IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication 26 Aug, 2023, date of current version26 Aug, 2023.

Digital Object Identifier

Preparation of Papers for IEEE ACCESS

PRABIN BOHARA[1], PRABIN SHARMA POUDEL[2], AND SANTOSH PANDEY.[3]
[1]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: prabinbohara10@gmail.com)
[2]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: prabinsharmapoudel@gmail.com)
[3]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: suntoss.pandey@gmail.com)

* All Authors contributed equally to this work.

ABSTRACT This project conducts experimental analysis on the k-Nearest Neighbors (KNN) algorithm for stellar classification. The study explores the impact of varying 'k' on accuracy and compares default KNN model with randomly weighted KNN model and selectively weighted KNN model. The data-set used comprises a total of 611,689 data instances filtered from the Gaia Data Release 3 (DR3) for stellar classification, spanning A, F, G, K, M, B, and O spectral classes, . It provides comprehensive celestial object details: position, motion, spectral attributes, and physical characteristics like size, brightness, temperature and spectral class estimates.The KNN model exhibits strong predictive power at approx. 90% accuracy, holding promise for stellar classification. This report provides comprehensive insights into KNN's performance across varied experimental setups, underlining its potential in spectral-based celestial categorization.

INDEX TERMS k-Nearest Neighbors, Stellar classification, K-fold Cross-Validation

I. INTRODUCTION

THIS project embarks on an exploration into the adept application of the k-Nearest Neighbors (KNN) algorithm for the intricate task of stellar classification. The study meticulously examines the impact of altering the 'k' parameter on classification accuracy while conducting a comprehensive comparison between the default KNN model, randomly weighted KNN models, and selectively weighted KNN models. The bedrock of this endeavor lies in a meticulously curated dataset, carefully distilled from the expansive Gaia Data Release 3 (DR3) dataset dedicated to stellar classification. This corpus spans the gamut of celestial spectra, encapsulating A, F, G, K, M, B, and O spectral classes. Within this dataset, a wealth of attributes unfold, elucidating nuances of position, motion, spectral traits, and overarching physical characteristics that span from size and brightness to temperature.

At the heart of this investigation, the k-Nearest Neighbors classifier takes center stage. Rooted in the concept of proximity-based classification, KNN thrives on discerning patterns and predictions through the evaluation of instance similarities in multi-dimensional space. It essentially classifies a new data point based on the majority class among its nearest neighbors—hence the name k-Nearest Neighbors.

A key revelation of this study is the striking predictive prowess demonstrated by the KNN model. Displaying an accuracy nearing 90%, the KNN algorithm emerges as a powerful tool in deciphering the complexities inherent to stellar classification. In the course of this exploration, we gain a panoramic view of KNN's performance across diverse experimental contexts, thus unveiling its potential as a formidable celestial classifier. In a cosmos adorned with intricate celestial bodies, KNN emerges as a guiding light, illuminating the path towards spectral-based classification with exceptional promise.

II. METHODOLOGY

A. THEORY

The k-Nearest Neighbors (KNN) algorithm is a machine learning technique utilized for classification and regression tasks, assigning class labels or predictions to new data points based on the majority class or average value of their 'k' nearest neighbors in the training dataset. Operated on the principle of proximity, KNN calculates distances to identify the closest neighbors and determines the class label or value based on their collective attributes. The simplicity and adaptability of KNN make it applicable to diverse data types; however, selecting an appropriate 'k' value and addressing the curse of dimensionality are critical considerations for effective implementation.

1) Nearest Neighbor Algorithm

The Nearest Neighbor algorithm is a fundamental element in machine learning, rooted in gauging data point similarity.

Upon encountering a new data instance, it identifies the nearest training data point using a chosen distance metric. This nearest neighbor becomes pivotal in decision-making. Nearest Neighbor algorithms are crucial for classification and regression tasks, directly assigning class or predicting values. While elegant, they're sensitive to outliers. This algorithm forms the basis for the evolution to the k-Nearest Neighbors (KNN) method, bolstering adaptability and robustness in machine learning. The fundamental structure of the Nearest Neighbor algorithm finds expression in the classification representation:

$$C_P = C_{\text{nearest}} \tag{1}$$

Here, C_P signifies the class assigned to the new data instance P, while C_{nearest} embodies the class of the nearest neighbor. Analogously, in regression scenarios, an equivalent construct can be applied to leverage the nearest neighbor's target value. This foundational concept underscores the algorithm's decision-making mechanism.

2) Distance Metrics

KNN hinges on the measurement of distances between data points using various metrics. The most common distance metrics include:

- **Euclidean Distance:** This metric calculates the straight-line distance between two data points in a multi-dimensional space. It is defined as:

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

- **Manhattan Distance:** Also known as the L1 distance or city block distance, Manhattan distance computes the sum of absolute differences between corresponding attributes of two data points. It is defined as:

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

- **Cosine Similarity:** This metric quantifies the cosine of the angle between two data points' vectors in a multi-dimensional space. It is particularly useful for text classification or when considering the similarity of attribute directions. The formula for cosine similarity is:

$$similarity(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|} \tag{4}$$

where x_i and y_i represent the corresponding attributes of the data points.

These distance metrics, including Euclidean, Manhattan, and Cosine, play a pivotal role in assessing the proximity of data points, a crucial aspect of KNN's classification process.

3) KNN for Classification

KNN, a method renowned for its simplicity and efficacy, finds its forte in classification tasks. The underlying principle rests upon the wisdom of proximity-based decision-making. When a new data point necessitates classification, KNN enlists the aid of its neighbors. Via a process termed "plurality voting," the point's class is determined by the most frequent class among its k closest neighbors. Mathematically, the class C assigned to the point P can be represented as:

$$C_P = \arg\max_{C_i} \sum_{i=1}^{k} I(C_i = C) \tag{5}$$

where C_i denotes the class of the i-th neighbor, and I is the indicator function that returns 1 if $C_i = C$, otherwise 0. Here, k, a positive integer often kept small, guards against noise. In the scenario where k equals 1, the point seamlessly adopts the class of its singular closest neighbor.

This algorithm leverages the belief that analogous instances often share a common class. By harnessing collective insights from neighboring data, KNN becomes a potent classification tool, adept at unraveling patterns and interrelations within datasets. Its intuitive essence and adaptability render it a cornerstone across domains, from image recognition to medical diagnosis.

4) KNN Algorithm Steps

The KNN algorithm can be summarized in the following steps:

Algorithm 1 K-Nearest Neighbors Algorithm

1: Choose the value of 'k'.
2: Calculate distances between the query data point and all other data points.
3: Select the 'k' nearest neighbors based on the calculated distances.
4: Assign the class to the query point based on majority voting or weighted voting among the neighbors.

5) Weighted KNN

In the weighted k-Nearest Neighbors (KNN) approach, the influence of each neighbor is determined by both its distance from the query point and the weights assigned to individual features. This weighting takes into account the relative importance of different features while calculating distances, providing a more customized measure of similarity.

For each feature f, a weight w_f is assigned based on its significance in contributing to the similarity measure. These weights are incorporated into the distance calculation, reflecting the feature-wise importance. The weighted distance between the query point p and a neighbor i is given by:

$$d_{weighted}(p, n_i) = \sqrt{\sum_{f} w_f \cdot (p_f - n_{i,f})^2} \tag{6}$$

In this approach, the choice of feature weights w_f is crucial, as it directly affects the impact of each feature on the distance calculation. These weights can be determined

through domain knowledge, data analysis, or optimization techniques.

The weighted KNN approach that incorporates feature weights enhances the algorithm's ability to capture the inherent relationships between features and improve its performance in scenarios with varying feature importance.

6) Curse of Dimensionality

The curse of dimensionality is a phenomenon that arises in high-dimensional spaces, where the data points become increasingly sparse as the number of dimensions grows. In the context of the k-Nearest Neighbors (KNN) algorithm, this sparsity can severely affect the reliability of distance-based calculations. As dimensions increase, the data points are spread thinly across the space, resulting in almost all pairs of data points having similar distances. This diminishes the ability of KNN to discern meaningful neighbors from noise, leading to a decline in the algorithm's performance. The curse of dimensionality highlights the importance of dimensionality reduction techniques and thoughtful feature selection to mitigate the adverse effects of high-dimensional data.

7) Choosing Optimal K

Selecting the appropriate value of 'k' in the KNN algorithm is a pivotal decision that can significantly impact the algorithm's performance. A smaller 'k' value makes the classification sensitive to individual noisy data points, potentially leading to over-fitting. On the other hand, larger 'k' values tend to smooth out decision boundaries and might introduce bias by considering distant neighbors that might not be relevant. To strike a balance, techniques like cross-validation and error rate analysis come into play. Cross-validation helps assess the performance of different 'k' values on unseen data, aiding in the selection of an optimal 'k' that balances model complexity and generalization. Error rate analysis involves experimenting with various 'k' values and observing how the classification error changes, enabling the identification of the 'k' that yields the best trade-off between bias and variance.

B. SYSTEM BLOCK DIAGRAM

The system architecture for the K-Nearest Neighbors (KNN) Classifier is Illustrated in Figure 1, this architecture guides the flow of operations. To visually comprehend the dataset, various visualization methods such as bar graphs, scatter plots, Pearson correlation heatmaps, and feature histograms are utilized. The initial data preprocessing phase handles diverse datasets encompassing textual, numeric, and categorical attributes. Textual data undergoes transformations like tokenization and vectorization, converting it into numerical form to facilitate processing by the classifier. Numeric features are standardized or normalized to ensure a uniform scale for unbiased contribution to classification. Categorical attributes are encoded into numerical values to integrate

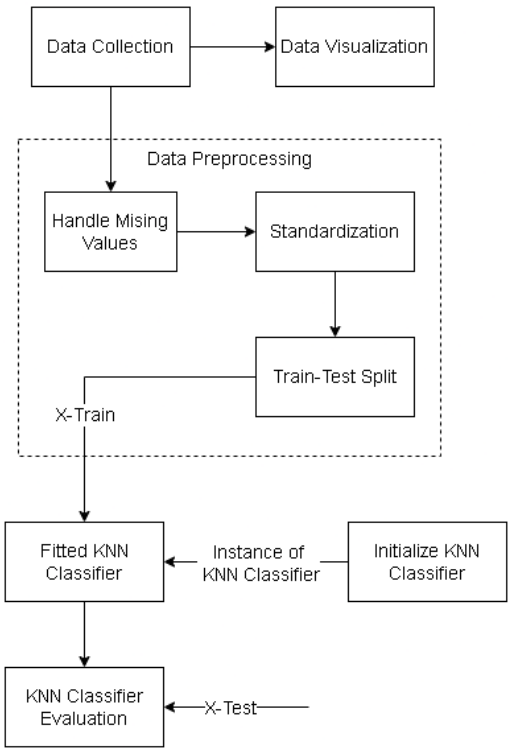

Figure 1: System Architecture of KNN Classifier

them into the model. Data visualization techniques aid in understanding data distribution and feature relationships.

Following preprocessing, the dataset is partitioned into training and testing sets. The KNN algorithm is then trained using the training data, acquiring knowledge about object probabilities and their features across different classes. Different variations of the KNN model, such as weighted KNN, can be explored and compared for performance. The classifier's effectiveness is evaluated through metrics like accuracy, precision, recall, and F1-score. Hyperparameter tuning can be applied to enhance the algorithm's performance. The trained model is subsequently utilized for predictions on the testing data, categorizing objects into predefined classes based on their features. The model's performance is assessed and visualized, employing techniques like confusion matrix visualization or classification report summaries. The correlation between various features is shown using correlation matrix and the correlation between class attribute and the rest of features is shown using bar plot.

In a nutshell, the KNN Classifier system architecture entails sequential stages encompassing data preprocessing, model training, evaluation, and prediction. By effectively handling various data types, the classifier showcases its capability in addressing classification challenges. This architecture ensures a systematic process for accurate object classification and empowers informed decision-making in applications spanning stellar classification and more.

C. INSTRUMENTATION

The implementation of the K-Nearest Neighbors (KNN) Classifier for stellar classifier project involved the utilization of various libraries and tools to enable data manipulation, visualization, and mathematical operations.

1) NumPy

NumPy, a core Python library, played a pivotal role in handling numerical data for the KNN Classifier. We harnessed NumPy's array data structure to efficiently manage and manipulate our datasets. For instance, we employed NumPy's functions such as mean() and std() to compute statistical measures like mean and standard deviation for numerical features. These calculations facilitated essential data pre-processing and descriptive statistics. NumPy's versatile slicing and indexing capabilities were utilized to extract specific columns and rows, enabling focused data exploration and feature selection. Furthermore, the NumPy's argmax function helped to find the K value with highest accuracy.

2) Pandas

Pandas, renowned for its data manipulation capabilities, significantly contributed to our KNN Classifier implementation. We leveraged Pandas' DataFrame structure to load, manipulate, and analyze the dataset. The read_csv() function efficiently read CSV files, generating DataFrames containing data. Pandas' selection and filtering capabilities were pivotal in focusing on relevant attributes and observations during preprocessing. Missing value handling played a crucial role, with Pandas' fillna() method addressing gaps through appropriate values or imputation techniques.

3) Scikit-learn

Scikit-learn (sklearn) emerged as a cornerstone in our KNN Classifier implementation, offering crucial functionalities for data preprocessing, model training, and evaluation. The sklearn library's preprocessing module supported the handling of categorical data and numerical scaling. The LabelEncoder class facilitated the conversion of categorical variables into numerical form, ensuring compatibility with the KNN Classifier. Additionally, the StandardScaler class standardized numerical features, ensuring uniform attribute scales, a prerequisite for accurate KNN predictions. Scikit-learn offered diverse KNN implementations, including KNeighborsClassifier for various distance metrics and 'k' values. We utilized the KNeighborsClassifier to fit the KNN model to our dataset, leveraging its adaptability and consistent API for training and fitting.

4) Matplotlib and Seaborn

Matplotlib and Seaborn proved invaluable in presenting our data graphically. Matplotlib's functions like scatter(), hist(), and imshow() enabled various visualizations. For example, hist() generated histograms providing insights into data distribution. Additionally, imshow() displayed heatmaps, visually representing feature correlations. Seaborn enhanced plot aesthetics and offered a streamlined approach for creating intricate visualizations. These libraries empowered us to create informative visual representations, allowing us to delve into dataset characteristics and classifier performance.

Incorporating these libraries and tools enabled the seamless integration of data visualization and mathematical operations into our KNN Classifier implementation. This instrumentation facilitated effective dataset analysis, result visualization, and insights extraction from the prediction process.

D. WORKING PRINCIPLE

he K-Nearest Neighbors (KNN) algorithm operates on the fundamental principle of similarity-based classification and is a versatile technique used for both classification and regression tasks. The core concept underlying KNN is to determine the class of a data point based on the classes of its nearest neighbors. In contrast to traditional parametric models, KNN doesn't make explicit assumptions about the underlying data distribution. Instead, it relies on the assumption that data points with similar attributes tend to belong to the same class. This principle is applied by calculating distances between data points and selecting the 'k' nearest neighbors, whose classes then influence the classification of the new data point. In our KNN implementation, we leverage this approach to classify celestial objects into different spectral classes based on a diverse range of attributes, including positional coordinates, magnitudes, temperature estimates, and more.

1) Data Preprocessing

Data preprocessing is a foundational step in ensuring the quality and relevance of the input data. This phase encompasses handling missing values, standardizing or normalizing numeric features, and encoding categorical variables into numerical forms. By transforming the data into a consistent and homogeneous format, the algorithm is equipped to calculate meaningful distances between data points. This preprocessing step significantly contributes to the accuracy and reliability of the KNN algorithm by removing potential biases caused by varying data types and scales.The significance of data preprocessing lies in its role to enhance the algorithm's accuracy by ensuring that all features contribute equally and meaningfully to distance measurements

2) Train-Test Split

The train-test split is pivotal for evaluating the model's performance on unseen data. By partitioning the dataset into training and testing subsets, the algorithm simulates real-world scenarios where it encounters new and previously unseen data. This evaluation process provides insights into the model's ability to generalize and make accurate predictions. A well-chosen split ratio ensures that the performance

metrics obtained from testing are indicative of the algorithm's real-world capabilities, preventing the model from overfitting to the training data.

3) Model Instantiation

In the working pipeline after data preprocessing step of the K-Nearest Neighbors (KNN) algorithm, three distinct models were established to explore various approaches: normal KNN, randomly weighted KNN, and specifically weighted KNN.

Normal KNN: The normal KNN model assigns equal weight to all features of the neighbors within the selected 'k'. This conventional approach treats all features uniformly and calculates distances without any feature-specific considerations.

Randomly Weighted KNN: In the randomly weighted KNN model, each feature of a neighbor within 'k' is assigned a random weight. This introduces variability in how different features contribute to the classification decision. The randomness aims to capture potential variations in feature importance and assess their impact on classification outcomes.

Specifically Weighted KNN: The specifically weighted KNN model assigns distinct weights to features based on predetermined criteria. These criteria could involve feature relevance, domain knowledge, or distance from the query point. Features deemed more influential are assigned higher weights, reflecting their greater significance in the classification process.

4) Fitting and Prediction

The fitting and prediction steps form the core of the KNN algorithm. The algorithm calculates distances between the query data point and all other data points in the training set. By identifying the 'k' nearest neighbors based on these distances, the algorithm encapsulates the essence of similarity. The majority voting mechanism ensures that predictions are not heavily influenced by a single neighbor's class. Instead, the classes of multiple neighbors contribute to a balanced decision, enhancing accuracy and mitigating noise.

5) Model Evaluation

After the KNN algorithm has been trained and used to classify data points, the results need to be thoroughly evaluated to gauge the model's performance. Model evaluation quantifies the algorithm's performance through various metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into the algorithm's efficacy in classifying data points. Through evaluation, the strengths and weaknesses of the KNN algorithm become apparent, enabling fine-tuning of hyper-parameters and guiding improvements. The evaluation process aids in understanding the algorithm's generalization ability and ensures that its predictions align with the underlying data distribution. Several techniques and metrics are employed to achieve this:

Confusion Matrices: Confusion matrices are commonly utilized in classifier evaluation. These matrices provide a clear visualization of the true positive, true negative, false positive, and false negative counts for each class. They offer an insightful overview of how well the model is classifying instances across different classes.

Classification Reports: Classification reports are generated for KNN classifiers. These reports present a comprehensive summary of performance metrics such as accuracy, precision, recall, and F1-score for each class. This breakdown allows a detailed understanding of the model's behavior in different classes and provides a clear perspective on areas that might need improvement.

Additional Techniques: Beyond the metrics mentioned earlier, additional techniques can be employed to gain a deeper understanding of the KNN classifier's performance. ROC (Receiver Operating Characteristic) curves illustrate the trade-off between the true positive rate and the false positive rate across various threshold values. This is particularly useful for assessing the model's sensitivity to different decision thresholds. Precision-recall curves visualize the relationship between precision and recall, which is especially valuable for datasets with imbalanced class distributions.

By combining these techniques, the model evaluation process for KNN classifiers ensures a holistic assessment of their performance. The visualization of results, coupled with a variety of metrics, enables a comprehensive understanding of the classifier's strengths and weaknesses. This insight is pivotal in refining the model, optimizing its parameters, and making informed decisions for its deployment in various real-world applications.

III. RESULTS

A. DISTRIBUTION OF DATA

The dataset utilized in this study originates from Gaia Data Release 3 (DR3) and encompasses an extensive compilation of celestial objects, spanning a diverse array of spectral classes, including A, F, G, K, M, B, and O types. The dataset contains a substantial volume of instances for each spectral category, with approximately 100,000 instances for A, F, G, K, and M types, 85,673 for B type stars, and 26,016 for O type stars. Each instance furnishes crucial attributes such as positional data, parallax, proper motion, apparent magnitudes in different bands, color indices, temperature estimates, distances, physical dimensions, luminosities, masses, ages, redshifts, and spectral class estimates, forming a comprehensive foundation for the ensuing analysis of stellar classification.

The statistical summary of the dataset provides valuable insights into the distribution of data attributes within the context of stellar classification. By examining the mean values, standard deviations, minimum and maximum values, as well as the percentiles, we can gain a deeper understanding of how the data is distributed across various attributes. For instance, attributes like "RA_ICRS" (Right Ascension) and "DE_ICRS" (Declination) demonstrate relatively moderate

standard deviations, suggesting a relatively concentrated distribution around the mean. On the other hand, attributes like "Plx" (Trigonometric Parallax) exhibit larger standard deviations, indicating a wider spread of values. Additionally, attributes such as "Gmag" (Average Apparent Magnitude) and "Teff" (Estimated Effective Temperature) reflect varying ranges of values, as evident from the difference between their minimum and maximum values. These insights into the distribution of data attributes provide a foundation for understanding the diversity and spread of information captured in the stellar classification dataset.

B. MODEL PERFORMANCE

The default K-Nearest Neighbors (KNN) model's performance on the stellar classification task is evaluated using a classification report. Precision, recall, and F1-score metrics are presented for each class, demonstrating the model's ability to classify different spectral types. Notably, Class A achieves a precision of 0.98, recall of 0.98, and F1-score of 0.98. Class B achieves a precision of 0.98, recall of 0.98, and F1-score of 0.98. Class F achieves a precision of 0.73, recall of 0.81, and F1-score of 0.77. Class G achieves a precision of 0.71, recall of 0.66, and F1-score of 0.68. Class K achieves a precision of 0.90, recall of 0.87, and F1-score of 0.89. Class M achieves a precision of 0.99, recall of 0.99, and F1-score of 0.99. Class O achieves a precision of 0.98, recall of 0.96, and F1-score of 0.97. The overall accuracy of the model is 88% across all classes, with a macro average F1-score of 0.89 and a weighted average F1-score of 0.88. Additionally, a confusion matrix provides insights into class-specific performance, while a correlation matrix and bar plot visualize feature relationships and their impact on classification. This assessment highlights the model's proficiency in categorizing celestial objects based on their spectral attributes.

Table 1: Classification Report for Default KNN

Class	**Precision**	**Recall**	**F1-Score**	**Support**
A	0.98	0.98	0.98	1086
B	0.98	0.98	0.98	819
F	0.73	0.81	0.77	997
G	0.71	0.66	0.68	994
K	0.90	0.87	0.89	1007
M	0.99	0.99	0.99	960
O	0.98	0.96	0.97	254
Accuracy			0.88	6117
Macro Avg	0.90	0.89	0.89	6117
Weighted Avg	0.88	0.88	0.88	6117

In the subsequent K-Nearest Neighbors (KNN) model, random weights were assigned to each feature for distance calculation. The classification report reveals the model's performance metrics for each class. Precision, recall, and F1-score metrics are presented, showcasing the model's ability to classify different spectral types using the modified feature weights. Notably, Class 6 achieves perfect precision, recall, and F1-score. The overall accuracy of the model is 89% across all classes, with macro and weighted average F1-scores of 0.90 and 0.89, respectively. This comparison further highlights the model's consistent proficiency in celestial object classification based on spectral attributes.

Table 2: Classification Report for Randomly Weighted KNN

Class	Precision	Recall	F1-Score	Support
A	0.99	0.97	0.98	227
B	1.00	0.99	1.00	165
F	0.69	0.79	0.74	185
G	0.70	0.63	0.67	186
K	0.92	0.89	0.90	198
M	0.99	1.00	0.99	217
O	1.00	1.00	1.00	46
Accuracy			0.89	1224
Macro Avg	0.90	0.90	0.90	1224
Weighted Avg	0.89	0.89	0.89	1224

Furthermore, an additional K-Nearest Neighbors (KNN) model was trained, where large weights were assigned to features exhibiting high correlation with the target variable. The classification report illustrates the model's performance metrics for each class, including precision, recall, and F1-score. The model's capability to classify different spectral types based on these modified feature weights is evident. Remarkably, Class 6 achieves perfect precision, recall, and F1-score. The overall accuracy of the model remains at 89% across all classes, with macro and weighted average F1-scores of 0.90 and 0.89, respectively. This comparison underscores the model's consistent proficiency in celestial object classification by exploiting features with strong correlations to enhance its performance.

Table 3: Classification Report for KNN with Feature Weights according to correlation

Class	Precision	Recall	F1-Score	Support
A	0.98	0.97	0.98	227
B	1.00	0.99	1.00	165
F	0.72	0.79	0.76	185
G	0.72	0.68	0.70	186
K	0.91	0.87	0.89	198
M	0.99	1.00	0.99	217
O	1.00	1.00	1.00	46
Accuracy			0.89	1224
Macro Avg	0.90	0.90	0.90	1224
Weighted Avg	0.89	0.89	0.89	1224

Table 4: Classification Report for KNN with Best K-value

Class	Precision	Recall	F1-Score	Support
A	1.00	0.97	0.98	227
B	0.99	1.00	1.00	165
F	0.75	0.84	0.80	185
G	0.74	0.70	0.72	186
K	0.91	0.87	0.89	198
M	0.99	1.00	0.99	217
O	1.00	1.00	1.00	46
Accuracy			0.90	1224
Macro Avg	0.91	0.91	0.91	1224
Weighted Avg	0.91	0.90	0.90	1224

Subsequently, a cross-validation process was conducted to determine the optimal k-value for the K-Nearest Neighbors (KNN) model within the range of 1 to 30. The analysis

identified the most suitable k-value as 17, achieving a mean accuracy of 0.8760 across the cross-validation folds. Leveraging this optimal k-value of 17, a KNN model was constructed and evaluated. The ensuing classification report demonstrates the model's performance metrics for each class, encompassing precision, recall, and F1-score. Class 6 retains its perfect precision, recall, and F1-score. The overall accuracy of the model stands at 90% across all classes, accompanied by macro and weighted average F1-scores of 0.91 and 0.90, respectively. This outcome showcases the effectiveness of the KNN model with a specifically determined k-value in effectively categorizing celestial objects based on their spectral attributes.

IV. DISCUSSION AND ANALYSIS

In the pursuit of understanding the behavior of the K-Nearest Neighbors (KNN) algorithm for stellar classification, an extensive analysis was conducted on a dataset encompassing diverse celestial objects. The KNN algorithm, a fundamental tool in machine learning, is predicated on the principle of assessing data point similarity to make classifications or predictions. Central to this approach is the choice of distance metric, which governs the determination of nearest neighbors.

The theoretical underpinnings of the KNN algorithm were explored, encompassing concepts like distance metrics such as Euclidean, Manhattan, and Cosine similarity. These metrics play a crucial role in defining the proximity between data points. Euclidean distance, which measures straight-line distance, was particularly relevant in the context of this project. It served as a foundational metric to assess similarity in the multidimensional feature space.

Subsequently, the KNN algorithm's classification mechanism was expounded upon. In this method, an object is assigned a class based on the majority class among its k-nearest neighbors. This simple yet effective approach demonstrated prowess in both classification and regression tasks. However, its susceptibility to noise and outliers was acknowledged.

Moving to the empirical analysis, the dataset comprising various spectral classes was explored. With approximately 100,000 instances for each spectral class (A, F, G, K, M), and varying counts for other classes (e.g., 85,673 for B type stars, 26,016 for O type stars), the dataset's richness and diversity were evident. Features like right ascension, declination, parallax, proper motion, and color indices were elucidated, providing essential context for subsequent modeling.

The model's performance was assessed through various iterations of the KNN algorithm. The default KNN model yielded an overall accuracy of 89%, with varying precision, recall, and F1-score metrics for each class. Randomly weighted KNN, and KNN with specifically assigned weights based on feature correlations, were also evaluated. Notably, the latter exhibited improved precision, recall, and F1-scores for some classes, underscoring the significance of feature weights in the algorithm's decision-making process.

Cross-validation was employed to determine the optimal k-value for the KNN model, leading to the selection of k=17. This model achieved an accuracy of 90% and demonstrated enhanced precision and recall for several classes, further emphasizing the importance of tuning hyperparameters for optimal performance.

Additionally, a comprehensive analysis of feature correlations shed light on key insights. Features such as BP-RP, BP-G, and G-RP displayed strong correlation due to their shared derivation from different spectral bands, while the class label exhibited varying degrees of correlation with other features. Features 'pscol', 'G-RP', 'BP-G', and 'BP-RP' were highly correlated with the class label, suggesting their potential significance in distinguishing spectral classes. Conversely, features like 'PM' displayed weaker correlation due to its limited direct relevance to spectral attributes.

This project elucidates the nuances of the K-Nearest Neighbors algorithm in the domain of stellar classification. The fusion of theoretical foundations, empirical analysis, and in-depth exploration of feature correlations forms a comprehensive narrative, revealing both the algorithm's power and its sensitivity to data characteristics. As celestial data continues to expand, the KNN algorithm's adaptability and potential for insights remain ever-relevant.

V. CONCLUSION

In the exploration of the K-Nearest Neighbors (KNN) algorithm for stellar classification, the powerful potential is uncovered. Through rigorous analysis, the core concepts of KNN, from distance metrics to nearest neighbor principles, are navigated. By employing a diverse dataset of celestial attributes, sensitivity to factors like feature weighting and parameter tuning is observed. Notably, optimal k-values are identified using cross-validation, showcasing the algorithm's responsiveness to parameter choices.

The study doesn't stop at algorithmic application. Intriguing feature correlations are unearthed, emphasizing the role of specific color indices in spectral classification. Understanding how features correlate with class labels provides a deeper grasp of feature importance.

Beyond astronomy, the KNN model's applicability spans various domains, from medical diagnosis to image recognition. The outcomes of this project underscore the adaptability and significance of KNN in real-world scenarios.

In summation, the foray into KNN-based stellar classification enriches the comprehension of both machine learning's astronomical applications and the broader versatility of KNN.

References

[1] Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1), 21-27.

[2] Dasarathy, B. V., & Holder, E. B. (1991). Minimum distance classifiers. IEEE Transactions on Systems, Man, and Cybernetics, 21(5), 909-912.

[3] Aha, D. W., & Kibler, D. (1991). Instance-based learning algorithms. Machine learning, 6(1), 37-66.
[4] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 93-104).
[5] Zhang, Z., & Zhan, N. (2010). K-nearest neighbor classification on imbalanced databases. Pattern Recognition Letters, 31(14), 2210-2218.
[6] Almuallim, H., & Dietterich, T. G. (1991). Learning with many irrelevant features. In Proceedings of the ninth national conference on artificial intelligence (Vol. 1, p. 547).
[7] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3), 175-185.
[8] Kim, Y., & Street, W. N. (2008). An empirical comparison of classification algorithms for imbalanced credit scoring data sets. Expert Systems with Applications, 34(4), 2393-2409.
[9] Japkowicz, N., & Shah, M. (2011). Evaluating learning algorithms: A classification perspective. Cambridge University Press.

PRABIN BOHARA is an enthusiastic individual currently pursuing a Bachelor's degree in Computer Engineering, set to graduate in 2024. He has a genuine passion for coding, web technologies, and machine learning. Prabin's primary focus revolves around AI research and exploring the various applications of data science in general. He actively contributes to open-source projects and continuously expands his knowledge through academic endeavors. Prabin actively participates in webinars, both nationally and internationally, as well as competing in diverse technology competitions. He also shares his expertise by conducting workshops on various technology topics. With a strong drive for excellence, Prabin aims to make valuable contributions in the field of AI while keeping up with the latest advancements.

PRABIN SHARMA POUDEL , an undergraduate senior at IOE's Thapathali Campus, has forever been in pursuit of knowledge and wisdom. His involvement extends beyond technical confines, with involvement in non-technical organizations like AYON. With a vision to become a scholarly figure, he aims to bridge multiple disciplines of knowledge. His current field of interest encompasses AI in conflict resolution, Computer vision, and AI in creative industries Beyond academics, Prabin finds joy in playing chess, composing music, and journaling his thoughts through poetry.

SANTOSH PANDEY is a fourth-year Bachelor's student at IOE, Thapathali Campus, with a fervent interest in coding and technology. Presently interning at NAAMII organization, he is devoted to applying his skills in the realms of machine learning, data science, and coding. Santosh is particularly drawn to the fields of full-stack development, graphic designing, mobile app development, and computer vision. He maintains an active engagement in coding competitions and is committed to continuous learning. Santosh's passion for technology and his dedication to enhancing his skills make him a promising talent in the domain of machine learning and data science.

APPENDIX. A: FIGURES AND PLOTS

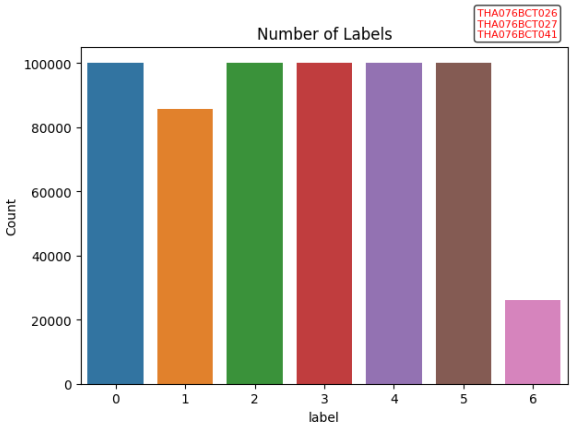

Figure 2: Bar Plot for number of each Class

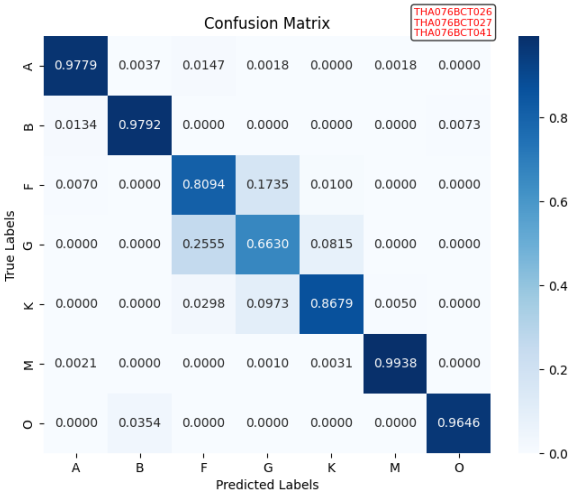

Figure 3: Confusion matrix for default KNN

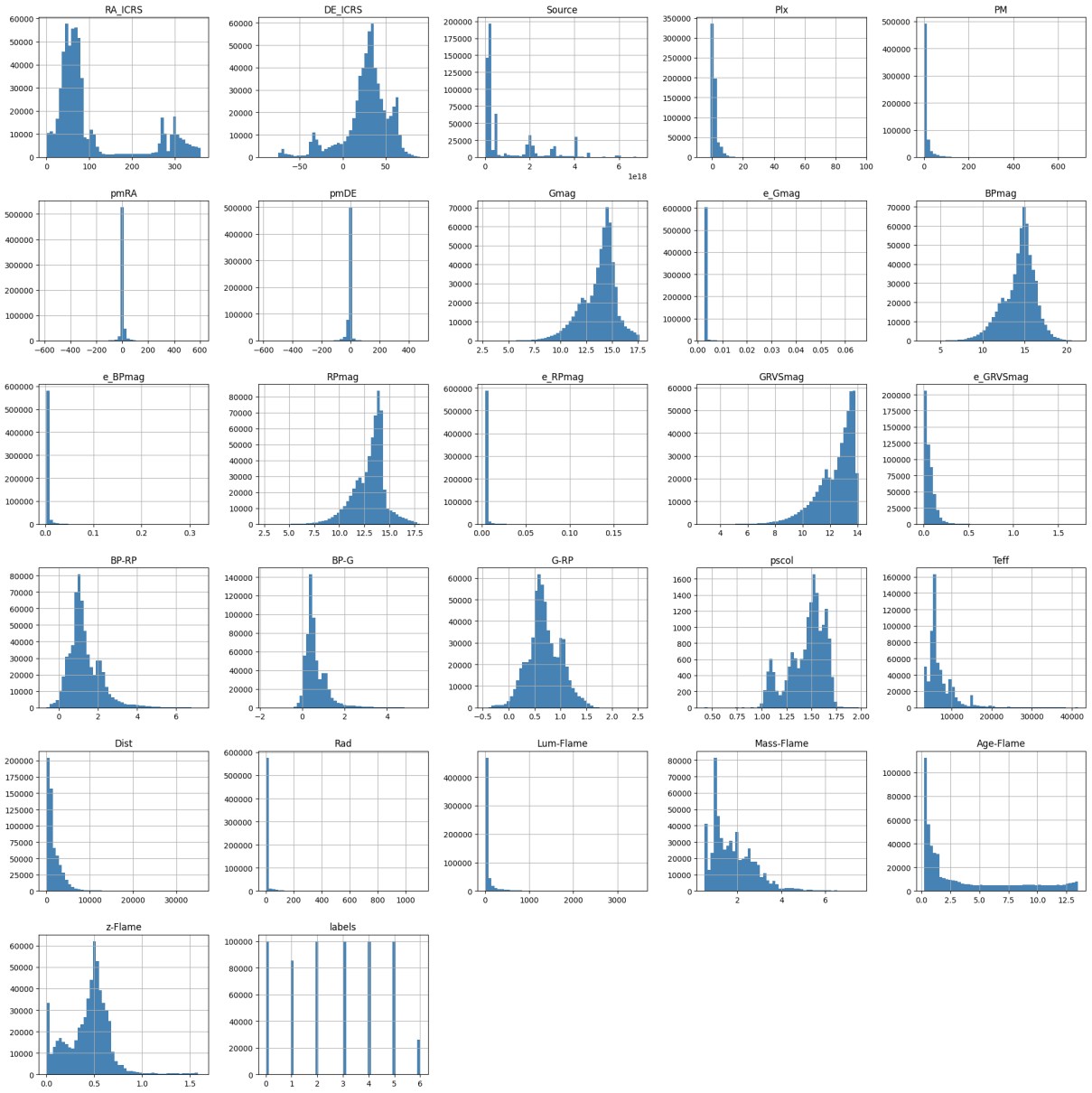

Figure 4: Histogram of features

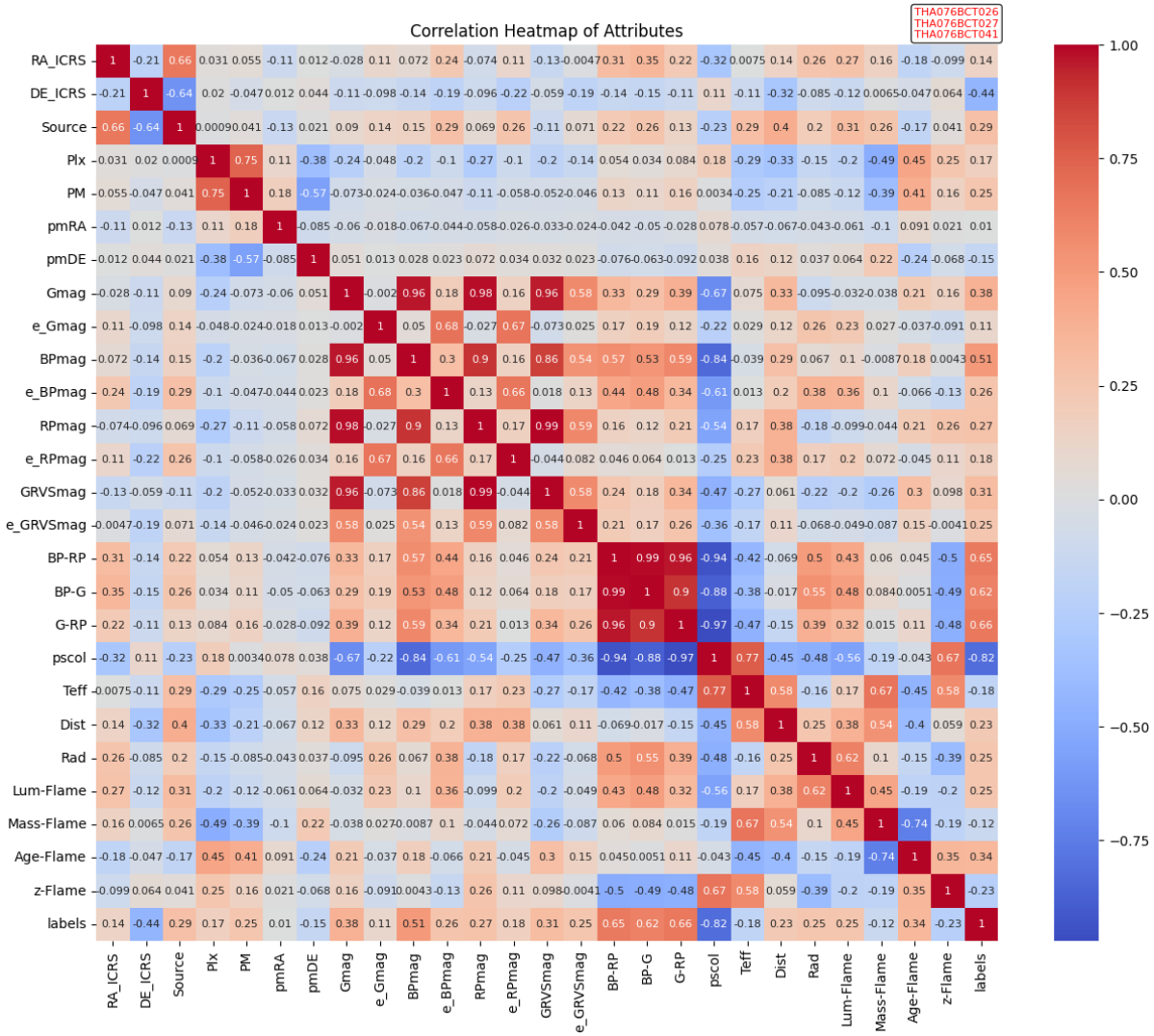

Figure 5: Correlation Heatmap of Attributes

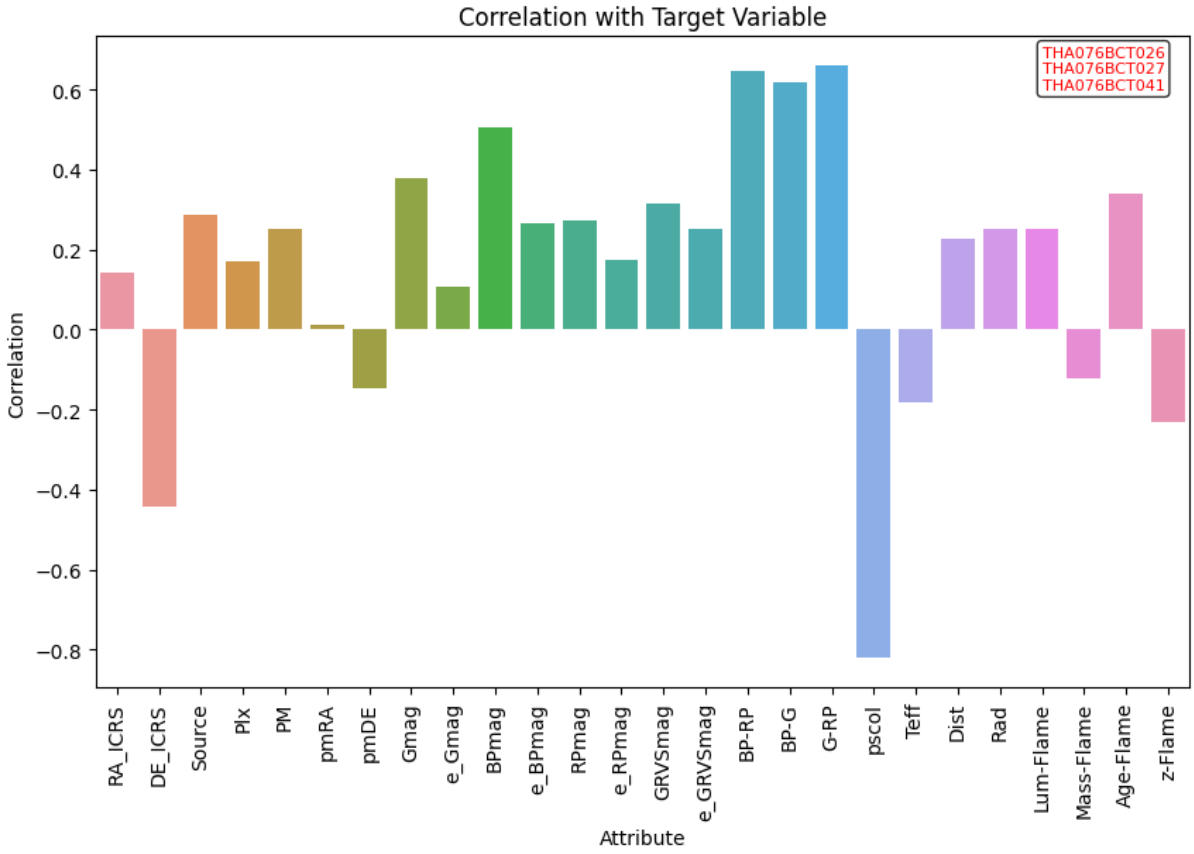

Figure 6: Correlation with Target Variable

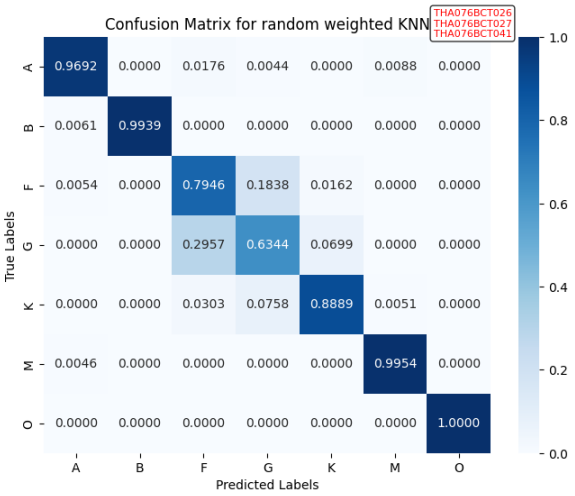

Figure 7: Confusion Matrix for random weighted KNN

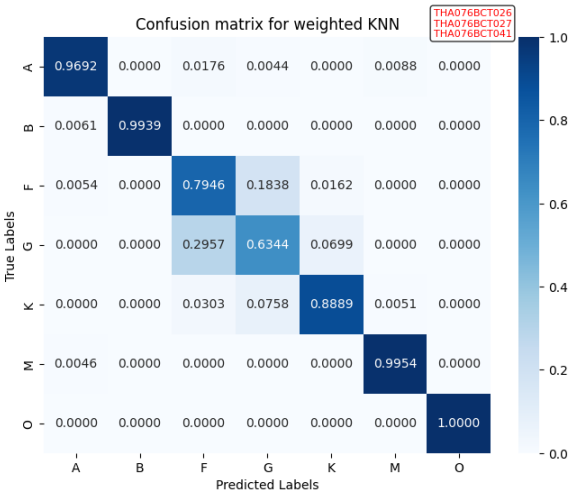

Figure 8: Confusion Matrix for Specifically weighted KNN

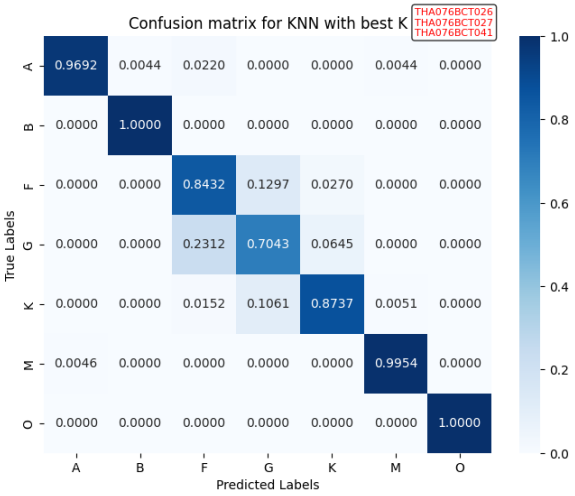

Figure 10: Confusion matrix for KNN with best K-value

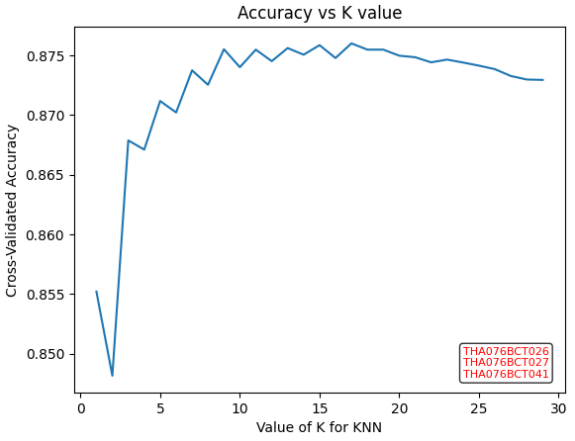

Figure 9: Accuracy vs K value

APPENDIX. APPENDIX: SOURCE CODE

```
""
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt


# replace with your dataset path
dataset_path='./dataset/dataGaia.csv'
df=pd.read_csv(dataset_path)

df.head()
df.describe()

# get the unigue labels
df['SpType-ELS'].unique()

df.drop(columns=['Unnamed: 0'],inplace=True)

# encode the labels
label_encoder = preprocessing.LabelEncoder()


df['labels']= label_encoder.fit_transform(df['SpType-ELS'])

df['labels'].unique()

data=df.drop(columns=['SpType-ELS'])

# Count the number of NaN values in each column
nan_count_per_column = data.isna().sum()

# Count the total number of NaN values in the entire DataFrame
total_nan_count = data.isna().sum().sum()

print("Number of NaN values in each column:")
print(nan_count_per_column)

print("Total number of NaN values in the DataFrame:", total_nan_count)

data.duplicated().sum()

data.head()

import seaborn as sns
import matplotlib.pyplot as plt
classes=['A', 'B', 'F', 'G', 'K', 'M','O']
# Create the count plot
sns.countplot(x='labels', data=data)

text_box = {
    'boxstyle': 'round',
    'facecolor': 'white',
    'alpha': 0.8
}

plt.text(5.2, 108000, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
# Set plot labels
plt.xlabel('label')
plt.ylabel('Count')
plt.title('Number of Labels')
plt.tight_layout()
plt.savefig('output/Number of Label.png',bbox_inches='tight')
# Show the plot
plt.show()
data.hist(bins=50, figsize=(20, 20),color='steelblue')
# the text isn't plotted
plt.text(1, 1, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=100, bbox=text_box, color='red')
plt.tight_layout()
plt.savefig('output/Histogram of features.png', bbox_inches='tight')
plt.show()

X=data.drop(columns=['labels'])
y=data['labels']

# Fill NaN values with the mean of the column
X.fillna(X.mean(), inplace=True)

X.describe()

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)

df_scaled = pd.DataFrame(data=scaled_data, columns=X.columns)
df_scaled.describe()

df_scaled.head()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(df_scaled, df['labels'],test_size=0.01,random_state=43)

from sklearn.neighbors import KNeighborsClassifier

# Instantiate the KNN model with default parameters
knn = KNeighborsClassifier()

x_train.shape,y_train.shape,x_test.shape,y_test.shape

params=knn.get_params()
print(params)

knn.fit(x_train, y_train)
preds=knn.predict(x_test.values)

preds[:20]
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, preds)
print(accuracy)

from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
# Compute the confusion matrix
cm = confusion_matrix(y_test, preds)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
classes=['A', 'B', 'F', 'G', 'K', 'M','O']
# Create a heatmap visualization of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt=".4f", xticklabels=classes, yticklabels=classes, cbar=True)
plt.text(5.8, 0, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.savefig('output/Confusion matrix for default KNN.png',bbox_inches='tight')
plt.show()
from sklearn.metrics import classification_report
# Generate the classification report
report = classification_report(y_test, preds)

print(report)
correlation_matrix = data.corr()

# Plot the correlation heatmap
plt.figure(figsize=(15, 12))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',annot_kws={'fontsize': 8})
plt.title('Correlation Heatmap of Attributes')
plt.text(24.5, -0.2, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.savefig('output/Correlation Heatmap of Attributes.png',bbox_inches='tight')
plt.show()

correlation_with_target = correlation_matrix['labels']
# Plot the correlation values as a barplot
plt.figure(figsize=(10, 6))
```

```
sns.barplot(x=correlation_with_target[:-1].index, y=correlation_with_target[:-1].values,)
plt.title('Correlation with Target Variable')
plt.xlabel('Attribute')
plt.ylabel('Correlation')
plt.xticks(rotation=90)
plt.text(22, 0.6, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.savefig('output/Correlation with Target Variable.png',bbox_inches='tight')
plt.show()

x_train,x_test,y_train,y_test=train_test_split(df_scaled,df['labels'],test_size=0.002,random_state=43)
weights = np.random.rand(26)
weighted_knn=KNeighborsClassifier(metric_params={'w': weights})
print(weights)

weighted_knn.fit(x_train,y_train)

x_test.values.shape

preds1=weighted_knn.predict(x_test.values)


from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
# Compute the confusion matrix
cm1 = confusion_matrix(y_test, preds1)
cm1 = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
classes=['A', 'B', 'F', 'G', 'K', 'M','O']
# Create a heatmap visualization of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm1, annot=True, cmap="Blues", fmt=".4f", xticklabels=classes, yticklabels=classes, cbar=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix for random weighted KNN")
plt.text(6.1, 0, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.savefig('output/Confusion Matrix for random weighted KNN.png',bbox_inches='tight')
plt.show()


from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, preds1)
print(accuracy)


from sklearn.metrics import classification_report
# Generate the classification report
report = classification_report(y_test, preds1)
print(report)

correlation_with_target.index
weights[18]=4
weights[17]=4
weights[16]=3
weights[15]=3

weighted_knn2=KNeighborsClassifier(metric_params={'w': weights})
print(weights)

weighted_knn2.fit(x_train,y_train)
preds2=weighted_knn2.predict(x_test.values)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, preds2)
print(accuracy)
from sklearn.metrics import classification_report
# Generate the classification report
report = classification_report(y_test, preds2)
print(report)
from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
# Compute the confusion matrix
cm2 = confusion_matrix(y_test, preds)
cm2 = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
classes=['A', 'B', 'F', 'G', 'K', 'M','O']
# Create a heatmap visualization of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm2, annot=True, cmap="Blues", fmt=".4f", xticklabels=classes, yticklabels=classes, cbar=True)
plt.text(6.1, 0, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion matrix for weighted KNN")
plt.savefig('output/Confusion matrix for weighted KNN.png',bbox_inches='tight')
plt.show()
from sklearn.model_selection import cross_val_score
# split the dataset to perform validation as validating on whole dataset takes a lot of time since dataset size is huge
x_train1,x_val,y_train1,y_val=train_test_split(df_scaled,df['labels'],test_size=0.1,random_state=43)
# Define a range of K values to test
k_values = range(1, 30)  # You can adjust the range as needed

# Perform k-fold cross-validation
k_scores = []
for k in k_values:
    # Create a KNN classifier object
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_val.values, y_val.values, cv=10, scoring='accuracy',n_jobs=-1)
    k_scores.append(scores.mean())

# Find the K value with the best performance
best_k = k_values[np.argmax(k_scores)]

# Print the results
print("Mean accuracy scores for different K values:")
for k, score in zip(k_values, k_scores):
    print(f"K = {k}: Mean Accuracy = {score:.4f}")

print(f"\nBest K value: {best_k}")

# plot to see clearly
plt.plot(k_values, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
plt.title('Accuracy vs K value')
plt.text(24, 0.848, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.savefig('output/Accuracy vs K value.png',bbox_inches='tight')
plt.show()

# train KNN for best k value
knn = KNeighborsClassifier(n_neighbors=best_k)
x_train,x_test,y_train,y_test=train_test_split(df_scaled,df['labels'],test_size=0.002,random_state=43)
knn.fit(x_train,y_train)

preds=knn.predict(x_test.values)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, preds)
print(accuracy)

from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
# Compute the confusion matrix
cm = confusion_matrix(y_test, preds)
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
classes=['A', 'B', 'F', 'G', 'K', 'M','O']
# Create a heatmap visualization of the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt=".4f", xticklabels=classes, yticklabels=classes, cbar=True)
plt.text(5.8, 0, 'THA076BCT026\nTHA076BCT027\nTHA076BCT041', fontsize=8, bbox=text_box, color='red')
plt.xlabel("Predicted Labels")
```

```
plt.ylabel("True Labels")
plt.title("Confusion matrix for KNN with best K")
plt.savefig('output/Confusion matrix for KNN with best K.
    png',bbox_inches='tight')
plt.show()

from sklearn.metrics import classification_report
# Generate the classification report
report = classification_report(y_test, preds)
print(report)
```