Classification Assignment
Earthquake Data Analysis

Aditya Kumar Singh
ID: 2021701010
October 16, 2021

1 EXPLORATORY DATA ANALYSIS (EDA)

1. Remove first 8 rows to remove the title of sheet and other additional information that are basically subtitles.

2. Rename the columns (i.e., to merge two rows that consists of column names). Below is the table for column names.

Index	Column Names
1	Sl. No.
2	YEAR
3	MONTH
4	DATE
5	ORIGIN TIME-(UTC)
6	ORIGIN TIME-(IST)
7	MAGNITUDE-Mw
8	MAGNITUDE-Mw1
9	MAGNITUDE-Mb
10	MAGNITUDE-Mb1

Index	Column Names
11	MAGNITUDE-Ms
12	MAGNITUDE-ML
13	LAT (N)
14	LONG (E)
15	DEPTH (km)
16	INTENSITY-MM
17	INTENSITY-MMI
18	INTENSITY-MME
19	LOCATION
20	REFERENCE

Table 1.1: Column Names

3. Now, let's see the number of *NaN* cells or entries in the dataframe for each column.

Index	Column Names	NaN
1	Sl. No.	0
2	YEAR	0
3	MONTH	18
4	DATE	57
5	ORIGIN TIME-(UTC)	31803
6	ORIGIN TIME-(IST)	52563
7	MAGNITUDE-Mw	12054
8	MAGNITUDE-Mw1	2507
9	MAGNITUDE-Mb	40706
10	MAGNITUDE-Mb1	2492

Index	Column Names	NaN
11	MAGNITUDE-Ms	169
12	MAGNITUDE-ML	169
13	LAT (N)	0
14	LONG (E)	0
15	DEPTH (km)	2178
16	INTENSITY-MM	52948
17	INTENSITY-MMI	52989
18	INTENSITY-MME	52989
19	LOCATION	43615
20	REFERENCE	1582

Table 1.2: NaN cells in Each Column

Since shape of the dataset = **52989** × **20** and keeping an eye on percentage of NaN cells in the dataframe (i.e., **32.916**%) we couldn't remove all the rows or cols with NaN cells as that could cause whole data wipe out and hence a significant loss. So what we can do, we can drop those columns thath contains a significant amount of NaN such as: INTENSITY-MME, INTENSITY-MMI, INTENSITY-MM, ORIGIN TIME-(IST), LOCATION, MAGNITUDE-Mb, and ORIGIN TIME-(UTC) whose NaN counts are over 50%.

4. Now coming to the columns whose existence doesn't at all matters or have any relevance are the ones that provide extra information about each data point. Since our **motive** is to predict the M_w (i.e., the magnitude of earthquake), *"Sl. No."*, and *"REFERENCE"* have no role in helping out for prediction.

5. Further, columns like *"MAGNITUDE-Mb1"*, *"MAGNITUDE-Ms"*, and *"MAGNITUDE-ML"* are related to magnitude and these features are obtained only when earthquake occurs. So it doesn't make sense to use these if we were to predict earthquake in future or at some location.

6. Now, *"MAGNITUDE-Mw"* and *"MAGNITUDE-Mw1"* are the same columns. The only aspect at where they differ is the number of NaN cells. If we remove the rows from DataFrame corresponding to NaN cells of *"MAGNITUDE-Mw"* and check for equality of both columns then we got **True** flag. So, it's better to keep the *"MAGNITUDE-Mw1"* column for our prediction purpose and drop the other one.

7. **Transforming YEAR, MONTH, and DATE columns.**
   a) Removed first two rows as it contains negative year.
   b) Since there are no zero months and date, we replaced 0 with NaN.

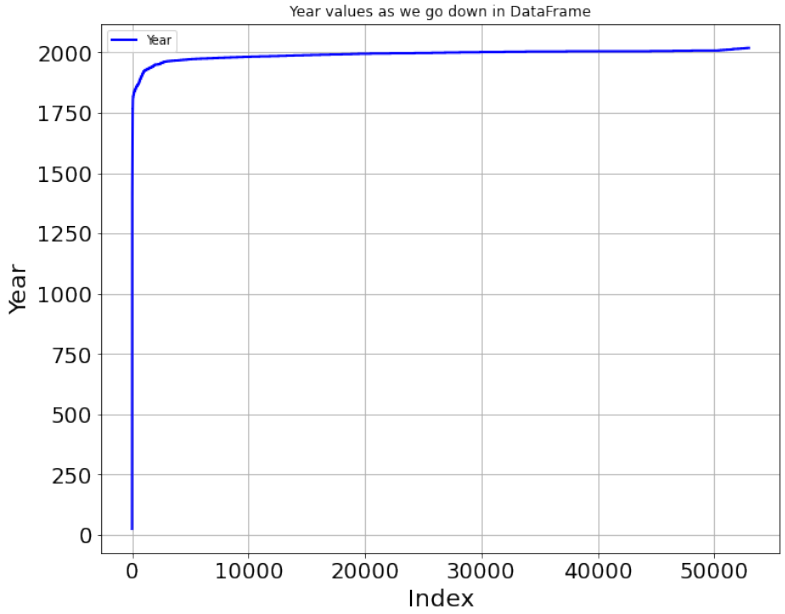

Figure 1.1: Year column plot

   c) Since the values in year column are in chronological order, we devised a method of imputaton for both date and month.
      - Put Month = 1 if no months have been registered for the same year, else put value = +1 to the the value from preceding month.
      - If preceding month = 12 for the same year, keep the current month as 12 and make one step increment in date. Follow the same procedure for date column too.
      - If date = 31 and month = 12, then change change the year itself with one step increment putting the date as 01/01/incremented year.
      - While, if date = 31 and month < 12, then the date would be 01/current_month + 1/current_year.

d) **Type Conversion for YEAR, MONTH and DATE.** Since YEAR is of *"object"*, and MONTH and DATE are of *"float64"* type, we typecast them all to *"Int32"* format.

8. **Cleaning *LAT (N)* and *LONG (E)* columns.** All entries in *LAT (N)* and *LONG (E)* columns are *string* type (i.e., object type) out of which most of them are just plain string of float type values while other cantains some special character like "N", "S", "E", "W", "-", "∘" (= degree), and " " (i.e, space character) which are pre-processed to obtain *float32* values.

9. **Cleaning and Imputing Depth Column.** Converted "0", and " " to NaN and then used **KNN**, **MICE** and it's different versions, and **DataWig** to impute Depth Column.
   - With *KNNImputer()* and *IterativeImputer()* class from *scikit-learn* package we implemented 4 different versions of imputaion with 3 versions of *IterativeImputer()*.
   - *KNNImputer()* imputes new sample by finding the samples in the training set "closest" (in terms of "Euclidean" distance) to it and averages these nearby points to fill in the value.
   - While the *IterativeImputer()* refers to a process where each feature is modeled as a function of the other features, e.g. a regression problem where missing values are predicted.
     - Each feature is imputed sequentially, one after the other that allows to use the prior imputed values as a part of a model in predicting subsequent features.
     - Sinec this process is repeated multiple times it is iterative in nature, allowing ever improved estimates of missing values to be calculated as missing values across all features are estimated.
     - This approach may be generally referred to as fully conditional specification **(FCS)** or multivariate imputation by chained equations **(MICE)**.
   - Different regression algorithms can be used to estimate the missing values for each feature, and hence we used 3 different regressors, namely:
     a) *BayesianRidge()*, the default one.
     b) *ExtraTreesRegressor(n_estimators=50, random_state=0)*.
     c) *RandomForestRegressor(n_estimators=50, random_state=0)*
   - *DataWig()* is a neural-network model that predict the imputed values using other features. This method proved to be inefficient for our dataframe and got *earlystopped* at 5^{th} epoch due to divergence.

10. Remove duplicate rows present in the dataframe, if any. And finally save all four dataframes.

2 Model Training with Hyperparameter Tuning & Cross-Validation

1. Loaded all four dataframes with different imputation methods.

2. To decide the threshold **T** for *Magnnitude* column, we scaled up the [0, 1] interval to the "range" of magnitue column (i.e., max($magnitude$) - min($magnitude$)) and shifted it by a constant = min($magnitude$). Now as **0.5** is a safe choice in [0, 1] interval,

correspondingly we chose that value in scaled and shifted interval as our threshold (i.e., = 5.3914). Again our choice for such threshold is motivated by the fact that distribution of *Magnitude* is mostly **gaussian** with a little skew towards left (**positively** skewed). So choosing a value in between would do the needful.

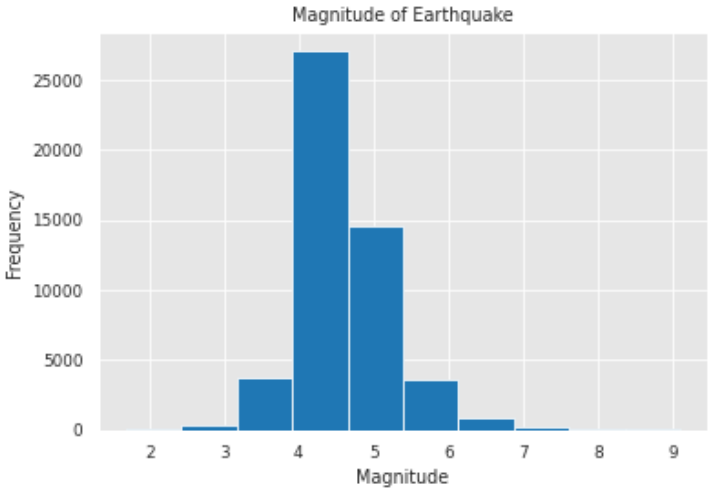

Figure 2.1: Magnitude Hitogram

3. Now split the dataset into "train_dev" and "test" set. Choose those instances whose *Magnitude* is NaN for "test" while the rest for "train_dev".

4. For each dataset (total 4), we performed *"GridSearch()"* with *"StratifiedKFold()"* with $K = 5$ as our cross-validation. We took 5 parameters for each of the 3 models (shown below) and trained it for 5 times in each fold, which totals to 25 run for 5 folds. And 75 runs for all 3 models. $75 \times 4 = 300$ runs for all 4 datasets.

n_neighbors (KNN)	max_depth (Decision Tree)	n_estimators (LightGBM)
50	3	10
80	5	25
120	7	50
150	9	100
180	11	150

Table 2.1: Parameters for Different Models

5. With "KNN" we obtained the following results:

Dataset Version	Best Parameter	ROC_AUC Score
IterativeImputer(estimator = Random forest regressor)	0.696671	n_neighbors = 120
IterativeImputer(estimator = Bayesian Ridge)	0.707918	n_neighbors = 80
IterativeImputer(estimator = Extra Trees regressor)	0.697251	n_neighbors = 80
KNNImputer()	0.702050	n_neighbors = 120

Table 2.2: KNN results

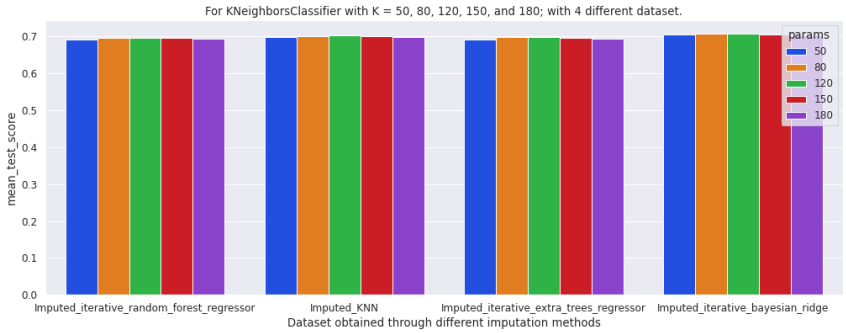

Figure 2.2: KNN Histograms

6. With “Decision Trees” we obtained the following results:

Dataset Version	Best Parameter	ROC_AUC Score
IterativeImputer(estimator = Random forest regressor)	0.786551	max_depth = 7
IterativeImputer(estimator = Bayesian Ridge)	0.787396	max_depth = 7
IterativeImputer(estimator = Extra Trees regressor)	0.789578	max_depth = 7
KNNImputer()	0.784880	max_depth = 7

Table 2.3: DT results

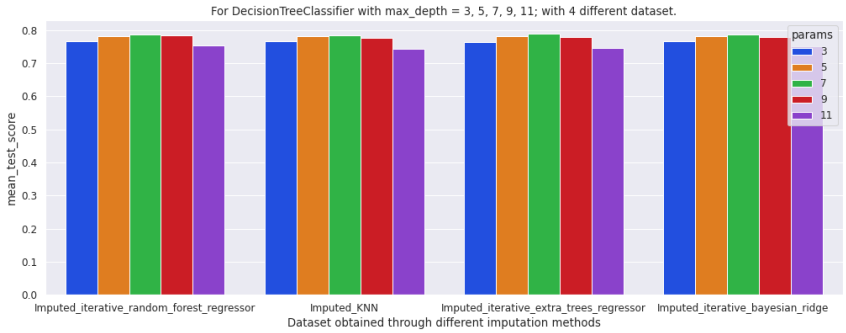

Figure 2.3: DT Histograms

7. With “LightGBM” (= Light Gradient Boosting Machine) we obtained the following:

Dataset Version	Best Parameter	ROC_AUC Score
IterativeImputer(estimator = Random forest regressor)	0.822354	n_estimators = 50
IterativeImputer(estimator = Bayesian Ridge)	0.821555	n_estimators = 50
IterativeImputer(estimator = Extra Trees regressor)	0.820878	n_estimators = 50
KNNImputer()	0.820575	n_estimators = 50

Table 2.4: LGBM results

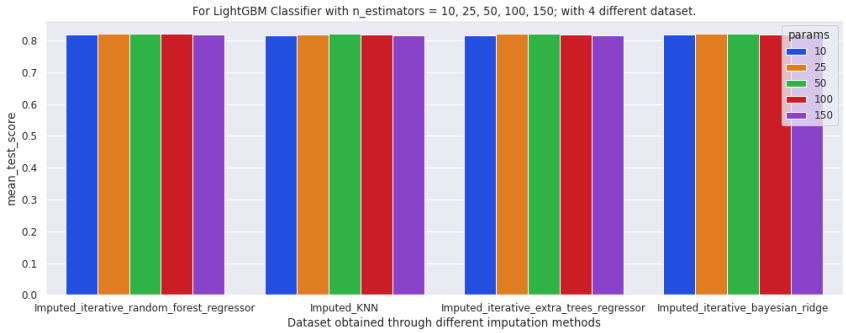

Figure 2.4: LGBM Histograms

8. Seeing above results we infer that, **MORE** or **LESS** every Imputation worked well and are at par with each other.
    - The one that gave the best performance with all the 3 models is *Random Forest Regressor* as estimator in *IterativeImputer()*.
    - *Grid Search* along with *Cross-Validation* seems to be promising for exhibiting the best parameters in a fair manner and the best model too.
    - Among all "LightGBM" proves to be the best model for this classifcation task.
    - **WHY LightGBM?**
        a) Since this is a boosting classifier it consists of iteratively (sequentially) learning weak classifiers with respect to a distribution with a motive to rectify the mistakes commited by the previous classifier i.e., basically tries to correctly classify the mis-classified sample from previous learner.
        b) And in this process more weight is being conferred to the mis-classified data while a successive learner (classifier) tries to learn.
        c) Finally, combining them generates strong classifier where all the learners decisions are respected through a voting criteria.

9. **What could be the best possible values of the parameters for respective classifier based on the ROC curves? Give Reasoning.**
    a) k-Neighbors Classifier: $k = n\_neighbors = 120$ for two of the dataset.
        - When the value of K or the number of neighbors is too low, the algorithm picks only the data-points that are closest to the given data sample, thus forming a very complex decision boundary. Such a model fails to generalize well on the test data set, thereby showing poor results.
        - The problem can be mitigated by tweaking "n_neighbors" parameter. As we increase the number of neighbors, the model starts to generalize well, but increasing the value too much would again drop the performance.
    b) Decision Tree: Depth of Tree = $max\_depth = 7$ for all four dataset.
        - If we restrict our decision trees to have a lower depth then we couldn't have enough splits to reach the pure leaf node where decisions are taken and hence pre-mature pruning causes model to underfit.

- And if we allow our tree to grow deeper, our model will become more complex since we are having enough splits which captures more information this causes overfitting as our model will fit perfectly for the training data and will not be able to generalize well on test set.

c) LightGBM: Number of Estimators = Number of Decision Trees = $n\_estimators$ = 50 for all four dataset.

- Here again we're facing the same scenario. Taking less than 50 trees doesn't allow the whole model to completely understand the data i.e., the mis-classification problem with each sequential weak learner is not completely eradicated. And taking ≥ 50 allowed it to understand the data more closely and hence overfit.

10. **If you have to choose only a subset of two features to predict earthquake, which ones would it be? Give Reasoning.**

a) We used *RandomForestClassifier()* model to rank the feature importance given the labels.

b) We got the following features as most important:

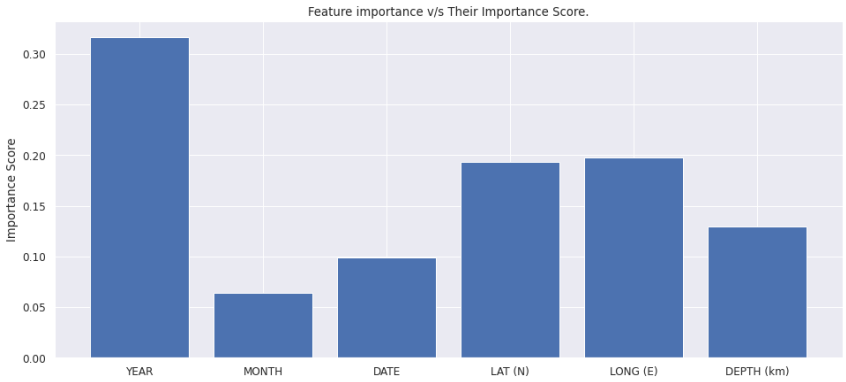

Figure 2.5: Feature Importance from Random Forest

c) Seeing above plot, we can assert that YEAR, LAT (N) and LONG (E) are the features among all that best explains the Magnitude. **WHY?** As per my intuition I came up with 3 reasons:

i. According to Longitude and Latidude, we can have information about Earthquake prone location or we can put it in this way that at given (Longitude, Latitude) the tectonic plate exhibit such motions that decide about the magnitude of earthquake and it's degree of severeness.

ii. Normally, at locations where tectonic plates collide (convergent boundary) or emerge (divergent boundary), we can witness earthquake, not all locations do that.

iii. YEAR column records the time-stamp or time-rate of change of plates boundary position which informs us regarding the tectonic activity which may occur in future (resulting in Earth-Quake). If time-rate is high then for sure in future there will be earthquakes and it's a earthquake prone zone. So given the time-rate and co-ordinates, one can predict is this a earth-quake prone zone or not and accordingly can predict when next the future earthquake will happen.

11. According to the 5-fold CV results, the **best model** is none other than *'LightGBM'* classifier with **ROC-AUC** score = **82.2**. Input Features used = YEAR, MONTH, DATE, LAT (N), LONG (E), and DEPTH (km).

12. We've tried introducing **ratio of Latitue and Longitude** as new feature column, and trained it using the *LightGBM(n_estimators = 50)* model with same 5 folds (Cross-validation) where we **lose** ROC_AUC score by 0.17%.