**Contents**

[1 Introduction 5](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177266)

[2 Workflow Overall 6](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177268)

[Application Flow 6](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177269)

[3 Data Ingestion and File Conversion 8](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177271)

[3.1Technical solution design 10](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177272)

[3.2Exceptions Scenarios 10](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177273)

[4 Exploratry Data Analysis 11](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177274)

[4.1Steps 11](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177275)

[4.2Technical solution design 11](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177276)

[4.3Exceptions Scenarios Module Wise 11](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177277)

[5 Graph-Based EDA 12](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177278)

[5.1Technical solution design 12](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177279)

[5.2Exceptions Scenarios Module Wise 13](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177280)

[7 Data Transformers( Pre-processing steps) 15](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177284)

[7.1Technical solution design 15](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177285)

[7.2Exceptions Scenarios Module Wise 15](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177286)

[8 ML Model Selection 16](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177287)

[8.1Technical solution design 16](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177288)

[8.2Exceptions Scenarios Module Wise 16](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177289)

[9 Model Tuning and Optimization 17](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177290)

[9.1Technical solution design 18](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177291)

[9.2Exceptions Scenarios Module Wise 18](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177292)

[10 Testing Modules 19](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177293)

[10.1Technical solution design 19](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177294)

[10.2Exceptions Scenarios Module Wise 20](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177295)

[11 Prediction Pipeline 21](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177296)

[11.1Technical solution design 22](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177297)

[11.2Exceptions Scenarios Module Wise 23](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177298)

[12 Deployment Strategy 24](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177299)

[12.1Technical solution design 25](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177300)

[12.2Exceptions Scenarios Module Wise 26](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177301)

[13 Monitoring 27](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177302)

[13.1Technical solution design 27](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177303)

[13.2Exceptions Scenarios Module Wise 27](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177304)

[14 Logging 28](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177305)

[14.1Technical solution design 28](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177306)

[14.2Exceptions Scenarios Module Wise 28](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177307)

[15 Hardware Requirements 29](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177308)

[15.1Requirements for model training 29](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177309)

[15.2Requirements for model testing 29](file:///C:\Users\Lenovo\Downloads\AutoNeuroLLD.docx#_Toc44177310)

**1.Introduction**

**Problem Statement**: We have been given a total of 54 attributes, these attributes contain Binary and Quantative attributes , and we to predict which Forest Cover-Typeis it from the given features. Each observation is 30m x 30m forest cover type determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from the data originally obtained from US Geological Survey (USGS) and USFS data.

The target variable has 7 different classes hence making this a Multi-Class Classification problem.

let’s look at the names of these types of Forest Cover:

1Spruce / Fir

2Lodgepole Pine

3Ponderosa Pine

4Cottonwood / Willow

5Aspen

6Douglas-fir

7Krummholz

**Dataset Info:**

No. of Instances581,011

No. of Attributes *(*Features*)1* 54

AssociateTaskClassification Dataset

CharacteristicMultivariate Attribute CharacteristicCategorical, Integer

Missing ValueNone

AreaLife

Target VariableForest Cover Type

**Attribute Info:**

Given is the attribute name, attribute type, the measurement unit and a brief description

No. of Attributes consists of :10 Quantative variable, 4 Binary Variable (Wilderness Area) and other 40 Binary Variable (*Soil Type*). Which makes a total of 54

**Variable/Features/Attributes.**

A screenshot of a cell phone

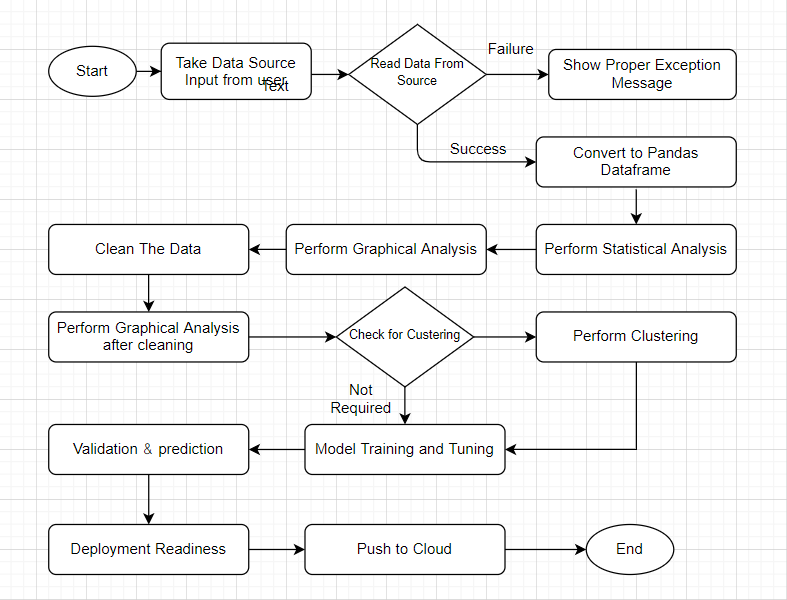
Description automatically generated

Dataset Link: [Download](https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz)

Downloaded From UCI: [Visit Site](https://archive.ics.uci.edu/ml/datasets/covertype)

**2. Workflow Overall**

**Application Flow**



**3.Data Ingestion and File Conversion**

* 1. **Raw Data Validation :** After Collect the data from the clientfirst we perform data validation, in the data validation we follow the certain type of steps like

1. sample file name validation
2. number of columns
3. name of the column’s length of date stamp in file,
4. length of time stamp in file.
5. Missing value in whole column
6. Data type of column
   1. **Data Storage in Database:** After the data validation we store the data into the data bases like SQL databases and No-Sql databases for integration of data.

In our project we have used No-Sql database Mongo Db. In the data base server, we have created train and test database and for each and every data base we have define a collection.

A screenshot of a cell phone

Description automatically generated

## 

## **A screenshot of a cell phone Description automatically generated**

## **Exceptions Scenarios**

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| User gives Wrong Data Source | Give proper error message | Ask the user to re-enter the details |
| User gives corrupted data | Give proper error message |  |

**4.Exploratry data analysis**

1. **Data Profiling**

After reading the data, automatically the following details should be shown:

1. The number of rows
2. The number of columns
3. Number of missing values per column and their percentage
4. Total missing values and it’s percentage
5. Number of categorical columns and their list
6. Number of numerical columns and their list
7. Number of duplicate rows
8. Number of columns with zero standard deviation and their list
9. Size occupied in RAM
10. **stats based analysis**
11. Column contributions/ importance
12. Check skewness of data
13. Check correlation between the variable
14. Check mean ,median, mode, standard deviation and variation of the variable

Elevation Aspect Slope Horizontal\_Distance\_To\_Hydrology Vertical\_Distance\_To\_Hydrology Horizontal\_Distance\_To\_Roadways Hillshade\_9am Hillshade\_Noon Hillshade\_3pm Horizontal\_Distance\_To\_Fire\_Points

count 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000 435759.000000

mean 2959.238476 155.587983 14.099321 269.421084 46.414213 2350.163928 212.146721 223.331477 142.544753 1979.963218

std 280.150241 111.904829 7.484419 212.427137 58.265370 1558.879443 26.774641 19.753319 38.242423 1323.404931

min 1860.000000 0.000000 0.000000 0.000000 -173.000000 0.000000 0.000000 0.000000 0.000000 0.000000

25% 2809.000000 58.000000 9.000000 108.000000 7.000000 1106.000000 198.000000 213.000000 119.000000 1024.000000

50% 2996.000000 127.000000 13.000000 218.000000 30.000000 1997.000000 218.000000 226.000000 143.000000 1710.000000

75% 3163.000000 260.000000 18.000000 384.000000 69.000000 3326.000000 231.000000 237.000000 168.000000 2550.000000

max 3857.000000 360.000000 66.000000 1390.000000 601.000000 7117.000000 254.000000 254.000000 254.000000 7172.000000

# **Graph-Based EDA**

1. Dist. plot for checking distribution of data

A screen shot of a computer

Description automatically generated

1. Count plot for checking count of data

A picture containing drawing

Description automatically generated

1. Box-plot for checking outliers

A screenshot of a cell phone

Description automatically generated

1. Correlation Heatmaps

A screenshot of a cell phone

Description automatically generated

# **5. Data Transformers( Pre-processing steps):**

1. **Missing value handling:**  **Handling Observation which has any Missing Values in it.** In our data set we don’t have a missing value so we don’t need to handle it…That's great!

A screenshot of a cell phone

Description automatically generated

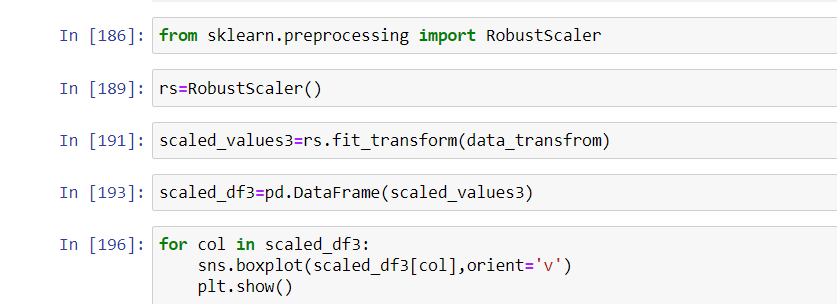
1. **Outlier handling Transform data into Box-Cox Transformation :** In our dataset we have a several Outliers so first I performed a transformation over the dataset to get a data in normal distribution or removing the outlier effect.

A screenshot of a cell phone

Description automatically generated

1. **Data Scaling/ Normalisation:**

we move to splitting our data to Train-Test Split is to scale the features to some specific range. This is called Feature Scaling. We will scale all feature values to specific like range of 0 to 1 or in standard normal distribution where standard deviation is 1 and mean is 0. but before we do this we will split the feature and target variables because we don’t want to scale our target variable. In our project we have used Robust scaler because this is robust for outliers.

****

1. **Handling the imbalance dataset:**

In the dataset target we have total 7 classes but it is imbalance we can clearly see in the count plot so we have to handle it

**A screenshot of a cell phone

Description automatically generated**

For handling a imbalanced dataset we have performed a Over Sampling technique with SMOTETomek module. from imblearn.combine import SMOTETomek

so after handling the imbalanced dataset we got the data look like this.

A screenshot of a cell phone

Description automatically generated

1. **Feature selection with Extra Tree Classifier:**

Since we already have lots of observation now to train the model, we also happen to have lots of features. This will make algorithm run very slowly, have difficulty in learning and also tend to overfit in training set and do worse in testing.To approach such a problem, we need to see how each feature has an impact on prediciting classes, and the best way to do this is by asking the models only. Classifiers like Extra Trees, Random Forest, Gradient Boosting Classifiers and AdaBoost offer an attribute called 'feature\_importance\_' with which we can see that which feature has more importance compared to others and by how much.

We have used a ETC(Extra Trees Classifier) and RFC(Random Forest Classifier) We can see that RFC and ETC show similar results, yes there are features which show-up different ranks but not of a great difference. Each feature show a little similar numbers.

| **ETC** |
| --- |
| **Elevation** | 0.197048 |
| **Horizontal\_Distance\_To\_Roadways** | 0.109549 |
| **Horizontal\_Distance\_To\_Fire\_Points** | 0.103660 |
| **Horizontal\_Distance\_To\_Hydrology** | 0.064404 |
| **Vertical\_Distance\_To\_Hydrology** | 0.059156 |
| **Aspect** | 0.052901 |
| **Hillshade\_Noon** | 0.046417 |
| **Hillshade\_9am** | 0.044147 |
| **Hillshade\_3pm** | 0.043711 |
| **Slope** | 0.038203 |

| **RFC** |
| --- |
| **Elevation** | 0.238605 |
| **Horizontal\_Distance\_To\_Roadways** | 0.117411 |
| **Horizontal\_Distance\_To\_Fire\_Points** | 0.110403 |
| **Horizontal\_Distance\_To\_Hydrology** | 0.060739 |
| **Vertical\_Distance\_To\_Hydrology** | 0.057661 |
| **Aspect** | 0.048340 |
| **Hillshade\_Noon** | 0.043726 |
| **Hillshade\_3pm** | 0.041459 |
| **Hillshade\_9am** | 0.040985 |
| **Wilderness\_Area4** | 0.035480 |

All these classifications tells us one thing in common, Numerical Features dominate when it comes to predicting forest classes. All that being said, I will now go with features that show up in the top in most classifiers. Top 20 would be a reasonable choice.

## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Data Pre-processor** |  |
| Method Name | impute\_missing\_values |  |
|  | Method Description | This method will be used to read data from a csv file or a flat file |
|  | Input parameter names | self,file\_name, header, names, use\_cols, separator |
|  | Input Parameter Description | filename: name of the file to be read  header: Row number(s) to be used as column names  names List of column names to use. If file contains no header row, then you should explicitly pass  ``header=None``.  Use\_cols: To load a subset of columns  Separator: Delimiter to use |
|  | output | A pandas Data frame |
|  | On Exception | Write the exception in the log file. Raise an exception |

## **Exceptions Scenarios**

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# **6. Model Building with Hyperparameter Optimization:**

After the pre-processing of the data we go towards to model building so we implemented a different type of models along with hyperparameter optimization but before applying the model building we make a cluster of data and apply a model building over the each and every cluster and found a model with respect to greater accuracy or a greater AUC score.

**Applying K-means :**Firstly, you will want to determine what the optimal k is given the dataset. In our case, because we used the **The Elbow Method.** Therefore, we do not need to determine the optimal k; however, we do need to identify the clusters!

Calculate the**Within-Cluster-Sum of Squared**  (WCSS) for **different values of k** and choose the k for which WCSS becomes first starts to diminish. In the plot of WCSS-versus-k, this is visible as an **elbow.**

A picture containing person, holding, air

Description automatically generated

A picture containing screenshot

Description automatically generated

And finally, we got K=3.For getting the optimal k value we

The we create 3 clusters

A screenshot of a cell phone

Description automatically generated

After the applying clustering we started a model building for each and every cluster.

We implemented two models for this study, a Bagging technique Random Forest classifier and a Boosting technique XGBoost classifier. The overall objectives of this research were to first construct the se t two predictive models, and second to compare and evaluate their respective classification accuracy or AUC score.

We perform a hyperparameter optimization technique for a model building.  **hyperparameter optimization** or **tuning** is the problem of choosing a set of optimal **hyperparameters** for a learning algorithm. A **hyperparameter** is a parameter whose value is used to control the learning process. There are various technique to choose a hyperparameter like:

1. **Manual Search**
2. **Random Search**
3. **Grid Search**

For performing hyperparameter optimization We used **Grid Search along with cross-validation. There** is class available in sklearn library called GridSearchCV

from sklearn.model\_selection import GridSearchCV

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

## **8. Model Evaluations:**

## After feeding our data and train the models to see how each model performs using 2 different evaluation technique first is **performance matrices** and second is a probabilistic method **auc-roc curve** and see which model performs the best.

**Accuracy Score:** Accuracy is the measure of the correct predicted data divided by total number of observations hence giving a value ranging between 0 and 1, while 0 is no correctly predicted class whereas 1 is all correctly predicted class. We can multiply the result by 100 to get the accuracy score in terms of percent.

**Roc-Auc Score:** A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The method was developed for operators of military radar receivers, which is why it is so named.

We have a 3 cluster, so we apply XGBoost Classifier and Random Forest Classifier each and very cluster.

But before this we have performed train\_test\_split over the data the dataset for validation of the model

**test\_size** — This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset. So in my case I have chosen 0.25 as my validation dataset and rest of the for training.

**train\_size** —This is the same as test size, but instead you tell the class what percent of the dataset you want to split as the training set. In my model building we choose 0.75 as my train data.

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size = 0.25, random state = 0)

**Cluster-0**

XGB0 Accuracy Score 0.7866507747318237

XGB0 Auc-Roc Score 0.9632743219235567

Random Forest0 Accuracy Score 0.6698450536352801

Random Forest0 Auc-Roc Score 0.9275703524425395

Final model=XGBoost0

**Cluster-1**

XGB1 Accuracy Score 0.8019281332164768

XGB1 Auc-Roc Score 0.9600465872028704

Random Forest1 Accuracy Score 0.6695880806310254

Random Forest1 Accuracy Score 0.9414377950468683

Final model=XGBoost1

**Cluster-2**

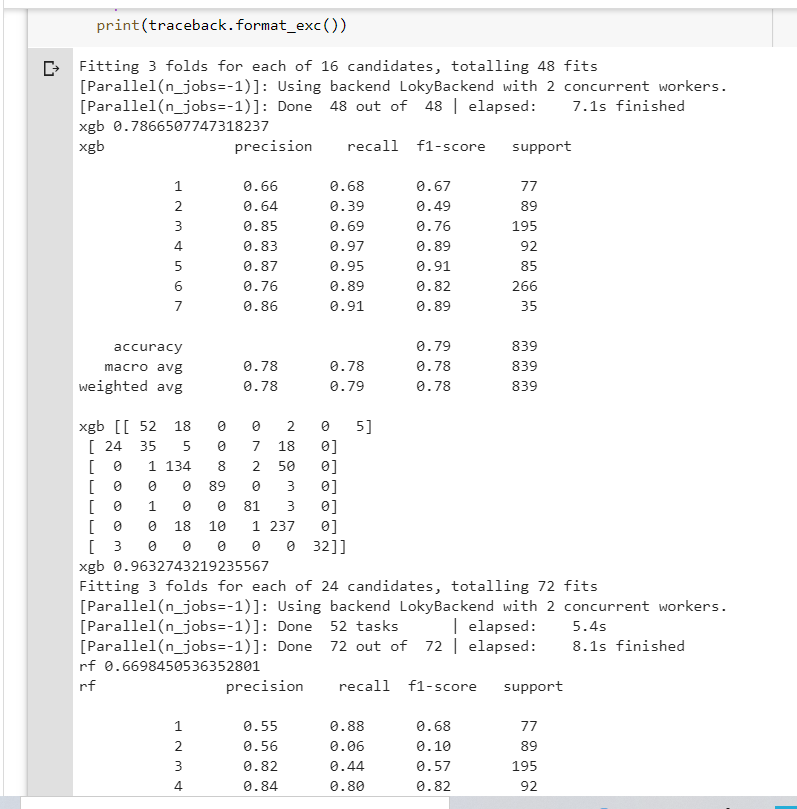
XGB2 Accuracy Score 0.7621951219512195

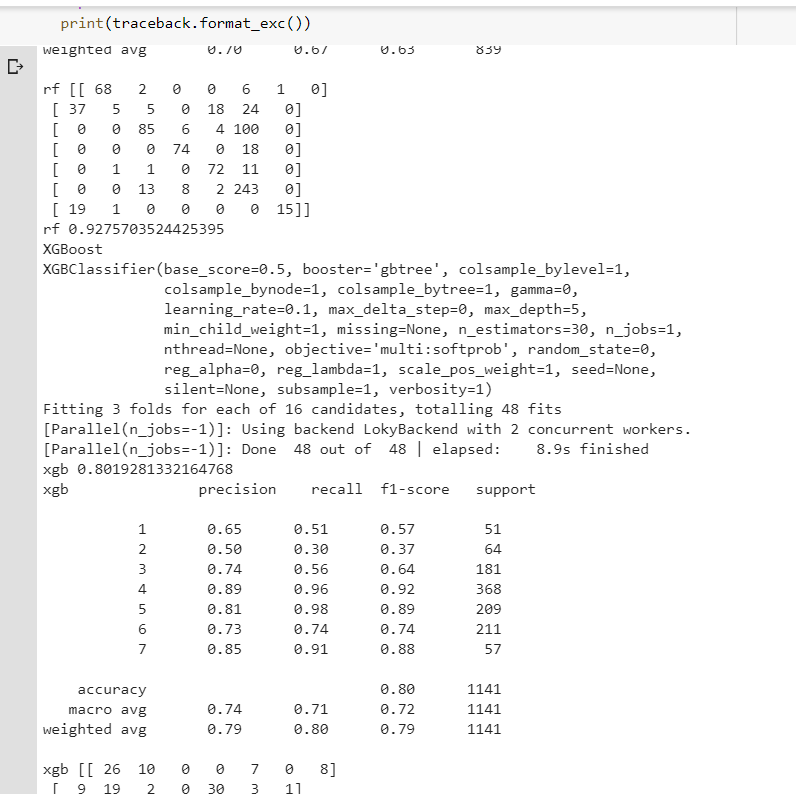
XGB2 Auc-Roc Score 0.9574533517694802

Random Forest2 Accuracy Score 0.5975609756097561

Random Forest2 Accuracy Score 0.9093521904980139

Final model=XGBoost2

****



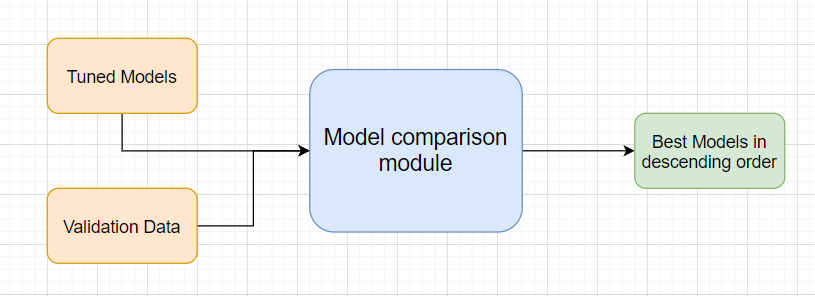
**9**.**Choosing Model:**

Out of 2 Models evaluated above model, which performs better? Lets see all the scores of all the given models scores : So to pick one model I would consider not only having best accuracy score but also having a best roc -auc score  since roc-auc socre  is more important as they give us relation between sensitivity and specificity and regarding area under curve and whatever area under curve is more we select that model.

So with that said, **I will be picking XGBoost Classifier as my final model to evaluate on the validation set and saw its performance on it.**

|  |  |  |
| --- | --- | --- |
| **Class Name** | **Model Tuner** |  |
| Method Name | get\_tuned\_knn\_model |  |
|  | Method Description | to get the hyper tuned KNN Model |
|  | Input parameter names | self, data |
|  | Input Parameter Description | Data: the training data |
|  | Hyperparameters to tune |  |
|  | On Exception | Write the exception in the log file. Raise an exception |

## Technical solution deigned

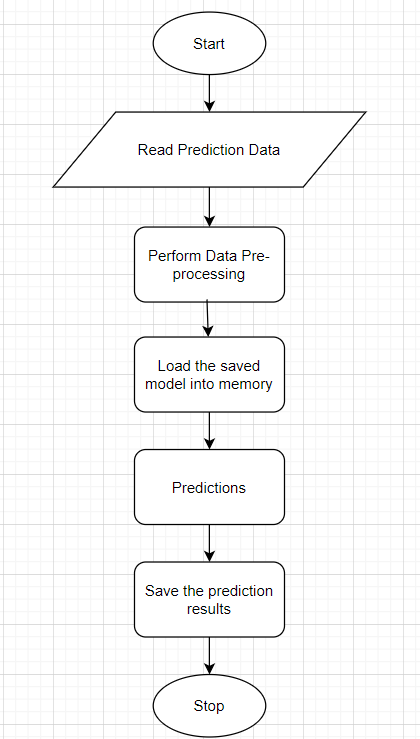


# **10.Prediction Pipeline :**

# After completion the training we go for prediction, but we can not do directly prediction because at the time of training we have follow certain pre-processing steps which is also necessary for prediction.so there are few steps

* 1. Use the validation data read modules
  2. Use the existing pre-processing module
  3. Load the model into memory
  4. Do prediction
  5. Store prediction results into the csv file(show sample predictions)

## Technical solution design



Phase 2:

UI for predictions

1.Home.html: It can be used for bulk prediction.

A screenshot of a computer

Description automatically generated

2. single\_value\_prediction.html: it can be used for single value prediction.

A screenshot of a computer

Description automatically generated

3.Result.html: it is used for showing the result or prediction. A screenshot of a cell phone

Description automatically generated

# **11.Deployment Strategy**

Take the cloud name as input

Prepare the metadata files based on cloud

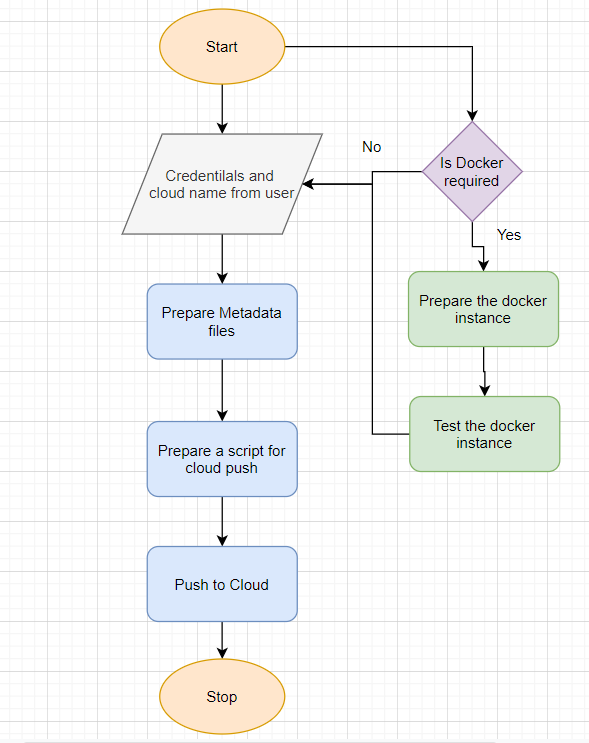
Phase 2:

Accept the user credentials

Prepare a script file to push changes

Docker instance

Push of the docker instance to cloud



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong Cloud credentials | Show error message | The user enters the correct data |
| Docker instance not working | Show error message | Fix the error |
| Cloud push failed | Show the error | Make corrections to the metadata  files |
| Cloud app not starting |  | Ask the user for cloud logs for debugging |

# **12.Logging :** we must create a logging framework for monitoring our application **logs** have been an essential part of troubleshooting **application** and infrastructure performance. They help provide visibility into how our **applications** are running on each of the various infrastructure components. **Log** data contains information such as out of memory exception or hard disk errors. There are some steps for logging.

* 1. Separate Folder for logs
  2. Logging of every step
  3. Entry to the methods
  4. Exit from the methods with success/ failure message
  5. Error message Logging
  6. Model comparisons
  7. Training start and end
  8. Prediction start and end
  9. Achieve asynchronous logging

## Technical solution design



## Common Logging Framework Code

|  |  |
| --- | --- |
| Class Name | App Logger |
| Method Name | log |
| Method Description | This method will be used for logging all the information to the file. |
| Input parameter names | self,file\_object, log message |
| Input Parameter Description | file object: the file where the logs will be written  log message: the message to be logged |
| output | A log file with messages |

# from datetime import datetime class App\_Logger: def \_\_init\_\_(self): pass

# def log(self, file\_object, log message):“””This method will be use for logging all the information to the file.””” self. Now = datetime.now() self. Date = self.now.date() self.current\_time = self.now.strftime("%H:%M:%S") file\_object. Write( str(self. Date) + "/" + str(self.current\_time) + "\t\t" + log message +"\n")

## **13.Retrainin approach:**

## In a simple word retraining means periodically train your Machine leaning model as per your need.When designing a machine learning system, it is important to understand how your data is going to change over time. A well-architected system should take this into account, and a plan should be put in place for keeping your models updated. One way to maintain models with fresh data is to train and deploy your models using the same process you used to build your models in the first place. As you can imagine this process can be time-consuming. How often do you retrain your models? Weekly? Daily?

## On the other hand, as you are manually retraining your models you may discover a new algorithm or a different set of features that provide improved accuracy.

## **Continuous learning**

## Another way to keep your models up to date is to have an automated system to continuously evaluate and retrain your models. This type of system is often referred to as continuous learning, and may look something like this:

1. Save new training data as you receive it. For example, if you are receiving updated prices of houses on the market, save that information to a database.
2. When you have enough new data, test its accuracy against your machine learning model.
3. If you see the accuracy of your model degrading over time, use the new data, or a combination of the new data and old training data to build and deploy a new model.

The benefit to a continuous learning system is that it can be completely automated.

It’s probably a still a good idea to review your process on a regular basis. As I mentioned before, you may find a different algorithm or a new set of features that improves your predictions, and this isn’t necessarily something a continuous learning system is good at.

Reference Link: [visit link](https://mlinproduction.com/model-retraining/#:~:text=Rather%20retraining%20simply%20refers%20to,should%20all%20remain%20the%20same.&text=It%20only%20involves%20changing%20the%20training%20data%20set.)

#### 14. What I learned:

1. How to research and investigate a real-world problem of interest.
2. How to perform data ingestion with no sql data base(Mongo DB).
3. How to accurately apply specific machine learning algorithms and techniques.
4. How to properly analyse and visualize your data and results for validity.
5. How to document and write a report of your work.

# **Hardware Requirements**

## Requirements for model training

The minimum configuration should be:

* 8 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor

## Requirements for model testing

The minimum configuration should be:

* 4 GB RAM
* 2 GB of Hard Disk Space
* Intel Core i5 Processor