K-Nearest Neighbors (KNN) Algorithm for Classification (January 2024)

Bishwambhar Dahal[1] and Sirjana Bhatta[1]

[1]Department of Electronics and Computer Engineering, IOE, Thapathali Campus, Kathmandu 44600, Nepal
Corresponding author: Bishwambhar Dahal(dahalbishwambhar@gmail.com) and Sirjana Bhatta(sirjanabhatta6@gmail.com)

Abstract The k-Nearest Neighbors (KNN) algorithm is a simple and widely used machine learning algorithm for both classification and regression tasks. This project investigates the performance of both standard KNN algorithm with default parameters and weighted k-Nearest Neighbors (KNN). The study aims to identify the optimal k value using k-fold cross-validation for the standard KNN algorithm and assess the benefits of the Weighted KNN approach in improving predictive accuracy. Using a carefully chosen "Waveform Database Generator" dataset, extensive experiments are conducted with both algorithms to evaluate their performance. The results indicate the superiority of the Weighted KNN algorithm over the standard KNN for certain k values. This research contributes to the understanding of both Weighted KNN and standard KNN algorithms' utility and provides valuable guidance.

Keywords *KNN, Regression, Weighted.*

I Introduction

The use of machine learning [1] algorithms has become increasingly prevalent in various fields, including the analysis and classification of complex datasets. In the field of machine learning, KNN [2] algorithm have proven to be very efficient tool for classification tasks. This algorithm relies on the idea of similarity between data points. They make predictions based on the input features' proximity to other data points in the dataset. KNN is particularly useful for analyzing and classifying complex datasets since it doesn't make assumptions about the underlying data distribution and can handle nonlinear relationships effectively.

In this study, we focus on the application of a KNN algorithm on the "Waveform Database Generator" dataset to predict the type of wave based on a set of key attributes and assess its effectiveness in predicting class of the wave. We seek to identify the most important features that contribute to the classification process.
The k-Nearest Neighbors (KNN) algorithm serves as a fundamental tool in the arsenal of machine learning techniques, applicable to a range of tasks spanning classification and regression. Its simplicity and versatility have rendered it a cornerstone in various domains. This project delves into a comprehensive exploration of the KNN algorithm, extending its examination to encompass both the standard KNN configuration with default parameters and the innovative approach of weighted k-Nearest Neighbors (KNN). The central objectives of this study encompass two distinct aims: the determination of the optimal k value through the utilization of k-fold cross-validation for the standard KNN algorithm, and the thorough evaluation of the efficacy brought forth by the Weighted KNN technique in augmenting predictive accuracy. This evaluation transpires through rigorous experimentation leveraging a meticulously selected "Waveform Database Generator" dataset, which forms the bedrock for this investigation. The outcomes of these extensive experiments unveil noteworthy insights, indicating the outperformance of the Weighted KNN algorithm vis-à-vis the conventional KNN under specific k values. This research makes a notable contribution to the realm of machine learning by shedding light on the nuanced advantages offered by both Weighted KNN and standard KNN methodologies, thereby offering practical insights and guidance to inform algorithmic choices and implementations.

II Methodology

A Brief Theory

The k-Nearest Neighbors (KNN) algorithm is a simple, versatile and widely used machine learning algorithm for both classification and regression tasks. The underlying principle behind KNN is the assumption that data points with similar features tend to belong to the same class.

KNN falls under the category of instance-based learning or lazy learning algorithms, because it does not explicitly build a model during the training phase. Instead, it

"lazily" use the training data directly during prediction. This algorithm retains the entire training dataset in memory and requires more memory space compared to eager learning algorithms that summarize data into a model.

In the context of classification, given a new data point, KNN identifies the k-nearest data points from the training set. Here, "k" refers to the number of nearest neighbors to consider when making a prediction for a new data point. It is a hyperparameter that we need to specify while applying the KNN algorithm. A common practice is to choose an odd value for k to avoid ties in the majority voting process. The hyperparameter "k" plays a significant role in the KNN algorithm's performance. A small value of "k" might lead to noisy predictions, making the model sensitive to outliers, while a large value of "k" can oversmooth decision boundaries and potentially cause underfitting. The choice of "k" involves a trade-off between model bias and variance. Selecting an appropriate value of "k" is essential to ensure the model generalizes well to unseen data and achieves optimal predictive accuracy.

During training, KNN simply stores the entire training dataset, including features and corresponding target labels. When a new data point needs to be classified or predicted, KNN finds the k-nearest neighbors to the new data point from the training dataset based on a chosen distance metric (e.g., Euclidean distance or Minkowski distance) and assigns the class label to the new data point based on the majority class among its k-nearest neighbors.

The choice of distance metric is crucial in the KNN algorithm, as it determines the similarity between data points. Commonly used distance metrics include the Euclidean distance, which considers both the magnitude and direction of differences between feature values, and the Manhattan distance, which measures distances along grid-like paths, considering only absolute differences. Additionally, Minkowski distance is a generalization of other distance metrics like Manhattan(r=1) and Euclidean (r=2) distances. This flexibility allows us to adjust the "shape" of the distance metric by choosing an appropriate value of "r" based on the characteristics of the data and the problem at hand.

The Weighted k-Nearest Neighbors (KNN) algorithm is an extension of the traditional KNN algorithm that enhances the accuracy by assigning different weights to the neighboring data points based on their distances from the query point. The Weighted KNN algorithm offers flexibility in choosing the weighting schemes, which directly influence the contribution of neighboring data points to the final prediction. Similar to KNN, the weighted variant belongs to the category of instance-based or lazy learning algorithms used for both classification and regression tasks. In weighted KNN, the weights can be determined using various weighting schemes, such as correlation coefficient.

Pearson correlation is a statistical measure that quantifies the strength and direction of a linear relationship between two continuous variables. It's widely used to understand how changes in one variable are associated with changes in another. In the context of machine learning, Pearson correlation can be leveraged to assign weights to attributes (features) in the K-Nearest Neighbors (KNN) algorithm. In the KNN algorithm, the idea of assigning weights to attributes based on Pearson correlation is to give more importance to features that exhibit a stronger linear relationship with the target variable. This approach assumes that features with higher correlation are more relevant in predicting the target variable. The Pearson correlation coefficient ranges from -1 to +1, where +1 indicates a strong positive correlation, -1 indicates a strong negative correlation, 0 indicates no correlation.
Z-score normalization, also known as standardization, is a data preprocessing technique widely used in the k-Nearest Neighbors (KNN) algorithm. It aims to scale and transform the features of a dataset to have a mean of 0 and a standard deviation of 1. This transformation ensures that all features contribute equally during the distance calculations in KNN, preventing certain features with larger magnitudes from dominating the similarity measurements. By normalizing the features using the z-score, KNN becomes less sensitive to differences in feature scales and magnitudes. Consequently, the algorithm can make more reliable predictions by giving appropriate weight to each feature during the nearest neighbor search. The z-score normalization should be applied to the training data before training the KNN model. During the prediction phase, the same normalization parameters (mean and standard deviation) calculated from the training data are applied to the new, unseen data points to ensure consistent scaling. It enhances the model's ability to capture meaningful patterns and relationships in the data while mitigating the impact of disparate feature magnitudes on the final outcomes. By normalizing the data, you bring all the features to a common scale, and this helps the model to treat all features equally in the learning process.

To apply z-score normalization to the dataset, for each feature, you calculate the mean and standard deviation of the feature across all data points. Then, you subtract the

mean from each data point and divide the result by the standard deviation. This process centers the data around zero with a standard deviation of 1.

Cross-validation is a technique for evaluating model performance, especially when dealing with limited data. It helps in assessing how well a model is likely to perform on unseen data. K-fold cross-validation is a specific variant of cross-validation where the dataset is divided into k subsets (folds) for training and validation.

B Mathematical Formulae

The Minkowski distance formula between two data points p and q in a feature space with n dimensions is given by:

$$d(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^r \right)^{\frac{1}{r}} \tag{1}$$

where,

- p_i and q_i are the values of the i-th feature of data points p and q, respectively,
- r is a positive constant representing the order of the Minkowski distance. It is a user-defined hyperparameter that determines the "shape" of the distance metric.

Let's assume we have a dataset 'X' with m records and n attributes i.e. having dimension $m \times n$. For each attribute j, the mean can be calculated as:

$$\mu = \frac{\sum_{i=1}^{m} x_{ij}}{m} \tag{2}$$

We apply z-score normalization along each attribute and the formula for z-score normalization is as follows:

$$z = \frac{x - \mu}{\sigma} \tag{3}$$

where,

- x is the original value of a data point in the dataset,
- μ is the mean of the feature,
- σ is the standard deviation of the feature.

The formula for calculating the Pearson correlation coefficient 'r' between two variables X and Y is:

$$r = \frac{\sum((X - \bar{X}) \cdot (Y - \bar{Y}))}{\sqrt{\sum(X - \bar{X})^2 \cdot \sum(Y - \bar{Y})^2}} \tag{4}$$

where,

- X and Y are the values of the two variables,
- $\bar{X}$ and $\bar{Y}$ are the means of X and Y.

The formula for accuracy is used to evaluate the performance of a classification model, including k-Nearest Neighbors (KNN). The formula for accuracy is as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5}$$

Mathematically, if TP represents the number of true positives (correctly predicted positive instances), TN represents the number of true negatives (correctly predicted negative instances), FP represents the number of false positives (incorrectly predicted positive instances), and FN represents the number of false negatives (incorrectly predicted negative instances), then the accuracy formula can be expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

C System Block Diagram

The Block diagram of the system is shown in figure (1).

D Working Principle

The process begins with a labeled dataset, i.e., the dataset name, containing features and corresponding class labels. To ensure that features are on a comparable scale and to prevent any particular feature from dominating the learning process, z-score normalization (standardization) is applied to the features. Z-score normalization transforms each feature to have a zero mean and unit variance, making it especially valuable when dealing with attributes that have different units or widely varying ranges.

To accurately evaluate the performance of the KNN model, the dataset is divided into two subsets: an 80% training set and a 20% testing set. The training set is utilized to train the KNN model, while the testing set is used to assess the model's accuracy.

A default KNN model is initialized with hyperparameters set to standard default values. In this case, the number of neighbors, denoted by 'k,' is set to 5, and the distance metric chosen is the Minkowski distance. The default KNN model is trained on the training data. During the training process, it memorizes the entire training dataset without explicitly learning model parameters; that's why it is considered a "lazy learner" approach. It stores the training instances in memory for later use during the prediction phase. During the testing or inference phase, predictions are made based on the k-nearest neighbors. The algorithm does not undergo a traditional training

phase with parameter learning.

Using the test set, the accuracy of the default KNN model is evaluated. During the testing or inference phase, predictions are made for each data point based on the distances between the data point and its k-nearest neighbors in the training set. For classification tasks, the majority class among the k-nearest neighbors determines the predicted class label of the new data point.

Furthermore, a randomly weighted KNN model is initialized, trained on the training set, and its accuracy is evaluated. To potentially improve model performance, a properly weighted KNN model is experimented with. For this, the correlation between various features and the class labels is analyzed. Based on this analysis, weights are assigned to the top two positively correlated features. The weights are assigned in ascending order of correlation strength, where the most correlated feature receives weight 1 and the second most correlated feature receives weight 2. This KNN model is again trained on the training set, considering both the feature values and their respective weights. During evaluation, predictions are made in a similar manner as with the aforementioned KNN model, but the weighted distances are taken into account to influence neighbor selection.

A systematic process is also executed to determine the optimal number of neighbors (k) for the KNN algorithm. By using a range of odd k values from 1 to 30, k-Fold cross-validation with 10 folds is applied for each k. This process assesses the model's accuracy across different subsets of the data, producing a mean accuracy for each k. The value of k that yielded the highest average performance across the folds was selected as the best k for that particular variant. Finally, the best k value and the corresponding accuracy score are evaluated. To visualize accuracy for different k values, a graph of k versus accuracy is plotted. The optimal 'k' is the value where the accuracy is highest. With the chosen optimal 'k,' both the unweighted KNN and weighted KNN models are re-implemented to evaluate their performance.

To visualize the performance of a classification model, a confusion matrix is constructed, and a classification report is generated. The confusion matrix and classification report provide a summary of the predictions made by the model on a test dataset. We evaluated the model's performance using appropriate evaluation metrics, such as accuracy, precision, recall, and F1-score.

E Instrumentation Details

In our study, we utilized the Python programming language to implement the KNN algorithm. The implementation was done in Jupyter Notebook, which is an interactive computing notebook environment. To begin, we import necessary libraries like numpy for numerical computations, matplotlib.pyplot for plotting visualizations, and scikit-learn's KNeighborsClassifier and `cross_val_score` for constructing the KNN model and conducting cross-validation, respectively. To read the dataset present in csv file for our machine learning task, we import pandas library, use it's `pd.read_csv` function. Additionally, we employed the `train_test_split()` function from the `sklearn.model_selection` module to split our dataset into training and test sets. This function takes parameters such as data, target, test_size, random_state and stratify. For evaluating the performance of our model, we utilized the `confusion_matrix()` function from the sklearn.metrics module to construct a confusion matrix and using `heatmap` function we draw the heatmap for every confusion matrices. Furthermore, we generated a classification report using the `classification_report()` function from the same module.

To initialize a KNN model we used `KNeighborsClassifier` class of sklearn and to visualize its parameters we used `get_params` method. For correlation analysis we used `corr` function with the method set to "pearson". We visualized the correlations using `heatmap` function. Using `random.rand` function of numpy we randomly initialized the weights. To instatiate weighted KNN model, we passed value 'w':weights to metric_params parameter while initialization.

Employing `cross_val_score`, the model undergoes k-fold cross-validation with 10 folds, and accuracy scores are accumulated. The accuracy scores obtained from different folds are aggregated to calculate the mean accuracy for the current 'k'. The best k value is found for which the accuracy is maximum. This is done using `argmax` function of numpy.

Overall, these libraries and functions were employed to implement and evaluate the decision tree classifier algorithm, as well as to visualize the results in a clear and informative manner.

F Dataset Description

The dataset titled ”Waveform Database Generator (written in C)” originates from the work of Breiman, Friedman, Olshen, and Stone’s book ”Classification and Regression Trees” (1984). It was contributed by David Aha and dates back to 11/10/1988. Past applications of this dataset encompass evaluations of different algorithms, including the Optimal Bayes classification rate, CART decision tree algorithm, and Nearest Neighbor Algorithm. Notably, these algorithms achieved varying accuracies ranging from 72% to 86%. The dataset comprises instances categorized into three distinct classes of waves, with each instance characterized by 21 attributes. This dataset does not contain missing attribute values, and the classes are evenly distributed, each accounting for 33% of the instances. In essence, the ”Waveform Database Generator (written in C)” dataset presents a structured collection with intrinsic noise, lending itself to evaluations of diverse algorithms and providing insights into the interplay between attributes and classes.

III EXPERIMENTAL RESULTS

In figure 3, we can visualize the correlation between the features of our dataset. From figure 4 we can choose the most positively correlated features of the dataset which will be useful during weight initialization.

The classification report for default KNN model is shown in figure 5. We see that the accuracy for default KNN for which no.ofnearestnegihbor=5 is found to be 80.30%. For the same model, the confusion matrix can be seen in figure 6.

Then we initialized a randomly weighted KNN model with default value of k i.e.k-5. In this approach, we assigned random weights to all features used in the kNN algorithm. This allowed us to observe the impact of varying feature importance on the model’s performance. The classification report for this KNN model is shown in figure 7. We see that the accuracy for default KNN for which no._of_nearest_neighbor=5 is found to be 75.90%. For the same model, the confusion matrix can be seen in figure 8.

The classification report for weighted kNN model with weights assigned manually to Top 4 Positively Correlated Features can be visualized in figure 9 and the confusion matrix is shown in figure 10. The accuracy of the model when the weights of top 4 positively correltaed features are updated manually as 1,2,3 and 4 is found to be 77.80%.

To determine the optimal value of k for each variant, we performed 10-fold cross-validation. The accuracies for different k-values are shown in figure 11. The graphical visualization of this result is shown in figure 12. From this analysis, the best value of k is found to be 25.

The kNN model initialized with number_of_neighbors equal to the best k value i.e. 25 is also evaluated. The classification report for this tuned model is presented in 9, displaying an accuracy of 83.30%. Additionally, the corresponding confusion matrix is depicted in Figure 10.

Overall the kNN model initialized with best_k outperformed the other models with an accuracy of 83.30%.

IV Discussion and Analysis

The conducted experiments with various k-Nearest Neighbors (kNN) model variations have yielded insightful findings. The results above show the average accuracy for different values of k in the k-nearest neighbors (KNN) algorithm. The baseline model, with k=5, achieved an accuracy of 80.30%. This initial accuracy served as a reference point for evaluating subsequent models.

The accuracy of randomly weighted kNN model is dropped to 75.90%. This is because of the random nature of the weights assigned to the features. While randomness can sometimes provide unexpected insights, it can also introduce noise and disrupt the natural patterns within the data. Each feature’s contribution to the distance calculations becomes arbitrary, and features that might have been important are now given equal weight to those that might be less relevant. So, when dealing with feature weights, it’s crucial to have a rationale behind weight assignment rather than relying on randomness. The decrease in accuracy emphasizes the importance of informed feature weighting strategies.

In the model where we assigned weights to the top 4 positively correlated features in a descending order (1, 2, 3, 4). The weighted assignment reflects a hierarchical ranking of feature importance, enabling the model to recognize the most influential features intuitively. The interplay between assigning weights to the top 4 positively correlated features and applying random weights to others can explain the observed trend of improved accuracy compared to the randomly weighted model, yet falling short of the default kNN model’s accuracy. The combination of positively correlated features with carefully chosen weights and randomly weighted features could dilute the meaningful patterns captured by the positively correlated features. The loss in accuracy tha default kNN might be due to the loss of stability and consistency introduced by the random weights.

It seems that the accuracy values vary depending on whether you perform 10-fold cross-validation or use a single train-test split. K-fold cross-validation helps in mitigating overfitting by providing a more comprehensive evaluation of the model's generalization performance. It is very benfecial to use cross-validation for model evaluation and hyperparameter tuning to obtain a more reliable estimate of the model's performance. The performance metrics obtained with cross-validation are likely to be closer to the model's true performance on unseen data. K-fold validation also helps in more robust assessment of the model. For this assessment, we used odd values from 1 to 30 as number of neighbors. We used odd values of "k" only because with an odd "k," there's no possibility of an equal number of nearest neighbors from different classes. Choosing a very high value of "k" can lead to underfitting. So, we limited the possible values of "k" within 30.

V Conclusion

In this lab, our exploration of the k-Nearest Neighbors (kNN) algorithm using the Waveform Database Generator dataset yielded valuable insights into its mechanics and adaptability. Through the effective utilization of Python libraries and the meticulous execution of various kNN models, we gained a profound understanding of its underlying principles. Beginning with the default kNN model, we established a performance baseline, which set the stage for subsequent enhancements. The introduction of the randomly weighted kNN model allowed us to assess the influence of weight distribution on predictive accuracy, emphasizing the importance of balanced contributions from neighbors. Leveraging correlations as weights in the correlation-based weighted model highlighted the significance of attribute interactions. Notably, the optimization of the nearest neighbor count, undertaken to fine-tune the algorithm's predictions, illustrated the critical nature of parameter tuning. Our evaluation process encompassed a range of robust metrics, including accuracy, precision, recall, and F1-score, enabling a comprehensive comparison of the models. Ultimately, this project elucidated the versatility of kNN, revealing its adaptability across different scenarios. The hands-on implementation and meticulous evaluation fostered a deep appreciation for kNN's capabilities, empowering us with the ability to leverage this algorithm for diverse datasets and predictive tasks.

REFERENCES

[1] Z.-H. Zhou, *Machine learning*. Springer Nature, 2021.

[2] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

Bishwambhar Dahal is a fourth-year student of Electronics, Communication, and Information Engineering. With a deep fascination for Artificial Intelligence (AI), He is driven by the potential of AI to transform industries and tackle complex challenges. His academic journey has equipped him with a strong foundation in AI concepts, including machine learning and data analysis. He possess a relentless curiosity and is always eager to explore the latest advancements in AI. His goal is to apply his knowledge and skills in AI to make meaningful contributions to research and development, pushing the boundaries of what is possible with intelligent algorithms.(THA076BEI009)

Sirjana Bhatta is a fourth-year student of Electronics, Communication, and Information Engineering with a keen interest in the field of Artificial Intelligence (AI). She possess a strong academic foundation and practical skills in machine learning, deep learning, and data analysis. Her passion lies in leveraging AI to revolutionize industries and solve complex problems. She is a motivated learner, constantly staying updated with the latest advancements in AI. She is seeking opportunities to contribute to AI research and development and make a positive impact on society. (THA076BEI036)

Appendix

A Figures

1 Block Diagram

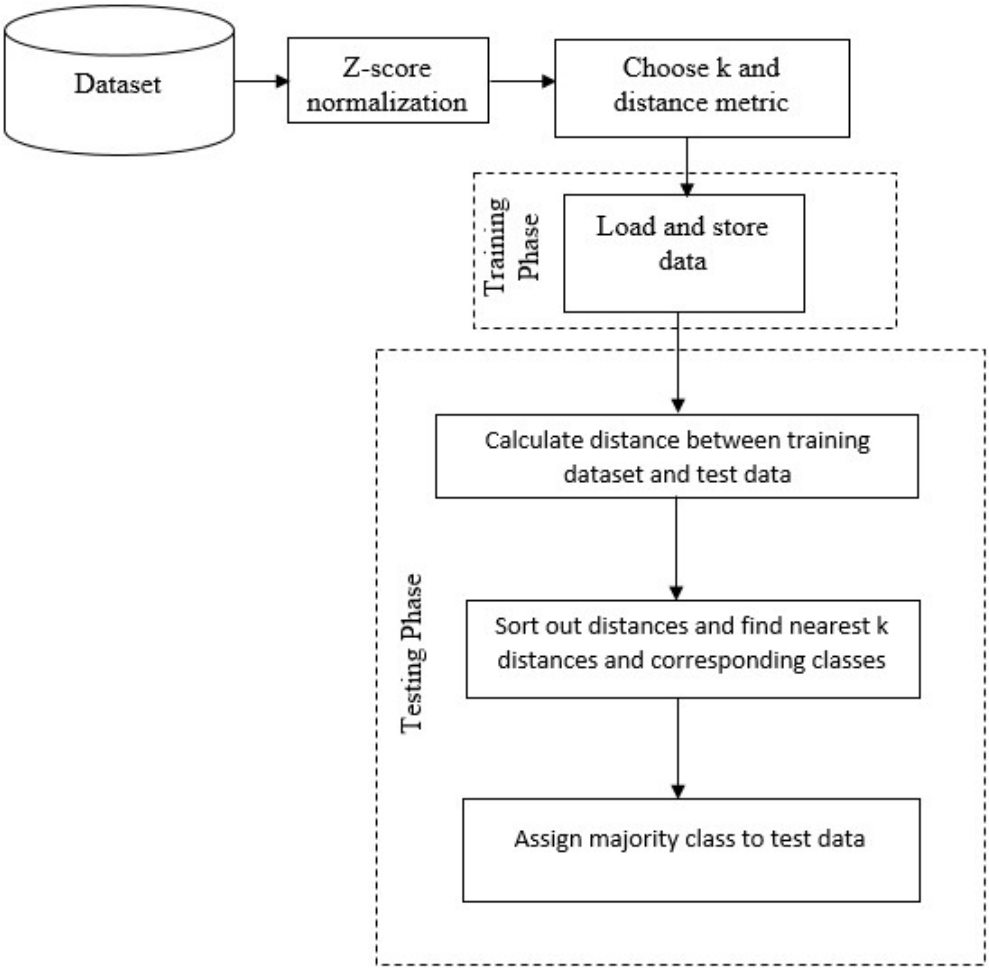

Figure 1: Block Diagram of the system

2 Dataset Visualization

	Source Port	Destination Port	NAT Source Port	NAT Destination Port	Action	Bytes	Bytes Sent	Bytes Received	Packets	Elapsed Time (sec)	pkts_sent	pkts_received	en_action
0	57222	53	54587	53	allow	177	94	83	2	30	1	1	0
1	56258	3389	56258	3389	allow	4768	1600	3168	19	17	10	9	0
2	6881	50321	43265	50321	allow	238	118	120	2	1199	1	1	0
3	50553	3389	50553	3389	allow	3327	1438	1889	15	17	8	7	0
4	50002	443	45848	443	allow	25358	6778	18580	31	16	13	18	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
65527	63691	80	13237	80	allow	314	192	122	6	15	4	2	0
65528	50964	80	13485	80	allow	4680740	67312	4613428	4675	77	985	3690	0
65529	54871	445	0	0	drop	70	70	0	1	0	1	0	2
65530	54870	445	0	0	drop	70	70	0	1	0	1	0	2
65531	54867	445	0	0	drop	70	70	0	1	0	1	0	2

65478 rows × 13 columns

Figure 2: Dataset

3 Result

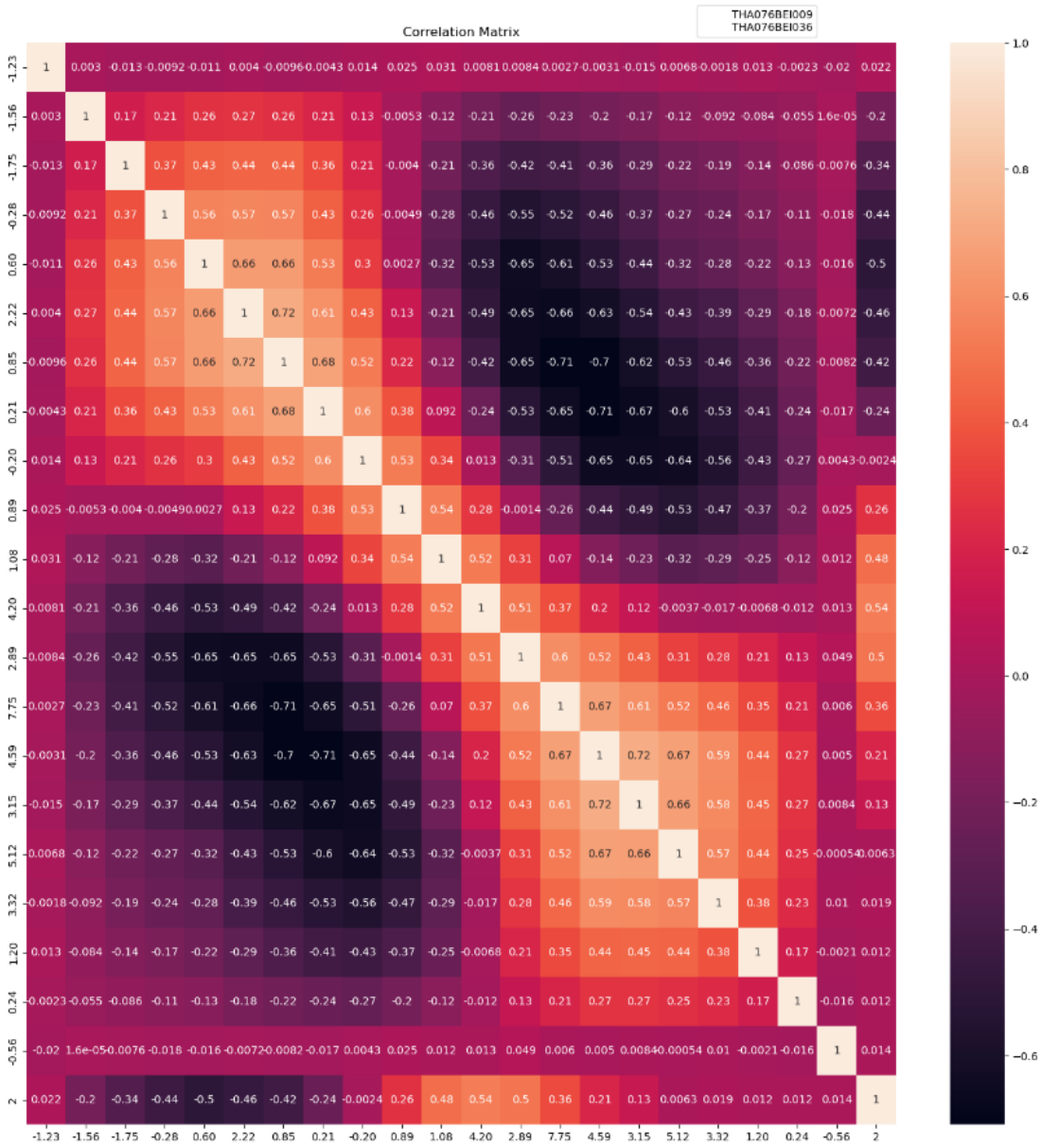

Figure 3: HeatMap of Correlation Matrix

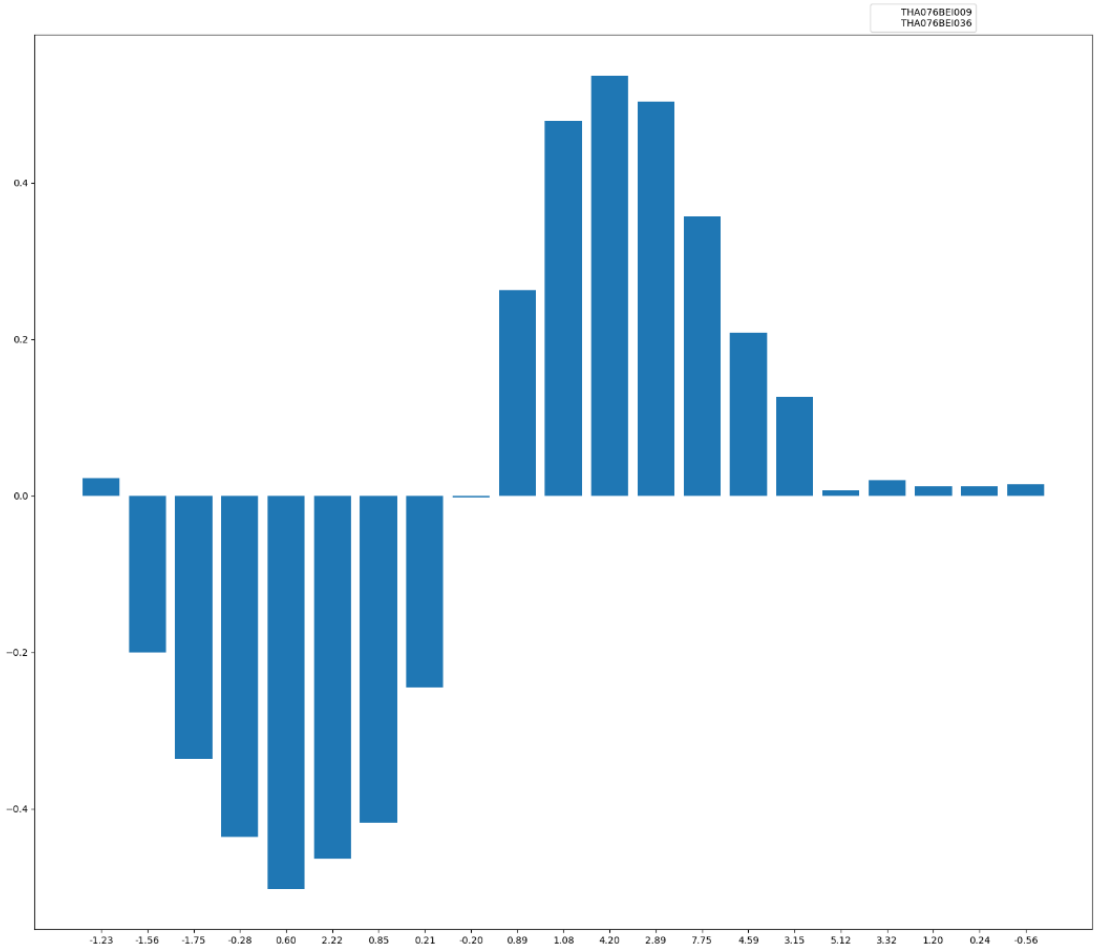

Figure 4: correlation matrix bar plots

	precision	recall	f1-score	support
0	0.8484	0.7078	0.7718	332
1	0.7753	0.8602	0.8156	329
2	0.7961	0.8407	0.8178	339
accuracy			0.8030	1000
macro avg	0.8066	0.8029	0.8017	1000
weighted avg	0.8066	0.8030	0.8018	1000

Figure 5: Classification report for default KNN

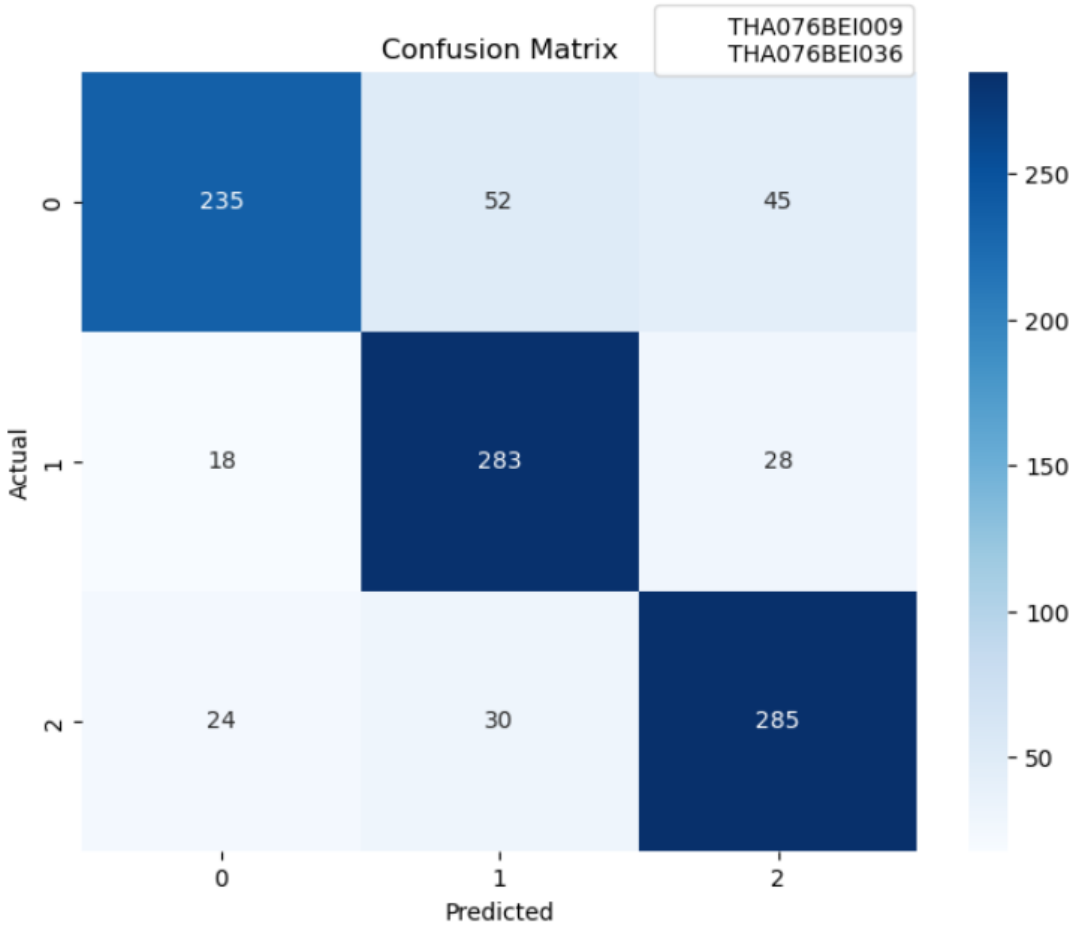

Figure 6: Confusion Matrix for default KNN

	precision	recall	f1-score	support
0	0.7687	0.6807	0.7220	332
1	0.7393	0.7842	0.7611	329
2	0.7703	0.8112	0.7902	339
accuracy			0.7590	1000
macro avg	0.7594	0.7587	0.7578	1000
weighted avg	0.7596	0.7590	0.7580	1000

Figure 7: Classification report for randomly weight intialized KNN

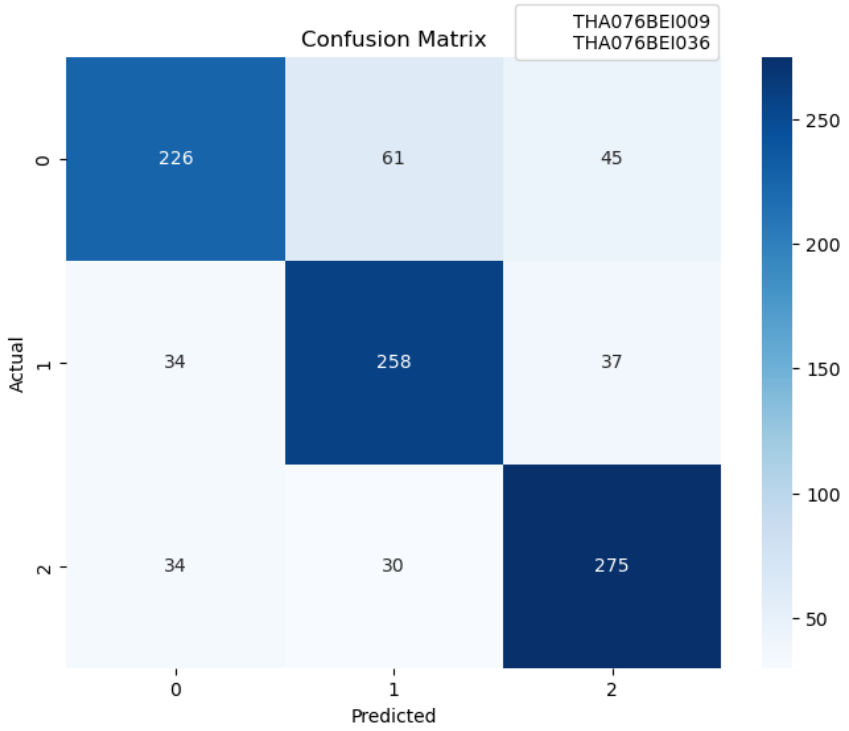

Figure 8: Confusion matrix for randomly weight intialized KNN

	precision	recall	f1-score	support
0	0.8121	0.7289	0.7683	332
1	0.7761	0.7690	0.7725	329
2	0.7527	0.8348	0.7916	339
accuracy			0.7780	1000
macro avg	0.7803	0.7776	0.7775	1000
weighted avg	0.7801	0.7780	0.7776	1000

Figure 9: Classification report for KNN with weights updated based on correlation

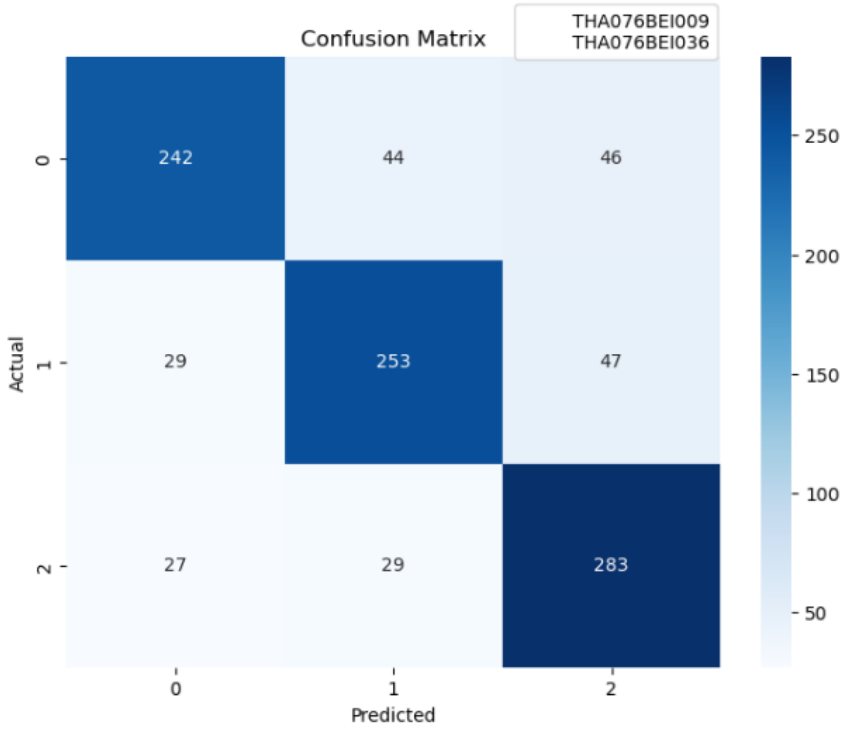

Figure 10: Confusion matrix for KNN with weights updated based on correlation

```
Average Accuracy for k=1: 0.7655
Average Accuracy for k=3: 0.7970
Average Accuracy for k=5: 0.8160
Average Accuracy for k=7: 0.8286
Average Accuracy for k=9: 0.8312
Average Accuracy for k=11: 0.8354
Average Accuracy for k=13: 0.8366
Average Accuracy for k=15: 0.8428
Average Accuracy for k=17: 0.8438
Average Accuracy for k=19: 0.8424
Average Accuracy for k=21: 0.8434
Average Accuracy for k=23: 0.8486
Average Accuracy for k=25: 0.8510
Average Accuracy for k=27: 0.8476
Average Accuracy for k=29: 0.8476
```

Figure 11: Accuracy for different values of k with 10-fold cross-validation

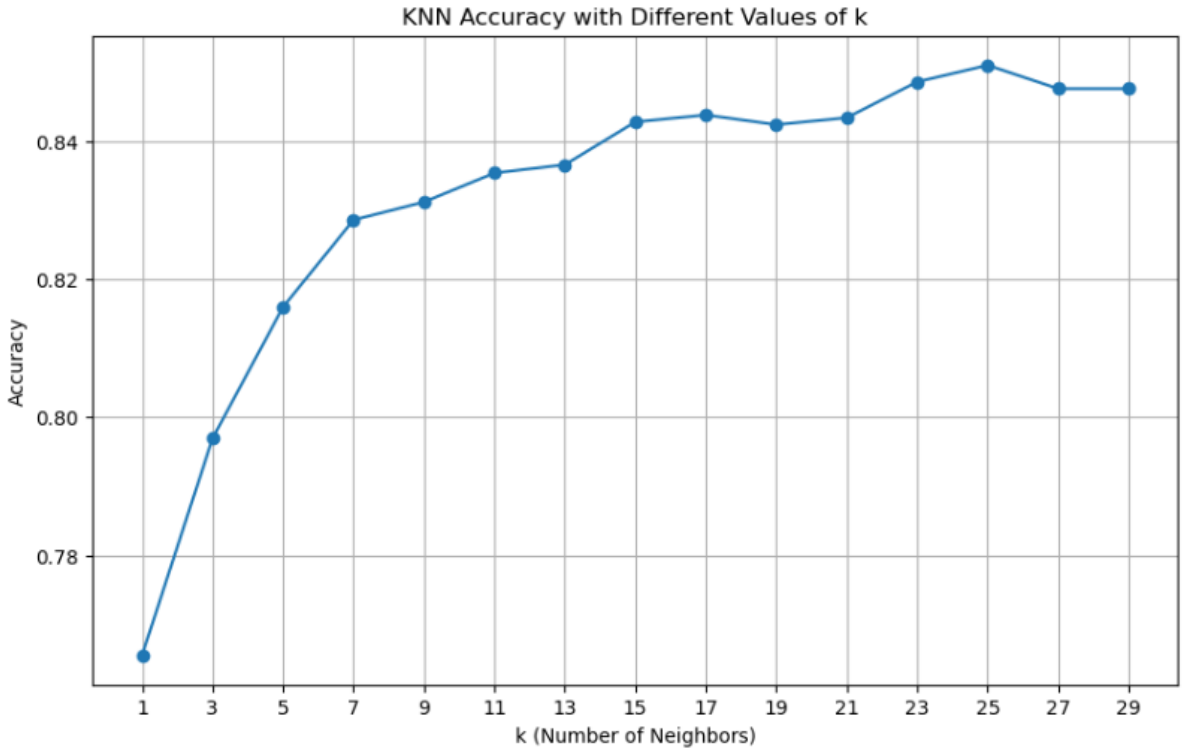

Figure 12: Graph of k vs accuracy for different values of k with 10-fold cross-validation

	precision	recall	f1-score	support
0	0.9004	0.7078	0.7926	332
1	0.8056	0.8815	0.8418	329
2	0.8127	0.9086	0.8579	339
accuracy			0.8330	1000
macro avg	0.8395	0.8326	0.8308	1000
weighted avg	0.8394	0.8330	0.8309	1000

Figure 13: Classification report for weighted KNN with best-K

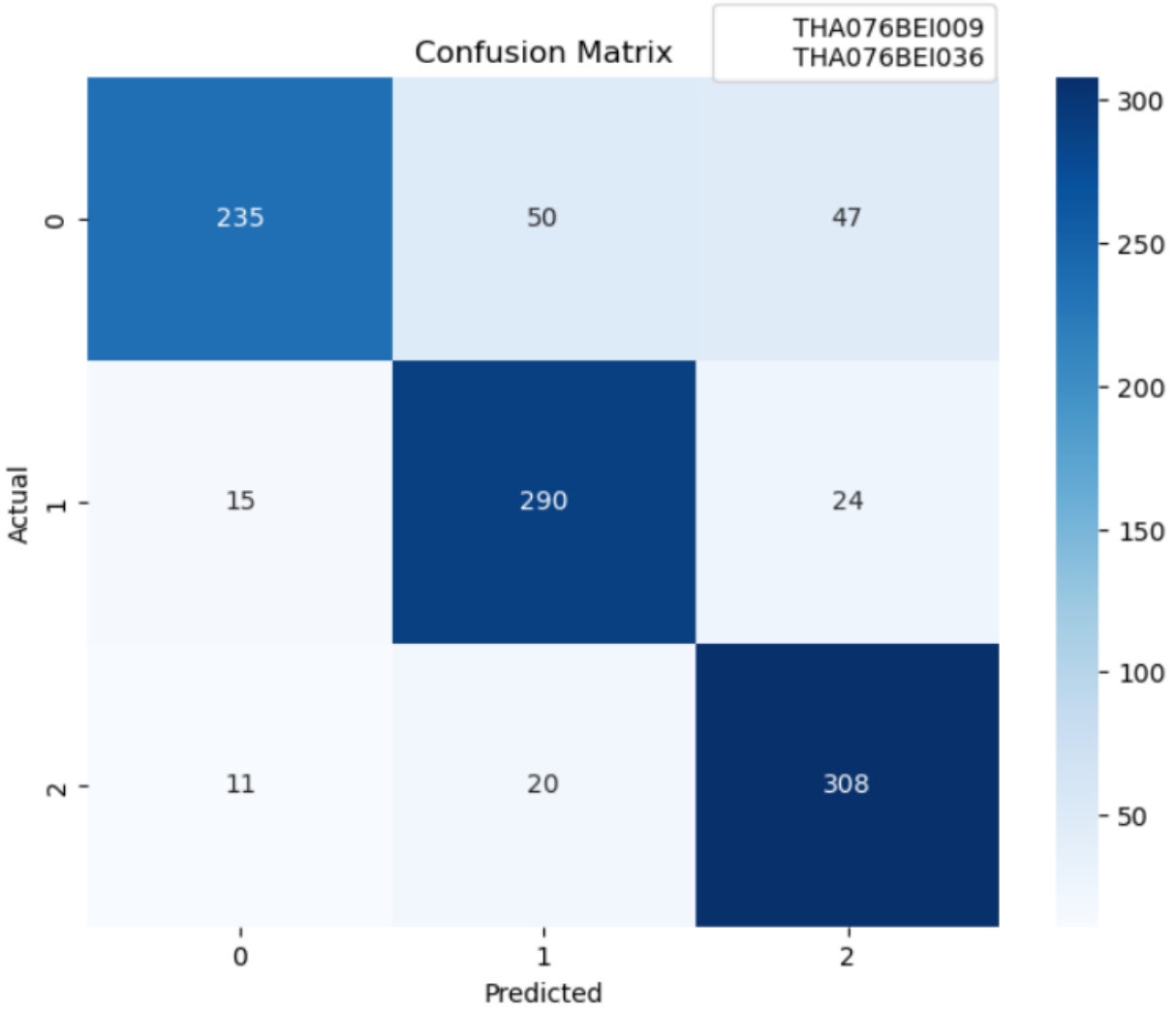

Figure 14: Confusion matrix for weighted KNN with best-K

	precision	recall	f1-score	support
0	0.9004	0.7078	0.7926	332
1	0.8056	0.8815	0.8418	329
2	0.8127	0.9086	0.8579	339
accuracy			0.8330	1000
macro avg	0.8395	0.8326	0.8308	1000
weighted avg	0.8394	0.8330	0.8309	1000

Figure 15: Classification report for weighted KNN with best-K

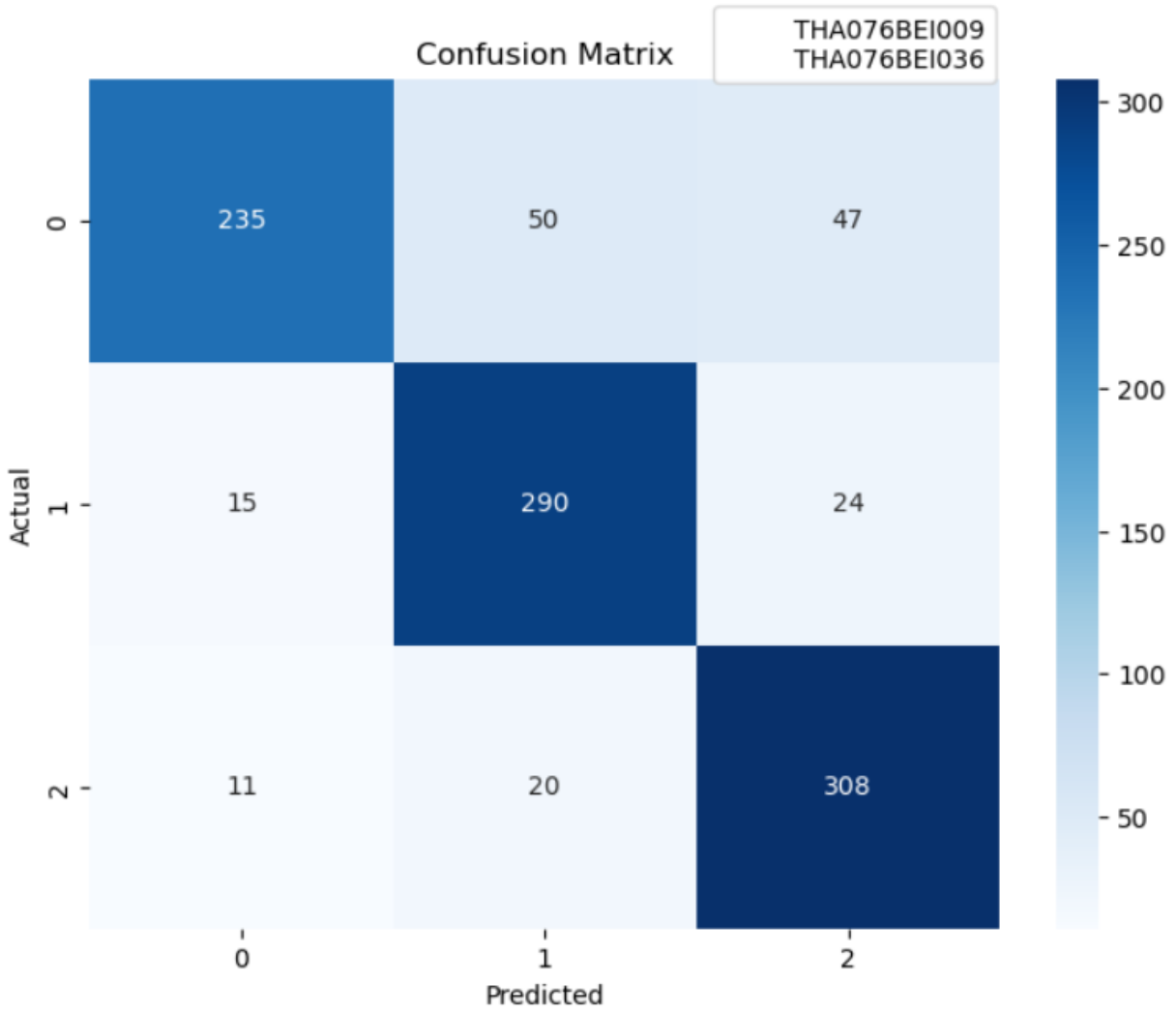

Figure 16: Confusion matrix for weighted KNN with best-K