# **PROJECT DOCUMENTATION**

Given below are the different steps performed as part of building the machine learning model for this project. The dataset that was used for training the machine learning model for the project is in the CSV file named 'incident\_event\_log\_train.csv' and the problem description is present in the file named ‘Incident\_response\_problem\_statement.txt'. Both the files are present in the GITHUB project directory.

## **Validation steps:**

Below are the validations that will be performed on the input file received from the client:

1. Check that file name format is ‘incident\_event\_log\_train.csv’
2. Check that file extension is .csv
3. Check that the total number of columns is 36
4. Check that each column names match

## **Pre-model Data pre-processing:**

The below are the pre-processing steps after the validations are completed on the input file:

1. Read csv into pandas dataframe
2. Use parse\_dates parameter if there are date features in the dataset
3. Check if dates are converted to date datatype else use pd.to\_datetime to convert into date datatype
4. Check for columns that can be deleted due to 0 standard deviation or majority null values (look for columns that can use other symbols like '?' to denote null value).
5. Delete the following columns since most of the values in these columns are irrelevant values ('?'):

cmdb\_ci : 119562 values are '?' (out of total 119998 values)

problem\_id : 118061 values are '?'

rfc : 119116 values are '?'

vendor : 119816 values are '?'

caused\_by : 119980 values are '?'

Delete the following columns as they are system created fields as they are generally automatically created by the system/db:

'sys\_created\_by','sys\_created\_at','sys\_updated\_by','sys\_updated\_at'

1. Calculate the difference between Closed date and Opened date columns and then convert this into minutes. Store this data in a new column (Time to Closure). We will use this column as the label column. During prediction, we will add the data in this column to the opened\_at date to get the closed\_at date using the below formula:

E.g formula to add minutes (if minutes is 7844) to date to get new date: df\_ir['opened\_at'][0] + pd.Timedelta(minutes = 7844)

1. Since only ‘Closed’ incident\_state rows will be used for this task (see Explanation point 1), to ensure that minimum information is lost from the other incident\_state instances of an incident, information is captured from the below columns and added as new columns to the ‘closed’ incident:
   1. ‘category’ column – Create new column ‘category\_chgd’. If the category changed for an incident, we will set the value to True else False. See Explanation point 3.
   2. ‘subcategory’ column - Create new column ‘sub\_category\_chgd’. If the subcategory changed for an incident, we will set the value to 1 else 0. See Explanation point 4.
   3. 'u\_symptom' column - Create new column ‘u\_symptom\_chgd’. If the u\_symptom changed for an incident, we will set the value to 1 else 0. See Explanation point 5.
   4. assignment\_group – similar to above
   5. assigned\_to – Similar to above
2. Creating a new dataframe with only unique Closed incidents with the highest value in the sys\_mod\_count column (Since there are some duplicate closed incidents, we will take the one with the highest modified count).
3. Remove incidents that have active = 'True' and 'Incident\_state' = Closed.

Reason: Closed incidents should not be active.

1. *Remove incidents that do not have all its sys\_mod\_count (modification count) records's. Since 7982 incidents (40% of the unique incidents in the dataset) do not have all its modified instances, we will not be performing this operation for this task.*

*Note: Information loss - Read Explanation point 2 below for more details.*

1. Create a incident\_event\_log\_feedback.csv file and copy incidents rows that are not being processed into this CSV to be provide back to the client as unprocessed records.
2. Remove outliers from the dataset: Filter incidents that took less than 100 days to close (less than 144000 minutes). They are 20491 out of 20768 records i.e more than 98.5% of the total dataset. The remaining 1.5% will be removed as outliers (Ranging from a closure of 100 to 350 days for 277 records).

## **Model specific Pre-processing steps:**

**Light GBM:**

* Replace ‘?’ by np.NaN so that they are treated by LightGBM as missing values.

**Note:**  LightGBM enables the missing value handle by default (use\_missing=True). When zero\_as\_missing=true, NA and zeros (including unshown values in sparse matrices (and LightSVM)) are treated as missing. xgboost and lightGBM both treat missing values as follows: it ignores them during split finding, then allocates them to whichever side reduces the loss the most.

* Extract day, month, weekday from opened date. Year is constant at 2016. So we will not consider the year.

Eg: df\_clsd\_uq['opened\_at\_day'] = df\_clsd\_uq['opened\_at'].dt.day

* Converting Impact, Urgency and Priority columns to numerical data since they are ordinal values. There are no null values in these columns to be handled before using the map function for this transformation.

eg: df\_clsd\_uq['impact\_new'] = df\_clsd\_uq['impact'].map({'1 - High':1,'2 - Medium':2,'3 - Low':3})

* Remove other not required columns:

drop\_list = ['number','incident\_state','active','opened\_at','notify','resolved\_at','closed\_at']

drop\_list = ['impact','urgency','priority']

df\_clsd\_uq.drop(drop\_list,axis=1,inplace = True)

Note: Notify: Notify is used only for contact type = Email (only 36 incidents). Since we are already capturing this information in contact\_type, drop this column.

* Convert categorical columns that were encoded to ‘int64’ data type to ‘int32’ datatype.

Reference: <https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html>

* To avoid target leakage (data leakage), we will first separate the label column (Time to Closure) do a train test split of the dataset (and then perform imputation of each set separately – for all other algorithms). Do a train test split and keep the test set aside for testing phase of the ML model.
* Execute the below steps on the train dataset first:
* Convert ‘object’ datatype columns (categorical columns) and other categorical columns that were created in the previous steps ('opened\_at\_day', 'opened\_at\_month', 'opened\_at\_weekday') to ‘category’ datatype columns using astype.

for col in to\_category:

X\_train[col] = X\_train[col].astype('category')

X\_test[col] = X\_test[col].astype('category')

* Categorical columns can either be marked as data type ‘category’ as we did in the previous step or we can follow the below steps as an alternative to handle categorical features in lightgbm:
  1. Convert categorical date to integer-encoded categorical features. We can use scikit learn’s label encoder for this purpose.
  2. Use categorical\_feature parameter to specify the categorical features.
  3. Note: Categorical features must be encoded as non-negative integers (int) less than Int32.MaxValue (2147483647). It is best to use a contiguous range of integers started from zero.
  4. Note2: Use min\_data\_per\_group, cat\_smooth to deal with over-fitting (when #data is small or #category is large).
  5. Note3: For a categorical feature with high cardinality (#category is large), it often works best to treat the feature as numeric, either by simply ignoring the categorical interpretation of the integers or by embedding the categories in a low-dimensional numeric space.
* Train LightGBM Regressor model:

gbm = lgb.LGBMRegressor(num\_leaves=31,

learning\_rate=0.05,

n\_estimators=20)

* Follow steps 8 and 9 on Test set and run prediction to get prediction results for Light GBM model.
* Check accuracy of model using mean\_squared\_error, RMSE etc
* **Method 2 for LightGBM:**

1. Label encode categorical columns: We will use label encoder to convert the categorical columns into numerical data. Since most of the categorical columns have significantly high cardinality, we will treat them as numeric columns after the conversion (we will ignore the categorical interpretation of the integers after label encoding)

Total number of values for caller\_id Column - 4823

Total number of values for opened\_by Column - 154

Total number of values for contact\_type Column - 2

Total number of values for location Column - 202

Total number of values for category Column - 37

Total number of values for subcategory Column - 203

Total number of values for u\_symptom Column - 338

Total number of values for assignment\_group Column - 64

Total number of values for assigned\_to Column - 185

Total number of values for closed\_code Column - 16

Total number of values for resolved\_by Column - 182

Note from official documentation: For a categorical feature with high cardinality (#category is large), it often works best to treat the feature as numeric, either by simply ignoring the categorical interpretation of the integers

1. Null values are left as if since lightGBM will handle the null values internally.
2. Label encoding steps:
   * Split data set into train-test datasets.
   * Create unique label encoder object for each categorical column in the train dataset and fit label encoder on train set only.
   * Create a dictionary to store the category value – Numerical Value pair for every category of every categorical column in the dataset. Note: We will later use this dictionary to transform the test dataset. We are using this methodology to ensure that we do not run into any issues if there are unseen categories in the test dataset. What we essential do when we transform the test categorical columns is to check if the category exists in the dictionary that we created as a key. If the category exists, then we extract the value else we input a NaN value. Light GBM will handle NaN values internally.
   * Then use the label encoder objects to transform each category of the train set.
   * Use the dictionary that was created in above step to transform the test dataset.

* **Method 3 for Light GBM**
  1. After converting the categorical features into numerical values using the label encoder (step c above) as detailed in the previous method, we will denote the categorical features using the categorical\_feature parameter when executing the fit method of the LightGBM regressor.

**Random Forest:**

1. Pre model pre-processing steps will be the same for random forest.
2. As part of pre-processing,
   1. Since all the missing values are in categorical columns, Replace ‘?’ by ‘Unknown’ so that Random forest will treat unknown values as another category.
3. After pre-processing, split the dataset into train test split.
4. Following steps on train dataset:
   1. Encode categorical columns that have a high number of categories using leave one out encoding (sample commands below):

import category\_encoders as ce

encoder = ce.LeaveOneOutEncoder(cols=lis\_X\_train2,random\_state= 42, sigma = 0.05)

X\_train\_tree1 = encoder.fit\_transform(X\_train,y\_train)

Note: sigma value of 0.05 which is the introduction of noise into the calculation

* 1. Convert bool data type columns and any other categorical columns that have few categories like opened\_at\_day,opened\_at\_month,opened\_at\_weekday into numerical using one hot encoding.
  2. Train using RandomForestRegressor

*Sample commands:*

from sklearn.ensemble import RandomForestRegressor

rnd = RandomForestRegressor(oob\_score=True,n\_jobs=-1,random\_state=10,verbose=3)

rnd.fit(X\_train\_tree1,y\_train)

* 1. Repeat steps on test dataset and predict the results.
  2. Check accuracy of model using mean\_squared\_error, RMSE etc

1. Hyperparamter tunning on random forest:
   1. Using GridSearch CV and below given parameters:

from sklearn.model\_selection import GridSearchCV

param\_grid = {'n\_estimators': [100,200,500],

'max\_features':['auto','sqrt'],

'max\_depth': [10,30,50,100],

'min\_samples\_leaf': [1, 2],

'min\_samples\_split': [2, 5]

}

**XGBoost:**

1. Use XGBoost regressor on the same dataset that was used to process random forest using the below commands (example):

import xgboost as xgb

from xgboost import XGBRegressor

xgbr = XGBRegressor(n\_jobs= -1, random\_state= 42, verbosity= 3)

xgbr.fit(X\_train\_tree1,y\_train)

1. Hyperparamter tunning on XGboost using the below:
   1. GridSearchCV:

param\_grid = {

'learning\_rate' : [0.01,0.1,0.3],

'max\_depth' : [4, 6],

'n\_estimators' : [100,200,500]

}

gridsearch = GridSearchCV(xgbr,param\_grid,n\_jobs = -1,verbose = 3)

gridsearch.fit(X\_train\_tree1,y\_train)

**SVM (Method 1):**

1. After the steps followed in random forest to encode the categorical features into numerical features using leave one hot encoding encoder and one hot encoding (steps shown below), we need to use standard scaler to scale the dataset for SVM processing.

Use LeaveOne out encoder to identify and transform all categorical columns that have more than 15 unique categories to numerical data. These columns are:

'caller\_id', 'opened\_by', 'contact\_type', 'location', 'category', 'subcategory', 'u\_symptom', 'assignment\_group', 'assigned\_to', 'closed\_code',

'resolved\_by'

**Leaveonehot encoder implementation:**

import category\_encoders as ce

encoder = ce.LeaveOneOutEncoder(cols=lis\_X\_train2,random\_state= 42, sigma = 0.05)

encoder.fit(X\_train,y\_train)

X\_train\_svm = encoder.transform(X\_train,y\_train)

For the rest of the categorical columns (which have less than 15 unique categories), use one hot encoding to transform the data into numerical data. These columns are:

# Converting bool, remaining object data type columns (contact\_type), opened\_at\_day,opened\_at\_month,opened\_at\_weekday into numerical using one hot encoding

Con\_oh = ['made\_sla','contact\_type','knowledge','u\_priority\_confirmation','category\_chngd','subcategory\_chngd','u\_symptom\_chngd','assignment\_group\_chngd','assigned\_to\_chngd','opened\_at\_day','opened\_at\_month','opened\_at\_weekday']

X\_train\_svm = pd.get\_dummies(X\_train\_svm, columns = Con\_oh)

Repeat the above steps on the test set also.

from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

X\_train\_svm\_scaled = scalar.fit\_transform(X\_train\_svm)

X\_test\_svm\_scaled = scalar.fit\_transform(X\_test\_svm)

1. We can then train the SVM model on the scaled dataset.
2. Hyper parameter tuning for SVM using the below parameters:

param\_grid = {'C':[0.1,1,10],

'epsilon': [0.1,1],

'gamma': ['scale','auto'],

'kernel': ['rbf','poly','sigmoid']

}

1. **SVM (Method 2):**
   1. Instead of converting the categorical features using leave one hot encoding, we can use one hot encoding on all the categorical features followed by applying PCA decomposition/transformation.
   2. The resultant dataset will NOT be a sparse matrix anymore.
   3. **Normalization/standardization before PCA is required:** For this scenario scaling is not required since the numerical features are all counts.

**Note:** Generally, when there are mixed datatypes in the dataset (Numerical and categorical features), then we can follow the below processes:

1) You can do standard scaler for only those columns that you want to scale (Numerical features), then perform one hot encoding on the appropriate columns (Categorical features) and then concatenate both sets of columns.

Reference: <https://stackoverflow.com/questions/43798377/one-hot-encode-categorical-variables-and-scale-continuous-ones-simultaneouely>

2) If there are many categorical features with each of these having high cardinality (number of categories are high), then we will need to perform dimensionality reduction if we apply one hot encoding on such features.

3) PCA requires normalization to be performed as a prerequisite. PCA is generally performed on numerical features.

Reference: <https://stackoverflow.com/questions/40795141/pca-for-categorical-features>

Steps:

* + - First convert all categorical columns into numerical features using one hot encoding
    - Split the dataset into train and test datasets
    - Apply PCA on train data set and then train the SVM model
    - Run prediction on test dataset by first applying PCA.

### **Explanation:**

1. The dataset has a number of rows for each incident number. For each incident number, there are multiple rows that seem to capture the life cycle of the incident (look at the incident\_state column) from the time it is created ('New' incident\_state) to the time it is closed ('Closed' incident\_state). Since our goal is to understand the time taken for final closure of an incident, we will take only one instance of each incident to learn this information. Since the maximum information is stored in the 'Closed' incident\_state for any given incident (like for eg: reassignment\_count,reopen\_count,sys\_mod\_count columns store the final count in the last state of an incident which is the 'Closed' incident state), we will take the 'Closed' instance of each incident. Since there are multiple closed records for some incidents, we will look up the sys\_mod\_count value for such cases. Due to these reasons, we will not consider any incident that does not have a ‘Closed’ state.
2. All instances (all incident\_states of an incident) of an incident should be provided so that relevant information can be captured from non-closed instances of an incident.
3. ‘category’ column - value changes for 1122 incidents. Since category can be an important field in determining the duration to solve an issue (For eg: issue can be assigned to wrong group if wrong category was chosen), we will capture this information. If the category changed for an incident, we will set the value to 1 else 0.
4. ‘subcategory’ - value changes for 1583 records. Treated similar to category.
5. 'u\_symptom' - value changes for 1608 records.

**Additional Notes:**

* To avoid target leakage (data leakage), we will first do a train test split of the data and then perform imputation of each set separately.
* Convert categorical variables into numerical variables on train set first
  1. Then apply the same steps on the test set
* Feature engineering:
  1. Capture data from other instances of the incident as new features before removing those records from the dataset.