About

This notebook is for the Assignment 4 of CIE4604 at TU Delft

Can also be found here

Team members:

Name	Student Number
Maxwell Lindsay	5243610
Pratyush Kumar	5359252

Contents

In [ ]:

```
import laspy
from scipy.spatial import cKDTree
import numpy as np
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import proj3d
from sklearn.preprocessing import MinMaxScaler


import geopandas as gpd
from shapely.geometry import Polygon, Point
import pandas as pd
pd.set_option('display.max_columns', 100)

from sklearn.model_selection import train_test_split, cross_val_score

# preprocessing and scoring
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
import pickle
```

In [ ]:

```
def get_spatial_subset(
    full_cloud, xmin: int, xmax: int, ymin: int, ymax: int
) -> laspy.lasdata.LasData:
    """
    Create a subset of a LAS file based on a bounding box in coordinates of the clou

    parameters:
    full_cloud: a laspy cloud object you want to subset from

    xmin: the values of the bounding box object
    xmax
    ymin
    ymax
```

```
    returns: laspy.lasdata.LasData object
    """

    # create empty laspy collection to put the filtered points in that has the same
    new_file = laspy.create(
        point_format=cloud.header.point_format, file_version=cloud.header.version,
    )

    # create matrices of boolean values
    a = cloud.x > xmin
    b = cloud.x < xmax
    c = cloud.y > ymin
    d = cloud.y < ymax
    # subset the points and put them into the new laspy file
    new_file.points = full_cloud.points[a & b & c & d]

    new_file.header.offsets = full_cloud.header.offsets
    new_file.header.scales = full_cloud.header.scales

    return new_file
```

In [ ]:

```
# load in the fully LAZ file
cloud = laspy.read(r"./data/AHN4_Noordwijk.laz")
```

In [ ]:

```
# define the area of interest
xcenter = 89401.42414516   #89400
bbbox_size_length= 170           #300
bbbox_size_width = 100           #300
ycenter = 472877.83174913  #472868
xmin = xcenter - 0.5 * bbbox_size_length
xmax = xcenter + 0.5 * bbbox_size_length
ymin = ycenter - 0.5 * bbbox_size_width
ymax = ycenter + 0.5 * bbbox_size_width

# create the subset
subset = get_spatial_subset(cloud, xmin, xmax, ymin, ymax)


# # saving the bounds as a shapefile

lat_point_list = [ymin, ymin, ymax, ymax, ymin]
lon_point_list = [xmin, xmax ,xmax, xmin, xmin]

polygon_geom = Polygon(zip(lon_point_list, lat_point_list))
crs_28992 = {'init':'epsg:28992'}
polygon = gpd.GeoDataFrame(index=[0], crs=crs_28992, geometry=[polygon_geom])
print(polygon.geometry)

# polygon.to_file(filename='./data/bounds_new.geojson', driver='GeoJSON') # geojson
polygon.to_file(filename='./data/bounds_new.shp', driver="ESRI Shapefile")
```

In [ ]:

```
# create a tree for all points in the subset
dataset = np.vstack((subset.X, subset.Y, subset.Z)).transpose()
tree = cKDTree(dataset)
```

In [ ]:

```
# setting up the extra dimensions to add back to the las file
dimension_list = [
    laspy.point.format.ExtraBytesParams("normal_angle", "float64"),
    laspy.point.format.ExtraBytesParams("linearity15", "float64"),
```

```
    laspy.point.format.ExtraBytesParams("planarity15", "float64"),
    laspy.point.format.ExtraBytesParams("scat15", "float64"),
    laspy.point.format.ExtraBytesParams("curve15", "float64"),
]

for dim in dimension_list:
    subset.add_extra_dim(dim)

# check that it was added correctly
list(subset.point_format.extra_dimension_names)
```

Out[ ]:
```
['Deviation',
 'Reflectance',
 'Amplitude',
 'normal_angle',
 'linearity15',
 'planarity15',
 'scat15',
 'curve15']
```

In [ ]:
```
# get an array for all the points
all_points = subset.points.array
```

In [ ]:
```
def _fit_plane_pca(point, tree, k) -> PCA:
    """
    internal function designed to fit a PCA model to the k nearest points to a point
    parameters:
    point: single point as structured ndarray of point data returned from laspy.poin
    tree: the KD tree for the points
    k: number of neighbors to consider

    returns:
    fitted pca model: sklean.decom
    """
    distances, neighbors_indices = tree.query((point[0], point[1], point[2]), k)

    neighbors_points = tree.data[neighbors_indices]
    # set up pca model
    pca = PCA(n_components=3)
    # fit the model to the points
    pca.fit(neighbors_points)
    return pca


def _find_angle(vector, axis="z"):
    axis_lookup = {"z": [0, 0, 1], "y": [0, 1, 0], "x": [1, 0, 0]}
    unit_vector = axis_lookup[axis]
    # find the angle with the z axis using the dot product
    angle = np.rad2deg(np.arccos(vector.dot(unit_vector)))

    # we want all the normals pointing in the same direction

    if angle > 90:
        # if the vector is pointing the wrong way, flip it
        vector = vector * -1
        # find the angle of the new vector
        angle = np.rad2deg(np.arccos(vector.dot(unit_vector)))

    return angle
```

In [ ]:
```
def pca_3d_shape(point_array: np.array, tree: cKDTree, k: int = 8) -> np.array:
```

```
    """
    takes an array of points from laspy, a kd tree, and an integer number of neighbo


    returns an array of:

    linearity, planarity, scattering, omnivariance, anisotropy, eigenentropy, eigsum

    """
    length_parray = len(point_array)
    normal_angle = np.zeros(length_parray)
    normal_vectors = np.zeros((length_parray, 3))
    # set up empty arrays
    linearity = np.zeros(length_parray)
    planarity = np.zeros(length_parray)
    scattering = np.zeros(length_parray)
    omnivariance = np.zeros(length_parray)
    anisotropy = np.zeros(length_parray)
    eigenentropy = np.zeros(length_parray)
    eigsum = np.zeros(length_parray)
    ch_cur = np.zeros(length_parray)
    norm_ang_x = np.zeros(length_parray)
    norm_ang_y = np.zeros(length_parray)

    for i, point in enumerate(point_array):

        # get a fitted pca model
        pca = _fit_plane_pca(point, tree, k)

        # get the Z value for the normal
        normal_vector = pca.components_.T[2]

        # find the angle with the z axis using the dot product
        angle = _find_angle(normal_vector, "z")
        angle_y = _find_angle(normal_vector, "y")
        angle_x = _find_angle(normal_vector, "x")

        # save the angle in an array
        normal_angle[i] = angle
        norm_ang_x[i] = angle_x
        norm_ang_y[i] = angle_y
        # save the vector in an array
        normal_vectors[i] = normal_vector

        # Find the eigenvalues of the matrix
        eival1, eival2, eival3 = pca.singular_values_

        # calculate the following values and assign them to the correct spot in the
        linearity[i] = (eival1 - eival2) / eival1
        planarity[i] = (eival2 - eival3) / eival1
        scattering[i] = eival3 / eival1
        omnivariance[i] = (eival1 * eival2 * eival3) ** (1 / 3)
        anisotropy[i] = (eival1 - eival3) / eival1
        eigenentropy[i] = -1 * (
            eival1 * np.log(eival1) + eival2 * np.log(eival2) + eival3 * np.log(eiva
        )
        eigsum[i] = eival1 + eival2 + eival3
        ch_cur[i] = eival3 / (eival1 + eival2 + eival3)

    pca_geo_array = np.vstack(
        [
            normal_angle,
            norm_ang_x,
            norm_ang_y,
```

```
            linearity,
            planarity,
            scattering,
            omnivariance,
            anisotropy,
            eigenentropy,
            eigsum,
            ch_cur,
        ]
    ).T

    return pca_geo_array
```

In [ ]:
```
# run the geoetric featuers
# NOTE this cell is *slow* (~4minutes?)
pca_results8 = pca_3d_shape(all_points, tree, k=8)
pca_results15 = pca_3d_shape(all_points, tree, k=15)
```

In [ ]:
```
# assign the calculated angle to the new dimension created above
subset.normal_angle = pca_results8[:, 0]
subset.linearity15 = pca_results15[:, 3]
subset.planarity15 = pca_results15[:, 4]
subset.scat15 = pca_results15[:, 5]
subset.curve15 = pca_results15[:, 10]

# save the file
subset.write("./data/subsetted_pk_code_20dec.las")
```

Loading subset from las file

subset already has normals and other computations

In [ ]:
```
subset = laspy.read(r"./data/subsetted_pk_code_20dec.las")
# subset = laspy.read(r"./data/fme_classifir.las")
```

In [ ]:
```
df_data_dict={}
for i in subset.point_format.dimension_names:
    if type(subset[i])==laspy.point.dims.SubFieldView:
        df_data_dict[i] = np.array(subset[i])
    else:
        df_data_dict[i] = subset[i]


las_df = pd.DataFrame(data=df_data_dict)

las_df.loc[:,'x'] = np.array(subset.x, dtype=np.float64)
las_df.loc[:,'y'] = np.array(subset.y, dtype=np.float64)
las_df.loc[:,'z'] = np.array(subset.z, dtype=np.float64)

# las_df.loc[:,'XYZ_point'] = [Point(xyz) for xyz in list(zip( las_df.X, las_df.Y, l
las_df.loc[:,'xy_point']  = [Point(xyz) for xyz in list(zip( las_df.x, las_df.y))]

las_df.head()
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	synthetic	key_point	withh

	X	Y	Z	intensity	return_number	number_of_returns	synthetic	key_point	withh
0	336463	400324	10399	1098	1	1	0	0	
1	336575	400651	10389	1141	1	1	0	0	
2	336432	399149	10422	1058	1	1	0	0	
3	336544	399481	10422	1046	1	1	0	0	
4	336648	399791	10426	1171	1	1	0	0	

Loading the labelled shapefile

labels to be attached to the points by spatial join
Labels are: | Class | Value | |----------|-------|| Trees | 1 || Grass | 2 || Buildings | 3 || Sand | 4 || Road | 5 |

In [ ]:
```
classDF = gpd.read_file('./data/labels_classes1.shp', epsg=28992)
```

In [ ]:
```
gdf = gpd.GeoDataFrame(las_df, crs="EPSG:28992", geometry=las_df.xy_point)
gdf.head()
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	synthetic	key_point	withh
0	336463	400324	10399	1098	1	1	0	0	
1	336575	400651	10389	1141	1	1	0	0	
2	336432	399149	10422	1058	1	1	0	0	
3	336544	399481	10422	1046	1	1	0	0	
4	336648	399791	10426	1171	1	1	0	0	

In [ ]:
```
joined = gdf.sjoin(classDF,how='inner')
joined.head()
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	synthetic	key_point	wit

	X	Y	Z	intensity	return_number	number_of_returns	synthetic	key_point	wit
157	336462	390478	10471	1113	1	1	0	0	
193	336498	389197	10432	1119	1	1	0	0	
194	336613	389523	10397	1100	1	1	0	0	
195	336706	389805	10413	1082	1	1	0	0	
196	336818	390143	10437	1156	1	1	0	0	

In [ ]:
```
# print( np.unique(joined.label) )
joined.to_pickle('./data/classfied_df.pkl')
gdf.to_pickle('./data/unclassfied_raw_df.pkl')
```

Random Forest

test train split

In [ ]:
```
# Load from pkl file, the classified df
labelled_df = pd.read_pickle('./data/classfied_df.pkl')
labelled_df.drop(['x','y','z','xy_point', 'index_right',  'edge_of_flight_line', 'cl
labelled_df.rename(columns={"label_right": "class"}, inplace=True)
labelled_df = gpd.GeoDataFrame(labelled_df, crs="EPSG:28992", geometry=labelled_df.g

labelled_df.head()
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	scan_angle	red	greer
157	336462	390478	10471	1113	1	1	-4333	59392	5580
193	336498	389197	10432	1119	1	1	-4333	31744	2739
194	336613	389523	10397	1100	1	1	-4333	47872	4428
195	336706	389805	10413	1082	1	1	-4333	58368	5478
196	336818	390143	10437	1156	1	1	-4333	57600	5401

In [ ]:
```
X = labelled_df.loc[:, 'intensity':'curve15']
y =labelled_df['label']

# uses a 75 25 split ratio
X_scaled = StandardScaler().fit_transform(X)
X_train, x_test, Y_train, y_test = train_test_split(X_scaled, y, train_size=0.6 ,ran
```

RF Classification

In [ ]:
```
classifier_dic = {
          'RandomForestClassifier_10': RandomForestClassifier(n_estimators=10),
          'RandomForestClassifier_50': RandomForestClassifier(n_estimators=50),
          'RandomForestClassifier_100': RandomForestClassifier(n_estimators=100)
          }


def classification_func(classifier_dic, X_train, Y_train, x_test, y_test, X, y):

    for name,classifier in classifier_dic.items():

        print(f'////////////////// heyyyy we are now classifying using \t\t {name}

        classifier.fit(X_train, Y_train)
        pred = classifier.predict(x_test)
        score = classifier.score(x_test, y_test)
        print(pred)
        print(score)
        scores = cross_val_score(classifier, X, y , scoring='accuracy', cv=None) #X
        print(f"Mean score: {scores.mean()}, Std dev: {scores.std()}")
```

In [ ]:
```
classification_func(classifier_dic,  X_train, Y_train, x_test, y_test, X_scaled, y)
```

```
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_10
[2 2 3 ... 1 1 1]
0.9424643841563766
Mean score: 0.8408220161421898, Std dev: 0.05412014909464752
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_50
[2 2 3 ... 1 1 1]
0.9486637117211716
Mean score: 0.8489357128658643, Std dev: 0.0650742173098791
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_100
[2 2 3 ... 1 1 1]
0.9482396534838403
Mean score: 0.8505006961980651, Std dev: 0.06260392038926334
```

Inference

looking at the above results, we can say that at n_estimators set at 10, we get fairly good score of 84.08% with a standard deviation of 5.41%, with a train test split of 60-40%.

Next We will run the classification for even smaller n_estimators

In [ ]:

```
classifier_dic2 = {
          'RandomForestClassifier_3': RandomForestClassifier(n_estimators=3 , n_jobs
          'RandomForestClassifier_5': RandomForestClassifier(n_estimators=5, n_jobs=
          'RandomForestClassifier_10': RandomForestClassifier(n_estimators=10, n_job
          'RandomForestClassifier_15': RandomForestClassifier(n_estimators=15, n_job
          'RandomForestClassifier_30': RandomForestClassifier(n_estimators=30, n_job

          }
classification_func(classifier_dic2,  X_train, Y_train, x_test, y_test, X_scaled, y)
```

```
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_3
[2 2 3 ... 1 1 1]
0.9260169825227427
Mean score: 0.8136317209926496, Std dev: 0.0527859858688641
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_5
[2 2 3 ... 1 1 1]
0.9348060943226679
Mean score: 0.8262747545442611, Std dev: 0.05678794978618239
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_10
[2 2 3 ... 1 1 1]
0.9417071373039992
Mean score: 0.8432432118635781, Std dev: 0.05106666779926342
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_15
[2 2 3 ... 1 1 1]
0.945311632321315
Mean score: 0.8429261619245747, Std dev: 0.060946039605037895
////////////////// heyyyy we are now classifying using                RandomFores
tClassifier_30
[2 2 3 ... 1 1 1]
0.9475783245660976
Mean score: 0.8470658052492585, Std dev: 0.06259998064612073
```

In [ ]:

```
final_classifier = classifier_dic2['RandomForestClassifier_10']
# save training to pickle
final_classifier.fit(X_train, Y_train)

pickle.dump(final_classifier, open('./data/RF_trainedModel.pkl', 'wb'))
```

In [ ]:

```
# load the model from disk
loaded_model = pickle.load(open('./data/RF_trainedModel.pkl', 'rb'))

gdf = pd.read_pickle('./data/unclassfied_raw_df.pkl')
# load testing for prediction from unlabelled points

toDeleteList = labelled_df.index.to_list()

gdf.drop(toDeleteList, inplace=True, axis=0)
gdf.drop(['x','y','z','xy_point', 'edge_of_flight_line', 'classification', 'user_dat

test_points = gdf.copy()

test_points.head()
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	scan_angle	red	green

	X	Y	Z	intensity	return_number	number_of_returns	scan_angle	red	green
0	336463	400324	10399	1098	1	1	-4333	42752	40448
1	336575	400651	10389	1141	1	1	-4333	41216	38912
2	336432	399149	10422	1058	1	1	-4333	54272	48896
3	336544	399481	10422	1046	1	1	-4333	39168	33792
4	336648	399791	10426	1171	1	1	-4333	40448	35072

In [ ]:

```
newX= test_points.loc[:, 'intensity':'curve15']
newX = StandardScaler().fit_transform(newX)

test_points.loc[:,'labels'] = loaded_model.predict(newX)
test_points
```

Out[ ]:

	X	Y	Z	intensity	return_number	number_of_returns	scan_angle	red	g
0	336463	400324	10399	1098	1	1	-4333	42752	4(
1	336575	400651	10389	1141	1	1	-4333	41216	3(
2	336432	399149	10422	1058	1	1	-4333	54272	4(
3	336544	399481	10422	1046	1	1	-4333	39168	3:
4	336648	399791	10426	1171	1	1	-4333	40448	3:
...	...	...	...	...	...	...	...	...	
893677	336620	375076	12926	1484	1	1	-3667	65024	6:
893678	336548	375371	12855	1475	1	1	-3667	64512	6:
893679	336480	375598	12656	1262	1	1	-3667	46848	4'

	X	Y	Z	intensity	return_number	number_of_returns	scan_angle	red	g
893770	336511	373576	12703	1199	1	1	-3667	34304	3
893771	336447	373866	12710	1129	1	1	-3667	32256	3

399395 rows × 20 columns

Questions

1. Describe the properties of the data (sub)set that you will classify:
   1.1. How many points?

In [ ]:
```
print(f'The selected subset contains {len(subset)} points')
```

```
The selected subset contains 894608 points
```

1.2. What area does it cover?

In [ ]:
```
print(
    f"Bounding box used to create the subset is between {xmin} and {xmax} longitude
)
```

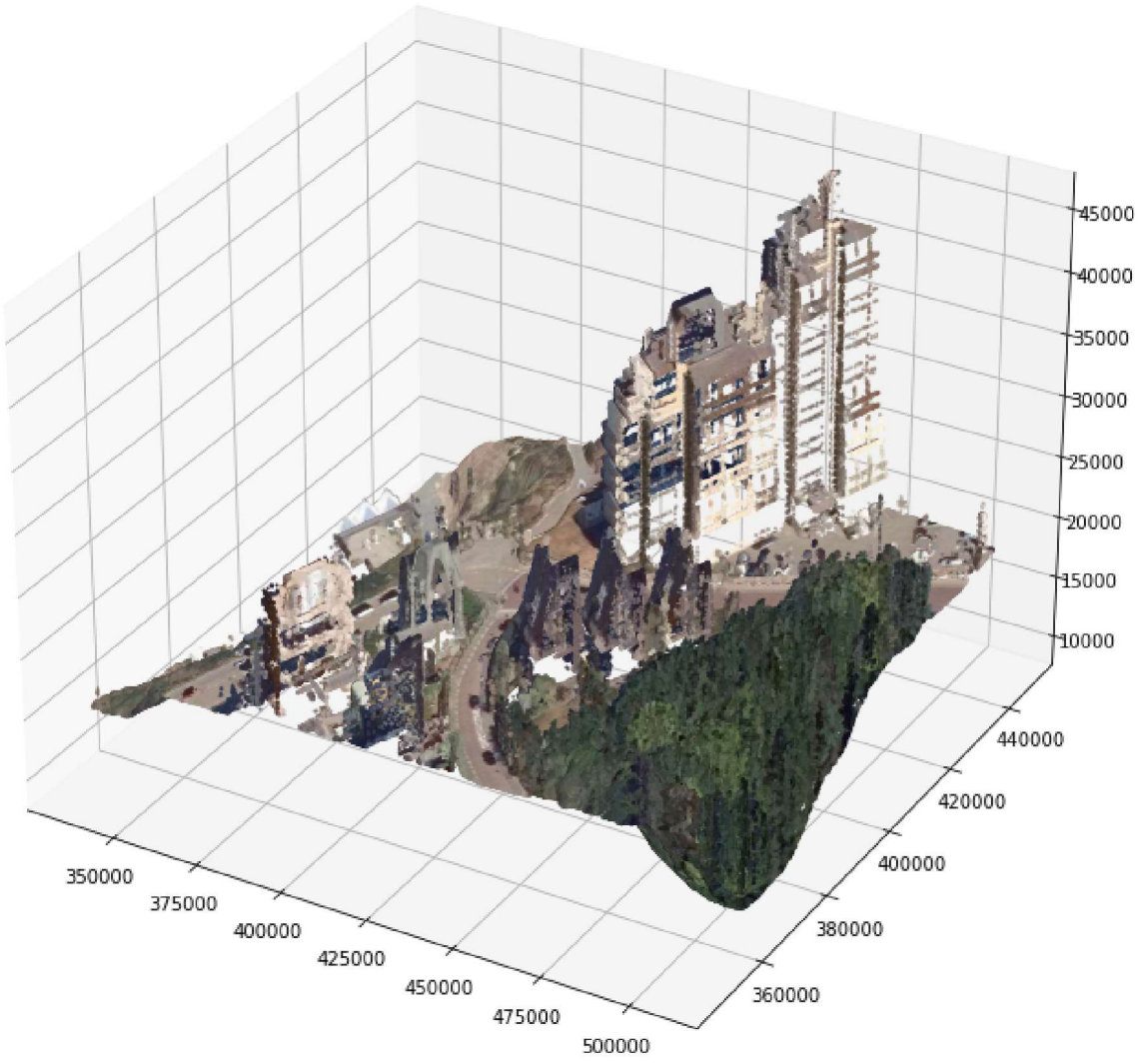
```
Bounding box used to create the subset is between 89316.42414516 and 89486.42414516
longitude and 472827.83174913 and 472927.83174913 latitude (referenced to the coordi
nates in the LAZ file [EPSG:28992])
```

1.3. What observed features will you use?

1.4. Visualize your final subset, including the useful observed features.

In [ ]:
```
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(111, projection="3d")
min_max_scaler = MinMaxScaler()
color = min_max_scaler.fit_transform(
    np.vstack([subset.red, subset.green, subset.blue]).T
)


print(color.shape)
ax.scatter(subset.X, subset.Y, subset.Z, s=1, c=color)
ax.set_zlim3d(bottom=subset.Z.min())

plt.show()
```

```
(894608, 3)
```



---

1. Describe $>$ 20 different geometric attributes, obtained using at least 2 different neighborhood sizes, to characterize your points. What are the dimensions of the data covariance matrix you use to compute the geometric features? Give one example on how you determine the PCA eigenvalues of one k-neighborhood. Indicate for each feature how it could help to distinguish your classes, given also the neighborhood sizes you consider.

Ans:

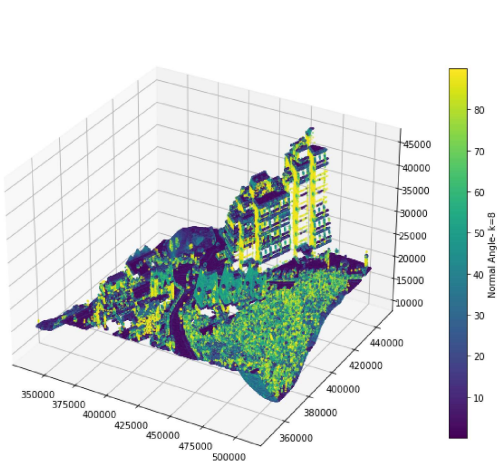
The geometric attributes were computed using a covariance matrix in 3D. To describe the geometry of real world features, fitting the PCA plane in 3D makes the most sense. For every point in the subset, the nearest *k* points were found using a KDtree, which is an efficient data structure for finding the nearest features in 3D space. When working with such a large cloud, the computational expense is very important to be practical to run on a personal computer. Once the k nearest neighbors are found, a PCA model is trained on the XYZ location of the k-neighborhood, which returns 3 vectors (the eigenvectors of the Covariance matrix). The three vectors point in direction of the most variance, the 2nd most variance, and normal to the other two, respectively. The trained PCA model also returns the eigenvalues covariance matrix, which can be used to compute the following features:

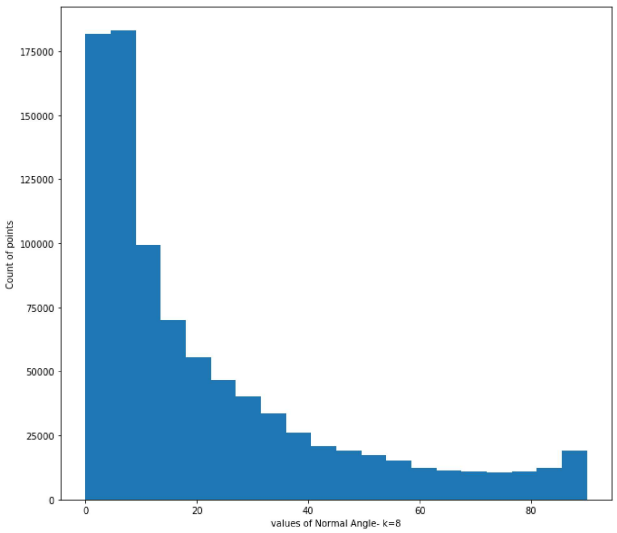
1. Compute all geometric features for all points in your subset using Python. Visualize selected results, e.g. by combining features in a false color visualization and/or using histograms. Which features are best at discriminating your classes? Why?

In [ ]:
```
def fc_vis_and_hist(point_cloud_subset, geometric_array, name):
    # create a 3d visualization
    fig = plt.figure(figsize=(24, 10))
    ax = fig.add_subplot(1, 2, 1, projection="3d")

    assert len(point_cloud_subset) == len(geometric_array)

    p = ax.scatter(subset.X, subset.Y, subset.Z, s=1, c=geometric_array)
    plt.colorbar(p, shrink=0.75, label=name)
    ax.set_zlim3d(bottom=subset.Z.min())
    # plt.title(name)

    # plt.show()

    # create a histogram
    # plt.figure(figsize=(12,7))
    ax = fig.add_subplot(1, 2, 2)

    ax.hist(geometric_array, bins=20)
    plt.suptitle(name)
    plt.xlabel(f"values of {name}")
    plt.ylabel("Count of points")
    plt.show()
```

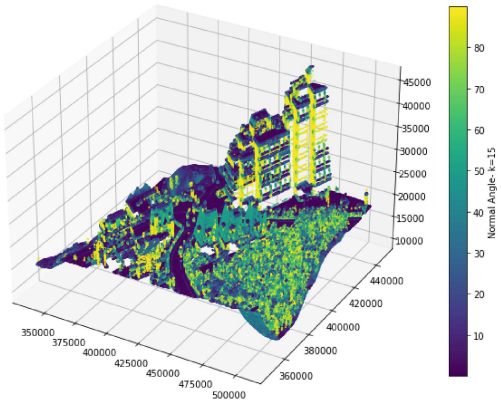
In [ ]:
```
geometric_feat_names = [
    "Normal Angle",
    "Angle to X axis",
    "Angle to Y axis",
    "Linearity",
    "Planarity",
    "Scattering",
    "Omnivariance",
    "Anisotropy",
    "Eigen Entropy",
    "Eigen Sum",
    "Change in Curvature",
]


for column8, column15, name in zip(
    pca_results8.T, pca_results15.T, geometric_feat_names
):
    fc_vis_and_hist(subset, column8, name + "- k=8")
    fc_vis_and_hist(subset, column15, name + "- k=15")
```

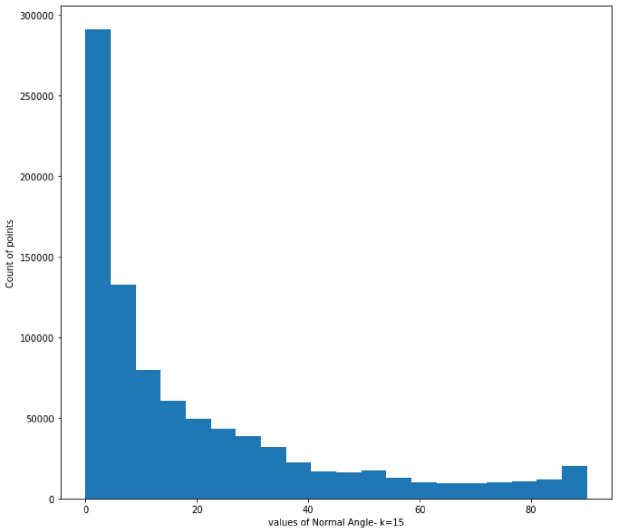
Normal Angle- k=8



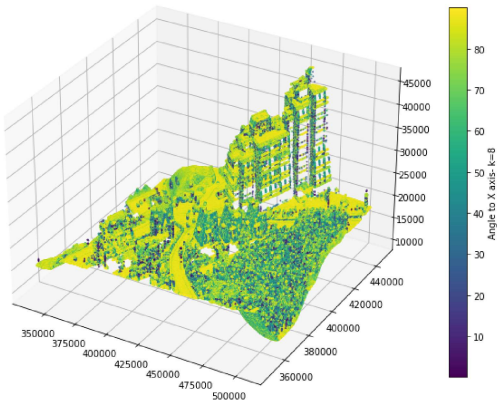


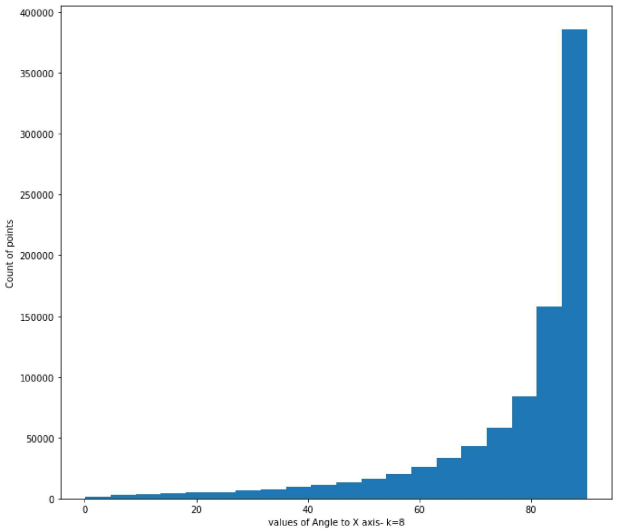
Normal Angle- k=15



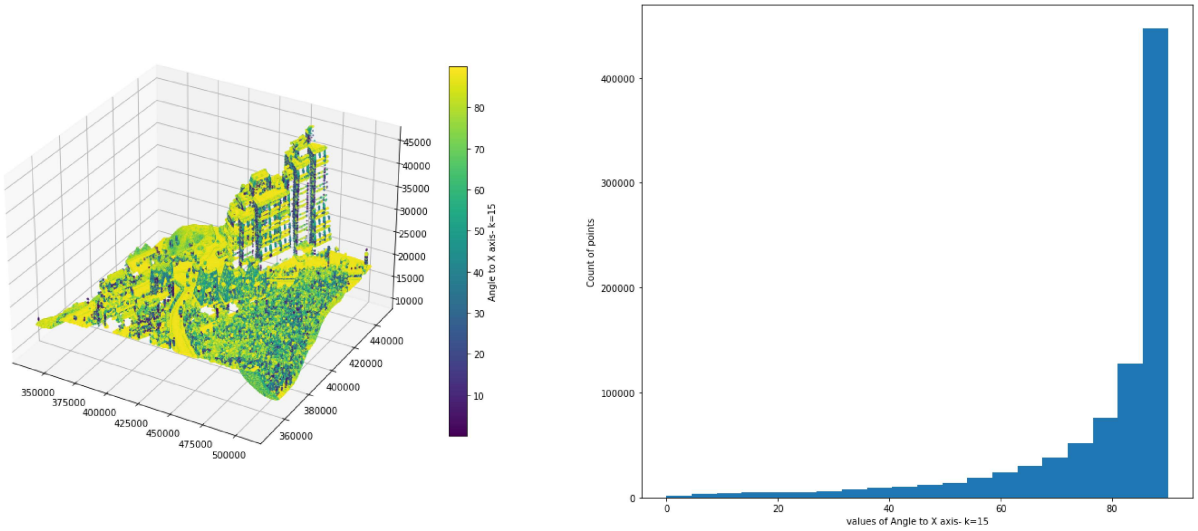


Angle to X axis- k=8

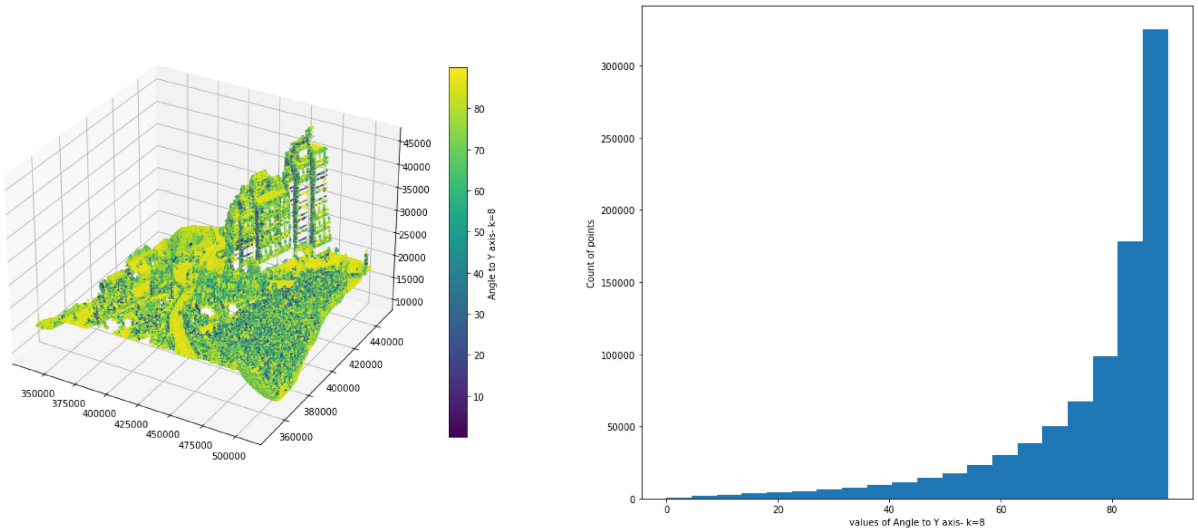




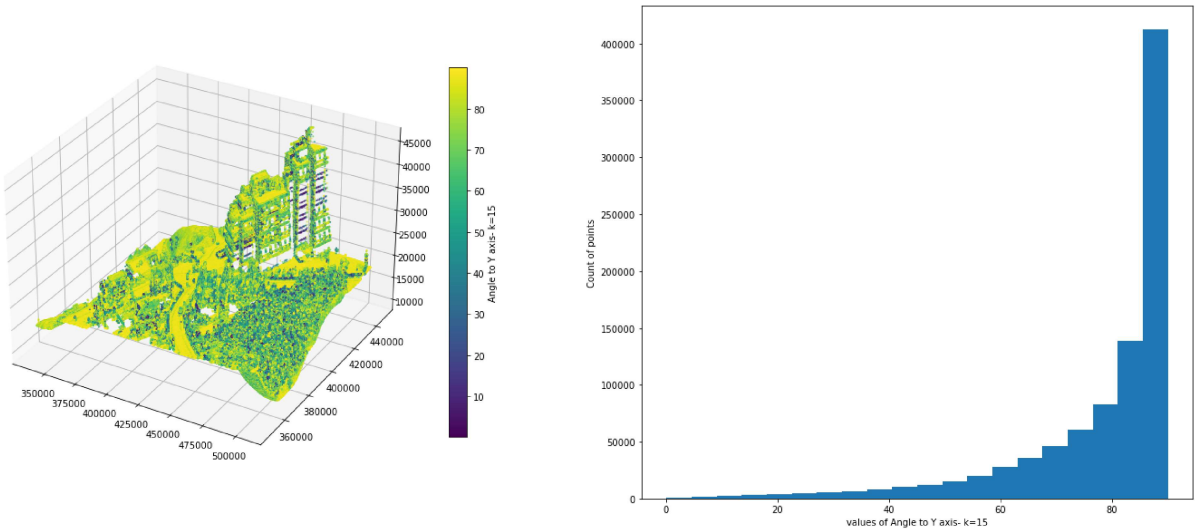
Angle to X axis- k=15



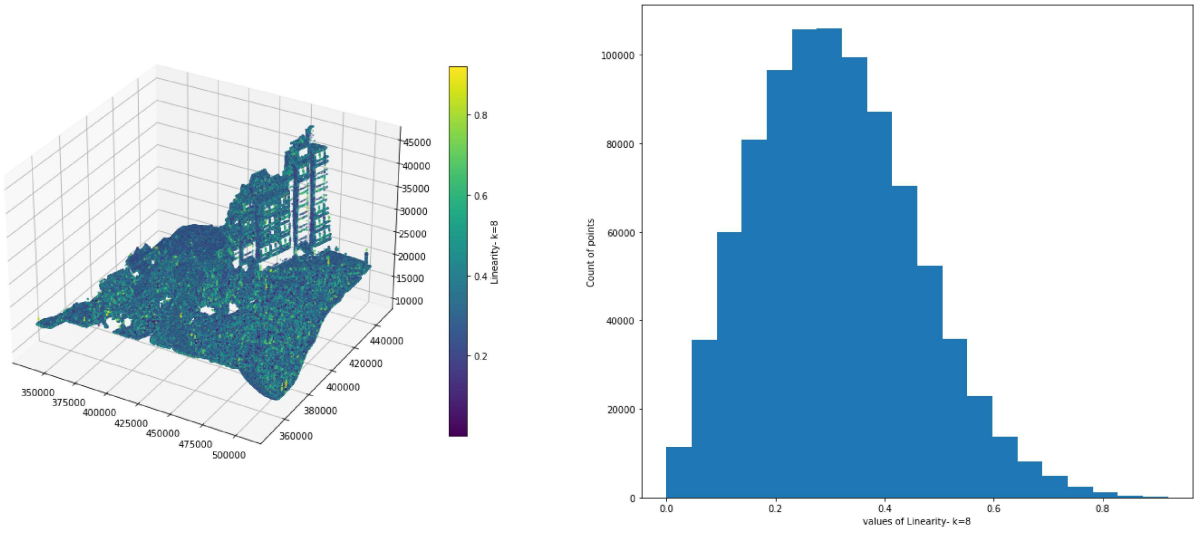
Angle to Y axis- k=8



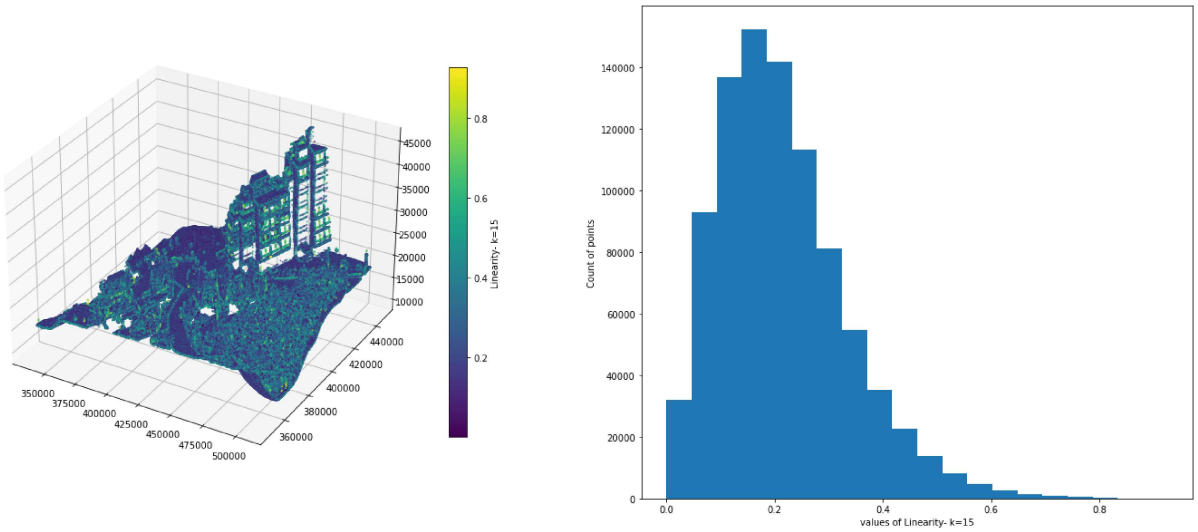
Angle to Y axis- k=15



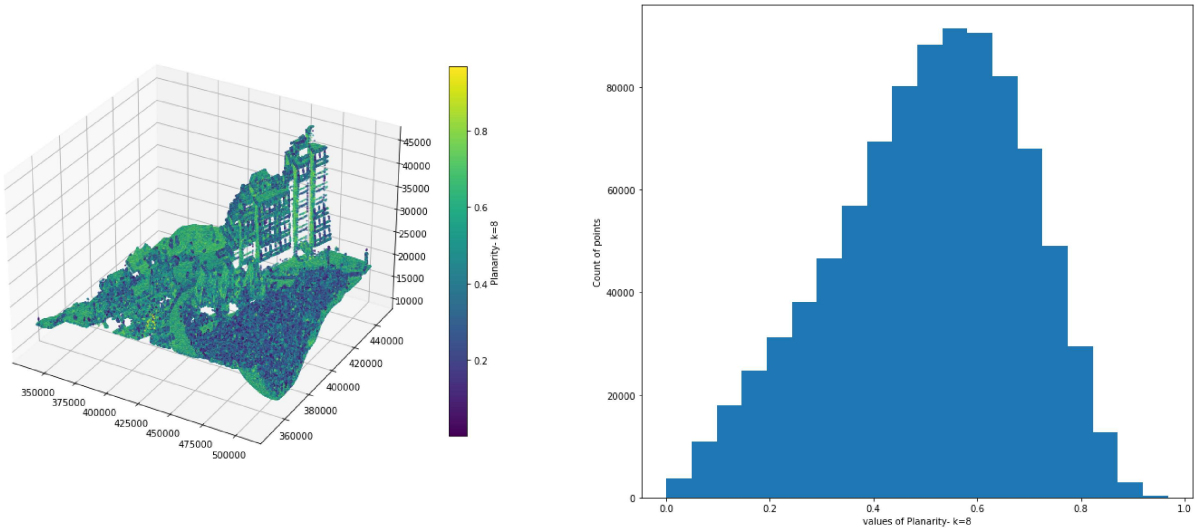
Linearity- k=8



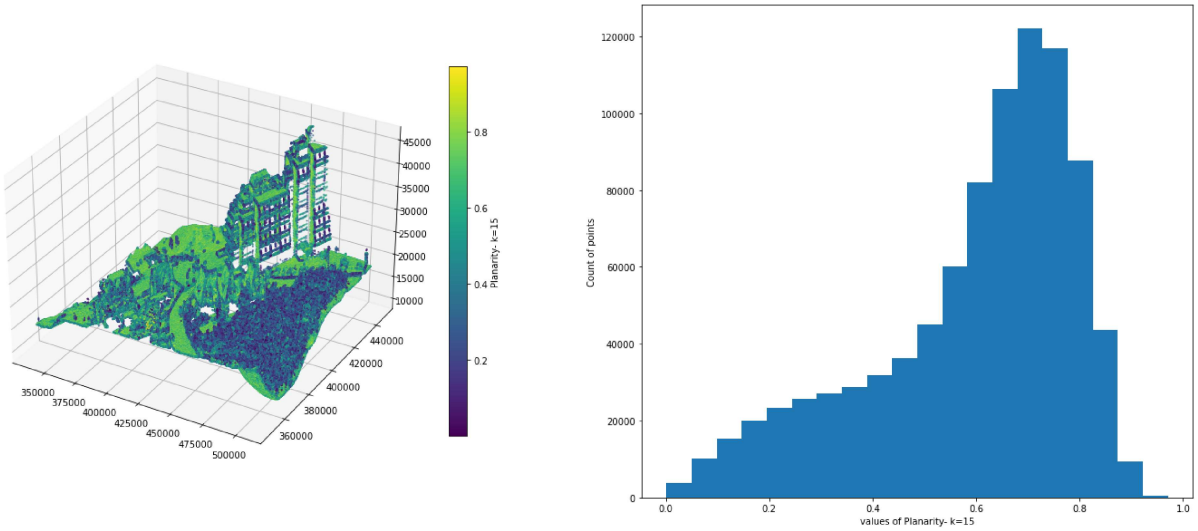
Linearity- k=15



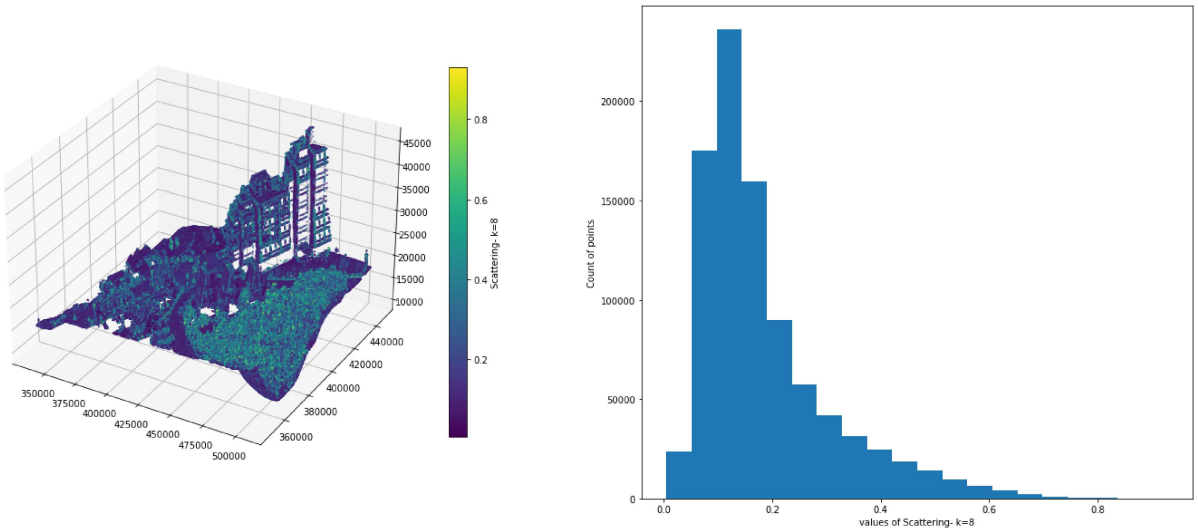
Planarity- k=8



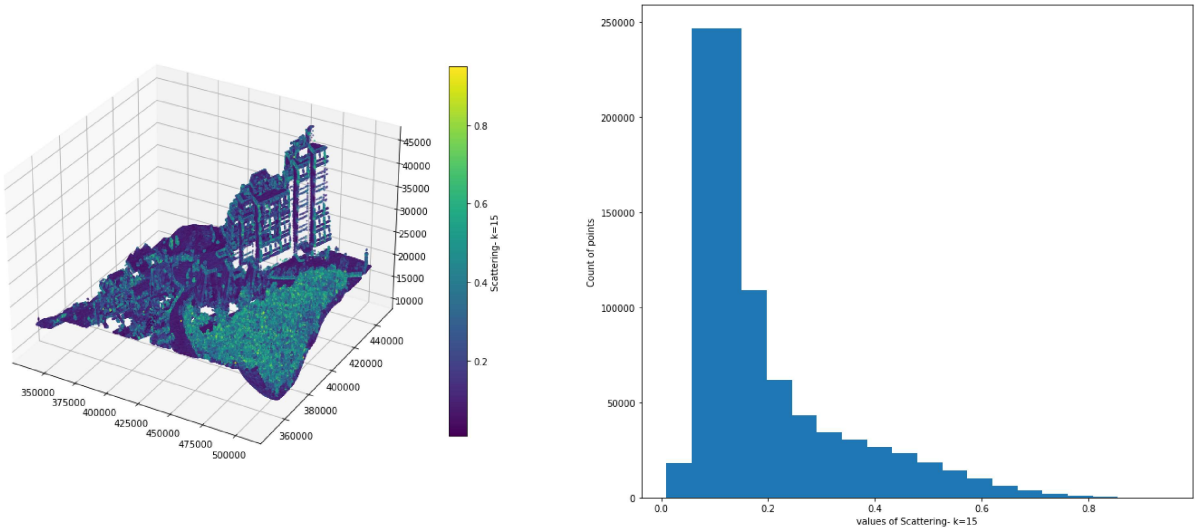
Planarity- k=15



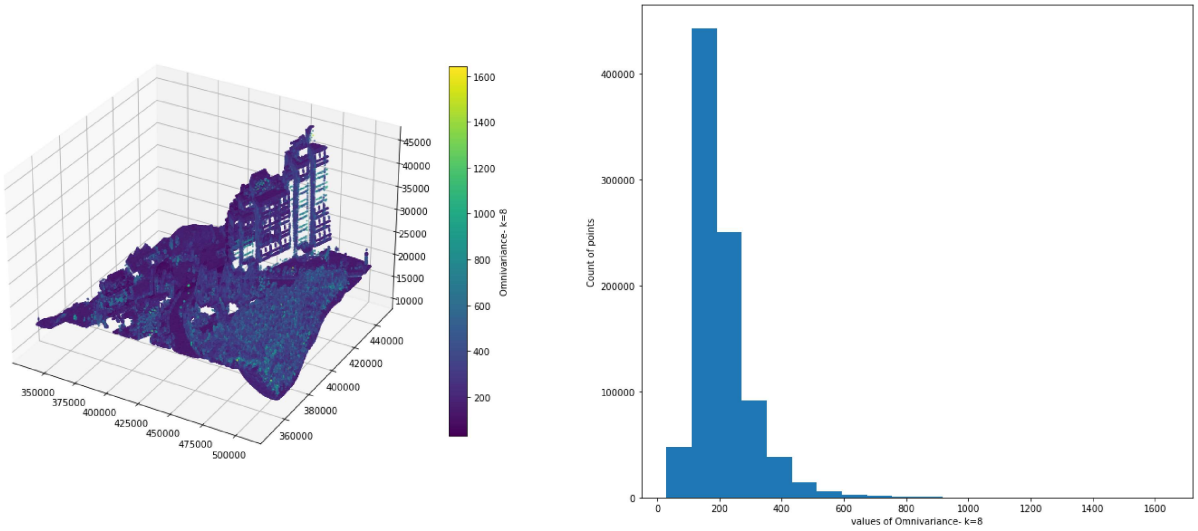
Scattering- k=8



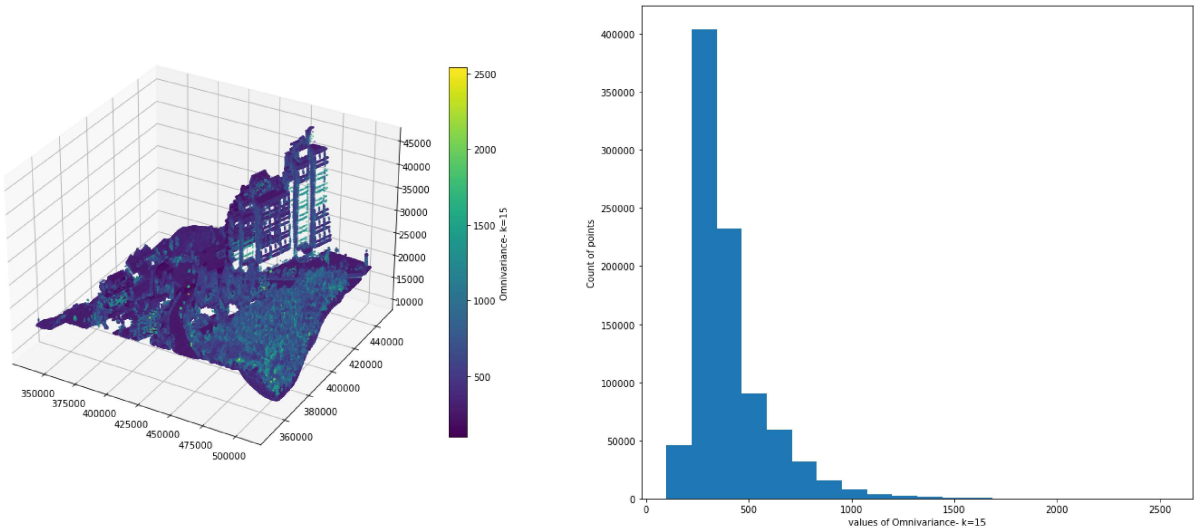
Scattering- k=15



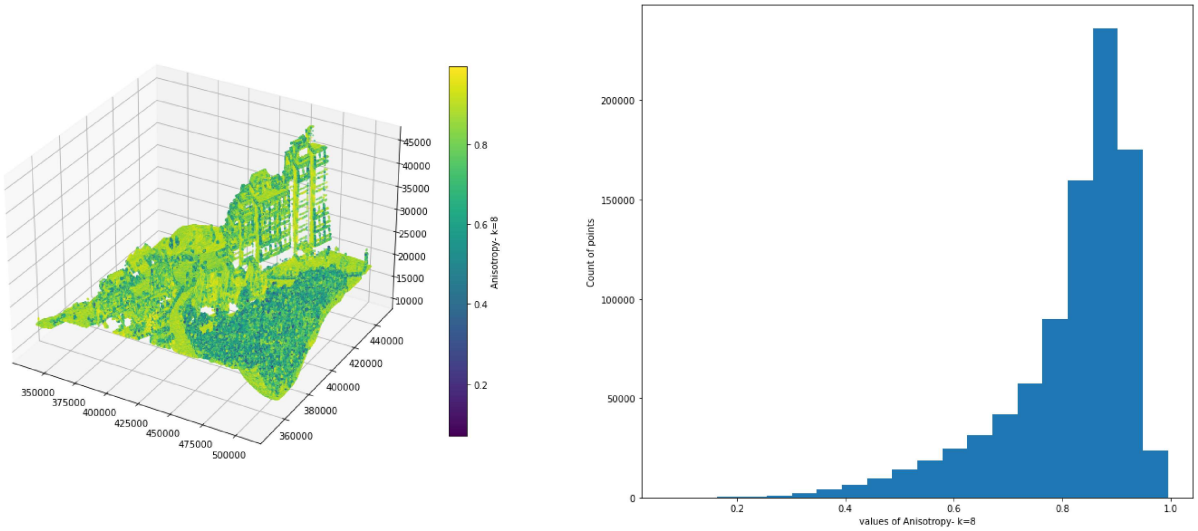
Omnivariance- k=8



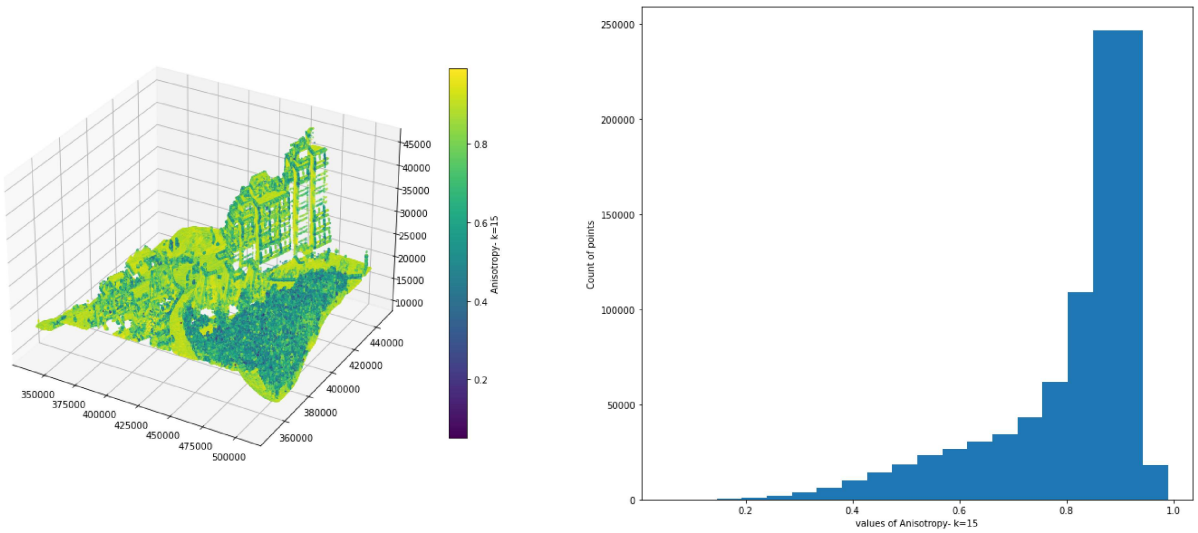
Omnivariance- k=15



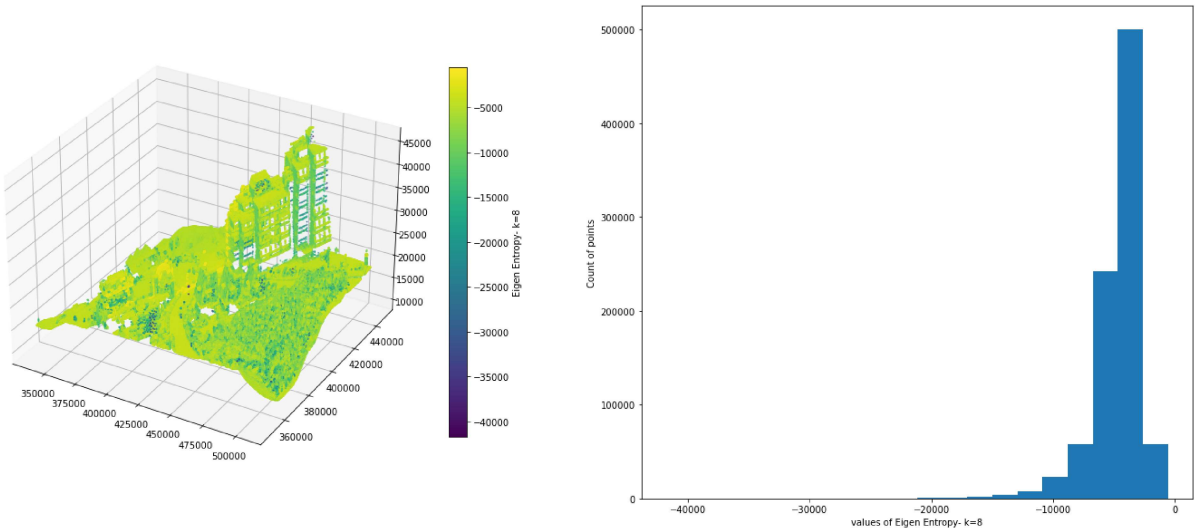
Anisotropy- k=8



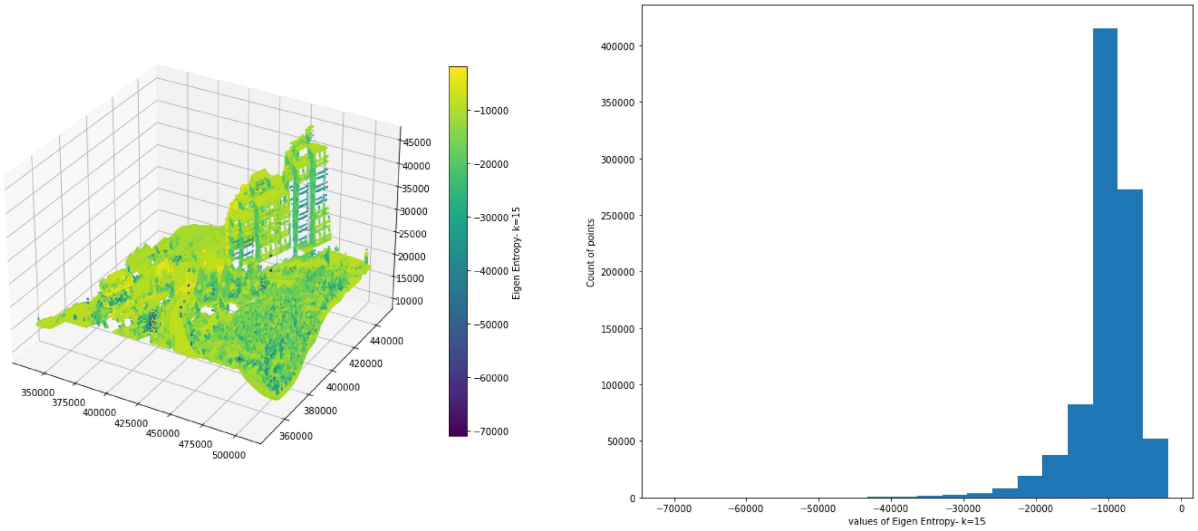
Anisotropy- k=15



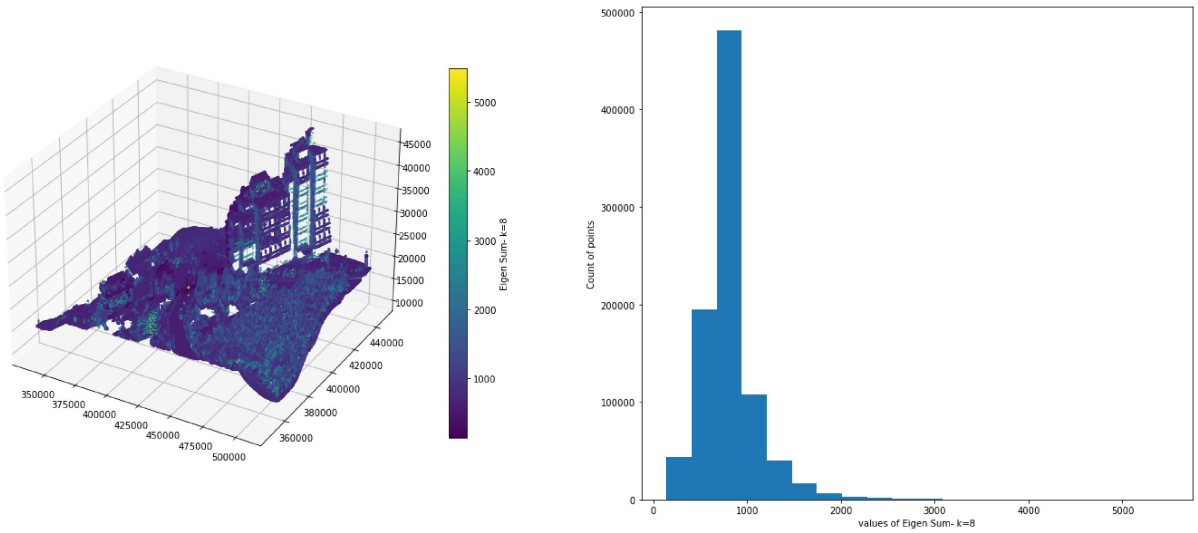
Eigen Entropy- k=8



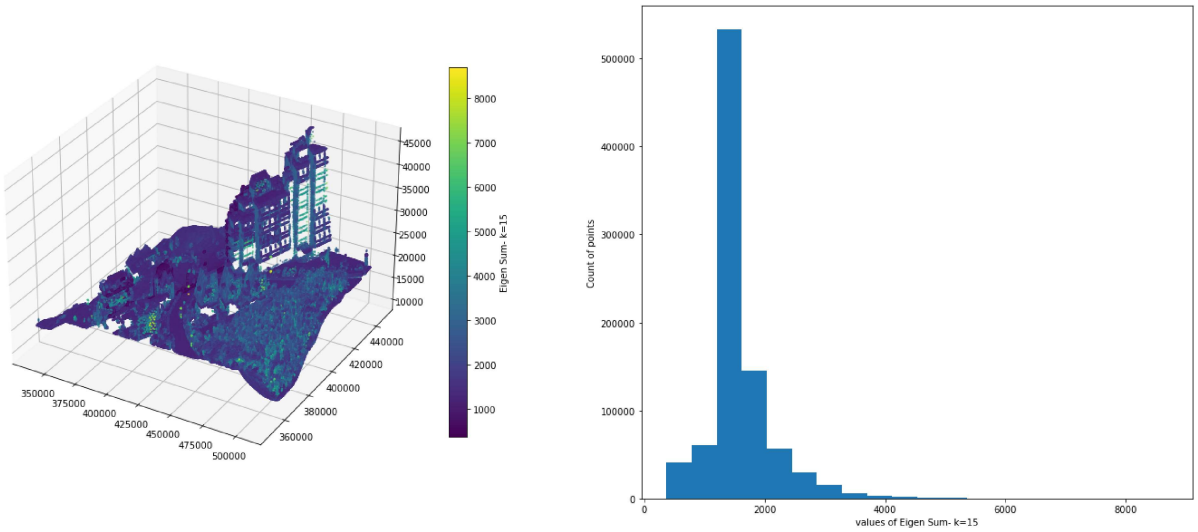
Eigen Entropy- k=15



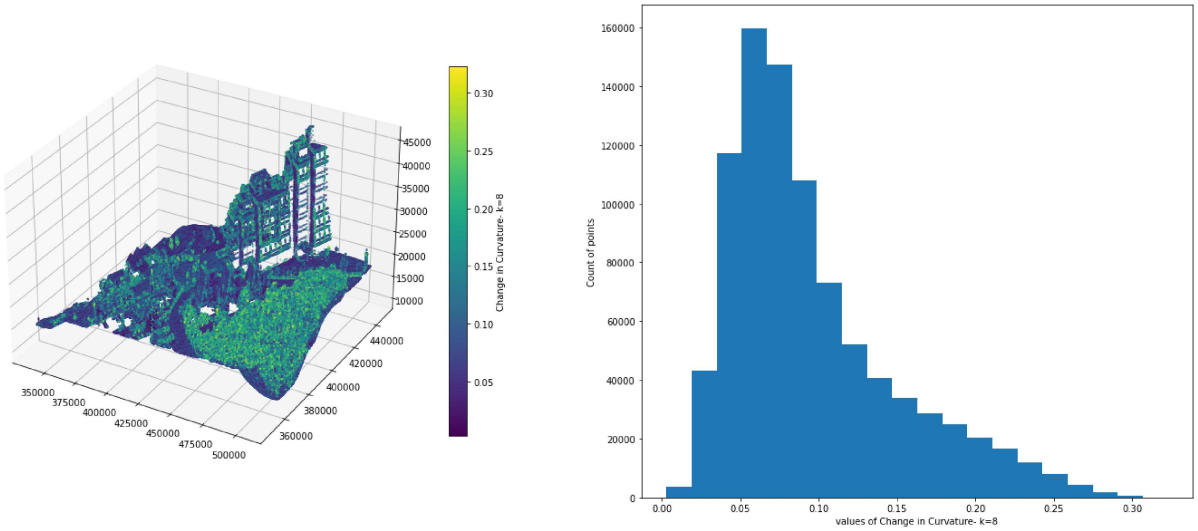
Eigen Sum- k=8

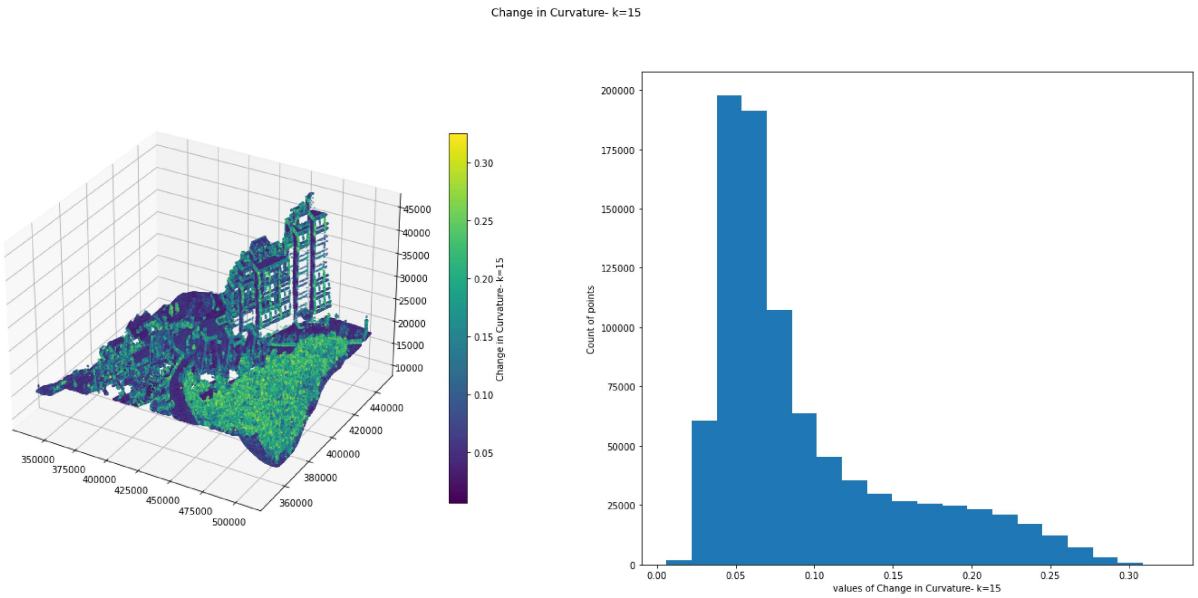


Eigen Sum- k=15



Change in Curvature- k=8





---

1. Describe and visualize your training data. Is your training data balanced? Make sure that a zone of your point cloud data is really 'unseen', that is that no training data is taken from that zone, so you can inspect if Random Forest also works there.

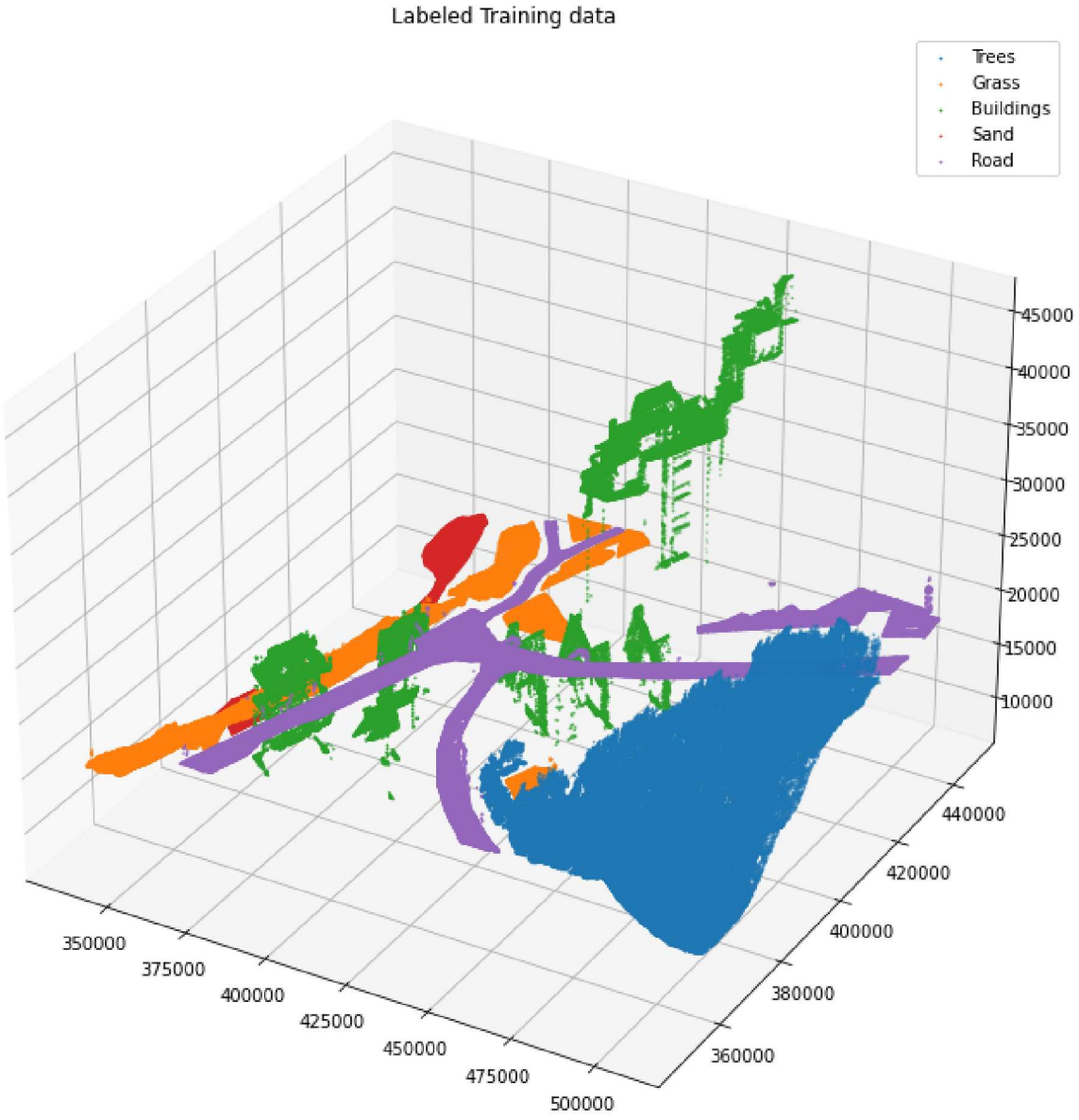
The training data is a set of 495,213 points that was created by manually creating polygons representing some desired classes, and the finding which points lie within those polygons when projected onto the 2d plane. The final count of training points for each category is:

Class	Number of points
Trees	190,242
Grass	105,998
Buildings	910,39
Sand	14,489
Road	93,455

The selected training data is shown in the figure below:

In [ ]:

```
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(111, projection="3d")

for i,name in enumerate(['Trees','Grass','Buildings','Sand','Road']):
    set = labelled_df[labelled_df.label==i+1]
    ax.scatter(set.X,set.Y,set.Z,label=name,s=0.5)
plt.title('Labeled Training data')
plt.legend()
plt.show()
```



---

1. Feed your training data to Random Forest using at least 10 of your best geometric and observed features. What settings did you use?

Summarising the mean and stddev of scores obtained from the random forest classifiers for different n_estimator values, we see:

n_estimators	mean score	stddev
3	0.8136317209926496	0.0527859858688641
5	0.8262747545442611	0.05678794978618239
10	0.8432432118635781	0.05106666779926342
15	0.8429261619245747	0.060946039605037895
30	0.8470658052492585	0.06259998064612073

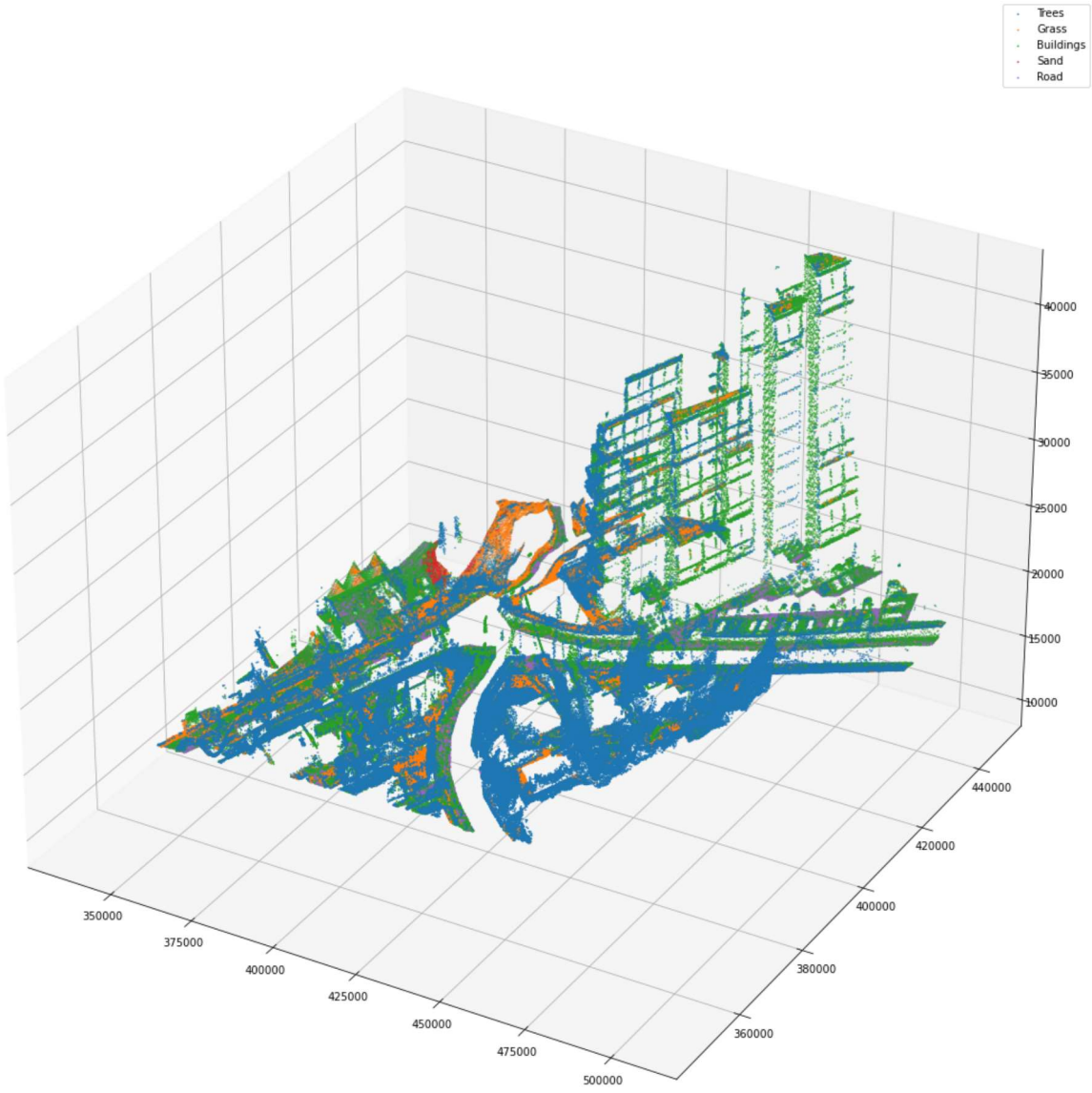
Looking at this, we choose to take n_estimators as 10 as the best possible value with all other parameters set to the default value.

---

1. Classify your point cloud data and visualize and discuss your result. What went well? Give also examples where the classifier mixed up classes. What are possible explanations for these confusions? How are the classification results on the unseen zone?

When the points which had been subset but never labelled are plotted using the classifier, we get the following figure

In [ ]:
```
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(111, projection="3d")

for i,name in enumerate(['Trees','Grass','Buildings','Sand','Road']):
    set = test_points[test_points.labels==i+1]
    ax.scatter(set.X,set.Y,set.Z,label=name,s=0.5)

plt.legend();
plt.show();
```



Upon observing the different predicted classes, we see that the following classes have been confused:

Confused classes	Possible reasoning

Confused classes	Possible reasoning
Building with trees	The vertical surfaces of building walls might have very similar geometric properties as those of trees and thus this leads to confusion for the classification model
Roads with buildings	Both buildings and roads are made from concrete, this leads to similar spectral signatures (RGB values, reflectance, intensities) and thus the highly confused classification