Practical Data Science Assignment 2 Report

By: Jialun Darren Huang (S3755729) RMIT
Email: s3755729@student.rmit.edu.au
Report created on: 10 June 2020

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show we I agree to this honor code by typing "Yes": ***Yes***

Table of Contents

Abstract/Summary

The aim of this report is to show what I have done for my Practical Data Science Assignment 2. In this assignment, we were told to choose 1 out of 3 available datasets as stated and proceed on data preparation, data exploration and data modelling. This report will explain what I have done, with reasoning and analysis and also, include any challenges and difficulties faced.

I have chosen the 2nd option (BLE RSSI Dataset for Indoor localization and Navigation Data Set).

Introduction

The (BLE RSSI Dataset for Indoor localization and Navigation Data Set) dataset was created to see the locations that are bounded by each beacon in the first floor of Waldo Library, Western Michigan University. Only the labelled dataset is managed in this assignment. For each location that the user moves to, a phone is used to detect how strong the RSSI reading is. In this assignment, I have to create models to predict the location where the user is based on the RSSI readings given by each beacon.

Methodology

Before we begin, we should first identify the aim of the experiment.

Aim of experiment

The aim of the experiment is to figure out the boundaries of each beacon in the first floor of the library. A beacon's signal has a limited range and by knowing the range of each beacon, we can figure out if there is any location in the library not having any signal.

In the future, we can also learn how popular each beacon is in the library. As a library has many designated corners, it is often that a beacon would definitely have more users compared to other beacons. This may result in poor connectivity as the number of users connected to a beacon affects the signal strength. Therefore, knowing this can help construct a better beacon boundary design, such that designated areas will have more beacons to accommodate this issue.

Data Preparation

As the data can be retrieved from the dataset webpage, there is no need to do any data gathering, however, we have to do data preparation. As I did not gather the data, I have to understand how the data looks like before we can explore any data.

Firstly, we can straight away see that the data has many negative values. This is so for our dataset as the RSSI measurements are negative values, as stated in the dataset webpage.

location	date	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
O02	10-18-2016 11:15:21	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:19	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:17	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:15	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:13	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:11	-200	-200	-82	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:09	-200	-200	-80	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
P02	10-18-2016 11:15:07	-200	-200	-86	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200

Next, on the top, we can see the label for each column. We can see that in terms of data exploration, the data column is not going to be useful so we have to keep in mind to remove it.

Finally, we have to check if there are any missing values. If there are missing values, we have to deal with it appropriately.

To begin, I read the dataset into pandas (beacon_data), named the columns as what is seen in the dataset and removed the data using .drop() method.

```
# Removing date column as it is not useful for data exploration and data modelling
beacon_data = beacon_data.drop("date", axis=1)
```

```
beacon_data
```

	location	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
1	O02	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
2	P01	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
3	P01	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
4	P01	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
5	P01	-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200

To check if there is any missing data, I used .value_counts().sum() to see how many rows have a value for each column. There should be 1420 rows as stated in beacon_data

```
locationIndex = beacon_data.columns.get_loc("location")
print(locationIndex)

lastBeaconIndex = beacon_data.columns.get_loc("b3013")
print(lastBeaconIndex)
```

```
0
13
```

```
columnNo = locationIndex

for i in range(locationIndex, lastBeaconIndex + 1):
    columnName = beacon_data.columns[columnNo]
    print(columnName)
    print()
    print(beacon_data[columnName].value_counts())
    print(beacon_data[columnName].value_counts().sum())
    print()

    columnNo = columnNo + 1
```

It turns out that there are no missing/NaN values. This means that if a value is -200, it is out of range, or a missing value as stated in the dataset webpage.

There are 2 ways to deal with -200 values. One is to set all -200 values to NaN values. However, I did not take this approach as this means that the rest of the data are still negative values, which may make data exploration an issue.

Instead, I add all values by 200. Since there are no missing values in the beacon columns, I can add all values by 200 so that all values in each row and column (except location column) are 0 and above.

	location	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
1	O02	0	0	0	0	0	122	0	0	0	0	0	0	0
2	P01	0	0	0	0	0	122	0	0	0	0	0	0	0
3	P01	0	0	0	0	0	123	0	0	0	0	0	0	0
4	P01	0	0	0	0	0	123	0	0	0	0	0	0	0
5	P01	0	0	0	0	0	123	0	0	0	0	0	0	0

I did an additional check to make sure that all numbers are 0 and above.

Data Exploration

I created 2 different graphs for my data exploration. A scatter plot to see the RSSI measurement for each location per beacon and a bar chart to count the number of locations picked up per beacon. For each beacon, I explored it with the location to see if there is anything I can learn. In total, I have 26 graphs, 2 for each beacon.

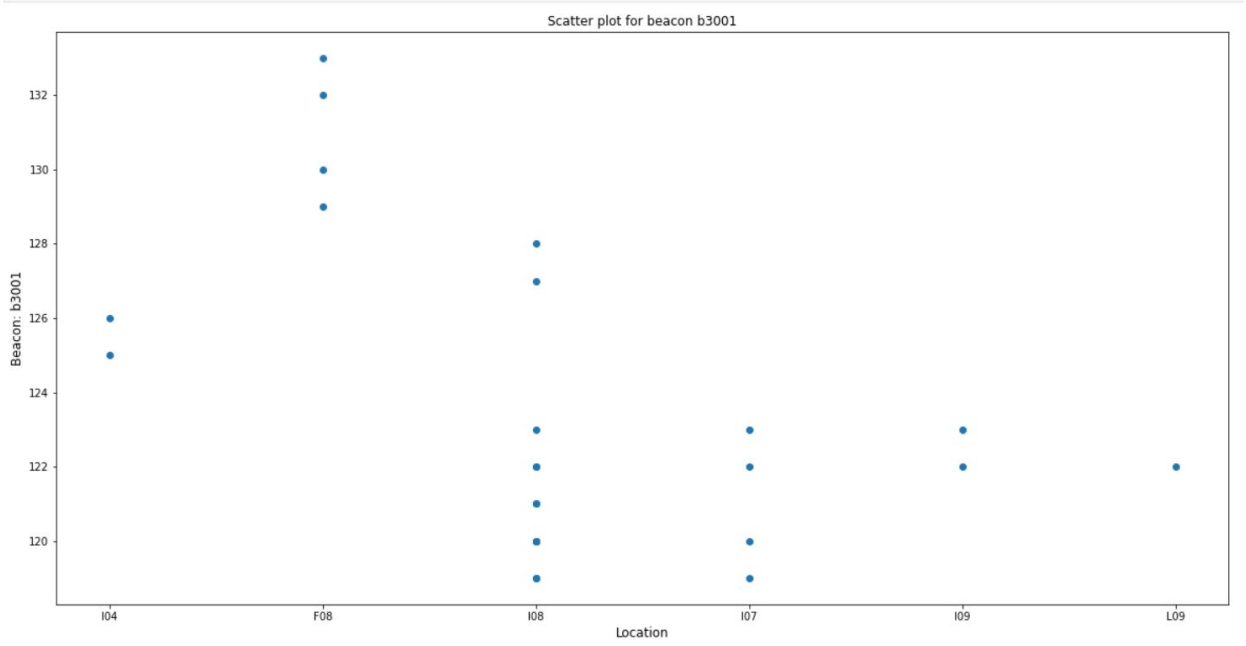
```
# Scatter Plot to see the RSSI value for each location per beacon
# Bar chart to count the number of locations picked up per beacon

columnNo = firstBeaconIndex

for i in range(firstBeaconIndex, lastBeaconIndex + 1):
    columnName = beacon_data.columns[columnNo]
    temp_beacon_data = beacon_data[["location", columnName]]
    for index, row in temp_beacon_data.iterrows():
        if(temp_beacon_data[columnName][index] == 0):
            temp_beacon_data = temp_beacon_data.drop(index)

    plt.figure(figsize=(20, 10))
    plt.title("Scatter plot for beacon " + columnName)
    plt.scatter(temp_beacon_data["location"], temp_beacon_data[columnName])
    plt.xlabel("Location", fontsize = 12)
    plt.ylabel("Beacon: " + columnName, fontsize = 12)
```

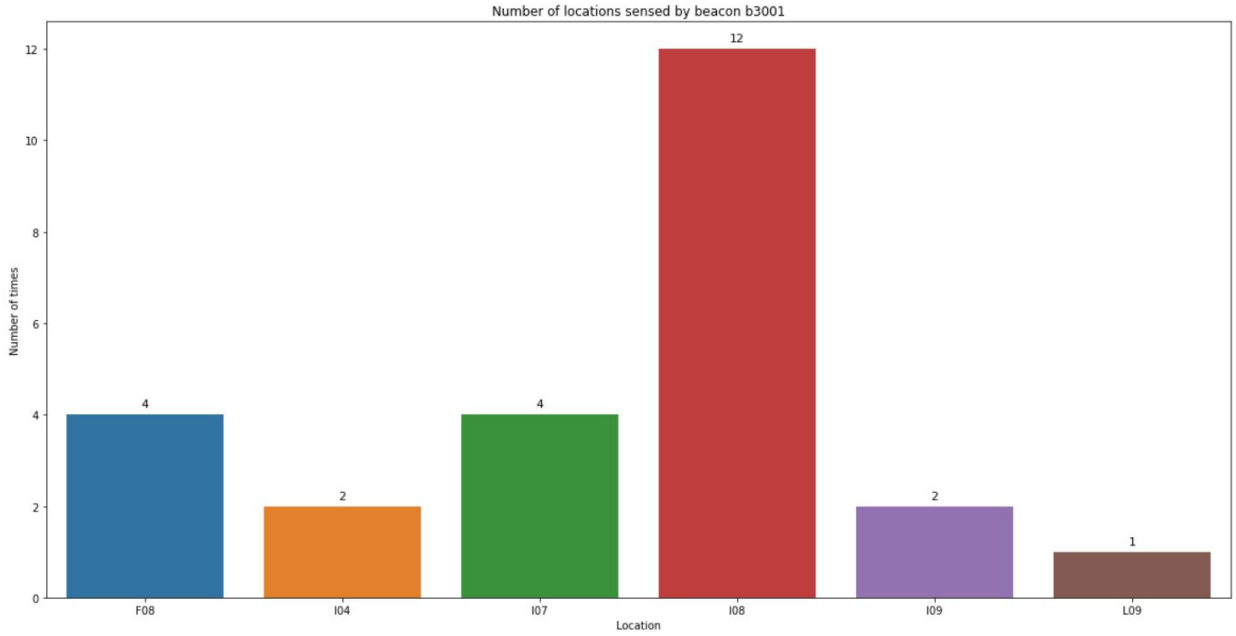
Scatter plot

```
    fig = plt.subplots(figsize = (20, 10))
    new_mainData = temp_beacon_data.groupby(["location"]).agg({columnName : "count"}).reset_index().sort_values("location")
    plt.title("Number of locations sensed by beacon " + columnName)
    g = sns.barplot(x = new_mainData["location"], y = new_mainData[columnName])

    ax = g
    ax.set(xlabel = "Location", ylabel = "Number of times")

    for index in ax.patches:
        ax.annotate(int(index.get_height()), (index.get_x() + index.get_width() / 2., index.get_height()),
            ha = "center", va = "center", fontsize = 11, color = "black", xytext = (0, 10), textcoords = "offset points")

    plt.show()


    columnNo = columnNo + 1
```

Bar Chart

Here is an example by looking at beacon (b3001)





Here we can see the RSSI values (adjusted to positive) for each beacon based on the location, and also the number of times a beacon has detected each location. The full 26 graphs are in the code.

```
: columnNo = firstBeaconIndex

for i in range(firstBeaconIndex, lastBeaconIndex + 1):
    count = 0
    columnName = beacon_data.columns[columnNo]
    for index, row in beacon_data.iterrows():
        if(beacon_data[columnName][index] != 0):
            count = count + 1
    print("beacon", str(i), ": ", str(count))
    columnNo = columnNo + 1
```

```
beacon 1 :  25
beacon 2 :  497
beacon 3 :  280
beacon 4 :  402
beacon 5 :  247
beacon 6 :  287
beacon 7 :  50
beacon 8 :  91
beacon 9 :  31
beacon 10 :  29
beacon 11 :  25
beacon 12 :  35
beacon 13 :  44
```

We can see that there are popular beacons like beacon3002 to beacon3006, as seen by the number of locations and times seen in each beacon's bar chart and their RSSI strength. However, we must not forget that firstly, this dataset does not contain all locations. Therefore, not all beacons have the same popularity. This is an issue as there is an imbalance of data per beacon.

Also, a location can be sensed by multiple beacon as this may be due to things like:

- the orientation of the phone
- the surrounding movements that may interfere with the connect between the phone and the beacon
- the movement of the user using the phone
- and many more...

One thing I would say that is missing from my data exploration is the number of beacons sensed by each location in the data. Since a location can be sensed by multiple beacons, knowing which beacons are sensed by each location can help in identifying the boundaries or overlapping of each beacon. If need to, there may be a restructure of beacon positions if the area of locations overlapped by 2 beacons is huge.

Data Modelling

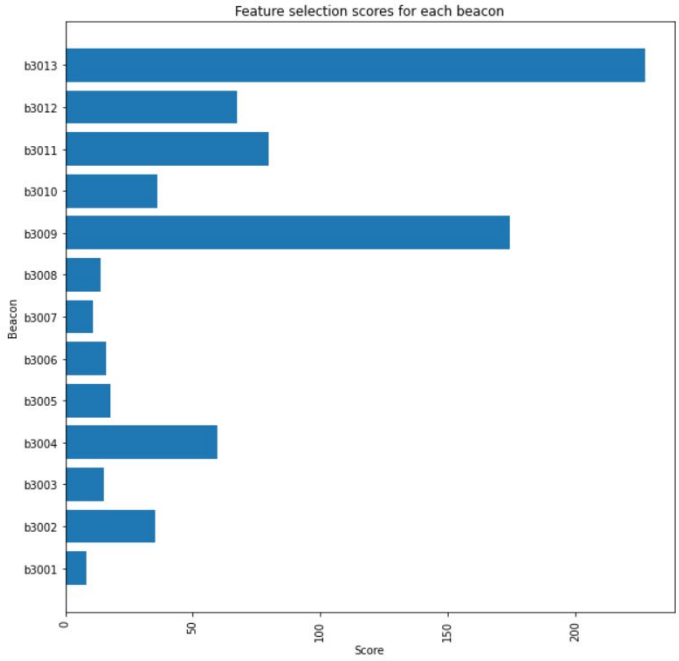
While the data can be both modelled in supervised or unsupervised learning, I have decided to go for supervised learning as we have the labels for the target, which is the location. Therefore, the two models that I use, as requested in the assignment, are K Nearest Neighbour (KNN) & Decision Tree (DT).

TrainValid and Testing data

Firstly, I split the whole 1420 data into TrainValid and Test datasets. The test dataset, which contains 20% of the original dataset will be the unseen data. The other 80% of TrainValid dataset will be used in feature selection before splitting into training and validating datasets during modelling.

Feature Selection

To start, I had to find out which features are at best to use, as using bad features may result in creating a poor model in predicting unseen data. I used SelectKBest as I am more used to and comfortable with it as my feature selection method, rather than hill climbing. I am also able to draw a graph to show which features have more impact on the dataset compared to others.



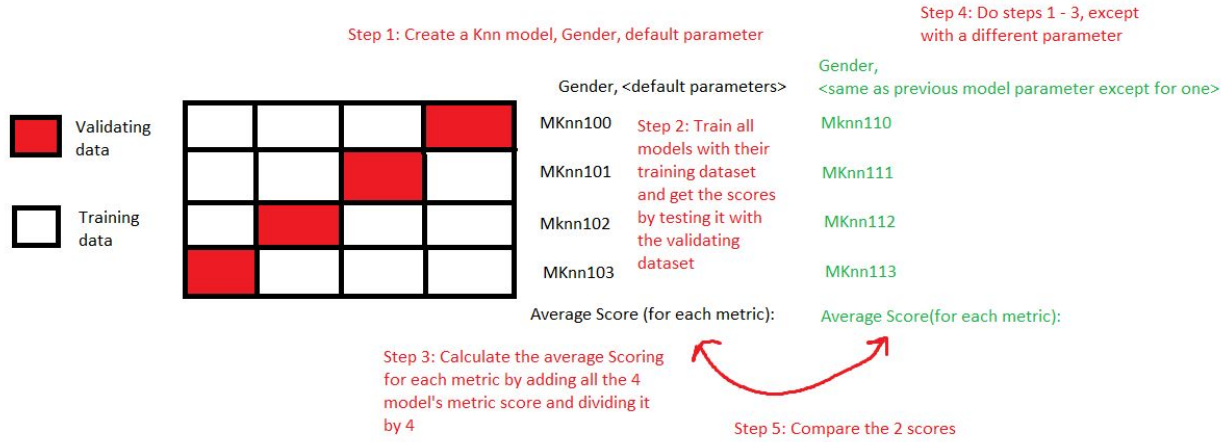
We can see that some beacons have more impact than others. I used a dummy KNN model to see how good selecting specific features will impact the predictability of the model. For this part, I used accuracy in the classification report as my metric scoring to determine how good the model is. Since I am doing feature selection, I do not need to use a more in depth metric scoring like f1-score, which will be used mainly during the modelling of each hyperparameter.

I created 2 dummy models, one to take in all features, and another to take the top 5 features (b3004, b3009, b3011, b3012 and b3013). The model that takes in all features has an accuracy of 0.35. However, it turns out the other model that takes just the top 5 features gives a low accuracy of 0.08.

I decided to add another dummy model to take in more features that are visually significant from the graph. Those are b3002 and b3010. However, with these 7 features, it still has a lower accuracy of 0.12. Therefore, I have decided that for my feature selection, I will take in all features. To me, this may be due to the number of data for each location being too little.

K fold Cross Validation

Before I begin my KNN and DT modelling, I would like to talk about cross validation. K fold Cross Validation is used as it takes in more data into the validating dataset when it comes to creating a better model. As TrainValid dataset has 1136 data left after splitting with testing dataset, I have decided to have 4 folds, as 1136/4 = 284 data. This means that for each fold, there will be 852 training data and 284 validating data.



Taken from my Assignment 1 Part 2 Report

F1-score

f1score = f1_score(“valid target”, “model”, average = "micro")
print(f1score)

Here we can see how my f1-score is calculated. The first variable in the method refers to the target, which is the location of either the validation or testing data. Next variable stores the model and finally, i set average = “micro”. By default, average = binary, however, since we have more than 2 locations, we cannot set average to binary. I set it to micro as it makes f1-score calculate all location classifications by counting the total true positives, false negatives and false positives for each location.

KNN modelling

For my KNN model, I used a brute force method on the hyperparameters to determine the best hyperparameter for the model. The hyperparameters I look into are n_neighbor, weights and p value. For each model, I adjusted each hyperparameter one by one, and took note of the average f1-score based on the 4 models created by cross validation. The maximum value n_neighbor and p can go up to is set to 10, as any more than 10 would take too much time to compute the best hyperparameters. I would say that this value is up to you, and based on how good your CPU in your laptop is. Ultimately, I found the best hyperparameters to be n_neighbor = 2, weights = distance and p = 2.

DT modelling

Before I start, I have to create an array of location labels. This is used during the display of the decision tree by graphviz.

```
location_labels = []

for index, row in beacon_data.iterrows():
    label = beacon_data["location"][index]
    size = len(location_labels)
    if size > 0:
        count = 0
        for i in range(size):
            if(location_labels[i] == label):
                count = count + 1
        if(count == 0):
            location_labels.append(label)
    else:
        location_labels.append(label)
```

Next, I want to know what hyperparameters I should look to change. They are max_depth, criterion, max_features, max_leaf_nodes, min_samples_leaf, min_samples_split. I created a dummy model that does not have any hyperparameter changes to see the expected f1-score and depth of my DT when I change my hyperparameters. It turns out that the depth is 27, less than 50, and the f1-score is about 0.281, which is about 0.3.

Using brute force again is definitely a way, however, it took me too much time to run each hyperparameter check and my jupyter notebook crashed since there are too many hyperparameters. I have decided to use what I know to fix some hyperparameters.

For criterion, I go with gini. While there is a difference in how it creates the decision tree model, it has little impact on which is better in predicting future data. However, I am going with the gini index as entropy takes a longer computational time to calculate due to it having a log function, while gini does not. For max_features, I am going with None. Based on the above feature selection calculation, we have determined that all features have to be considered. For min_samples_split and min_samples_leaf, I am going with 5 and 2 respectively. This is to reduce overfitting. The default values of split = 2 and leaf = 1 might not be feasible when the data is very hard to predict well. Finally, since I have no way to figure out the best max_depth, and this is the only hyperparameter I have left, I can brute force my way for this hyperparameter.

Initially, I used a part of the concept for hill climbing. While max_depth increases by 1 from brute force, I took the average f1-score (due to cross validation). If the current model f1-score is lesser than the previous model, then I can assume that the previous model has the better max_depth and any further increase in max_depth will not give a better f1-score.

However, it turns out that when I start my max_depth at 5, the best max_depth is 8. This does not make sense as the dummy model has a max_depth of 27. Also, when I checked, the f1-score is 0.15, which is way lower than expected. This means that I cannot use this idea and have to go back to brute forcing my max_depth instead. I set my max_depth to stop at a value of 50 and ran my brute forcing max_depth. This time, max_depth = 26 gives an f1-score of 0.239. But, this is still significantly lower than the dummy model. Therefore, I decided to run the same brute force again, except with default parameters, just like the dummy model. True

enough, the dummy model appeared better than the model with my theory. The max_depth is still 26.

With this, the final DT model is created, with only one hyperparameter max_depth, set to 26. I also created the visual decision tree by graphviz, and this can be seen in the last line of the code.

Results

Both KNN and DT model that are created with the best hyperparameters I have found are tested with the testing data to see how good my model can predict unseen data. Unfortunately, the f1-score turns out to be 0.306 and 0.288 respectively. I found out that I am not the only one having a poor score as others have said in the discussion board to also have a similar score to mine. I can assume that this is because the labelled dataset does not contain a lot of data for each location, and results in less information that the model can learn. Therefore the f1-score is low.

Discussion

As said before, this dataset comes with flaws. Firstly, the dataset does not have enough data for each location. Since the dataset is small for each location, there is not enough data for the model to learn what can or cannot be accepted for each location. This leads to a very low metric score.

Also, not all beacons have the same number of locations. This leads to an imbalance in data exploration and modelling. There is not enough data for each beacon. I can only assume without knowing that the unlabelled dataset gives more relevant information and data we need. We can use unsupervised learning techniques like K Means and DBSCAN to train the models with all the labelled and unlabelled data.

I also faced problems when doing DT modelling. I could not use brute force and my idea of hill climbing failed me. This caused a lot of wasted time and information in the code. However, I feel that there is no harm in documenting such code as it is a learning experience for me knowing what can or cannot be done, and I would like to share it to everyone reading my code and report. Also, as said in data exploration, I feel that it would be good to know the beacons sensed by each location.

Conclusion

In conclusion, while the dataset may have some flaws as discussed in the above discussion, I am happy to show you my thought process and reasoning for what I have done in my data during data preparation, data exploration and data modelling. I hope that the report can show you how to create a model to draw the boundaries of each beacon, based on the data gathered.

References

Dataset - BLE RSSI Dataset for Indoor localization and Navigation Data Set
https://archive.ics.uci.edu/ml/datasets/BLE+RSSI+Dataset+for+Indoor+localization+and+Navigation#

Sklearn libraries
https://scikit-learn.org/stable/index.html