Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

Sensor Fault Detection

Rishabh Kumar, Prashant Kumar, Vaibhav Yadav, K.prabhanjan kumar
Computer science and engineering
(Noida Institute of Engineering & Technology)
Greater Noida, India

Abstract **- - Here the system we are focusing on is the Air Pressure system (APS) which generates pressurized air that is used in various functions in a truck, such as braking and gear changes. In this given dataset positive class corresponds to component failures due to particular components of the APS system. The negative class corresponds to failures not related to the APS system, Aps system uses natural air which is easily and readily available anywhere and have long term sustainability which is not present in hydraulics system when used and that's why it has an advantage over hydraulics which is also used in braking and gear changes. - The problem is to reduce the cost due to unnecessary repairs,**

INTRODUCTION

The problem is to reduce the cost due to unnecessary repairs
So it is required to minimize the false predictions that is the total cost
when a faulty truck is left unchecked or treated as not faulty and when a
perfectly alright truck is repaired without need. -This is a binary classification
problem Assume , Cost 1=10, Cost 2 = 600 - The total cost of a prediction modelis the sum of `Cost_1` multiplied by the number of Instances of type 1 failure
and `Cost_2` with the number of instances with type 2 failure, which result in
the `Total_cost`. Here `Cost_1` refers to the cost when an unnecessary check
needs to be done by a mechanic/worker at a workshop that is when the truck is not
faulty, while `Cost_2` refer to the cost of missing a faulty truck, which may cause
a total breakdown of the truck which will result in a heavy cost of repair.
- `Total_cost = Cost_1 * No_Instances + Cost_2 * No_Instances.` -
In the problem statement we can observe that, we have to reduce both
false positives and false negatives. But more importantly we have to reduce false
negatives because the total cost due to false negative is 60 times higher than
the false positives.

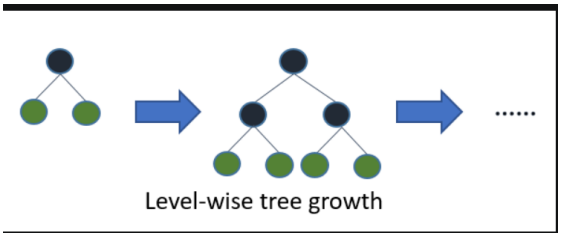
In this project we have used various ml algorithms like few gradient boosting algorthims
For ex-catboost classfier, Xgboost classifier, and lightgbm and decision trees are the
The backend of all these algorithms. In random forests, the results of decision trees are aggregated
at the end of the process, while in Gradient boosting it doesn't happen so and instead
aggregates the results of each decision tree along the way to calculate the final result.

In this project, before applying lightgbm classifier on large dataset we have performed
Hyperparameter tuning to get the best params for lightgbm to work on and give the best results which might take a lot of time because it is done using grid search cv
ich tests for each pair of values provided as a combination of values

Advantages of using lightgbm over XGboost

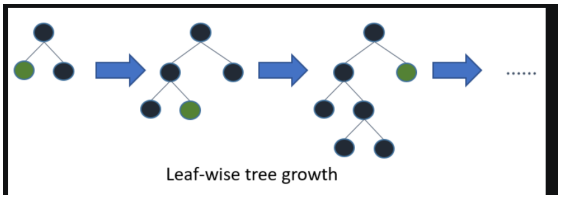
Xgboost algorithm is a type of gradient boosting algorithm that works on splitting data level wise while lightgbm is also a type of gradient boosting algorithm which works on splitting data leaf wise.

Both work well on binary classification problems and where dataset is large but lightgbm outperforms Xgboost algorithm when data is large but before that it requires some hyperparameter tuning.
Gradient boosting can be applied in 2 scenarios like
1. Regression 2.Classification



Level wise growth shown in Xgboost classifier algorithm

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023



Leaf wise growth in lightgbm classifier algorithm

Both LightGBM and XGBoost accept only numerical features . This means that the nominal features in our data need to be encoded into numerical features.

Lightgbm works better than Xgboost in case of sensor data because Xgboost by default treats the categorical variables as numerical variables with order. But lightgbm has a parameter to check whether the column is categorical or numerical and handles this issue with ease by splitting on equality

Working of Lightgbm Algorithm

Xgboost algorithm is a type of gradient boosting algorithm that works on splitting data level wise while lightgbm is also a type of gradient boosting algorithm which works on splitting data leaf wise.

In boosting we have a baseline model which predicts on some values of data say 100 values , And gives errors i.e. , e1,e2,e3 and so on and next model M1 is fed with these errors and data to work on it learns from these errors what mistakes not to repeat. This process goes on repeating till the last model. All of these internally use decision trees and splitting on basis of entrophy, gain etc but lightgbm uses binning of data features which is also used in histograms which is one of the reasons for better performance of lightgbm Secondly lightgbm uses exclusive feature bundling technique to bundle the features which takes values which are mutually exclusive so giving them unique values for ex – male -1 , female-0 will be bundled to a new feature taking values 11 or 10 based on male or female which makes it fast.

Sensor data is a case of highly imbalanced dataset so we have performed smote+tomek technique to balance the data and to oversample the minority class.

We have dropped the columns having more than 60% missing values and imputed the null values in the rest using simple imputer and the constant strategy with filling by zero in place of null.
Manually encoded positive class to be 1 and negative to be 0.

It also uses gradient based one side sampling technique in which we sort gradient data in asc order and take top 20 percent from it out and form a bin and then from left 80 percent we randomly take 10 percent data and mark it as bottom selection is one sided and hence it is named so one sided sampling technique , this feature increases its efficiency a lot and this process is repeated several times.

Advantages of using lightgbm classifier

Xgboost works faster on cpu while on gpu lightgbm performs way better than Xgboost

- Lightgbm is prone to overfitting in case of small datasets<10,000 rows while in our case dataset is fairly large with nearly 36,000 rows and 163 colums and out of these 162 are numerical while only 1 is categorical column , lightgbm has max_depth parameter to limit the tree depth but still the tree will grow leaf wise only.
- Faster training speed and higher efficiency: Light GBM uses a histogram-based algorithm i.e it buckets continuous feature values into discrete bins which fasten the training procedure.
- Lower memory usage: Replaces continuous values to discrete bins which results in lower memory usage.
- Better accuracy than any other boosting algorithm: It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy.
- Compatibility with Large Datasets: It is capable of performing equally well with large datasets with a significant reduction in training time as compared to XGBoost.

Disadvantages

- It is prone to overfitting.

Hyperparamter tuning in lightgbm

```
'num_leaves': [10,20,31,56, 127],
    'reg_alpha': [0.1, 0.5],
    'min_data_in_leaf': [30, 50, 100, 300,
400],
    'lambda_l1': [0, 1, 1.5],
    'lambda_l2': [0, 1],
    'learning_rate':[0.01,0.2,0.5,0.7]
```

Parameters are passed in a grid of params to test for the best params in case of sensor data.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 05, May-2023

Hyperparameter tuning might take a long time if performed using grid search cv since it matches all possible combinations of intermediate hyperparameters to find the one which gives best accuracy, which makes it computationally very expensive

Num_leaves*:* This is the main parameter to control the complexity of the tree , in good practice we should keep the value of this parameter to be less than (2^max_depth) in our case 56 is best value of this parameter

Min_data_in_leaf: This is a very important parameter to prevent over-fitting in a leaf-wise tree. Its optimal value depends on the number of training samples and num_leaves. Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset**,** optimal value for our case comes out to be 400

Learning_rate: If this parameter is set too high the model will skip some of the important information to learn while if kept too low it will leanr very slowly and tend to overfit default value is 0.1 while in our case value 0.7 gave the best results.
Max_depth: It is by default set to -1 , it is used to set the tree depth explicity ,-1 means to infinite depth till best results.

Conclusion and future work possible:

Lightgbm classifier gave an accuracy of 99.74% which is the highest followed by Xgboost with 99.67% accuracy and since hyperparameter tuning is a broad field, it has a huge scope for improvements.

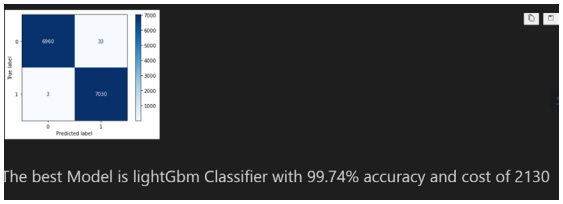
Model	Imputation_method	Total_cost
lightgbm	Simple Imputer-Constant	2130
lightgbm	Mice	4830
XGBClassifier	Knn-Imputer	4070
XGBClassifier	Simple Imputer-Mean	3420
XGBClassifier	Median	5210
lightgbm	PCA	33140

```
print("Final lightgbm Classifier Accuracy Score (Train) :", final_model.score(X_train,y_train))
print("Final lightgbm Classifier Accuracy Score (Test) :", accuracy_score(y_pred,y_test))

Final XGBoost Classifier Accuracy Score (Train) : 1.0
Final XGBoost Classifier Accuracy Score (Test) : 0.997433338086411

print("Final XGBoost Classifier Cost Metric(Test) :",total_cost(y_test, y_pred))

Final XGBoost Classifier Cost Metric(Test) : 2130
```



For better accuracy

- Use large max_bin (may be slower)
- Use small learning_rate with large num_iterations
- Use large num_leaves (may cause over-fitting)
- Use bigger training data
- Try dart

For dealing with overfitting

- Use small max_bin
- Use small num_leaves
- Use min_data_in_leaf and min_sum_hessian_in_leaf
- Use bagging by set bagging_fraction and bagging_freq
- Use feature sub-sampling by set feature_fraction
- Use bigger training data
- Try lambda_l1, lambda_l2 and min_gain_to_split for regularization
- Try max_depth to avoid growing deep tree

REFERENCES

[1] Books: • [1.1]"Fault Detection and Diagnosis in Engineering Systems" by Janos Gertler
[1.2]"Sensor and Data Fusion: A Tool for Information Assessment and Decision Making" by H.B. Mitchell
•[1.3] "Introduction to Autonomous Robots: Sensor Systems and Algorithms" by Nikolaus Correll, et al.
•[1.4] "Fault Detection, Supervision and Safety of Technical Processes" by L. Ljung
•[1.5] "Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data" by J. Nathan Kutz

[2] Research Papers: •[2.1] "Sensor Fault Detection and Identification Using Nonlinear Principal Component Analysis" by Xuebo Zhang, et al.
- [2.2]"Sensor Fault Detection in a Complex System Using Analytical Redundancy Relations" by Daniel Simon
- [2.3]"An Improved Sensor Fault Detection and Diagnosis Method Based on Dynamic Principal Component Analysis" by Qi Yan, et al.

[2.4] "Data-Driven Sensor Fault Detection and Isolation for Dynamic Systems" by Liuping Wang, et al.
[2.5] "A Review on Sensor Fault Detection and Diagnosis Methods" by Qiang Chen, et al. 3.