# Comparison of Classification Models and their impact on Indoor Wireless Localization dataset

Data Set: Indoor Wireless Localization
Author: Mana Mehraein,
Email: mehraein@usc.edu
May 2020

## 1. Abstract

For this project, "Indoor Wireless Localization" dataset from UCI was selected. The dataset has the data collected from 7 different wifi routers in 4 different locations. The data is the signal strength of each wifi router visible on a smartphone. [1] Each of the columns (features) are independent since they are the signal strength of that individual wifi-router. This is a classification problem. In order to find the best classification model, ten different models were tested to train the model. Most of these models were selected since assumed they should yield good accuracy results. Models such as SVM and Random Forest are among those models (97.25% and 98.25% respectively). The other reason to select some of the models such as linear regression model was to show that not all good and popular models can be used for classifications. ($R^2$ 78.40%). The attempt was to use almost all the classification models taught in this class, and see their impact on the training of thiss dataset. It was interesting to see that KNearestNeighbor classifier yielded the best test accuracy result of 98.75%. Also, it was interesting to see that this dataset worked not that good with models that used linear decision boundaries. The report goes into more detail as how each of classifiers worked and compares the result of each one to the others. Also the reason why some classifiers work better compare to the rest are explained in more detail in the report.

## 2. Introduction

### 2.1. Problem Statement and Goals

The dataset selected for this project is "Indoor Wireless Localization" from UCI. According to UCI, the dataset is collected in indoor spaces by observing signal strengths of seven wifi signals visible on a smartphone. [1]
Different classification methods used in Mathematical Pattern recognition class was applied on the dataset, and different reasons why the classification method worked or did not work on the dataset was analyzed.

## 3. Approach and Implementation

To start, the dataset was read and analyzed. Different methods for preprocessing, feature selection, and engineering were tested. Training, validation, and test datasets

were set. Different classification methods were tested on data. Accuracy score, precision, and recall were calculated for each method. For some of the classifiers, GridSearchCV method was used to find the best possible parameter selection. Confusion matrix was also plotted for each method to find possible sources of error. At the end the best classifier based on the performance on our dataset was selected. The final plot of classified data was also plotted. More detailed for each process are available in the following sections.
For this project, Python was used. The major libraries used were panda, numpy, and scikit. Other than these libraries, seaborn was used for mapping confusion matrices, itertool, scipy, matplotlib were also used for different purposes. Mlxtend library was also used for final plotting. [2]

## 3.1. Preprocessing

First, the data was read from csv file. Simple command of "data.head()" gave us an understanding of the data. There were 7 columns of data for different signals fetched from the wireless routers, and one column for location which showed which room resulted in which reading from each of the wifi routers. The Location column was separated and named as train_label, and the rest were used as dataset.
describe() command gave the general view of the dataset, number of data in each column, means, min, max, and standard deviation. Also 25, 50, and 75 percentiles were calculated for each column.

| | Location | WS1 | WS2 | WS3 | WS4 | WS5 | WS6 | WS7 |
|---|---|---|---|---|---|---|---|---|
| count | 1600.000000 | 1600.000000 | 1600.000000 | 1600.000000 | 1600.000000 | 1600.000000 | 1600.000000 | 1600.000000 |
| mean | 2.500000 | -52.325000 | -55.589375 | -54.960000 | -53.494375 | -62.671250 | -80.994375 | -81.691250 |
| std | 1.118384 | 11.351317 | 3.420793 | 5.279993 | 11.519729 | 9.073223 | 6.508038 | 6.525434 |
| min | 1.000000 | -74.000000 | -74.000000 | -73.000000 | -77.000000 | -86.000000 | -97.000000 | -98.000000 |
| 25% | 1.750000 | -61.000000 | -58.000000 | -58.000000 | -63.000000 | -69.000000 | -86.000000 | -86.000000 |
| 50% | 2.500000 | -55.000000 | -56.000000 | -55.000000 | -56.000000 | -64.000000 | -82.000000 | -83.000000 |
| 75% | 3.250000 | -46.000000 | -53.000000 | -51.000000 | -46.000000 | -56.000000 | -77.000000 | -78.000000 |
| max | 4.000000 | -10.000000 | -45.000000 | -42.000000 | -13.000000 | -39.000000 | -62.000000 | -64.000000 |

This shows all columns have values and there is no missing value.
Next standardization was done on the data. In general, in order to make the data standard, the mean and standard deviation of the features which are normally columns are calculated. The mean will be subtracted from each value and then divided by the standard deviation. This means all numeric features will be centered around a mean of 0 and then expressed in terms of standard deviations. This process will make the model more robust and makes the features comparable to each other.
The description of data changed as follow:

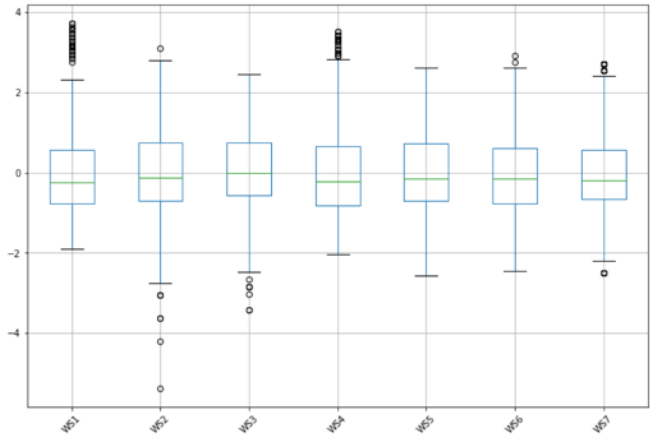
| | WS1 | WS2 | WS3 | WS4 | WS5 | WS6 | WS7 |
|---|---|---|---|---|---|---|---|
| count | 1.600000e+03 | 1.600000e+03 | 1.600000e+03 | 1.600000e+03 | 1.600000e+03 | 1.600000e+03 | 1.600000e+03 |
| mean | -3.981988e-15 | -1.524926e-15 | -2.430001e-16 | -1.106754e-16 | 1.353084e-15 | 5.555278e-16 | -8.398837e-16 |
| std | 1.000313e+00 | 1.000313e+00 | 1.000313e+00 | 1.000313e+00 | 1.000313e+00 | 1.000313e+00 | 1.000313e+00 |
| min | -1.910067e+00 | -5.383658e+00 | -3.417740e+00 | -2.041105e+00 | -2.571968e+00 | -2.460131e+00 | -2.500041e+00 |
| 25% | -7.644674e-01 | -7.049180e-01 | -5.759384e-01 | -8.254184e-01 | -6.977376e-01 | -7.693854e-01 | -6.605074e-01 |
| 50% | -2.357291e-01 | -1.200755e-01 | -7.578136e-03 | -2.175753e-01 | -1.464932e-01 | -1.545688e-01 | -2.006241e-01 |
| 75% | 5.573783e-01 | 7.571883e-01 | 7.502355e-01 | 6.507721e-01 | 7.354978e-01 | 6.139521e-01 | 5.658481e-01 |
| max | 3.729808e+00 | 3.096558e+00 | 2.455316e+00 | 3.516318e+00 | 2.609729e+00 | 2.919515e+00 | 2.711970e+00 |

Looking at the means, they are all almost zero, and standard deviations are almost 1. This means the data was successfully standardized.

## 3.2. Feature engineering (if applicable)

For checking the need for feature engineering on data, first was checked to make sure there are no null values or NAN values in the dataset, which the answer was negative. All the columns had int64 datatype which meant there was no need to fix the datatypes. Also, since all the data was integer meant there was no need for one hot encoding or use of other methods.

In order to check the distribution of the data, the boxplot was plotted as the result is as follow:




This shows there are no significant outliers. We will double check this by calculating the skews. Normally distributed data should have skewness close to zero. If the result of skew was >0, it means it has more weight in the left tail and if it was <0 means more weight on the right. The result for each of Wifi router datasets are as follow:

```
skew1: 1.0616418498661164
skew2: -0.2636070641830l216
skew3: -0.2588857304866265
skew4: 0.846194491503386
skew5: 0.3942235277409816
skew6: 0.43629642032775834
skew7: 0.48893599953670425
```

This result shows there might be some outliers in wifi_1 data set. In order to fix that, the data quantile for 10% and 90% were calculated. Any data higher than 90% was substituted with the value for 90% and any data smaller than

10% was substituted with the value for 10 percentiles. This is known as Quantile-based Flooring and Capping. The new skew results are as follow:

```
skew1: 0.399051057209738
skew2: -0.26360706418301216
skew3: -0.2588857304866265
skew4: 0.846194491503386
skew5: 0.3942235277409816
skew6: 0.43629642032775834
skew7: 0.48893599953670425
```

It is interesting to know that the classification accuracies resulting from this outlier fix was resulted in lower accuracy in all classification models. (PDF copy of the Jupyter notebook file with all the result is attached and will be submitted for possible review). Therefore, the dataset **without outlier fix** was used in this report. Since the data was collected from 4 different rooms, it is possible that as mentioned in the project dataset description, the geometry of the rooms may not have been symmetric, and as a result some data might seem to be outliers, but actually in this example it is adding value to the system.

## 3.3. Feature dimensionality adjustment

Since the dataset in this experiment is not a huge dataset, reducing the data might not be a really good idea. The more the data, the easier for the algorithm to detect patterns of the data. But at the same time, if the data is a confusing data, putting a threshold and removing the less significant data might help the machine learning process. We tested this hypothesis.

### 3.3.1 PCA

For feature dimensionality adjustment, PCA (Principal Component Analysis) and Random Forest were used to check the impact on the accuracy.

PCA is a linear dimensionality reduction method that uses singular value decomposition (SVD) to project data to a lowed dimensional space. Two orthogonal lines will be used for this method. The first one would be the one that fits the data on a line with more space between the data. The second line is orthogonal to the first line and data might have less space between them on the second line. Therefore the first line is the best choice, and the second one is the second best choice.

PCA was used with Random forest to check the impact of feature dimension reduction on the data. Using Random Forest by itself, resulted in test accuracy (on validation data) of 98% and misclassification of 5 out of 320 validation data points. By adding PCA, the accuracy was dropped to 92.5% and 24 out of 320 points were misclassified. For this problem, even if the wifi signal is a small value, still is significant in helping the model to learn the boundary of the location. Therefore removing those values will hurt the model.

As a result, data dimension reduction was not used for most of the classification.

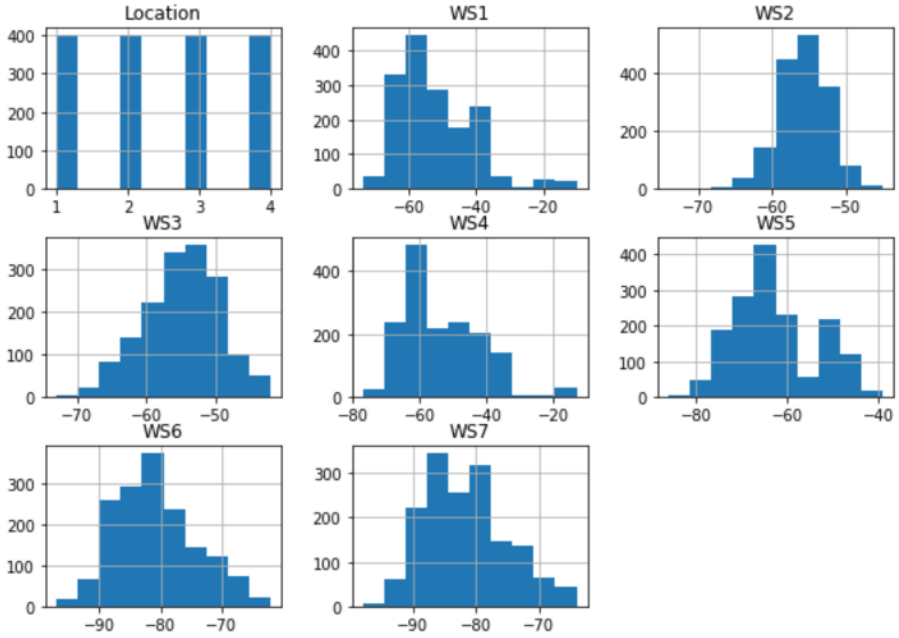
### 3.3.2. LDA

Linear Discriminant Analysis is another method that apart from classification, is used for dimension reduction as well. LDA is very similar to PCA. The

difference is that LDA chooses axis to minimize the distance between points in different categories. LDA reduces the dimension to minimum C-1. It was interesting to see that changing n_components which was used for feature reduction, did not change accuracy rate. Later on, this method will be used for classification.

## 3.4. Dataset Usage

As mentioned in previous section, first the training dataset was read. Then using head() and describe() it was analyzed to see if any data manipulations was needed. Since there was no missing data, it was checked to see if there are any null or NAN values, which the answer was negative. Next the data type was checked, and all the data were int64. This showed the data was ready to be used. histogram of the data was plotted to check the distribution of the data.




The first column which is 'Location' was separated to be training label. The rest of data was used for training data. As can be seen from the histograms, it seems the data has almost gaussian distribution. This was later used to choose the right form of classifiers (like Gaussian Naïve Bayes classifier).

Training data originally had (1600,7) shape, meaning 1600 rows and 7 columns. The data was split using train_test_split function. Test_size was set to be 0.2 which means 20% of training data was used for validation. By this, training data was set to be (1280,7) and validation data (320,7).

Next the data was standardized.

Using boxplot, the plot further showed the distribution of the data. Using skew() function, the skew for each feature (column) was calculated to find out the possible need of dealing with outliers..

The code to process test data was also written, but it was not used until the model was fully trained. At the end it was changed from 'Markdown' to

'Code' on Jupyter notebook to calculate the test accuracy for different classifiers.

In order to do cross validation, some different approaches were used.

GridSearchCV from scikit library searches over parameter values selected to find the best value for the hyperparameters. It was set so that it does 3 fold validation and check the accuracy rate to give the best accuracy rate with less possibility for overfitting. GridSearchCV was used for finding the best values for hyperparameters for some of the classifiers such as SVM, Random Forest, and Gradient Boosting Regressor.

Apart from that, cross_validation() function with cv=3 meaning 3 fold cross validation was used for the other classifiers.

## 3.5. Training and Classification

Different classifiers used to train the model. Following each of the classifiers used for this project will be presented.

### 3.5.1. Naïve Bayes

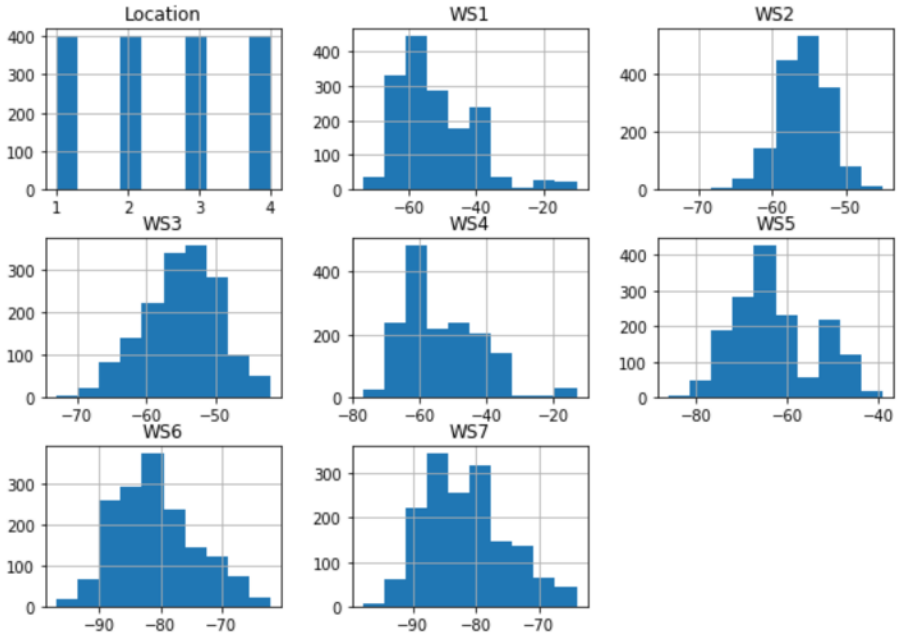
Naïve Bayes was used as the baseline system.

Naïve bayes is a supervised machine learning model that uses theorem of conditional probabilities of Bayes for classification.

It is called Naïve because it considers the features or columns to be strongly independent. It also gives the same weight to each feature. Therefore, if the features are actually independent, Naïve Bayes could work well.

In Naïve Bayes model, the probability of given set of input for all values of the class will be calculated and then the output with maximum probability will be selected. This method is simple and very fast.

As previously mentioned, the histogram of data is as follow:




Looking at the result, it looks very much like gaussian, therefore Gaussian Naïve bayes was used to analyze this data. There are no parameters used for this model.

### 3.5.2. SVM

SVM or support vector machine is one of the powerful classification models. SVM is supervised learning. SVM uses a hyperplane to classify the data points. SVM's job is to find a hyperplane that best separates points in a hypercube. SVM works best if the data is linearly separable. SVM tries to fit the widest possible margin between the nearest points of two different class. By setting up hyperparameter C, we can penalize the model for outliers. Smaller C's have less penalties for outliers which make them soft margins, and higher C's have higher penalties for outliers making them hard margins. With setting up C too high, the model will try to bend the decision boundaries and extend boundary which might cause overfitting problems. Apart from C, we can also choose different gamma points. According to scikit-learn documentations gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. [3]
SVC function uses RBF (Radial Basis Function) kernel by default.
Scikit learn library, has different functions and methods to calculate SVM. We will try two different functions to see the effect of them on the dataset.

#### 3.5.2.1. SVC()

SVM algorithm as 'SVC()' command uses one vs. one approach.
Using GridSearchCV with 3fold validation, C=5 and Gamma=1 gave the best accuracy for SVC function using training and validation sets.

#### 3.5.2.2. LinearSVC()

Linear SVC() function uses One vs. All approach.
With setting C=5, max_iteration to 1000 and tolerance = 1e-3, we noticed that SVC gives slightly better result. Max_iteration is the number of iterations that this model should train, and tolerance refers to the stopping criterion for model training. This means the training will be stopped when loss in that iteration is less than the tolerance value specified.

### 3.5.3. LDA

Linear Discriminant Analysis tries to separate classes using axes in a way that all instances of a class be in the same quadrant. LDA is used to find a linear combination of features that characterizes or separate two or more classes of objects or events. [4]
As is obvious from the name, LDA uses linear decision boundary.
LDA uses SVD as a solver.

### 3.5.4. QDA

Quadratic Discriminant Analysis is another supervised learning models and works similar to LDA. The difference is that QDA uses quadratic decision boundaries.
QDA too finds axes to best separate the classes so that all instances of a class are in the same quadrant.

### 3.5.5. SGD

Stochastic Gradient Descent is another classification model that uses supervised learning. SGD uses Cross entropy to measure how well the estimated probabilities match the actual labels. SGD will try to minimize the error by trying to minimize the cross entropy. SGD uses gradient descent and will start at some values of w and b (as weight and bias). It will iteratively change the value and try to make it better to finally be able to reach minima or maxima. SGD uses optimization algorithm to have the least loss.

For this section, max_iteration of 10,000 was selected and the tolerance is 1e-3. As mentioned before, Max_iteration is the number of iterations that this model should train, and tolerance refers to the stopping criterion for model training. This means the training will be stopped when loss in that iteration is less than the tolerance value specified.

### 3.5.6. KNN

KNN is K-nearest Neighbor classifierr. KNN classifies each point in the new test data based on the nearest neighbors of that data and the labels they have. So, for example if the test data point was among data points with most of them with label class 2, it will classify the point as class 2.

In the KNeighborsClassifier(), k is n_neighbors. For this part, GridSearchCV identified n_neighbors=3 as best value for this parameter.

### 3.5.7. RF

Random Forest classifier is a combination of decision trees and is considered supervised learning. This method is computationally efficient and is very fast. Random forest uses multiple trees checking different features and each of them are trained recursively. The feature tree with the most information gain will be selected and divided into left and right subtrees. The training set of each of the trees are randomly sampled from the training set. Those samples have the same size as the sample root. Each tree would be trained with the best threshold depending on the features. This process will be done for all the trees. At the end, the result information gain from all the trees will be calculated and the most voted result which gives the greatest information gain would be selected. Having multiple trees will help the system prevent the overfitting problem.

In general RF tends to become overfit, for instance it gave accuracy score of 100% for training data. Therefore, GridSearchCV was used to calculate the best values. N_estimators which is the number of trees, was set to be 100, and max_depth which is maximum depth of the tree was set to be 30.

### 3.5.8. Linear Regression and Logistic Regression

Linear regression is not a good model for classification problems. This model is mostly used to find the relationship between variable and forecasting. It is used to find a linear relationship between input and output (data and label).

One of the reasons Linear regression does not work is that the predicted value is continuous, but for classification problems that is not the case. It should say if the data belongs to a class or not.

Instead, Logistic regression is used for classification problems and uses a probability range between 0 to 1 for prediction. In multiclass cases, LogisticRegression uses one vs. rest model. The penalty for misclassification for this model is set to be L2 norm with C=1.

To compare these two models and their impact on our data, both were used on the model and the results will be delivered in next section.

### 3.5.9. Gradient Boosting

Gradient boosting model is a decision tree model which uses an ensemble of weak prediction models for prediction.

The parameters used for this model are num_estimators, max_depth, and learning_rate. num_estimators and max_depth are the same as the ones for Random Forest. Learning rate shrinks the contribution of each other by the value selected for learning_rate.

GridSearchCV was used to find the best values and they are as follow:

```
{'learning_rate': 0.01, 'max_depth': 4, 'n_estimators': 500}
```

## 4. Analysis: Comparison of Results, Interpretation

If a random classifier prediction was used for a 4-class model which predicts the values randomly, it would give a ¼ or 25% accuracy for our model. If the accuracy rate of each of the models are similar to this value, it means it did not learn anything. Following is the result of accuracy rates for training, validation and testing datasets.

It is worth mentioning that while dividing training set to training and validation sets, random selection between groups was disabled and 80% was divided to be training set and 20% as validation set.

| Classifier Model | Training_Accurracy | Validation_Accuracy | # of misclassified points (out of 320) | Final Testing_Accuracy | # of misclassified points (out of 400) |
|---|---|---|---|---|---|
| Naïve Bayes | 98.52 | 98.44 | 5 | 97.75 | 9 |
| SVM(SVC) | 99.94 | 98.13 | 4 | 97.25 | 11 |
| SVM(LinearSVC) | 97.81 | 97.81 | 7 | 97 | 12 |
| LDA | 97.27 | 97.19 | 9 | 97 | 12 |
| QDA | 98.13 | 97.81 | 7 | 98.5 | 6 |
| SGD | 97.5 | 97.8 | 7 | 95.75 | 17 |
| KNN | 99.06 | 98.44 | 5 | 98.75 | 5 |
| RF | 100 | 98.44 | 5 | 98.25 | 7 |
| LinearRegression (R^2) | 77.93 | 78.77 | NA | 78.4 | NA |
| LogisticRegression | 97.06 | 95.31 | 15 | 96.5 | 14 |
| GradientBoostingRegression | 99.86 | 97.31 | NA | 96.82 | NA |
| RF with PCA | 100 | 92.5 | 24 | 88.75 | 45 |

Following, each of the classifiers will be separately analyzed:

### 4.1. Naïve Bayes

**Accuracy results and # of mislabeled points:**

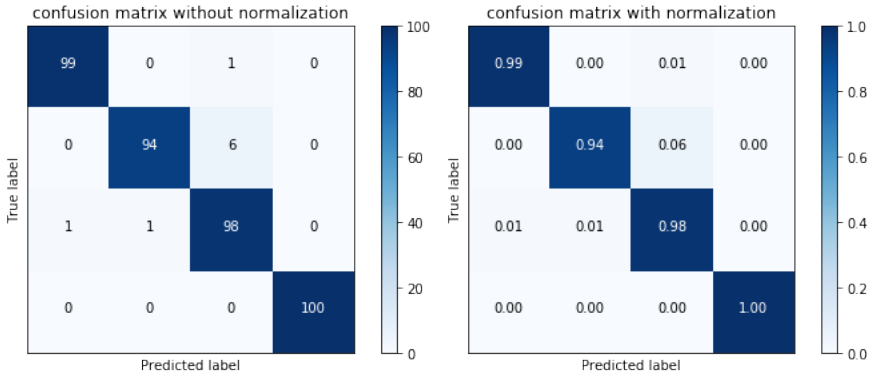
```
Total # of mislabeled points: 9 out of 400
***** Training Accuracy with GNB: 0.985
***** Test Accuracy with GNB: 0.9775
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | 0.99 | 0.99 | 0.99 | 100 |
| 2 | 0.99 | 0.94 | 0.96 | 100 |
| 3 | 0.93 | 0.98 | 0.96 | 100 |
| 4 | 1.00 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.98 | 400 |
| macro avg | 0.98 | 0.98 | 0.98 | 400 |
| weighted avg | 0.98 | 0.98 | 0.98 | 400 |

**Confusion Matrices:**




Although Naïve Bayes is one of the simplest classifiers, it worked very well with our dataset. As mentioned earlier, if features are independent, the classifier works actually well, as this is the case with this dataset. Looking at the confusion matrix, it seems the classifier mostly had trouble differentiating class 2 and 3, and in 6 instances (out of 400) instead of class 2, it predicted the result as class 3.

## 4.2. SVM

For SVM, you can find the results for SVC() and LinearSVC() functions from scikit library separately as follow:

### 4.2.1. SVC()

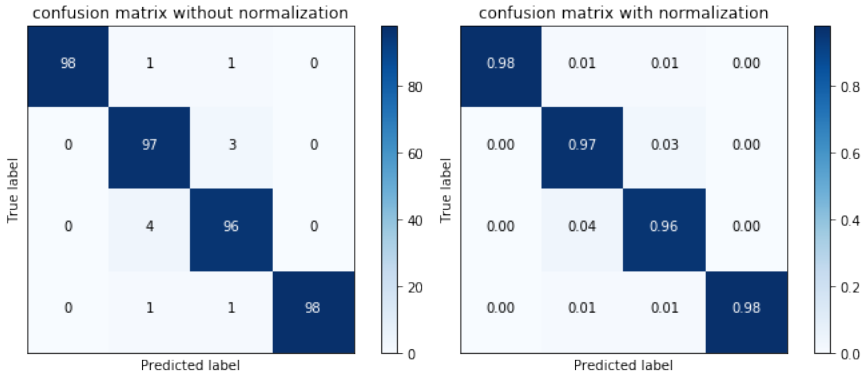
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 11 out of 400
***** Training Accuracy with SVM : 0.999375
***** Test Accuracy with SVM : 0.9725
```

**Precision, Recall and F1-scores for each class:**

| | **precision** | **recall** | **f1-score** | **support** |
|---|---|---|---|---|
| 1 | 1.00 | 0.98 | 0.99 | 100 |
| 2 | 0.94 | 0.97 | 0.96 | 100 |
| 3 | 0.95 | 0.96 | 0.96 | 100 |
| 4 | 1.00 | 0.98 | 0.99 | 100 |
| accuracy | | | 0.97 | 400 |
| macro avg | 0.97 | 0.97 | 0.97 | 400 |
| weighted avg | 0.97 | 0.97 | 0.97 | 400 |

**Confusion Matrices:**




### 4.2.2. LinearSVC()

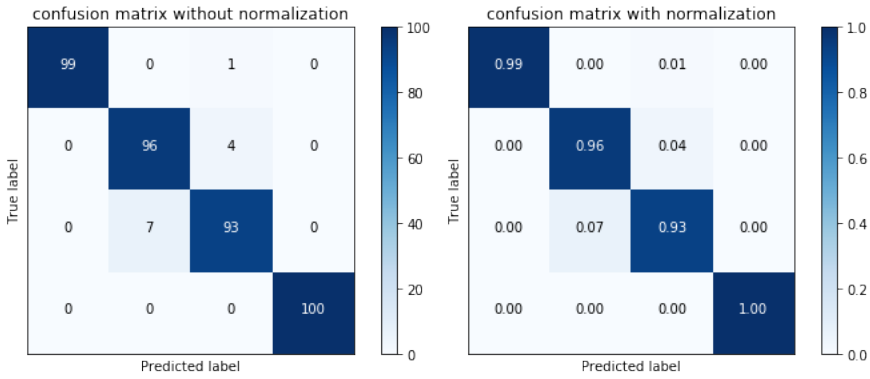
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 12 out of 400
***** Training Accuracy with linearSVM: 0.978125
***** Test Accuracy with linearSVM: 0.97
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |
| 2 | 0.93 | 0.96 | 0.95 | 100 |
| 3 | 0.95 | 0.93 | 0.94 | 100 |
| 4 | 1.00 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.97 | 400 |
| macro avg | 0.97 | 0.97 | 0.97 | 400 |
| weighted avg | 0.97 | 0.97 | 0.97 | 400 |

**Confusion Matrices:**




As the results for both SVM functions indicates, both functions have almost similar test accuracy results, but with SVC doing slightly better. Checking confusion matrices and precision results indicate, this classifier also had the most trouble to differentiate classes 2 and 3 from each other. In general, this classifier also worked well with this dataset. SVC had the least number of

misclassified points (4) compared to the rest of classifiers for verification model.

## 4.3. LDA

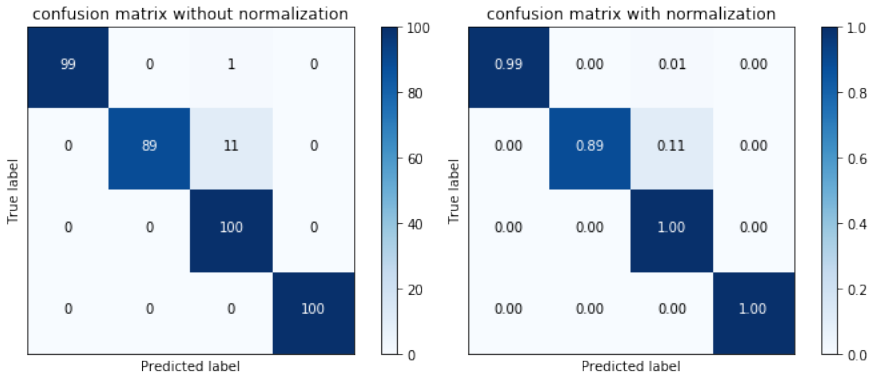
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 12 out of 400
***** Training Accuracy with LDA: 0.9725
***** Test Accuracy with LDA: 0.97
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |
| 2 | 1.00 | 0.89 | 0.94 | 100 |
| 3 | 0.89 | 1.00 | 0.94 | 100 |
| 4 | 1.00 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.97 | 400 |
| macro avg | 0.97 | 0.97 | 0.97 | 400 |
| weighted avg | 0.97 | 0.97 | 0.97 | 400 |

**Confusion Matrices:**




LDA test results and accuracy rate are very good. Looking at precision rates and confusion matrices, LDA had trouble classifying class 2, and misclassified 11 data points instead of class 2 as class 3. It did very well for the rest of classes. It is interesting to see that it classified all the class 3 points correctly but misclassified 11 points from class 2 in class 3 group. This means in this classification, the line between class 2 and 3 was in a way that all class 3 points were correctly classified, but 11 points from class 2 were misplaced in class 3.

## 4.4. QDA

**Accuracy results and # of mislabeled points:**

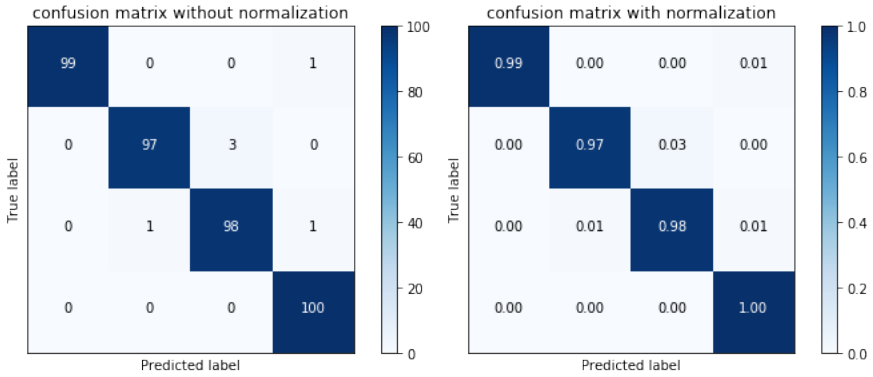
```
Total # of mislabeled points: 6 out of 400
***** Training Accuracy with QDA: 0.980625
```

```
***** Test Accuracy with QDA: 0.985
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |
| 2 | 0.99 | 0.97 | 0.98 | 100 |
| 3 | 0.97 | 0.98 | 0.98 | 100 |
| 4 | 0.98 | 1.00 | 0.99 | 100 |
| accuracy | | | 0.98 | 400 |
| macro avg | 0.99 | 0.98 | 0.98 | 400 |
| weighted avg | 0.99 | 0.98 | 0.98 | 400 |

**Confusion Matrices:**




QDA did better than LDA in total accuracy rates. It seems the boundaries of the rooms do not all have straight lines, and therefore quadratic decision boundary did a better job in classifying the data points compared to the linear discriminant analysis models with linear decision boundaries. Looking at the confusion matrix, the misclassification of 11 points for class 3 instead of class 2 decreased to 3 points. This means the boundary between class 2 and 3 is not a line and has more like a quadratic form.

Comparing the missing prediction points for validation and test datasets, QDA was the only classifier that did better with more points in test dataset. The rest of classifiers with increasing the data points from 320 to 400, had more miss predictions, but not QDA. (ofcourse validation and test dataset were different)

## 4.5. SGD

**Accuracy results and # of mislabeled points:**

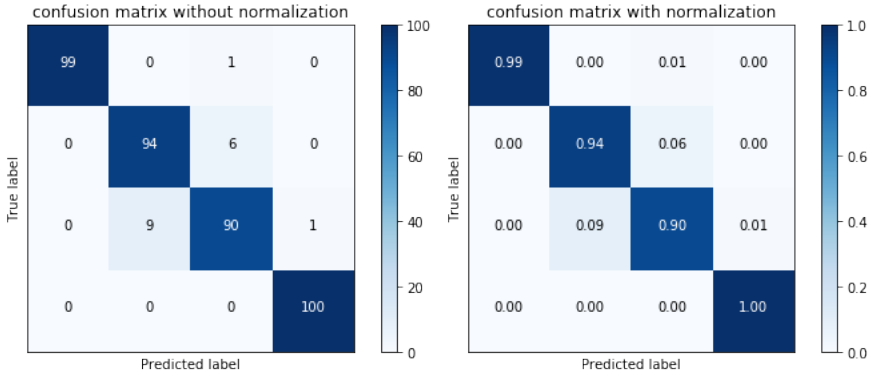
```
Total # of mislabeled points: 17 out of 400
***** Training Accuracy with SGD: 0.970625
***** Test Accuracy with SGD: 0.9575
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |

| | | | | |
|---|---|---|---|---|
| 2 | 0.91 | 0.94 | 0.93 | 100 |
| 3 | 0.93 | 0.90 | 0.91 | 100 |
| 4 | 0.99 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.96 | 400 |
| macro avg | 0.96 | 0.96 | 0.96 | 400 |
| weighted avg | 0.96 | 0.96 | 0.96 | 400 |

**Confusion Matrices:**




Although the test accuracy result of 95% is a really good value, but for this dataset and compared to the rest of classifiers, SGD did not do as well. We see the misclassification of class 2 and 3 for this classification model as well.

## 4.6. KNN

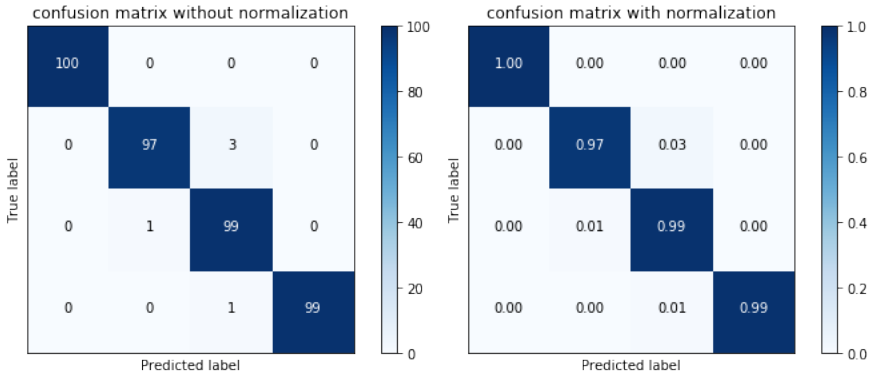
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 5 out of 400
***** Training Accuracy with KNN: 0.99125
***** Test Accuracy with KNN: 0.9875
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 100 |
| 2 | 0.99 | 0.97 | 0.98 | 100 |
| 3 | 0.96 | 0.99 | 0.98 | 100 |
| 4 | 1.00 | 0.99 | 0.99 | 100 |
| accuracy | | | 0.99 | 400 |
| macro avg | 0.99 | 0.99 | 0.99 | 400 |
| weighted avg | 0.99 | 0.99 | 0.99 | 400 |

**Confusion Matrices:**




KNearestNeighbor classifier did very well for all ranges of accuracy rates and had the highest accuracy rate between all classifiers. It is interesting to mention that for both validation and test procedures, KNN only misclassified 5 points which is the lowest among the classifiers, especially for testing dataset. KNN also had trouble with class 2 and 3.

## 4.7. RF

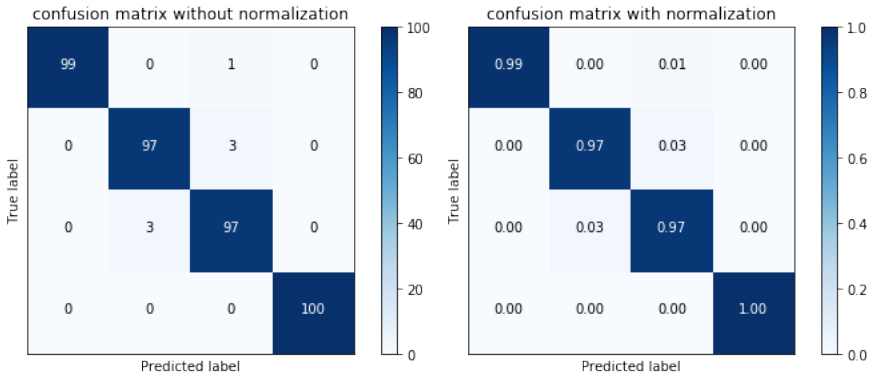
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 7 out of 400
***** Training Accuracy with RF: 1.0
***** Test Accuracy with RF: 0.9825
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |
| 2 | 0.97 | 0.97 | 0.97 | 100 |
| 3 | 0.96 | 0.97 | 0.97 | 100 |
| 4 | 1.00 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.98 | 400 |
| macro avg | 0.98 | 0.98 | 0.98 | 400 |
| weighted avg | 0.98 | 0.98 | 0.98 | 400 |

**Confusion Matrices:**



As mentioned before, Random Forest uses many different trees with different values and possibilities to train the model. Training accuracy of 100% is too high and means the model learned the training set too well. The test and validation accuracy rates are not too low compared to the training rate; therefore, we cannot definitively say it is overfitting. During training, initially n_estimator=500 and max_depth=20 was used and the results were higher. But GridSearchCV suggested lower hyperparameters to lower the possibility of overfitting. We see the misclassification of class 2 and 3 in this model as well.

## 4.8. Linear Regression and Logistic Regression

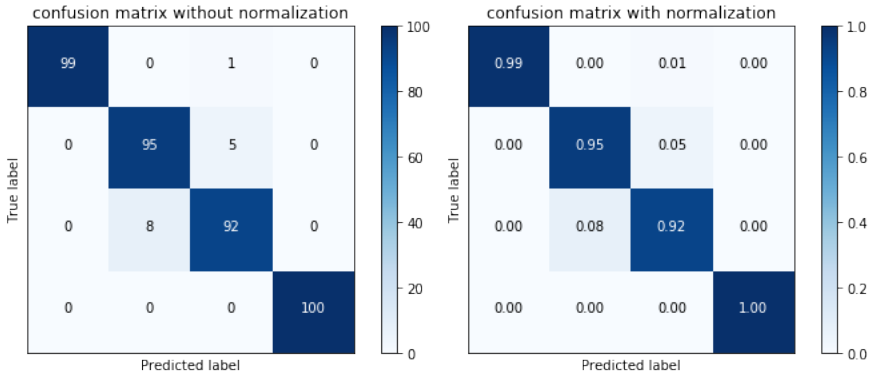
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 14 out of 400
***** Training Accuracy with RF: 0.970625
***** Test Accuracy with RF: 0.965
```

**Precision, Recall and F1-scores for each class:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 0.99 | 0.99 | 100 |
| 2 | 0.92 | 0.95 | 0.94 | 100 |
| 3 | 0.94 | 0.92 | 0.93 | 100 |
| 4 | 1.00 | 1.00 | 1.00 | 100 |
| accuracy | | | 0.96 | 400 |
| macro avg | 0.97 | 0.96 | 0.97 | 400 |
| weighted avg | 0.97 | 0.96 | 0.97 | 400 |

**Confusion Matrices:**




As mentioned before, linear regression is not a good model for classifications. The results obtained by this model confirms it. For regression models we do not calculate accuracy scores and instead we calculate $R^2$.
Training $R^2$=78.12%, Testing $R^2$=78.41%, and MSE was 0.2699. The lower the MSE the better the model. This indicates the features (columns and Wifi readings) are rather independent.

LogisticRegression which is being used for both regression and classification models gave better accuracy rates.
This model too had trouble classifying class 2 and 3 datasets.

## 4.9. Gradient Boosting

**Accuracy results:**

**Training accuracy:** 0.990945571441357
**$R^2$:** 0.9701116759379634
**MSE:** 0.03736040507754567

As mentioned before, Gradient Boosting model uses several weak regression models, and using decision trees, try to get the best predictions. The accuracy results are high which makes this model as one of the best models for classification of this dataset.

## 4.10. RF with PCA

As mentioned earlier, as part of feature dimension reduction techniques, PCA was used and the result was tested with RandomForest classifier and SVM. The results are as follow:

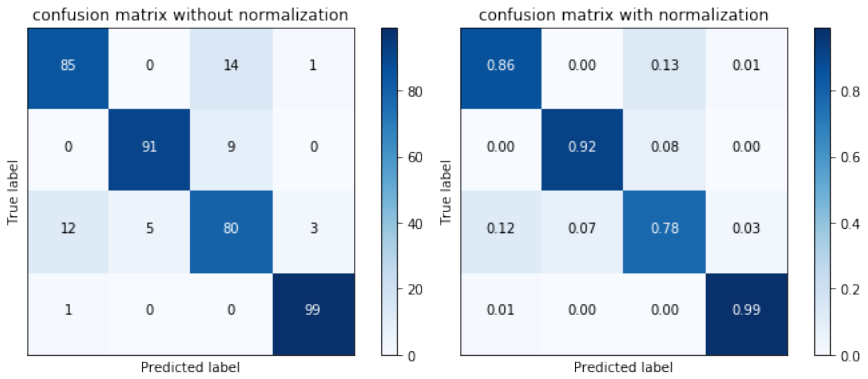
**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 45 out of 400
***** Training Accuracy with RF: 1.0
***** Test Accuracy with RF: 0.8875
```

**Precision, Recall and F1-scores for each class:**

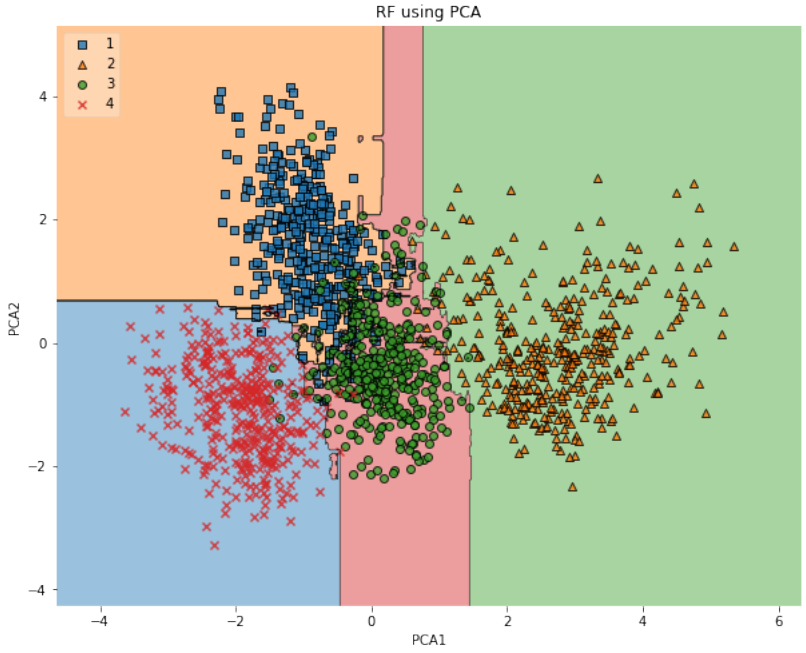
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.87 | 0.86 | 0.86 | 100 |
| 2 | 0.94 | 0.91 | 0.92 | 100 |
| 3 | 0.78 | 0.79 | 0.79 | 100 |
| 4 | 0.96 | 0.99 | 0.98 | 100 |
| accuracy | | | 0.89 | 400 |
| macro avg | 0.89 | 0.89 | 0.89 | 400 |
| weighted avg | 0.89 | 0.89 | 0.89 | 400 |

**Confusion Matrices:**



Looking at the results, we see clearly PCA did not work well with this model and dataset. I believe the reason is that the data collected for this dataset, is the frequency captured in different parts of each locations, and knowing each of the data helps train the model better. Reducing the dimension of this dataset will result in less essential data points which affects the accuracy rates. For this model, even if there is a weaker frequency for a wifi router, it means the user was farther away from that routher. This is by itself a clue as where the user was for the model and losing it will result in lower accurracy rates.

Based on the data, the possible plot of the data points and decision boundaries using RF is presented as follow:



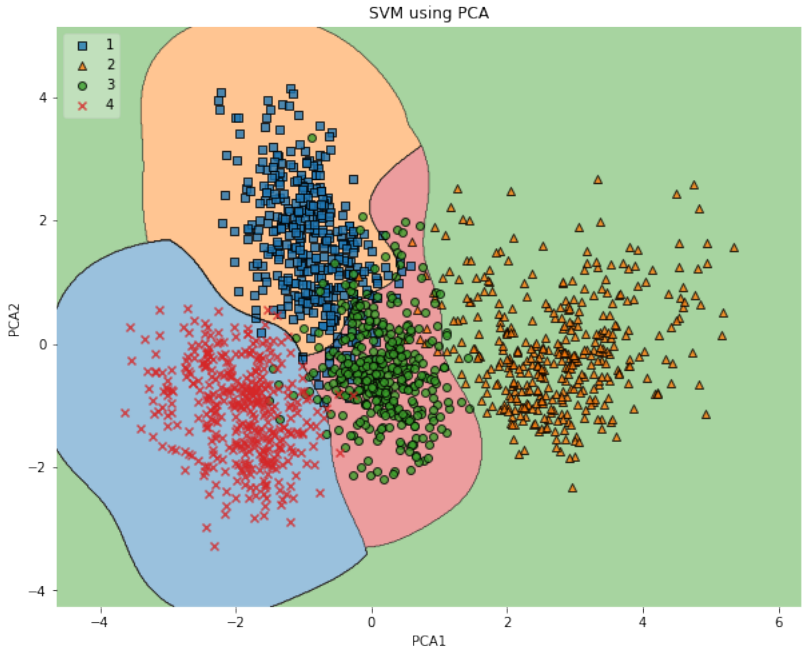

## 4.11. SVM with PCA

The data manipulated using PCA was fed into SVM classifier to check the results with this classifier as well.

**Accuracy results and # of mislabeled points:**

```
Total # of mislabeled points: 39 out of 400
***** Training Accuracy with SVM: 0.935625
***** Test Accuracy with SVM: 0.9025
```

As seen by the accuracy results, SVM worked poorly with PCA manipulated data compared to the whole dataset. The plot for SVM classification is as follow:




As can be seen in this plot, there are a lot of overlapping with data from different classes, and in this case, it is not just limited between class 2 and 3.

**Best classifiers based on Test Accuracy results ranked:**

| Classifier Model | Final Testing_Accuracy |
|---|---|
| KNN | 98.75 |
| QDA | 98.5 |
| RF | 98.25 |
| Naïve Bayes | 97.75 |
| SVM(SVC) | 97.25 |
| SVM(LinearSVC) | 97 |
| LDA | 97 |
| GradientBoostingRegress | 96.82 |
| LogisticRegression | 96.5 |
| SGD | 95.75 |
| RF with PCA | 88.75 |
| LinearRegression (R^2) | 78.4 |

## 5. Contributions of each team member

All the project was done individually.

## 6. Summary and conclusions

In this project I tried to try almost all the classifiers we learned in this class and see how they act on this dataset. The dataset was not a challenging dataset in case of data cleaning. It is also a very good dataset for classification, and if the classifiers are selected correctly, a good prediction result can be obtained.

Almost all of the classifiers had trouble classifying class 2 and 3. Probably the word 'trouble' is a strong word in this example, but they had the most miss classified data between these two classes. Most of the classifiers that draw a line as decision boundary had trouble getting the boundary between Location 2 and 3 right. For instance, LDA had 11 points misclassified as class 3, but this number changed to 3 for QDA with quadratic decision boundary. Also, it is possible that there was not a hard wall between these two locations and as a result the frequencies captured by the devices have some overlaps for these two locations. Either way, probably having more data points could make the model learn their differences better and as a result be able to do a better prediction.

Each dataset can yield different results with different classifiers, and not all classifiers work well with all kinds of data. Therefore, it is up to machine learning engineer to learn about the dataset and learn about the problem and come up with the best possible solution for each different case.

For follow-on work I would like to learn more about location 2 and 3 to be able to come up with a better data manipulation system or tweaking the classifiers to be able to further improve the accuracy results.

## References


[1] UCI Machine Learning Repository: Wireless Indoor Localization Data Set. (2020). Retrieved 7 May 2020, from http://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization

[2] Raschka, S. (2020). Installation - mlxtend. Retrieved 7 May 2020, from http://rasbt.github.io/mlxtend/installation/

[3] 1.4. Support Vector Machines — scikit-learn 0.22.2 documentation. (2020). Retrieved 7 May 2020, from https://scikit-learn.org/stable/modules/svm.html

[4] Linear discriminant analysis. (2020). Retrieved 7 May 2020, from https://en.wikipedia.org/wiki/Linear_discriminant_analysis