International Institute of Information Technology Bangalore

PROJECT ELECTIVE REPORT

CS-901

Semantic Segmentation of 3D point cloud

GitHub Repository: https://github.com/Bambo0st/Segmentation3D_PointCloud

Adithya Nagaraja Somasalle

January 2024- May 2024

Contents

1 Objective

To develop a machine learning model for semantic segmentation of 3D point cloud.

2 Dataset

The dataset used for training the model is Hessigheim 3D (H3D) dataset. H3D dataset contains train folder and test folder for Mar2018 and Mar2019. Each folder contains .laz files of point clouds over a village in Germany.

It contains 11 classes namely Low Vegetation, Impervious Surface, Vehicle, Urban Furniture, Roof, Facade, Shrub, Tree, Soil, Vertical Surface and Chimney.

3 Segmentation Visualization Using Cloud Compare

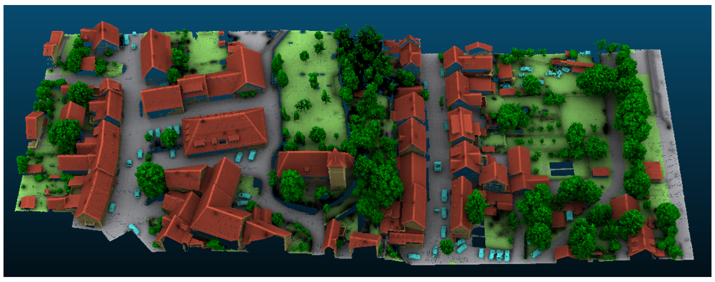


Figure 1: Segmentation on portion of train dataset
,

Here I used the classification values to separate each of the classes and manually gave custom colours to each of these classes and merge all the custom coloured classes.

4 Feature Extraction

Cloud Compare offers an option to calculate several features of a point cloud like PCA, Eigen value, linearity etc. Feature extraction is performed on train data and stored in txt format for later use.

5 Training Classifier Cloud Compare's Canupo plugin

Cloud Compare offers Canupo plugin which can train classifier but the drawback is that is only a binary classifier. It works great for predicting classes like trees. For training multiple classes we will have to run the classifier nearly $logn$ number of times where n is the number of classes. Clearly this approach is cumbersome.

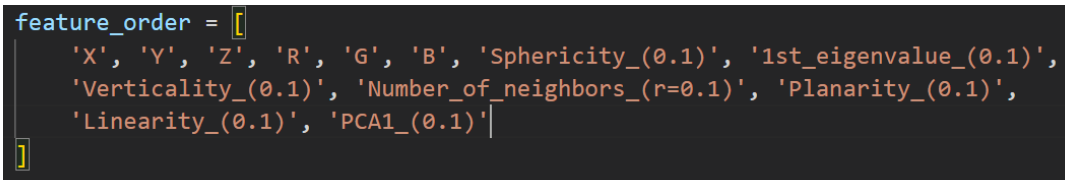
6 Training Classifier with machine learning

6.1 $train - test\ split$

For training a random forest we first need to convert into .txt format so that we can read this data in python and pass it to random forest model. Mar2019 data contains 119,029,358 number of points, training random forest on these many points results in my computer and kaggle to crash, so I have considered only a part of this data for train and test. Let the name of our train data be $miniTrain$ as we are only choosing a part of the complete train data. All the lines containing NaN values were removed before passing into RandomForest.

6.2 Adding Features of neighbourhood radius $r = 1$

The following features were considered for training the random forest



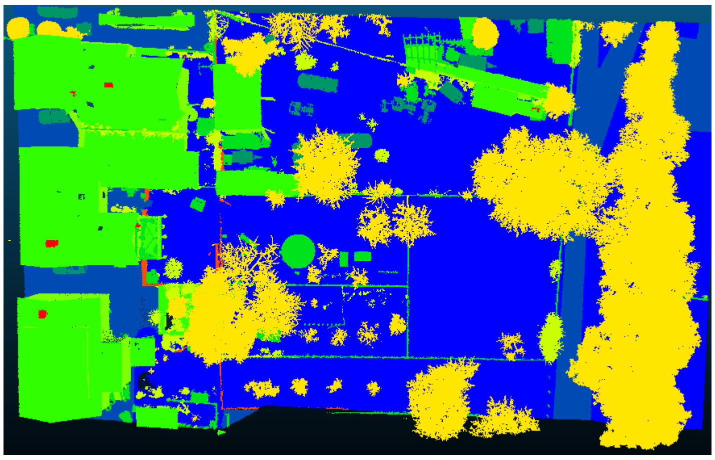
```
feature_order = [
    'X', 'Y', 'Z', 'R', 'G', 'B', 'Sphericity_(0.1)', '1st_eigenvalue_(0.1)',
    'Verticality_(0.1)', 'Number_of_neighbors_(r=0.1)', 'Planarity_(0.1)',
    'Linearity_(0.1)', 'PCA1_(0.1)'
]
```

Metrics	**Results**
Accuracy	0.57471
F1-score	0.54322

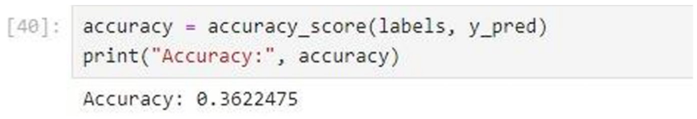
6.3 Adding Features of neighbourhood radius $r = 1$ and $r = 0.125$

6.3.1 Set max threshold for number of points for each class

- Allot only 200000 points per class, if any class has number of points lesser than 200000 retain it.

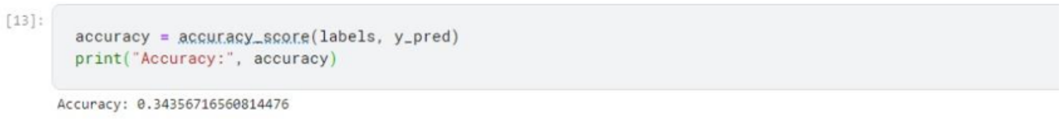


- Allot only 200000 points per class, if any class has number of points lesser than 200000 than duplicates data belonging to that class randomly until the number of points belonging to that class is 200000



```
[40]: accuracy = accuracy_score(labels, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.3622475
```

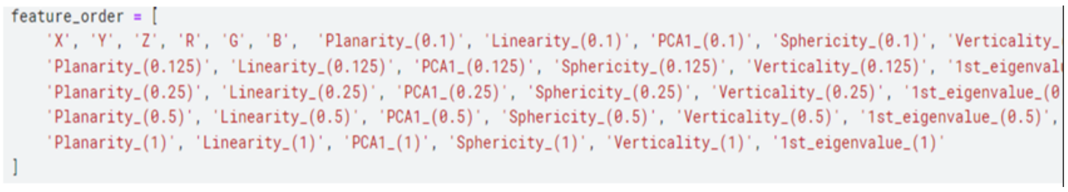
- Another attempt was to just set the threshold as the class having maximum number of points.



```
[13]: accuracy = accuracy_score(labels, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.34356716560814476
```

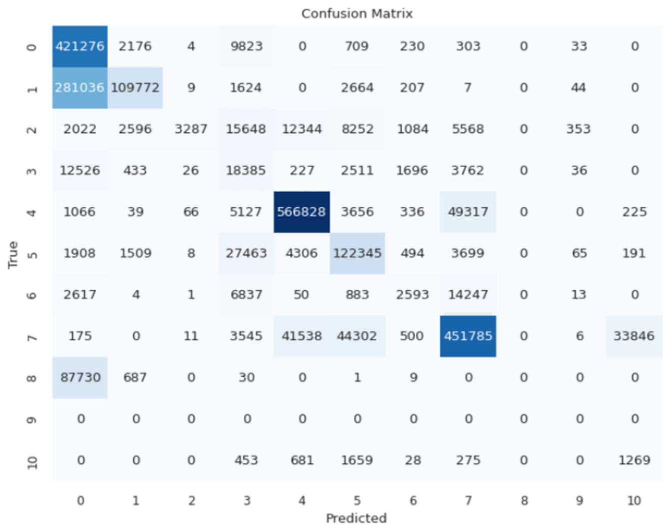
6.4 Multi-Scale neighbourhood extraction

The following features were considered


```
feature_order = [
    'X', 'Y', 'Z', 'R', 'G', 'B',  'Planarity_(0.1)', 'Linearity_(0.1)', 'PCA1_(0.1)', 'Sphericity_(0.1)', 'Verticality_
    'Planarity_(0.125)', 'Linearity_(0.125)', 'PCA1_(0.125)', 'Sphericity_(0.125)', 'Verticality_(0.125)', '1st_eigenval
    'Planarity_(0.25)', 'Linearity_(0.25)', 'PCA1_(0.25)', 'Sphericity_(0.25)', 'Verticality_(0.25)', '1st_eigenvalue_(0
    'Planarity_(0.5)', 'Linearity_(0.5)', 'PCA1_(0.5)', 'Sphericity_(0.5)', 'Verticality_(0.5)', '1st_eigenvalue_(0.5)',
    'Planarity_(1)', 'Linearity_(1)', 'PCA1_(1)', 'Sphericity_(1)', 'Verticality_(1)', '1st_eigenvalue_(1)'
]
```

Results:

Metrics	**Results**
Accuracy	0.705809
F1-score	0.680613



True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	421276	2176	4	9823	0	709	230	303	0	33	0
1	281036	109772	9	1624	0	2664	207	7	0	44	0
2	2022	2596	3287	15648	12344	8252	1084	5568	0	353	0
3	12526	433	26	18385	227	2511	1696	3762	0	36	0
4	1066	39	66	5127	566828	3656	336	49317	0	0	225
5	1908	1509	8	27463	4306	122345	494	3699	0	65	191
6	2617	4	1	6837	50	883	2593	14247	0	13	0
7	175	0	11	3545	41538	44302	500	451785	0	6	33846
8	87730	687	0	30	0	1	9	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	453	681	1659	28	275	0	0	1269

Figure 2: Confusion Matrix

Extending the features for $r = 2$ and $r = 3$

Results:

Metrics	**Results**
Accuracy	0.7223
F1-score	0.7056

6.5 Using XGBoost on miniTrain

Instead of training *miniTrain* with RandomForest, I tried training it with XGBoost with *early stop rounds* **Results:**

Metrics	**Results**
Accuracy	0.72719
F1-score	0.69322

6.6 Using ensemble of random forest models

Since we can't pass the entire data of original train (kaggle crashes), I split the entire train data including features (27 GB) into 5 files and train random forest on each of these data seperately. At the end for predicting classes on *test data* I take the majority vote among them.

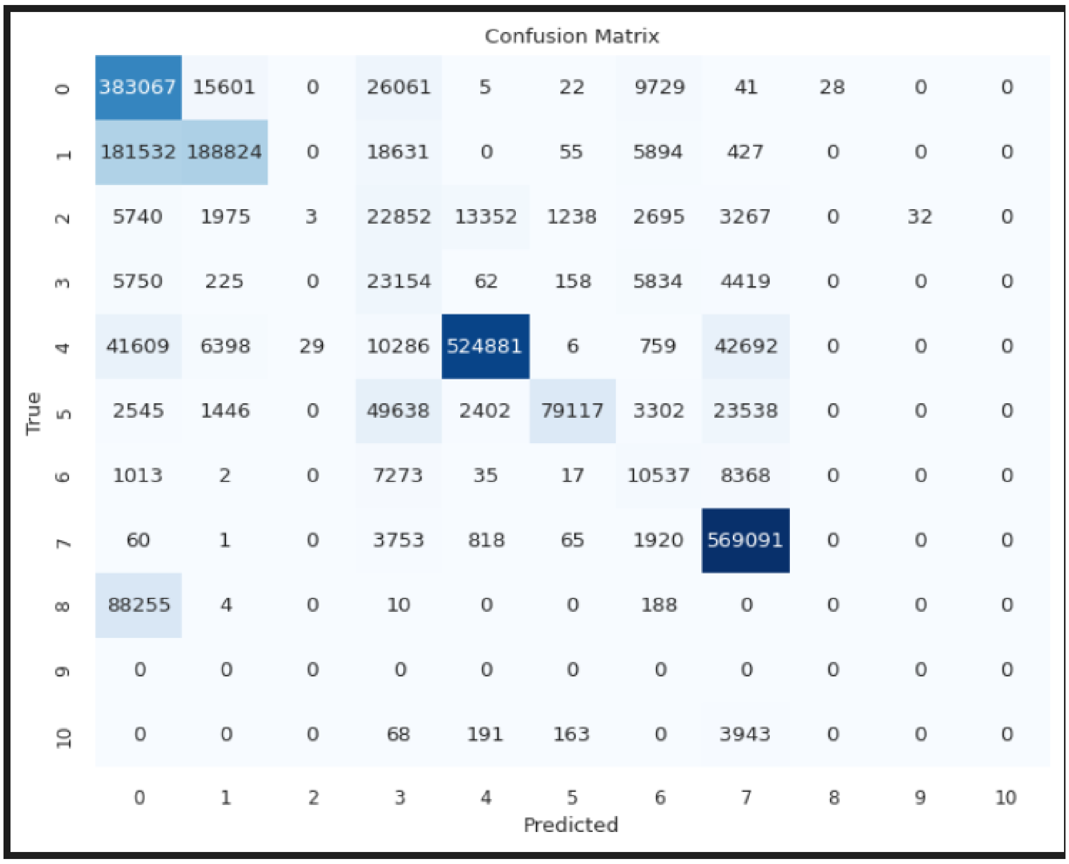
5 models are generated namely model1,model2, model3, model4 and model5.

Results:

Table 1: Model Evaluation

Model	Accuracy	F1 Score
model1	0.1724	0.0916
model2	0.6676	0.6683
model3	0.4681	0.4719
model4	0.7345	0.7080
model5	0.6471	0.6245
ensemble	0.7395	0.7292

Since the model1 had bad accuracy it wasn't involved in majority vote classifier.



True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	383067	15601	0	26061	5	22	9729	41	28	0	0
1	181532	188824	0	18631	0	55	5894	427	0	0	0
2	5740	1975	3	22852	13352	1238	2695	3267	0	32	0
3	5750	225	0	23154	62	158	5834	4419	0	0	0
4	41609	6398	29	10286	524881	6	759	42692	0	0	0
5	2545	1446	0	49638	2402	79117	3302	23538	0	0	0
6	1013	2	0	7273	35	17	10537	8368	0	0	0
7	60	1	0	3753	818	65	1920	569091	0	0	0
8	88255	4	0	10	0	0	188	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	68	191	163	0	3943	0	0	0

Figure 3: Confusion matrix of ensemble model

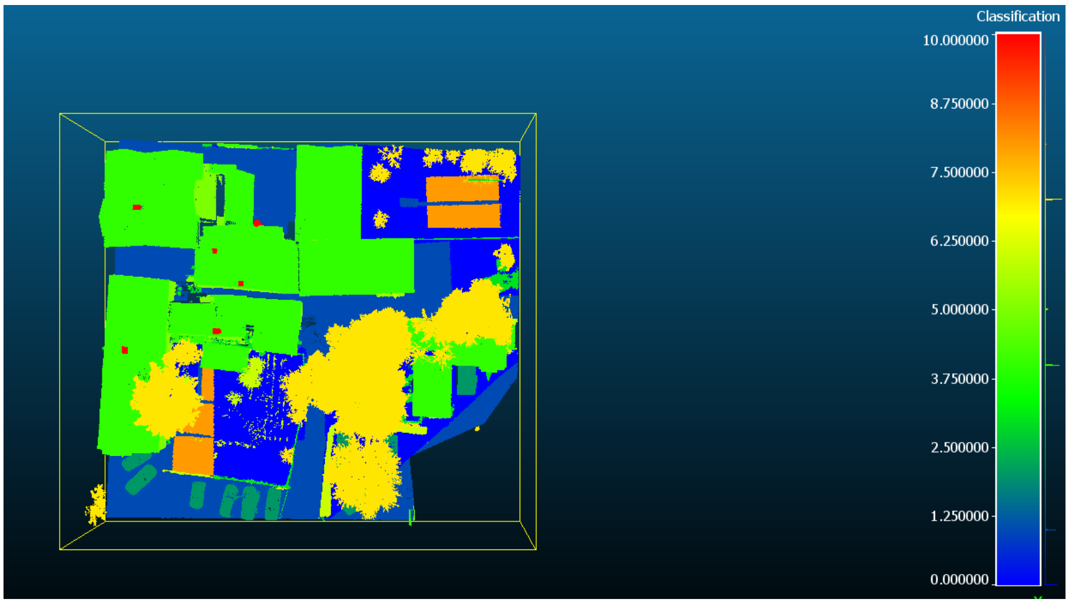


Figure 4: Original Test Data

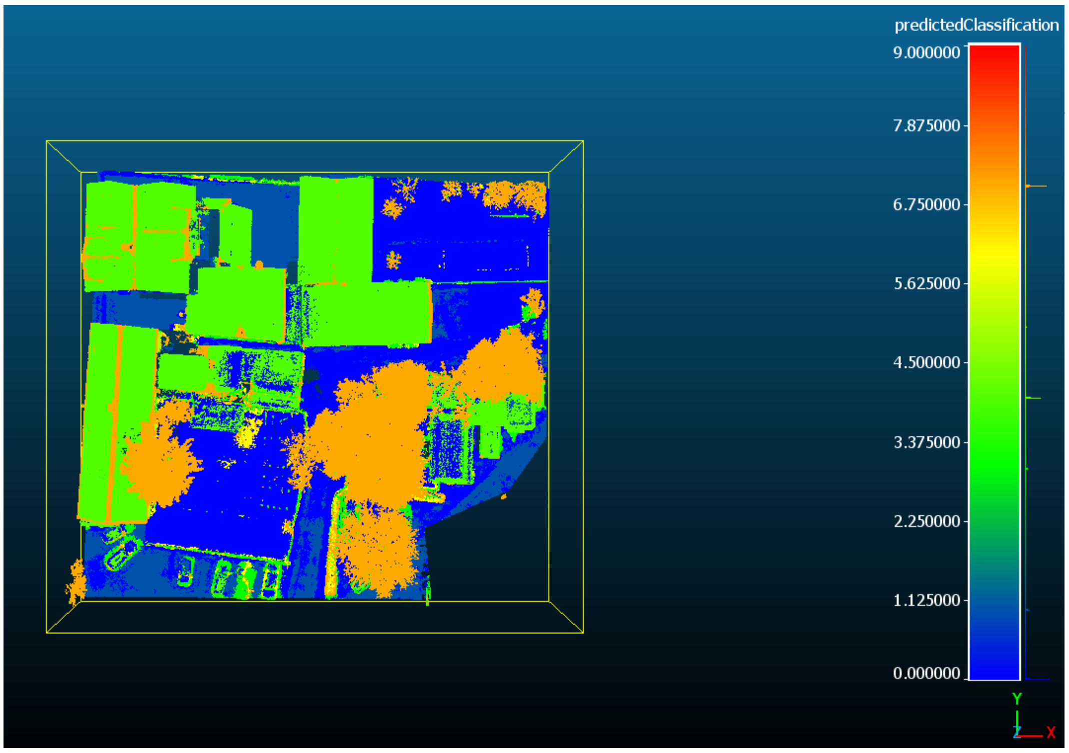


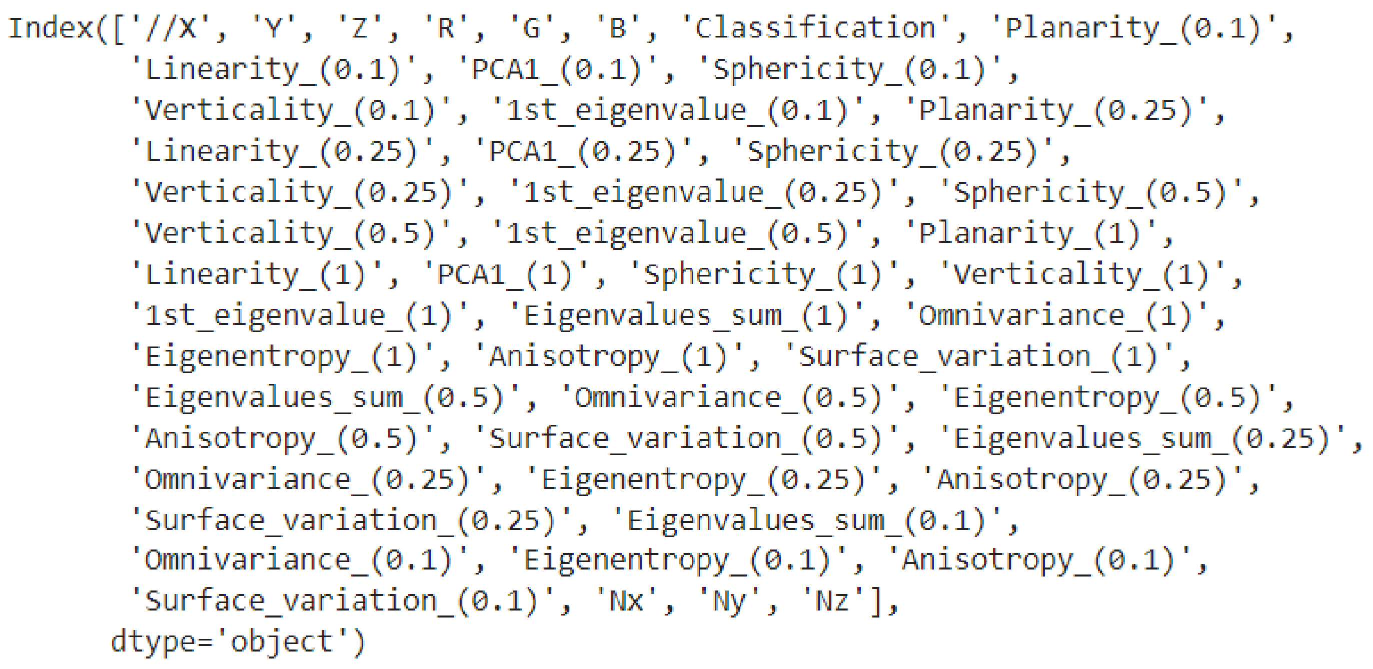
Figure 5: Ensemble model predictions

Overall the ensemble method gave the best accuracy and F1 score among all. The major drawback of this model is that it failes to identify classes 9,10 and 11.

7 Updating Features and Data

7.1 All features considered

In this I proceeded to upload the following features for getting better accuracy.

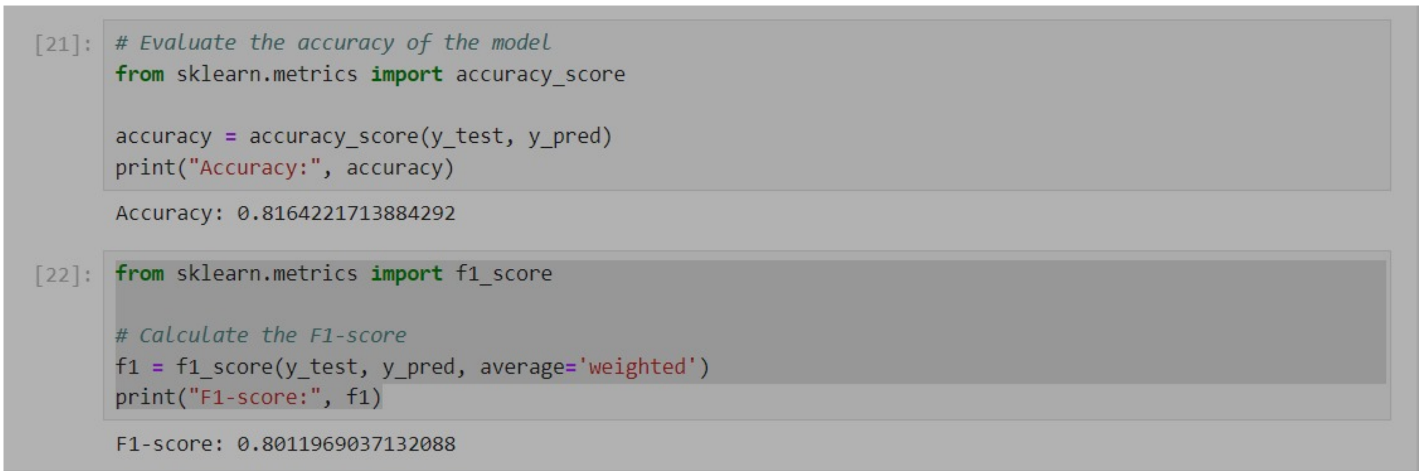


```
Index(['//X', 'Y', 'Z', 'R', 'G', 'B', 'Classification', 'Planarity_(0.1)',
       'Linearity_(0.1)', 'PCA1_(0.1)', 'Sphericity_(0.1)',
       'Verticality_(0.1)', '1st_eigenvalue_(0.1)', 'Planarity_(0.25)',
       'Linearity_(0.25)', 'PCA1_(0.25)', 'Sphericity_(0.25)',
       'Verticality_(0.25)', '1st_eigenvalue_(0.25)', 'Sphericity_(0.5)',
       'Verticality_(0.5)', '1st_eigenvalue_(0.5)', 'Planarity_(1)',
       'Linearity_(1)', 'PCA1_(1)', 'Sphericity_(1)', 'Verticality_(1)',
       '1st_eigenvalue_(1)', 'Eigenvalues_sum_(1)', 'Omnivariance_(1)',
       'Eigenentropy_(1)', 'Anisotropy_(1)', 'Surface_variation_(1)',
       'Eigenvalues_sum_(0.5)', 'Omnivariance_(0.5)', 'Eigenentropy_(0.5)',
       'Anisotropy_(0.5)', 'Surface_variation_(0.5)', 'Eigenvalues_sum_(0.25)',
       'Omnivariance_(0.25)', 'Eigenentropy_(0.25)', 'Anisotropy_(0.25)',
       'Surface_variation_(0.25)', 'Eigenvalues_sum_(0.1)',
       'Omnivariance_(0.1)', 'Eigenentropy_(0.1)', 'Anisotropy_(0.1)',
       'Surface_variation_(0.1)', 'Nx', 'Ny', 'Nz'],
      dtype='object')
```

Figure 6: Features

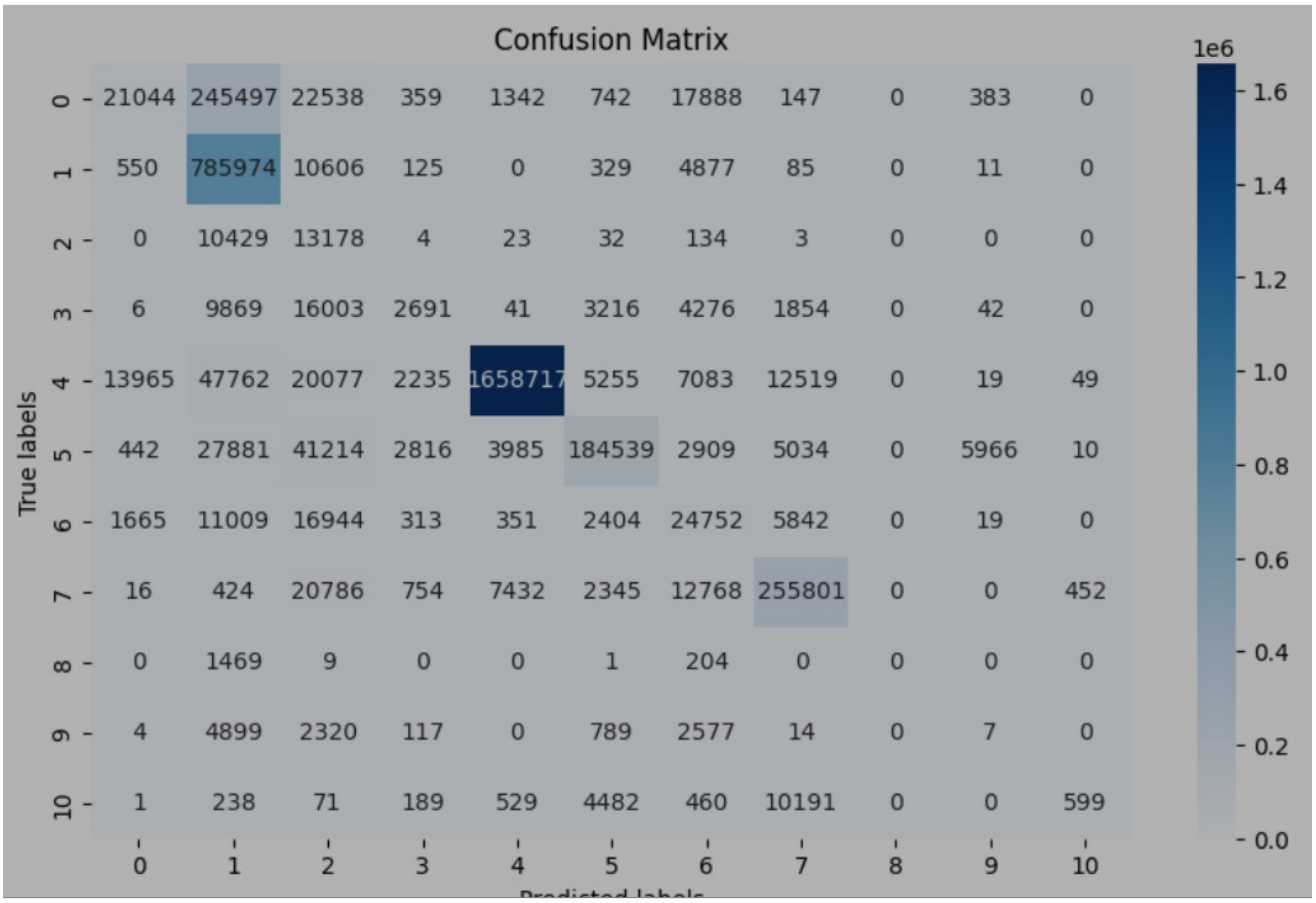
The scale considered were r = 0.1, 0.25, 0.5 and 1.

Results:



```
[21]: # Evaluate the accuracy of the model
      from sklearn.metrics import accuracy_score

      accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.8164221713884292

[22]: from sklearn.metrics import f1_score

      # Calculate the F1-score
      f1 = f1_score(y_test, y_pred, average='weighted')
      print("F1-score:", f1)

      F1-score: 0.8011969037132088
```

Figure 7

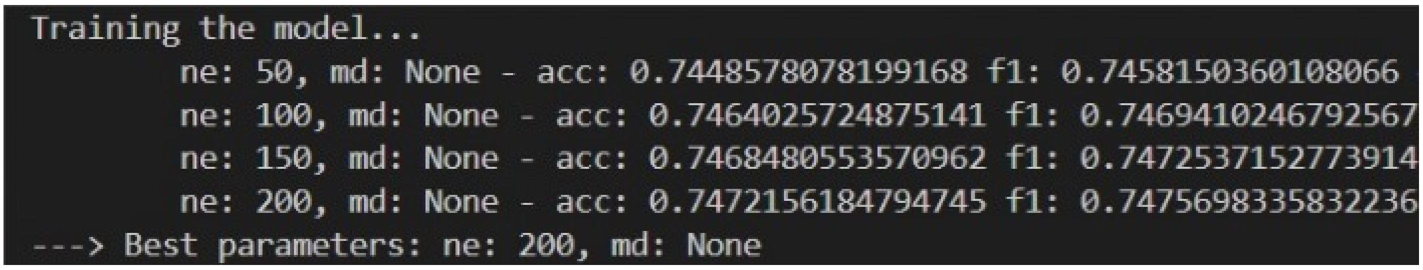


	0	1	2	3	4	5	6	7	8	9	10
0	21044	245497	22538	359	1342	742	17888	147	0	383	0
1	550	785974	10606	125	0	329	4877	85	0	11	0
2	0	10429	13178	4	23	32	134	3	0	0	0
3	6	9869	16003	2691	41	3216	4276	1854	0	42	0
4	13965	47762	20077	2235	1658717	5255	7083	12519	0	19	49
5	442	27881	41214	2816	3985	184539	2909	5034	0	5966	10
6	1665	11009	16944	313	351	2404	24752	5842	0	19	0
7	16	424	20786	754	7432	2345	12768	255801	0	0	452
8	0	1469	9	0	0	1	204	0	0	0	0
9	4	4899	2320	117	0	789	2577	14	0	7	0
10	1	238	71	189	529	4482	460	10191	0	0	599

Figure 8: Confusion Matrix

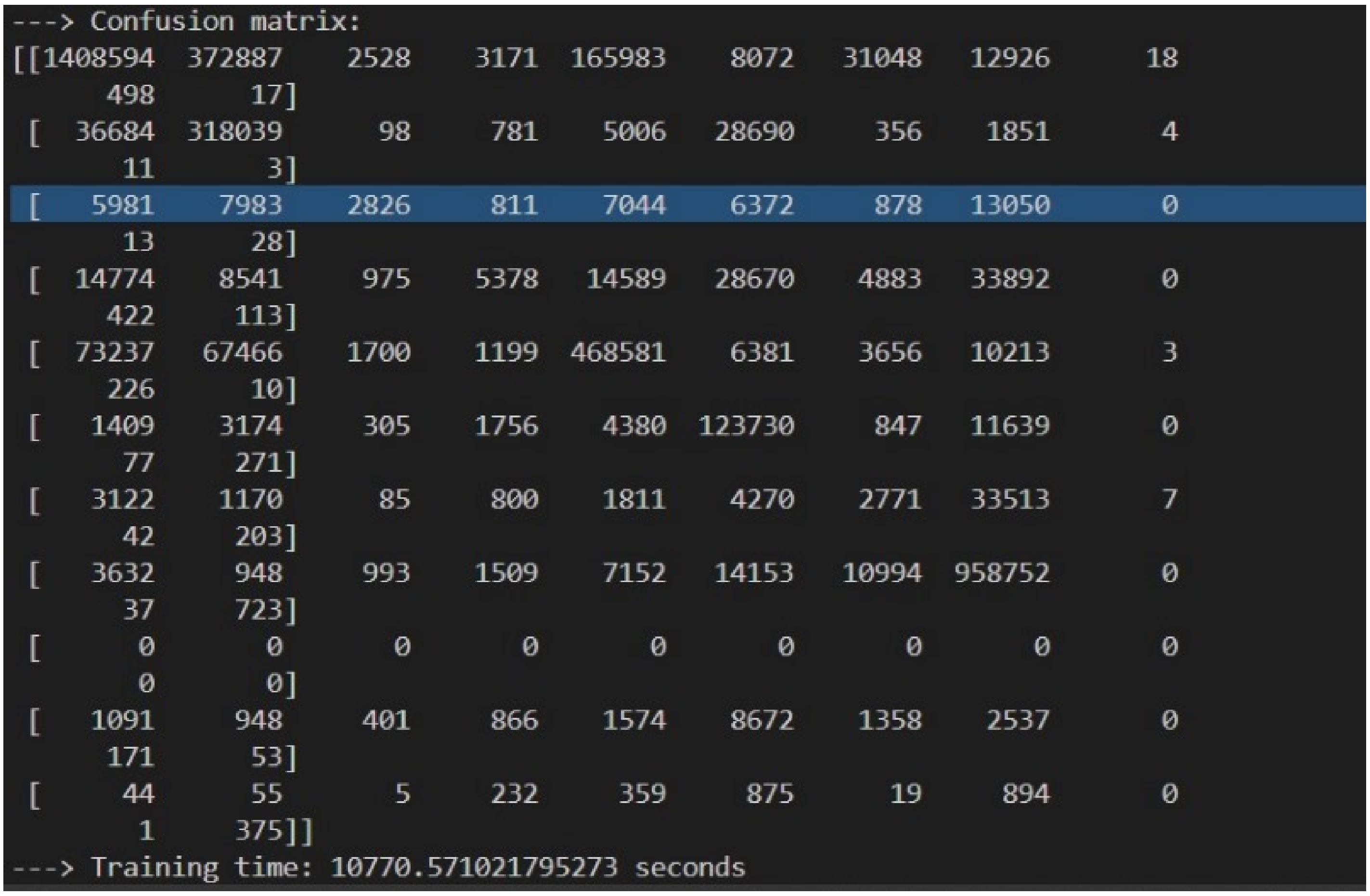
7.2 Random Forest Hyper-Parameter Tuning

Performing Hyper parameter tuning on n-estimators parameter of Random Forest.



```
 Training the model...
        ne: 50, md: None - acc: 0.7448578078199168 f1: 0.7458150360108066
        ne: 100, md: None - acc: 0.7464025724875141 f1: 0.7469410246792567
        ne: 150, md: None - acc: 0.7468480553570962 f1: 0.7472537152773914
        ne: 200, md: None - acc: 0.7472156184794745 f1: 0.7475698335832236
 ---> Best parameters: ne: 200, md: None
```

Figure 9: Hyper-Parameter Tuning

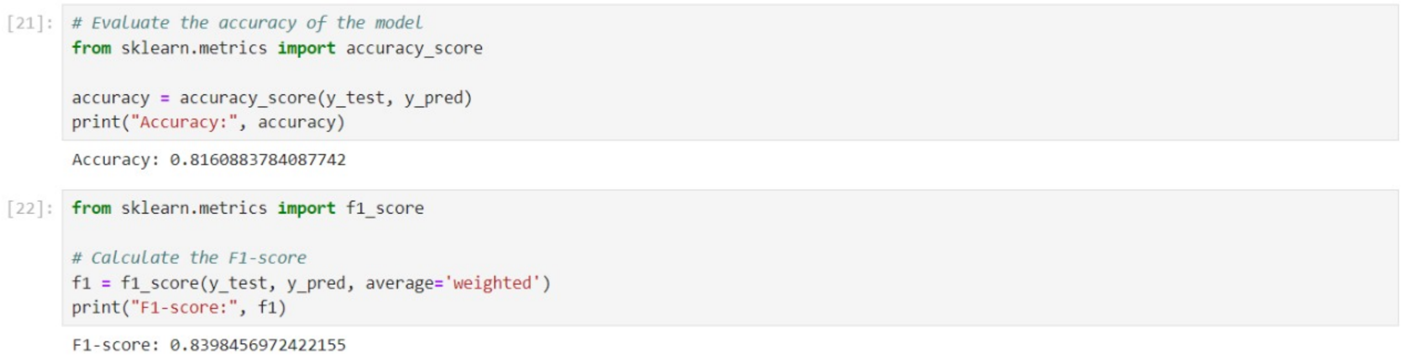


```
---> Confusion matrix:
[[1408594  372887    2528    3171  165983    8072   31048   12926      18
      498      17]
 [  36684  318039      98     781    5006   28690     356    1851       4
       11       3]
 [   5981    7983    2826     811    7044    6372     878   13050       0
       13      28]
 [  14774    8541     975    5378   14589   28670    4883   33892       0
      422     113]
 [  73237   67466    1700    1199  468581    6381    3656   10213       3
      226      10]
 [   1409    3174     305    1756    4380  123730     847   11639       0
       77     271]
 [   3122    1170      85     800    1811    4270    2771   33513       7
       42     203]
 [   3632     948     993    1509    7152   14153   10994  958752       0
       37     723]
 [      0       0       0       0       0       0       0       0       0
        0       0]
 [   1091     948     401     866    1574    8672    1358    2537       0
      171      53]
 [     44      55       5     232     359     875      19     894       0
        1     375]]
---> Training time: 10770.571021795273 seconds
```

Figure 10: Confusion Matrix of best n-estimator model

7.3 Removing X,Y from features

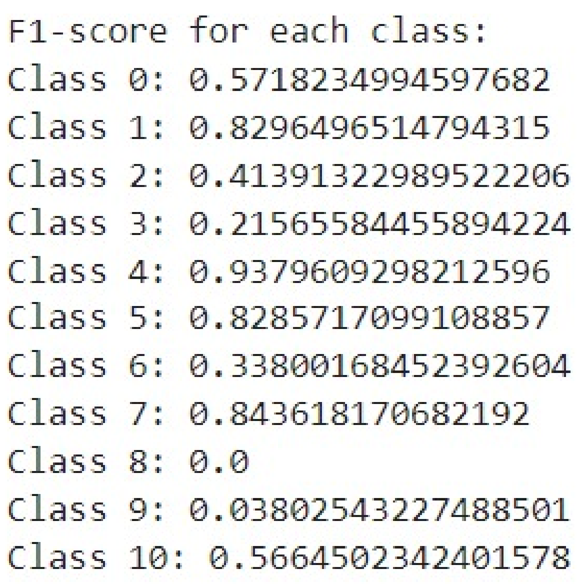
Now we remove X,Y from features for training, as this might mislead out data.



```
[21]: # Evaluate the accuracy of the model
      from sklearn.metrics import accuracy_score

      accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.8160883784087742

[22]: from sklearn.metrics import f1_score

      # Calculate the F1-score
      f1 = f1_score(y_test, y_pred, average='weighted')
      print("F1-score:", f1)

      F1-score: 0.8398456972422155
```

Figure 11: Removing X,Y from features

The corresponding f1-score per class for above is as follows



```
F1-score for each class:
Class 0: 0.5718234994597682
Class 1: 0.8296496514794315
Class 2: 0.4139132298952206
Class 3: 0.21565584455894224
Class 4: 0.9379609298212596
Class 5: 0.8285717099108857
Class 6: 0.33800168452392604
Class 7: 0.843618170682192
Class 8: 0.0
Class 9: 0.03802543227488501
Class 10: 0.5664502342401578
```

Figure 12: f1-score per class

7.4 Thresholding features based on correlation values

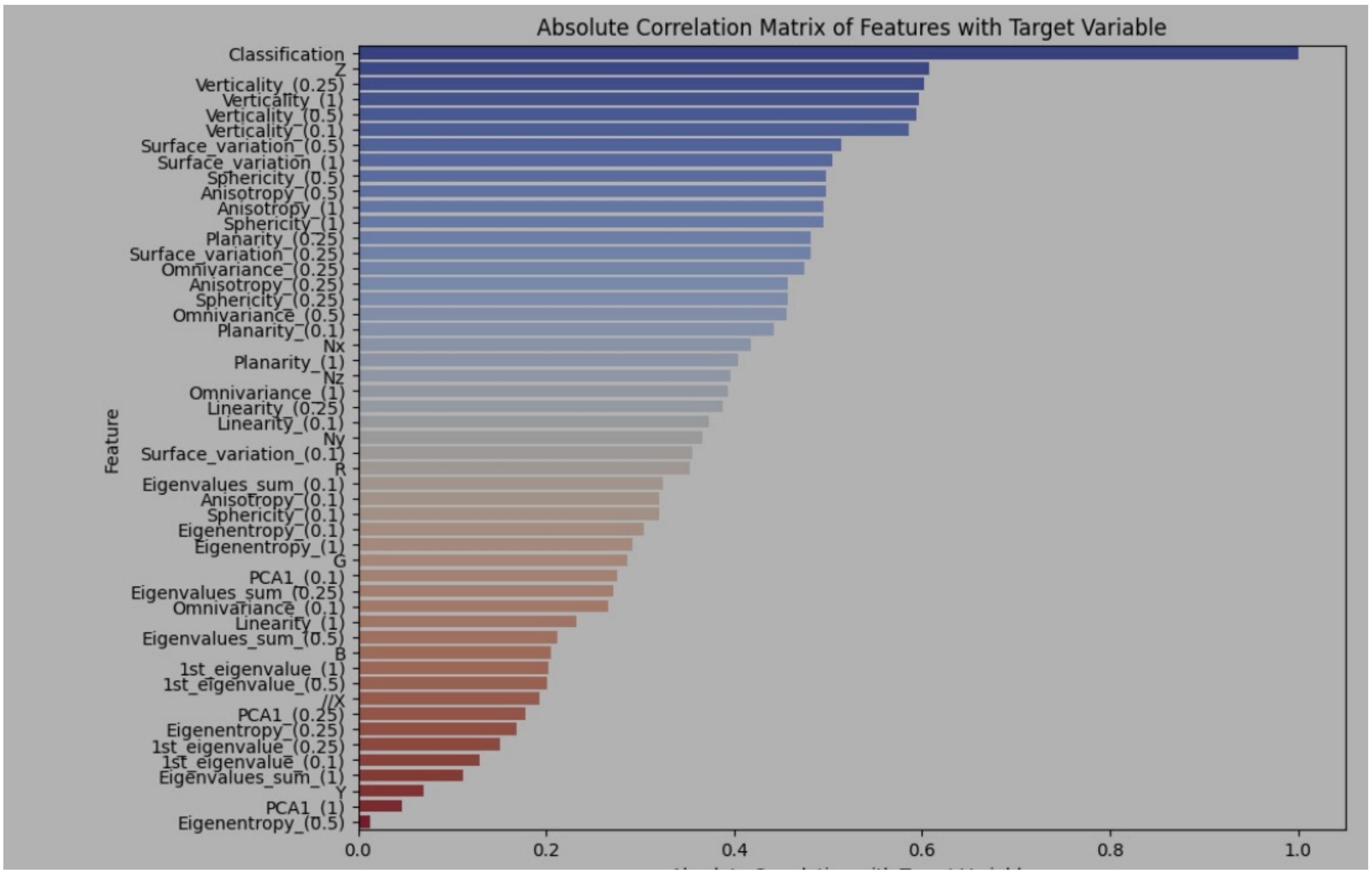
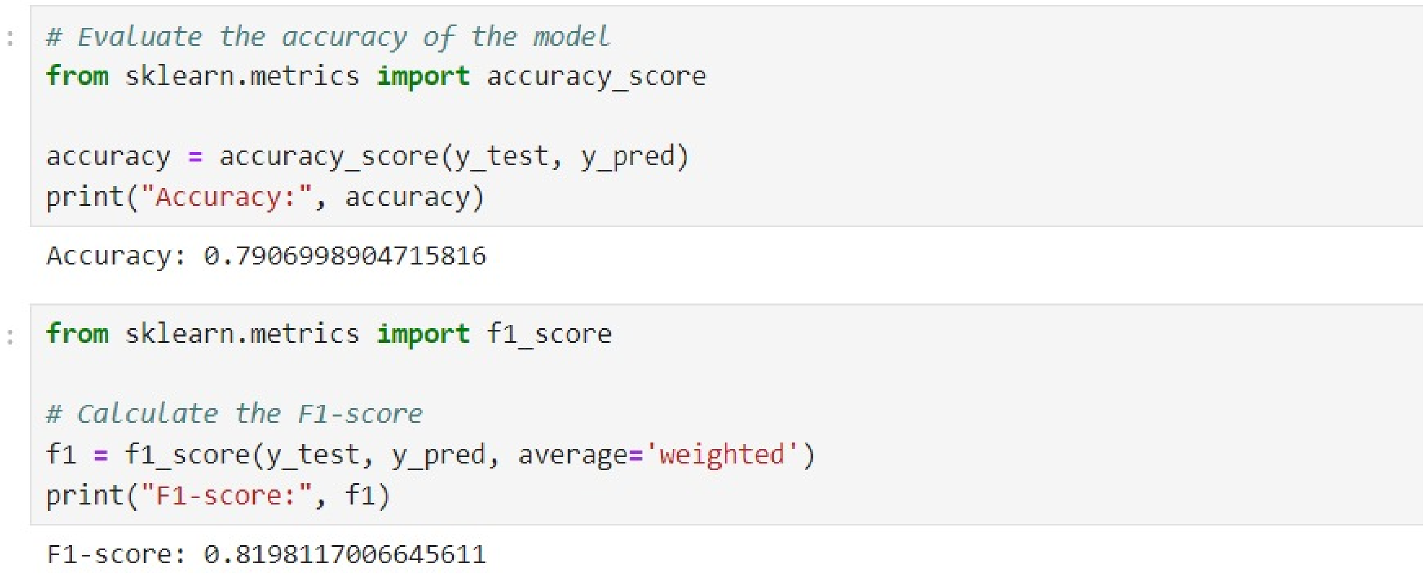


Figure 13: Correlation of features with Classification



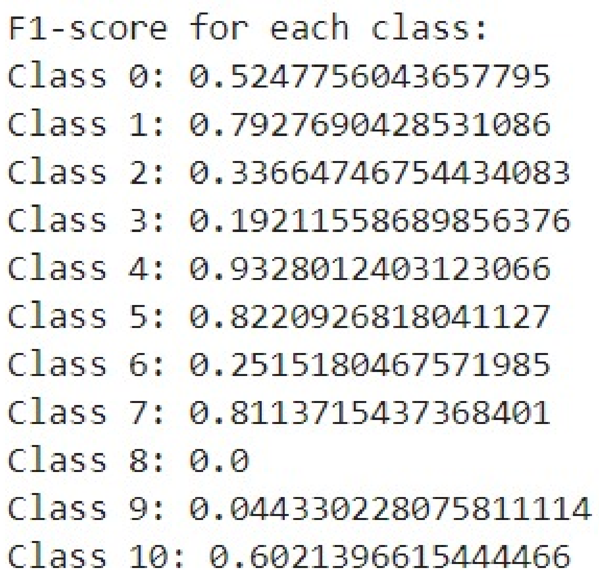
```
# Evaluate the accuracy of the model
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7906998904715816
```

```
from sklearn.metrics import f1_score

# Calculate the F1-score
f1 = f1_score(y_test, y_pred, average='weighted')
print("F1-score:", f1)
```

```
F1-score: 0.8198117006645611
```

Figure 14: Results



```
F1-score for each class:
Class 0: 0.5247756043657795
Class 1: 0.7927690428531086
Class 2: 0.33664746754434083
Class 3: 0.19211558689856376
Class 4: 0.9328012403123066
Class 5: 0.8220926818041127
Class 6: 0.2515180467571985
Class 7: 0.8113715437368401
Class 8: 0.0
Class 9: 0.044330228075811114
Class 10: 0.6021396615444466
```

Figure 15: f1-score per class

Overall we can see that the model using no X,Y gives the best accuracy, f1-score.

7.5 train and test images

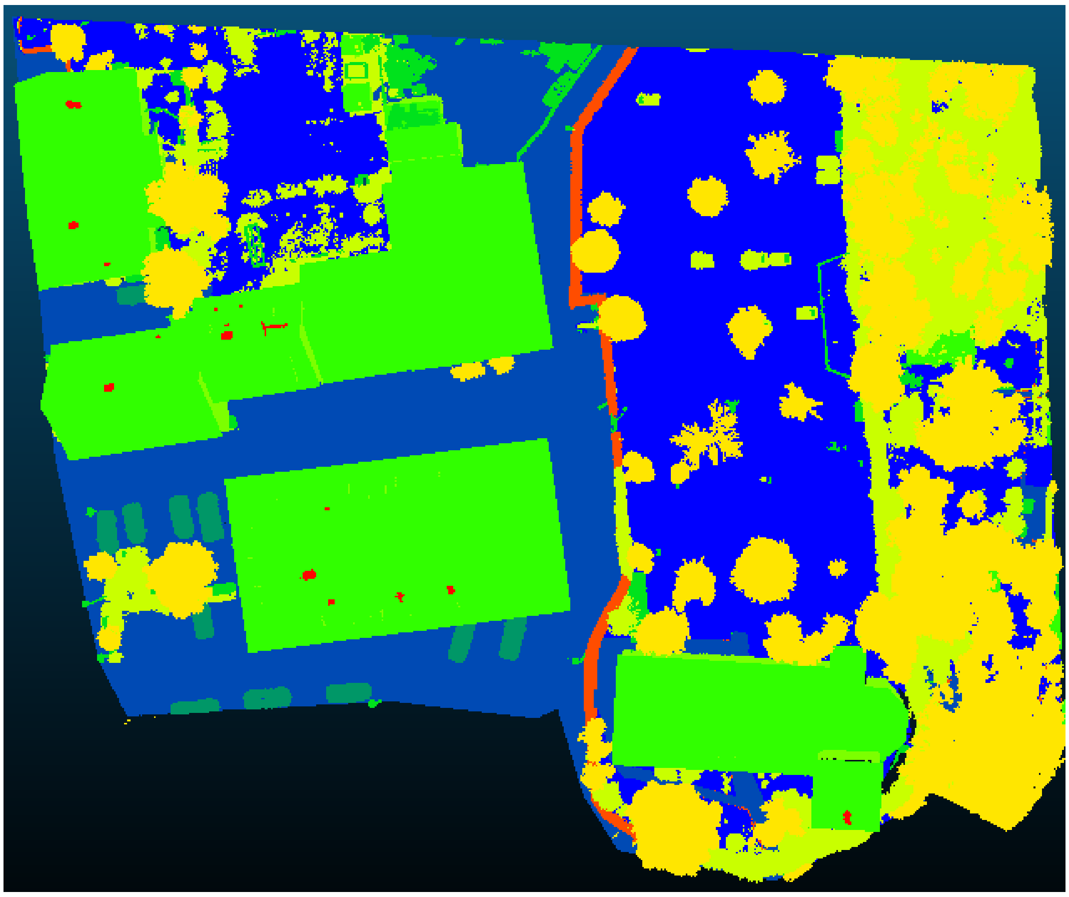


Figure 16: train-1

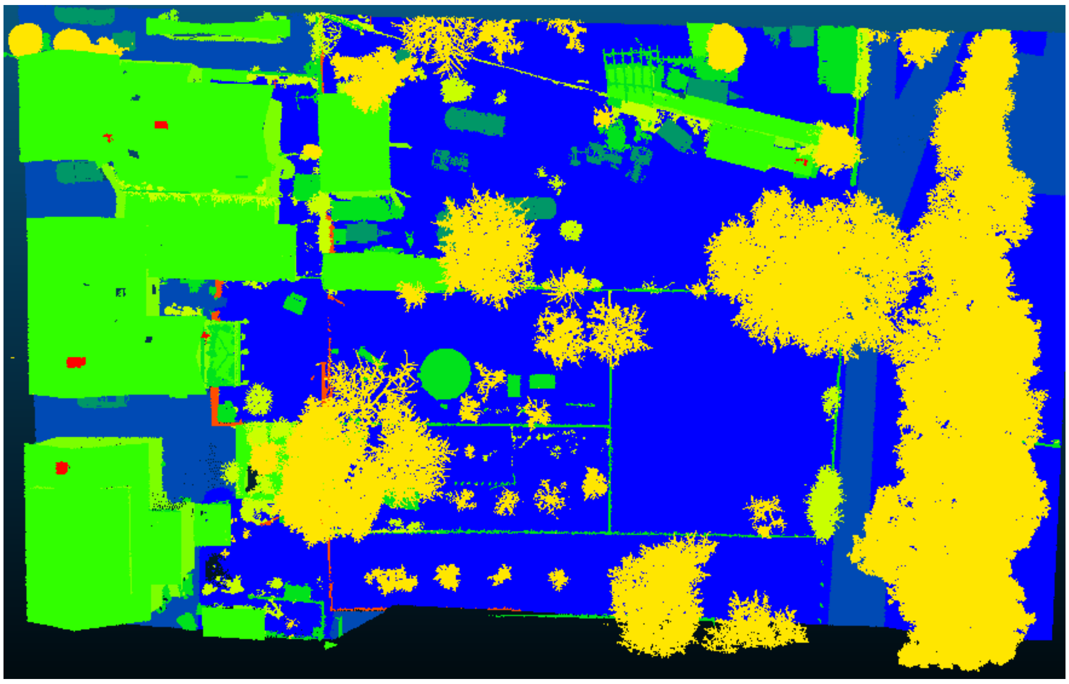


Figure 17: train-2

The above images are $train - 1$ and $train - 2$ which are concatenated for training data.

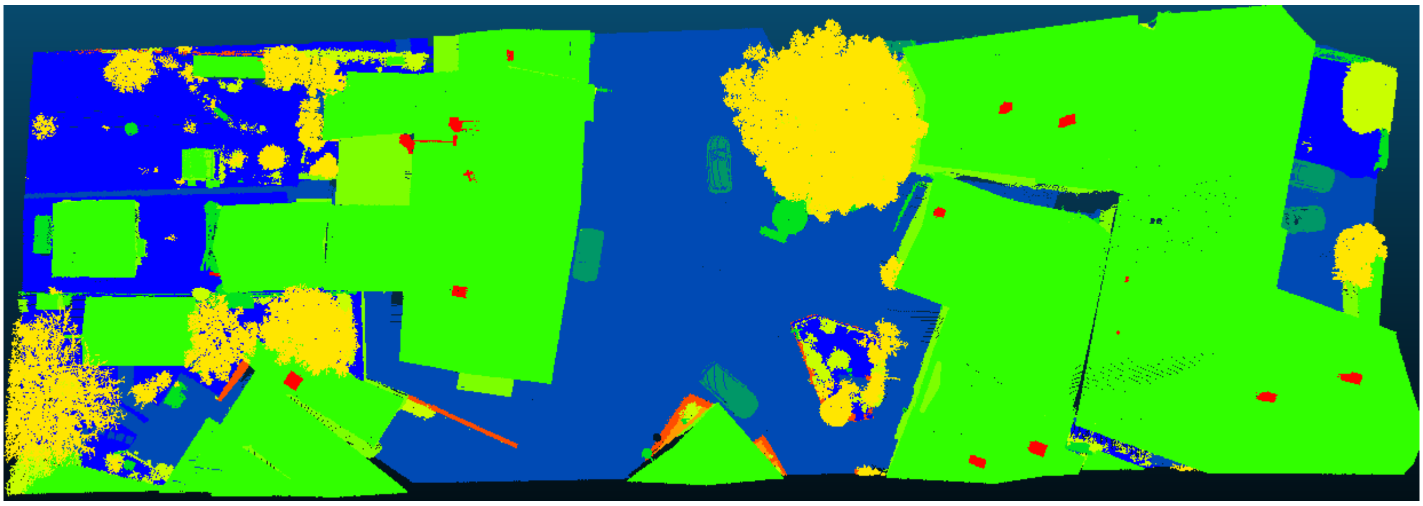


Figure 18: test-ground truth

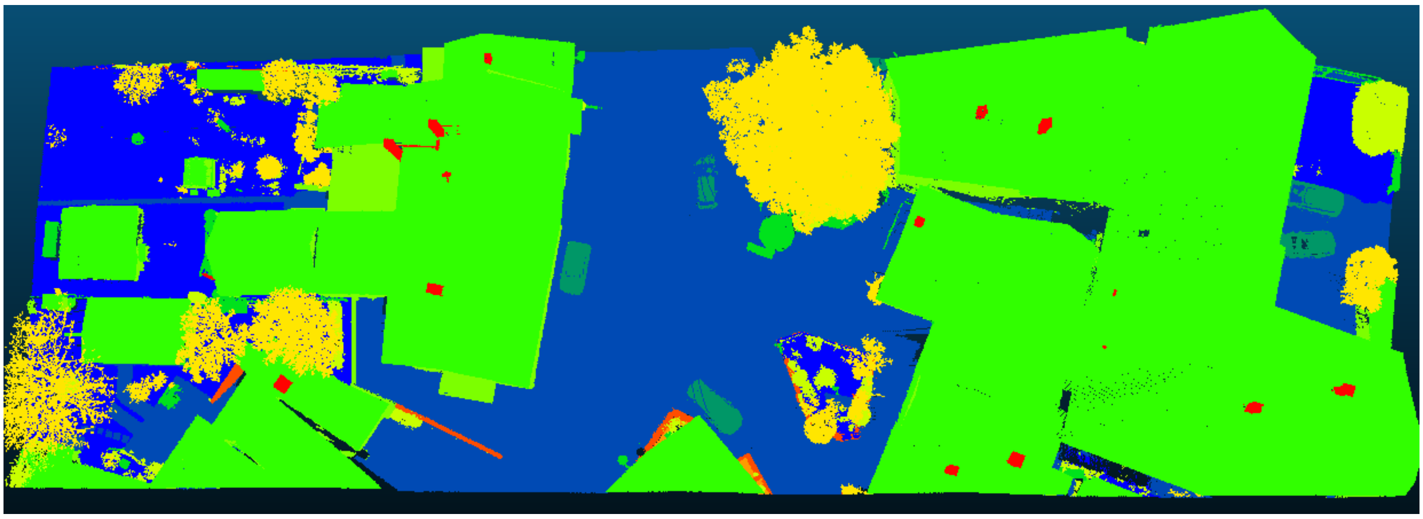


Figure 19: test-predicted value

In the above 2 images you can see that the points are classified almost similarly, but the thing we notice is that the classification of prediction values are lesser dense than the grount truth values. For better understanding look at the bottom left of both images and focus on the tree (yellow). You find that the point become less denser.

If you look at the car in the center of figure beside the tree we notice that in predicted value the car isn't completely classified as car, some points are misclassified as ground points.

We observe the following from confusion matrix:

- Low Vegetation and Impervious Surface are highly misclassified with each other, the reasoning behind it can be that the height of these classes are highly similar.
- Roof class is highly classified correctly.
- Facade class has good amount of accuracy in classification.
- Tree class has a great amount of accuracy in classification.
- Many of the vehicle points are misclassified with ground class.

Overall, I have achieved *accuracy* of 81.608% and f1-score of 83.984% when considering 44 features to train (X, Y not considered for training).