EE 259 PROJECT

A Classification Problem

BLE RSSI Dataset for Indoor localization and Navigation

Deepanshu Saini

May 13, 2018

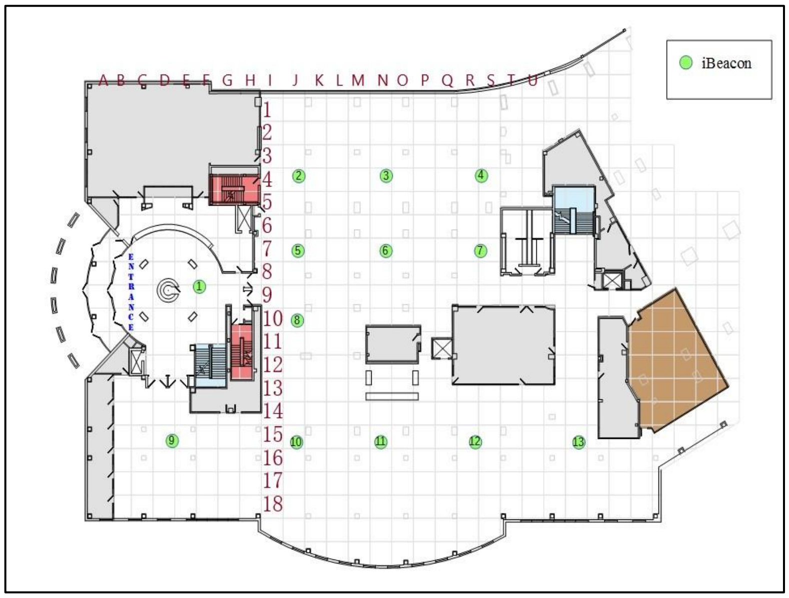
Department of Electrical Engineering

San José State
UNIVERSITY

Table of Contents

Data Set Description

- The dataset was created using the RSSI readings of an array of *13 ibeacons*.
- Data was collected using *iPhone 6S*. RSSI measurements are *negative* values.
- Bigger RSSI values indicate closer proximity to a given iBeacon (e.g., RSSI of *-65* represent a closer distance to a given iBeacon compared to RSSI of *-85*).
- For *out-of-range* iBeacons, the RSSI is indicated by *-200*.
- The locations related to RSSI readings are combined in one column consisting a letter for the column and a number for the row of the position.
- The attached figure depicts the layout of the iBeacons as well as the arrangement of locations [1]

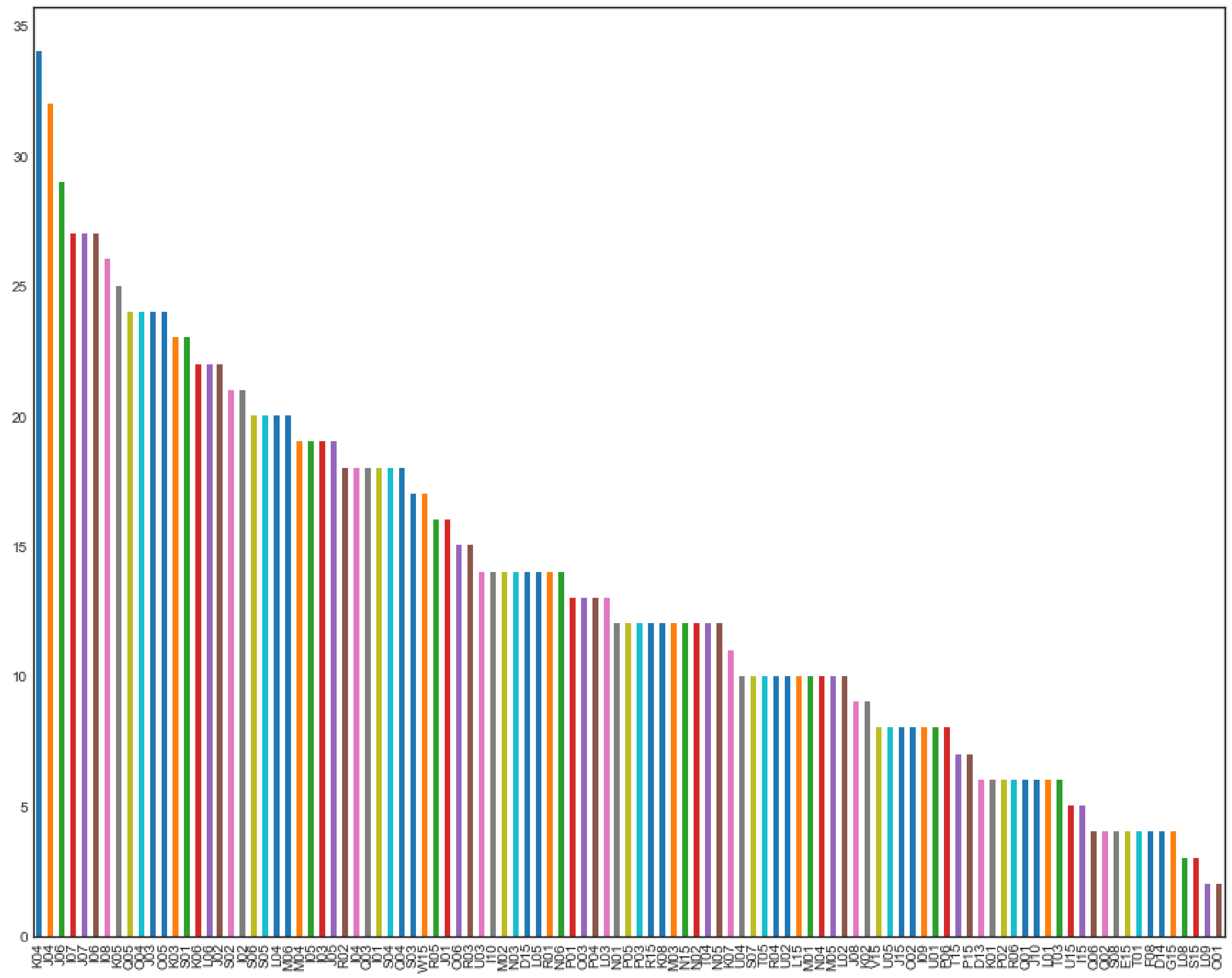


- There dataset has a total of **1420** instances with **15** attributes (13 ibeacon *RSSI* values columns, 1 *date* column, 1 *location* column)
- The date column is a series data with continuous values of time in the format (mm-dd-yyyy hh:mm:ss)

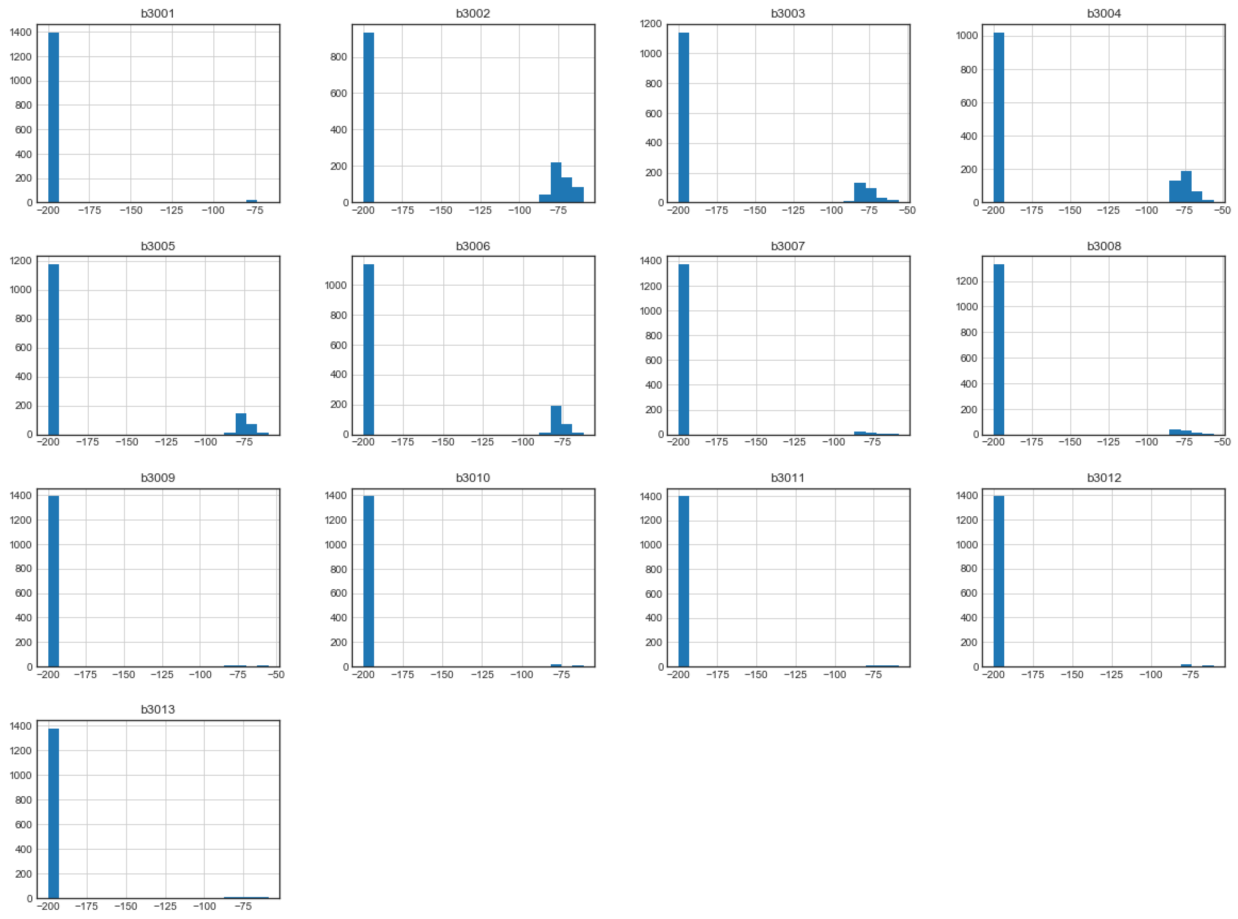
location	date	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
O02	10-18-2016 11:15:21	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:19	-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
P01	10-18-2016 11:15:17	-200	-200	-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
R01	10-18-2016 11:15:05	-200	-200	-200	-75	-200	-200	-200	-200	-200	-200	-200	-200	-200

Data Set Visualization

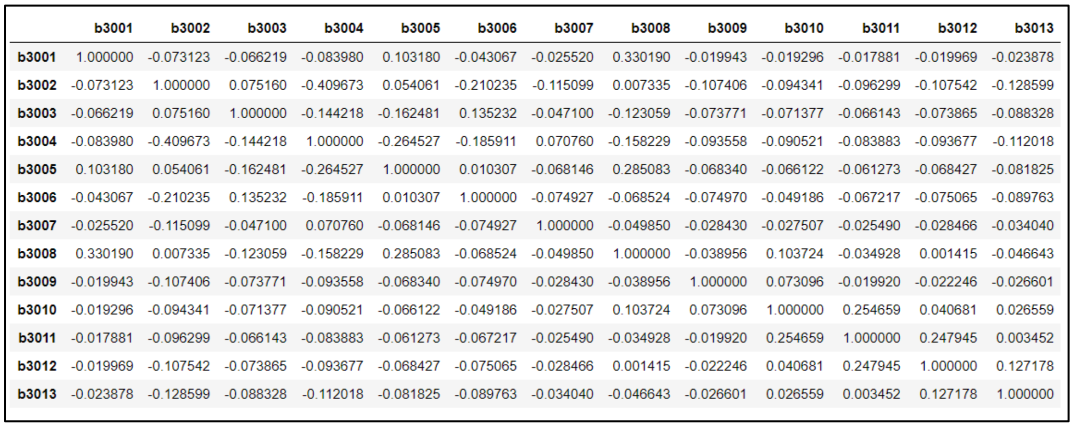
- The location column contains **105** unique values (classes) out of all the 468 (18 *rows* x 26 *columns*) possible combinations as described in the floor map.
- The *K04* position with a value count of **34** is the most frequently visited whereas *O01and L09* are the least frequently visited positions with a value count of just **2**. Thus, the dataset is quite imbalanced.
- The graph below shows the value counts of all the *105* positions given in the dataset



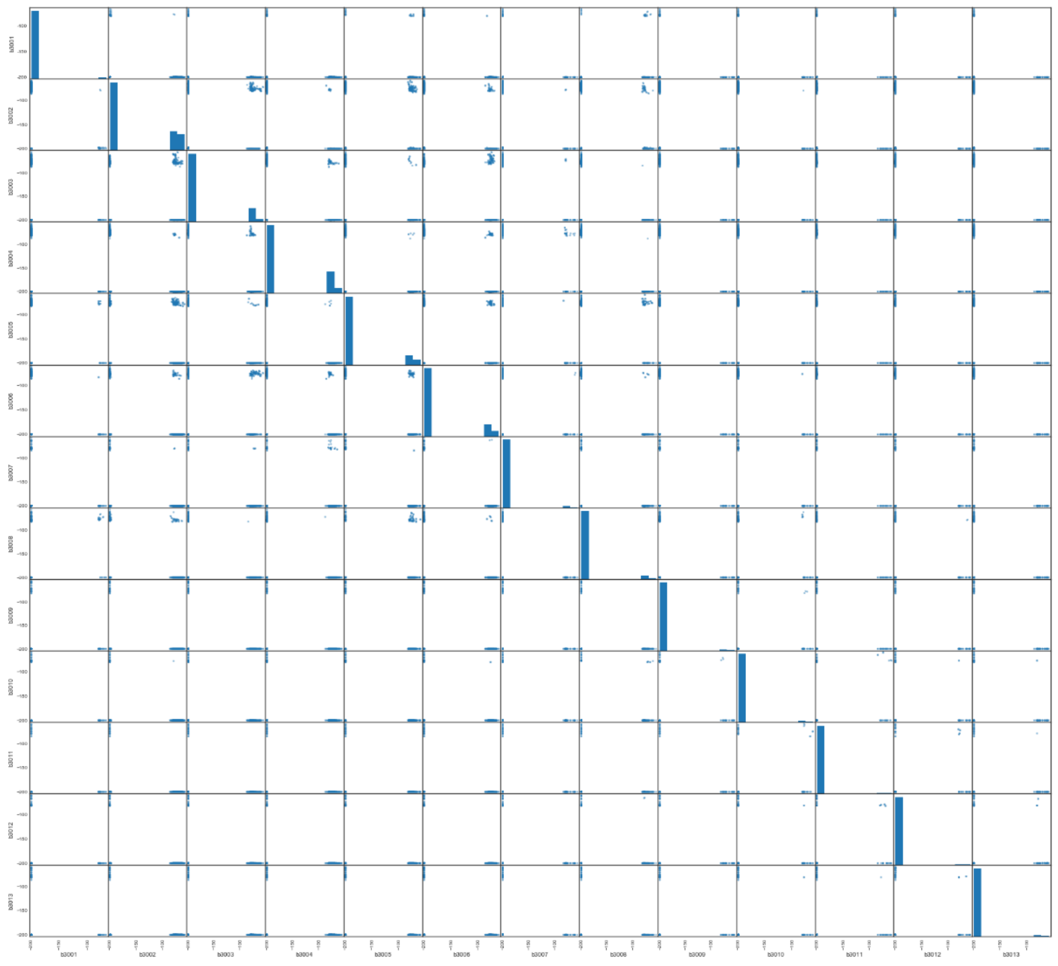
- The figure below shows the histogram plots for all the 13 input features.
- As we can see in in the plot, the beacons 2-6 have the maximum number of activations (values greater than -200) in the dataset.
- The rest of the beacons remain inactive/dormant for the most part of the time.



- The graphs below show the scatter matrix plot for all the 13-input feature set.
- As we can see in in the plot, the beacons which are close to each other have high correlation as compared to the beacons which are located far apart.

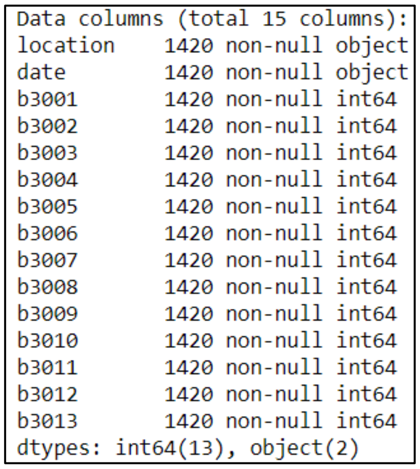


	b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
b3001	1.000000	-0.073123	-0.066219	-0.083980	0.103180	-0.043067	-0.025520	0.330190	-0.019943	-0.019296	-0.017881	-0.019969	-0.023878
b3002	-0.073123	1.000000	0.075160	-0.409673	0.054061	-0.210235	-0.115099	0.007335	-0.107406	-0.094341	-0.096299	-0.107542	-0.128599
b3003	-0.066219	0.075160	1.000000	-0.144218	-0.162481	0.135232	-0.047100	-0.123059	-0.073771	-0.071377	-0.066143	-0.073865	-0.088328
b3004	-0.083980	-0.409673	-0.144218	1.000000	-0.264527	-0.185911	0.070760	-0.158229	-0.093558	-0.090521	-0.083883	-0.093677	-0.112018
b3005	0.103180	0.054061	-0.162481	-0.264527	1.000000	0.010307	-0.068146	0.285083	-0.068340	-0.066122	-0.061273	-0.068427	-0.081825
b3006	-0.043067	-0.210235	0.135232	-0.185911	0.010307	1.000000	-0.074927	-0.068524	-0.074970	-0.049186	-0.067217	-0.075065	-0.089763
b3007	-0.025520	-0.115099	-0.047100	0.070760	-0.068146	-0.074927	1.000000	-0.049850	-0.028430	-0.027507	-0.025490	-0.028466	-0.034040
b3008	0.330190	0.007335	-0.123059	-0.158229	0.285083	-0.068524	-0.049850	1.000000	-0.038956	0.103724	-0.034928	0.001415	-0.046643
b3009	-0.019943	-0.107406	-0.073771	-0.093558	-0.068340	-0.074970	-0.028430	-0.038956	1.000000	0.073096	-0.019920	-0.022246	-0.026601
b3010	-0.019296	-0.094341	-0.071377	-0.090521	-0.066122	-0.049186	-0.027507	0.103724	0.073096	1.000000	0.254659	0.040681	0.026559
b3011	-0.017881	-0.096299	-0.066143	-0.083883	-0.061273	-0.067217	-0.025490	-0.034928	-0.019920	0.254659	1.000000	0.247945	0.003452
b3012	-0.019969	-0.107542	-0.073865	-0.093677	-0.068427	-0.075065	-0.028466	0.001415	-0.022246	0.040681	0.247945	1.000000	0.127178
b3013	-0.023878	-0.128599	-0.088328	-0.112018	-0.081825	-0.089763	-0.034040	-0.046643	-0.026601	0.026559	0.003452	0.127178	1.000000

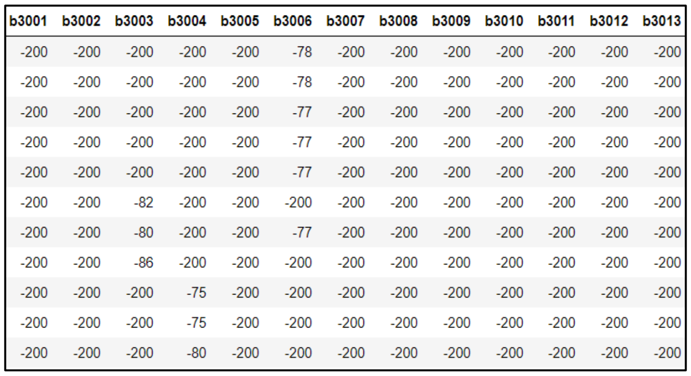


Data Set Cleaning

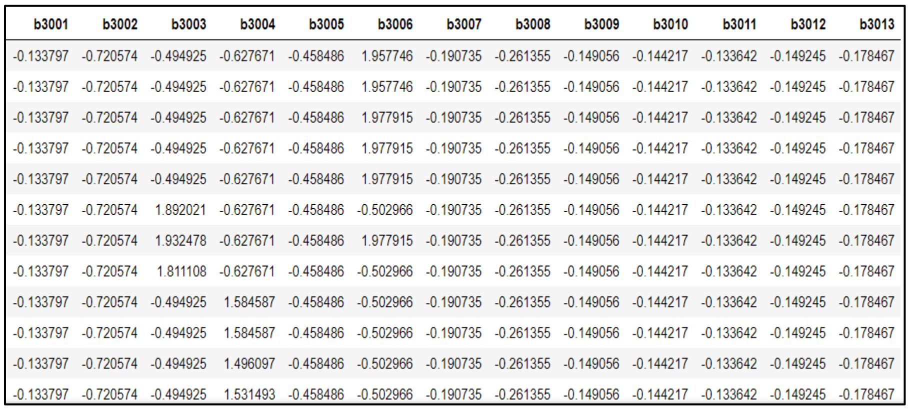
- The dataset originally contains 1420 data instances with no missing values
- The date column is dropped as it's doesn't affect the training process and has no role in predicting the locations.
- The RSSI values of the 13 input features are normalized by applying the Sklearn's StandardScalar() method to the label map. This method takes care of the outliers.



```
Data columns (total 15 columns):
location     1420 non-null object
date         1420 non-null object
b3001        1420 non-null int64
b3002        1420 non-null int64
b3003        1420 non-null int64
b3004        1420 non-null int64
b3005        1420 non-null int64
b3006        1420 non-null int64
b3007        1420 non-null int64
b3008        1420 non-null int64
b3009        1420 non-null int64
b3010        1420 non-null int64
b3011        1420 non-null int64
b3012        1420 non-null int64
b3013        1420 non-null int64
dtypes: int64(13), object(2)
```



b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-82	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-80	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-86	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-75	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-75	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-80	-200	-200	-200	-200	-200	-200	-200	-200	-200



b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.957746	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.957746	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.892021	-0.627671	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.932478	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.811108	-0.627671	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.584587	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.584587	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.496097	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.531493	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467

Related Work

This section discusses the original paper titled "*Semisupervised Deep Reinforcement Learning in Support of IoT and Smart City Services*", that used this dataset. The paper, proposes a semi supervised deep reinforcement learning (DRL) model that fits smart city applications. The proposed approach is applied in a smart campus project as part of a smart city. [5]

Method

The DRL framework acts as a learning mechanism in support of smart IoT services. The proposed model uses a small set of labeled data along with a larger set of unlabeled ones. This paper is the first attempt that extends the semisupervised reinforcement learning approach using DRL. The proposed model consists of a deep variational autoencoder network that learns the best policies for taking optimal actions by the agent.

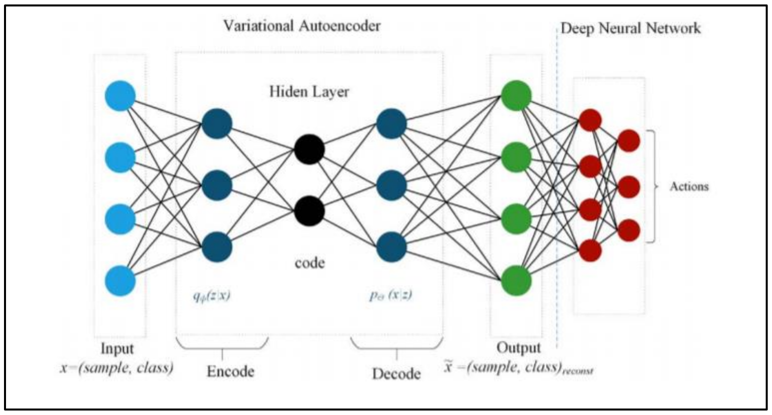


Fig. High-level concept of a variational autoencoder adopted for DRL. [5]

As a use case, the paper experimented with the proposed model in an indoor localization system. The system is represented as a set of positions that are labeled by row and column numbers. Each position is also associated with the set of RSSI values from the set of deployed iBeacons. The agent observes the environment by receiving RSSI values at each time. The design requires the agent to take action based on the three most recent RSSI observations.

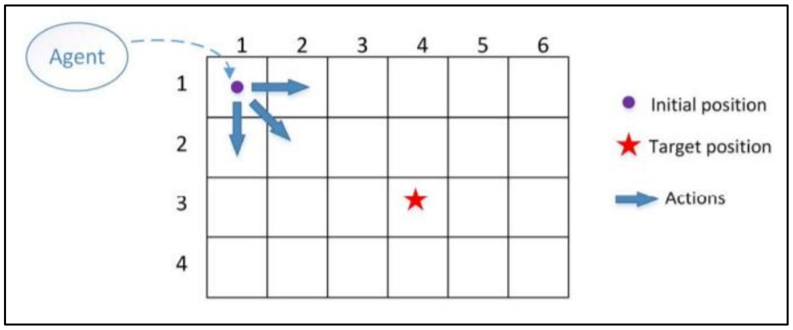


Fig. Illustration of a typical indoor environment for DRL. [5]

Results

The experimental results illustrate that the proposed semisupervised DRL model can generalize the positioning policy for configurations where the environment data is a mix of labeled and unlabeled data and achieve better results compared to using a set of only labeled data in a supervised model. The results show an improvement of 23% on the localization accuracy in the proposed semisupervised DRL model. Also, in terms of gaining rewards, the semisupervised model outperforms the supervised model by receiving at least 67% more rewards.

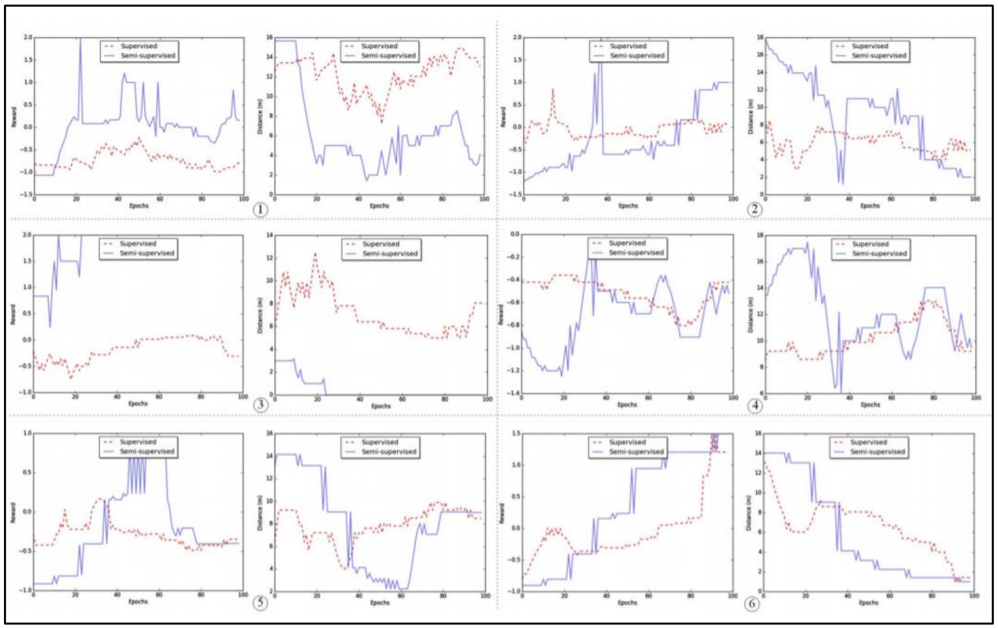


Fig. Obtaining rewards and distances in six episodes with a supervised and semisupervised DRL models. [5]

Conclusion

This paper shows that IoT applications in general, and smart city applications in specific where context-awareness is a valuable asset can benefit immensely from unlabeled data to improve the performance and accuracy of their learning agents. Furthermore, the semisupervised DRL is a good solution for many IoT applications since it requires little supervision by giving a rewarding feedback as it learns the best policy to choose among alternative actions.

Model Development

Classification Models

K-Nearest Neighbor

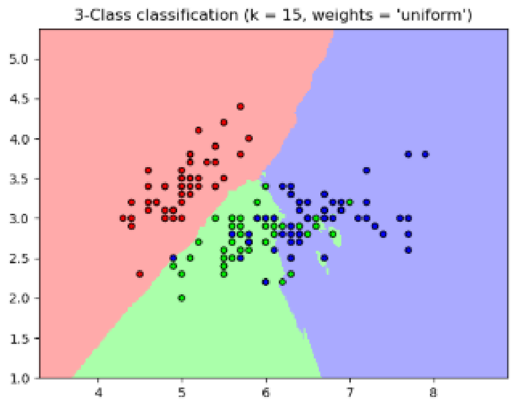
Description

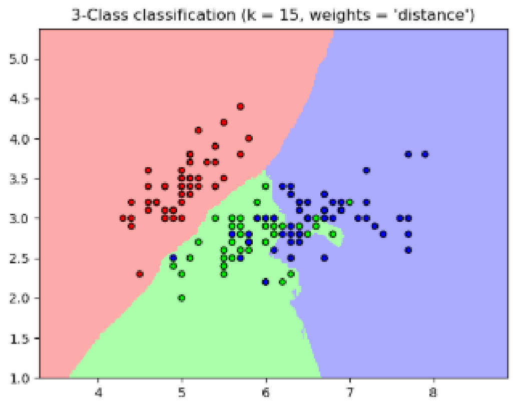
In pattern recognition, the **k-Nearest Neighbor's (k-NN)** algorithm is a non-parametric method used for classification and regression. [2] In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

Scikit-learn implements two different nearest neighbor's classifiers: **KNeighborsClassifier** implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. *RadiusNeighborsClassifier* implements learning based on the number of neighbors within a fixed radius r of each training point, where r is a floating-point value specified by the user.

The k-neighbor's classification in **KNeighborsClassifier** is the more commonly used of the two techniques. The optimal choice of the value k is highly data-dependent: in general, a larger k suppresses the effects of noise, but makes the classification boundaries less distinct.





Support Vector Machines

Description

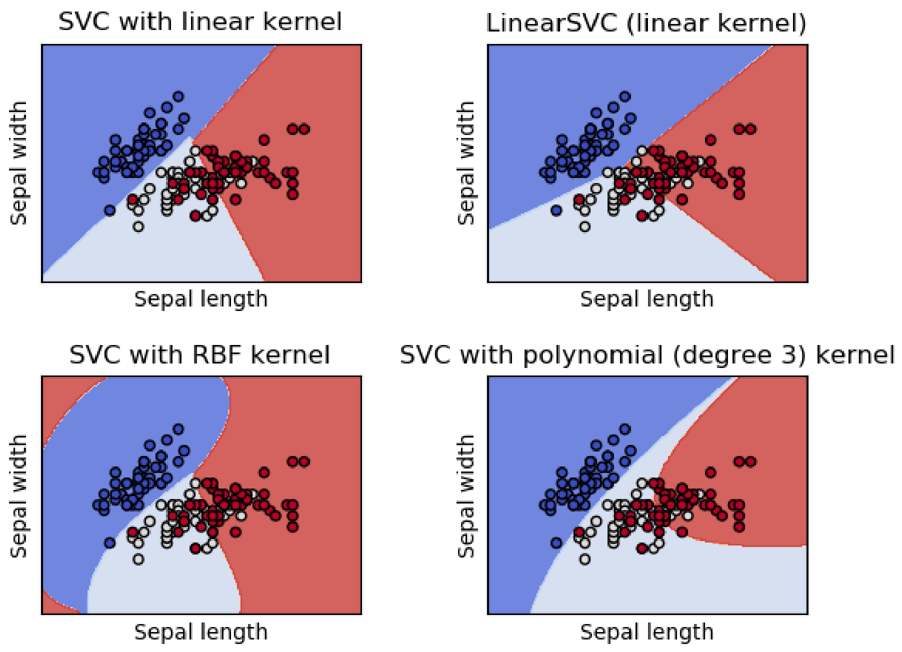
Support vector machines (**SVMs**) are a set of supervised learning methods used for classification, regression and outliers' detection. [3] The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

Scikit-learn implements 3 different support vector classifiers: **SVC,** *LinearSVC and NuSVC***.** The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data.



Random Forest

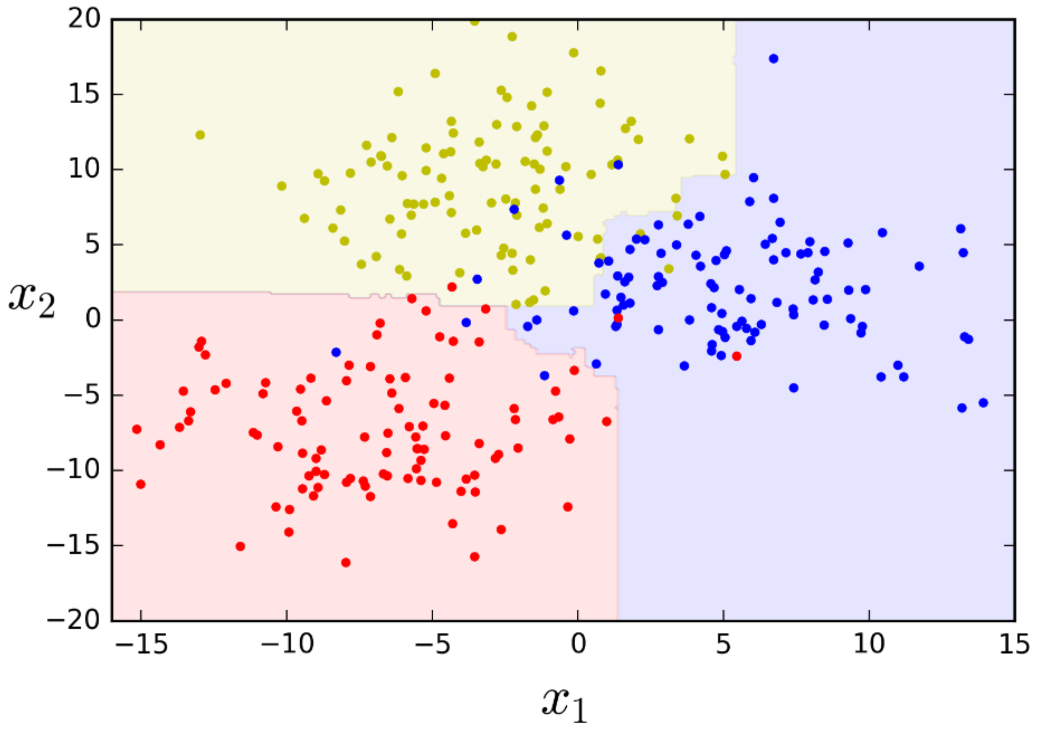
Description

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. [4] Random decision forests correct for decision trees' habit of overfitting to their training set.

In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. Because of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

Scikit-learn implements random forst using the **RandomForestClassifier**. The implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.



Data Pre-processing:

1. Loading the dataset in a pandas dataframe
2. Extracting the input features (13 *beacons*) from the dataset
3. Extracting the normalization parameters i.e. min, max and mean values
4. Normalizing the input features and then Encoding them
5. Extracting the output feature (*location*)
6. Encoding the output labels
7. Splitting the data using the train_test_split method
8. The tools and methods used for data pre-processing are:
   - pandas for data structuring
   - numpy for linear algebra (mostly dealing with matrices)
   - sklearn for encoding and data sampling
   - matplotlib for plotting quality figures and plots

Model Training:

1. *Defining* the classifier methods for all the 3 models (K-NN, SVM, Random Forest)
2. *Fitting* the training data (X_train, y_train) over the 3 classifiers
3. *Predicting* the test results using the trained classifier models
4. Calculating the models' *Accuracy*

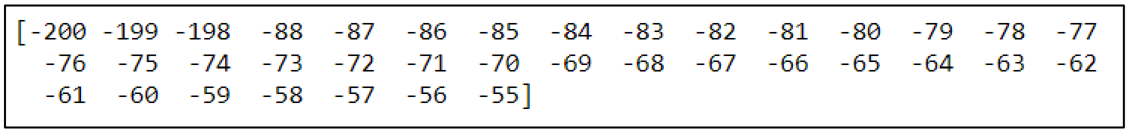
Fine-tuning Models & Feature Set

To improve *accuracy*, the classifier models were fine-tuned and the feature sets were optimized over successive iterations of the baseline model.

<u>Baseline Model</u>

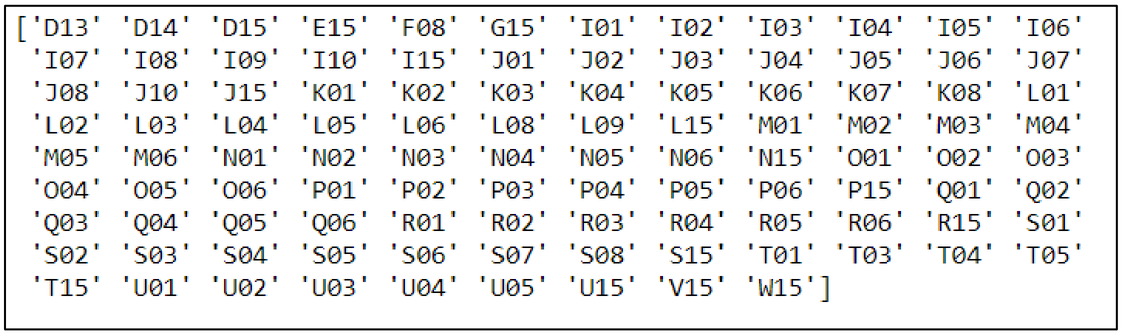
1. In the baseline model, raw values (without normalization or encoding) of the input and output feature set are used for training the classifier models
2. The dataset is loaded into a panda's data frame.
3. The *RSSI* values are used as the input features and *location* is used as the output feature.
4. Both the input and output features were extracted into separate data frames.
5. Scikit learns' train_test_split method is used to create training and test sets

6. The input data frame consists of 1420 instances (rows) of the RSSI values for the 13 beacons (columns) ranging from -200 to -55.



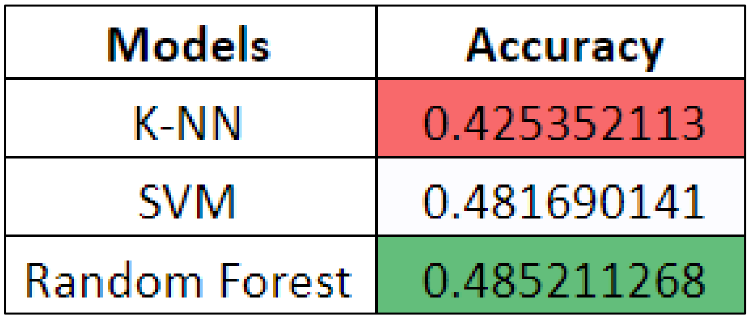
```
[-200 -199 -198  -88  -87  -86  -85  -84  -83  -82  -81  -80  -79  -78  -77
  -76  -75  -74  -73  -72  -71  -70  -69  -68  -67  -66  -65  -64  -63  -62
  -61  -60  -59  -58  -57  -56  -55]
```

7. The output feature consists of 1420 instances of locations (categories) and has 105 unique combinations of the x (A-Z) and y (1-18) coordinates.

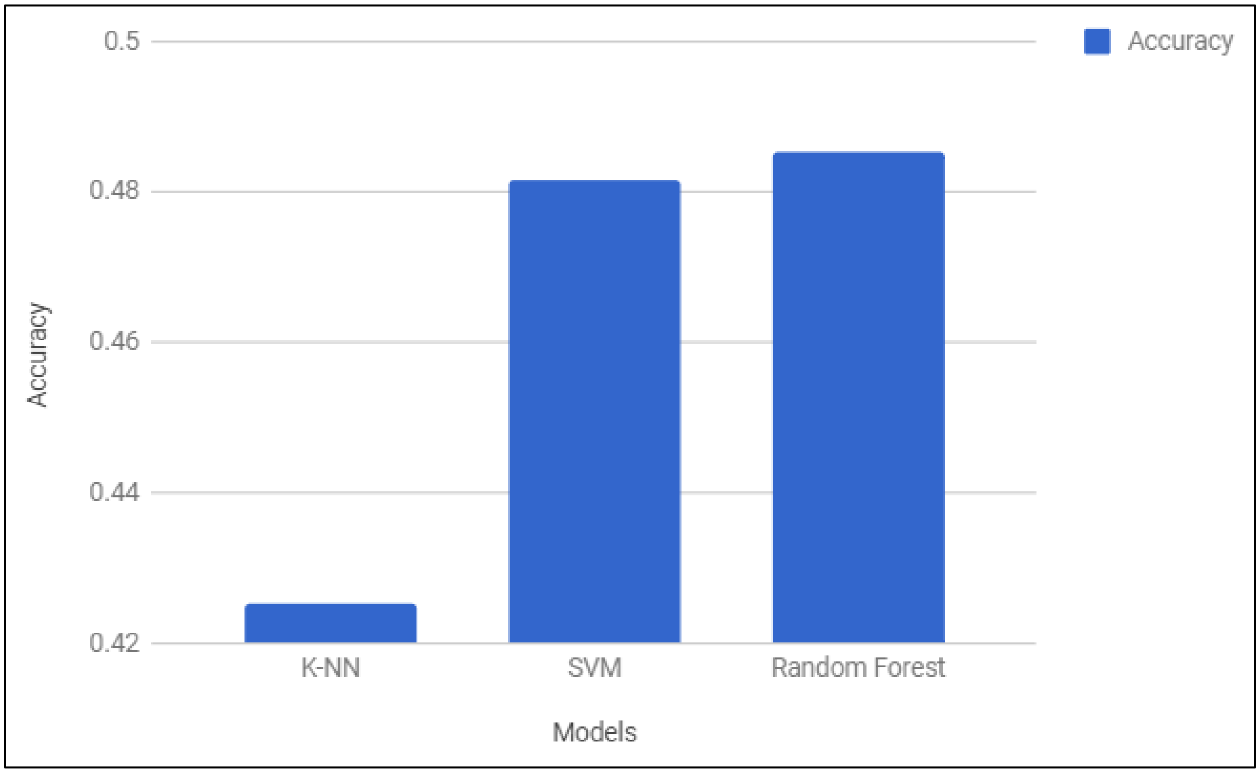


```
['D13' 'D14' 'D15' 'E15' 'F08' 'G15' 'I01' 'I02' 'I03' 'I04' 'I05' 'I06'
 'I07' 'I08' 'I09' 'I10' 'I15' 'J01' 'J02' 'J03' 'J04' 'J05' 'J06' 'J07'
 'J08' 'J10' 'J15' 'K01' 'K02' 'K03' 'K04' 'K05' 'K06' 'K07' 'K08' 'L01'
 'L02' 'L03' 'L04' 'L05' 'L06' 'L08' 'L09' 'L15' 'M01' 'M02' 'M03' 'M04'
 'M05' 'M06' 'N01' 'N02' 'N03' 'N04' 'N05' 'N06' 'N15' 'O01' 'O02' 'O03'
 'O04' 'O05' 'O06' 'P01' 'P02' 'P03' 'P04' 'P05' 'P06' 'P15' 'Q01' 'Q02'
 'Q03' 'Q04' 'Q05' 'Q06' 'R01' 'R02' 'R03' 'R04' 'R05' 'R06' 'R15' 'S01'
 'S02' 'S03' 'S04' 'S05' 'S06' 'S07' 'S08' 'S15' 'T01' 'T03' 'T04' 'T05'
 'T15' 'U01' 'U02' 'U03' 'U04' 'U05' 'U15' 'V15' 'W15']
```

8. The final test accuracy scores obtained by the 3 classifiers are plotted in the graph below
9. As you can see from the plots, all the 3 classifiers performed quite poorly.
10. Random Forests performed the best out of the 3 with an accuracy of **0.485**
11. Support Vector Classifier was just a little behind with an accuracy of **0.482**
12. K-Nearest Neighbour had the lowest accuracy (**0.425**) out of all the 3 classifiers.

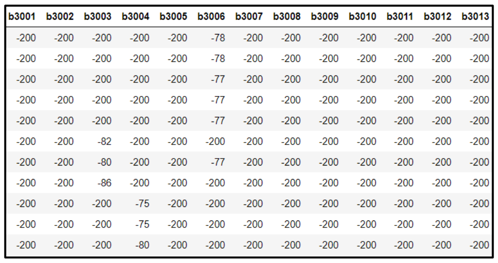


Models	Accuracy
K-NN	0.425352113
SVM	0.481690141
Random Forest	0.485211268

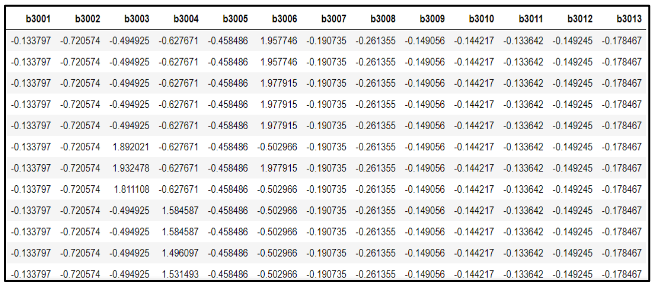


Iteration-1 Model

1. One of the reasons for the baseline model performing poorly was that the RSSI values in the 13 input features were not normalized and had a large variance.
2. Another reason was a large number (105) of output categories to predict.
3. The Iteration-1 model rectified these problems to improve the classifiers' accuracy
4. In the Iteration-1 model, the input features were normalized by applying Sklearn's StandardScalar() method to the label map.

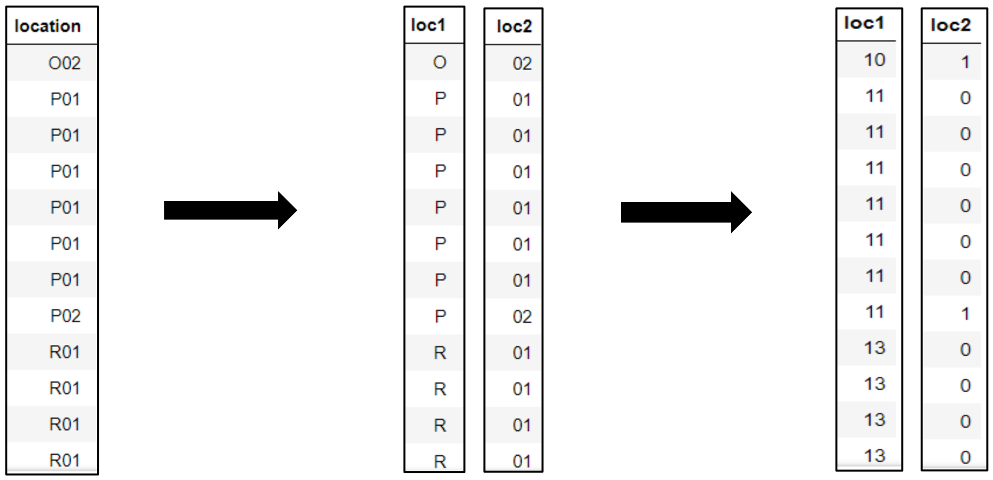


b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-78	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-82	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-80	-200	-200	-77	-200	-200	-200	-200	-200	-200	-200
-200	-200	-86	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-75	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-75	-200	-200	-200	-200	-200	-200	-200	-200	-200
-200	-200	-200	-80	-200	-200	-200	-200	-200	-200	-200	-200	-200



b3001	b3002	b3003	b3004	b3005	b3006	b3007	b3008	b3009	b3010	b3011	b3012	b3013
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.957746	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.957746	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.892021	-0.627671	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.932478	-0.627671	-0.458486	1.977915	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	1.811108	-0.627671	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.584587	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.584587	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.496097	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467
-0.133797	-0.720574	-0.494925	1.531493	-0.458486	-0.502966	-0.190735	-0.261355	-0.149056	-0.144217	-0.133642	-0.149245	-0.178467

5. The ouput feature is first split into 2 separate features (columns) for the x and y co-ordinates and then encoded by applying Sklearn's LabelEncoder() method to both the features separately.

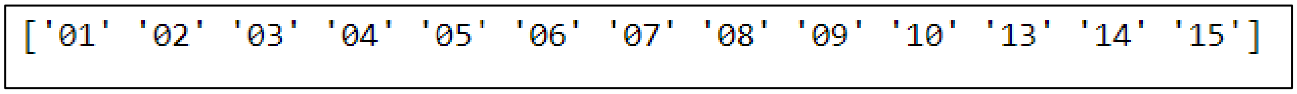


location
O02
P01
P01
P01
P01
P01
P01
P02
R01
R01
R01
R01

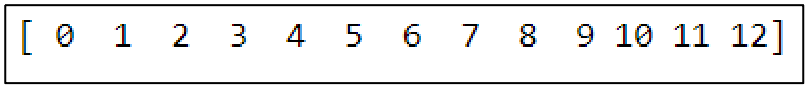
loc1	loc2
O	02
P	01
P	01
P	01
P	01
P	01
P	01
P	02
R	01
R	01
R	01
R	01

loc1	loc2
10	1
11	0
11	0
11	0
11	0
11	0
11	0
11	1
13	0
13	0
13	0
13	0

6. The x-coordinate (loc1) with 13 unique labels were encoded into 13 numerical values (0-12)

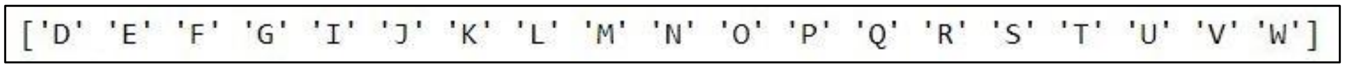


```
['01' '02' '03' '04' '05' '06' '07' '08' '09' '10' '13' '14' '15']
```

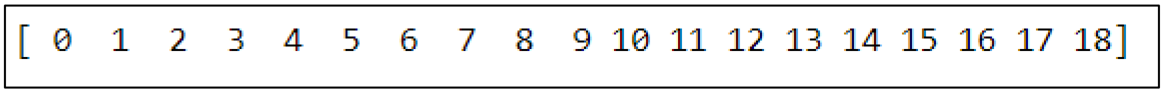


```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12]
```

7. The y-coordinate (loc2) with 19 unique labels were encoded into 19 numerical values (0-18)



```
['D' 'E' 'F' 'G' 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W']
```

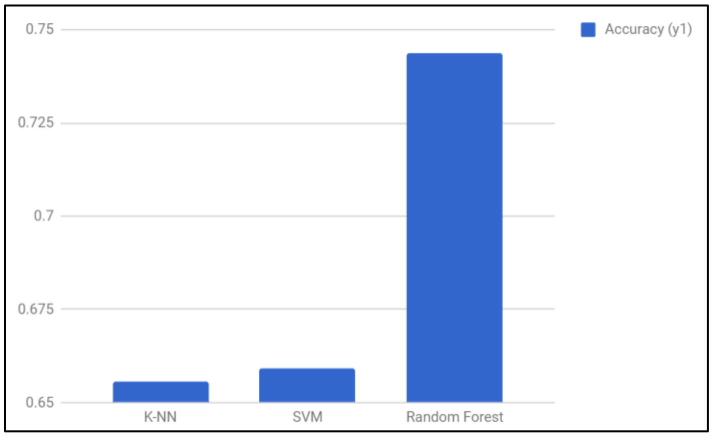


```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

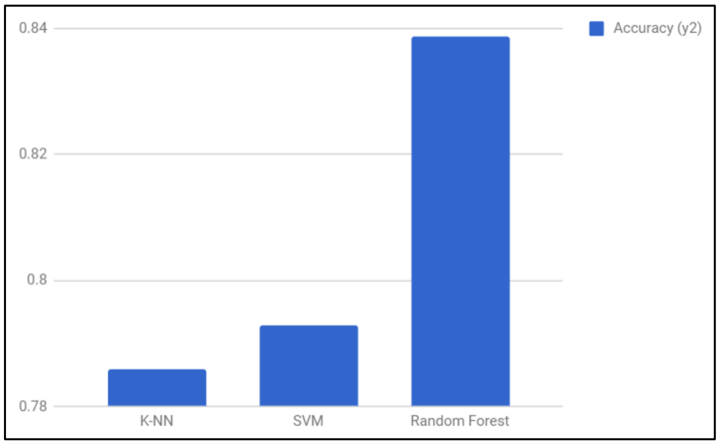
8. Thus, the 105 output categories are now converted and reduced to 2 different output labels with just 13 and 19 categories respectively
9. The classifiers are trained to predict the x-coordinate and y-coordinate separately.

10. The final test accuracy scores for the x-coordinate (y1) and y-coordinate (y2) obtained by the 3 classifiers are plotted in the graphs below.
11. As you can see from the plots, normalizing and encoding the dataset drastically improved the classifiers performance
12. Random Forest classifier clearly outperformed the K-NN and SVM classifiers for both the (y1 and y2) output labels.

Models	Accuracy (y1)
K-NN	0.6556338028
SVM	0.6591549296
Random Forest	0.7436619718

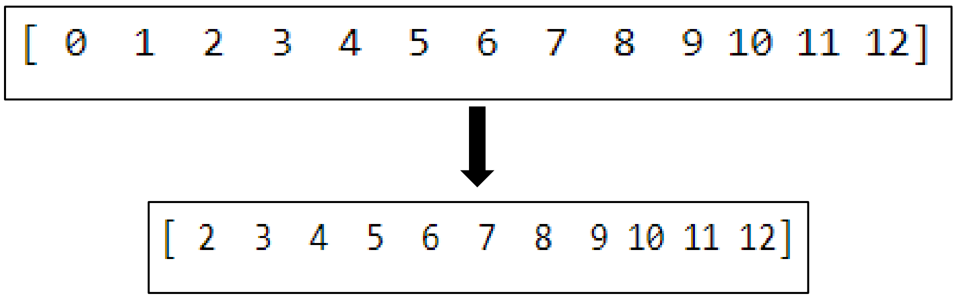


Models	Accuracy (y2)
K-NN	0.785915493
SVM	0.7929577465
Random Forest	0.8387323944



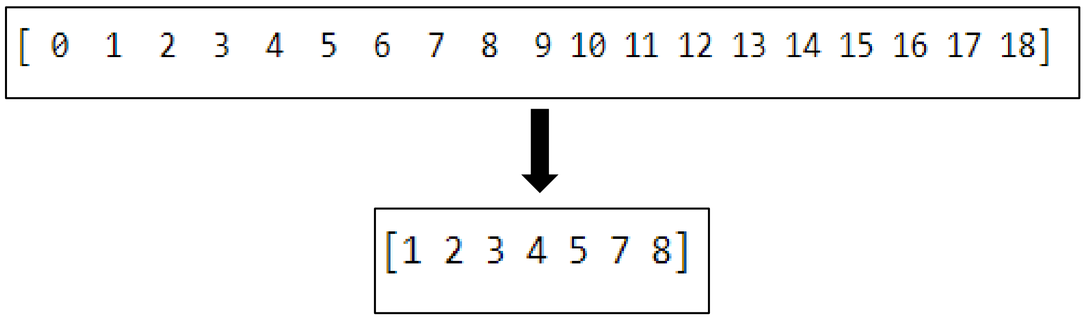
Iteration-2 Model

1. The Iteration-2 model was developed to further improve the accuracy of the classifier models.
2. In the Iteration-2 model, built on the Iteration-1 model.
3. The input features were normalized by applying Sklearn's StandardScalar() method to the label map.
4. A custom encoder function was defined to encode both the output labels y1 and y2.



```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12]
```

```
[ 2  3  4  5  6  7  8  9 10 11 12]
```



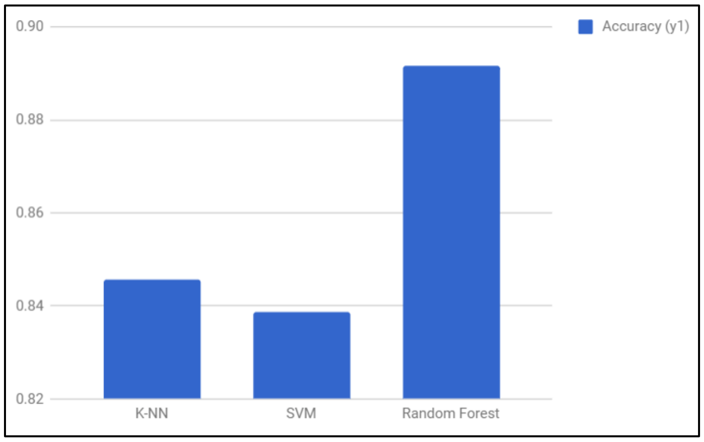
```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

```
[1 2 3 4 5 7 8]
```

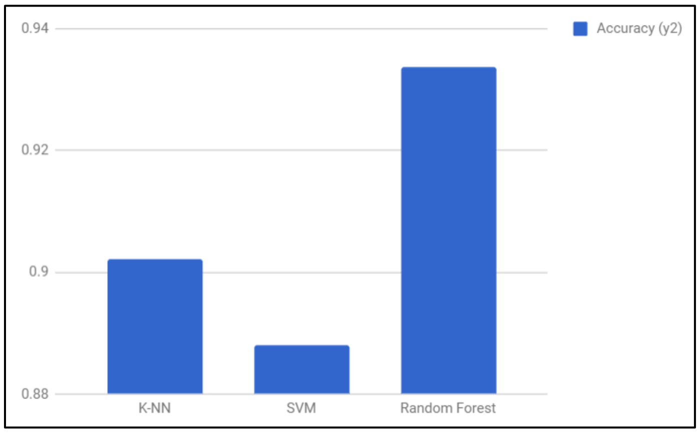
5. The encoder function optimized the labels to further decrease the total number of categories in these output labels
6. Thus the 13 and 19 categories in y1 and y2 output labels used in the previous model are now reduced to 11 and 8 respectively.

7. The final test accuracy scores for the x-coordinate (y1) and y-coordinate (y2) obtained by the 3 classifiers are plotted in the graphs below.
8. As you can from the plots, Random Forest gave the best performance again with an accuracy of **0.93**.
9. K-NN, with an accuracy of **0.90**, was able to classify the output labels better than SVM which got an accuracy of 0.88

Models	Accuracy (y1)
K-NN	0.8457746479
SVM	0.8387323944
Random Forest	0.8915492958



Models	Accuracy (y2)
K-NN	0.9021126761
SVM	0.888028169
Random Forest	0.9338028169



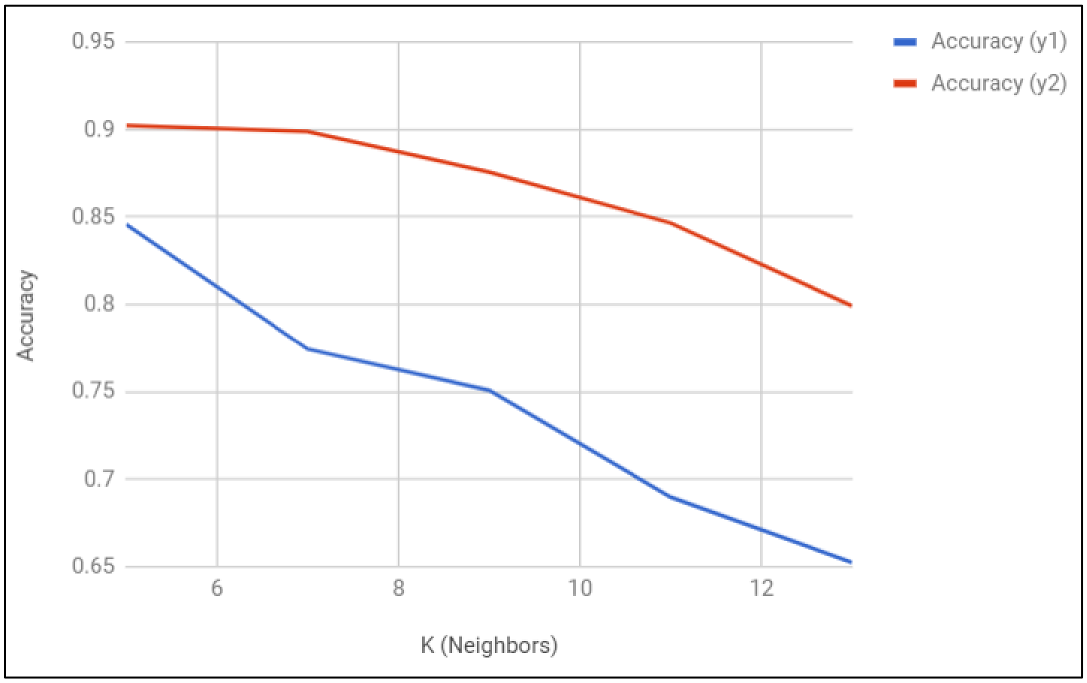
Performance

The performance of the models was measured by fine-tuning and varying their hyper-parameters

<u>K-NN</u>

- Accuracy was used to select the optimal value of K using the largest value.
- The Accuracy results were plotted as a function of K as shown in the graph below
- As you can see in the plots, the accuracy falls drastically as we increase the value of K
- The final value used for the model is K = 5

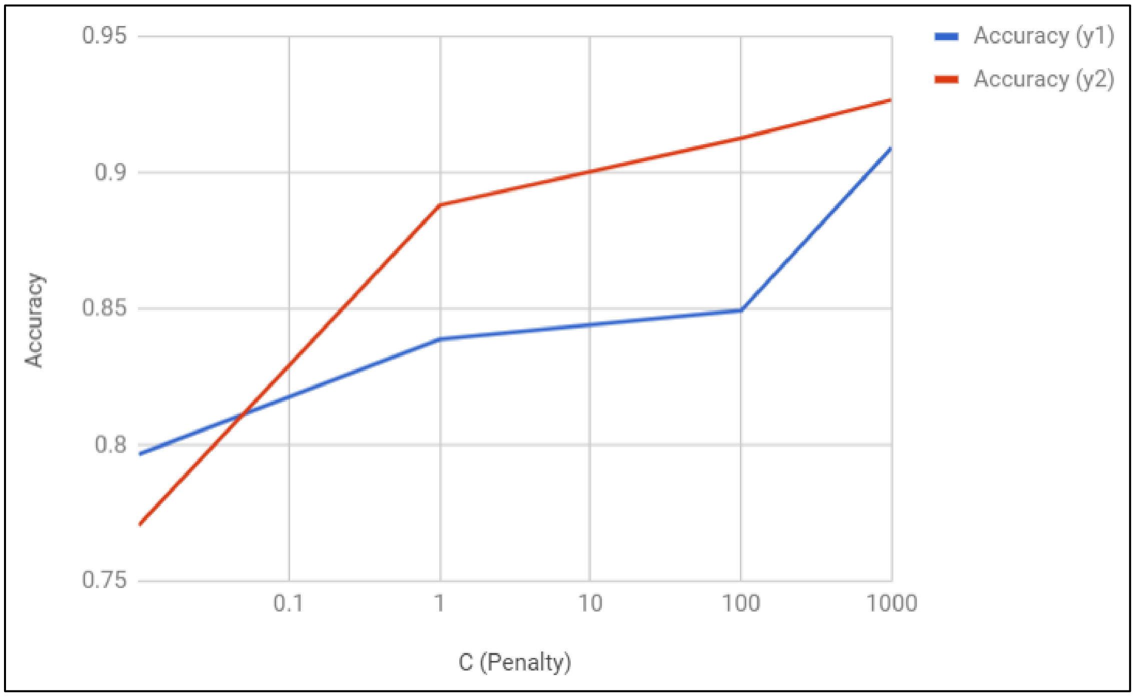
K	Accuracy (y1)	Accuracy (y2)
5	0.8457746479	0.9021126761
7	0.7744301123	0.8987323944
9	0.7507843626	0.8755311435
11	0.6898875633	0.8464791231
13	0.6525709996	0.7992176324



SVM

- Accuracy was used to select the optimal value of C which is the Penalty parameter.
- The Accuracy results were plotted as a function of C as shown in the graph below
- As you can see in the plots, the accuracy improves as we increase the value of C.
- The final value used for the model is C = 1000

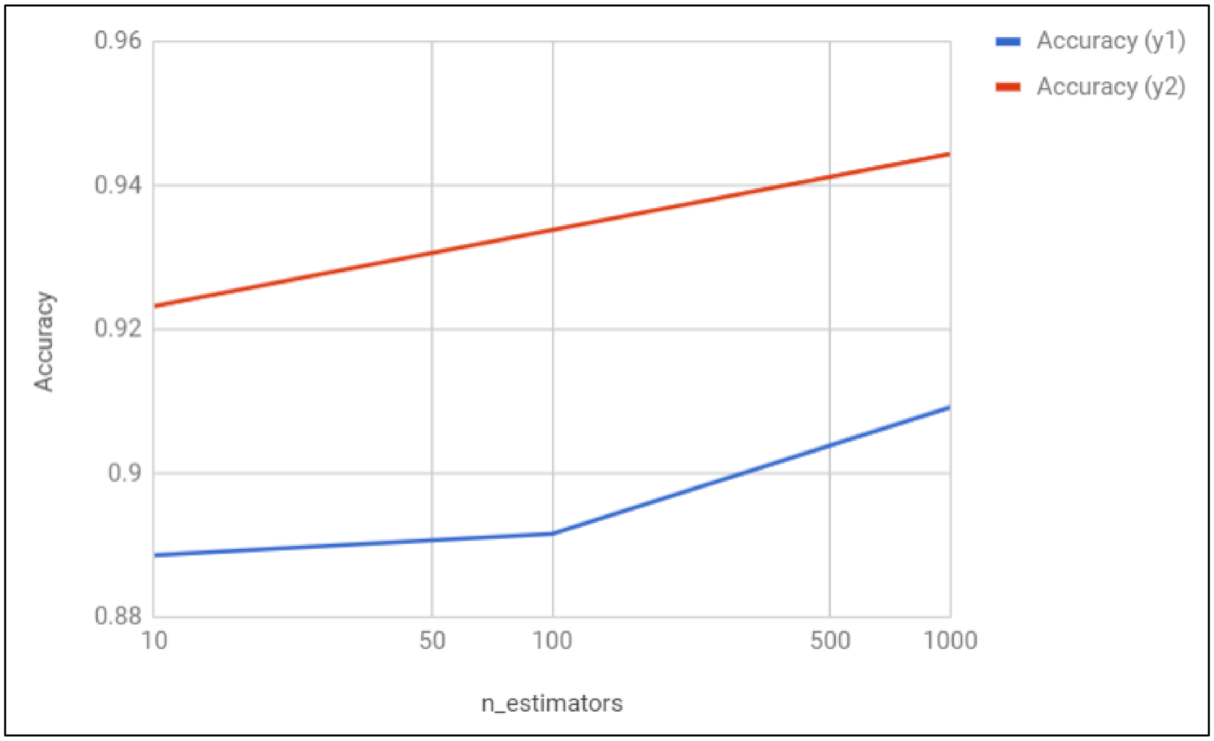
C	Accuracy (y1)	Accuracy (y2)
0.01	0.7964788732	0.7704225352
1	0.8387323944	0.8880226761
100	0.8492957746	0.9126760563
1000	0.9091549296	0.9267605634



Random Forest

- Accuracy was used to select the optimal value of n_estimators which is the number of trees used in the forest.
- The Accuracy results were plotted as a function of n_estimators as shown in the graph below
- As you can see in the plots, the accuracy improves as we increase the value of n_estimators.
- The final value used for the model is C = 1000

n_estimators	Accuracy (y1)	Accuracy (y2)
10	0.8885915493	0.9232394366
100	0.8915492958	0.9338028169
1000	0.9091549296	0.9443661972



Discussions & Conclusions

This project is used study a classification problem using a dataset in the UCI Machine Learning Repository and demonstrate the use of various classification models on the dataset.

The dataset used for this project is the *BLE RSSI Dataset for Indoor localization and Navigation* found on the website (https://archive.ics.uci.edu/ml/datasets). This is a *Multivariate* classification problem. The dataset contains RSSI readings gathered from an array of Bluetooth Low Energy (BLE) iBeacons in a real-world and operational indoor environment for localization and navigation purposes.

The 3 classification models chosen for this project are K-Nearest Neighbors (K-NN), Support Vector Machines (SVM) and Random Forests (RF).

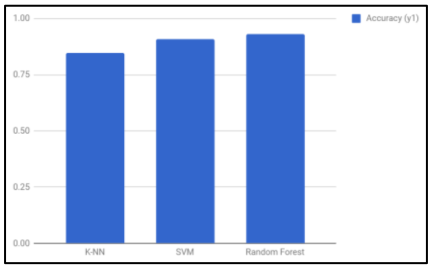
The project uses 3 iterative versions of the model architecture on which the 3 classifiers are trained. A comparative study of all the 3 architectures is shown in the table below:

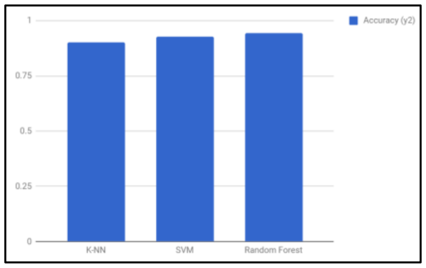
Model	Train:Test Split	Input Features	Output Labels	Feature Normalization	Label Encoding	Label Optimization
Baseline Model	**80:20**	**13**	**1**	**No**	**No**	**No**
Iteration-1 Model	**80:20**	**13**	**2**	**Yes**	**Yes**	**No**
Iteration-2 Model	**80:20**	**13**	**2**	**Yes**	**Yes**	**Yes**

Based on the final Accuracy results of the 3 classifier models, it can be concluded that random Forests gave the best performance for this classification problem as shown in the graphs below:

Models	Accuracy (y1)
K-NN	0.8457746479
SVM	0.9091549296
Random Forest	0.9291749296

Models	Accuracy (y2)
K-NN	0.9021126761
SVM	0.9267605634
Random Forest	0.9443661972





References

[1] "UCI Machine Learning Repository: BLE RSSI Dataset for Indoor Localization and Navigation Data Set." Archive.ics.uci.edu. N. p., 2018. Web. 9 May 2018.
[2] "K-Nearest Neighbors Algorithm | Wikiwand." Wikiwand. N. p., 2018. Web. 11 May 2018.
[3] "1.4. Support Vector Machines - Scikit-Learn 0.19.1 Documentation." Scikit-learn.org. N. p., 2018. Web. 11 May 2018.
[4] "Random Forest | Wikiwand." Wikiwand. N. p., 2018. Web. 11 May 2018.
[5] M. Mohammadi et al., "Semi-Supervised Deep Reinforcement Learning in Support of IoT and Smart City Services", IEEE Internet of Things J., pp. 1-12, 2017.