**Indoor localization and Wi-Fi Fingerprinting**

DSAI 3201 Machine Learning: Individual Report Part 2

Indoor localization and Wi-Fi Fingerprinting Using Machine Learning and Deep Learning

Joy Anne P. Dela Cruz[[1]](#footnote-1)\*

Student, University of Doha for Science and Technology, 60301959@udst.edu.qa

Indoor localization is a widely used system for determining the position of devices indoors. As part of the University of Doha for Science and Technology's DSAI-3201 Machine Learning Course, this project will analyze the UJIIndoorLoc database to explore Wi-Fi fingerprinting techniques for indoor positioning. This second half of the paper will discuss my personal specific contributions to the project, challenges, and technical details of the work.

CCS CONCEPTS • Wi-Fi Fingerprinting • Data Exploration • Machine Learning • Convolutional Neural Networks

**Additional Keywords and Phrases:** Wi-Fi Fingerprinting, Indoor localization

1. Introduction

Indoor localization, also known as indoor positioning system (IPS), is a technology that determines the position of objects or people within enclosed spaces [1]. There are many ways of applying indoor localization; this project will focus on using Wi-Fi Fingerprinting. There are three components to apply indoor localization in this method. (1) Wi-Fi access points that broadcast signals to be received by (2) mobile devices; these signals will be input in (3) an algorithm that processes the data to give an estimated location. Wi-Fi Fingerprinting technique relies on signal strength data which represents the distance of the Wi-Fi access point from a mobile device. It involves training the fingerprinting algorithm to learn the RSSI (Received Signal Strength Indicator) at various points, making markings in the indoor environment creating a fingerprint database [2]. Then, a mobile device can detect signals from the access points and the database matches this current information with the ones stored from the training phase to estimate the device's location. This group project combines the effort of Asma and I to create and implement the algorithm using Machine Learning and Deep Learning.

* 1. Dataset

The UJIIndoorLoc dataset is a Multi-Building, Multi-Floor indoor localization database designed to test Indoor Positioning Systems that rely on WLAN/Wi-Fi fingerprinting. It was created by Torres-Sospedra et al [3]. This dataset contains 528 attributes. Of these, 520 are RSSI values, which represent signal intensities from different Wi-Fi Access Points (WAPs). These values help determine the presence and strength of Wi-Fi signals at various locations. The dataset also includes positional attributes: Longitude and Latitude (provide geographic coordinates), Floor and Building ID (indicate where the data was recorded). The other features in this dataset will provide context but they will not be used in the model: Space ID, Relative Position, User ID, Phone ID, and Timestamp.

* 1. Objective

With my knowledge of Wi-Fi fingerprinting, my task is to explore, clean, and prepare this dataset for model development. I am assigned to develop two models using RSSI features:

• Classification Model – Predicts Building ID and Floor using Wi-Fi fingerprints.

• Enhanced Regression Model – Predicts Longitude and Latitude to determine precise location coordinates based on Wi-Fi fingerprints.

These models will help improve indoor localization, which is essential for navigation in areas where GPS is unreliable.

1. contribution and Technical details

This section of the paper will discuss half of the group project done by me.

* 1. Data Preprocessing

After conducting our own data exploration and analysis, we shared our findings of the dataset and brainstormed possible approaches to cleaning and preparing the data. We agreed to keep the data preparation consistent between each other and so I standardized our data preparation with a function that reads the csv path, replaces 100 where no signal found to -105, removes some RSSI columns that were never detected, checks for missing values, and scales the RSSI columns and the coordinates of each record. This function will be used throughout the model development.

* 1. Prediction Evaluation

To ensure consistent evaluations of models between the both of us, I created two functions that compare model predictions and versus the wanted predictions by calculating certain metrics and visualizing the predicted vs actual to evaluate the models consistently.

def load\_and\_clean\_dataset(path):

print(f"Loading {path} and applying the same preprocessing")

df = pd.read\_csv(path)

print(">> The values in WAPxxx columns with 100 is turned into -105")

RSSI\_columns = [col for col in df.columns if col.startswith("WAP")]

df[RSSI\_columns] = df[RSSI\_columns].replace(100, -105)

print(f">> Dropped {len(WAP\_building\_not\_detected)} WAP columns that were never detected.")

df.drop(columns=WAP\_building\_not\_detected, inplace=True)

RSSI\_columns = [col for col in df.columns if col.startswith("WAP")]

print(">> Checking if there are any missing values:", df.isna().sum().any())

print(">> Scaling the numerical columns (RSSI\_columns and Coordinate Columns) with MinMaxScaler")

scaler = MinMaxScaler(feature\_range=(0, 1))

df[RSSI\_columns] = scaler.fit\_transform(df[RSSI\_columns])

scaler\_coords = MinMaxScaler(feature\_range=(0, 1))

df[["LONGITUDE", "LATITUDE"]] = scaler\_coords.fit\_transform(df[["LONGITUDE", "LATITUDE"]])

return df

When evaluating the classification model, the following metrics were calculated: accuracy, precision, recall, and F1. The macro, micro, and weighted calculations of precision, recall, and F1 score were calculated—focusing on improving at F1-macro as our classes are not equal. Then, to visualize the predictions versus actual, a confusion matrix heatmap will be created by the function.

def evaluate\_classification(modelName, yb\_test, yb\_pred, yf\_test, yf\_pred):

combined\_data = [ (yb\_test, yb\_pred, "Building ID"), (yf\_test, yf\_pred, "Floor")]

# Classification Report

for y\_true, y\_pred, title in combined\_data:

print(f"============== {modelName} {title} ==============")

print(">> Accuracy:".ljust(22), accuracy\_score(y\_true, y\_pred))

print(">> Precision macro:".ljust(22), precision\_score(y\_true, y\_pred, average='macro'))

print(">> Precision micro:".ljust(22), precision\_score(y\_true, y\_pred, average='micro'))

print(">> Precision weighted:".ljust(22), precision\_score(y\_true, y\_pred, average='weighted'))

print(">> Recall macro:".ljust(22), recall\_score(y\_true, y\_pred, average='macro'))

print(">> Recall micro:".ljust(22), recall\_score(y\_true, y\_pred, average='micro'))

print(">> Recall weighted:".ljust(22), recall\_score(y\_true, y\_pred, average='weighted'))

print(">> F1 Score macro:".ljust(22), f1\_score(y\_true, y\_pred, average='macro'))

print(">> F1 Score micro:".ljust(22), f1\_score(y\_true, y\_pred, average='micro'))

print(">> F1 Score weighted:".ljust(22), f1\_score(y\_true, y\_pred, average='weighted'))

print(classification\_report(y\_true, y\_pred, digits=3))

# Confusion Matrix Heatmap

plt.figure(figsize=(12, 5))

for i, (y\_true, y\_pred, title) in enumerate(combined\_data):

cm = confusion\_matrix(y\_true, y\_pred)

plt.subplot(1, 2, i+1)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title(title)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.suptitle(f"Confusion Matrix {modelName}")

plt.tight\_layout()

plt.show()

Its regression counterpart takes in similar parameters but computes for the R², mean-squared error (MSE), root-mean-squared error (RMSE), and mean-absolute error (MAE)—now focusing on lowering RSME as RMSE punishes outliers more and it is more interpretable than MSE. Instead of a confusion matrix, an actual versus predicted scatterplot is created instead.

def evaluate\_regression(modelName, yn\_test, yn\_pred, yt\_test, yt\_pred):

combined\_data = [ (yn\_test, yn\_pred, "Longitude"), (yt\_test, yt\_pred, "Latitude")]

# Regression Report

for y\_true, y\_pred, title in combined\_data:

print(f"\n============== {modelName} {title} ==============")

print(">> R2:".ljust(10), r2\_score(y\_true, y\_pred))

print(">> MSE:".ljust(10), mean\_squared\_error(y\_true, y\_pred))

print(">> RMSE:".ljust(10), np.sqrt(mean\_squared\_error(y\_true, y\_pred)))

print(">> MAE:".ljust(10), mean\_absolute\_error(y\_true, y\_pred))

# Actual vs Predicted Scatter Plot

plt.figure(figsize=(12, 5))

for i, (y\_true, y\_pred, title) in enumerate(combined\_data):

plt.subplot(1, 2, i+1)

sns.scatterplot(x=y\_true, y=y\_pred, alpha=0.5)

plt.plot([min(y\_true), max(y\_true)], [min(y\_true), max(y\_true)], 'r--')

plt.title(title)

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.grid(True)

plt.suptitle(f"Actual vs Predicted {modelName}")

plt.tight\_layout()

plt.show()

For both classification and regression, I made a function that compares the F1 macro and RMSE respectively on the test split and validation dataset. This can help visualize the difference of performance between models. These functions are meant to be used throughout the model development process.

The trainingData.csv provided the UJIIndoorLoc dataset is extensive enough to split the dataset for both training and testing and so I promoted the idea that we keep the validationData.csv untouched up until the end. Therefore, we can evaluate if our models are able to generalize enough to work well on never-before-seen data simulating a real-life deployment of our models.

* 1. Basic Model Classification

The basic model classification started with a training-testing split: 80% for training and 20% for testing. I kept the random state consistent at 42 to make the code reproducible.

* + 1. Decision Tree

I started with the simplest multi-class classification I know which is a decision tree classifier. I used the one from the Scikit-learn library.

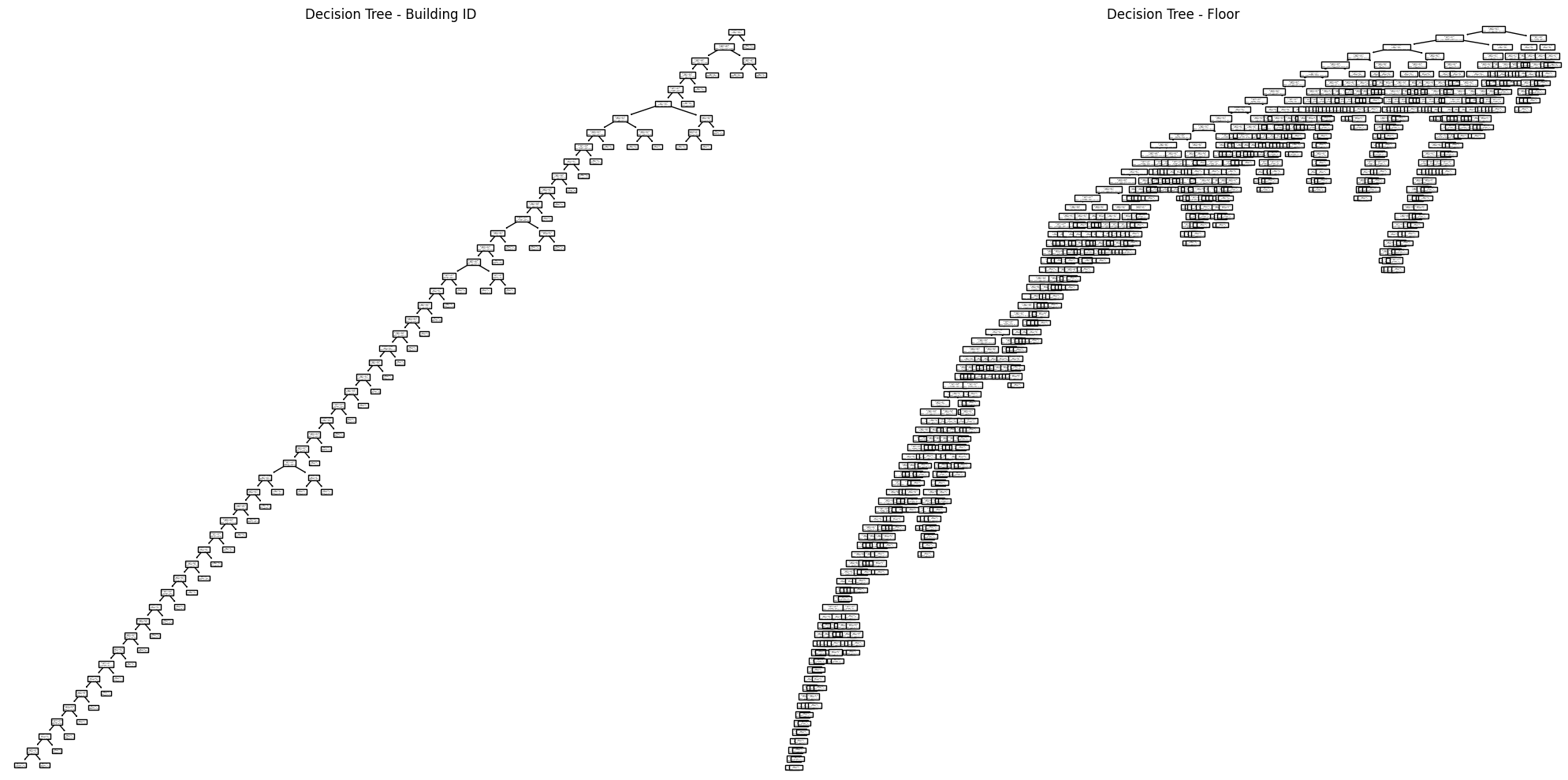


Figure 1: Decision Tree Classifier for Building and Flood

The floor decision tree is more comprehensive and fuller as there are more floors to predict. There are five floors in the dataset compared to only three buildings. The branches of both trees mostly only go to one side which was a concern at first but after evaluating their metrics and applying cross-validation with good results, it stopped being a concern. This model performed extremely well considering its simplicity and interpretability. During the initial testing using default parameters, all metrics are already above 0.99 for predicting the building and 0.97 for predicting the floor. This includes the F1 score macro that we are focusing on. It preformed excellently predicting the buildings—3,987 records are used for testing, and it only got the buildings of 9 records wrong. All those 9 records predicted building 1 when it was in fact building 2. The decision tree for predicting the floor also performed very well, with all metrics at over 0.97. If there's a mistake in the prediction, it's usually the floor directly above or the floor directly below. There were only 18 records predicted out of this range.

A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 2: Decision Tree Classifiers Actual vs Predicted

The results look very promising. If I were doing a personal project, I would settle for this model and apply some hyperparameter tuning and cross-validation to make sure that my model is not overfitting but for the sake of this project, I did more models to compare this one from.

* + 1. Support vector machine

Next, I tried creating a Support Vector Machine (SVM). An SVM is can be used for classification and regression; it works by **finding the optimal hyperplane** that best separates different classes in a high-dimensional space. This SVM is also from the Scikit-learn library. When training the model initially, I tried to keep it as simple as possible. Usually for SVM, PCA is performed but for the first trial, I tried without PCA as I wanted to keep the structure of the columns. I even attempted to visualize the SVM predictions with PCA2 and PCA3 but the graph did not produce much insight and therefore I ended up not putting it in the final product. I kept the kernel at linear because our dataset is of high dimensionality. I also kept the C at 1 which is default I also kept the C parameter at 1, which is the default. The C value controls the trade-off between maximizing the margin and minimizing classification errors. Keeping it at 1 is a reasonable start for initial testing.

The SVM models perform slightly better than the decision tree models. When it comes to predicting the building, SVM performs marginally better, only misclassifying 8 records instead of 9. However, when predicting the floor, there's a slightly bigger gap between the decision tree and the SVM model. Instead of 0.744, the F1 macro score for SVM is 0.9933.

A graph of a number of numbers and a number of squares

AI-generated content may be incorrect.

Figure 3: SVM Classifiers Actual vs Predicted

* + 1. Random Forest

Next, I wanted to improve on our tree models, since the decision tree classifier performed so well for being so simple. I tried to create a random forest, again using the Scikit-learn library random forest classifier. Again using default parameters, the building prediction are similar to the SVM building classifier, but it performed slightly better than SVM when it comes to predicting the flood with a F1 macro score of 0.9965, a very marginal improvement.

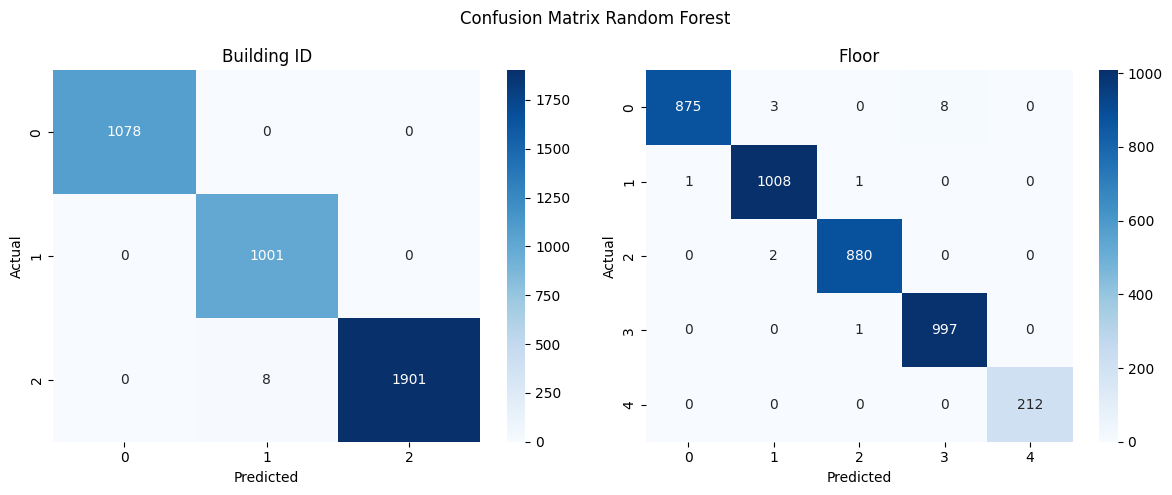
****

Figure 4: Random Forest Classifiers Actual vs Predicted

* + 1. Cross validation

With the simple models working very well with default parameters, I applied some cross-validations with 10 splits to make sure that the model generalizes well enough and it's not merely memorizing the data. I chose 10 splits because it's a commonly used trade-off: it provides a reliable estimate of generalization performance without being too computationally expensive. It also ensures that each fold still has enough data for training and testing. The results of the cross-validations seem consistent with the results from our simple train-test splits from earlier.

Table 1: Average Cross Validation Metrics on Initial Classification Models

| Metric | SVM (Building) | RF (Building) | Tree (Building) | RF (Floor) | SVM (Floor) | Tree (Floor) |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | 0.997793 | 0.997743 | 0.996890 | 0.996740 | 0.993178 | 0.974319 |
| Precision Macro | 0.997725 | 0.997962 | 0.996829 | 0.997230 | 0.994047 | 0.974701 |
| Precision Micro | 0.997793 | 0.997743 | 0.996890 | 0.996740 | 0.993178 | 0.974319 |
| Precision Weighted | 0.997810 | 0.997758 | 0.996914 | 0.996764 | 0.993203 | 0.974387 |
| Recall Macro | 0.997902 | 0.997602 | 0.996874 | 0.997208 | 0.993367 | 0.976212 |
| Recall Micro | 0.997793 | 0.997743 | 0.996890 | 0.996740 | 0.993178 | 0.974319 |
| Recall Weighted | 0.997793 | 0.997743 | 0.996890 | 0.996740 | 0.993178 | 0.974319 |
| F1 Macro | 0.997805 | 0.997774 | 0.996839 | 0.997209 | 0.993692 | 0.975401 |
| F1 Micro | 0.997793 | 0.997743 | 0.996890 | 0.996740 | 0.993178 | 0.974319 |
| F1 Weighted | 0.997793 | 0.997742 | 0.996891 | 0.996741 | 0.993177 | 0.974306 |

* + 1. Real-life model deployment simulation

Now, to test these models further, and to truly make sure that there is little to no overfitting and the model generalizes well enough for real-world deployment, I now used the validationData.csv which has not been touched or seen by the models yet. All three models display similar results when introduced in new datasets. The building classifiers all performed similarly to the initial train-test splits. However, the floor classifiers performed around 10% worse across all models. The decision tree floor classifier was especially bad, with only an F1 macro score of 0.7389 compared to the previous 0.9745 \ single-split F1 macro score. This suggests that the decision tree floor classifier overfitted the training data and does not represent the true model to predict the floor.

Among all three models, Random Forest performed the best when predicting a new dataset, maintaining an F1 macro score of 0.991 when predicting the building and 0.8845 when predicting the floor. These results are from our Random Forest with default parameters.

* + 1. Hyperparameter tuning

Selecting Random Forest as the best classifier for both building and floor, I aim to improve its predictions with hyperparameter tuning. So far, I was using the default parameters of the Random Forest classifier from scikit-learn. Adding the correct parameters should help the model perform better.

The hyperparameter tuning for my basic classification model will be using differential evolution. Differential evolution is a metaheuristic algorithm I learned from fundamentals of AI. I know this algorithm is good for hyperparameter tuning and in most cases perform better and maybe faster than traditional grid search. I will be applying differential evolution hyperparameter tuning. In my other class, we applied differential evolution manually without any library. But for this application, I will be using the Differential Evolution function from SciPy Optimize. As this is outside of the scope of our Machine Learning lectures, I ask the help of ChatGPT 4o to help me create the code for this. The main idea of differential evolution is that there will be a starting population, in our case it will be a starting population of multiple random forest; and to improve the population, the differences between the random forest will be adapted by each other, exploring the whole search space and ideally finding the best outcome based on the fitness function—which in my case is the F1 macro score.

The hyperparameter tuning did not improve the results. It slightly improved the predictions in the trainingData.csv for both location predictions but when used to predict for building ID on a new dataset, the tuned model performed marginally worse. This could indicate that tuning the building classifier introduced more variance even when tuning with CV. There are also very marginal improvements from tuning the random forest floor classifier, the validation F1 macro score was 0.8845, but after tuning, it's increased slightly by 0.8950.

A graph of different colored bars

AI-generated content may be incorrect.

Figure 5: Basic classification models F1-macro score comparison on test split and validation dataset

Overall, the complexity of our model kept increasing as well as the complexity of the code but there were diminishing returns as the predictions did not get much better. Our original simple model of a decision tree was already good with the trainingData.csv. If we wanted more generalization, the SVM classifiers is sufficient—consistently producing F1 scores over 87%. But now, after using an ensemble model and applying hyperparameter tuning to it, our results improved but not by that much. Overall, I am already satisfied with the results of my predictions.

* 1. Advanced Model Regression

Now moving on to the second part of my model development task, I will apply advanced regression models to predict the coordinates of a record based on the RSSI columns.

To make my code cleaner and reusable, I created some generalized functions. A data loader function prepares and feeds the data to the model during training. It handles the batching, shuffling, and conversion of arrays to tensors—making it much easier to train the model without manually looping over each sample. I also made a train regressor function, which runs training loops over epochs. A get prediction function gets the array of predictions from the model for evaluation. The same evaluation function already created in the beginning will be used. Another function to unscale error is created to convert scaled RMSE back to meters for model prediction evaluation

* + 1. Neural Network

I created a fully connected neural network (NN) class with a default hidden size of 64, a learning rate of 0.001, and a loss function using MSE that runs for 100 epochs. I used lab 7 as a guide to make it. For the initial run, again, I didn’t make it complicated and used the hyperparameters already given. The NN had an RMSE of 0.0315 for longitude and 0.0233 for latitude. Using the reverse scaling function the predictions is around 12.29 meters off in longitude and 6.32 meters off in latitude. I am not familiar enough with coordinates to know if this is acceptable but from my experience on trying to locate my device indoors, the prediction errors are around what I would see in apps like Google Maps. Beyond real life experience, the RMSE value is already very low for both regressors.

A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 6: Neural Network Regressors Actual vs Predicted

* + 1. Convolutional Neural Network

Next, I attempted a Convolutional Neural Network (CNN) for regression. I had no real experience with CNNs before this. I have run CNN models, but I never had to make them from scratch hence I needed the help of ChatGPT 4o to correct the errors in my code. In those projects, CNNs mostly perform well and so I want to see how it will work with this dataset. From the lectures, I know that CNNs are typically used for structured grid-like data to detect patterns, so I reshaped the RSSI features into a 2D vector. I used the same given hyperparameters as NN. The CNN predictions are very similar to NN. The longitudinal predictions are marginally better with 0.03038 RMSE while the game for the latitudinal predictions is a bit wider with an RMSE of 0.032361.

A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 7: Convolutional Neural Network Regressors Actual vs Predicted

* + 1. Cross validation

The cross-validation metrics are similar to the simple split test. Meaning that the model does not overfitted to the dataset and generalized as well. For NN, the RSME actually improved, only having a longitudinal RMSE value of 0.0278—better than the previous 0.03150—and a latitudinal RSME of 0.02417—beating the previous 0.0233. The CNN metrics remain similar.

Table 2: Average Cross Validation Metrics on Initial Regression Models

| Metric | NN Longitude | NN Latitude | CNN Longitude | CNN Latitude |
| --- | --- | --- | --- | --- |
| R2 | 0.9921 | 0.9901 | 0.9909 | 0.9842 |
| MSE | 0.00078 | 0.00060 | 0.00091 | 0.00097 |
| RMSE | 0.02785 | 0.02418 | 0.02996 | 0.03084 |
| MAE | 0.01966 | 0.01587 | 0.02074 | 0.02138 |

* + 1. Real-life model deployment simulation

From the CV, neural network MLP seems to be the better model among the two, and so I will try it on the untouched the validationData.csv if that is truly the case and the NN model generalizes the predictions enough. The new dataset predictions for NN longitude and latitude have RMSE of 0.0535 and 0.0538 respectively while for CNN it’s 0.0565 and 0.0675. These results confirm that NN is the better model between the two and that model still predicts well with a complete new dataset.

* + 1. Hyperparameter tuning

I now applied hyperparameter tuning to the chosen neural network model. Similar to the previous hyperparameter tuning done in the basic classification model, I applied differential evolution. I needed the help of ChatGPT 4o to create the objective functions, as when I tried to do it manually, I was making a lot of errors and so the tool helped me correct the syntax and logic of my code. After applying hyperparameter tuning for the neural network, the predictions for the validate data CSV improved a lot. The validation RMSE now beats the RMSE of the validation train test splits. The RMSE of the longitude is 0.0333, while for the latitude, it's 0.04334. That is an average error of 13.00 meters and 11.74 meters for the unseen data. Similar to the previous parts the return on adding complexity to code is minimal.

A graph of different sizes and colors

AI-generated content may be incorrect.

Figure 8: Advanced regression models RMSE comparison on test split and validation dataset

Overall, besides developing my assigned models, I contributed to keeping the evaluations consistent and reproducible for me and my partner to use.

1. Challenges faced

This project was relatively smooth sailing, although there were still some challenges I had to face. My partner and I wanted to keep predictions as similar as possible for fair comparison of models so I solved this by building a generalized loading and cleaning function that can be applied to all models. I also struggled with building CNNs from scratch and implementing differential evolution tuning, both of which I resolved by seeking help from ChatGPT 4o to debug and correct my code; looking at my previous works from the labs and other projects also helped. Another challenge was evaluating whether the models would generalize well to unseen data. To address this, I promoted the idea of setting aside validationData.csv until the final stage and applied cross-validation throughout. Finally, interpreting the regression results in real-world terms (meters) was difficult at first, but I created a function to reverse the scaling and calculate actual distance errors. I did not know how coordinates worked so I had to search how to turn the scaled coordinated to an interpretable measure. These experiences helped me gain confidence in structuring experiments, debugging ML pipelines, and interpreting evaluation metrics in practical terms.

1. Lesson learned

This project gave me several valuable lessons. It pushed me to apply my previous machine learning in an unfamiliar dataset with unfamiliar concepts that I had to research about. I was also able to experiment with deep learning models—which I have not tried before besides the labs in our machine leaning class. I also must keep the interpretability of the results, and I did this by converting the RMSE into real-world units like meters. From a technical perspective, I became more comfortable working with Scikit-learn's classifiers. They are relatively easy to use and the most help I needed was learning which function to import and from which library package. Writing code for neural networks was also confusing. I knew the concepts from the lectures but actually writing them in code was challenging but I used this as a learning opportunity.

Working with a partner when coding is new to me. I have worked with others before but usually I would be working as an intern, or we would be assigned different parts without much overlap but the evaluation for our two parts each had to be consistent and so one of the biggest lessons I learned was the importance of reproducibility and standardization. Creating reusable functions for data loading, preprocessing, and evaluation saved me a lot of time and helped maintain consistency—especially when collaborating with Asma.

The documentation of the use of AI tools made me more conscious of how I used it. I haven’t written a report in a while with very minimal AI use. I learned to use these tools as help instead of primarily using them. Even when using ChatGPT 4o, the code it gave is very messy and it kept rewriting functions I already had. I sometimes found it better to get the idea from it but write the code myself. If there is an error, I ask it to debug which works well with my workflow and coding style. I would definitely use the same system in other projects.

The biggest lesson is gravitative towards simplicity over complexity. Applying the deep learning models were good practice but the results are not much better the simple models. Even after implementing CNNs and applying differential evolution for hyperparameter tuning, the improvements were often marginal. This made me more critical about whether added complexity is justified, especially when simpler models like SVM and random forest were already performing well.

USE OF A.I. Tools

While the use of A.I. tools for this project is kept to a minimum, these tools are in the following aspects: creating the pair plots in data exploration, using the stack method in the histograms of RSSI values, combining different plots in subplots with idx, and iterating over rows for detecting RSSI in WAP columns are assisted by OpenAI’s GPT 4o model. The formatting of references is also aided by online A.I. tools like Scribbr’s citation generator and, again, GPT4o.

The second part of the project involved the help of ChatGPT 4o in correcting errors in the syntax and working with library functions specifically in writing code for Differential Evolution for both basic and advanced models, debugging and correcting CNN implementation, and creating the objective function for tuning CNN.

REFERENCES

1. Magda Chelly and Nel Samama. 2009. New techniques for indoor positioning, combining deterministic and estimation methods. In Proceedings of the European Navigation Conference - Global Navigation Satellite Systems (ENC-GNSS 2009), Naples, Italy, 1–12. <https://hal-01367483>.
2. E. Cheng. 2025. Location fingerprinting—What is it, and should you choose it as your IPS technology? Pointr. Retrieved February 5, 2025, from <https://www.pointr.tech/blog/location-fingerprinting-what-is-it-should-you-choose-it>.
3. J. Torres-Sospedra, R. Montoliu, A. Martinez-Us, T. Arnau, and J. Avariento. 2014. UJIIndoorLoc, UCI Machine Learning Repository. <https://doi.org/10.24432/C5MS59>.
4. Android Open Source Project. n.d. Wi-Fi AP/AP concurrency. Android Open Source Project. Retrieved from <https://source.android.com/docs/core/connect/wifi-ap-ap-concurrency>.
5. MetaGeek. n.d. Wi-Fi Signal Strength Basics. MetaGeek. Retrieved from <https://www.metageek.com/training/resources/wifi-signal-strength-basics/>.
6. M. Pierce. n.d. WiFi Signal Strength: A No-Nonsense Guide. TechGrid. Retrieved from <https://techgrid.com/blog/wifi-signal-strength>
7. N. Lang. 2024. What is the MinMax Scaler? Data Basecamp. Retrieved February 16, 2025, from <https://databasecamp.de/en/ml/minmax-scaler-en>.
8. IBM. 2024. What is PCA? Retrieved February 16, 2025, from <https://www.ibm.com/think/topics/principal-component-analysis>.

1. [↑](#footnote-ref-1)