Implementation of K-Nearest Neighbors (July 2024)

Kristina Ghimire (THA077BCT023)[1], Punam Shrestha (THA077BCT038)[1]

[1]Department of Electronics and Computer Engineering, IOE, Thapathali Campus, Kathmandu 44600, Nepal

Corresponding author: Kristina Ghimire(ghimirekristina10@gmail.com)

ABSTRACT
This study explores the application of K-Nearest Neighbors (KNN) classifiers on Room Occupancy Estimation datasets to predict the number of people in a room using sensor data. The research involves normalizing the data before analysis and experimenting with different distance metrics to optimize the number of neighbors (k) for accurate predictions. The results demonstrate that KNN effectively estimates room occupancy based on sensor data. Specifically, by adjusting class distributions and using Manhattan distance, the KNN model achieved a perfect accuracy of 100%. Even without class adjustments, using Manhattan distance still achieved a high accuracy of 99%, outperforming Euclidean distance, which is a specific form of Minkowski distance. The study evaluates both default and weighted KNN models to compare their performance, highlighting KNN's ability to analyze diverse datasets and provide reliable predictions.

INDEX TERMS Euclidean, K-Nearest Neighbors, Manhattan, Minkowski, Normalization, Resampling.

I INTRODUCTION

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm used for classification and regression. It predicts the correct class for test data by measuring the distance between the test data and all training points, then selects the K nearest points. It is a type of instance-based learning, or lazy learning, where the algorithm makes predictions based on the distance between a given data point and its nearest neighbors in the training dataset. For regression, the predicted value is the average of the K-selected training points. KNN is powerful, simple, and versatile, but it can struggle with large, high-dimensional, and imbalanced datasets. Proper data preprocessing, feature selection, and parameter tuning are essential to maximize its performance.

KNN is easy to understand and implement. It is a lazy learning algorithm, meaning there is no explicit training phase or model to build. The computation happens during the prediction phase, which can benefit small datasets. It works well for multi-class problems and non-linear decision boundaries. It can handle various types of distance metrics (e.g., Euclidean, Manhattan) and can be adapted to different kinds of data (numerical, categorical). Choosing k (number of neighbors) allows flexibility in balancing bias and variance. The algorithm can also be adapted with different weightings (e.g., weighted KNN).

However, distance calculations are performed for each query point against all training points, which can be slow and resource-intensive, especially for large datasets. KNN stores all training data, requiring significant memory, particularly with large datasets. Performance can degrade if the dataset contains irrelevant or redundant features, making proper feature selection and scaling crucial. Features with larger scales can dominate the distance metric, so normalization or standardization is often necessary. The performance of KNN heavily depends on the choice of k and the distance metric, often requiring cross-validation and experimentation. In high-dimensional spaces, distance becomes less meaningful, and KNN can suffer from poor performance. Dimensionality reduction techniques like PCA (Principal Component Analysis) might be needed. Since KNN is a lazy learner, it defers computation until prediction time, which can be slow compared to algorithms that build a model during the training phase. It can also be biased towards the majority class in imbalanced datasets, possibly requiring techniques like resampling or adjusting class weights.

KNN can be used for different applications like image recognition based on similarity with recognized images, recommender systems to suggest similar products to the user, and so on. The study of KNN can be useful to know how data are classified and grouped based on similarity. Most importantly KNN shows how supervised learning works to make predictions based on trained datasets. This study gives you a complete overview of the working of KNN, different variants and parameters of KNN, and data preprocessing methods used in KNN.

II RELATED WORK

KNN is widely studied and used in classification and regression problems.

Claudia Chițu et al. [1] explore methods for predicting building occupancy. The authors employ supervised learning techniques to analyze data and make estimations about how many people are likely present in a building at any given time. They discuss their approach and findings, focusing on using data-driven models to improve accuracy in occupancy predictions. The paper contributes insights into practical applications of machine learning in building management and occupancy monitoring.

Cunningham et al. [2] focus on k-nearest Neighbor (k-NN) classifiers, a fundamental machine learning algorithm. It includes practical Python examples to illustrate how k-NN works and how it can be implemented for various classification tasks. The authors likely discuss the principles behind k-NN, where predictions are made based on the majority class of nearby data points in a feature space. They may cover topics such as selecting the optimal value of k, handling different distance metrics, and evaluating classifier performance. This paper serves as a useful resource for both beginners and practitioners looking to understand and apply k-NN in real-world scenarios using Python programming.

These works contribute to the understanding and use of KNN to make more accurate predictions based on given features.

III METHODOLOGY

A DATASET DESCRIPTION

The evaluation of KNN for classifying unseen data involved experiments focused on Room Occupancy Estimation. This dataset aims to accurately predict the number of people in a room using various sensors: 4 temperature sensors for Celsius degrees, 4 light sensors for Lux, 4 sound sensors for Volts, 1 sensor for parts per million (PPM), a CO_2 slope sensor, and 2 binary PIR sensors for motion detection. It includes 18 features in total, with the target being the Room Occupancy count. While the dataset includes date and time features useful for time-based analysis, these were excluded from the classification task. The study explores different approaches to enhance KNN's prediction accuracy on this dataset.

B THEORETICAL FORMULATION

K-Nearest Neighbor is the use of different distance or similarity metrics to find the closeness between the instances. The closeness will depend on the value of K. So, this algorithm is simple, easy to use, and powerful for finding similarities. This algorithm is not trained on data, but each time similarity between the instances is checked to make predictions. To obtain better accuracy, it is important to use the best distance metrics. The data must be normalized to reduce the impact of high-ranged data on the prediction.

1 KNN Algorithm

Step-1: Determine parameter K.

Step-2: Calculate the distance between the test instance and all the training instances.

Step-3: Sort the distances and determine K-nearest neighbors.

Step-4: Gather the labels of the K-nearest neighbors.

Step-5: Use simple majority voting or weighted voting.

The algorithm begins by determining the parameter K, which specifies the number of neighbors to consider. It then calculates the distance between the new data point and each point in the labeled training dataset using a chosen distance metric, such as Euclidean or Manhattan distance. After computing these distances, the algorithm sorts them to identify the K-nearest neighbors. For classification tasks, KNN assigns the class label based on the majority class among these neighbors. For regression tasks, it predicts the value by averaging the values of the K-nearest neighbors. The choice of K significantly impacts the algorithm's performance: smaller K values can lead to overfitting, making the model too sensitive to noise, while larger K values might overlook local patterns, resulting in less accurate predictions.

2 Variants of KNN

Default KNN

The default version of KNN uses a straightforward approach. Initially, it involves selecting the parameter k, typically an odd number to avoid ties in voting. The algorithm then calculates the distance from the new data point to all points in the training dataset. After identifying the k nearest neighbors, it makes a prediction based on majority voting for classification tasks or averaging for regression tasks. The decision boundaries created by default KNN closely follow the data distribution in the feature space. While this enables the algorithm to capture complex patterns, it also may lead to noise. default KNN uses parameters without additional weighting, ensuring simplicity and ease of use.

Weighted KNN

Weighted KNN modifies the default approach by assigning weights to neighbors based on their distance to the query point. This variant of KNN begins by setting k and creating a function for distance metrics which calculates the distances and assigns weights to these distances. Weighting can be randomly initialized or calculated as the inverse of distance. For classification tasks, the final prediction is made through weighted majority voting, where closer neighbors have a stronger influence. For regression tasks, the prediction is made by weighted averaging of the k nearest neighbors' target values. Weighted KNN is advantageous as it helps smooth decision boundaries and reduces the impact of outliers and noisy data.

3 Choosing the value of K

The selection of k is the most important step to enhance the performance of the KNN algorithm. The value of k determines the number of neighbors considered when making predictions. A smaller k results in more flexible decision boundaries but can be overly sensitive to noise and outliers, leading to overfitting due to which the model gives less accurate results. Conversely, a larger k provides smoother decision boundaries but may fail to capture local patterns effectively, resulting in underfitting. Therefore, it is essential to choose k carefully to achieve a balance between bias and variance. There exist formal ways to select K. One way is the rule of thumb which is to take k as the square root of the number of examples. Also, k=1 is often used for better performance but it can be sensitive to noise and may lead to overfitting. Another way is K-fold cross-validation.

Cross-validation is a widely used technique to select the optimal value of k in KNN. It involves dividing the dataset into multiple training and validation sets, allowing the algorithm to be evaluated on different subsets of data. In our case, 10 cross-folds are used. By performing cross-validation, one can assess how different values of k perform on unseen data. This iterative process helps in identifying the k that offers the best generalization performance, ensuring that the model neither overfits nor underfits the data. Visualizing the accuracy for different k-neighbor values helps to identify the best k-value for the highest accuracy.

C MATHEMATICAL FORMULAE

1 Distance Metrics

Distance metrics are the main step in KNN for measuring similarity and determining the nearest neighbors. Distance metrics help in knowing the importance of different features. For example, if a feature has more impact on the prediction, it will affect the distance more and thus have a greater influence on the classification. There are different distance metrics which are listed below:

Minkowski Distance

Minkowski distance is a generalization encompassing both Euclidean and Manhattan distances, offering flexibility through the parameter p. This metric can be adapted to different data characteristics, influencing its behavior from Manhattan-like(when $p = 1$) to Euclidean-like(when $p = 2$) based on p. Understanding these trade-offs is crucial for applying Minkowski distance effectively in various machine learning tasks. The formula of Minkowski distance is shown in equation 1.

Euclidean Distance

Euclidean Distance, also known as L2 distance or L2 norm is a commonly used distance metric in machine learning, defined as the straight-line distance between two points in Euclidean space. It provides a clear geometric interpretation of the distance between points, which aids in understanding data structure. However, it requires careful preprocessing like normalization, and may not perform well in high-dimensional or categorical data spaces. The formula of Euclidean distance is shown in equation 3.

Manhattan Distance

Manhattan Distance, also known as L1 distance, or L1 norm, or city block distance, is a metric where the distance between two points is the sum of the absolute differences of their Cartesian coordinates. Its simplicity and computational efficiency make it suitable for certain applications. It is less affected by outliers compared to Euclidean distance but may not capture relationships in continuous spaces as effectively. The formula of Manhattan distance is shown in equation 2.

Hamming Distance

Hamming distance measures the difference between two strings of equal length, making it suitable for binary or categorical data. It counts the number of positions at which corresponding symbols differ. While straightforward for categorical data, it's limited by its requirement for equal-length strings and sensitivity to feature scaling in more complex data scenarios. The formula of Hamming distance is shown in equation 4.

2 Evaluation Metrics

For evaluation, different metrics are used which are given below:

Precision

Precision measures the accuracy of positive predictions, reflecting the ratio of true positives to total predicted positives. High precision indicates reliable positive predictions with few false positives, essential for applications sensitive to incorrect classifications. The formula of precision is shown in equation 5.

Recall

Recall (sensitivity) measures the proportion of true positive predictions relative to all actual positives. Balancing recall with precision is crucial; high recall ensures identifying most positives but may increase false positives, impacting overall model performance. The formula of recall is shown in equation 6.

F1 Score

The F1 score combines precision and recall into a single metric using their harmonic mean. A higher F1 score indicates a balanced performance between precision and recall, making the model suitable for applications requiring both accurate identification of positives and low false positives. The formula of the f1 score is shown in equation 7.

Macro Average

Macro average calculates the mean performance metric (e.g., precision, recall, F1 score) across all classes in multi-class classification, treating each class equally. A higher macro average signifies consistent performance across diverse categories, indicating strong generalization capabilities. The formula of the macro average is shown in equation 8.

Weighted Average

Weighted average considers class distribution by calculating a mean weighted by instances in each class. Higher weighted averages indicate robust model performance across all classes, demonstrating the ability to handle imbalanced data and make accurate predictions. The formula of the weighted average is shown in equation 9.

Accuracy

Accuracy measures correctly predicted instances relative to the total number of instances in a dataset. High accuracy indicates effective model performance suitable for real-world applications, accurately predicting both positive and negative instances. The formula of accuracy is shown in equation 10.

D DATA PREPROCESSING

1 Rectifying class imbalance

In datasets, classes aren't always balanced. Some classes might have many more examples than others, causing models to favor those classes during predictions due to their sheer numbers. To fix this, we can balance the classes by resampling the data. Upsampling involves increasing the number of instances in the minority class, while downsampling decreases instances in the majority class. This ensures that each class contributes equally to the model training, improving its ability to predict all classes accurately, not just the majority.

2 Normalization of data

Normalization involves scaling numerical data into a standardized range. There are several normalization methods. Z-score normalization adjusts data to have a mean of 0 and a standard deviation of 1. On the other hand, min-max scaling transforms data to fit within a fixed range, typically between 0 and 1. These techniques ensure that data from different scales can be compared directly and are essential for many machine learning algorithms to perform effectively and efficiently. Normalization is an important part of models like K-Nearest Neighbors because all features are used for evaluation and large-ranging features will affect the model evaluation even though it has less correlation with target data.

IV INSTRUMENTATION DETAILS

Python is widely recognized for its simplicity and readability as a popular interpreted programming language that executes code line by line, aiding in quick error detection and resolution without the need for variable declarations before assignment. JupyterLab serves as an interactive web-based environment supporting multiple programming languages, prominently Python, allowing users to develop and share code, equations, visualizations, and textual content, particularly favored in data science for its flexibility and intuitive interface. Matplotlib, a robust library, facilitates the creation of diverse plots such as lines, scatter plots, histograms, and more, offering extensive customization options suitable for producing high-quality visualizations in data analysis and presentations. Seaborn, built upon Matplotlib, specializes in crafting informative and visually appealing statistical graphics that seamlessly integrate with Pandas data structures, ideal for exploring complex datasets and generating elegant statistical plots. It simplifies tasks like visualizing variable relationships, analyzing data distributions, and comparing categories through its diverse functions and customizable settings. In our classification tasks, we employ sklearn's train_test_split for dividing datasets into training and testing subsets, alongside metrics for evaluating model performance with metrics like accuracy score, classification report, and confusion matrix. Utilizing sklearn's neighbors, specifically KNeighborsClassifier, we train our KNN model, optimizing the choice of k through k-fold cross-validation provided by sklearn's model_selection.

V EXPERIMENTAL RESULTS

A Normalization of Dataset

The dataset comprises a variety of features with different ranges, which can significantly impact the performance of our model. To ensure that all features are on a similar scale and to standardize them, we fit the features into the StandardScaler model and transform them accordingly.

The correlation between the features in the dataset is illustrated in figure 1. Additionally, the correlation of different features with the target variable is presented in figure 2. The feature S1_light exhibits a high correlation with a value of 0.85. Following this, S2_light and S3_light show correlations of 0.79. Moreover, S1_temp correlates with 0.7, and S7_PIR is correlated at 0.69. This correlation information is effectively visualized in a bar plot, as shown in figure 3.

B Using Default KNN

To evaluate the KNN model, we begin by splitting the dataset into 80% training data and 20% testing data. We then utilize the default KNN model to train our model. The default parameters are as follows: n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, and metric='minkowski'. Here, the number of neighbors to check for making a prediction is 5. The term 'uniform' indicates that all the points within each neighborhood are weighted equally. The 'auto' setting allows the algorithm to determine the most suitable approach, which can be either 'ball_tree', 'kd_tree', or 'brute'. The leaf_size is set to 30, representing the number of data points in each grouping for ball_tree or kd_tree. This parameter can influence the speed of searching and the memory required to store the tree. The parameter p refers to the power parameter for Minkowski distance, where p=2 denotes the use of Euclidean distance. Minkowski is the default distance metric employed.

After training the model on the training dataset, we test it on the test dataset and obtain the results, which are shown in table 4. Given that the room occupancy count of 0 has the highest count of 1668 in the training dataset, we achieved 100% precision, recall, and f1 score for this class. For class 1, we achieved a precision of 0.98, a recall of 0.95, and an f1 score of 0.96. Similarly, for class 2, the precision was 0.96, the recall was 0.95, and the f1 score was 0.95. For class 3, the precision was 0.93, the recall was 0.99, and the f1 score was 0.96. Consequently, the overall accuracy of the model was 0.99. The macro average precision was 0.97, and the weighted average was 0.99. The macro average recall was also 0.97, with a weighted average of 0.99. The macro average f1 score was 0.97, and the weighted average was 0.99.

The confusion matrix, displayed in figure 4, shows that count 0 has the highest true positive count of 1665. The ROC curve of the model, visualized with an AUC of 1.00, is shown in figure 7. The precision-recall curve for all the classes (0, 1, 2, and 3) is depicted in figure 6, where class 0 exhibits a precision-recall area of 1.00. To determine the optimal number of neighbors (k) for the highest accuracy, we plotted accuracy scores for different k-neighbors, as shown in figure 5.

C Using KNN with Manhattan Distance

In this section, we use the KNN model with Manhattan distance to train the dataset. All the parameters are the same as the default KNN with p=1 denoting the use of Euclidean distance. and we train the KNN model.

Upon training the KNN model with the p=1 parameter, we achieve a model with perfect performance, boasting 100% weight average precision, recall, and f1 score. Thus, the overall accuracy of the model is also 100%, as displayed in table 2. Although the confusion matrix, shown in figure 8, indicates that 7 instances are misclassified, this number is almost negligible and rounds down to 0.00.

Additionally, the precision-recall curves for classes 0, 1, 2, and 3 are illustrated in figure 10, with class 0 showing a precision-recall area of 1.00. The accuracy plot shows fluctuating accuracies over different k-neighbors which is shown in figure 9.

D Using weighted KNN

In this section, we use the weighted KNN model to train the dataset. The default parameters are n_neighbors=5, weights='uniform', algorithm='auto', and leaf_size=30. To add a customized touch, we employ a user-defined function for the metric parameter, which calculates the distance using random weights for each feature and the points. The random weights list is provided as a dictionary parameter in metric_params. Using these settings, we train the KNN model.

Upon training the KNN model with the default parameters, we achieve a model with perfect performance, boasting 100% precision, recall, and f1 score across all classes. Thus, the overall accuracy of the model is also 100%, as displayed in table 5.

The confusion matrix for the weighted KNN model is shown in figure 11, where count 0 again has the highest true positive count of 1665. The ROC curve of the model, with an AUC of 1.00, is presented in figure 13. Additionally, the precision-recall curves for classes 0, 1, 2, and 3 are illustrated in figure 12, with class 0 showing a precision-recall area of 1.00.

E Using K-fold validated K-value for KNN

In this section, we use the KNN model after selecting the k value using k-fol cross-validation to train the dataset. We train the KNN model using the best k-value. The cross-validation is done using 10 folds and the accuracy for different k-values is visualized in a plot which is shown in figure 21. The best k-value is 11. So, using this model is trained.

Upon training the KNN model, we achieve a model with 99% precision, recall, and f1 score having a weighted average and 96% having a macro average. Thus, the overall accuracy of the model is 99%, as displayed in table 3.

The confusion matrix for the weighted KNN model is shown in figure 20, where count 0 again has the highest true positive count of 1665.

F After Resampling

The dataset initially contains a majority of instances with a count of 0. To address this imbalance, we need to resample the data. The plots before and after resampling, depicting the majority class, are shown in figures 14 and 15, respectively. Due to the resampling process, there are some changes in the correlation between different features and the target variable, which is shown in figure 16. In this resampled dataset, the feature S3_Temp now shows the highest correlation of 0.74, followed by S3_light with a correlation of 0.72.

The precision-recall curves for all classes (0, 1, 2, and 3) are depicted in figure 19, with class 0 demonstrating a precision-recall area of 1.00. To identify the best number of neighbors (k) for achieving the highest accuracy, we plotted accuracy scores for different k-neighbors, as shown in figure 18.

Comparing the accuracy of KNN models using different distance metrics as shown in figure 22, the Manhattan distance KNN and k-fold validated KNN performs well without resampling the data, and the resampled KNN model outperforms all the models.

VI DISCUSSION AND ANALYSIS

Before resampling, the dataset predominantly consists of instances where the room occupancy count is 0. This causes the models to heavily favor predictions of 0 due to the sheer number of such instances. The dataset has four types of room occupancy counts. Resampling is crucial to balance the class counts. Initially, there were over 8000 instances with a room occupancy count of 0, while the counts for the other types were fewer than 1000. To balance the dataset, we upsampled the instances of the other occupancy counts to match the count of 0, increasing the total instances from 10,000 to 32,000. This significant increase in size can improve the model's accuracy, but it also risks leading to overfitting.

When examining the correlation matrix before and after resampling the imbalanced class, we observe a shift in the correlation of features with the target. Initially, the top five correlated features were S1_light, S3_light, S2_light, S1_Temp, and S7_PIR. The ”S” denotes the sensors, as shown in the correlation matrix. Before resampling, sensors capturing data related to light had a high impact on the room occupancy count. After resampling, these sensors no longer had a dominant influence on the target values. Instead, all features, including light and temperature data, had a more balanced impact, as shown in the post-resampling correlation matrix.

When we look at the confusion matrices for all models, we notice significant changes. Before resampling, the confusion matrices for default KNN, KNN with Manhattan distance, KNN using k-neighbors from the k-fold cross-validation method and weighted KNN look similar. The room occupancy count of 0 had the maximum true positives (1667) due to its larger size. Counts for occupancy levels 2 and 3 had around 120 true positives, while level 1 had around 90. After resampling, the true positives for all occupancy counts became more balanced.

Using weighted KNN instead of the default Euclidean distance KNN caused a slight decrease in the accuracy rate from 99% to 98%. This decrease might be attributed to the random weights assigned to each feature, potentially assigning less weight to the most important features. Besides accuracy, the macro average precision, recall, and F1 score also decreased from 97% to 93%. But, using Manhattan distance KNN has the highest accuracy of 100% compared to both weighted KNN and default Euclidean distance KNN. Besides accuracy, the weighted average precision, recall, and F1 score also achieved 100%. However, resampling improved the accuracy rate from 99% to 100%. Initially, despite having 10,000 instances, the imbalance in the classes caused lower accuracy. Resampling improved the model’s performance, as reflected in the accuracy, precision, recall, and F1 score, all of which reached 100% for the resampled data. KNN model using k-neighbor value selected from cross-validation model performed better than other weighted KNN and default Euclidean distance KNN but less than manhattan distance KNN.

The ROC curve for both the imbalanced and resampled classes showed an AUC of 1, indicating a perfect true positive and true negative rate. The model achieved high accuracy for both scenarios. Without resampling, an accuracy of 99% was observed, which is already high. The weighted KNN also showed an AUC of 1.

The precision-recall curve for the multiclass problem was obtained using the one-vs-rest approach. For this dataset, four precision-recall curves were plotted for the four different classes, as shown in the precision-recall curves. There was a slight difference in the area covered by these curves for different KNN models, including those with default parameters and those with random feature weights. However, after resampling, the precision-recall curves for all classes showed a perfect area of 1.00. Using KNN with Manhattan distance shows a better precision-recall curve for all classes.

The accuracy for different values of k can be visualized for both the default KNN and the KNN model after resampling. Observing the plot, $k = 1$ had the highest accuracy at 99.3%, while $k = 2$ had 99.2%. However, using a low k value makes the predictions highly sensitive to noise, which can lead to overfitting. After the cross-validation method, the best value of k is found to be 11 which is used for model training. This is a good value of k which is sensitive to data and also prevents overfitting.

Comparing the accuracy of models using different distance metrics, Manhattan distance KNN performed well. But, among all the models, the model-trained resampled data achieved the best result.

In summary, the initial dataset was heavily imbalanced, with a predominant number of instances having a room occupancy count of 0. This imbalance skewed model predictions towards 0. Resampling balanced the dataset by upsampling the less frequent occupancy counts, significantly increasing the dataset size from 10,000 to 32,000 instances. This balancing improved model accuracy but also posed a risk of overfitting due to the enlarged dataset. The correlation matrix analysis showed that resampling reduced the dominance of light-related sensors on the target variable, balancing the impact of all features. The confusion matrix comparisons revealed that resampling balanced the true positives across all occupancy counts. Using weighted KNN resulted in a slight decrease in accuracy and other metrics, possibly due to sub-optimal feature weighting. But, using Manhattan distance KNN improved the accuracy of the model without the need of resampling. Nonetheless, resampling led to perfect scores across all metrics, demonstrating its effectiveness in improving model performance. The ROC and precision-recall curves supported these findings, showing that resampling provided a more balanced and accurate model. Finally, while a low k value can yield high accuracy, it also increases sensitivity to noise, underscoring the need for careful selection of k to avoid overfitting.

VII CONCLUSION

This study aimed to investigate the impact of different distance metrics for the k-nearest neighbors (KNN) model. We evaluated the model's accuracy using various distance metrics. Before training, the dataset was normalized. To assess the effect of resampling, we trained models both with and without resampling the data. Our experiments showed that the Manhattan distance metric produced the highest accuracy among Euclidean and weighted distances when data was not resampled. However, the model with resampled data achieved 100% accuracy, outperforming all other models. These results underscore the importance of choosing the right distance metric for better performance.

These findings are based on the Room Occupancy estimation dataset and may vary with different datasets. Additionally, there are many ways to preprocess data to improve performance. Besides distance metrics, other parameters of the KNN model can also be evaluated to enhance the model's performance. Visualization tools like ROC curves, precision-recall curves, confusion matrices, and accuracy plots for different k-neighbors were helpful in the evaluation process. The results also indicate that the model is overfitted, suggesting that further steps are needed to generalize the model.

In conclusion, this study highlights the importance of distance metrics in classification accuracy. It provides a strong foundation for further exploration of the KNN algorithm and its applications. Future research could investigate additional distance metrics, different datasets, and other preprocessing techniques to optimize the KNN model. Additionally, exploring methods to prevent overfitting and generalize the model across various contexts would be valuable.

VIII APPENDICES

A Equations

Minkowski Distance

$$\text{d(x, y)} = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} \tag{1}$$

Manhattan Distance or L1 norm

The special case of Minkowski distance with $p = 1$.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

Euclidean Distance or L2 norm

The special case of Minkowski distance with $p = 2$.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}$$

Hamming Distance

$$d(x, y) = \sum_{i=1}^{n} (1 - \delta(x_i, y_i)) \tag{4}$$

Precision

$$\text{Precision} = \frac{\text{True Positives}}{\text{Predicted Positives}} \tag{5}$$

Recall

$$\text{Recall} = \frac{\text{True Positives}}{\text{Actual Positives}} \tag{6}$$

F1 Score

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

Macro Average

$$\text{Macro Average} = \frac{1}{J}\sum_{j=1}^{J} \text{Metric}_j \tag{8}$$

Weighted Average

$$\text{Weighted Average} = \sum_{j=1}^{J} \frac{N_j}{N} \cdot \text{Metric}_j \tag{9}$$

Accuracy

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}} \tag{10}$$

where,

True Positive = Model predicted correctly for true data
False Positive = Model predicted incorrectly for true data
True Negative = Model predicted correctly for false data
False Negative = Model predicted incorrectly for false data
Metric can be either precision, recall, or F1 score.

B Figures

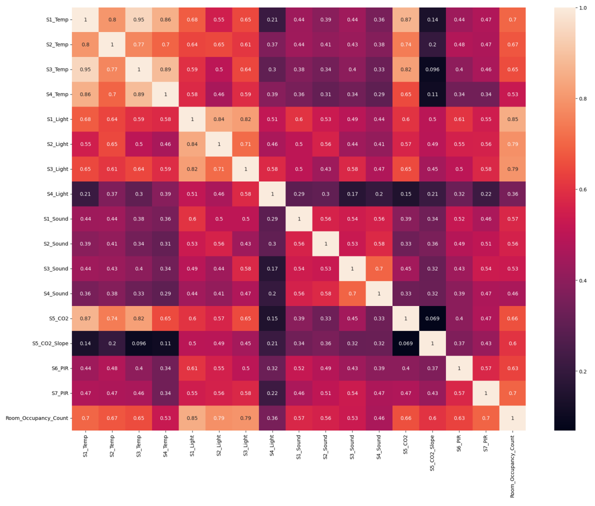

Figure 1: Correlation of Dataset

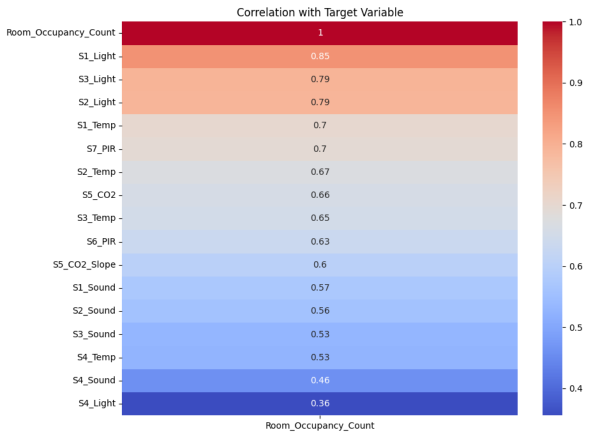

Figure 2: Correlation of Dataset with Target

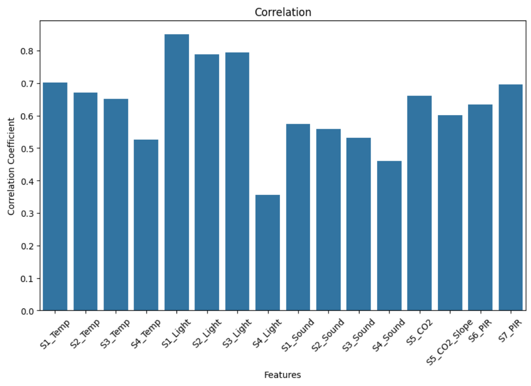

Figure 3: Barplot of Correlation of Dataset with Target

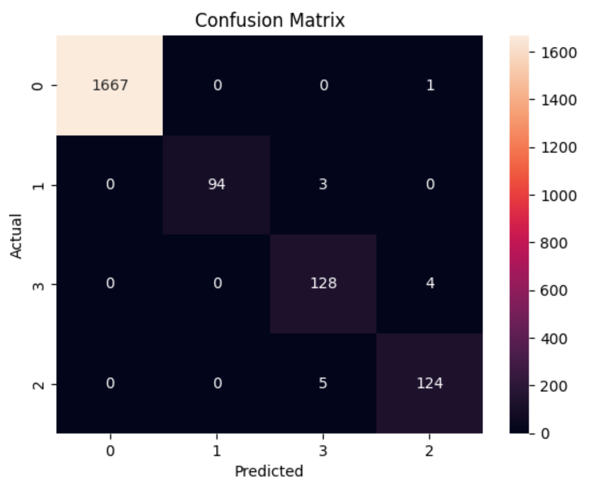

Figure 4: Confusion Matrix using Default KNN

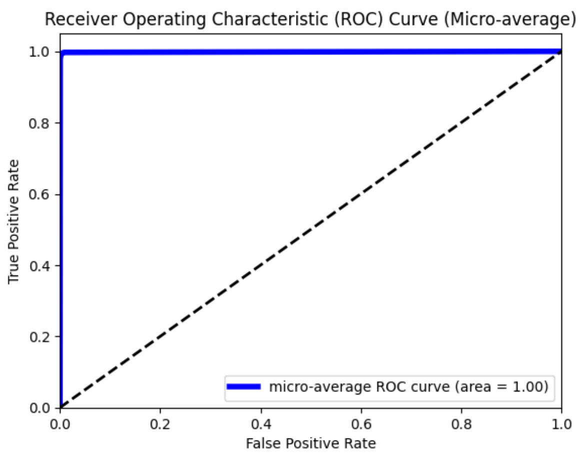

Figure 7: ROC curve using default KNN

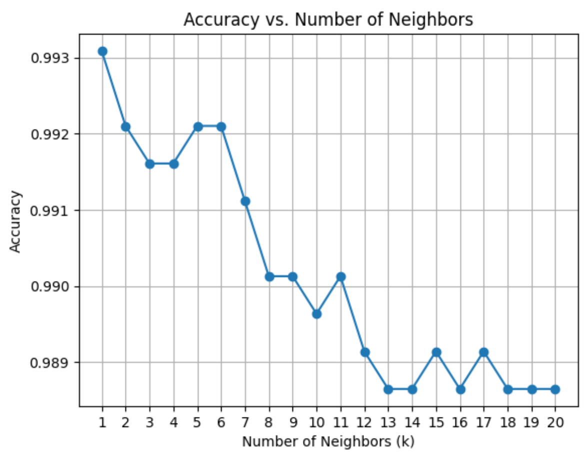

Figure 5: Accuracy for different k-neighbors

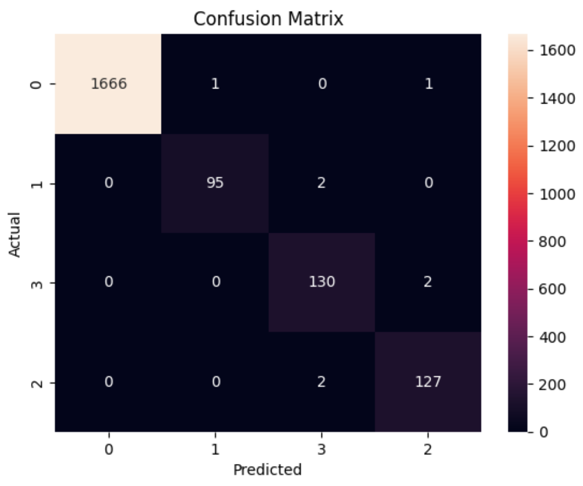

Figure 8: Confusion Matrix using KNN having Manhattan distance metric

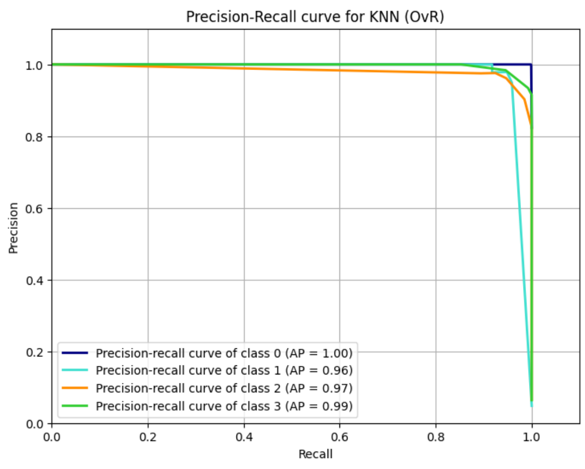

Figure 6: Precision-recall curve using default KNN

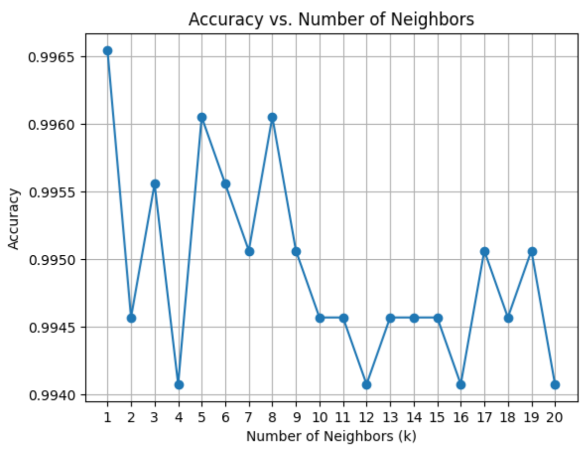

Figure 9: Accuracy for different k-neighbors using KNN having Manhattan distance metric

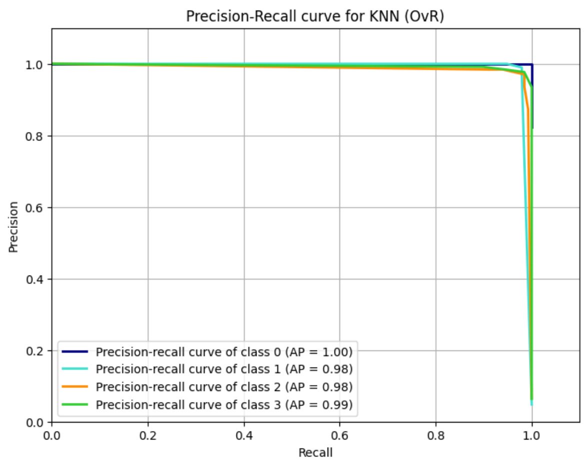

Figure 10: Precision-recall curve using KNN having Manhattan distance metric

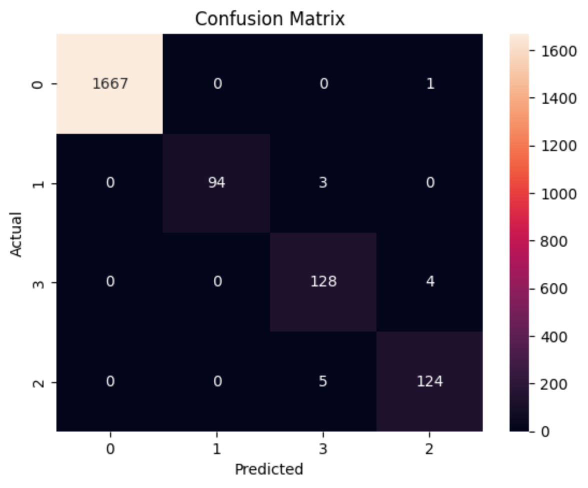

Figure 11: Confusion Matrix using random features weight

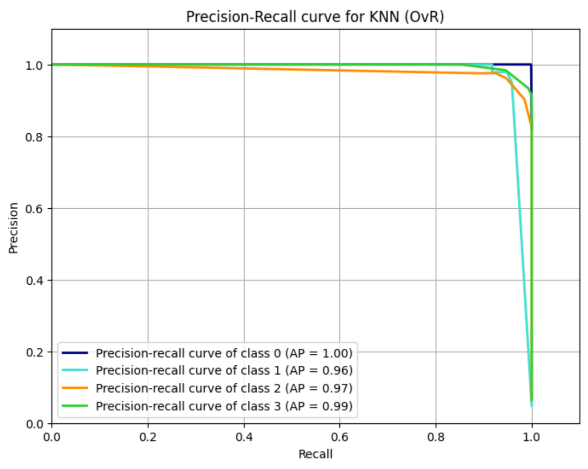

Figure 12: Precision-recall curve using random features weight

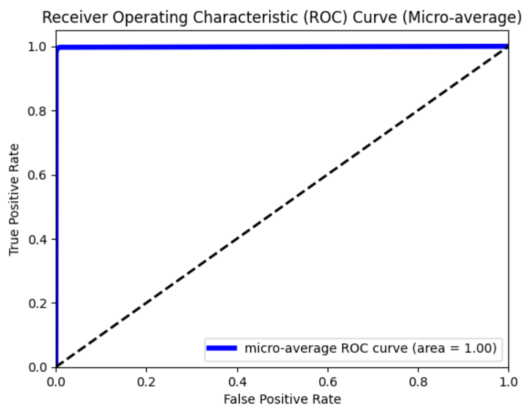

Figure 13: ROC curve using random features weight

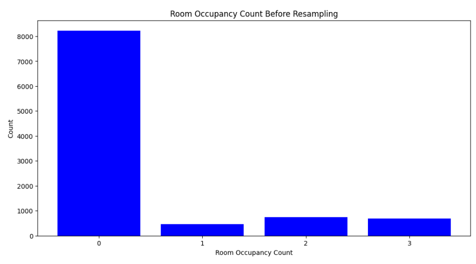

Figure 14: Room Occupancy Count Before Resampling

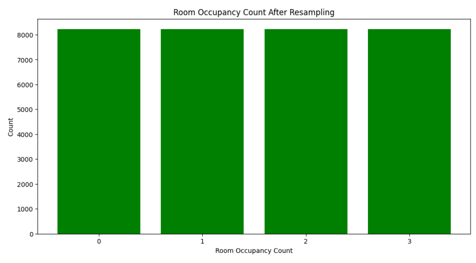

Figure 15: Room Occupancy Count After Resampling

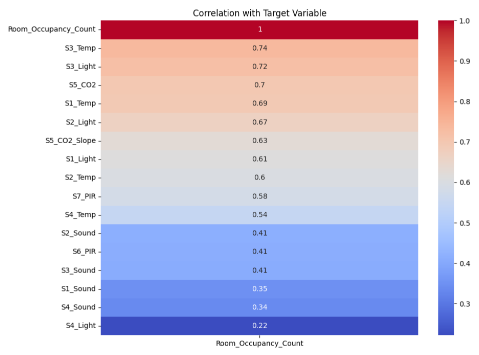

Figure 16: Correlation of Dataset features with target after resampling

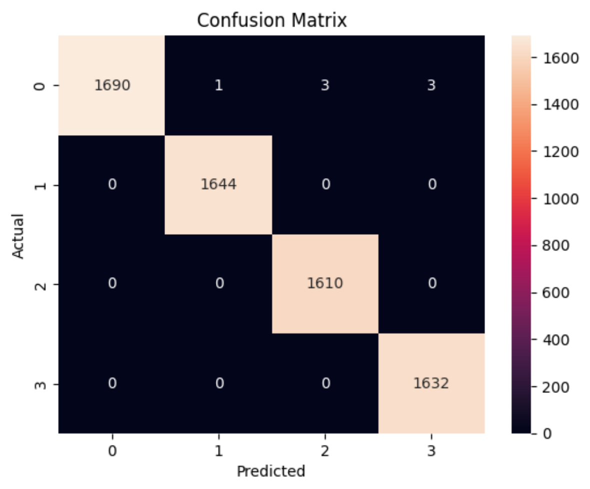

Figure 17: Confusion matrix after resampling

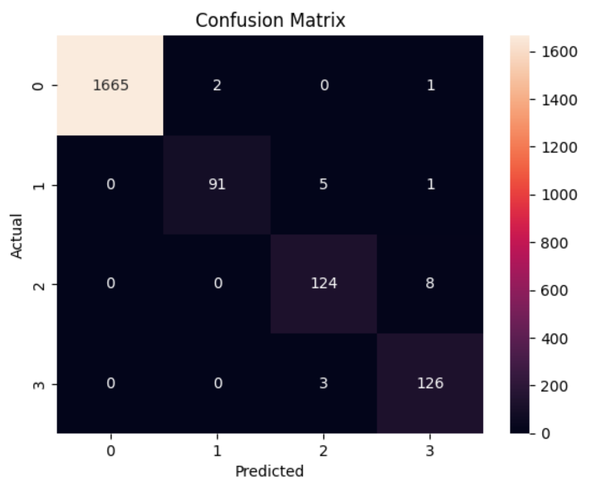

Figure 20: Confusion Matrix using k-fold validated k-neighbors

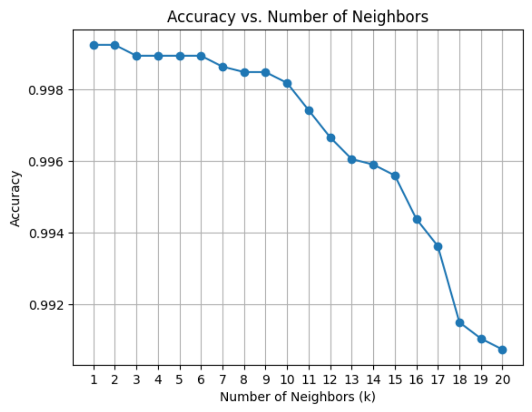

Figure 18: Accuracy plot after resampling

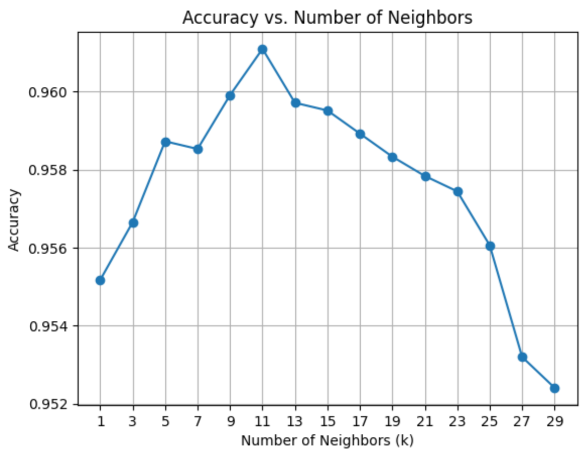

Figure 21: Accuracy for different k-neighbors using k-fold cross validation

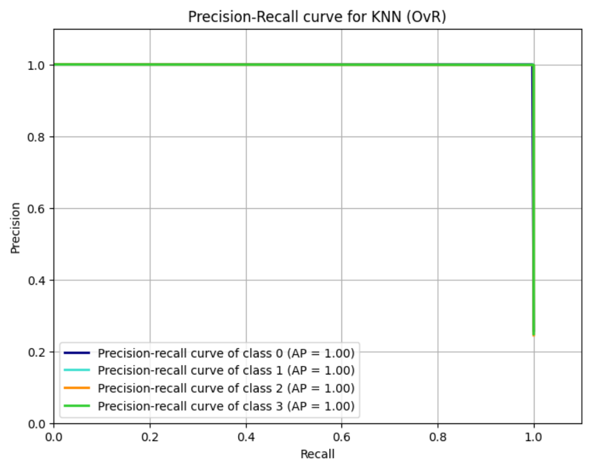

Figure 19: Precision-recall curve after resampling

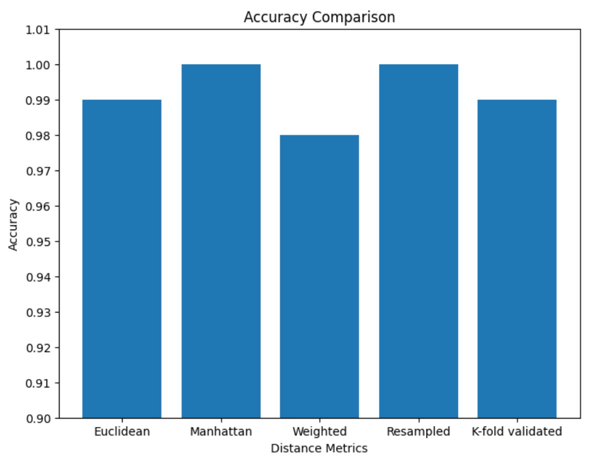

Figure 22: Accuracy Comparison

C Tables

Table 1: Default KNN Classification report

Room Occupancy	Precision	Recall	F1 score
0	1.0	1.0	1.0
1	0.98	0.95	0.96
2	0.96	0.95	0.95
3	0.93	0.99	0.96
accuracy			0.99
macro avg	0.97	0.97	0.97
weighted avg	0.99	0.99	0.99

Table 2: Classification report using KNN having Manhattan Distance metric

Room Occupancy	Precision	Recall	F1 score
0	1.0	1.0	1.0
1	0.99	0.98	0.98
2	0.97	0.98	0.98
3	0.98	0.98	0.98
accuracy			1.0
macro avg	0.98	0.99	0.99
weighted avg	1.0	1.0	1.0

Table 3: Classification report using Weighted KNN

Room Occupancy	Precision	Recall	F1 score
0	1.0	1.0	1.0
1	0.96	0.93	0.94
2	0.85	0.89	0.87
3	0.90	0.91	0.90
accuracy			0.98
macro avg	0.93	0.93	0.93
weighted avg	0.98	0.98	0.98

Table 4: Classification report using K-fold cross-validation for KNN

Room Occupancy	Precision	Recall	F1 score
0	1.0	1.0	1.0
1	0.98	0.95	0.96
2	0.94	0.94	0.94
3	0.93	0.98	0.95
accuracy			0.99
macro avg	0.96	0.96	0.96
weighted avg	0.99	0.99	0.99

Table 5: Classification report using after resampling

Room Occupancy	Precision	Recall	F1 score
0	1.0	1.0	1.0
1	1.0	1.0	1.0
2	1.0	1.0	1.0
3	1.0	1.0	1.0
accuracy			1.0
macro avg	1.0	1.0	1.0
weighted avg	1.0	1.0	1.0

REFERENCES

[1] C. Chiţu, G. Stamatescu, and A. Cerpa, "Building occupancy estimation using supervised learning techniques," in *2019 23rd International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE, 2019, pp. 167–172.

[2] P. Cunningham and S. J. Delany, "k-nearest neighbour classifiers: (with python examples)," *arXiv preprint arXiv:2004.04523*, 2020.

Kristina Ghimire is currently pursuing her undergraduate degree in Computer Engineering at IOE, Thapathali Campus. Her research interests encompass various areas, including data mining, machine learning, and deep learning.(THA077BCT023)

Punam Shrestha is currently pursuing her undergraduate degree in Computer Engineering at IOE, Thapathali Campus. Her research interests encompass various areas, including data mining, and web development.(THA077BCT038)