“Forest Cover Type Prediction” assignment

Kaagle name: Baglan Aitu

In this assignment our task is to predict the cover type of forest based on given following datasets:

- x_train, x_test – data of features used as input to a predicting model.
- y_train, y_test_baseline – data of expecting outcomes.

```
X=pd.read_csv("../input/dmmlassignment2/x_train.csv")
y=pd.read_csv("../input/dmml-assignment/y_train.csv")
actual_dataset = pd.read_csv("../input/dmml-assignment/x_test.csv")
```

Data Analyzing

1. Data size

```
print(X.shape)
```

Output: (15120, 55) which means there are 15120 instances and 55 attributes.

2. Checking for attribute datatypes

I checked data types with *print(X.dtypes),* all attributes inferred as *int64* as it supposed to be.

3. Data Statistics

```
dataset = pd.concat([X, y], axis=1)
dataset.groupby('Cover_Type').size()
```

```
Cover_Type
1    148288
2    198310
3     25028
4      1923
5      6645
6     12157
7     14357
dtype: int64
```

We can see that classes have unequal presence. Imbalanced data typically refers to classification problems. I tried re-balancing technique to handle this issue in *Data Cleaning* part.

4. Skewness of the distribution

It is a degree of distortion from a normal distribution. As a normal, values should be close to zero. As you can see in screenshot below, absolute value of all attribute values are between 0 and 10 which shows less skew.

```
print(X.skew())
```

```
id                                  8.089951e-16
Elevation                          -8.125494e-01
Aspect                              4.006402e-01
Slope                               7.888967e-01
Horizontal_Distance_To_Hydrology    1.139741e+00
Vertical_Distance_To_Hydrology      1.789010e+00
Horizontal_Distance_To_Roadways     7.137404e-01
Hillshade_9am                      -1.179937e+00
Hillshade_Noon                     -1.063233e+00
Hillshade_3pm                      -2.779948e-01
Horizontal_Distance_To_Fire_Points  1.286524e+00
Wilderness_Area_1                   2.047296e-01
Wilderness_Area_2                   4.054743e+00
Wilderness_Area_3                   2.592907e-01
Wilderness_Area_4                   3.575867e+00
Soil_Type_1                         1.376538e+01
Soil_Type_2                         8.601362e+00
Soil_Type_3                         1.084869e+01
Soil_Type_4                         6.602953e+00
Soil_Type_5                         1.908932e+01
Soil_Type_6                         9.208538e+00
Soil_Type_7                         7.675521e+01
Soil_Type_8                         5.656367e+01
Soil_Type_9                         2.234087e+01
Soil_Type_10                        3.859232e+00
Soil_Type_11                        6.650246e+00
Soil_Type_12                        4.063743e+00
Soil_Type_13                        5.504275e+00
Soil_Type_14                        3.107025e+01
Soil_Type_15                        4.509462e+02
Soil_Type_16                        1.415128e+01
Soil_Type_17                        1.289681e+01
Soil_Type_18                        1.716171e+01
Soil_Type_19                        1.180521e+01
Soil_Type_20                        7.742432e+00
Soil_Type_21                        2.597794e+01
Soil_Type_22                        3.813341e+00
Soil_Type_23                        2.673834e+00
Soil_Type_24                        4.949529e+00
Soil_Type_25                        3.501050e+01
Soil_Type_26                        1.491070e+01
Soil_Type_27                        2.293218e+01
Soil_Type_28                        2.428734e+01
Soil_Type_29                        1.514655e+00
Soil_Type_30                        4.027136e+00
Soil_Type_31                        4.429172e+00
Soil_Type_32                        2.857201e+00
Soil_Type_33                        3.161099e+00
Soil_Type_34                        1.881701e+01
Soil_Type_35                        1.749440e+01
Soil_Type_36                        6.683077e+01
Soil_Type_37                        4.267103e+01
Soil_Type_38                        5.845239e+00
Soil_Type_39                        6.257588e+00
Soil_Type_40                        7.941790e+00
dtype: float64
```

5. Correlation

Correlation shows us how attributes are related to each other. It can be used as a prediction of our target variable.

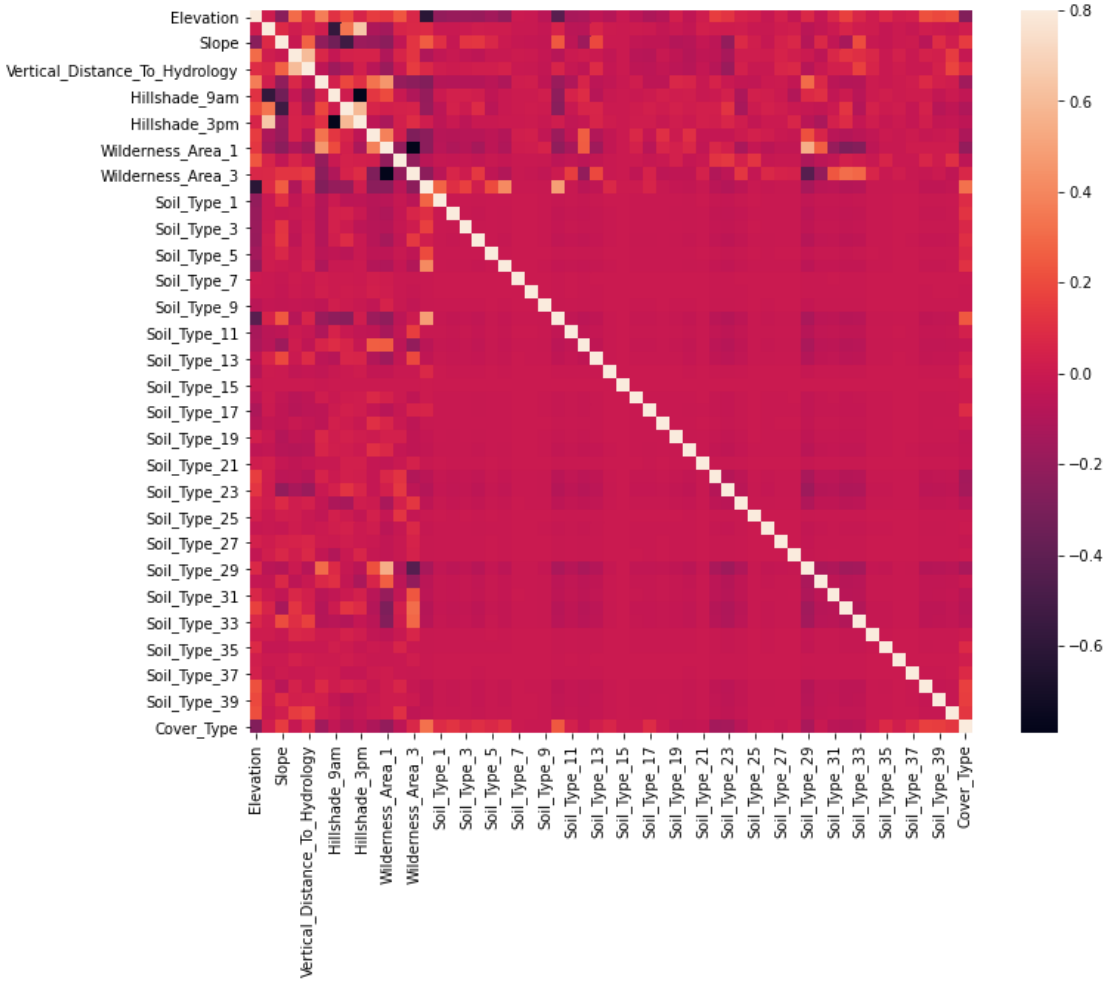
```
# Correlation

#sets the number of features considered
size = 10
#create a dataframe with only 'size' features
data=X.iloc[:,:size]
#get the names of all the columns
cols=data.columns
# Calculates pearson co-efficient for all combinations
data_corr = data.corr()
# Set the threshold to select only only highly correlated attributes
threshold = 0.5
# List of pairs along with correlation above threshold
corr_list = []
#Search for the highly correlated pairs
for i in range(0,size): #for 'size' features
    for j in range(i+1,size): #avoid repetition
        if (data_corr.iloc[i,j] >= threshold and data_corr.iloc[i,j] < 1) or (data_corr.iloc[i,j] < 0 and data
            corr_list.append([data_corr.iloc[i,j],i,j]) #store correlation and columns index
#Sort to show higher ones first
s_corr_list = sorted(corr_list,key=lambda x: -abs(x[0]))
#Print correlations and column names
for v,i,j in s_corr_list:
    print ("%s and %s = %.2f" % (cols[i],cols[j],v))
```

```
Hillshade_9am and Hillshade_3pm = -0.78
Aspect and Hillshade_3pm = 0.65
Horizontal_Distance_To_Hydrology and Vertical_Distance_To_Hydrology = 0.61
Hillshade_Noon and Hillshade_3pm = 0.59
Aspect and Hillshade_9am = -0.58
Slope and Hillshade_Noon = -0.53
```

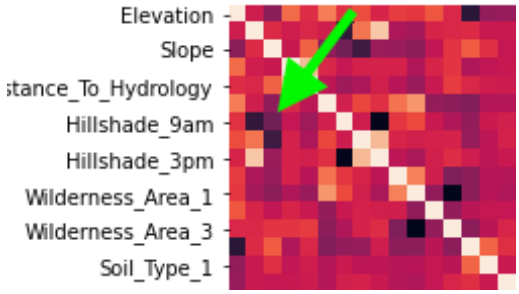
Strong correlation was observed between above shown attributes. I used these correlated attributes for data normalization in *Data Cleaning* part.

6. Correlation Matrix

```
#correlation matrix
corrmat = dataset.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



The heatmap is also one of the best ways to get a quick overview of relationships between attributes. As previous correlation technique showed, I did not see the high correlation between *Hillshade_9am* and *Hillshade_3pm*.



This is why normalization between these attributes did not give me good updates at the output.

Data Cleaning

1. Dropping first columns

After examining the attributes of dataset, I got rid the first columns of order numbers which have no benefit for prediction.

```
X = X[X.columns[1:]]
y = y[y.columns[1:]]
```

2. Removing rows with missing value & filling it with mean value

Next thing I did was looking for missing values in given dataset. First of all, I concatenated *x_train* and *y_train* datas.

```
result = pd.concat([X, y], axis=1)
print(result.describe())
```

	Elevation	Aspect	Slope
count	388526.000000	388526.000000	388526.000000
mean	2964.034016	156.545956	14.226134
std	277.758536	111.908913	7.426231
min	1859.000000	0.000000	0.000000
25%	2816.000000	59.000000	9.000000
50%	2999.000000	129.000000	13.000000
75%	3166.000000	262.000000	19.000000
max	3858.000000	360.000000	61.000000

	Hillshade_3pm	Horizontal_Distance_To_Fire_Points	...	Soil_Type_32	\
count	388526.000000	388526.000000	...	388526.000000	
mean	142.864846	1984.222739	...	0.093819	
std	38.084925	1321.950873	...	0.291577	
min	1.000000	0.000000	...	0.000000	
25%	119.000000	1027.000000	...	0.000000	
50%	143.000000	1714.000000	...	0.000000	
75%	169.000000	2553.000000	...	0.000000	
max	254.000000	7168.000000	...	1.000000	

I decided to remove rows with zeros values in *Aspect* and *Horizontal_Distance_To_Fire* attributes, and filled it with its mean value since I don't think these attributes might have 0 values in reality.

There was *Slope* attribute with 0 min value. As it shows angle, I thought it was reasonable value. The rest *Soil_Types* were not modified.

```
# Removing rows with missing value

from numpy import nan
print(result.shape) # number of matrix before removing rows with zero values

result[["Horizontal_Distance_To_Fire_Points"]] = result[["Horizontal_Distance_To_Fire_Points"]].replace(0, nan)
result.dropna(inplace=True)
#result.fillna(result[["Horizontal_Distance_To_Fire_Points"]].mean(), inplace=True)

result[["Aspect"]] = result[["Aspect"]].replace(0, nan)
result.dropna(inplace=True)
#result.fillna(result[["Aspect"]].mean(), inplace=True)

result.fillna(result.mean(), inplace=True)


y = result.iloc[:, -1:]
X = result.iloc[:,:-1]
print(y.shape)
print(X.shape)
```

```
(406708, 55)
(403259, 1)
(403259, 54)
```

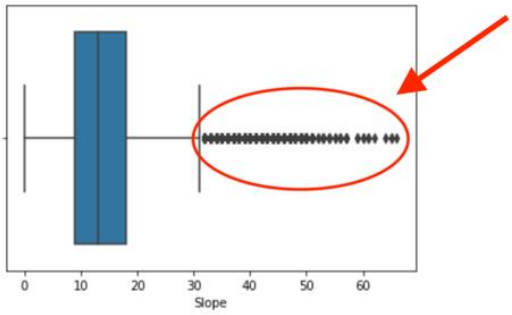
3. Outliers

Outliers are observations unlike the rest observations. It shifts our output result. Actually, there is no precise way to define and identify outliers in general because of the specifics of each dataset. Despite this I used visualization tools to discover and detect outliers in the dataset.

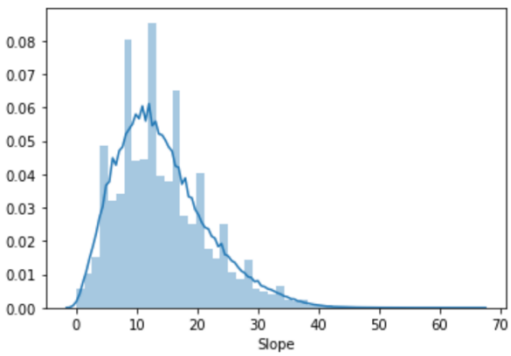
Almost all attributes of *x_train* had different numbers, but I did not take them as outliers. One example:

```
import seaborn as sns
sns.boxplot(x=X['Slope'])
```

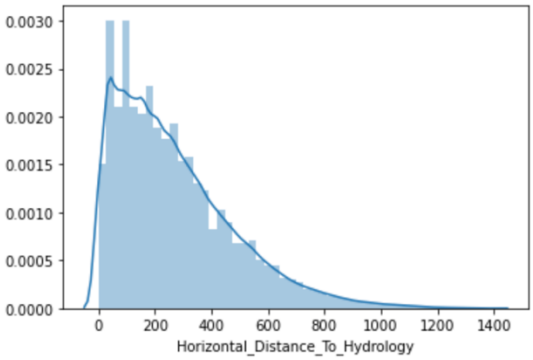
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa67a2e39d0>
```




```
import seaborn as sns
sns.distplot(dataset['Slope']);
```



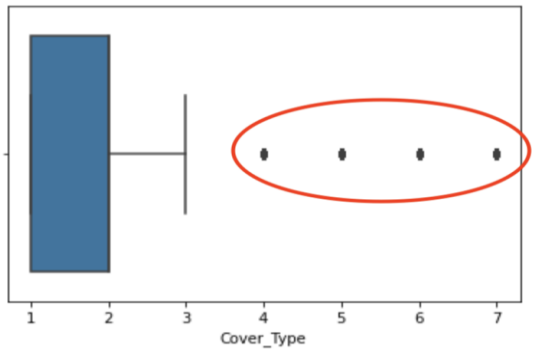
```
import seaborn as sns
sns.distplot(dataset['Horizontal_Distance_To_Hydrology']);
```



I visualized the plot of *y_train* dataset and detected 4 outliers as it shown down below.

```
import seaborn as sns
sns.boxplot(x=y['Cover_Type'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa67a08f410>
```



After analyzing directly the dataset I detected the value 7 which looks different from the rest population.

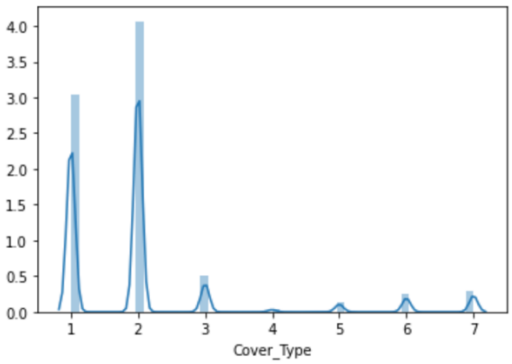
```
y.head(30)
```

Out[35]:

	Cover_Type
0	1
1	2
2	2
3	1
4	1
5	1
6	2
7	2
8	1
9	1
10	2
11	2
12	2
13	7
14	2

Then I found other different values:

```
import seaborn as sns
sns.distplot(dataset['Cover_Type']);
```

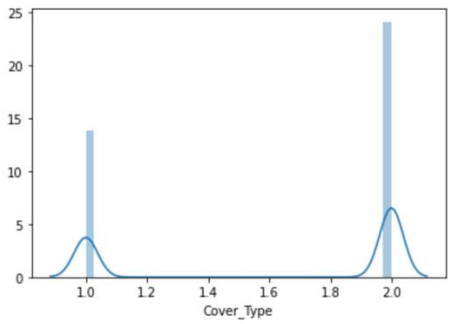


I replaced them with its mean value.

```
# Removing outliers from y_train
median = y.loc[y['Cover_Type']<3, 'Cover_Type'].median()
y['Cover_Type'] = np.where(y['Cover_Type'] > median, median, y['Cover_Type'])
```

	Cover_Type
0	1.0
1	2.0
2	2.0
3	1.0
4	1.0
5	1.0
6	2.0
7	2.0
8	1.0
9	1.0
10	2.0
11	2.0
12	2.0
13	2.0
14	2.0
15	2.0

```
import seaborn as sns
sns.distplot(y['Cover_Type']);
```



I repeated the same algorithm for *x_train* attributes *Slope* until *Wilderness_Area_4*.

(I did not notice big changes in the accuracy result after handling outliers of x_train starting from Slope until *Wilderness_Area_4)*

4. Re-balancing

To handle imbalanced data I tried over-sampling. By command *dataset.groupby('Cover_Type').size()* I found the length of biggest class and filled other classes to its size.

```
max_size = 198310

lst = [dataset]
for class_index, group in dataset.groupby('Cover_Type'):
    lst.append(group.sample(max_size-len(group), replace=True))
dataset = pd.concat(lst)

y = dataset.iloc[:, -1:]
X = dataset.iloc[:,:-1]
```

(It gave me result 99.2 % on Kaggle notebook and 56% accuracy on ITK challenge, since it was the case of overfitting).

5. Normalization

As correlation table showed above, I decided to normalize values between highly correlated numbers instead of normalization of whole dataset.

```
# Normalization between highly correlated attributes
result = pd.concat([X["Hillshade_9am"],X["Hillshade_3pm"]], axis=1)
result=((result-result.min())/(result.max()-result.min()))*2
X["Hillshade_9am"] = result.iloc[:, :-1]
X["Hillshade_3pm"] = result.iloc[:, -1]

result = pd.concat([X['Horizontal_Distance_To_Hydrology'],X['Vertical_Distance_To_Hydrology']], axis=1)
result=((result-result.min())/(result.max()-result.min()))*2
X['Horizontal_Distance_To_Hydrology'] = result.iloc[:, :-1]
X['Vertical_Distance_To_Hydrology'] = result.iloc[:, -1]

result = pd.concat([X['Slope'],X['Hillshade_Noon']], axis=1)
result=((result-result.min())/(result.max()-result.min()))*2
X['Slope'] = result.iloc[:, :-1]
X['Hillshade_Noon'] = result.iloc[:, -1]
```

(I did not notice big changes in the accuracy result).

6. Splitting arrays

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 10, stratify=y
# test_size = 0.30, random state = 10
x_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
x_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

7. Other

Here I added two data cleaning operations which I did not include into my algorithm. The reason for that they were irrelevant for this task.

a) Dropping *Soil_Type* attribues

```
# Dropping columns from 13 to 40
numbers = range(13,40)
for c in numbers:
    X[X.columns[c:]].drop(rem,axis=1,inplace=True)
```

At first glance, a lot of zeros seemed to me as missing values. I tried to drop them all, but it did not give me any development at the output. I decided to leave them as it were.

b) Converting to positive

```
# Converting to positive values
for c in X.columns:
    X[c] = abs(X[c])
```

Initially, I thought that attributes like *Vertical_Distance_To_Hydrology* cannot take negative values. After finding out that it can, I decided to not include it to my model.

Prediction model

Model	Accuracy
Random Forest (from lab 6)	92.796 %
Logistic Regression	61.82%
KNN classifier	96.11%
Bagging Classifier & Decision Tree Classifier	97.102%
Balanced Bagging Classifier & Decision Tree Classifier	97.03%
Extra Tree Classifier	94.59
AdaBoost Classifier	94.59%
Gradient Boosting Classifier	94.59%

```
# By far best result
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

#Base estimator
base_estimator = DecisionTreeClassifier(random_state=10,max_depth=13)

n_list = [100]

for n_estimators in n_list:
    #Set the base model
    model = BaggingClassifier(n_jobs=-1,base_estimator=base_estimator, n_estimators=n_estimators, random_state

forest = BaggingClassifier(n_estimators=20) # initially 10
forest.fit(x_train, y_train.values.ravel())
```

I chose Ensemble method for prediction, exactly combined decision trees by using Bagging (Bootstrap Aggregation). The main principle behind the ensemble model is that a group of learners come together to form a strong learner.

In summary, this assignment was full of experiments. I utilized many pre-processing techniques in practice and my conclusions look as following:

- To get good prediction we need to find suitable classifier for given task.
- I noticed good developments from removing missing values from the dataset (x_train and y_train) and outliers from y_train.
- I understood that re-balancing technique like over-sampling is not good for making prediction model as it led to overfitting. Instead of increasing accuracy it drastically decreased it. I did not tried under-sampling, but I guess it would give the similar results.
- I did not notice good developments from removing outliers from x_train, because they weren't outliers as showed me *sns.distplot*.
- Correlation Matrix is best indicator of class correlation.
- Normalization technique gave little changes in accuracy.
- There are some data cleaning methods which cannot be applied for model because of the specifics of given dataset. In my case, they are dropping *Soil_Type* attribues and taking absolute values of dataset.