropping all of them will lead to some loss of information. This information might be crucial to the model's development.
- We shall drop the correlated features based on least importance using Random Forest Classifier

Dropping features with a correlation greater than 80% by using Random Forest Classifier

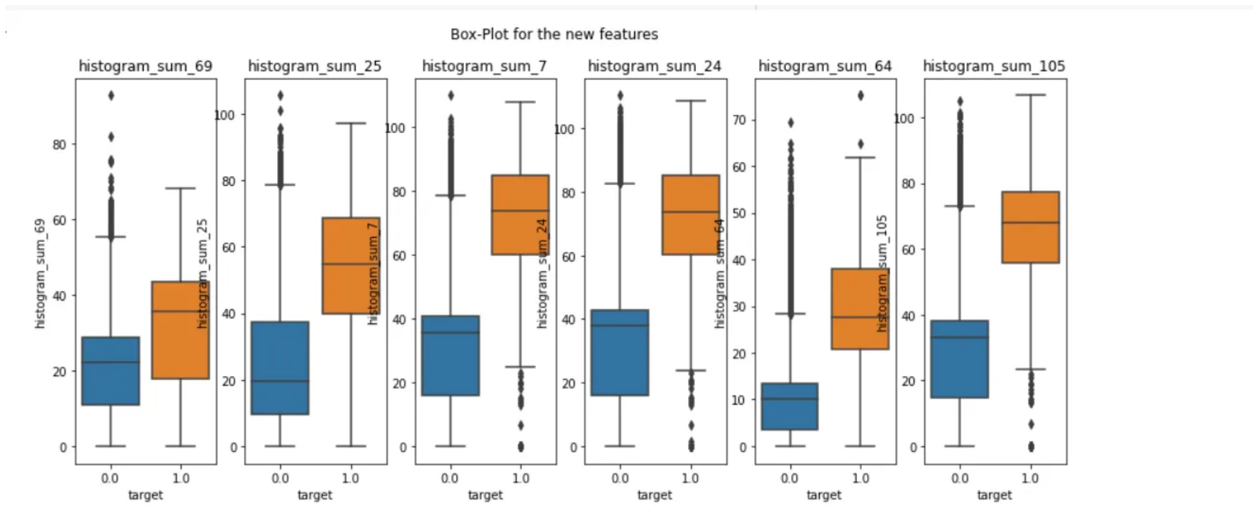
Features with higher importance among the highly correlated features

Among the highly correlated features, we shall discard the features which are not considered to be important by the Random Forest Classifier.

Feature Engineering

Logarithmic Transform on the top 5 features with the most number of unique values
Find the features with the most number of unique values
Box-plot for the new features
From the new features, we can observe that in all cases the least value of class 0 is always less than the least value of class 1. This information might be useful in distinguishing between the classes.
Exponentiation of the Sum of histogram features belonging to a sensor
Box-plot for the new features

- From the new features, we can observe that in all cases the maximum value of class 1 is always greater than the maximum value of class 1
- We can observe the new features are able to distinguish one class from the other quite decently.



Preparing Data for the model

Using SMOTE and random-under sampling on train dataset to counter the imbalance
A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn the decision boundary.
One way to solve this problem is to oversample the examples in the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model.
An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class. This is a type of data augmentation for tabular data and can be very effective.
Perhaps the most widely used approach to synthesizing new examples is called the **Synthetic Minority Oversampling Technique** or SMOTE for short.
Oversampling the minority class by 10% and Undersampling the majority class by 10%
Normalize the data
Normalization is a technique often applied as part of data preparation for **machine learning**. The goal of **normalization** is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.
We do not want the model to get impacted by extreme values.

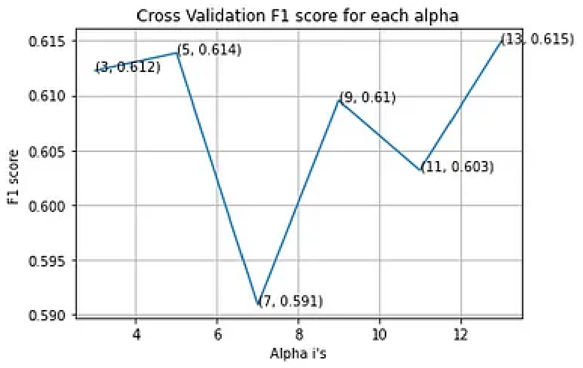
Machine Learning Models

Algorithms like Logistic Regression and Linear SVM work well for linearly separable data and algorithms like Random Forest and XGBoost work for nonlinearly separable data. As the data is numeric, using Naive Bayes would not be fruitful(Naive Bayes is used as a baseline model for text classification problems). Let us try using a few machine learning algorithms and see how well they perform on our data.

K-Nearest Neighbour

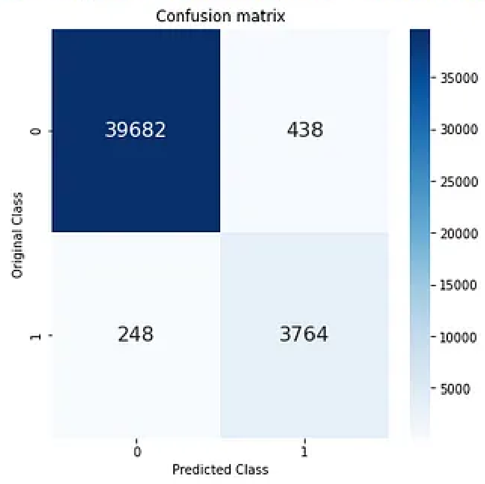
KNN can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.
Using the cross-validation dataset let us determine the best hyperparameter for training the model.
Choosing the best hyperparameter using the cross-validation dataset
Performance of the cross-validation dataset across various hyperparameter values
Confusion matrix and the percentage of misclassified points
The best Hyper parameter-K is 13
The KNN model does not perform well as it is overfitting on the training data .
It isn't performing well on the test data.

```
Best alpha (hyperparameter) : 13
```

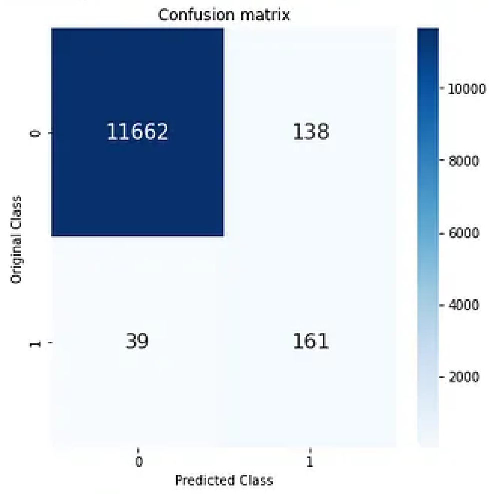


```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  0.9164840516191867
F1_score for cv data :  0.6148867313915858
F1_score for test data :  0.6452905811623246
```

```
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  1.5544276262122723
```



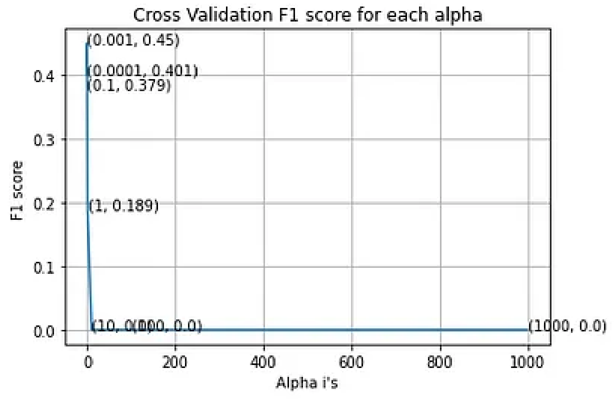
```
****************************************************************************************************
Confusion Matrix for the Test Data

Percentage of misclassified points  1.4749999999999999
```



```
****************************************************************************************************
```

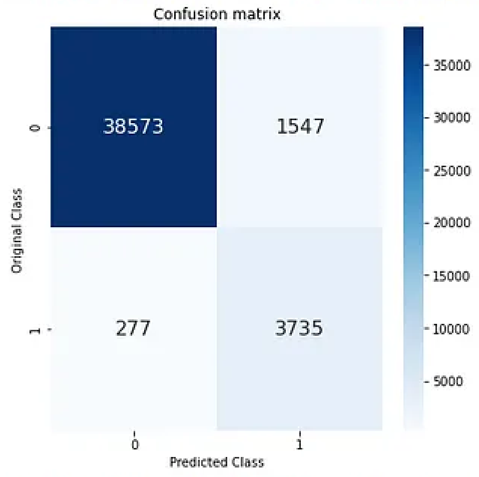
Support Vector Machine
SVM plots the features in an n-dimensional space and tries to separate the classes using margin maximizing hyperplane. SVM tries to minimize hinge loss.
Choosing the best hyperparameter using the cross-validation dataset
Performance of the cross-validation dataset across various hyperparameter values
The best hyperparameter value is 0.01
The model performs worse than the KNN model on both the train and test dataset.
So far the KNN is the best model

```
 Best alpha (hyperparameter) : 0.001
```

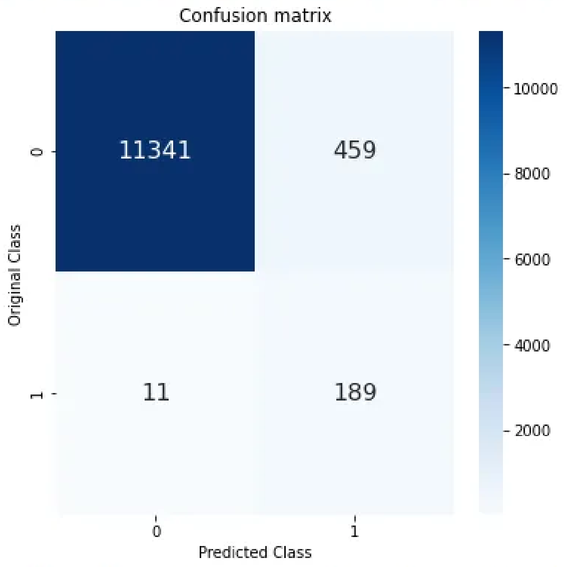


```
 Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  0.8037443511943191
F1_score for cv data :  0.4172932330827067
F1_score for test data :  0.4457547169811321
```

```
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  4.133055379316596
```



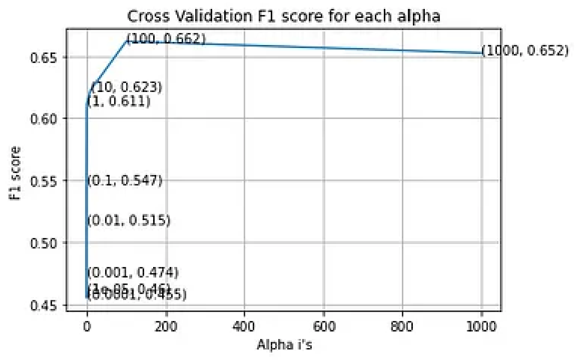
```
Confusion Matrix for the Test Data

Percentage of misclassified points  3.916666666666667
```



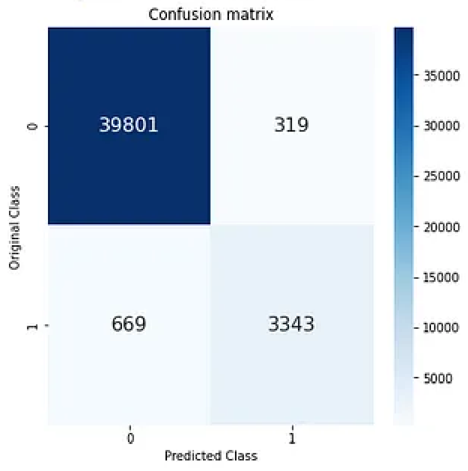
```
******************************************************************************************
```

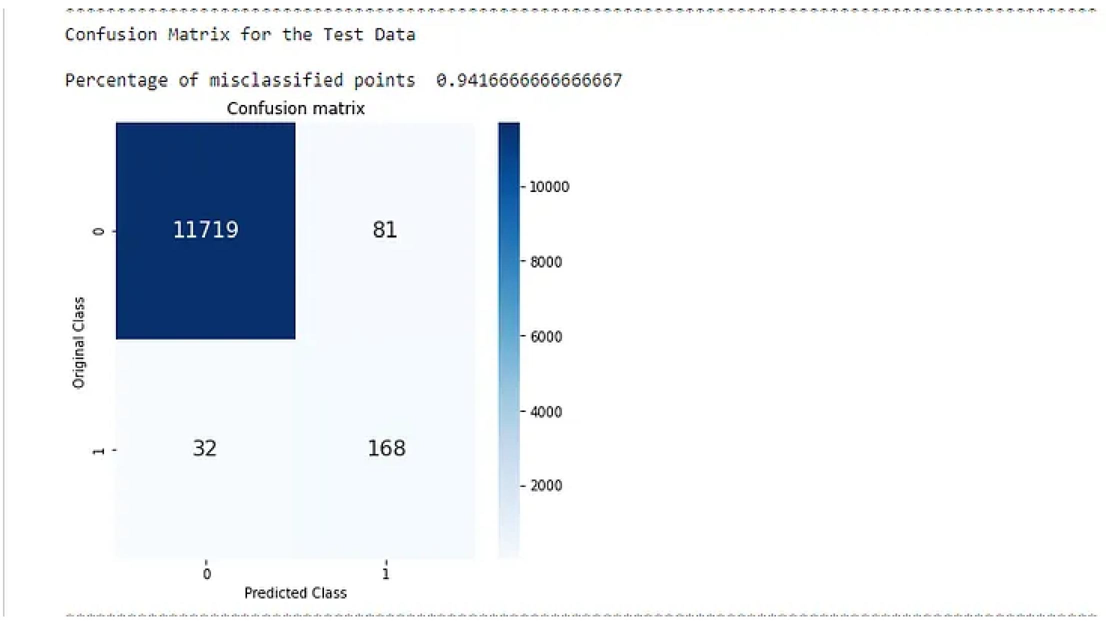
Logistic Regression
Logistic Regression is a classification technique. It classifies data into two or more categories. It gives the probabilistic interpretation because of its sigmoid function which makes it a favourite algorithm of many .C is the hyperparameter in the Logistic Regression.
Training the model
Performance of the cross-validation dataset across various hyperparameter values
The best hyperparameter value is 100
The results are encouraging with respect to SVM and KNN models.
The model performs well on the train and test dataset.
So far, Logistic regression has given the best results.

```
Best alpha (hyperparameter) : 100
```



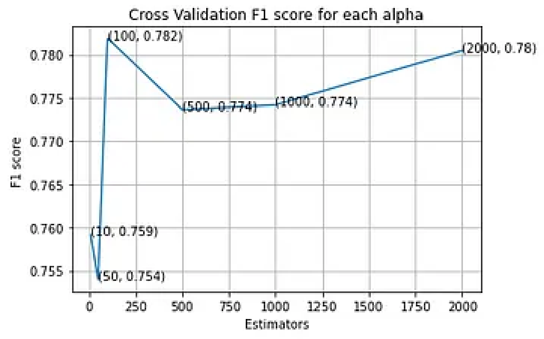
```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  0.8712535835287986
F1_score for cv data :  0.6643109540636042
F1_score for test data :  0.7483296213808462
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  2.238738330463156
```



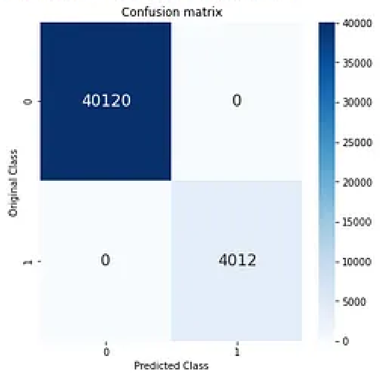

```
Confusion Matrix for the Test Data

Percentage of misclassified points  0.9416666666666667
```

Random Forest Classifier
Random forest is a bagging technique. Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we have a collection of decision trees. To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest).
Choosing the best hyperparameter using the cross-validation dataset
Performance of the cross-validation dataset across various hyperparameter values
The best hyperparameter value is 100(Number of decision Trees)
The model performs well on the train and test dataset but is overfitting
So far, Random forest performed the best

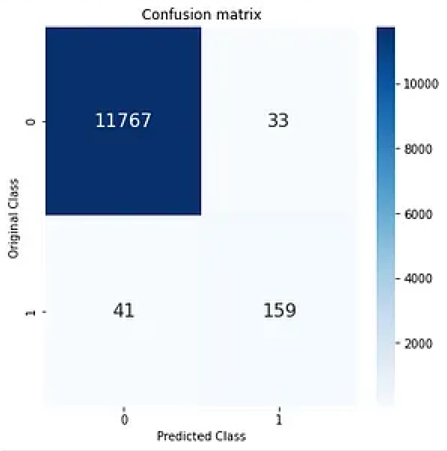
```
Best alpha (hyperparameter) : 100
```



```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  1.0
F1_score for cv data :  0.7755102040816326
F1_score for test data :  0.8112244897959185
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```



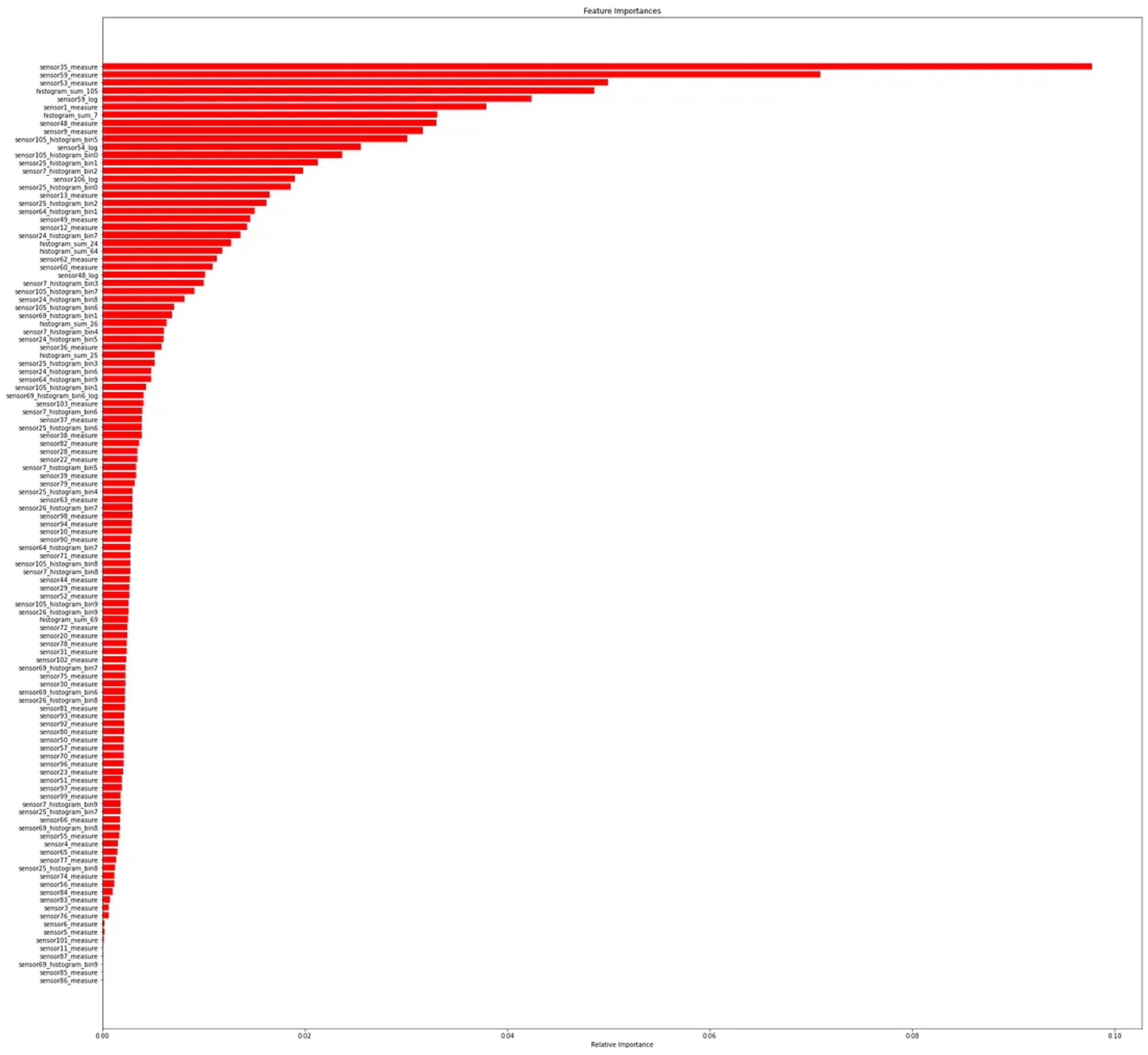
```
----------------------------------------------------------------------------------------------------
Confusion Matrix for the Test Data

Percentage of misclassified points  0.6166666666666667
```



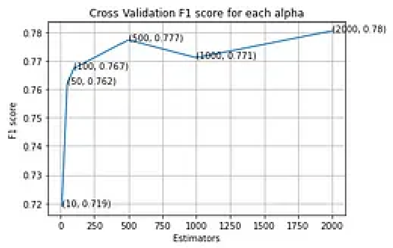
Random Forest Classifier with important Features

Relative feature importance using Random Forest

After observing the feature importance, we have discarded the least important 25 features and have trained a random forest model.
Dropping the least 25 important features
Performance of the cross-validation dataset across various hyperparameter values
The best hyperparameter value is 1000(Number of decision Trees)
The model performs well on the train and test dataset but is overfitting
The model hasn't performed better than the random forest classifier using all the features.

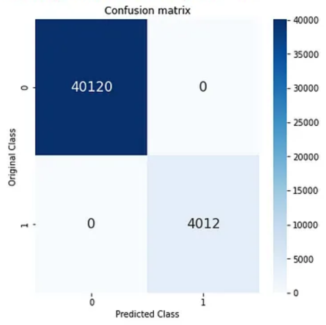


```
Best alpha (hyperparameter) : 2000
```

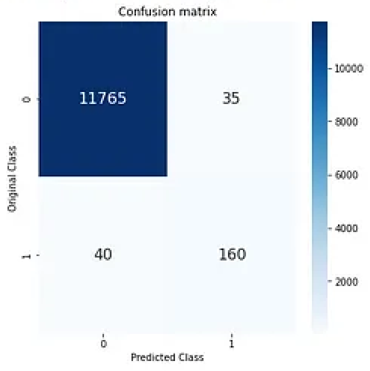


```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  1.0
F1_score for cv data :  0.7741935483870969
F1_score for test data :  0.810126582278481
```

```
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```



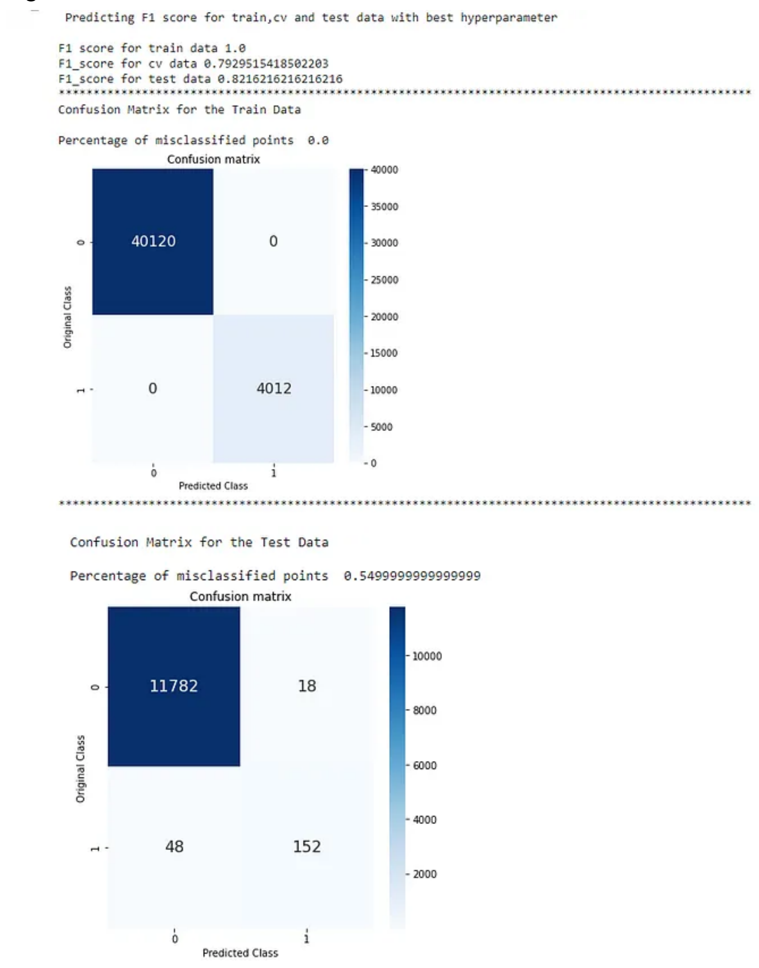
```
Confusion Matrix for the Test Data

Percentage of misclassified points  0.625
```



XGBoost using Randomized CV hyperparameter tuning

- The XGBoost stands for eXtreme Gradient Boosting, which is a boosting algorithm based on gradient boosted decision trees algorithm.
- **XGBoost** is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.
- Gradient Boosting is an approach where new models are trained to predict the residuals (i.e. errors) of prior models. A regular machine learning model, like a decision tree, trains a single model on a dataset and uses that for prediction. On the other hand, boosting trains models in succession with each model is trained to correct the errors made by the previous model.

Tuning the model with various hyperparameter values
Generally, scale_pos_weight is the ratio of the number of negative classes to the positive class.
The model overfits but the results are promising
The model performs well on the test data
This performed well than all the other models until this point
It gives a test F1-Score of 0.821

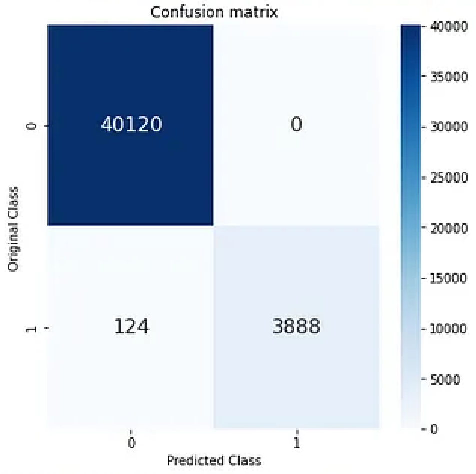


```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data 1.0
F1_score for cv data 0.7929515418502203
F1_score for test data 0.8216216216216216
*******************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```

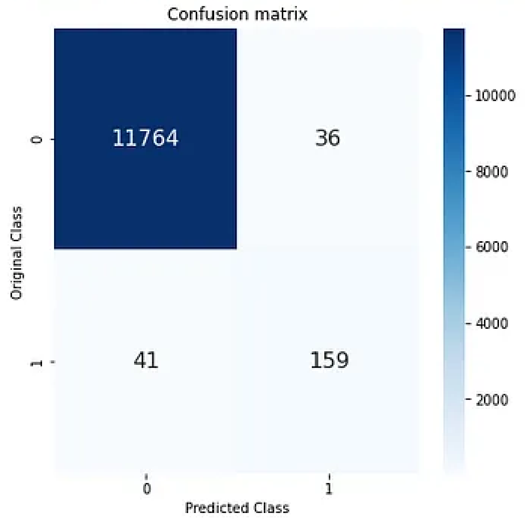
```
*******************************************************************************************

Confusion Matrix for the Test Data

Percentage of misclassified points  0.5499999999999999
```

Voting Classifier
A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output based on their highest probability of chosen class as the output.
It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each of them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.
Considering the amount of time it takes to train the voting classifier model, We can conclude that voting classifier isn't a good option.
The model doesn't overfit as much as the XGBOOST and random forest model but the performance on the test data has also reduced.
We have used all the previously trained models to build the voting classifier.
It gives a test F1-Score of 0.805

```
 Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data :  0.9843037974683544
F1_score for cv data :  0.7530364372469637
F1_score for test data :  0.8050632911392405
***************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  0.2809752560500317
```

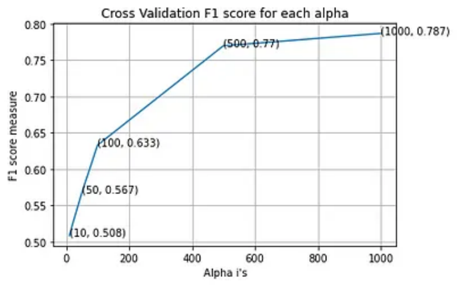


```
---------------------------------------------------------------------------------------------------
Confusion Matrix for the Test Data

Percentage of misclassified points  0.6416666666666667
```



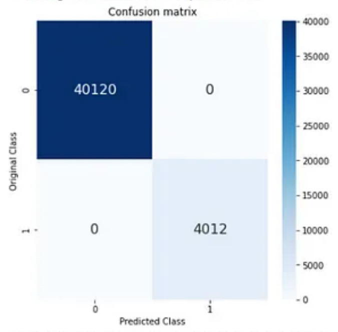
XGBoost using cross-validation dataset for hyperparameter tuning
Choosing the best hyperparameter using the cross-validation dataset. Performance of the cross-validation dataset across various hyperparameter values. The best hyperparameter value is 1000. This has provided the best performance out of all the models. The model outputs an F1-Score of 0.829 on the test data

```
Best alpha (hyperparameter) : 1000
```

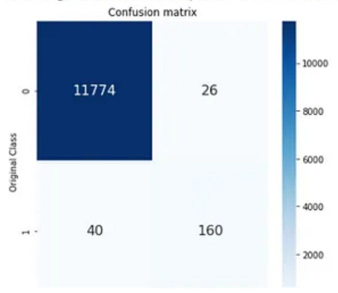


```
Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data 1.0
F1_score for cv data 0.7868852459016393
F1_score for test data 0.8290155440414508
```

```
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```

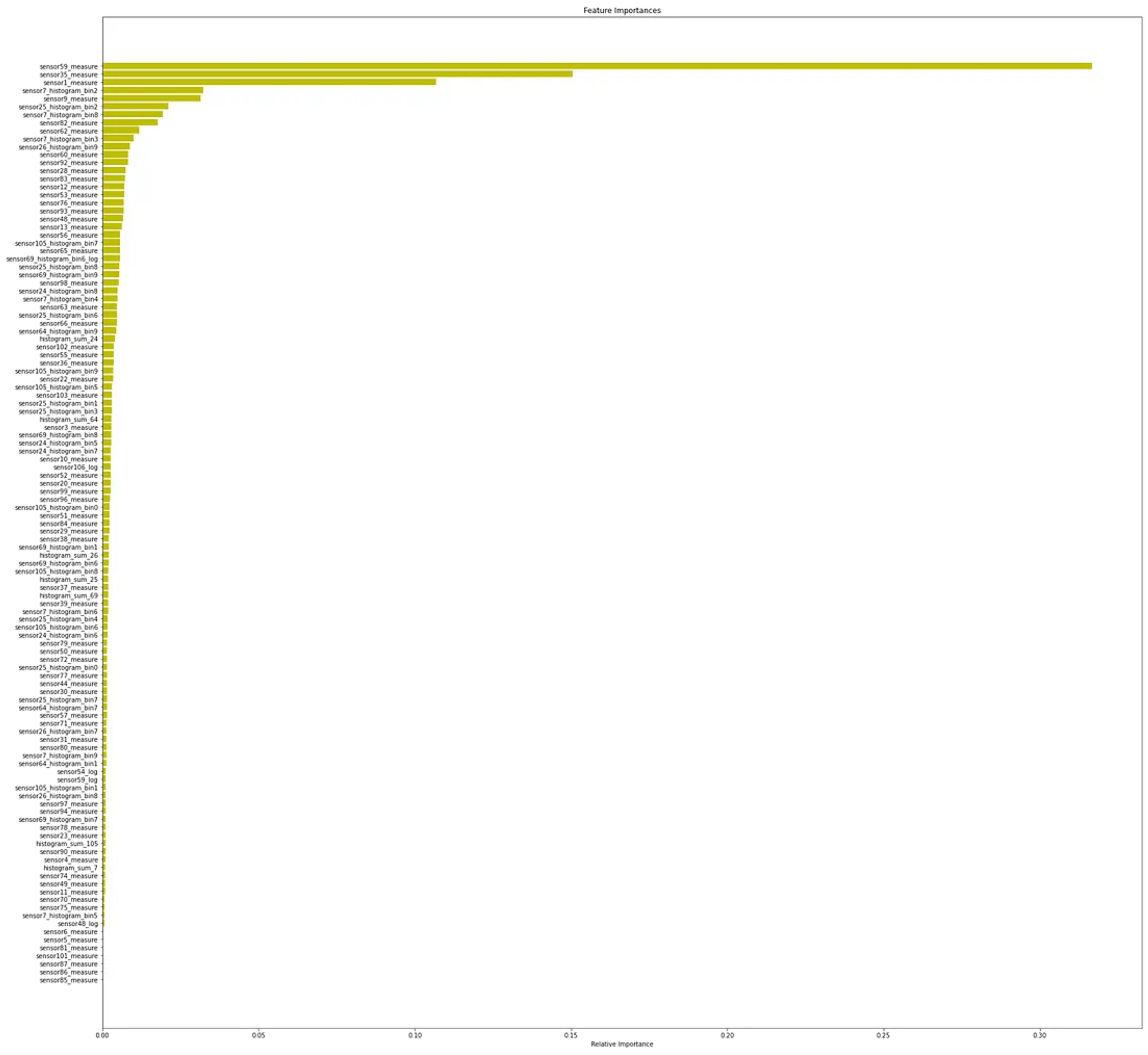


```
****************************************************************************************************
Confusion Matrix for the Test Data

Percentage of misclassified points  0.5499999999999999
```

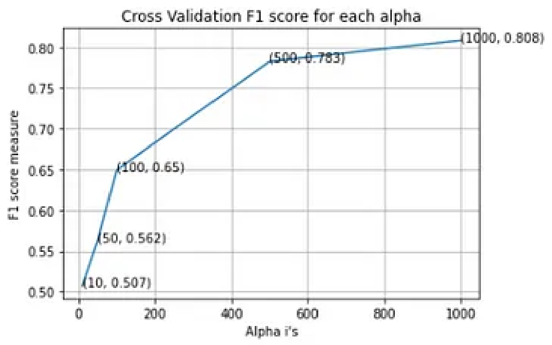


XGBoost on important features

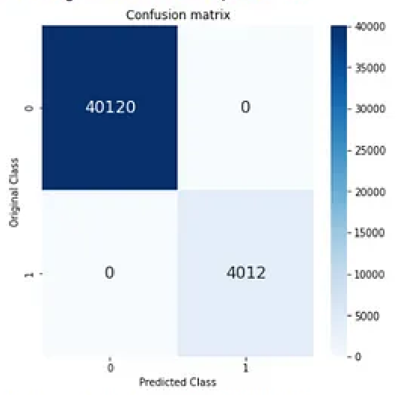
Unlike the random forest model, many features have less importance towards the model training in XGBOOST. We will discard the least 45 important features and train the model.Dropping the least 45 important features. Performance of the cross-validation dataset across various hyperparameter values There isn't any improvement over the initial XGBOOST model trained using the cross-validation dataset for training. There is a misclassification rate of 0.608 % on test data with an F1-score of 0.810.



```
Best alpha (hyperparameter) : 1000
```

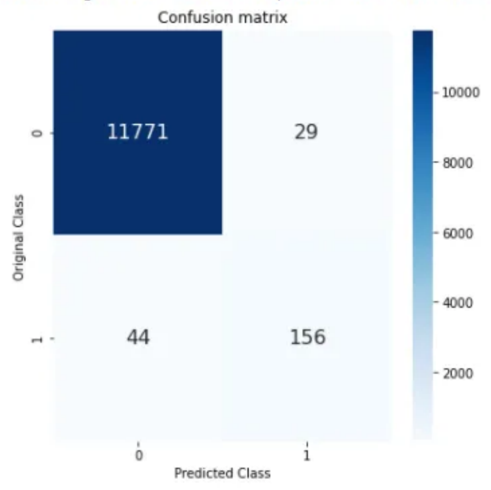


```
 Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data 1.0
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```

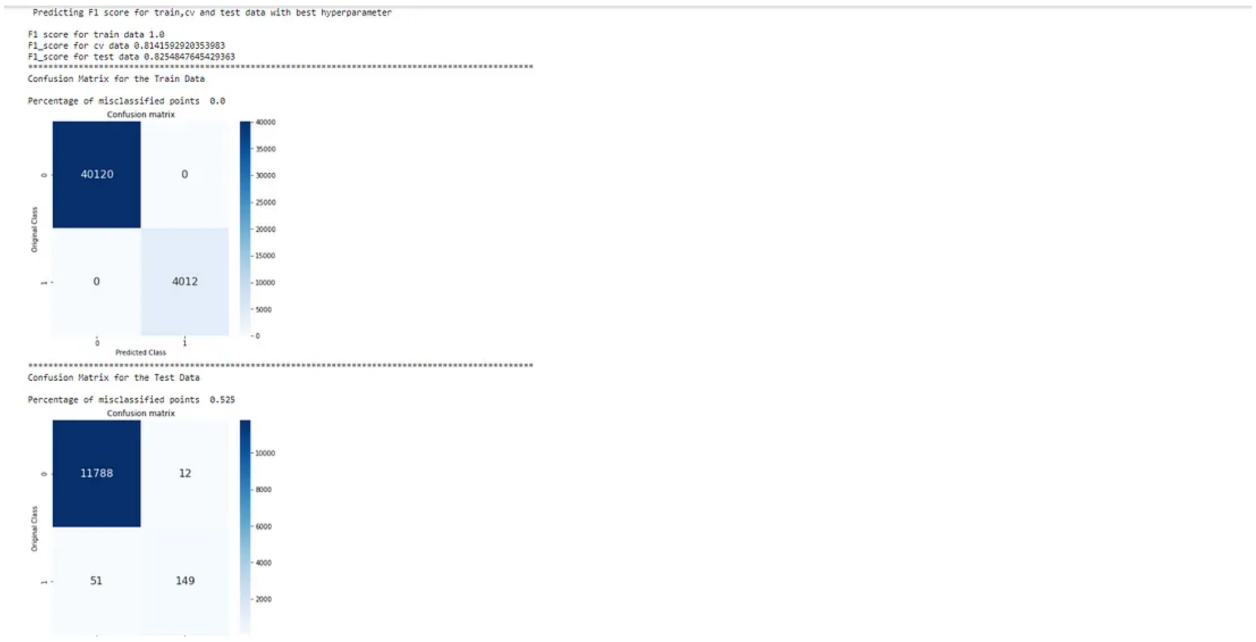


```
F1_score for cv data 0.8083333333333333
F1_score for test data 0.8103896103896104
****************************************************************************************************
```

```
 Confusion Matrix for the Train Data

 Percentage of misclassified points  0.6083333333333333
```



XGBoost with important features using Randomized CV hyperparameter tuning
Tuning the best hyperparameters based on the important features This is the second-best model after the Xgboost using Cross-validation dataset for hyperparameter tuning. It has a test misclassification rate of 0.525 and F1 Score of 0.825



```
 Predicting F1 score for train,cv and test data with best hyperparameter

F1 score for train data 1.0
F1_score for cv data 0.8141592920353983
F1_score for test data 0.8254847645429363
****************************************************************************************************
Confusion Matrix for the Train Data

Percentage of misclassified points  0.0
```

```
****************************************************************************************************
Confusion Matrix for the Test Data

Percentage of misclassified points  0.525
```

Summary

Logistic Regression:

Decision Boundary Sensitivity:

Majority Class Influence: Logistic Regression tends to learn decision boundaries based on the majority class, leading to challenges when dealing with imbalanced datasets. The model may struggle to effectively delineate boundaries for the minority class, which can impact classification accuracy.

Limited Non-Linearity:

Linear Assumption: Logistic Regression assumes a linear decision boundary. In scenarios where complex relationships or non-linearities exist, typical in imbalanced datasets, Logistic Regression may not capture the intricate patterns required to distinguish the minority class effectively.

K-Nearest Neighbors (KNN):

Sensitivity to Local Density:

Majority Class Dominance: KNN classifies instances based on the majority class within their vicinity. In an imbalanced dataset where the minority class instances are sparse, KNN may be biased towards the majority class, resulting in suboptimal performance for the minority class.

Distance Metric Impact:

Ineffective Separation: The performance of KNN is affected by the choice of distance metric. If the minority class is not well-separated in the feature space, the distance metric may not accurately capture the true relationships between instances, further impacting model performance.

In summary, the challenges faced by Logistic Regression and KNN in an imbalanced dataset are rooted in their inherent characteristics, such as sensitivity to class distribution and assumptions about decision boundaries. These factors contribute to their performance rankings in the given scenario.

XGBoost with Cross-Validation for Hyperparameter Tuning:

Challenges with Imbalanced Data:

Dominance of Majority Class: XGBoost, while powerful, may struggle with imbalanced datasets, where the majority class overshadows the minority class. Cross-validation may not fully address this imbalance, potentially leading to a model that prioritizes accuracy on the majority class.

XGBoost using Important Features with Randomized CV for Parameter Tuning:

Challenges with Imbalanced Data:

Insufficient Minority Class Representation: Despite focusing on important features, this model may still grapple with imbalanced datasets where the minority class is underrepresented. Randomized cross-validation might not consistently capture the nuances of the minority class, impacting overall F1 score.

XGBoost using Cross-Validation Dataset for Tuning:

Challenges with Imbalanced Data:

Imbalance Impact on Cross-Validation: While using the entire cross-validation dataset for tuning enhances generalization, it might not effectively address class imbalance. XGBoost may lean towards optimizing for the majority class, resulting in a model that may not generalize well to the minority class.

XGBoost using Randomized Cross-Validation:

Challenges with Imbalanced Data:

Randomness and Imbalance Interaction: Randomized cross-validation introduces an element of randomness. While it adds robustness, it may not consistently handle imbalanced datasets, potentially leading to variability in performance on the minority class.

Voting Classifier:

Challenges with Imbalanced Data:

Aggregation Bias: Voting Classifier combines predictions from multiple models. If individual models struggle with imbalanced data, the aggregated result may still be biased towards the majority class, impacting the overall F1 score.

Random Forest:

Challenges with Imbalanced Data:

Majority Class Dominance: Random Forest may be inclined to prioritize accuracy on the majority class, especially in the presence of imbalanced data. The ensemble nature may not inherently address the challenges posed by class imbalance.

Logistic Regression:

Challenges with Imbalanced Data:

Decision Boundary Sensitivity: Logistic Regression models may struggle to accurately delineate boundaries for the minority class, especially when the majority class dominates. The linear assumption may not capture the complexity of imbalanced datasets.

K-Nearest Neighbors (KNN):

Challenges with Imbalanced Data:
Sensitivity to Local Density: KNN's reliance on local density for classification may lead to biased predictions in favor of the majority class. Sparse instances of the minority class can hinder KNN's ability to effectively classify minority class instances.
The selection of the best model, XGBoost using cross-validation for hyperparameter tuning, can offer

several benefits to the application. Here's how the application can benefit from this model:
Improved F1 Score:
The model's ability to optimize hyperparameters through cross-validation enhances its F1 Score in predicting outcomes. This results in more reliable and precise predictions, contributing to the overall performance of the application.

Effective Handling of Imbalanced Data:
XGBoost has inherent capabilities to handle imbalanced datasets. Its ensemble nature allows it to learn complex relationships within the data, mitigating challenges posed by class imbalance. This ensures better performance on both majority and minority classes.

Generalization to Unseen Data:
The use of cross-validation helps in building a model that generalizes well to unseen data. This is crucial for the application to maintain predictive performance when faced with new instances or scenarios.

Robustness and Flexibility:
XGBoost is known for its robustness and adaptability to different types of datasets. This flexibility ensures that the model can accommodate variations in data patterns and continue to provide accurate predictions as the application evolves.

Efficient Use of Important Features:
The model's ability to optimize hyperparameters and focus on important features ensures that it leverages the most relevant information for predictions. This leads to more efficient use of computational resources and reduces the risk of overfitting.

Scalability:
XGBoost is designed for scalability, making it suitable for applications that may experience an increase in data volume over time. As the application grows, the model can efficiently handle larger datasets while maintaining its predictive performance.

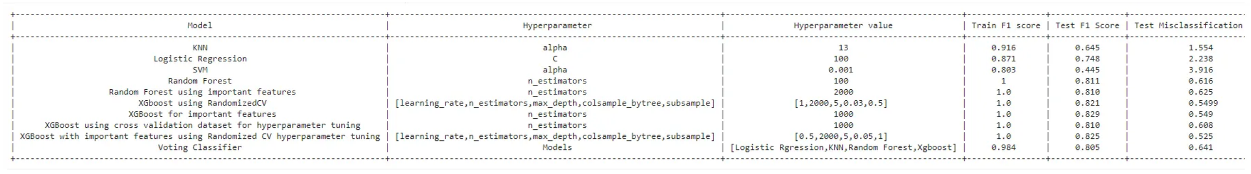
Interpretability and Explainability:

XGBoost provides insights into feature importance, aiding in the interpretability of the model. This feature is valuable for stakeholders seeking to understand the factors influencing predictions, enhancing the transparency of the application.

Adaptability to Diverse Scenarios:

XGBoost's versatility allows it to perform well across various types of problems, making it a robust choice for different scenarios. Whether the application involves classification or regression tasks, XGBoost can be adapted to meet diverse requirements.

The summary of all the models trained along with their F1-Scores and misclassification rate have been mentioned below.
XGBoost using cross-validation dataset for hyperparameter tuning has performed the best with an F1-Score of 0.829



```
+------------------------------------------------------------------+----------------------------------------------------------+-----------------------------------------+----------------+---------------+--------------------------
|                              Model                               |                      Hyperparameter                      |           Hyperparameter value          | Train F1 score | Test F1 Score | Test Misclassification
+------------------------------------------------------------------+----------------------------------------------------------+-----------------------------------------+----------------+---------------+--------------------------
|                               KNN                                |                          alpha                           |                    13                   |     0.916      |     0.645     |          1.554
|                       Logistic Regression                        |                            C                             |                   100                   |     0.871      |     0.748     |          2.238
|                               SVM                                |                          alpha                           |                  0.001                  |     0.803      |     0.445     |          3.916
|                          Random Forest                           |                       n_estimators                       |                   100                   |       1        |     0.811     |          0.616
|               Random Forest using important features             |                       n_estimators                       |                   2000                  |      1.0       |     0.810     |          0.625
|                    XGboost using RandomizedCV                    | [learning_rate,n_estimators,max_depth,colsample_bytree,subsample] |           [1,2000,5,0.03,0.5]           |      1.0       |     0.821     |          0.5499
|                   XGBoost for important features                 |                       n_estimators                       |                   1000                  |      1.0       |     0.829     |          0.549
|      XGBoost using cross validation dataset for hyperparameter tuning      |                n_estimators                |                   1000                  |      1.0       |     0.810     |          0.608
| XGBoost with important features using Randomized CV hyperparameter tuning | [learning_rate,n_estimators,max_depth,colsample_bytree,subsample] |           [0.5,2000,5,0.05,1]           |      1.0       |     0.825     |          0.525
|                        Voting Classifier                         |                          Models                          | [Logistic Rgression,KNN,Random Forest,Xgboost] |     0.984      |     0.805     |          0.641
+------------------------------------------------------------------+----------------------------------------------------------+-----------------------------------------+----------------+---------------+--------------------------
```

Future Work

One can definitely try Deep learning models to improve the performance by a great margin. Most of the dataset contained Nan values. If a proper dataset with more sensor readings had been given we could have improved the model's performance by a greater margin. We can try using RNN architecture to preserve the sequential information of the time-related histogram bin features

Google Collab Link:

Predictive_equipment_failure.ipynb

Github Repository Link:

https://github.com/SATWIK-SJSU/CMPE257_Team-Project-6