|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ramses Meza  July 2021  <https://github.com/lordtable/UTDSGC_C4T3> |  | Data Analytics Course 4.3: Evaluating Techniques for WiFi Indoor Location |

Business Acumen

Although outdoors geo-location is a mature technology based on GPS, indoors location using the same technology is not technically feasible due to the difficulties of GPS signal-triangulation to work indoors. Indoors positioning is required when dealing with large facilities, such as college campuses, factories and warehouses, helping improving the logistics related to the correct positioning & navigation. Several efforts are being made in industry and academia to provide solutions for indoors location, being *WiFi Fingerprinting* one of those, which is very promising. IoT analytics has been commissioned by a client to carry out a project consisting on building a model that can correctly predict labeled in-campus locations based on the observed WiFi signal strength from several WiFi hotspots. Such model is expected to be embedded on a mobile app.

**Data Management, Cleaning & Pre-processing**

The dataset was provided on a .csv file format and locally stored. The file is structured, where each row corresponds to an instance/sample (19,937 instances) and each column corresponds to a feature or attribute (total 529). Each instance is a measurement of WiFi signal strength (RSSI) of 520 WiFi hotspots (located all around a college campus). The data collection for each instance consisted on an individual recording his/her location on campus based on a categorical coding and register the RSSI (WiFi strength) of each of the 520 hotspots at that location. Data features are (after re-ordering in Microsoft Excel):

* Location Longitude (numeric)
* Location Latitude (numeric)
* Floor ID (Categorical-Integer): 0🡪4
* Building ID (Categorical-Integer): 0🡪2
* Space ID (Categorical-Integer): Equivalent to room number. 1🡪254
* RelativePosition (Categorical-Integer): Relative position with respect to Space ID: 1 (inside) or 2 (outside).
* User ID (Categorical-Integer): 1🡪18
* Phone ID (Categorical-Integer): 1🡪24
* Timestamp (Integer): Unix time of instance capture.
* WAP001🡪WAP520 (integer): Intensity of WiFi signal (RSSI) for each of the 520 WLAN or WiFi hotspots across the campus, measure on decibels (dB): -104 dB (very weak signal)🡪0 (very strong signal). +100 dB means no signal detected.

The dataset was saved as a RStudio dataframe for data analysis. After executing a quick EDA via *summarytools*, no missing values were reported. The *Dplyr* library was used for further preprocessing, and it was found that 637 instances were duplicated and removed.

**Feature selection & engineering**

Given the nature of the problem and the very large number of features, I decided to spend a considerable amount of time to reduce the dimensionality of the dataset by different avenues, in order to end up with a leaner version that can be more easily managed by the machine learning algorithms to test in terms of classification performance and computing time. The later was also helped by taking advantage of R parallelization capabilities on the *doParallel* library, allocating 4 cores out of the 8 available. The sub-sections to follow describe each of the Feature Selection & Engineering steps taken to accomplish the dimensionality reduction.

**Create Target Feature *Loc* & Feature Selection**

This data analytics approach is a classification one, therefore, a single target feature is needed that contains all the pertinent information that describes a unique campus location code that can be easily interpreted. I created the feature *Loc*, which consisted on the combination of *Building ID, Floor ID, Space ID* and *RelativePosition*, which are all coded themselves. An example of the new *Loc* feature is ***1 0 203 1***, which is interpreted as Building #1, Floor 0 (ground level), Room/Space #203, inside of the room. The *Loc* target variable was then converted from *character* to *factor* data type, allowing to verify that there are 905 distinct labeled locations. Creating this new feature made its feature predecessors irrelevant, therefore I removed those accordingly, along with *Latitude* and *Longitude*, with the data dimensions going from 529 to 525. *Timestamp* was also removed, since I assumed that RSSI is time-invariant, making the new data dimensionality 524. *UserId* and *PhoneId* were also considered irrelevant, since it is expected that in case of a new instance requiring location, the only info available would be the WAPs signal strength, not any factor describing the user and its mobile device. Removing these yielded a data dimensionality of 522.

**Removal of low-variance features**

Considering that the range of RSSI is -104 db (weakest) to 0dB (strongest), and +100 dB represents un-detected, I decided to treat the undetected value as a no-strength signal, substituting it in the dataset by -110 dB instead. I considered that this helped on reading RSSI ranges on a more intuitive way without regards of the nature of the no-detection. This also contributed on a much easier removal of zero and low-variance WAP features. By inspecting the histograms of several WAP features as generated by the *summarytools* report, I noticed that those with larger variance were characterized by having a number of distinct values in the order of 30-80, while the low variance ones having much less than 30. Also, the low-variance features were characterized by very high frequency ratios, in the order of hundreds and above. Based on these observations I applied the *nearZeroVar()* function to remove the zero and low-variance features by using a frequency ratio threshold *freqCut*=400, and *uniqueCut*=0.3. Inspection of the new dataset via *summarytools* allowed confirming that the surviving WAP features were those having a larger variance. The new data dimensionality was reduced to 200 predictors.

**Data splitting & Principal Components Analysis**

As dictated by best practices, I split the dataset randomly into a training and a testing set, where the former represents 70% of the samples of the new dataset, while the latter consists of the remaining 30%. Since this classification problem is aimed at predicting a location, we are not interested on determining feature importance, therefore the preservation of the original features is not a priority. Although the current dimensionality has being greatly reduced from 529 to 200, it is still a very large one, so this makes the data worth of performing Principal Component Analysis (PCA). This was performed as follows:

1. Determine component eigenvectors & eigenvalues based on the training set only. This implicitly includes the rotations/transformations to be applied to the WAP features.
2. Apply the PCA rotations/transformation to the entire dataset.
3. Perform a training/testing split of the transformed dataset (PCA) using the same random seed and partition proportions as used before.

Performing the process this way, we guarantee that the PCA transformation is based only on data to be seen by a machine learning algorithm, therefore any test data does not influence the transformation, but undergo the PCA already in place.

Figure 1 depicts a plot of the PCA cumulative variance explained versus the PCA components (in descending order of importance) from the training set. We can deduct that, for instance: the first 20 principal components already account for about 69% of the total variance of the dataset, the first 95 components explain 95% of the total variance, and the first 134 components account for 98% of the total variance. If we settle for 95% of total variance explained, then we could potentially further reduce the dimensionality from 200 to 95 features.

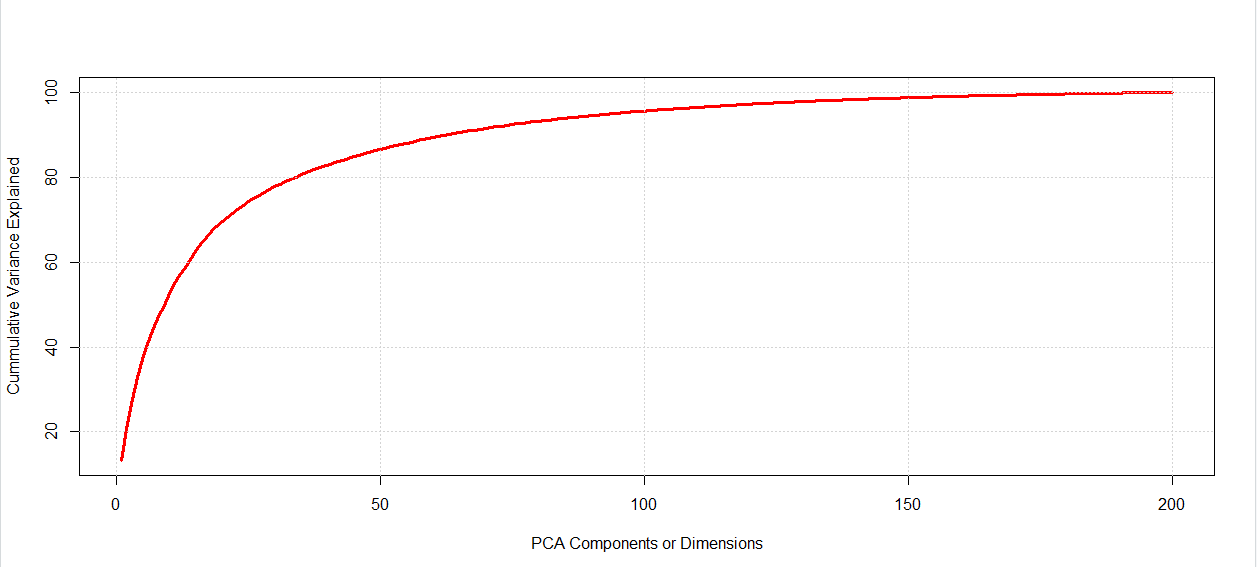


Figure 1: PCA Cumulative Variance Explained (training set) vs PCA components (on descending order of importance)

**Dataset selection for modelling**

A decision needed to be made in terms of the dataset to be used going forward: either the dataset with un-transformed (preserved WAPs) predicting features (200 features), hereby called baseline, or the dataset with the first 95 principal components acting as predicting features, hereby called PCA dataset. I decided to follow a quick modelling route in order to make a decision based on accuracy and running time performances. For that, I used the k-nearest neighbors (kNN) algorithm with its default settings in a classification mode, since it is a very fast algorithm with reduced hyper-parametrization that can assist on the comparison. Figure 2 depicts a comparison of the accuracy of the generated kNN models applied to both baseline and PCA datasets for some sample *k* values. The first observation is that, as *k* increases, the accuracy decreases, as expected for the algorithm. The other observation, and the most important one, is that for a given *k* value, the accuracy of the models derived from both the baseline and PCA datasets is very close to each other. This means that even tough the first 95 principal components account for 95% of the total variance of the baseline dataset, they are enough to yield the same level of accuracy as the baseline dataset with its 200 predictors. It is also important to emphasize that each kNN run was timed by means of the *proc.time()* function, and each kNN run on the PCA dataset took about half of the running time of that on the baseline dataset. Based on this evidence, I made the assumption that the PCA dataset would benchmark favorably if used on other more sophisticated and time-consuming modelling algorithms of machine learning. From this point forward, the only dataset to be used was the PCA.

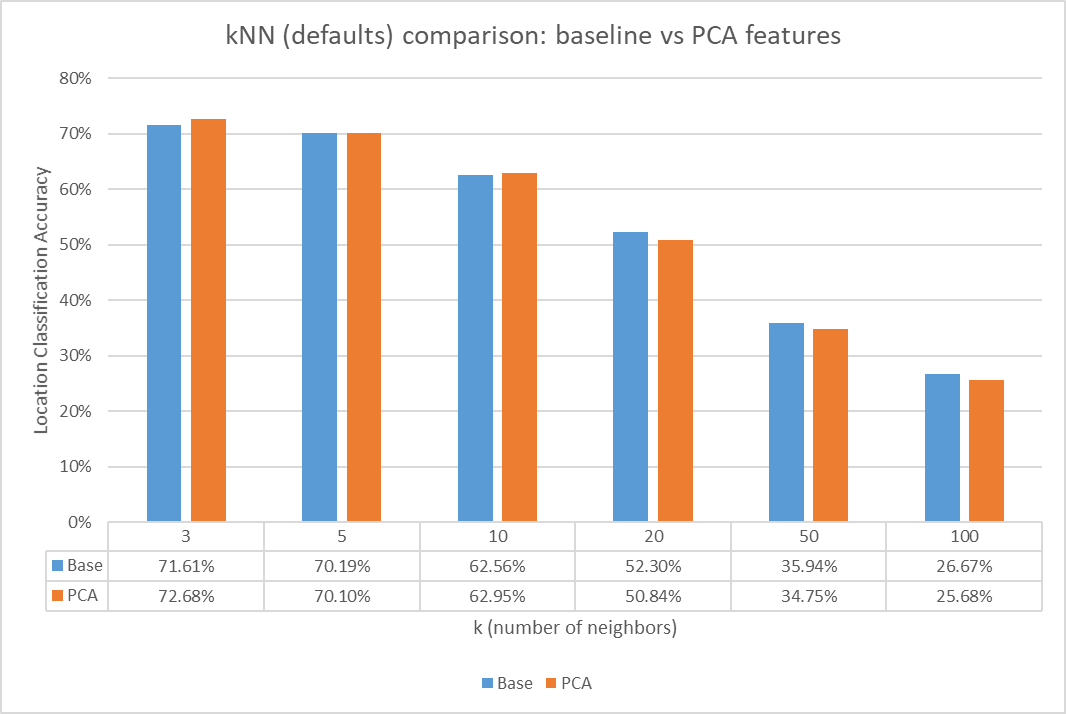


Figure 2: Performance comparison of baseline vs PCA datasets in terms of model accuracy as yielded by kNN for some k values.

**Predictive Model Building & Evaluation**

The PCA-transformed dataset was selected as the data to be used for model building. The corresponding training portion of the dataset was used to train the following three (3) algorithms:

* **k-Nearest Neighbors (kNN)**: Grid-based hyper-parametrization tuning consisted only on testing a *k* range 1🡪10 using accuracy as the metric for final parametrization set selection criteria.
* **C5.0 Decision Tree (C5.0)**: Grid-based hyper-parametrization tuning consisted on testing the number of trials (=1,5,10,15 & 20) and *winnow* (TRUE or FALSE), using accuracy as the metric for final parametrization set selection criteria.
* **Random Forest (RF)**: Grid-based hyper-parametrization tuning consisted only on testing a *mtry* (=1,2,3,4 & 5) using accuracy as the metric for final parametrization set selection criteria.

Gradient Boosting was also attempted, but it took a very long time to run (> 48 hrs) without finalizing, therefore I decided to abort it. All the models above also incorporated 10-fold cross-validation.

Figure 3 shows summary tables for each algorithm with the tuned parameters that yielded the highest accuracy, as well as the resulting *kappa*. The model-building or training process was also timed. Location predictions were made with each tuned model out of the test set, confusion matrices were obtained, and from these, the corresponding accuracy and *kappa* for the predictions. For the same tuned model/algorithm, we can see that accuracy for training and testing is very close, indicating that there is a low risk of overfitting. Secondly, kappa values are very close to their corresponding accuracy values, indicating that the observed level of accuracy is hardly due to chance alone, signifying a high statistical significance.

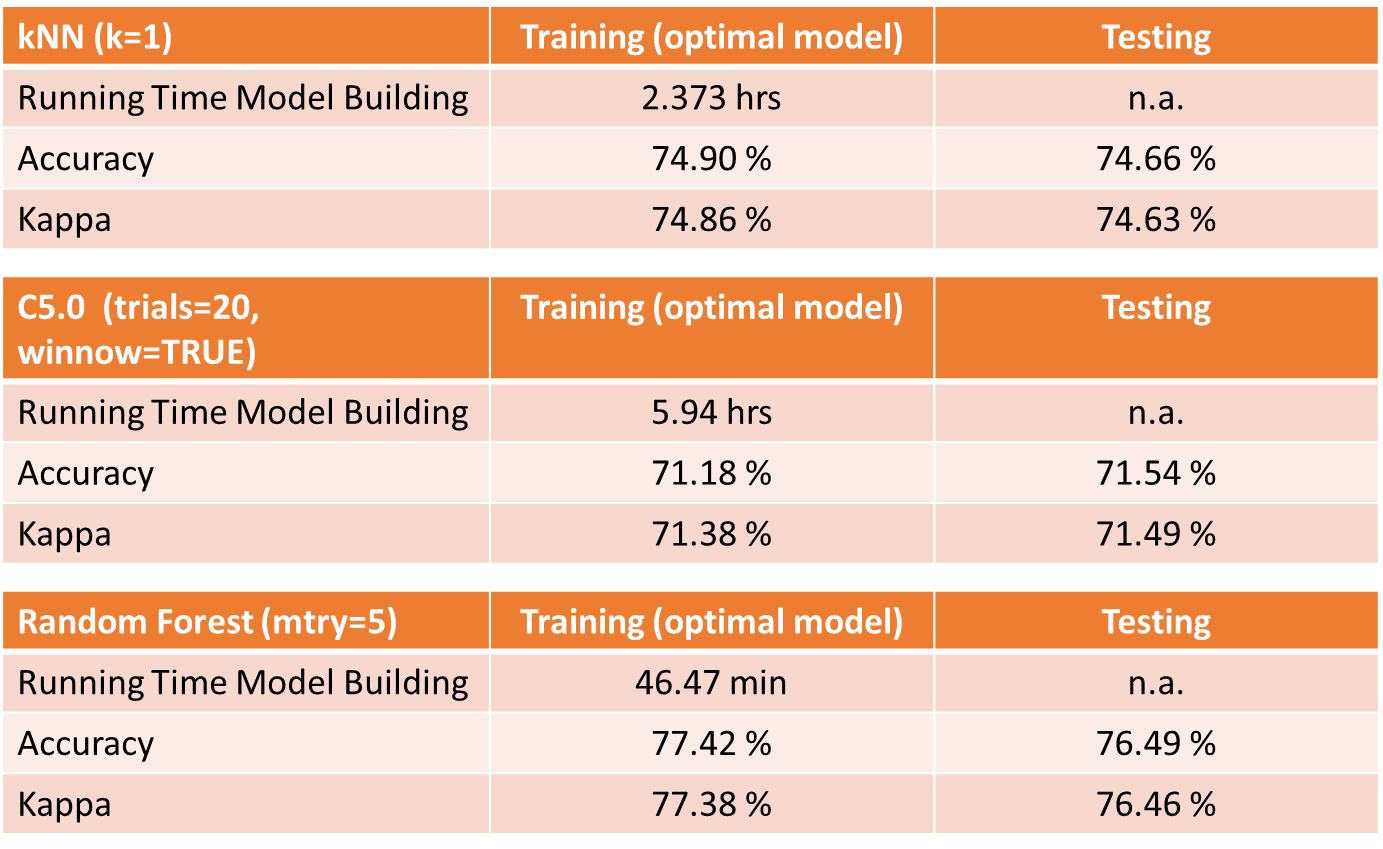


Figure 3: Performance comparison three (3) different tuned models.

To emphasize the statistical-robustness of the built models, the *resamples()* function was used to obtain a distribution of accuracy and *kappa* metrics yielded during the 10-fold cross-validation applied on the training set. The summary of those is depicted on Figure 4, confirming that, for a given model, accuracy and kappa are very close to each other, and for a given model and metric, the metric’s distribution spread is quite narrow, indicating robustness on the metrics reported on Figure 3.

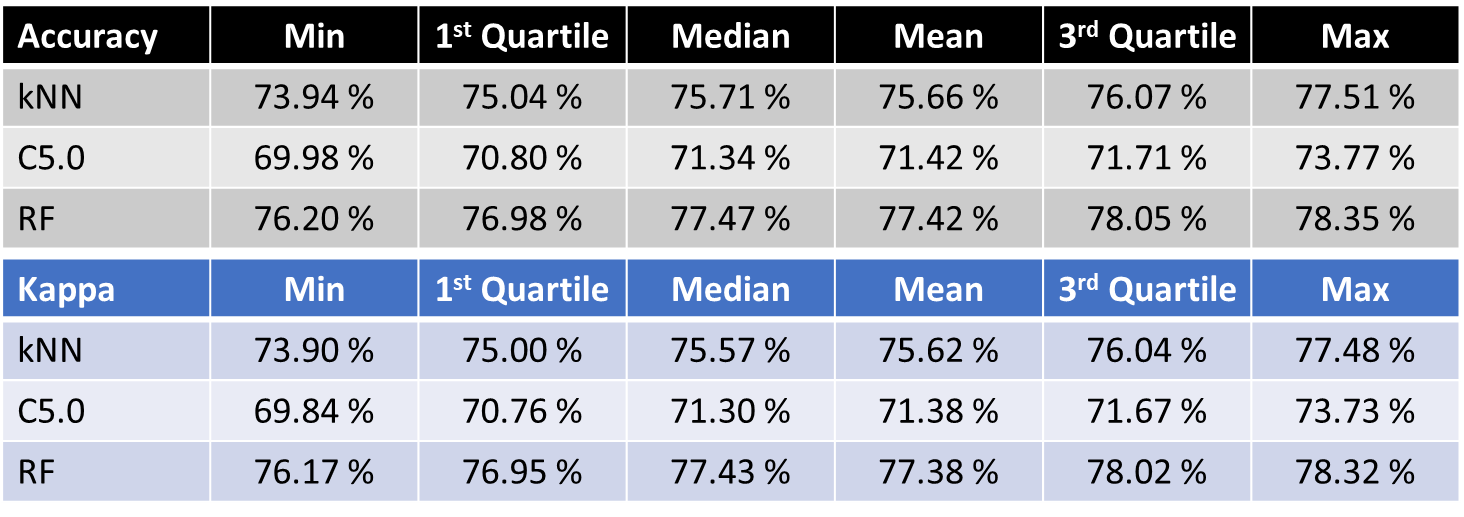


Figure 4: Performance comparison three (3) different tuned models based on accuracy and kappa distribution metrics (training set).

**The Random Forest (RF) model yields the highest accuracy and kappa (>76 %), while executing in the lowest amount of time (< 1 hour), when applied to a PCA-transformed dataset, therefore it is the recommended model at the current stage of work**.

**Recommendations for future work**

There may be some approaches that, either separately or combined, can help improving the modelling performance and execution time, some of those are shown below (without a specific order):

* **Tiered-modelling approach**: Since the goal is to build a model embedded on a mobile app, different tiers or tranches of models can be built, to funnel the user from the more general to the more specific location. For instance, the app may be built in such a way that, when the user enters a campus, he/she only indicates the building the user wishes to go to. The corresponding model the app will use would be the one built from a subset of instances that include all the WLAN sensors of all buildings located nearest each building entrance. Then, when the user is at the building entrance of interest, the next location tier kick is, which is the floor. In this case, a model using only the WLAN sensors of that building kicks-in, and so on. This approach can contribute to significantly reduce the number of instances and features used, as well to increase the accuracy with the disadvantage of requiring building several models.
* **Attempt other algorithms**: Using the same PCA data used here, other algorithms may be attempted. One good candidate may be Naïve Bayes, only requiring the additional step of binning/discretizing each PCA value.
* **Take-in less PCA components**: In this project, I settled for 95% of cumulative variance explained, which corresponded to the first 95 principal components. There may be a chance that we settle for a lower cumulative variance explained, which translates on a much lower number of principal components to take as predictors. For instance, 90% of cumulative variance can be explained by the first 63 principal components; or 80% by the first 34 components. Testing of accuracy and runtime for each of these sets may still yield an acceptable performance.