The City College Of New York

Computer Science Department

Senior Design I

Professor Yunhua Zhao

# Summary Report

Assignment #1

Team: NetDesign

Jair Ruiz

Fall 2023

# Table of Contents

## Introduction

This assignment is designed to foster proficiency in key areas of machine learning, specifically focusing on the implementation of the brute force K-NN algorithm for binary classification. By undertaking this task, we will gain hands-on experience in developing a fundamental yet powerful algorithm for pattern recognition and decision-making. Additionally, the assignment emphasizes the importance of data pre-processing and feature selection techniques to ensure the quality and relevance of input data, enhancing the effectiveness of the K-NN model. Furthermore, it highlights the significance of model evaluation techniques in assessing the performance and reliability of machine learning algorithms, enabling students to gain insights into their strengths and limitations. Through this comprehensive approach, students will develop a well-rounded understanding of essential concepts in machine learning while honing their practical skills in algorithm implementation, data manipulation, and model assessment.

## Libraries used

- Numpy
- Collections
- Matplot
- Pandas
- Seaborn

## Implemented Functions and classes:

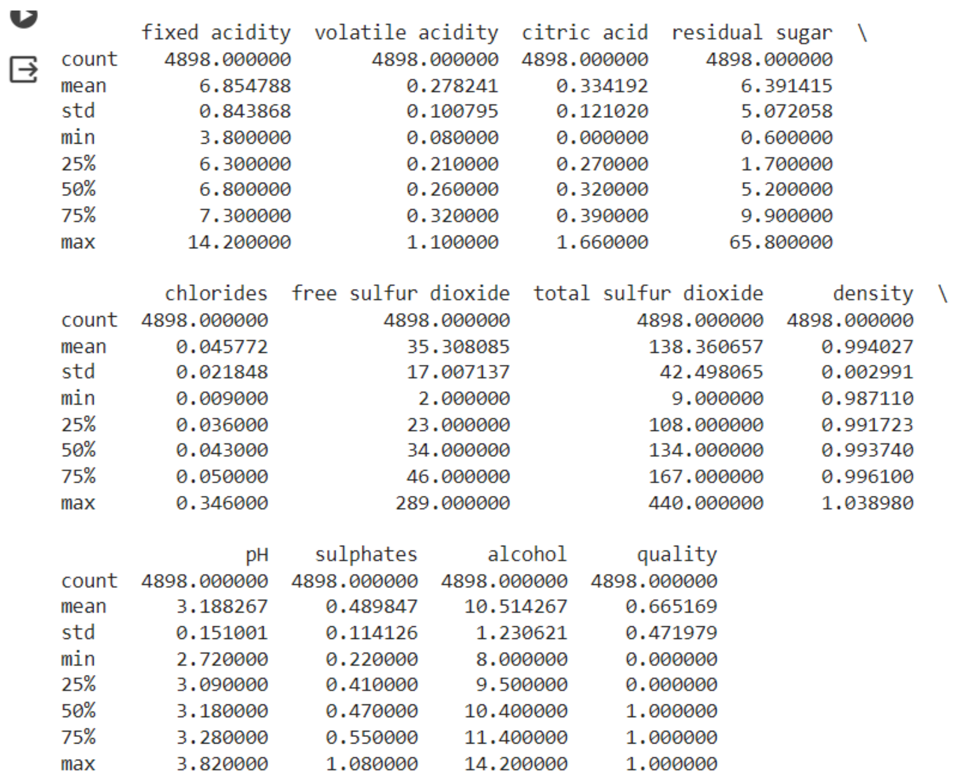
- **Euclidean Distance Function**: Computes the Euclidean distance between two vectors.
- **Manhattan Distance Function:** Calculates the Manhattan distance between two vectors.
- **Accuracy and Generalization Error Function:** Computes the accuracy and generalization error of two vectors.
- **Precision, Recall, and F1 Score Functions**: Computes precision, recall, and F1 score, which are crucial metrics for evaluating binary classification models.
- **Confusion Matrix Function**: Generates the confusion matrix, providing a detailed breakdown of model performance.
- **Receiver Operating Characteristic (ROC) Curve Generation Function**: Generates the ROC curve, illustrating the trade-off between true positive rate and false positive rate across different classification thresholds.
- **Area Under Curve (AUC) Calculation Function:** Computes the area under the ROC curve, providing a single scalar value to quantify the model's discriminatory power.

- **Precision-Recall Curve Generation Function**: Generates the precision-recall curve, depicting the relationship between precision and recall for various classification thresholds.
- **KNN Classifier Model Class:**
    - **fit():** Stores relevant values as instance variables for the KNN classifier model.
    - **predict():** Uses stored instance variables to predict labels for input samples.
- **Partition Function:** Splits data into training and test sets based on a specified partition size.

## Results and Analysis

### Summarizing the data in terms of mean, standard deviation, and quartiles.

To achieve this, we used the Pandas library method describe(), which provides a summary of descriptive statistics for numerical columns in a DataFrame. When applied to a DataFrame, describe() generates a statistical summary that includes measures such as count, mean, standard deviation, minimum, maximum, and quartiles for each numerical column in the DataFrame.
In the image above, we have the results of calling the method, containing a statistical description of each feature in the dataframe.



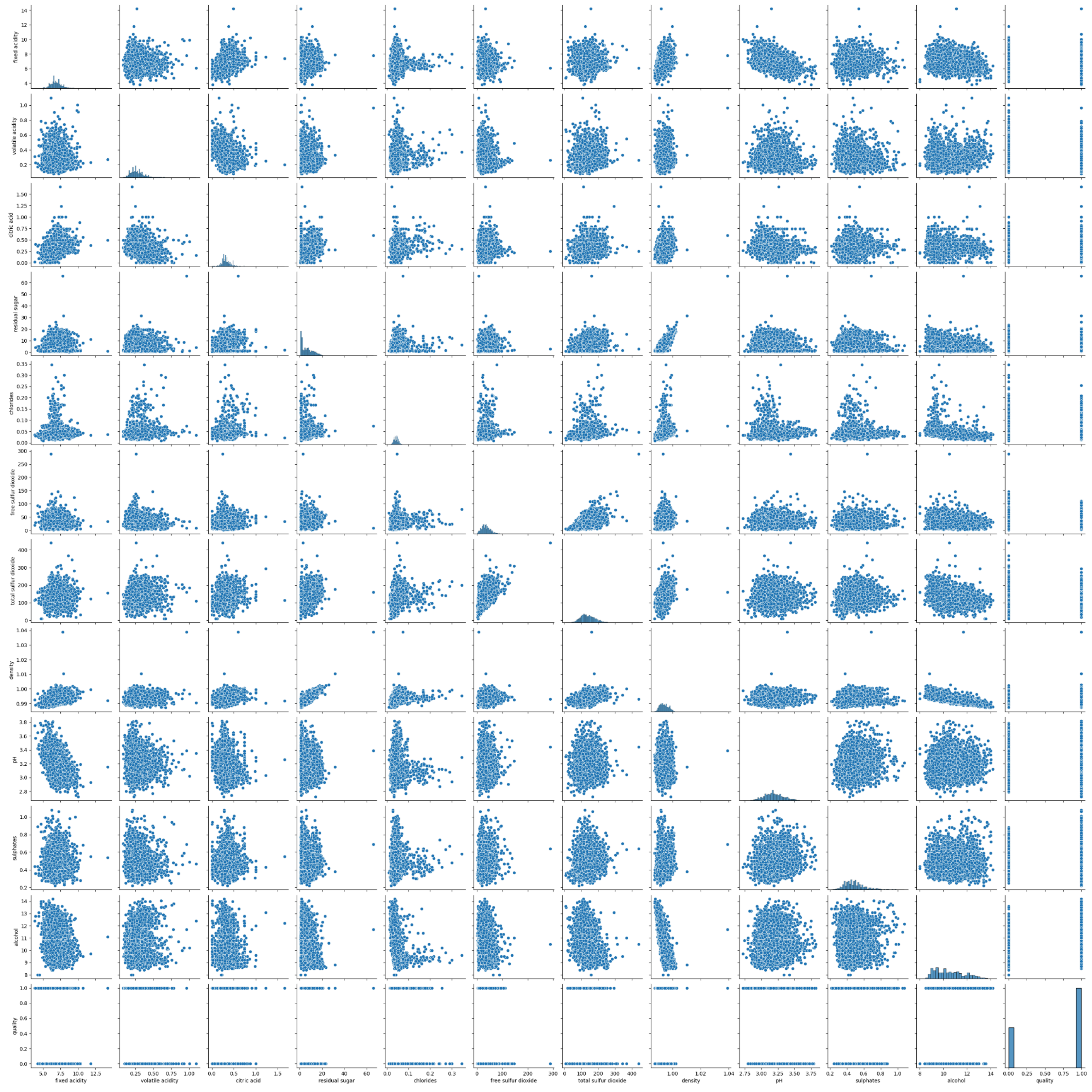
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    4898.000000       4898.000000  4898.000000     4898.000000
mean        6.854788          0.278241     0.334192        6.391415
std         0.843868          0.100795     0.121020        5.072058
min         3.800000          0.080000     0.000000        0.600000
25%         6.300000          0.210000     0.270000        1.700000
50%         6.800000          0.260000     0.320000        5.200000
75%         7.300000          0.320000     0.390000        9.900000
max        14.200000          1.100000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  4898.000000          4898.000000           4898.000000  4898.000000
mean      0.045772            35.308085            138.360657     0.994027
std       0.021848            17.007137             42.498065     0.002991
min       0.009000             2.000000              9.000000     0.987110
25%       0.036000            23.000000            108.000000     0.991723
50%       0.043000            34.000000            134.000000     0.993740
75%       0.050000            46.000000            167.000000     0.996100
max       0.346000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
count  4898.000000  4898.000000  4898.000000  4898.000000
mean      3.188267     0.489847    10.514267     0.665169
std       0.151001     0.114126     1.230621     0.471979
min       2.720000     0.220000     8.000000     0.000000
25%       3.090000     0.410000     9.500000     0.000000
50%       3.180000     0.470000    10.400000     1.000000
75%       3.280000     0.550000    11.400000     1.000000
max       3.820000     1.080000    14.200000     1.000000
```

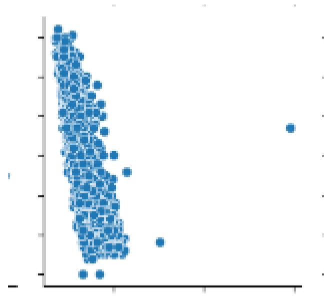
## Dropping redundant features

Pair plots, created using the Seaborn package in Python, are a powerful visualization tool commonly employed in exploratory data analysis (EDA). These plots display pairwise relationships between different variables in a dataset, typically numerical variables, by creating scatterplots for each pair of variables and histograms along the diagonal for single-variable distributions.
The image below shows all the pair plots for every feature in the dataset.

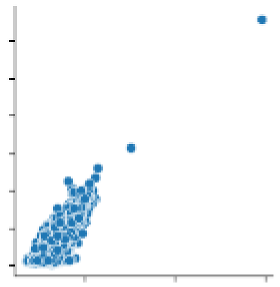


To know which features to drop, we can see this plots and see which ones have the higher correlation, identifying the shape that the points make along the graph. The more the cluster shape is like a line, the higher the correlation is, and this means that we have to drop one of these features.
I noticed two situations where the correlation is high:

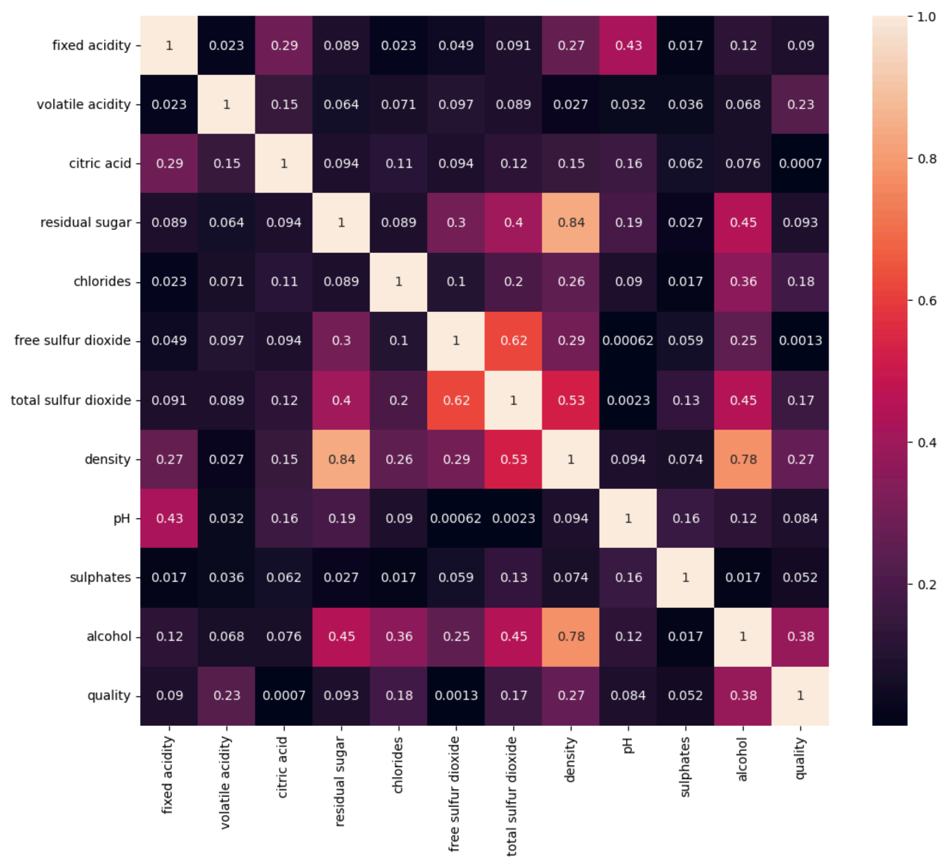
Density vs alcohol



Density vs residual sugar



It is intuitive to conclude that these two plots have a high correlation by looking at the shape, but to make sure we can use the heatmap() method from the seaborn package. By looking at the heatmap, we can see that my assumption is correct, assuming our threshold is 0.7.



## Naive approach vs standardized data

| | Naive Approach | Standardized Approach |
|---|---|---|
| Accuracy | 0.6969387755102041 | 0.7836853605243992 |
| F1 Score | 0.7826530612244897 | 0.843956043956044 |

From the provided results, it's evident that applying feature standardization to the data before training the model has led to a significant improvement in both accuracy and F1 score compared to the naive approach. The standardized approach achieved a higher accuracy of approximately 0.783 compared to the naive approach's accuracy of around 0.697. The F1 score, which considers both precision and recall, also shows a noticeable improvement in the standardized approach, with a score of approximately 0.844 compared to the naive approach's score of around 0.784. These results suggest that standardizing the features has helped the KNN classifier model perform better. Standardization is particularly beneficial for KNN and other distance-based algorithms because it ensures that all features contribute equally to the distance computations, preventing features with larger scales from dominating the distance calculations. Consequently, the model trained on standardized features makes more accurate predictions and achieves a higher F1 score, indicating better overall performance in terms of precision and recall. Therefore, it's advisable to use standardized data rather than unscaled data for further analysis and modeling.

## Uniform vs Distance

| | Uniform | Distance |
|---|---|---|
| Accuracy | 0.7836853605243992 | 0.6775510204081633 |
| F1 Score | 0.843956043956044 | 0.7523510971786833 |

Based on these results, the KNN classifier with uniform weighting outperformed the one with distance weighting in terms of both accuracy and F1 score. The model, using uniform weights, achieved a higher accuracy of approximately 0.784 compared to using distance-based weights, which had an accuracy of around 0.678. Similarly, the F1 score for the model using uniform weights was higher, with a score of approximately 0.844 compared to using distance-based weights, which had an F1 score of around 0.752.
These results suggest that, in this particular scenario, using uniform weights leads to better model performance compared to distance-based weights. While inverse distance weighting can sometimes provide improved performance, especially when data points closer to the query point are considered more influential, it seems that in this case, the uniform weighting scheme is more effective. Therefore, it is advisable to stick with uniform weights for the KNN classifier model in this context.

## Model Evaluation

For euclidean distance function:

- K=1, Distance Metric= Euclidean, Weights=uniform:
    - Accuracy: 0.7990
    - F1 Score: 0.8472
    - Generalization Error: 0.2010
    - Precision: 0.8323
    - Recall: 0.8626
    - Confusion Matrix: [[237 110]
      [ 87 546]]
- K=1, Distance Metric= Euclidean, Weights=distance:
    - Accuracy: 0.7990

- F1 Score: 0.8472
    - Generalization Error: 0.2010
    - Precision: 0.8323
    - Recall: 0.8626
    - Confusion Matrix: [[237 110]
      [ 87 546]]

- K=5, Distance Metric= Euclidean, Weights=uniform:
    - Accuracy: 0.7429
    - F1 Score: 0.8085
    - Generalization Error: 0.2571
    - Precision: 0.7789
    - Recall: 0.8404
    - Confusion Matrix: [[196 151]
      [101 532]]
- K=5, Distance Metric= Euclidean, Weights=distance:
    - Accuracy: 0.6765
    - F1 Score: 0.7408
    - Generalization Error: 0.3235
    - Precision: 0.7678
    - Recall: 0.7156
    - Confusion Matrix: [[210 137]
      [180 453]]

- K=9, Distance Metric= Euclidean, Weights=uniform:
    - Accuracy: 0.7500
    - F1 Score: 0.8159
    - Generalization Error: 0.2500
    - Precision: 0.7779
    - Recall: 0.8578
    - Confusion Matrix: [[192 155]
      [ 90 543]]
- K=9, Distance Metric= Euclidean, Weights=distance:
    - Accuracy: 0.5673
    - F1 Score: 0.6326
    - Generalization Error: 0.4327
    - Precision: 0.7006
    - Recall: 0.5766
    - Confusion Matrix: [[191 156]
      [268 365]]

- K=11, Distance Metric= Euclidean, Weights=uniform:
    - Accuracy: 0.7490
    - F1 Score: 0.8142
    - Generalization Error: 0.2510
    - Precision: 0.7800
    - Recall: 0.8515
    - Confusion Matrix: [[195 152]
      [ 94 539]]
- K=11, Distance Metric= Euclidean, Weights=distance:

- Accuracy: 0.5286
- F1 Score: 0.5954
- Generalization Error: 0.4714
- Precision: 0.6680
- Recall: 0.5371
- Confusion Matrix: [[178 169]
  [293 340]]

For manhattan distance function:

- K=1, Distance Metric= Manhattan, Weights=uniform:
  - Accuracy: 0.7990
  - F1 Score: 0.8465
  - Generalization Error: 0.2010
  - Precision: 0.8354
  - Recall: 0.8578
  - Confusion Matrix: [[240 107]
    [ 90 543]]
- K=1, Distance Metric= Manhattan, Weights=distance:
  - Accuracy: 0.7990
  - F1 Score: 0.8465
  - Generalization Error: 0.2010
  - Precision: 0.8354
  - Recall: 0.8578
  - Confusion Matrix: [[240 107]
    [ 90 543]]

- K=5, Distance Metric= Manhattan, Weights=uniform:
  - Accuracy: 0.7510
  - F1 Score: 0.8149
  - Generalization Error: 0.2490
  - Precision: 0.7839
  - Recall: 0.8483
  - Confusion Matrix: [[199 148]
    [ 96 537]]
- K=5, Distance Metric= Manhattan, Weights=distance:
  - Accuracy: 0.6592
  - F1 Score: 0.7280
  - Generalization Error: 0.3408
  - Precision: 0.7513
  - Recall: 0.7062
  - Confusion Matrix: [[199 148]
    [186 447]]

- K=9, Distance Metric= Manhattan, Weights=uniform:
  - Accuracy: 0.7602
  - F1 Score: 0.8245
  - Generalization Error: 0.2398

- Precision: 0.7819
- Recall: 0.8720
- Confusion Matrix: [[193 154]
                                         [ 81 552]]

- K=9, Distance Metric= Manhattan, Weights=distance:
  - Accuracy: 0.5551
  - F1 Score: 0.6299
  - Generalization Error: 0.4449
  - Precision: 0.6807
  - Recall: 0.5861
  - Confusion Matrix: [[173 174]
                                         [262 371]]

- K=11, Distance Metric= Manhattan, Weights=uniform:
  - Accuracy: 0.7612
  - F1 Score: 0.8227
  - Generalization Error: 0.2388
  - Precision: 0.7904
  - Recall: 0.8578
  - Confusion Matrix: [[203 144]
                                         [ 90 543]]
- K=11, Distance Metric= Manhattan, Weights=distance:
  - Accuracy: 0.5286
  - F1 Score: 0.6085
  - Generalization Error: 0.4714
  - Precision: 0.6563
  - Recall: 0.5671
  - Confusion Matrix: [[159 188]
                                         [274 359]]

The performance evaluation results showcase the effectiveness and sensitivity of the K-NN classifier to various hyperparameter configurations, including the number of neighbors (K), distance metric (Euclidean or Manhattan), and weighting scheme (uniform or distance-based).

Accuracy and Generalization Error:
Accuracy serves as a fundamental metric, reflecting the classifier's ability to correctly classify instances. Higher accuracy values suggest better overall performance. Conversely, generalization error quantifies the model's performance on unseen data, providing insights into potential overfitting or underfitting issues. In our evaluations, accuracy values range from approximately 52.86% to 79.90%, indicating notable variations across different parameter settings. Moreover, the generalization error ranges from around 20.10% to 47.14%, emphasizing the importance of selecting optimal hyperparameters to balance model complexity and generalization ability.

F1 Score, Precision, and Recall:
F1 score, precision, and recall collectively offer a comprehensive view of the classifier's predictive capabilities, particularly in binary classification tasks. The F1 score balances precision

and recall, highlighting the trade-off between correctly identifying positive instances and minimizing false positives and false negatives. Across our evaluations, F1 scores vary from approximately 59.54% to 84.72%, showcasing the impact of different parameter configurations on the model's ability to achieve a harmonious balance between precision and recall.

Confusion Matrix:
The confusion matrix provides granular insights into the classifier's performance by detailing true positive, true negative, false positive, and false negative predictions. By examining the distribution of predictions across these categories, we gain valuable insights into the classifier's strengths and weaknesses. For instance, a balanced distribution of true positives and true negatives signifies robust predictive performance, while disproportionate false positives or false negatives may indicate areas for improvement.

The choice of hyperparameters significantly influences the K-NN classifier's performance, as evidenced by the diverse range of evaluation metrics across different parameter settings.
Lower values of K tend to yield higher accuracy but may be susceptible to overfitting, leading to increased generalization error on unseen data.
The Euclidean and Manhattan distance metrics exhibit distinct performance characteristics, with their suitability depending on the dataset's underlying structure and characteristics.
Weighting schemes, particularly distance-based weighting, can enhance predictive accuracy by assigning greater importance to closer neighbors during classification.
Variations in performance metrics underscore the importance of hyperparameter optimization and careful parameter selection to maximize the classifier's predictive power while minimizing overfitting or underfitting risks.

## Conclusion

In this report, we extensively evaluated the performance of the K-NN (K-Nearest Neighbors) classifier across various hyperparameter configurations, aiming to achieve optimal predictive accuracy and generalization ability. Through systematic experimentation, we investigated the impact of hyperparameters such as the number of neighbors (K), distance metric (Euclidean or Manhattan), and weighting scheme (uniform or distance-based) on classifier performance.
Our findings revealed the sensitivity of the K-NN classifier to hyperparameter settings, with significant variations observed in performance metrics such as accuracy, F1 score, precision, recall, and generalization error. Notably, we observed trade-offs between model complexity, predictive accuracy, and generalization ability, emphasizing the importance of striking a balance between these factors for robust model performance.
Lower values of K tend to yield higher accuracy but may lead to overfitting, while larger values of K enhance generalization but may sacrifice accuracy on the training data.

The choice between Euclidean and Manhattan distance metrics depends on dataset characteristics, with each metric offering distinct advantages in specific scenarios.
Distance-based weighting schemes can improve classifier performance by assigning greater importance to closer neighbors during classification, especially for larger values of K.
Based on our analysis, we recommend practitioners to conduct thorough hyperparameter tuning and evaluation using techniques such as cross-validation to identify optimal parameter settings.
Additionally, careful consideration should be given to the trade-offs between model complexity, predictive accuracy, and generalization ability when selecting hyperparameters.