Wi-Fi Fingerprinting in Python

The goal of this project is to train a learning classifier that will allow our client to obtain the location of a certain individual inside of a complex and unfamiliar space. The Global Positioning System, or GPS, is a very accurate positioning system; however, it is best suited for outdoor localization, because satellite signals get lost as they try to cross walls and other materials to get inside of a building. Therefore, our positioning system will be based on the corresponding Received Signal Strength Intensity (RSSI) of Wireless Access Points (WAPs) recorded on a mobile phone. We will be using the UJIIndoorLoc Data Set obtained from the UCI Machine Learning Repository. The data set contains 21,048 features/labels points, which 19,937 will be used for training and the remaining 1,111 for validation of the classifiers’ performance. The features in the data set include 520 WAPs with their corresponding RSSI, UserID, PhoneID and Timestamp. The labels in the data set include Longitude, Latitude, Floor, BuildingID, SpaceID and RelativePosition. We will use the training data to train a Random Forest, K-Nearest Neighbor and Support Vector Classifier separately, and test their performance on the validation set; in order to decide on the best model, we will focus on the Accuracy, Kappa and Fit Time of each classifier.

Even though there are 523 features that could be used, the UserID, PhoneID and Timestamp will be removed for this project, because they do not add useful information to the localization of the devices. Also, the labels that we will aim to predict will only be the floor and building that a person is located in. We are left with 520 WAPs as features and the Building/Floor of a person as the labels to be predicted. The WAPs’ signal is represented by a 100 for no signal, -104 for a very weak signal and 0 for the strongest signal; therefore, first, we are going to rescale these values to make 0 mean there is no signal, 1 being a very weak signal and 105 the strongest signal recorded. Next, due to the high number of features given, we will need to apply a dimensionality reduction to the features. I will use Principal Component Analysis to represent the 520 WAPs’ signal variance; after doing so, 95% of the variance/information was contained in the first 110 components produced by the PCA and 99% in the first 221. Then due to the large training set, in order to efficiently train and tune each classifier, samples of 2000, 4000 and 8000 were randomly obtained from the full training dataset. Last but not least, I combined the floor and building labels into one label representing the location; for example, location 1.3 meant that the mobile device was in building 1, floor 3. I know this will not affect the performance of the classifiers, since I tested the classifiers’ performance with the labels separate and combined; combining the features, however, helped simplify the code and improved the efficiency of the classifiers’ training, tuning and testing.

The classifiers were fitted using 10-fold cross validation. Also, classifiers were first trained with the 2,000-sample data set and 110 principal components, in order to acquire an estimate of how long each classifier would take to fit. Before increasing the training data set’s size or number of principal components, in order to make the hyperparameter tune process efficient, the hyperparameters are all initially tuned with the 2,000-sample data set and 110 principal components. The classifiers’ hyperparameters were efficiently tuned by using sklearn.model\_selection’s GridSearchCV and passing it a dictionary with the hyperparameters and their corresponding possible values. After comparing the accuracy and kappa test scores associated with varying hyperparameters, the hyperparameter values associated with the best classifier performance were used for the rest of the process. Then, the principal components used in training were increased and decreased in order to find the number of principal components needed for the classifier to acquire the best accuracy and kappa test scores. Finally, the last step of the classifier tuning process was to increase the sample size used to train the classifier until either, it took too long to fit the classifier or the performance of the classifier began to suffer. After completing this process on the three classifiers, I was able to decide which classifier model was best suited for the Wi-Fi fingerprinting in Python problem by comparing each classifier’s performance metrics. The Random Forest Classifier performed best with the full 19,937-sample training set, 110 principal components and ‘n\_estimators’ being 250. Random Forest fitted in 8 minutes and 55 seconds and achieved an accuracy and kappa test score of 0.9379 and 0.9303, respectively. The K-Nearest Neighbors Classifier achieved an accuracy and kappa test score of 0.9136 and 0.9033, respectively, with a fit time of 13 seconds. The K-Nearest Neighbors achieved this performance by using the full 19,937-sample training set, 50 principal components, ‘n\_neighbors’ of 1 and ‘p’ value of 2. Last but not least, the Support Vector Classifier was able to reach an accuracy and kappa test score of 0.9064 and 0.8953, respectively, with a fit time of 1 second. The Support Vector Classifier used the 8000-sample training set, 325 principal components, ‘C’ of 0.1, ‘kernel’ set to poly, ‘degree’ of 2 and ‘gamma’ set to auto. Below is a linear graph that visually compares the classifiers’ performance metrics discussed above.

Figure 1: Classifier Performance Metrics

Even though, Random Forest took the longest time to fit, it outperformed the other two classifiers in what were the most important performance metrics, accuracy and kappa. If Random Forest had only a slight advantage in accuracy and kappa, I might have gone with a faster and slightly less accurate classifier, however, the Random Forest Classifier achieved significantly higher accuracy and kappa testing scores. Therefore, I recommend our client to use the optimized Random Forest Classifier discussed above. In the future, in order to improve the performance metrics of the classifiers, it would be useful to try different feature engineering methods like Independent Component Analysis (ICA) instead of PCA, and see if performance increases. Due to the time constraint of the project, only three classifiers were trained and tested. Future work might also look into using other classifiers to predict the location of the mobile device, and/or change the process used to predict the device’s location; for example, instead of directly predicting the building and floor, we can first predict the longitude and latitude of the mobile device using a regression algorithm and from there, predict the building and floor. A more advanced project might also be able to locate a mobile device as it moves inside the building and/or even predict where the client might go next, instead of only locating a stationary device.