**FOREST COVERTYPE CLASSIFICATION**

**Aim:**

To classify the forest covertype based on the attributes obtained from Department of Forest Services in US. Forest Covertype is of 7 types and hence it is a multilabel classification.

**Dataset:**

Source of the Dataset: <https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/>

Number of instances(observations: 581012

Number of Attributes: 12 measures, but 54 columns of data(10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables)

Metadata of the Dataset:

|  |  |
| --- | --- |
| **Name** | **Measurement** |
| Elevation | meters |
| Aspect | azimuth |
| Slope | degrees |
| Horizontal\_Distance\_To\_Hydrology | meters |
| Vertical\_Distance\_To\_Hydrology | meters |
| Horizontal\_Distance\_To\_Roadways | meters |
| Hillshade\_9am | 0 to 255 index |
| Hillshade\_Noon | 0 to 255 index |
| Hillshade\_3pm | 0 to 255 index |
| Horizontal\_Distance\_To\_Fire\_Points | Meters |
| Wilderness\_Area (4 binary columns) | 0(absence) or 1 (presence) |
| Soil\_Type (40 binary columns) | 0(absence) or 1 (presence) |
| Cover\_Type (7 types) | 1 to 7 |

Forest Cover Type Classes: 1 -- Spruce/Fir

2 -- Lodgepole Pine

3 -- Ponderosa Pine

4 -- Cottonwood/Willow

5 -- Aspen

6 -- Douglas-fir

7 – Krummholz

**Sampling of Dataset**

The dataset contains 581012 obervations and 55 attributes.

Random sampling is performed on dataset and obtained 200000 samples.

dataset=df.sample(200000,random\_state=1)

CoverType\_sampling.ipynb contains the logic for sampling the observations.

**Exploratory Data Analysis:**

**Check for missing values:**

**cover** is the dataset variable name used in the entire code.

The heatmap of the dataset indicates that there are no null values in the dataset. If there are null values we have to perform imputation techniques and if more null values we need to drop that columns.

Chart

Description automatically generated

**Compare the correlation values and drop the column**

We are checking for the correlation between columns and drop one of the columns which is highly correlated with other column.

**In general, Correlation** is used to summarize the strength and direction of the linear association between two quantitative variables

**Condition used in the code:**

If the correlation value between x and y columns is greater than 0.5, drop x.

After the correlation drop technique, the number of columns reduced from 55 to 48.

Text

Description automatically generated

**Feature Selection Using Chi Square Distribution**

Chi square distribution is used to measure the degree of association between two categorical variables

Chi square uses hypothesis testing.

* H0: The attribute X has no role to play in the forest covertype ( The feature is not important)
* H1: The attribute X has a role to play in forest covertype (The feature is important)

X represents all the independent variables.

chi\_scores=chi2(X,y)

#chi\_cores yields the chi square statistic and p-values

Higher the chi square value indicates to reject the H0 and the feature is more important. The last 15 columns which are having less chi square value are dropped

**Feature Selection Using Anova Test**

ANOVA is a popular feature selection techniques that can be used for numerical input data and a categorical (class) target variable. ANOVA is an acronym for “analysis of variance” and is a parametric statistical hypothesis test for determining whether the means from two or more samples of data (often three or more) come from the same distribution or not.

ANOVA assumes the below hypothesis

H0: Means of all groups are equal.

H1: At least one mean of the groups are different.

aov\_table = sm.stats.anova\_lm(mod, typ=2)

Table

Description automatically generated

Pr > F – This is the p-value associated with the F statistic of a given source.

If PR>F is greater than 0.05, we have to remove that column. But in our case all continuous variables has less than 0.05 value, hence we are not dropping any columns.

**Data Visualization**

sns.countplot(cover['Cover\_Type'])

Chart, bar chart

Description automatically generated

There are 7 classes in target variable, Class 1 and 2 covers most of the data distribution

plt.scatter(cover['Elevation'],cover['Slope'])

Chart, scatter chart

Description automatically generated

cover.boxplot(by='Cover\_Type',column=['Elevation'])

Chart, box and whisker chart

Description automatically generated

Inference: Compared the distribution of Elevation with Cover\_Type to understand if the interval of Elevation values directly matches with the Cover\_type but there are overlapping intervals

**Splitting the input features and target variable**

x=~Cover\_Type (All columns except Cover\_Type)

y=Cover\_Type

**Train Test split**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25,random\_state=42)

train\_test\_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. Train\_test\_split() uses the following parameters.

* arrays — the dataset to be split;
* test\_size — the size of the test set.
* train\_size — the size of the train set. Its behavior is complementary to the test\_size variable.
* random\_state — before applying to split, the dataset is shuffled. The random\_state variable is an integer that initializes the seed used for shuffling. It is used to make the experiment reproducible.

**CLASSIFICATION MODEL**

**Random Forest Classifier** is used in this dataset.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction

ACCURACY OF THE MODEL: 0.93726

**Classification Report:**

precision recall f1-score support

1 0.95 0.93 0.94 18138

2 0.93 0.96 0.95 24327

3 0.91 0.94 0.92 3145

4 0.89 0.83 0.86 235

5 0.93 0.72 0.81 825

6 0.90 0.84 0.87 1575

7 0.97 0.91 0.94 1755

accuracy 0.94 50000

macro avg 0.93 0.88 0.90 50000

weighted avg 0.94 0.94 0.94 50000

We could see the class 4,5,6 having less recall and f1-score which is due to the data imbalance in the dataset. This is can be resolved by **class weight technique.**

Class Distribution:

1 54768

2 73271

3 9166

7 5260

6 4418

5 2410

4 707

Class weights modify the loss function directly by giving a penalty to the classes with different weights. It means purposely increasing the power of the minority class and reducing the power of the majority class.

clf = RandomForestClassifier(n\_estimators = 100,class\_weight='balanced')

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y))

**Classification Report:**

precision recall f1-score support

1 0.95 0.93 0.94 18138

2 0.94 0.96 0.95 24327

3 0.91 0.94 0.93 3145

4 0.89 0.84 0.86 235

5 0.93 0.71 0.80 825

6 0.89 0.84 0.87 1575

7 0.97 0.91 0.94 1755

accuracy 0.94 50000

macro avg 0.93 0.88 0.90 50000

weighted avg 0.94 0.94 0.94 50000

There is no much increase in the recall value of the classes. There is a slight increase in the recall value of Class 5

To improve further, we can focus more on

* Feature Engineering
* Turning the threshold of algorithm
* Using advanced algorithms