2018

Assignment 2: Point Cloud Classification

JHNLYD001

LYDIA JOHN

Contents

1 The Problem

Using the lidar point cloud, we can classify each point using the features that each point has. Some features that are already measured are the RGB values, the intensities and of course the x, y, z coordinates. Other features which are also of importance in classification need to be interpolated from these existing features and require additional calculations. In order to accurately classify each point, the point neighbourhood is analysed. Features such as the point normal and the corresponding eigen value can be calculated using these neighbourhoods.

The point cloud classified in this assignment was taken as a subset of points from a lidar scan of the city of Cape Town. Approximately 50 000 points are used for the classification.

The classification of the points can be carried out based on various combinations of the features. Depending on the purpose for which the classification is required we can evaluate our results and decide on the best combination, as discussed in the analysis chapter on page 5.

The quality of the point cloud (the resolution) and density of points also playa large role in the quality of the classification and is further discussed in the Methodology chapter.

Lidar scans were provided and were processed using Cloud Compare software, classified in a Python program, and then visualized in ArcScene.

2 Methodology

The processing of the lidar point cloud subset is done using a python program that organises the data in a Pandas Data Frame, performs a Principal Components Analysis, calculates the normal vector the plane fitted on the neighbourhood of points, and lastly, performs a classification by clustering. The python code for these calculations can be found in Appendix A.

2.1 Point Cloud Extraction

In Cloud Compare an area containing a fair amount of variation in surface types is selected and extracted using the selection tool and then saving the selected region as a .csv file.

The subset I chose has an elevated road surrounded by vegetation and trees and buildings with both flat and gabled roofs. The high variation will make it easier to visualise the classification and gauge the quality and reliability of it.

The csv file contains a number of attributes for each of the points and is read into a python code for further manipulation using the Pandas module. Each point from the lidar point cloud has RGB values, XYZ coordinate values, and an intensity value, among other attributes.

The Pandas data frame makes reading and extracting the relevant data much easier and organises the data in a way that is easy to understand and navigate. The data is essentially organized as a sequence of columns and rows and is labelled, making working with and manipulating the data easier and more flexible.

2.2 Obtaining Point Neighbourhoods

The point neighbourhoods are used to calculate the normal vector at each point. To get a unique solution for the calculation of a plane and its normal, a minimum of three points is required. Therefore, in order to solve the problem and obtain a least squares solution to improve the reliability of the result, a larger neighbourhood is generated. The larger the neighbourhood, the longer the iteration and the more reliable the result since it will contain minimized error. Using the k-Nearest Neighbours function in Python from the Scikit-learn library, a neighbourhood of 35 points is generated for each point and the distances and indices are saved from a ‘ball tree’ data frame. The ball tree structure is a space partitioning data structure. The indices give us the set of points in the neighbourhood of each point. The indices are referenced for the majority of the calculations as seen in the code attached in Appendix A

2.3 Feature Extraction and Normalization

Making use of the arrangement of the neighbourhoods as indices, a Principal Components Analysis is performed on each neighbourhood to get the eigen vectors and eigen values. Then, since we know that the largest eigen vector is the normal, we can use this to sort and derive the normal of each point.

Other features extracted directly from the data is the intensity of the point and the height (z value). These features however, need to be normalized before we can use them to perform the unsupervised classification.

Normalization of the data is necessary because if it is not done, the range of values in each feature in the data will act as a weight when determining the cluster, which is typically undesired since it biases the classification and decreases its reliability. Normalization of the data basically removes the outliers and aligns the data to a normal distribution making the results more reliable and speeds up the processing. In this specific instance, the normalization is feature scaling and the equation for this calculation is seen below.

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Eq. 1 Feature Scaling

2.4 Clustering

Using the Scikit-learn library, an unsupervised classification of the points is done by clustering. The clustering method used is K-means. The arguments taken in are the number of classes and the list of features. To compute k-means the 'fit' function is used with the array of features as it's input parameter.

The K-means algorithm aims to choose centroids that minimise the *inertia*, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_j - \mu_i||^2)$$

Eq. 2: K-means

Inertia, or the within-cluster sum of squares criterion, can be recognized as a measure of how internally coherent clusters are.

Since there is a lot of variation in the image and a large set of points, 200 classes are input for the seed points for the classification.

Obviously different combinations of features will give a different quality of classification depending on the purpose of the classification and therefore a total of five combinations of features are used and analysed in the next chapter.

3 Results & Analysis

As mentioned before, the final visualisation and analysis was performed in ArcScene.
For each combination of features that were used in the clustering, we can see the difference in the quality of the classification.

Figure 1 below shows the first combination which includes Eigen Values and Normal for each point.

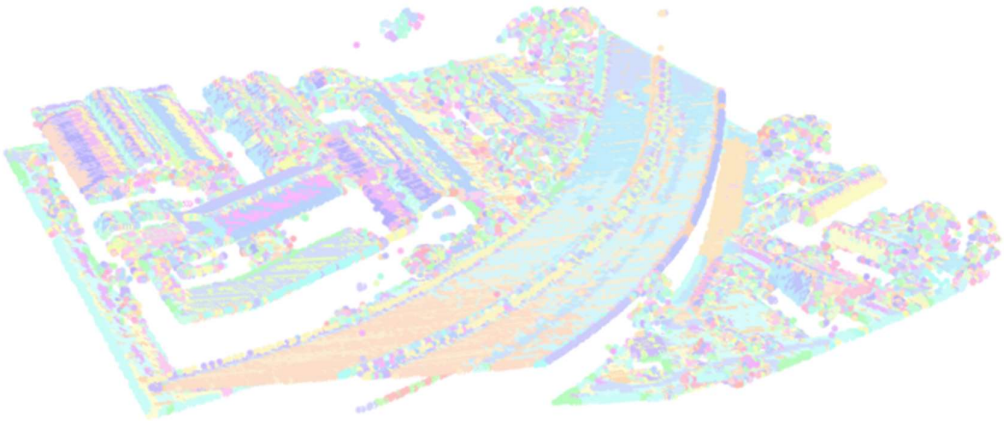

Figure 1: Clustering using Eigenvalues and Normal

The relative size of eigenvalues indicates the smoothness or roughness of edges. Steep changes in direction of normal vectors are indicative of a change in the orientation and height of features and are useful in showing the edges of buildings. The combination of these features as seen above is useful for smooth surfaces, but is not very distinctive on objects like trees and buildings with roofs of varied heights.

Figure 2 below shows the combination of the height and normal features of each point.

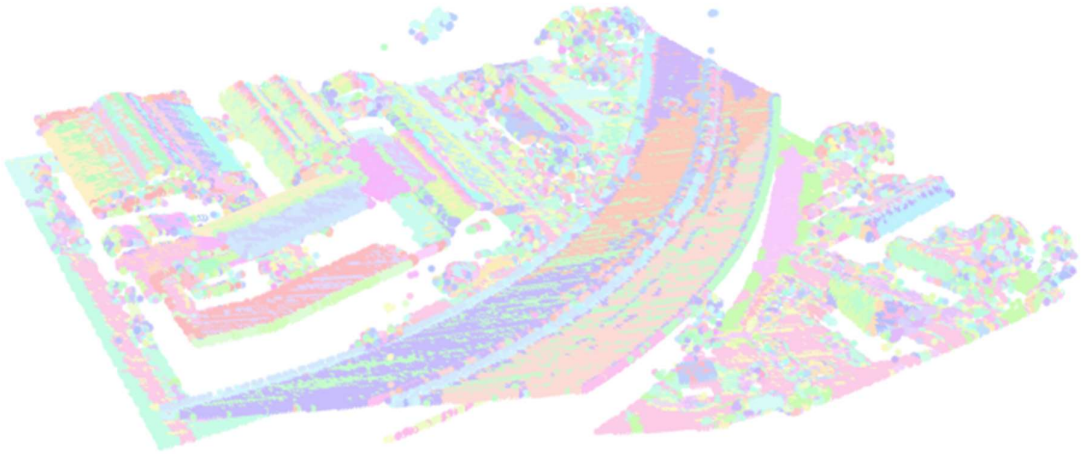

Figure 2: Clustering using Height and Normal

Both these features indicate steep changes in height which gives a detail model of the elevation of the area but does not clearly distinguish between the different surface types in the scene. We can easily distinguish differences in height, but not identify the objects very easily based on the classification.

Figure 3 below shows the classification of the scene based on intensity and normal of each point.

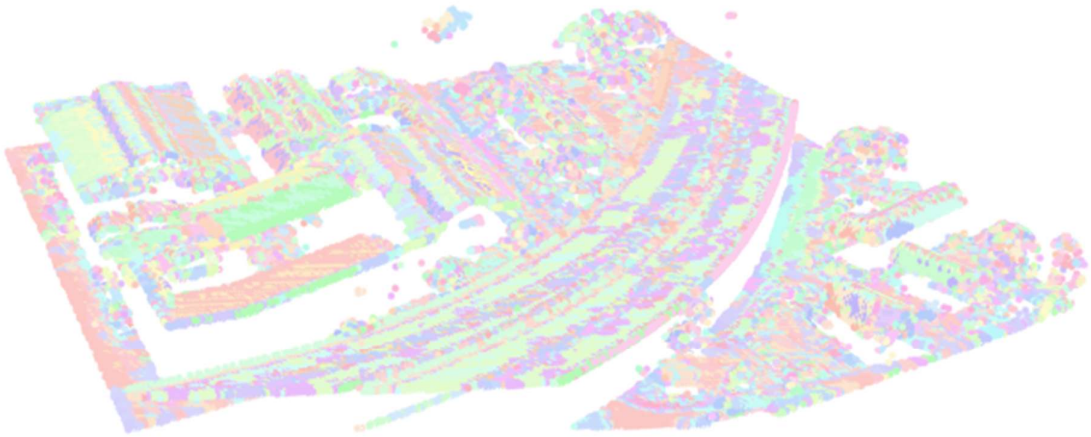

Figure 3: Clustering using Intensity and Normal

From the combination of these features we can see some variation in object edges that are differently orientated to each other but there seems to be a merging of objects that have variation over a very small surface area and therefore the classification proves to be very poor using this combination of features.

Figure 4 below shows the classification of the scene based on eigenvalues, height and normal.

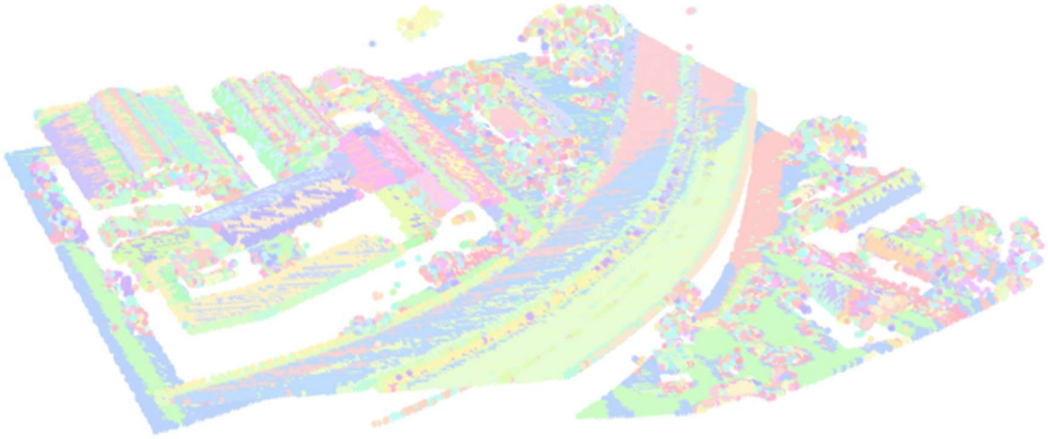

Figure 4: Clustering using Eigenvalues, Intensity & Normal

We can see here a much better and more clear distinction between objects that have different surface types and heights and can identify building edges distinctly. Vegetation is also easily distinguished from buildings due to the differences in textures as modelled using the eigenvalues.

Figure 5 below shows the last combination which includes all the features. These are the eigenvalues, the intensity, normal and height of each point.

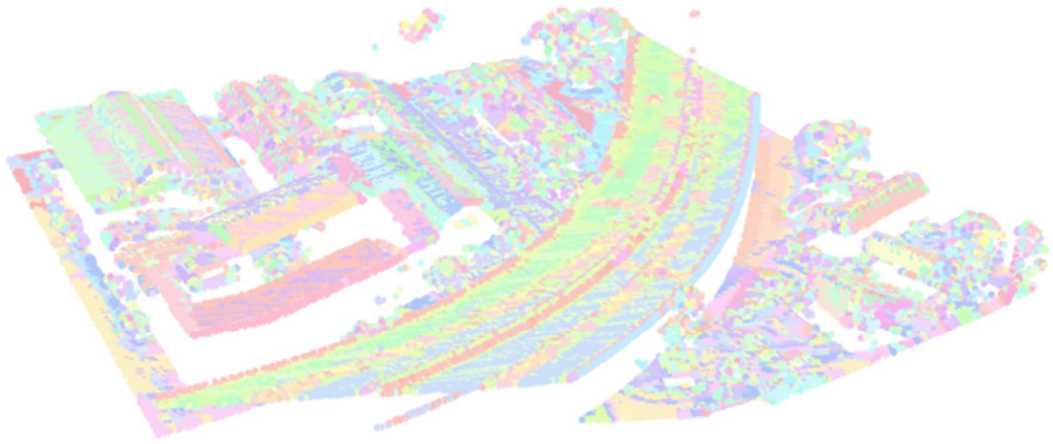

Figure 5: Clustering using Eigenvalues, Intensity, Normal, Height

Here each object in the scene is distinctly outlined and we can also see the different orientations of gabled roofs. The surface types are also distinguishable from each other and we can observe a lot more detain here than in any of the other clustering.

4 Conclusion

It is conclusive that the more features we use, the better the classification. However, more features take a longer computing time and is sometimes unnecessary. We can therefore further conclude that the best combination of features is also dependent on the purpose for which the classification of the point cloud is required.
The different features describe different attributes and enhance the objects in different ways that may sometimes have a greater importance over others. The selection of features can also be utilized to remove noise from the point cloud.

Suggestion to extend program to segment point cloud

The point cloud can be segmented by using the neighbourhoods and the location of the individual points in relation to each other to model a surface. We can segment the points based on their classification, and then classify the neighbourhood. A different segment can be identified by a steep change in the position of normal or great difference between eigenvalues, intensities and/or height.

There is always room for improvement when it comes to perfecting the solution, but as far as my own knowledge and ability to put my understanding into practice goes, the results are conclusive to a relatively good classification of the point cloud. As mentioned before, improvements will be based on the purpose of the clustering and should be carried out to aid that.

```
import pandas as pd
df = pd.read_csv('W55B - CloudSubSection.txt')
dfArray = df[['//X','Y','Z']].values
#arranging data into a structured network based on attributes and then another
network using xyz coordinates

from sklearn. neighbors import NearestNeighbors
from sklearn import preprocessing
from sklearn.cluster import KMeans
import numpy as np


neighbors = NearestNeighbors(n_neighbors = 35, algorithm =
'ball_tree').fit(dfArray)   #calculating distances to other points in and keeping
smallest distances to get nearest neighbors
distances, indices = neighbors.kneighbors(dfArray)




def PCA(data, correlation = False, sort = True):
#calcuating eigenvectors and eigenvalues for normal and plane calculation at each
point
    mean = np.mean(data, axis=0)
    data_adjust = data - mean
    if correlation:
        matrix = np.corrcoef(data_adjust.T)
    else:
        matrix = np.cov(data_adjust.T)

    eigenvalues, eigenvectors = np.linalg.eig(matrix)

    if sort:
        sort = eigenvalues.argsort()[::-1]
        eigenvalues = eigenvalues[sort]
        eigenvectors = eigenvectors[:,sort]

    return eigenvalues, eigenvectors


def best_fitting_plane(points, equation=False):
#using eigenvector and eigenvalues to fit a plane to the neigbourhood and find
normal
    w, v = PCA(points)
    normal = v[:,2]
    point = np.mean(points, axis=0)
```

```
    if equation:
        a, b, c = normal
        d = -(np.dot(normal, point))
        return a, b, c, d

    else:
        return w, point, normal



def normalization(n):
#normalizing function for features to remove biases
    min_max_scaler = preprocessing.MinMaxScaler()
#takes in 2D array and uses minimum and maximum values to scale array
    np_scaled = min_max_scaler.fit_transform(n)
    df_normalized = pd.DataFrame(np_scaled)
#returns panda dataframe of normalized data

    return df_normalized


feature_lst1 = []
feature_lst2 = []
feature_lst3 = []
feature_lst4 = []
feature_lst5 = []
xs = []
ys = []
zs = []
#empty lists to make arranging feature values easier
intensity = []
height = []

for i in range(len(indices)):
#populating lists with values from pandas dataframe
    xs1 = df.loc[i]['//X']
    ys1 = df.loc[i]['Y']
    zs1 = df.loc[i]['Z']
    intensity_vals = df.loc[i]['Intensity']

    xs.append(xs1)
    ys.append(ys1)
    zs.append(zs1)
    intensity.append(intensity_vals)
    height.append(zs1)
```

```
ints = (np.array(intensity)).reshape(-1,1)
#converting lists to 2D array for normalization function
int_norm = normalization(ints)

hts = (np.array(height)).reshape(-1,1)
ht_norm = normalization(hts)

for n,a in enumerate(indices):
#finding list index(n) of each of the indices and the index itself(a)
    iPCA = []
    for b in a:                                                                    #for
each point in the neigbourhood(indices)
        iPCA.append([df.iloc[b]['//X'],df.iloc[b]['Y'],df.iloc[b]['Z']])
#takes xyz coordinate of each point in neighbourhood and makes list for PCA

    eivals, pt, norms = best_fitting_plane(iPCA)
    evreshape = eivals.reshape(-1,1)
    ev_norm = normalization(evreshape)
#converting to 2D array for normalization

    ht = ht_norm.iloc[n][0]
    intens = int_norm.iloc[n][0]
#normalized height and intensity features for each point at the n index in pandas
dataframe
#    evr2 = (np.array(ev_norm)).reshape(1,-1)
    ev = ev_norm[0][0],ev_norm[0][1],ev_norm[0][2]
    norml = norms.tolist()
    normal = norml[0],norml[1],norml[2]

    feature_lst1.append([ht, intens, normal])
    feature_lst2.append([])
    feature_lst3.append([])
    feature_lst4.append([])
#list of feature values for k means clustering
    feature_lst4.append([])



kmeans1 = KMeans(n_clusters=10, random_state=0).fit(feature_lst1)
kmeans_labels1 = kmeans1.labels
labels1 = {'X': xs, 'Y': ys, 'Z': zs, 'kmeans labels': kmeans_labels1}
labelled1 = pd.DataFrame(data = labels1)
labelled1.to_csv('classified1.csv')

kmeans2 = KMeans(n_clusters=10, random_state=0).fit(feature_lst2)
kmeans_labels2 = kmeans2.labels
labels2 = {'X': xs, 'Y': ys, 'Z': zs, 'kmeans labels': kmeans_labels2}
labelled2 = pd.DataFrame(data = labels2)
```

```
labelled2.to_csv('classified2.csv')

kmeans3 = KMeans(n_clusters=10, random_state=0).fit(feature_lst3)
kmeans_labels3 = kmeans3.labels
labels3 = {'X': xs, 'Y': ys, 'Z': zs, 'kmeans labels': kmeans_labels3}
labelled3 = pd.DataFrame(data = labels3)
labelled3.to_csv('classified3.csv')

kmeans4 = KMeans(n_clusters=10, random_state=0).fit(feature_lst4)
kmeans_labels4 = kmeans4.labels
labels4 = {'X': xs, 'Y': ys, 'Z': zs, 'kmeans labels': kmeans_labels4}
labelled4 = pd.DataFrame(data = labels4)
labelled4.to_csv('classified4.csv')

kmeans5 = KMeans(n_clusters=10, random_state=0).fit(feature_lst5)
kmeans_labels5 = kmeans5.labels
labels5 = {'X': xs, 'Y': ys, 'Z': zs, 'kmeans labels': kmeans_labels5}
labelled5 = pd.DataFrame(data = labels5)
labelled5.to_csv('classified5.csv')
```