Final Project Complete Dec 8

December 8, 2019

1 W207 Project - Forest Cover, Kaggle

Abstract: Our team, comprised of **Sudipto Dasgupta, Jeff Li, and Karthik Srinivasan** has been tasked the Forest Cover Type Prediction problem on Kaggle (https://www.kaggle.com/c/forest-cover-type-prediction). In this competition we are asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type.

We apply a combination of machine learning models and data preprocessing techniques to the data in order to try and predict the cover type.

Forests and the biodiversity they contain are declining at an alarming rate throughout the world. Prediction of cover type can help in assessing the rate of deforestation and taking a corrective action to nurture the flora and fauna.

1.1 Part 1: Data Processing, Exploratory Data Analysis

Dataset description

Labels The study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. Each observation is a 30m x 30m patch. We are asked to predict an integer classification for the forest cover type. The seven types are:
1 - Spruce/Fir
2 - Lodgepole Pine
3 - Ponderosa Pine
4 - Cottonwood/Willow
5 - Aspen
6 - Douglas-fir
7 - Krummholz

The training set (15120 observations) contains both features and the Cover_Type. The test set contains only the features. We will predict the Cover_Type for every row in the test set (565892 observations).

Data Fields Elevation - Elevation in meters
Aspect - Aspect in degrees azimuth

Slope - Slope in degrees
Horizontal_Distance_To_Hydrology - Horizontal Distance to nearest surface water features
Vertical_Distance_To_Hydrology - Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways - Horizontal Distance to nearest roadway
Hillshade_9am (0 to 255 index) - Hillshade index at 9am, summer solstice
Hillshade_Noon (0 to 255 index) - Hillshade index at noon, summer solstice
Hillshade_3pm (0 to 255 index) - Hillshade index at 3pm, summer solstice
Horizontal_Distance_To_Fire_Points - Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns, 0 = absence or 1 = presence) - Wilderness area designation
Soil_Type (40 binary columns, 0 = absence or 1 = presence) - Soil Type designation
Cover_Type (7 types, integers 1 to 7) - Forest Cover Type designation

The wilderness areas are: 1 - Rawah Wilderness Area
2 - Neota Wilderness Area
3 - Comanche Peak Wilderness Area
4 - Cache la Poudre Wilderness Area

The soil types are: 1 - Cathedral family - Rock outcrop complex, extremely stony.
2 - Vanet - Ratake families complex, very stony.
3 - Haploborolis - Rock outcrop complex, rubbly.
4 - Ratake family - Rock outcrop complex, rubbly.
5 - Vanet family - Rock outcrop complex complex, rubbly.
6 - Vanet - Wetmore families - Rock outcrop complex, stony.
7 - Gothic family.
8 - Supervisor - Limber families complex.
9 - Troutville family, very stony.
10 - Bullwark - Catamount families - Rock outcrop complex, rubbly.
11 - Bullwark - Catamount families - Rock land complex, rubbly.
12 - Legault family - Rock land complex, stony.
13 - Catamount family - Rock land - Bullwark family complex, rubbly.
14 - Pachic Argiborolis - Aquolis complex.
15 - unspecified in the USFS Soil and ELU Survey.
16 - Cryaquolis - Cryoborolis complex.
17 - Gateview family - Cryaquolis complex.
18 - Rogert family, very stony.
19 - Typic Cryaquolis - Borohemists complex.
20 - Typic Cryaquepts - Typic Cryaquolls complex.
21 - Typic Cryaquolls - Leighcan family, till substratum complex.
22 - Leighcan family, till substratum, extremely bouldery.
23 - Leighcan family, till substratum - Typic Cryaquolls complex.
24 - Leighcan family, extremely stony.
25 - Leighcan family, warm, extremely stony.
26 - Granile - Catamount families complex, very stony.
27 - Leighcan family, warm - Rock outcrop complex, extremely stony.
28 - Leighcan family - Rock outcrop complex, extremely stony.
29 - Como - Legault families complex, extremely stony.
30 - Como family - Rock land - Legault family complex, extremely stony.

31 - Leighcan - Catamount families complex, extremely stony.
32 - Catamount family - Rock outcrop - Leighcan family complex, extremely stony.
33 - Leighcan - Catamount families - Rock outcrop complex, extremely stony.
34 - Cryorthents - Rock land complex, extremely stony.
35 - Cryumbrepts - Rock outcrop - Cryaquepts complex.
36 - Bross family - Rock land - Cryumbrepts complex, extremely stony.
37 - Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony.
38 - Leighcan - Moran families - Cryaquolls complex, extremely stony.
39 - Moran family - Cryorthents - Leighcan family complex, extremely stony.
40 - Moran family - Cryorthents - Rock land complex, extremely stony.

Import Dependencies

[1]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
import warnings
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
from sklearn.ensemble import  GradientBoostingClassifier
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC, LinearSVC
from sklearn.neural_network import MLPClassifier

#Additional imports
import graphviz
from sklearn import tree
import xgboost as xgb

from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
import json
import pickle
from IPython.display import Image
```

[2]:
```
%%javascript
    IPython.OutputArea.auto_scroll_threshold = 9999
```

```
<IPython.core.display.Javascript object>
```

[3]:
```
pd.set_option('display.max_columns',100)
warnings.filterwarnings(action='ignore')
```

Read all Data Two data files are provided from this Kaggle challenge. - train.csv: This is train data, and already has Cover_Type column populated. - test.csv: For each row in this file, Cover_Type needs to be predicted as a column. The resulting file will be submitted to Kaggle.

[4]:
```
data = pd.read_csv("train.csv")
data_predict = pd.read_csv("test.csv" , header = None)
print ( data_predict.shape , data.shape)
```

```
(565893, 55) (15120, 56)
```

There are 565893 records in test file and 15120 records in train file.

Data Preparation and Split We split the train data into train, dev, & test. The split is done using random sampling. We split the data accordingly: - train_data : 12000 records
- dev_data : 1000 records
- test_data : 2120 records

All EDA will be done on the train data.

[5]:
```
X = np.array(data.as_matrix(columns=data.columns[1:55]))
Y = np.array(data["Cover_Type"].tolist())
shuffle = np.random.permutation(np.arange(X.shape[0]))
X, Y = X[shuffle], Y[shuffle]
train_df = data.iloc[shuffle,:].iloc[:12000 , :]
print('data shape: ', X.shape)
print('label shape:', Y.shape)
test_data, test_labels = X[13000:], Y[13000:]
dev_data, dev_labels = X[12000:13000], Y[12000:13000]
train_data, train_labels = X[:12000], Y[:12000]
```

```
data shape:  (15120, 54)
label shape: (15120,)
```

1.1.1 Exploratory Data Analysis

Step 1 : Describe the train data

[11]:
```
display(train_df.describe())
```

```
                Id     Elevation        Aspect         Slope  \
count  12000.000000  12000.000000  12000.000000  12000.000000
```

mean	7553.249833	2748.473750	157.101667	16.491667
std	4379.699551	417.973474	110.189944	8.465444
min	1.000000	1863.000000	0.000000	0.000000
25%	3765.750000	2375.000000	65.000000	10.000000
50%	7513.000000	2751.000000	126.000000	15.000000
75%	11361.250000	3102.000000	261.000000	22.000000
max	15119.000000	3849.000000	360.000000	52.000000

	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	\
count	12000.000000	12000.000000	
mean	227.352750	51.196000	
std	211.041781	61.607116	
min	0.000000	-134.000000	
25%	67.000000	5.000000	
50%	180.000000	32.000000	
75%	324.000000	80.000000	
max	1343.000000	554.000000	

	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	\
count	12000.000000	12000.000000	12000.000000	
mean	1715.329417	212.557333	219.057000	
std	1323.880746	30.630362	22.837317	
min	0.000000	0.000000	99.000000	
25%	765.000000	196.000000	207.000000	
50%	1317.000000	220.000000	223.000000	
75%	2271.000000	235.000000	235.000000	
max	6811.000000	254.000000	254.000000	

	Hillshade_3pm	Horizontal_Distance_To_Fire_Points	Wilderness_Area1	\
count	12000.000000	12000.000000	12000.000000	
mean	135.364000	1509.044417	0.236583	
std	45.755742	1103.985712	0.425002	
min	0.000000	0.000000	0.000000	
25%	107.000000	725.000000	0.000000	
50%	138.000000	1250.000000	0.000000	
75%	167.000000	1984.000000	0.000000	
max	248.000000	6993.000000	1.000000	

	Wilderness_Area2	Wilderness_Area3	Wilderness_Area4	Soil_Type1	\
count	12000.000000	12000.000000	12000.000000	12000.000000	
mean	0.032333	0.420917	0.310167	0.024333	
std	0.176891	0.493727	0.462581	0.154088	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	1.000000	1.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

	Soil_Type2	Soil_Type3	Soil_Type4	Soil_Type5	Soil_Type6
count	12000.000000	12000.000000	12000.000000	12000.000000	12000.000000
mean	0.039750	0.062667	0.057250	0.011667	0.043333
std	0.195379	0.242373	0.232329	0.107385	0.203615
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	Soil_Type7	Soil_Type8	Soil_Type9	Soil_Type10	Soil_Type11
count	12000.0	12000.000000	12000.000000	12000.000000	12000.000000
mean	0.0	0.000083	0.000750	0.143333	0.027667
std	0.0	0.009129	0.027377	0.350427	0.164023
min	0.0	0.000000	0.000000	0.000000	0.000000
25%	0.0	0.000000	0.000000	0.000000	0.000000
50%	0.0	0.000000	0.000000	0.000000	0.000000
75%	0.0	0.000000	0.000000	0.000000	0.000000
max	0.0	1.000000	1.000000	1.000000	1.000000

	Soil_Type12	Soil_Type13	Soil_Type14	Soil_Type15	Soil_Type16
count	12000.000000	12000.000000	12000.000000	12000.0	12000.000000
mean	0.014583	0.029583	0.010917	0.0	0.007750
std	0.119883	0.169442	0.103915	0.0	0.087696
min	0.000000	0.000000	0.000000	0.0	0.000000
25%	0.000000	0.000000	0.000000	0.0	0.000000
50%	0.000000	0.000000	0.000000	0.0	0.000000
75%	0.000000	0.000000	0.000000	0.0	0.000000
max	1.000000	1.000000	1.000000	0.0	1.000000

	Soil_Type17	Soil_Type18	Soil_Type19	Soil_Type20	Soil_Type21
count	12000.000000	12000.000000	12000.000000	12000.000000	12000.000000
mean	0.039750	0.003750	0.003083	0.009667	0.000833
std	0.195379	0.061125	0.055444	0.097847	0.028857
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

	Soil_Type22	Soil_Type23	Soil_Type24	Soil_Type25	Soil_Type26
count	12000.000000	12000.000000	12000.000000	12000.000000	12000.000000
mean	0.022250	0.050583	0.016583	0.000083	0.003583
std	0.147502	0.219154	0.127709	0.009129	0.059756
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000

```
max        1.000000      1.000000      1.000000      1.000000      1.000000

       Soil_Type27   Soil_Type28   Soil_Type29   Soil_Type30   Soil_Type31  \
count  12000.000000  12000.000000  12000.000000  12000.000000  12000.000000
mean       0.000917      0.000583      0.085833      0.047250      0.021667
std        0.030264      0.024146      0.280129      0.212182      0.145599
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

       Soil_Type32   Soil_Type33   Soil_Type34   Soil_Type35   Soil_Type36  \
count  12000.000000  12000.000000  12000.000000  12000.000000  12000.000000
mean       0.045417      0.040667      0.001667      0.006500      0.000667
std        0.208225      0.197525      0.040792      0.080363      0.025812
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

       Soil_Type37   Soil_Type38   Soil_Type39   Soil_Type40    Cover_Type
count  12000.000000  12000.000000  12000.000000  12000.000000  12000.000000
mean       0.002000      0.049000      0.043833      0.030167      4.002500
std        0.044678      0.215877      0.204733      0.171053      2.001498
min        0.000000      0.000000      0.000000      0.000000      1.000000
25%        0.000000      0.000000      0.000000      0.000000      2.000000
50%        0.000000      0.000000      0.000000      0.000000      4.000000
75%        0.000000      0.000000      0.000000      0.000000      6.000000
max        1.000000      1.000000      1.000000      1.000000      7.000000
```

Observations:

1. All fields are continous or binary. There are no text fields.
2. Soil_Type fields are binary.
3. Wilderness Area fields are binary.
4. The following fields appear to be continuous:
    - 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

Step 2: Count zero and NA values

[12]:
```
def count_zero_for_a_column(train_df, column_name):
    count_obj = train_df.apply(lambda x: True if x[column_name] == 0 else False␣
 ↪, axis=1)
    no_rows = len(count_obj[count_obj == True].index)
    print ( column_name , ' : ' , no_rows )
print ("\033[1m" , 'Count of zero rows by column :' , "\033[0;0m")
column_list = ['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']
for column_name in column_list:
    count_zero_for_a_column(train_df, column_name)
```

```
 Count of zero rows by column :
Elevation  :  0
Aspect  :  86
Slope  :  3
Horizontal_Distance_To_Hydrology  :  1264
Vertical_Distance_To_Hydrology  :  1506
Horizontal_Distance_To_Roadways  :  3
Hillshade_9am  :  1
Hillshade_Noon  :  0
Hillshade_3pm  :  70
Horizontal_Distance_To_Fire_Points  :  1
```

Observations: The columns "Horizontal Distance to Hydrology" and "Vertical Distance to Hydrology" have a significant number of records ($> 10\%$) which are zero. This could be possible for vegetation which are very close to ground water.

[13]:
```
print ("\033[1m" , 'Count of NA rows by column :' , "\033[0;0m")
for i , row in enumerate(train_df.count(axis = 0)):
    if train_df.shape[0] != row:
        print (train_df.columns[i])
```

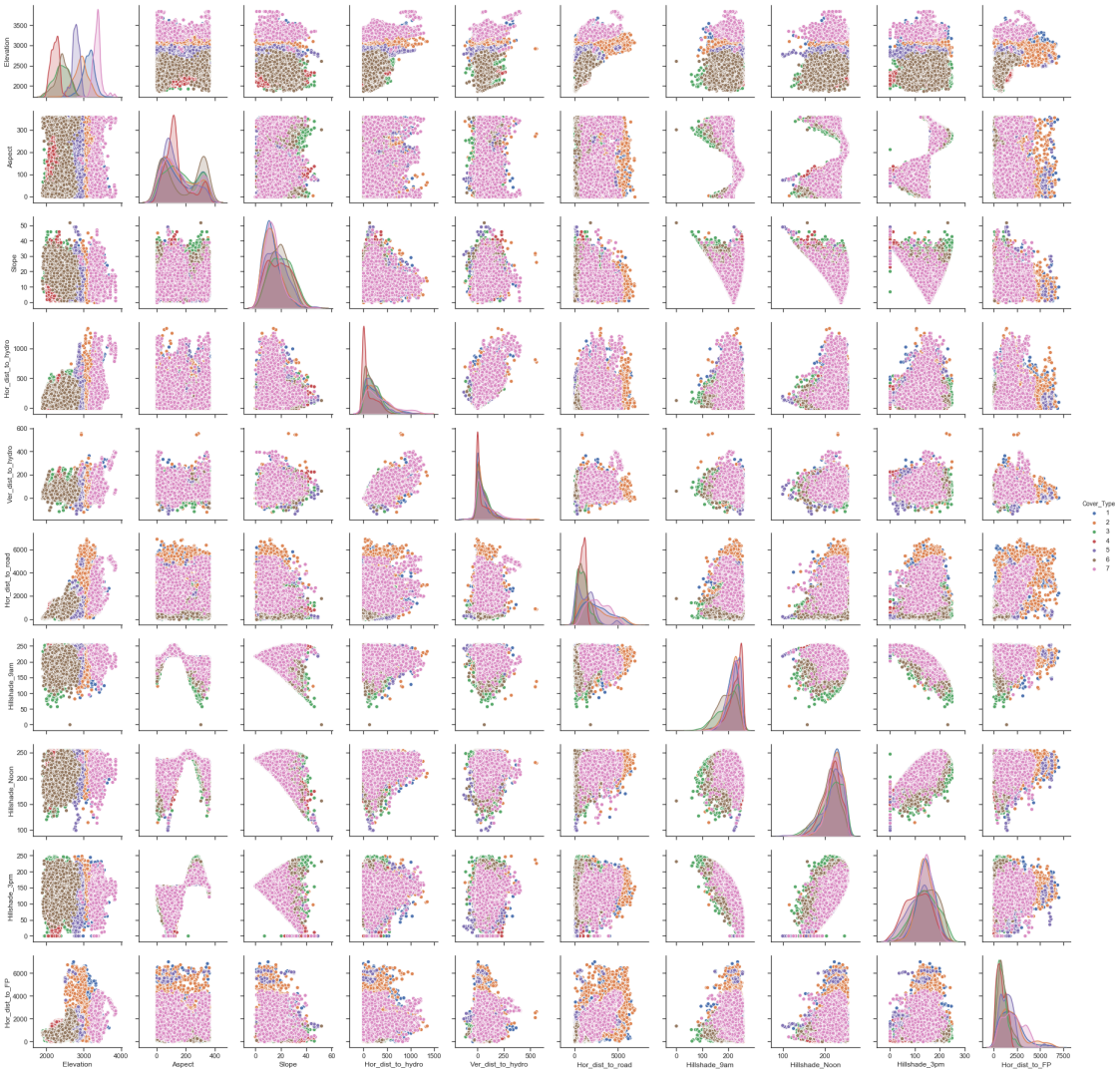
```
 Count of NA rows by column :
```

Obeservation : There are no NA values in the data.

Step 3 : Scatter plot for continous fields

[12]:
```
sns.set(style="ticks")
plot_df = train_df[['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' , 'Cover_Type' ]]
```

```
plot_df = plot_df.rename(columns={'Horizontal_Distance_To_Hydrology':␣
↪'Hor_dist_to_hydro',
                                  'Vertical_Distance_To_Hydrology':␣
↪'Ver_dist_to_hydro',
                                  'Horizontal_Distance_To_Roadways':␣
↪'Hor_dist_to_road',
                                  'Horizontal_Distance_To_Fire_Points':␣
↪'Hor_dist_to_FP'
                                  })

chart = sns.pairplot( plot_df , vars = ['Elevation', 'Aspect',␣
↪'Slope','Hor_dist_to_hydro','Ver_dist_to_hydro', 'Hor_dist_to_road',
      'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm','Hor_dist_to_FP'] ,␣
↪hue = 'Cover_Type')
```

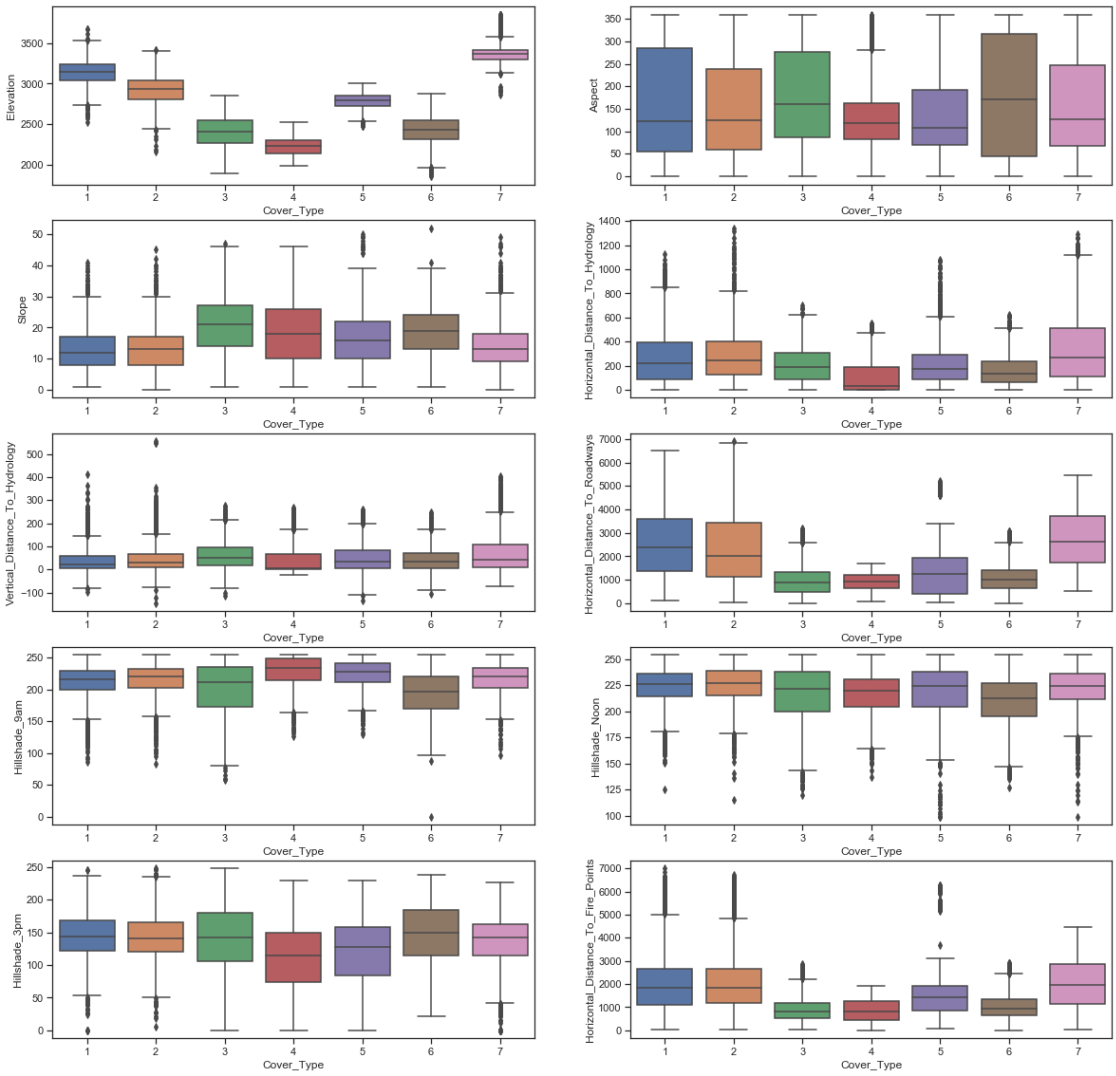


Observations:

1. The graph shows that there is a correlation between Hillshade_3PM, Hillshade_9AM, Hillshade_Noon, although it is not linear.
2. Hillshade_3PM , Hillshade_9AM , Hillshade_Noon seem to be correlated to Aspect, although the relationship does not appear to be linear.
3. Hillshade_3PM , Hillshade_9AM , Hillshade_Noon seem to be correlated to Slope, although the relationship does not appear to be linear.
4. The distribution of elevation is different for different values of Cover_Type. As a result, we expect Elevation to be a significant parameter in our models.

Step 4 : Box Plots by Cover Type for the Continous Variables

[15]:
```
fig, axes = plt.subplots(5, 2)
fig.set_size_inches(20, 20)
column_list = ['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' ]
i = 0
for column_name in column_list:
    row_index = i // 2
    column_index = i % 2
    sns.boxplot(x="Cover_Type", y=column_name, data=data ,␣
↪ax=axes[row_index][column_index])
    i = i + 1
```

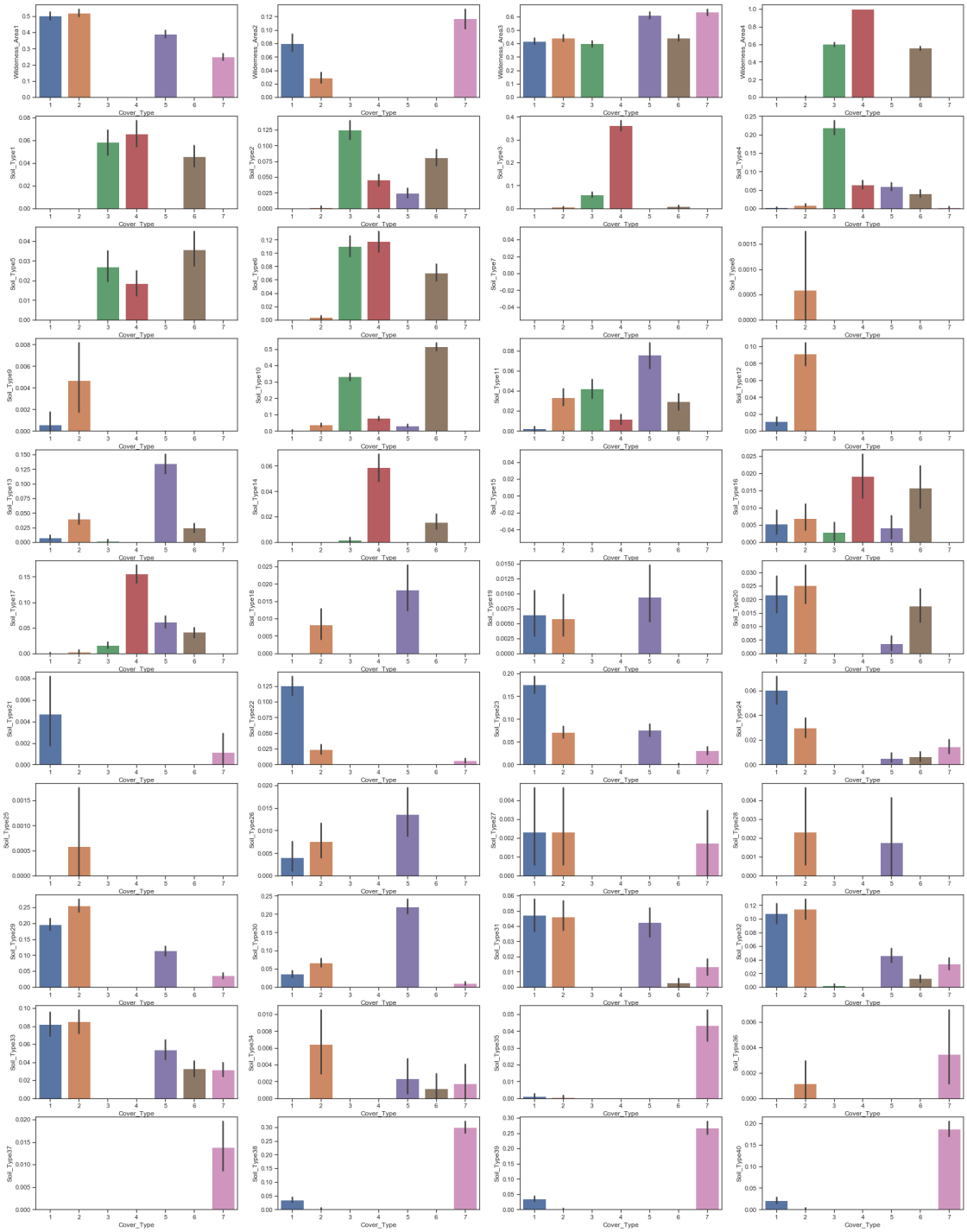


Observations : 1. Mean Elevation is highest for Cover_Type 7. 2. Mean Slope is lower for Cover_Type 1, 2 and 7. 3. Mean Horizontal Distance to Roadways and Horizontal Distance to Fire Points are higher for Cover_Type 1, 2, & 7. 4. Elevation varies based on Cover_Type. Therefore, we expect the elevation to be significant parameter for our models.

Step 5 : Bar Graphs by Cover Type for Soil Type and Wilderness Area Variables

[16]:
```
column_list = ['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' ]
fig, axes = plt.subplots(11, 4)
fig.set_size_inches(30, 40)
i = 0
```

```
for column in train_df.columns:
    if column not in column_list and column != 'Cover_Type' and column != 'Id':
        row_index = i // 4
        column_index = i % 4
        sns.barplot(x="Cover_Type", y=column, data=train_df ,␣
 ↪ax=axes[row_index][column_index])
        i = i + 1
```



Observations: 1. Soil_Type8 and Soil_Type25 only exists for Cover_Type 2. 2. Soil_Type37 only exists for Cover_Type 7. 3. Soil_Type40, Soil_Type39, Soil_Type38, Soil_Type36, Soil_Type_35 mainly exists for Cover_Type 7. 4. Wilderness_Area2 only exists for Cover_Type 1, 2, 7. 5. Wilderness_Area2 mainly exists for Cover_Type 3, 4, 6. 6. Soil_Type7 and Soil_Type15 is 0 for all cover types. We do not expect these parameters as a significant parameter for models.

Step 6 : Feature Engineering - Remove Constant Columns We remove the columns Soil_Type7 and Soil_Type15 since they are constant across all data points.

[17]:
```
print (train_data.shape , dev_data.shape , test_data.shape)
```

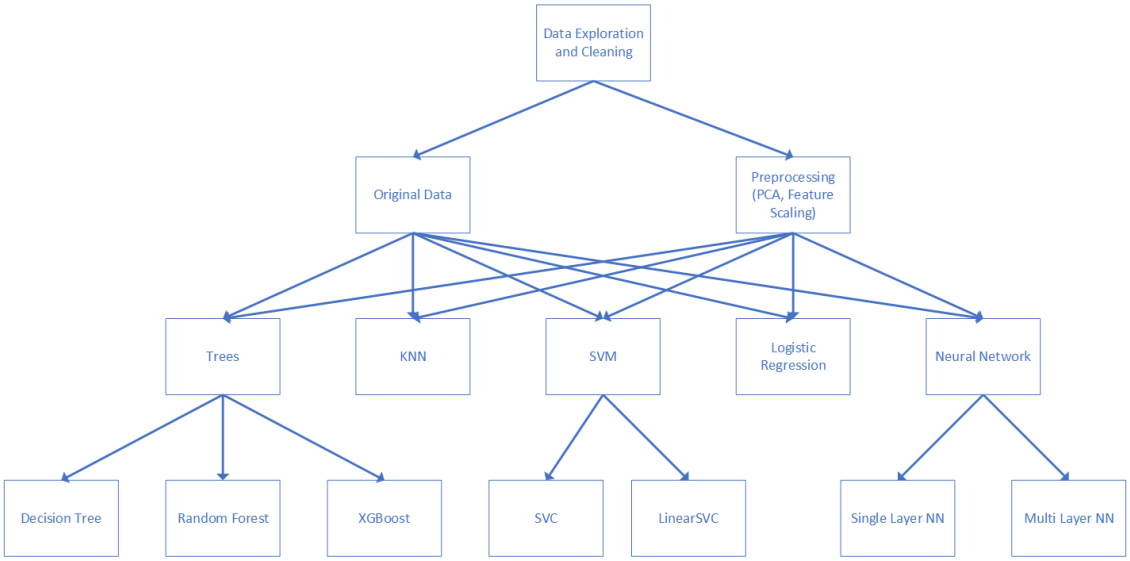
```
(12000, 54) (1000, 54) (2120, 54)
```

[18]:
```
column_to_remove = []
for i, column_name in enumerate(train_df.columns):
    if column_name in  ['Soil_Type7' ,'Soil_Type15'] :
        column_to_remove.append(i)
print (column_to_remove)
train_data = np.delete(train_data, column_to_remove, axis=1)
dev_data = np.delete(dev_data, column_to_remove, axis=1)
test_data = np.delete(test_data, column_to_remove, axis=1)
print (train_data.shape , dev_data.shape , test_data.shape)
```

```
[21, 29]
(12000, 52) (1000, 52) (2120, 52)
```

1.2 Part 2: Preprocessing & Modelling approaches

1.2.1 Modelling Approaches Overview



In order to discover what models best predict cover type, we will examine several types of Machine Learning models, which are listed below.

Models: - Logistic Regression - Support Vector Machine - SVC - Linear SVC - Neural Network - One Layer NN
- Multi Layer NN - K Nearest Neighbors - Trees - Random Forest - Decision Tree - XGBoost

Based on our EDA, the scales on several data points like Elevation are drastically different from attributes like Soil Type. We will apply a series of different scalers to try to account for this.

- Scaling Techniques
  - Robust Scaler - removes the median and scales the data according to the quantile range
  - MinMax Scaler - scales data from 0 to 1
  - Standard Scaler - standardizes features by removing the mean and scaling to unit variance

Furthermore, we also try applying dimensionality reduction to see this will yield any improvement in predictive performance. We set a threshold on the explained variance ratio of 95%, and use this threshold to select the number of principal components.

We loop through all combinations of models, scaling techniques, and dimensionality reduction to try and discover the optimal prediction of forest type. Furthermore, we also loop through a set of model specific hyperparameters to improve accuracy.

1.2.2 Machine Learning Class

[37]:
```
class MLModel():
    """
    Parent class for all ML models
    """
    def __init__(self, modelName='LogisticRegression'):
        self.modelName = modelName
        self.scaler = None
        self.pca = None
        self.npca = None
        self.scaler_type = None
        self.X_train = None
        self.X_dev = None
        self.predicted = None

        if modelName == 'LogisticRegression':
            self.model = LogisticRegression(penalty='l2', solver='newton-cg',␣
↪tol=0.001,random_state=1,
                                            multi_class='auto', max_iter=1000,␣
↪verbose=0)
        elif modelName == 'DecisionTree':
            self.model = DecisionTreeClassifier(random_state=1)
        elif modelName == 'RandomForest':
            self.model = RandomForestClassifier(random_state=1)
        elif modelName == 'GradientBoosting':
```

```
            self.model = GradientBoostingClassifier(random_state=1)
        elif modelName == 'KNearestNeighbor':
            self.model = KNeighborsClassifier()
        elif modelName == 'Xgboost':
            self.model = xgb.XGBClassifier(random_state=1)
        elif modelName == 'LinearSVM':
            self.model = LinearSVC(random_state=1, multi_class='crammer_singer')
        elif modelName == 'SVC':
            self.model = SVC(random_state=1)
        elif modelName == 'NeuralNet':
            self.model = MLPClassifier(random_state=1)
        else:
            raise Exception('Model ' + modelName + ' not implemented...')

    def grid_search(self, train_data, train_labels, dev_data, dev_labels,␣
↪params,
                    pca_var_threshold=None, scaler_type=None, print_out=False):
        grd_model = GridSearchCV( self.model,  param_grid = params␣
↪,return_train_score = 1, cv=3, n_jobs=-1)
        self.X_train = train_data
        self.X_dev = dev_data

        if scaler_type is not None:
            [self.X_train, self.X_dev] = self.scale_data(scaler_type , ␣
↪train_data , dev_data)
            if pca_var_threshold is not None:
                [self.X_train, self.X_dev] = self.pca_transform(self.X_train,␣
↪self.X_dev, pca_var_threshold)
        else:
            if pca_var_threshold is not None:
                [self.X_train, self.X_dev] = self.pca_transform(self.X_train,␣
↪self.X_dev, pca_var_threshold)

        grd_model.fit(self.X_train,train_labels)
        predicted= grd_model.predict(self.X_dev)

        if (print_out):
            print ( "\033[1m" ,  self.modelName , "\033[0;0m" )
            print ("Best fit parameters :")
            print (grd_model.best_params_)
            print ("Best fit model F1 score :")
            print (metrics.f1_score(dev_labels, predicted , average='micro'))

        self.classification_report = classification_report(predicted,dev_labels␣
↪)
        self.best_model = grd_model
```

```
        self.best_metrics = metrics.f1_score(dev_labels, predicted ,␣
↪average='micro')
        self.predicted = predicted
        self.cm = metrics.confusion_matrix(dev_labels,predicted)

    def scale_data(self, scaler_type ,  X_train , X_dev):
        self.scaler_type = scaler_type

        if scaler_type == 'MinMax' :
            scaler = MinMaxScaler(feature_range=(0, 1))
        elif scaler_type == 'Robust':
            scaler = RobustScaler()
        elif scaler_type == 'Standard':
            scaler = StandardScaler()
        else:
            print('Unrecognized scaler ' + scaler_type + ' ... reverting to␣
↪MinMax')
            scaler = MinMaxScaler(feature_range=(0, 1))
            self.scaler_type = 'MinMax'

        scaled_X_train = scaler.fit_transform(X_train)
        scaled_X_dev = scaler.transform(X_dev)

        self.scaled_X_train = scaled_X_train
        self.scaled_X_dev = scaled_X_dev
        self.scaler = scaler

        return([scaled_X_train, scaled_X_dev])

    def pca_transform(self, X_train , X_dev, var_threshold=0.95):
        pca = PCA(n_components=X_train.shape[1])
        pca.fit(X_train)
        df_pca = pd.DataFrame()
        df_pca['NumPrinComponents'] = np.arange(start=1 , stop=X_train.shape[1])
        df_pca['ExplainedVariance'] = pd.Series(pca.explained_variance_ratio_)
        df_pca['CumExplainedVariance'] = pd.Series(np.cumsum(pca.
↪explained_variance_ratio_))
        print(df_pca.head())
        npca = df_pca.loc[df_pca.CumExplainedVariance > var_threshold,:].
↪NumPrinComponents.iloc[0]
        self.npca = npca
        pcaModel = PCA(n_components=npca)
        X_train_pca = pcaModel.fit_transform(X_train)
        X_dev_pca = pcaModel.transform(X_dev)
        return([X_train_pca, X_dev_pca])

    def drawConfusionMatrix(self):
```

```
        plt.figure(figsize=(10,10))
        plt.imshow(self.cm, interpolation='nearest', cmap=plt.cm.Wistia)
        classNames = [str(i+1) for i in range(self.cm.shape[0])]
        plt.title('Confusion Matrix For ' + self.modelName, fontsize = 18)
        plt.ylabel('True label', fontsize = 14)
        plt.xlabel('Predicted label', fontsize = 14)
        tick_marks = np.arange(len(classNames))
        plt.xticks(tick_marks, classNames, rotation=0, fontsize=14)
        plt.yticks(tick_marks, classNames, fontsize=14)
        for i in range(len(classNames)):
            for j in range(len(classNames)):
                plt.text(j,i, str(self.cm[i][j]), size='large')
        plt.show()
```

[38]:
```
scaler_experiments = [None, 'MinMax', 'Robust']
pca_experiments = [None, 0.95]
model_experiments = [
    {'modelName':'KNearestNeighbor','params':{ 'n_neighbors' : [3, 5, 7, 9] }},
    {'modelName':'LogisticRegression','params':{ 'C' : [ 0.1, 1.0, 100.0]}},
    {'modelName':'SVC','params':{ 'C' : [10.0]}},
    {'modelName':'LinearSVM','params':{ 'C' : [10.0]}},
    {'modelName':'DecisionTree','params':{ 'max_depth' : [5, 15, 30, 50] }},
    {'modelName':'RandomForest','params':{ 'n_estimators' : [ 20 , 40 ,100 ,␣
 ↪200 ]  }},
    {'modelName':'GradientBoosting','params':{ 'n_estimators' : [ 20 , 40 ,100␣
 ↪,200]}},
    {'modelName':'Xgboost','params': {'objective':['multi:softmax'],'max_depth':
 ↪ [3,5,7],'subsample': [0.8],
          'n_estimators': [20, 40, 100, 200]}},
    {'modelName':'NeuralNet','params':{ 'hidden_layer_sizes' :␣
 ↪[(50,),(50,20),(100,),(100, 20)]}}
]
```

[40]:
```
model_list = []
ctr=1
results_df = pd.DataFrame()
for scaler_type in scaler_experiments:
    for pca_var in pca_experiments:
        for model_type in model_experiments:
            print('-------------------------------------')
            print('Model #' + str(ctr))
            print('Scaler: ' + str(scaler_type))
            print('PCA Variance Threshold: ' + str(pca_var))
            print('Model: ' + model_type['modelName'])
            print('Parameters: ' + str(json.dumps(model_type['params'])))

            model = MLModel(modelName=model_type['modelName'])
```

```
            model.grid_search(train_data, train_labels, dev_data, dev_labels,␣
↪model_type['params'],
                        pca_var_threshold=pca_var, scaler_type=scaler_type,␣
↪print_out=False)

            print('PCA Components: ' + str(model.npca))
            print('Best Parameters: ' + str(json.dumps(model.best_model.
↪best_params_)))
            print('Best F1 Score: ' + str(model.best_metrics))
            print('-------------------------------------')
            model_list.append(model)

            results_df = results_df.append(pd.DataFrame({'Model#':
↪[ctr],'ModelName':[model.modelName],'Scaler':[str(scaler_type)],
                                                'PCA: VarianceThreshold':
↪[str(pca_var)],
                                                'PCA: Number of Components':
↪[str(model.npca)],
                                                'Best Parameters':[str(json.
↪dumps(model.best_model.best_params_))],
                                                'F1 Score':[str(model.
↪best_metrics)]}))

            ctr +=1
```

```
-------------------------------------
Model #1
Scaler: None
PCA Variance Threshold: None
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
PCA Components: None
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.817
-------------------------------------
-------------------------------------
Model #2
Scaler: None
PCA Variance Threshold: None
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
PCA Components: None
Best Parameters: {"C": 100.0}
Best F1 Score: 0.677
-------------------------------------
-------------------------------------
Model #3
```

```
Scaler: None
PCA Variance Threshold: None
Model: SVC
Parameters: {"C": [10.0]}
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.138
-------------------------------------
-------------------------------------
Model #4
Scaler: None
PCA Variance Threshold: None
Model: LinearSVM
Parameters: {"C": [10.0]}
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.527
-------------------------------------
-------------------------------------
Model #5
Scaler: None
PCA Variance Threshold: None
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
PCA Components: None
Best Parameters: {"max_depth": 15}
Best F1 Score: 0.7829999999999999
-------------------------------------
-------------------------------------
Model #6
Scaler: None
PCA Variance Threshold: None
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.838
-------------------------------------
-------------------------------------
Model #7
Scaler: None
PCA Variance Threshold: None
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.802
-------------------------------------
```

```
------------------------------------
Model #8
Scaler: None
PCA Variance Threshold: None
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.854
------------------------------------
------------------------------------
Model #9
Scaler: None
PCA Variance Threshold: None
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
PCA Components: None
Best Parameters: {"hidden_layer_sizes": [100]}
Best F1 Score: 0.584
------------------------------------
------------------------------------
Model #10
Scaler: None
PCA Variance Threshold: 0.95
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.703
------------------------------------
------------------------------------
Model #11
Scaler: None
PCA Variance Threshold: 0.95
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
```

```
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"C": 0.1}
Best F1 Score: 0.521
-------------------------------------
-------------------------------------
Model #12
Scaler: None
PCA Variance Threshold: 0.95
Model: SVC
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"C": 10.0}
Best F1 Score: 0.137
-------------------------------------
-------------------------------------
Model #13
Scaler: None
PCA Variance Threshold: 0.95
Model: LinearSVM
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"C": 10.0}
Best F1 Score: 0.261
-------------------------------------
-------------------------------------
Model #14
Scaler: None
PCA Variance Threshold: 0.95
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
```

```
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"max_depth": 15}
Best F1 Score: 0.639
-------------------------------------
-------------------------------------
Model #15
Scaler: None
PCA Variance Threshold: 0.95
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.705
-------------------------------------
-------------------------------------
Model #16
Scaler: None
PCA Variance Threshold: 0.95
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
2                  3           0.035416              0.983657
3                  4           0.011011              0.994668
4                  5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.627
-------------------------------------
-------------------------------------
Model #17
Scaler: None
PCA Variance Threshold: 0.95
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.725824              0.725824
1                  2           0.222417              0.948241
```

```
2                   3           0.035416              0.983657
3                   4           0.011011              0.994668
4                   5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.675
-------------------------------------
-------------------------------------
Model #18
Scaler: None
PCA Variance Threshold: 0.95
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                   1           0.725824              0.725824
1                   2           0.222417              0.948241
2                   3           0.035416              0.983657
3                   4           0.011011              0.994668
4                   5           0.004072              0.998740
PCA Components: 3
Best Parameters: {"hidden_layer_sizes": [100, 20]}
Best F1 Score: 0.582
-------------------------------------
-------------------------------------
Model #19
Scaler: MinMax
PCA Variance Threshold: None
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
PCA Components: None
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.7810000000000001
-------------------------------------
-------------------------------------
Model #20
Scaler: MinMax
PCA Variance Threshold: None
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
PCA Components: None
Best Parameters: {"C": 100.0}
Best F1 Score: 0.679
-------------------------------------
-------------------------------------
Model #21
Scaler: MinMax
PCA Variance Threshold: None
```

```
Model: SVC
Parameters: {"C": [10.0]}
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.676
-------------------------------------
-------------------------------------
Model #22
Scaler: MinMax
PCA Variance Threshold: None
Model: LinearSVM
Parameters: {"C": [10.0]}
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.674
-------------------------------------
-------------------------------------
Model #23
Scaler: MinMax
PCA Variance Threshold: None
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
PCA Components: None
Best Parameters: {"max_depth": 15}
Best F1 Score: 0.7829999999999999
-------------------------------------
-------------------------------------
Model #24
Scaler: MinMax
PCA Variance Threshold: None
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.836
-------------------------------------
-------------------------------------
Model #25
Scaler: MinMax
PCA Variance Threshold: None
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.801
-------------------------------------
-------------------------------------
Model #26
```

```
Scaler: MinMax
PCA Variance Threshold: None
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.857
-------------------------------------
-------------------------------------
Model #27
Scaler: MinMax
PCA Variance Threshold: None
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
PCA Components: None
Best Parameters: {"hidden_layer_sizes": [100, 20]}
Best F1 Score: 0.763
-------------------------------------
-------------------------------------
Model #28
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.747
-------------------------------------
-------------------------------------
Model #29
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
```

```
PCA Components: 27
Best Parameters: {"C": 100.0}
Best F1 Score: 0.668
-------------------------------------
-------------------------------------
Model #30
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: SVC
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"C": 10.0}
Best F1 Score: 0.65
-------------------------------------
-------------------------------------
Model #31
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: LinearSVM
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"C": 10.0}
Best F1 Score: 0.649
-------------------------------------
-------------------------------------
Model #32
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
```

```
PCA Components: 27
Best Parameters: {"max_depth": 15}
Best F1 Score: 0.737
-------------------------------------
-------------------------------------
Model #33
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.793
-------------------------------------
-------------------------------------
Model #34
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
4                  5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.7410000000000001
-------------------------------------
-------------------------------------
Model #35
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.217181              0.217181
1                  2           0.178087              0.395268
2                  3           0.071100              0.466368
3                  4           0.056114              0.522482
```

```
4                   5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.787
-------------------------------------
-------------------------------------
Model #36
Scaler: MinMax
PCA Variance Threshold: 0.95
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                   1           0.217181              0.217181
1                   2           0.178087              0.395268
2                   3           0.071100              0.466368
3                   4           0.056114              0.522482
4                   5           0.035903              0.558385
PCA Components: 27
Best Parameters: {"hidden_layer_sizes": [100, 20]}
Best F1 Score: 0.767
-------------------------------------
-------------------------------------
Model #37
Scaler: Robust
PCA Variance Threshold: None
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
PCA Components: None
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.762
-------------------------------------
-------------------------------------
Model #38
Scaler: Robust
PCA Variance Threshold: None
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
PCA Components: None
Best Parameters: {"C": 100.0}
Best F1 Score: 0.678
-------------------------------------
-------------------------------------
Model #39
Scaler: Robust
PCA Variance Threshold: None
Model: SVC
Parameters: {"C": [10.0]}
```

```
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.737
-------------------------------------
-------------------------------------
Model #40
Scaler: Robust
PCA Variance Threshold: None
Model: LinearSVM
Parameters: {"C": [10.0]}
PCA Components: None
Best Parameters: {"C": 10.0}
Best F1 Score: 0.684
-------------------------------------
-------------------------------------
Model #41
Scaler: Robust
PCA Variance Threshold: None
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
PCA Components: None
Best Parameters: {"max_depth": 15}
Best F1 Score: 0.7829999999999999
-------------------------------------
-------------------------------------
Model #42
Scaler: Robust
PCA Variance Threshold: None
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.838
-------------------------------------
-------------------------------------
Model #43
Scaler: Robust
PCA Variance Threshold: None
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.802
-------------------------------------
-------------------------------------
Model #44
Scaler: Robust
PCA Variance Threshold: None
```

```
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
PCA Components: None
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.856
-------------------------------------
-------------------------------------
Model #45
Scaler: Robust
PCA Variance Threshold: None
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
PCA Components: None
Best Parameters: {"hidden_layer_sizes": [100, 20]}
Best F1 Score: 0.805
-------------------------------------
-------------------------------------
Model #46
Scaler: Robust
PCA Variance Threshold: 0.95
Model: KNearestNeighbor
Parameters: {"n_neighbors": [3, 5, 7, 9]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"n_neighbors": 3}
Best F1 Score: 0.756
-------------------------------------
-------------------------------------
Model #47
Scaler: Robust
PCA Variance Threshold: 0.95
Model: LogisticRegression
Parameters: {"C": [0.1, 1.0, 100.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"C": 100.0}
```

```
Best F1 Score: 0.67
-----------------------------------
-----------------------------------
Model #48
Scaler: Robust
PCA Variance Threshold: 0.95
Model: SVC
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"C": 10.0}
Best F1 Score: 0.756
-----------------------------------
-----------------------------------
Model #49
Scaler: Robust
PCA Variance Threshold: 0.95
Model: LinearSVM
Parameters: {"C": [10.0]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"C": 10.0}
Best F1 Score: 0.668
-----------------------------------
-----------------------------------
Model #50
Scaler: Robust
PCA Variance Threshold: 0.95
Model: DecisionTree
Parameters: {"max_depth": [5, 15, 30, 50]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"max_depth": 30}
```

```
Best F1 Score: 0.697
-------------------------------------
-------------------------------------
Model #51
Scaler: Robust
PCA Variance Threshold: 0.95
Model: RandomForest
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.807
-------------------------------------
-------------------------------------
Model #52
Scaler: Robust
PCA Variance Threshold: 0.95
Model: GradientBoosting
Parameters: {"n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"n_estimators": 200}
Best F1 Score: 0.746
-------------------------------------
-------------------------------------
Model #53
Scaler: Robust
PCA Variance Threshold: 0.95
Model: Xgboost
Parameters: {"objective": ["multi:softmax"], "max_depth": [3, 5, 7],
"subsample": [0.8], "n_estimators": [20, 40, 100, 200]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
```

```
Best Parameters: {"max_depth": 7, "n_estimators": 200, "objective":
"multi:softmax", "subsample": 0.8}
Best F1 Score: 0.807
-------------------------------------
-------------------------------------
Model #54
Scaler: Robust
PCA Variance Threshold: 0.95
Model: NeuralNet
Parameters: {"hidden_layer_sizes": [[50], [50, 20], [100], [100, 20]]}
   NumPrinComponents  ExplainedVariance  CumExplainedVariance
0                  1           0.240048              0.240048
1                  2           0.179523              0.419571
2                  3           0.156816              0.576387
3                  4           0.097485              0.673872
4                  5           0.055970              0.729843
PCA Components: 20
Best Parameters: {"hidden_layer_sizes": [100, 20]}
Best F1 Score: 0.813
-------------------------------------
```

1.3 Part 3: Results & Evaluation

1.3.1 Best and worst model analysis

The best scoring models are displayed below :

[42]:
```
results_df.sort_values(by='F1 Score', ascending=False).head(5)
```

[42]:

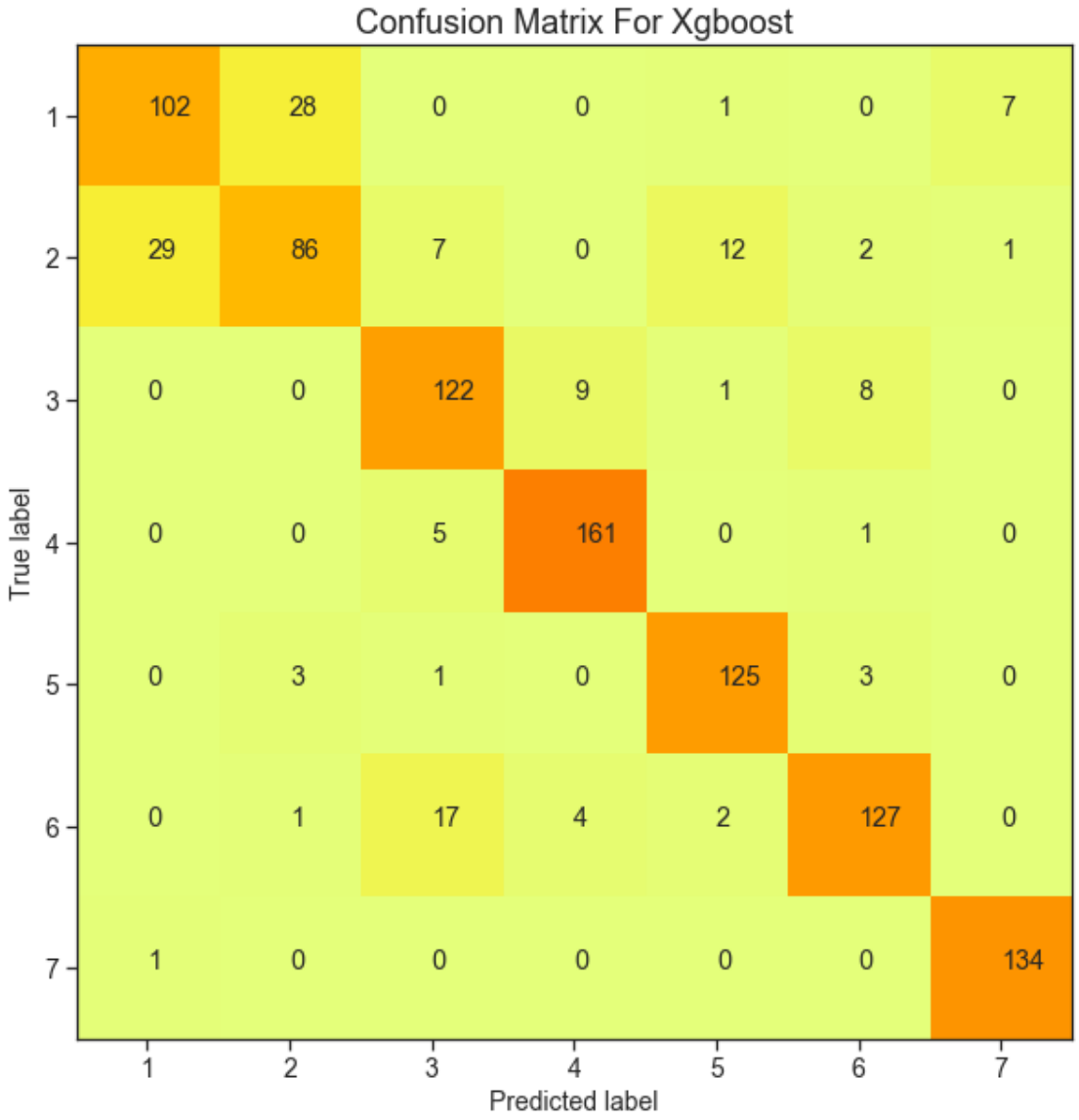
	Model#	ModelName	Scaler	PCA: VarianceThreshold	PCA: Number of Components	Best Parameters	F1 Score
0	26	Xgboost	MinMax	None	None	{"max_depth": 7, "n_estimators": 200, "objecti…	0.857
0	44	Xgboost	Robust	None	None	{"max_depth": 7, "n_estimators": 200, "objecti…	0.856
0	8	Xgboost	None	None	None	{"max_depth": 7, "n_estimators": 200, "objecti…	0.854
0	42	RandomForest	Robust	None	None	{"n_estimators": 200}	0.838
0	6	RandomForest	None	None	None		

```
0                                {"n_estimators": 200}     0.838
```

We observe that XGBoost and Random Forest performs the best. XGBoost with MinMax scaler appears to have the highest accuracy.

Confusion matrix for the best model :

[43]:
```
best_model_id = results_df.sort_values(by='F1 Score',␣
 ↪ascending=False)['Model#'].values[0]
model_list[best_model_id-1].drawConfusionMatrix()
```



The worst scoring models are displayed below :

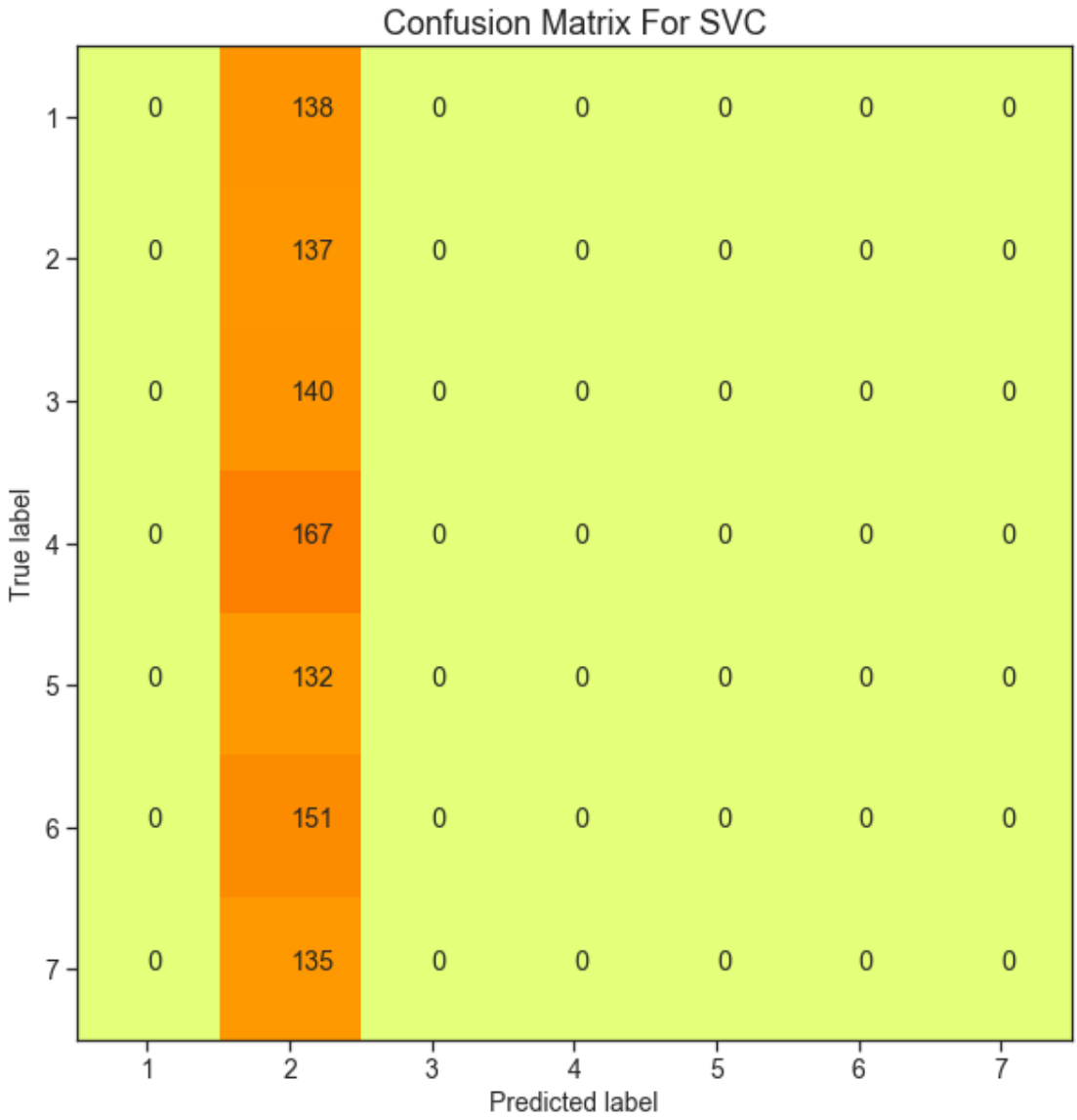
[44]:
```
results_df.sort_values(by='F1 Score').head(5)
```

[44]:

	Model#	ModelName	Scaler	PCA: VarianceThreshold	PCA: Number of Components	Best Parameters	F1 Score
0	12	SVC	None	0.95	3	{"C": 10.0}	0.137
0	3	SVC	None	None	None	{"C": 10.0}	0.138
0	13	LinearSVM	None	0.95	3	{"C": 10.0}	0.261
0	11	LogisticRegression	None	0.95	3	{"C": 0.1}	0.521
0	4	LinearSVM	None	None	None	{"C": 10.0}	0.527

We observe that SVM Models are worst performing. Below is the confusion matrix for the worst performing model. It predicts the same class for all inputs.

[45]:
```
worst_model_id = results_df.sort_values(by='F1 Score')['Model#'].values[0]
model_list[worst_model_id-1].drawConfusionMatrix()
```



Upon investigating further, SVMs appear to perform poorly when the data is not scaled. This is observed from the table below.

[46]:
```
#SVM models
results_df.loc[results_df.ModelName.str.contains('SV'),:]
```

[46]:

	Model#	ModelName	Scaler	PCA: VarianceThreshold	PCA: Number of Components	\
0	3	SVC	None	None	None	
0	4	LinearSVM	None	None	None	
0	12	SVC	None	0.95	3	
0	13	LinearSVM	None	0.95	3	
0	21	SVC	MinMax	None	None	

```
0      22  LinearSVM  MinMax                  None                  None
0      30        SVC  MinMax                  0.95                    27
0      31  LinearSVM  MinMax                  0.95                    27
0      39        SVC  Robust                  None                  None
0      40  LinearSVM  Robust                  None                  None
0      48        SVC  Robust                  0.95                    20
0      49  LinearSVM  Robust                  0.95                    20

  Best Parameters F1 Score
0     {"C": 10.0}    0.138
0     {"C": 10.0}    0.527
0     {"C": 10.0}    0.137
0     {"C": 10.0}    0.261
0     {"C": 10.0}    0.676
0     {"C": 10.0}    0.674
0     {"C": 10.0}     0.65
0     {"C": 10.0}    0.649
0     {"C": 10.0}    0.737
0     {"C": 10.0}    0.684
0     {"C": 10.0}    0.756
0     {"C": 10.0}    0.668
```

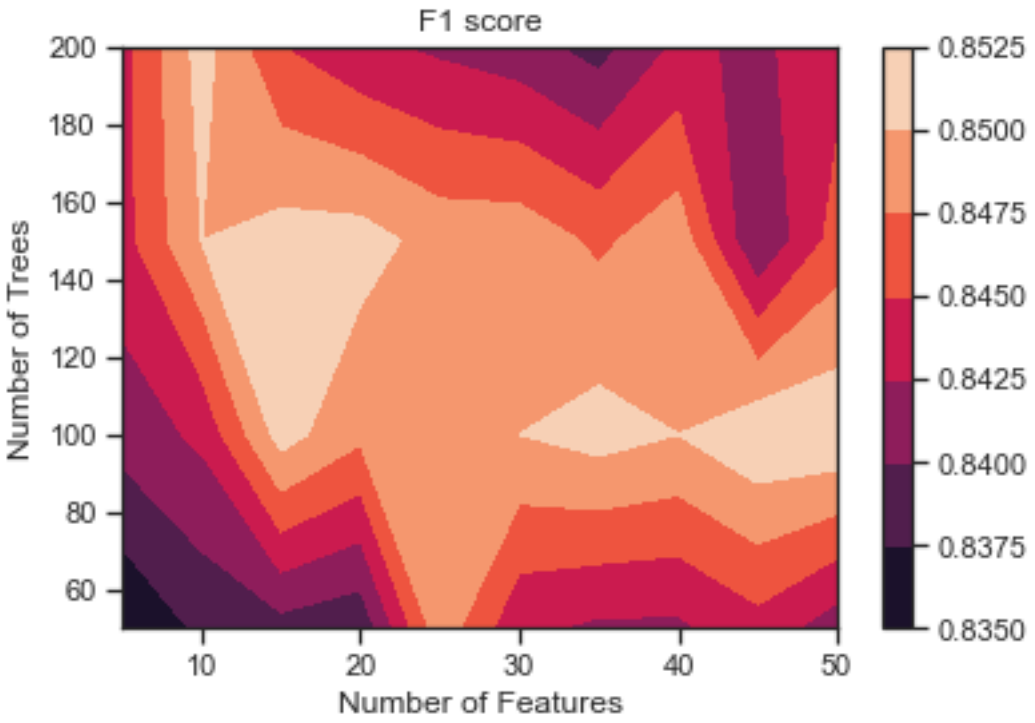
1.3.2 Best Model: Hyperparameter Tuning

Since our best performing model is Random Forest and XGBoost, we investigated Random Forest further with additional hyperparameter tuning.

[ ]:
```
scaler_type ='Robust'
pca_var = None

max_features = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
n_estimators = [ 50, 100, 150, 200 ]
num_features = []
num_trees = []
f1_scores = []
models = []
model_no = []
ctr = 1
for nfeatures in max_features:
    for ntrees in n_estimators:
        model_experiments = [

            {'modelName':'RandomForest','params':{ 'n_estimators' : [ ntrees ] ␣
↪, 'max_features':[nfeatures]}}
        ]
        for model_type in model_experiments:
            print('-------------------------------------')
```

```
            print('Scaler: ' + str(scaler_type))
            print('PCA Variance Threshold: ' + str(pca_var))
            print('Model: ' + model_type['modelName'])
            print('Parameters: ' + str(json.dumps(model_type['params'])))

            model = MLModel(modelName=model_type['modelName'])
            model.grid_search(train_data, train_labels, dev_data, dev_labels,␣
 ↪model_type['params'],
                      pca_var_threshold=pca_var, scaler_type=scaler_type,␣
 ↪print_out=False)

            print('PCA Components: ' + str(model.npca))
            print('Best Parameters: ' + str(json.dumps(model.best_model.
 ↪best_params_)))
            print('Best F1 Score: ' + str(model.best_metrics))
            print('-------------------------------------')
            models.append(model)
            num_features.append(nfeatures)
            num_trees.append(ntrees)
            f1_scores.append(model.best_metrics)
            model_no.append(ctr)
            ctr += 1
best_model_df = pd.DataFrame({'ModelNo':model_no,'n_estimators':
 ↪num_trees,'n_features':num_features,'f1_scores':f1_scores})
```

[49]:
```
plt.contourf(max_features,n_estimators, np.array(f1_scores).
 ↪reshape((len(n_estimators),len(max_features))))
plt.title('F1 score')
plt.xlabel('Number of Features')
plt.ylabel('Number of Trees')
plt.colorbar()
plt.show()
```



One of the benefits of trees is that it allows you to easily look at feature importances. Below are the top 10 important features for the best model.

[68]:
```
best_model_id = best_model_df.sort_values(by='f1_scores',␣
↪ascending=False)['ModelNo'].values[0]
res_model = models[best_model_id-1]
feature_importances = pd.DataFrame({'Features':data.columns[1:53],
                                    'Importance':res_model.best_model.
↪best_estimator_.feature_importances_})
feature_importances.sort_values(by='Importance', ascending=False).head(10)
```

[68]:

	Features	Importance
0	Elevation	0.328307
5	Horizontal_Distance_To_Roadways	0.092003
9	Horizontal_Distance_To_Fire_Points	0.075111
3	Horizontal_Distance_To_Hydrology	0.063705
6	Hillshade_9am	0.051584
4	Vertical_Distance_To_Hydrology	0.048288
1	Aspect	0.041068
13	Wilderness_Area4	0.040914
7	Hillshade_Noon	0.038436
8	Hillshade_3pm	0.035040

1.3.3 Prediction on Test Data

We run the prediction on test data for the best model.

[59]:
```
scaler_type ='Robust'
pca = None
best_model_id = best_model_df.sort_values(by='f1_scores',␣
 ↪ascending=False)['ModelNo'].values[0]
res_model = models[best_model_id-1]
if scaler_type is not None:
    scaled_X_test = res_model.scaler.transform(test_data)
    if model.pca is not None:
        scaled_pca_X_test = res_model.pca.transform(scaled_X_test)
        predictions = res_model.predict(scaled_pca_X_test)
    else:
        predictions = res_model.best_model.best_estimator_.
 ↪predict(scaled_X_test)
else:
    predictions = res_model.best_model.best_estimator_(X_test)

print(classification_report(predictions,test_labels ))

print(metrics.f1_score(test_labels, predictions , average='micro'))
print(metrics.confusion_matrix(test_labels,predictions))
```

```
              precision    recall  f1-score   support

           1       0.77      0.80      0.78       292
           2       0.68      0.76      0.72       254
           3       0.79      0.87      0.83       271
           4       0.96      0.91      0.94       300
           5       0.95      0.87      0.91       356
           6       0.88      0.85      0.86       324
           7       0.96      0.93      0.95       323

    accuracy                           0.86      2120
   macro avg       0.86      0.86      0.86      2120
weighted avg       0.87      0.86      0.86      2120

0.8589622641509433
[[233  43   1   0   5   0  20]
 [ 49 193   4   0  30   7   2]
 [  0   4 235  15   9  33   0]
 [  0   0   6 274   0   4   0]
 [  0   9   3   0 311   6   0]
 [  0   4  22  11   1 274   0]
 [ 10   1   0   0   0   0 301]]
```

1.3.4 Summary of Results

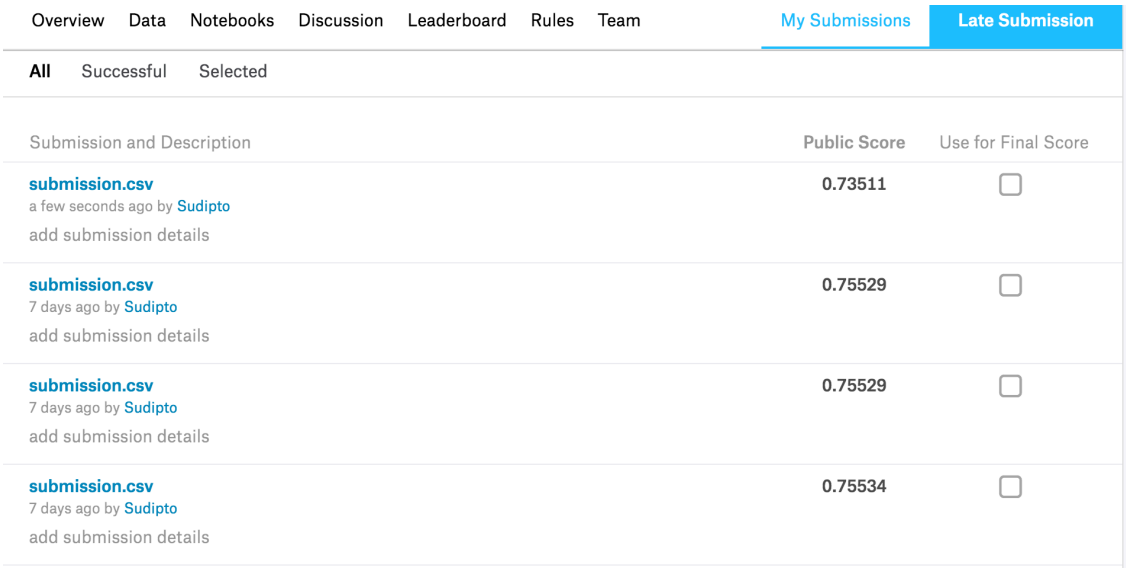
- After looping through multiple models, trees appear to yield the best results. More specifically, the top performing models appear to be Random Forest & XGBoost - yielded F1 Score of .857.
- PCA did not appear to help much for Tree Based Models. Additionally, scaling does not appear to matter much for tree based methods
- We decided to go with Random Forest - even the best performing models has difficulty differentiating between cover types 1 & 2.
- The worst performing models are SVM - Linear SVCs & SVMs appear to severely underperform without any scaling (~14%). We hit 75% on SVC while using Robust Scaler. Scaling is important to draw a hyperplane through the data. SVCs have difficult with non scaled data, since if one feature has very large values (e.g. Elevation), it will dominate the other features when calculating the distance.
- Since Random Forest is our best model, we dug deeper to improve cover type classification performance even further.
- We hit 85.25% while looping through these additional features. We can see several potential local minima when looking generating a heatmap of F1 scores.
- As far as significant features go, we decided to look at feature importances generated from the best Random Forest model. The most important features appear to be:
  - Elevation
  - Horizontal Distance to Roadways
  - Horizontal Distance to Firepoints
  - Horizontal Distance to Hydrology
  - Vertical Distance to Hydrology
- From our box plot in the EDA section, where we look at the different variables broken down by Cover_Type, we can clearly see that Elevation appear to be clearly delineated between different labels.

1.3.5 Submission to Kaggle

We output the data below and submit the best results to Kaggle (screenshot included below).

[62]:
```
data_predict = pd.read_csv("test.csv")
predict_data = np.array(data_predict.as_matrix(columns=data.columns[1:55]))
predict_data = np.delete(predict_data, column_to_remove, axis=1)
scaler_type ='Robust'
pca = None
best_model_id = best_model_df.sort_values(by='f1_scores',␣
 ↪ascending=False)['ModelNo'].values[0]
res_model = models[best_model_id-1]
if scaler_type is not None:
    scaled_X_predict = res_model.scaler.transform(predict_data)
    if model.pca is not None:
```

```
        scaled_pca_X_predict = res_model.pca.transform(scaled_X_predict)
        predictions = res_model.predict(scaled_pca_X_predict)
    else:
        predictions = res_model.best_model.best_estimator_.
↪predict(scaled_X_predict)
else:
    predictions = res_model.best_model.best_estimator_(X_test)
df_predicted = pd.DataFrame()
df_predicted['Id'] = data_predict['Id']
df_predicted['Cover_Type'] = predictions
export_csv = df_predicted.to_csv('submission.csv' , header=True , index = None)
```



1.4 Part 4: Appendix

1.4.1 Room for Improvement

There are several areas of improvement that may help to bump up cover type classification accuracy even further.

- Undoing One-hot Encoding of Dataset for Random Forest - the way the variable soil type was encoded in the provided data set is very similar to one hot encoding. However, when one hot encoding a variable, we run the risk of dispersing the feature importance of soil type across the 40 one hot encoded variables.
  - We discussed undoing the one hot encoding, but ran into instances where one row would have 2 soil types.
  - We discussed either removing this from our training/dev data, or creating a "new combination soil type", but decided to leave soil type as it is.
- We could have added more hyperparameters for all of the Machine Learning models - however, we were extremely mindful of adding additional compute time.
- Our approach for Neural Networks was relatively simple, and only used the built in one with

Scikit learn. We could have looped through additional hyper parameters for Neural Networks to bump up performance.

1.4.2 Additional approaches

Ensemble Approach 1 We have observed from the confusion matrix that the models are most confused between Cover Type 1 and 2. In Ensemble Approach 1 we do the following :

1. We train a Random Forest model (Model 1) to predict whether the Cover Type is 1, 2 (we call this class 0) or 3 , 4 , 5 , 6 ,7 (we call this class 1)
2. We train a second Random Forest model (Model 2) only for Cover Type 1, 2 to predict Cover Type
3. We train a third Random Forest model (Model 3) only for Cover Types 3,4,5,6,7 to predict Cover Type
4. For prediction we first predict the class based on Model 1. In case it predicts 0 then we use Model 2 to predict the Cover Type. In case it predicts 1 then we use Model 3 to predict the Cover Type

[87]:
```
#Prepare train and dev data for model 1
test_model_1 = MLModel(modelName='RandomForest')
test_model_2 = MLModel(modelName='RandomForest')
test_model_3 = MLModel(modelName='RandomForest')
pca_components = None
scaler_type = None
params = {  'n_estimators' : [ 20, 30 , 40 ,50 , 200 ]  }
train_labels_ens_1 = np.where(train_labels> 2 , 1 , 0)
dev_labels_ens_1 = np.where(dev_labels> 2 , 1 , 0)
test_model_1.grid_search(train_data, train_labels_ens_1, dev_data,␣
 ↪dev_labels_ens_1, params, pca_var_threshold=pca_components,␣
 ↪scaler_type=scaler_type , print_out=False)
train_labels_ens_2 = train_labels[np.where(train_labels <= 2)]
train_data_ens_2 = train_data[np.where(train_labels <= 2)]
test_model_2.grid_search(train_data_ens_2, train_labels_ens_2, dev_data,␣
 ↪dev_labels, params, pca_var_threshold=pca_components,␣
 ↪scaler_type=scaler_type , print_out=False)
train_labels_ens_3 = train_labels[np.where(train_labels > 2)]
train_data_ens_3 = train_data[np.where(train_labels > 2)]
test_model_3.grid_search(train_data_ens_3, train_labels_ens_3, dev_data,␣
 ↪dev_labels, params, pca_var_threshold=pca_components,␣
 ↪scaler_type=scaler_type, print_out=False)
```

[88]:
```
def predict_approach_ensemble_7(data , labels , test_model_1 , test_model_2,␣
 ↪test_model_3):
    predicted_1 = test_model_1.best_model.predict(data)
    predicted_2 = test_model_2.best_model.predict(data)
    predicted_3 = test_model_3.best_model.predict(data)
    predicted_final = np.empty(labels.size , dtype=int)
    for i in range(labels.size):
```

```
        if predicted_1[i] == 0:
            predicted_final[i] = predicted_2[i]
        else:
            predicted_final[i] = predicted_3[i]
    return predicted_final
predicted = predict_approach_ensemble_7(dev_data , dev_labels , test_model_1 ,␣
 ↪test_model_2, test_model_3)
print ("Best fit model F1 score :")
print(metrics.f1_score(dev_labels, predicted , average='micro'))
```

```
Best fit model F1 score :
0.845
```

Ensemble Approach 2 We have observed from the confusion matrix that the models are most confused between Cover Type 1 and 2. In Ensemble Approach 2 we do the following :

1. We train a Random Forest model
   - Model 1 to predict class type 0, 3, 4, 5, 6, 7, where:
     - 0 represents cover type 1, 2
     - 3,4,5,6,7 reperesents the original cover types
2. We train a second Random Forest model (Model 2) only for cover type 1, 2
3. For prediction we first predict the class based on Model 1. In case it predicts 0 then we use Model 2 to further classify between 1 & 2.

[89]:
```
train_labels_ens_1 = np.where(train_labels> 2 , train_labels , 0 )
dev_labels_ens_1 = np.where(dev_labels> 2 , dev_labels , 0)
```

[90]:
```
test_model_1 = MLModel(modelName='RandomForest')
test_model_2 = MLModel(modelName='RandomForest')
pca_components = None
scaler_type = None
params = {  'n_estimators' : [ 20, 30 , 40 ,50 , 200 ]  }
train_labels_ens_1 = np.where(train_labels> 2 , train_labels , 0 )
dev_labels_ens_1 = np.where(dev_labels> 2 , dev_labels , 0)
test_model_1.grid_search(train_data, train_labels_ens_1, dev_data,␣
 ↪dev_labels_ens_1, params, pca_var_threshold=pca_components,␣
 ↪scaler_type=scaler_type , print_out=False)
train_labels_ens_2 = train_labels[np.where(train_labels <= 2)]
train_data_ens_2 = train_data[np.where(train_labels <= 2)]
test_model_2.grid_search(train_data_ens_2, train_labels_ens_2, dev_data,␣
 ↪dev_labels, params, pca_var_threshold=pca_components,␣
 ↪scaler_type=scaler_type , print_out=False)
```

[91]:
```
def predict_approach_ensemble_8(data , labels , test_model_1 , test_model_2):
    predicted_1 = test_model_1.best_model.predict(data)
    predicted_2 = test_model_2.best_model.predict(data)
    predicted_final = predicted_1
```

```
    for i in range(predicted_1.size):
        if predicted_1[i] == 0:
            predicted_final[i] = predicted_2[i]
    return predicted_final
predicted = predict_approach_ensemble_8(dev_data , dev_labels , test_model_1 ,␣
 ↪test_model_2)
print ("Best fit model F1 score :")
print(metrics.f1_score(dev_labels, predicted , average='micro'))
```

```
Best fit model F1 score :
0.847
```

Unsupervised learning approach - GMM with PCA We also tried an unsupervised learning approach, where we loop through a combination of Gaussian Mixture models and dimensionality reduction via Principal Component Analysis in order to try and predict labels. We intialize 7 Gaussian Mixture Models for each label. Then we loop through a number of clusters, principal components, and covariance matrix types for GMMs.

Utilizing this approach yields a dev accuracy of approximately 70% in the best scenario.

[11]:
```
## Gaussian Mixture Models

#The following code will run instantiate 7 instances of Gaussian Mixture Models.
 ↪ (1 for each label.)
#This code will then loop through various combinations of principal components␣
 ↪until it finds a combination
#of GMMs/PCA components that will best predict the dev labels.

def gaussian_mixture_model(train_data_input, train_labels_input,␣
 ↪test_data_input, test_labels_input):
    pd.set_option('display.width', 1000) #set display window
    def GMM_PCA_test(pca_components, mixture_components, covariance_type):
        pca = PCA(n_components=pca_components)
        principal_components_train = pca.fit_transform(train_data_input)␣
 ↪#        Fit the model with X and apply the dimensionality reduction on X.

        #Initialize Gaussian Mixture Models (1 for each label)
        gmm1 = GaussianMixture(n_components = mixture_components,␣
 ↪covariance_type = covariance_type) #first gmm for positive examples
        gmm2 = GaussianMixture(n_components = mixture_components,␣
 ↪covariance_type = covariance_type) #second gmm for negative examples
        gmm3 = GaussianMixture(n_components = mixture_components,␣
 ↪covariance_type = covariance_type) #third gmm for negative examples
        gmm4 = GaussianMixture(n_components = mixture_components,␣
 ↪covariance_type = covariance_type) #fourth gmm for negative examples
        gmm5 = GaussianMixture(n_components = mixture_components,␣
 ↪covariance_type = covariance_type) #fifth gmm for negative examples
```

```
        gmm6 = GaussianMixture(n_components = mixture_components,␣
↪covariance_type = covariance_type) #sixth gmm for negative examples
        gmm7 = GaussianMixture(n_components = mixture_components,␣
↪covariance_type = covariance_type) #seventh gmm for negative examples

        #T/F Boolean mask depending on value of train_label
        tf_train_array_1 = (train_labels_input == 1) #T/F boolean masking array␣
↪if train_label = 1
        tf_train_array_2 = (train_labels_input == 2) #T/F boolean masking array␣
↪if train_label = 2
        tf_train_array_3 = (train_labels_input == 3) #T/F boolean masking array␣
↪if train_label = 3
        tf_train_array_4 = (train_labels_input == 4) #T/F boolean masking array␣
↪if train_label = 4
        tf_train_array_5 = (train_labels_input == 5) #T/F boolean masking array␣
↪if train_label = 5
        tf_train_array_6 = (train_labels_input == 6) #T/F boolean masking array␣
↪if train_label = 6
        tf_train_array_7 = (train_labels_input == 7) #T/F boolean masking array␣
↪if train_label = 7

        #split principal components transformed array into 7 (by categories of␣
↪covertype/label)
        pc_train_1 = principal_components_train[tf_train_array_1]
        pc_train_2 = principal_components_train[tf_train_array_2]
        pc_train_3 = principal_components_train[tf_train_array_3]
        pc_train_4 = principal_components_train[tf_train_array_4]
        pc_train_5 = principal_components_train[tf_train_array_5]
        pc_train_6 = principal_components_train[tf_train_array_6]
        pc_train_7 = principal_components_train[tf_train_array_7]

        #Fit split Principal Component data to respective Gaussina Mixture Model
        gmm1.fit(pc_train_1)
        gmm2.fit(pc_train_2)
        gmm3.fit(pc_train_3)
        gmm4.fit(pc_train_4)
        gmm5.fit(pc_train_5)
        gmm6.fit(pc_train_6)
        gmm7.fit(pc_train_7)

        pca_dim_red_test = pca.transform(test_data_input) #Apply dimensionality␣
↪reduction to test_data/dev_data

        #Fit PCA reduced test/dev data to our Gaussian Mixture Model
        gmm_test1 = np.exp(gmm1.score_samples(pca_dim_red_test))
        gmm_test2 = np.exp(gmm2.score_samples(pca_dim_red_test))
```

```
        gmm_test3 = np.exp(gmm3.score_samples(pca_dim_red_test))
        gmm_test4 = np.exp(gmm4.score_samples(pca_dim_red_test))
        gmm_test5 = np.exp(gmm5.score_samples(pca_dim_red_test))
        gmm_test6 = np.exp(gmm6.score_samples(pca_dim_red_test))
        gmm_test7 = np.exp(gmm7.score_samples(pca_dim_red_test))

        stacked_gmms = np.stack((gmm_test1, gmm_test2, gmm_test3, gmm_test4,␣
↪gmm_test5, gmm_test6, gmm_test7))

        predicted_results = np.argmax(stacked_gmms, axis=0) + 1 #return maximum␣
↪value of stacked array and return array position + 1 (since array position␣
↪starts at 0 and labels start at 1)

        comp_array = (predicted_results == test_labels_input) * 1 #compare our␣
↪predicted results against our test labels

        accuracy_numerator = np.sum(comp_array) #how many matches we got

        accuracy_denominator = test_labels_input.shape[0] #length of array

        accuracy = accuracy_numerator/accuracy_denominator #calculate accuracy

        #print("The accuracy is: " + str(accuracy))

        return (accuracy, predicted_results)

    #initialize covariance matrix type vars to loop through

    cov_matrix_type = ['spherical', 'diag', 'tied', 'full']

    #initiialize vars
    optimum_pca_components = 0
    optimum_mixture_components = 0
    optimum_cov_matrix_type = 0
    max_accuracy = 0
    best_labels = np.array([0])

    #lists to append to for our Pandas table/output
    n_components_array = []
    pca_components_array = []
    mixture_components_array =  []
    cov_matrix_type_array = []
    cur_accuracy_array = []

    #( (3 mean vector + 3 covariance matrix) x 2 components ) x 2 classes = 24␣
↪parameters
```

```
   for i in range(1,16): #num of PCA components we're testing - 1 through 10
       for j in range(1,16): #num of GMM mixture components we're testing - 1␣
↪through 10
           for k in cov_matrix_type: #loop through covariance matrix types
               if ((i+i) * j * 2) <= 100: #try combination of parameters  if␣
↪there would be 1000 or fewer parameters
                   model = GMM_PCA_test(i, j, k)
                   cur_accuracy = model[0]
                   #print("# of PCA Components: " + str(i) + ", # of Mixture␣
↪Components: " + str(j) + ", covariance matrix type: " + str(k) + ", accuracy:
↪ " + str(cur_accuracy))
                   pca_components_array.append(i)
                   mixture_components_array.append(j)
                   cov_matrix_type_array.append(k)
                   cur_accuracy_array.append(cur_accuracy)
                   n_components_array.append((i+i) * j * 2)
                   if cur_accuracy > max_accuracy:
                       max_accuracy = cur_accuracy
                       optimum_pca_components = i
                       optimum_mixture_components = j
                       optimum_cov_matrix_type = k
                       best_labels = model[1]
                   else:
                       continue
               else:
                   continue

   df = pd.DataFrame({"PCA_Components": pca_components_array,
                      "GMM_Mixture_Components":mixture_components_array,
                      "Covariance_Matrix_type": cov_matrix_type_array,
                      "N_Components": n_components_array,
                      "Accuracy":cur_accuracy_array,
                    })


   print("******Finding the best combination of PCA Components, Mixture␣
↪Components, Covariance Matrix Type******")
   display(df.sort_values(by=['Accuracy'], ascending = False)) #display pd␣
↪table sorted by highest accuracy value

   print("The combination of parameters that resulted in the best accuracy was:
↪")
   print("Optimum number of PCA components: " + str(optimum_pca_components))
   print("Optimum number of GMM components: " +␣
↪str(optimum_mixture_components))
   print("Covariance Type: " + str(optimum_cov_matrix_type))
```

```
    print("Max accuracy: " + str(max_accuracy))



#Fit_all_data_scalers will scale the data according to the type of data passed␣
 ↪in.

def fit_all_data_scalers( scaler_type ,  train_data , test_data , dev_data ):
    column_list = ['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']
    for column_name in column_list:
        if scaler_type == 'MinMax' :
            scaler = MinMaxScaler(feature_range=(0, 1))
        if scaler_type == 'Robust':
            scaler = RobustScaler()
        else:
            scaler = StandardScaler()
        column_index = train_df.columns.get_loc(column_name) - 1
        train_data[:,column_index] = scaler.fit_transform(train_data[:
 ↪,column_index].reshape(-1,1)).reshape(-1)
        test_data[:,column_index] =  (scaler.transform(test_data[:
 ↪,column_index].reshape(-1,1)).reshape(-1))
        dev_data[:,column_index] =  (scaler.transform(dev_data[:,column_index].
 ↪reshape(-1,1)).reshape(-1))
    return train_data , test_data , dev_data


scaler_type = 'MinMax'
train_data_trans , test_data_trans , dev_data_trans = fit_all_data_scalers(␣
 ↪scaler_type ,  np.copy(np.float32(train_data)) , np.copy(np.
 ↪float32(test_data)) , np.copy(np.float32(dev_data)) )


#Run Gaussian Mixture Model/PCA approach with scaled data
gaussian_mixture_model(train_data_trans, train_labels, dev_data_trans,␣
 ↪dev_labels)
```

******Finding the best combination of PCA Components, Mixture Components, Covariance Matrix Type******

	PCA_Components	GMM_Mixture_Components	Covariance_Matrix_type	N_Components	Accuracy
250	12	1	tied	48	0.705
254	12	2	tied	96	0.705
251	12	1	full	48	0.704

```
242              11                    1                  tied           44     0.703
243              11                    1                  full           44     0.703
..              ...                  ...                   ...          ...       ...
31                1                    8                  full           32     0.280
44                1                   12             spherical           48     0.279
17                1                    5                  diag           20     0.278
55                1                   14                  full           56     0.276
28                1                    8             spherical           32     0.276

[268 rows x 5 columns]


The combination of parameters that resulted in the best accuracy was:
Optimum number of PCA components: 12
Optimum number of GMM components: 1
Covariance Type: tied
Max accuracy: 0.705
```