Assignment 2: Data Modelling

Linh Thuy Do (s3927777)

Table of Contents

Task 1: Regression

1. Introduction

The objective of this task is to explore the relationship between alcohol content and density in a given dataset. Using a sample of 200 instances, we aim to model this relationship through a simple linear regression. The dependent variable in this case is alcohol, and the independent variable is density. We will visually explore the relationship between these two variables and evaluate the linear model.

2. Data Preparation

The dataset consists of several chemical properties of wines, including **alcohol** and **density**. Before proceeding with the analysis, the following steps were undertaken to ensure the data is suitable for modeling:

- **Loading the dataset**: The dataset was imported from A2data.csv, which includes various chemical attributes of wine.
- **Data cleaning**: Any missing values in the alcohol and density columns were identified and removed. This step is essential because missing values can lead to erroneous model estimates.
- **Random sampling**: To make the computational process more efficient, a random sample of 200 instances was drawn from the cleaned dataset. This random selection ensures that the sample is representative of the overall population while reducing potential biases.

After sampling, we validated that there were no missing values, and the data was in the correct format for both alcohol and density, ensuring the dataset was fully prepared for further analysis.

3. Simple Linear Model

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables (Stats Made Easy, 2022). In this task, **simple linear regression** was employed, which involves modeling the dependent variable (alcohol) as a linear function of a single independent variable (density). Therefore, the general equation is:

$$\text{Alcohol} = \beta_0 + \beta_1 \times \text{Density} + \epsilon$$

Where:

- β_0 is the **intercept**,
- β_1 is the **coefficient** for density, which indicates the magnitude and direction of the effect of density on alcohol.
- ϵ is the error term.

Using the sampled dataset, the linear regression model was fitted to the data, yielding the following regression equation:

$$\text{Alcohol} = 354.38 - 345.97 \times \text{Density}$$

⇨ **Interpretation of Coefficients:**

- ***Intercept ($\beta_0 = 354.38$)***: This value represents the alcohol content when the density is zero. While this value is not meaningful in a practical sense (since density cannot be zero), it is mathematically necessary for the regression model.
- ***Coefficient for Density ($\beta_1 = -345.97$)***: The coefficient for density is -345.97, which suggests that for each unit increase in density, the alcohol content decreases by approximately 345.97 units, holding all other factors constant. This large negative coefficient emphasizes the strong inverse relationship between density and alcohol. A small change in density results in a significant decrease in alcohol content.

4. Visualizing

a. Scatter Plot

A scatter plot was created to visualize the relationship between alcohol content and density using a random sample of 200 data points. This plot shows alcohol on the y-axis and density on the x-axis. Each blue dot on the graph represents a data point from the sample. To further evaluate the fit of the linear model, the regression line was superimposed on the scatter plot. This visualization, presented in *Figure 1*, shows how well the model captures the observed data points.

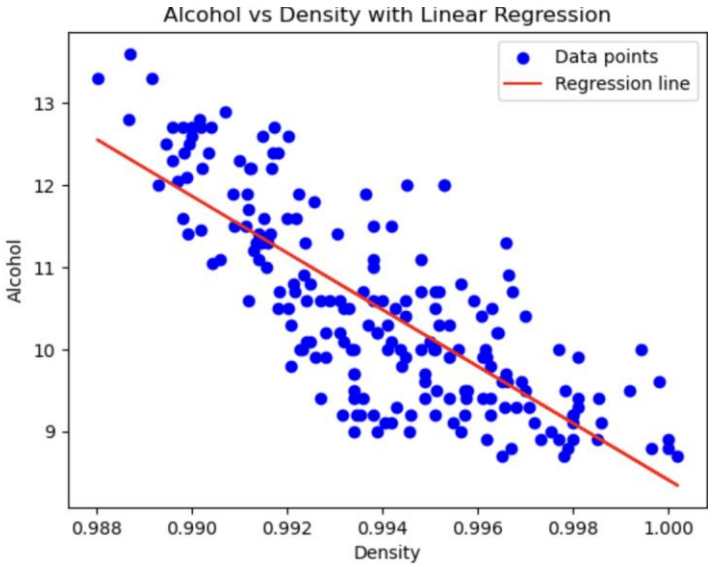

Figure 1: Scatter Plot with Linear Regression Line

b. Interpretation

The regression line fits the data reasonably well, capturing the overall negative trend between alcohol and density. Most of the data points lie close to the regression line, further indicating that the model captures a significant portion of the variance in alcohol content. However, there is some scatter around the line, which could be explained by other factors affecting alcohol content.

5. Evaluation and Discussion

The linear model effectively captures the negative relationship between alcohol and density. However, it's important to note that while the linear regression provides a good overall fit, some variability remains unexplained by the model. There are several points scattered above and below the regression line, which may suggest that additional factors not captured in this simple model could influence the relationship between alcohol and density. Future models might consider including additional variables or exploring non-linear relationships to improve accuracy.

Future work could include examining other variables that may impact the alcohol content, such as sugar content or acidity, to build a more comprehensive model. Additionally, testing non-linear models could provide a better fit for capturing the more complex nature of the relationship between alcohol content and density.

Task 2: Classification

1. Introduction

In Task 2, we classified the wine quality using two classifiers: k-Nearest Neighbours (kNN) and Decision Tree, along with a modified version of the kNN classifier. The primary objective was to compare the performance of these models and identify a method to enhance the classification performance. A random sample of 500 instances was taken from the dataset, ensuring that the data had no missing values.

For the analysis, we trained the models using 80% of the sample data and evaluated the performance of the remaining 20%. This report compares the accuracy, precision, recall, F1-score, and confusion matrices for the three models, offering insights into the strengths and weaknesses of each approach.

2. Dataset Overview

The dataset consists of multiple features, including chemical properties like fixed acidity, volatile acidity, citric acid, etc., and the target variable, quality, which represents the wine's quality on a scale. After cleaning, we had a balanced subset of the dataset (500 instances) to train and evaluate our models. Besides, the cleaned dataset was split into training and testing subsets, with 80% allocated for training and 20% for testing. This split allowed for an effective evaluation of the classifier's performance on unseen data.

3. kNN

a. Model Overview

The standard kNN classifier was implemented using $k = 3$, which was chosen based on empirical testing and consideration of the dataset's characteristics. A smaller value of k tends to capture local structure, which is beneficial in many cases.

b. Evaluation Metrics

Accuracy: The accuracy of the standard kNN classifier is 34%, meaning that the model correctly predicted 34% of the instances in the test set. Given that the dataset is imbalanced, accuracy alone does not provide a comprehensive view of model performance. It is essential to delve deeper into class-wise performance using precision, recall, and F1-score.

Precision and Recall:

- Class 5 and class 6 have precision values of 0.26 and 0.46, respectively, indicating that many instances were incorrectly classified into these classes.
- Recall is 0.32 for class 5 and 0.47 for class 6, meaning that a significant portion of the true instances of these classes were correctly identified, but with many false positives.

F1-Score: The F1-score is highest for class 6 at 0.46, which reflects the model's better performance for this class compared to others. However, the relatively low F1-scores for other classes (e.g., 0.29 for class 5 and 0.20 for class 7) indicate that the model struggles to balance precision and recall for many classes.

c. Confusion Matrix Analysis

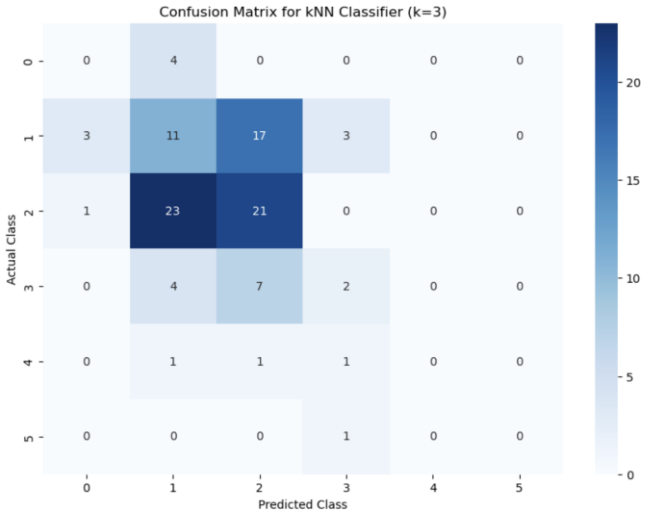

Figure 2: Confusion Matrix for KNN Classifier

The confusion matrix for the kNN classifier (k = 3) revealed the following:

- **Class 4:** The model misclassified all 4 instances as belonging to class 5, indicating that the kNN algorithm struggled to separate class 4 from class 5.
- **Class 5:** While 11 instances of class 5 were correctly classified, a significant number (17 instances) were misclassified as class 6, showing that the model found it difficult to differentiate between these two adjacent quality categories. This confusion can be attributed to the similarity in wine characteristics between the two quality classes.
- **Class 6:** Although 21 instances of class 6 were correctly predicted, 23 instances were incorrectly classified as class 5, reinforcing the point that there is a high overlap between these two classes. The lack of weighting in standard kNN means that the model gives equal importance to all neighbors, leading to these misclassifications.
- **Class 7:** The model correctly identified only 2 instances of class 7, with 7 instances being misclassified as class 6. This suggests that class 7, though distinct, shares characteristics with class 6, resulting in high misclassification rates.
- **Minority classes (3, 8, and 9):** For classes with very few instances (such as 3, 8, and 9), the model failed to make correct predictions. Class 3 had no correct predictions, and both classes 8 and 9 had 0 correct predictions, indicating that the classifier was biased towards the majority classes and was unable to handle minority classes effectively.

d. Summary of the Results

- The kNN model struggles with overlapping classes, particularly between class 5 and class 6, due to the equal weighting of neighbors and the lack of any mechanism to prioritize closer points.
- The confusion matrix shows a strong bias towards the majority classes (5 and 6), with minority classes receiving little attention. This is expected with kNN, as it is highly sensitive to the distribution of the training data.
- The performance metrics suggest that while the model can perform reasonably well for the majority classes, it fails to generalize for the minority classes, leading to poor performance across those categories.

e. Strengths and Weaknesses

Strengths:

- ***Simplicity:*** The standard kNN is a straightforward algorithm that is easy to understand and implement. It does not require extensive model training, as it simply relies on storing the training data and classifying new instances based on the majority class of the nearest neighbors.
- ***No Assumptions:*** kNN does not make any assumptions about the distribution of the data. This can be an advantage when working with non-linear and complex datasets where other models may fail to capture underlying patterns.
- ***Local Decision Making:*** Since kNN relies on local data points for decision-making, it can perform well on data with well-separated classes where the neighbors of a point are mostly of the same class.

Weaknesses:

- ***Sensitive to Class Imbalance:*** As seen in the analysis, the standard kNN struggles with imbalanced datasets, where the majority classes dominate the predictions, and minority classes are often misclassified. This is because the majority class will often have more neighbors simply due to its larger size, skewing the results.
- ***Overlapping Classes:*** The model has difficulty distinguishing between classes that overlap, such as class 5 and class 6 in this case. kNN does not give any weighting to closer neighbors, so points from different but nearby classes can confuse the model, leading to misclassifications.
- ***Computational Complexity:*** kNN is computationally expensive, especially when applied to large datasets. Since it relies on calculating the distance between a new data point and all training points during prediction, it can be slow for large datasets or when many neighbors are considered.
- ***Lack of Feature Weighting:*** Standard kNN gives equal importance to all features, which can be problematic in datasets where some features are more informative than others.

4. Modified kNN

a. Proposed Modification

The modified kNN classifier introduced distance-based weighting, where neighbors closer to a query point were given more influence in determining the class label. This approach was implemented to overcome some of the limitations of the standard kNN algorithm.

b. Evaluation Metrics

Accuracy: The modified kNN classifier achieved an accuracy of 39%, an improvement over the standard kNN. This suggests that using a distance-based weighting scheme helps the model make more informed classification decisions, particularly for classes with higher overlap.

Precision and Recall:

- The precision for class 5 improved to 0.33 (compared to 0.26 in standard kNN), and for class 6, it increased to 0.48. This indicates that the model was more precise in predicting these classes, though the overall performance still leaves room for improvement.
- Recall for class 5 improved slightly to 0.38, while for class 6 it increased to 0.53. The model was better able to identify true instances of these classes without significantly increasing false positives.

F1-Score: The F1-score for class 6 increased to 0.51, reflecting the improvement in both precision and recall. For class 5, the F1-score improved slightly to 0.35. This shows that the modified kNN is better at balancing precision and recall, leading to more robust performance in overlapping classes.

c. Confusion Matrix Analysis

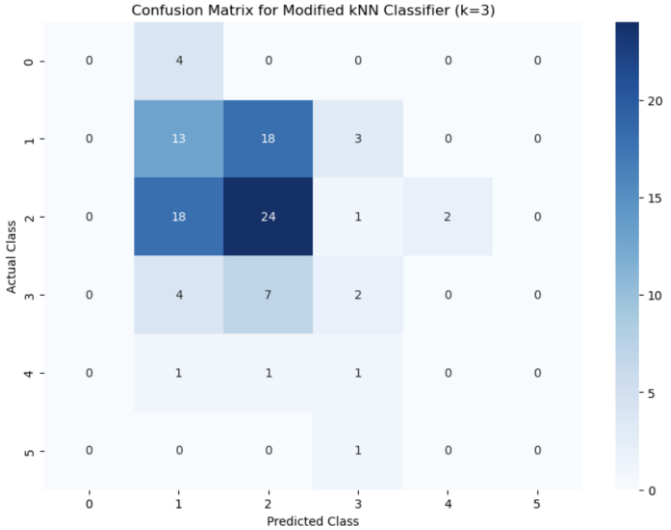

Figure 3: Confusion Matrix for Modified kNN Classifier

The confusion matrix for the modified kNN showed:

- **Class 4:** Similar to standard kNN, the model failed to correctly classify any instances of class 4, showing that distance-based weighting did not significantly improve performance for this class.
- **Class 5:** The model classified 13 instances correctly (compared to 11 in the standard kNN) and reduced the number of misclassifications into class 6 (18 instances instead of 17). The use of weighting helped improve the precision of class 5, albeit marginally.
- **Class 6:** The model classified 24 instances correctly, slightly improving over the standard kNN (21 correct predictions). There was also a decrease in the number of instances misclassified as class 5, thanks to the weighting scheme that prioritized closer neighbors.
- **Class 7:** The results for class 7 remained consistent with the standard kNN, with 2 correct classifications and 7 misclassified instances.
- **Minority classes (3, 8, and 9):** Similar to the standard kNN, the modified kNN struggled with these minority classes, as no significant improvement was observed in their predictions.

d. Summary of the Results

The modified kNN classifier performed better in distinguishing between classes 5 and 6, reducing the degree of confusion between these neighboring classes. This improvement is directly attributable to the distance-based weighting scheme, which gives higher importance to nearer neighbors.

Although the accuracy and F1-scores for the majority classes improved, the model still struggles to correctly classify minority classes (e.g., 3, 4, 8, and 9). This reflects a common limitation of kNN-based methods when dealing with highly imbalanced datasets.

The overall improvement in accuracy and metrics suggests that the modified kNN is more effective in situations where class overlap is a major concern, but further improvements would be needed to address the dataset's imbalance.

e. Strengths and Weaknesses

Strengths:

- ***Improved Handling of Overlapping Classes:*** The introduction of distance-based weighting in the modified kNN helps mitigate issues with overlapping classes. By giving more importance to closer neighbors, the model makes more informed decisions when dealing with nearby points from different classes. This is evident from the improved performance in distinguishing between class 5 and class 6.
- ***Better Precision and Recall:*** The modified kNN provides better precision and recall, particularly for the majority classes, as it reduces the effect of far-away neighbors that may belong to different classes. This leads to fewer false positives and slightly improved classification accuracy.
- ***Enhanced Flexibility:*** By introducing weights, the modified kNN offers more flexibility in determining the contribution of neighbors. This allows the model to be more adaptive to different datasets, improving performance on datasets with a high degree of overlap or noise.

Weaknesses:

- ***Still Sensitive to Class Imbalance:*** Although the modified kNN shows improved performance compared to the standard kNN, it still suffers from misclassification of minority classes. For example, classes 3, 4, 8, and 9 continue to be misclassified frequently, reflecting that the model does not address class imbalance effectively.
- ***Requires Hyperparameter Tuning:*** The effectiveness of the modified kNN depends on selecting appropriate values for the weighting function and the number of neighbors (k). Poor choices can lead to overfitting or underfitting. This makes the model more complex to tune compared to standard kNN.
- ***Increased Computational Cost:*** While the modified kNN provides improved results, it still suffers from the high computational cost associated with standard kNN. In fact, the use of weights adds another layer of complexity, as the model must calculate distances and apply weights to each neighbor during prediction.
- ***Difficulty with High-Dimensional Data:*** Like standard kNN, the modified version does not perform well with high-dimensional data. The curse of dimensionality can lead to poor distance calculations, making the model less effective in high-dimensional spaces where meaningful patterns are harder to capture.

f. Why the Modified kNN Outperforms the Original kNN

The modified kNN classifier can lead to better performance in this case due to its distance-based weighting, which makes the model more robust in several key areas where the standard kNN struggles.

- **Overlapping Classes**: The standard kNN treats all neighbors equally, leading to misclassification, especially between overlapping classes like 5 and 6. The modified kNN prioritizes closer neighbors, improving classification accuracy in these cases.
- **Handling Noise**: Distance weighting reduces the influence of noisy data points. Unlike the standard kNN, which can misclassify based on distant outliers, the modified kNN downplays these influences, enhancing prediction accuracy.

- **Imbalanced Data**: While not a complete solution to class imbalance, the modified kNN mitigates bias toward majority classes by emphasizing closer, more relevant neighbors. This results in better representation for minority classes and improved precision in the confusion matrix.
- **Smoother Decision Boundaries**: The standard kNN can create jagged decision boundaries, leading to overfitting. The modified kNN's distance weighting produces smoother boundaries, resulting in a more robust model that generalizes better to unseen data.
- **Precision and Recall**: The modified kNN improves precision and recall for majority classes, such as class 6, by focusing on nearby points likely to belong to the correct class. This results in lower false positives and improved metrics, enhancing overall classification effectiveness.

5. Decision Tree (and Comparison)

a. Model Overview

The Decision Tree classifier is a non-parametric model that splits the data based on the most informative features. We tuned the decision tree parameters, such as maximum depth and minimum samples per leaf, and found the optimal values using cross-validation.

b. Evaluation Metrics

The results indicate that the decision tree classifier correctly predicted the class labels for 46% of the test samples. The precision and recall are reasonably balanced at 0.47 and 0.46, respectively, showing that the model is moderately effective in distinguishing between the classes. The F1-score of 0.46 further confirms that the model's performance across precision and recall is consistent, though not particularly high.

c. Confusion Matrix Analysis

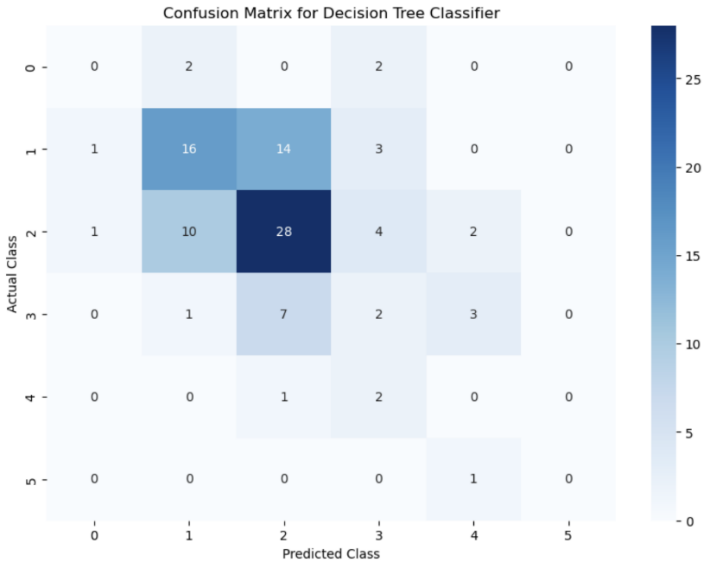

Figure 4: Confusion Matrix for Decision Tree Classifier

The confusion matrix for the Decision Tree showed:

- **Class 4:** None of the samples from class 4 were correctly predicted, indicating that the decision tree struggled significantly with this class. It misclassified half of them as class 5 and the other half as class 7.
- **Class 5:** Class 5 had a substantial number of correct predictions (16 out of 34). However, there were still a significant number of misclassifications, mainly into class 6, indicating some overlap between the features of classes 5 and 6.
- **Class 6:** This class had the highest number of correctly predicted instances (28 out of 45), showing that the model performed relatively well in class 6. However, there were still misclassifications into classes 5, 7, and 8.
- **Class 7:** The performance of class 7 was mediocre, with only 2 out of 13 instances correctly predicted. This class was confused with classes 6 and 8, suggesting that these classes might have similar features in the dataset.
- **Class 8:** The model failed to predict class 8 correctly in most cases, and many instances were classified as class 7.
- **Class 9:** With only one instance in the dataset, class 9 was correctly predicted once, but there is not enough data to draw significant conclusions about this class.

The decision tree model shows a clear tendency to misclassify minority classes. For example, class 4 was consistently misclassified, and classes 7 and 8 also showed poor performance. These errors suggest that the decision tree struggles to learn discriminative boundaries between underrepresented classes, which is a common issue with decision trees, especially when the dataset is imbalanced. Moreover, the confusion between classes 5 and 6 implies that the decision tree may not have learned clear separations between them, potentially due to feature overlap or insufficient data for those distinctions.

d. Summary of the Results

The decision tree classifier achieved an overall accuracy of 0.46, indicating that it correctly predicted the class labels for 46% of the test instances. While the classifier performed reasonably well for majority classes, such as class 6 (with a recall of 0.62 and precision of 0.56) and class 5 (with a recall of 0.47 and precision of 0.55), it struggled significantly with minority classes. For example, class 4 had no correctly classified instances, and class 8 showed similar poor performance with no accurate predictions.

The confusion matrix highlights that there were substantial misclassifications between classes with overlapping features. For instance, many instances from class 5 were misclassified as class 6, and vice versa. Classes with fewer instances, such as class 7 and class 8, were often confused with neighboring classes, indicating a potential difficulty in distinguishing between similar or underrepresented classes.

e. Strengths and Weaknesses

Strengths:

- ***Interpretability:*** One of the key advantages of decision trees is their transparency. The tree structure can be visualized, making it easy to understand how the model makes decisions. This interpretability is valuable in domains where explaining model decisions is important.
- ***Handling of Non-linear Relationships:*** Decision trees are flexible and capable of capturing non-linear relationships in the data, as evidenced by their ability to make complex decision boundaries.
- ***Versatility:*** Decision trees can handle both categorical and continuous features, making them applicable to a wide variety of datasets.

Weaknesses:

- ***Overfitting:*** Decision trees are prone to overfitting, especially when they grow too deep. The model tends to fit the noise in the training data rather than generalizing well to unseen data. Pruning the tree or setting constraints such as maximum depth or minimum samples per leaf can help mitigate this issue.
- ***Class Imbalance Sensitivity:*** As seen in this task, decision trees are sensitive to class imbalance. The model tends to favor majority classes and perform poorly on minority classes, which can be problematic in datasets with uneven class distributions. Techniques such as class weighting or resampling can be employed to address this issue.
- ***Instability:*** Decision trees can be unstable, meaning that small changes in the data can lead to a completely different tree structure. This instability can make predictions less reliable, particularly in noisy datasets.

f. Comparison

Model	Accuracy	Weighted Precision	Weighted Recall	Weighted F1-score
kNN	0.34	0.37	0.34	0.33
Decision Tree	0.46	0.47	0.46	0.46

- Accuracy: The decision tree classifier outperformed the kNN classifier in terms of accuracy, correctly predicting 46% of the instances compared to kNN's 34%. This indicates that the decision tree was more effective at finding the correct classification boundaries in this dataset.
- Precision: The decision tree also demonstrated higher precision compared to kNN. This suggests that the decision tree made fewer false positive predictions across all classes, being more selective when predicting positive cases.
- Recall: The recall for the decision tree was substantially higher than that of kNN, meaning the decision tree was better at identifying true positive cases. In particular, classes like 5 and 6 saw higher recall rates under the decision tree than under kNN, which had more difficulty with identifying the correct class for positive samples.
- F1-Score: The F1-score for the decision tree was also superior to kNN, indicating a better overall balance between precision and recall. This balance is important in cases where the dataset may be imbalanced, as it prevents the model from heavily favoring either precision or recall.

⇨ ***Confusion Matrix:***

- **Decision Tree:** The decision tree performed better in predicting majority classes like 5 and 6, with fewer misclassifications in these groups. However, it still struggled with minority classes such as 4, 7, and 8, showing a tendency to misclassify them into neighboring classes.
- **kNN:** The kNN classifier struggled more across the board, particularly with minority classes. It had greater difficulty in identifying the correct classes, leading to more widespread misclassification across the matrix. For instance, class 5 was often confused with other classes, and minority classes were rarely classified correctly.

⇨ ***Generalization to Minority Classes:***

- **Decision Tree:** The decision tree demonstrated some capability to handle minority classes but struggled with extreme cases like class 4 and class 8, where it failed to make correct predictions. It performed better than kNN but still exhibited significant bias toward the majority classes.

- **kNN:** kNN was more susceptible to misclassifying minority classes, particularly due to its reliance on proximity in feature space. It tended to favor the majority classes more heavily, with a substantial number of incorrect classifications for minority classes.

Task 3: Clustering

1. Introduction

In this task, two clustering algorithms, k-Means, and DBSCAN, were applied to a dataset to explore the structure of the data based on its input variables. The task excluded the target variable (quality) from the clustering process, though it could be used for evaluation purposes.

This report summarizes the performance of k-Means and DBSCAN, discusses the effects of varying their key parameters, and provides a comparison of the two methods. We also examine the limitations of each method and suggest possible improvements.

2. Data Preprocessing

Before clustering, a random sample of 300 instances from the dataset was selected, ensuring no missing values were present. Data standardization was performed using *StandardScaler* to normalize features, which is essential for distance-based clustering algorithms like k-Means and DBSCAN.

3. k-Means Clustering

k-Means is a partitioning method that divides the dataset into k clusters. It minimizes within-cluster variance (inertia) by iteratively assigning points to the nearest cluster center and recalculating the centroids. The choice of k is critical for the algorithm's success, and it directly impacts the quality of the clusters.

a. Results of k-Means

k	Inertia	Silhouette Score	Calinski-Harabasz Score
2	200242.86	0.5198	456.04
3	135116.60	0.3894	407.39
4	105106.90	0.3425	375.74
5	85474.99	0.3417	362.05
6	74039.78	0.3223	342.18
7	66251.69	0.3051	323.24
8	57297.59	0.3267	325.68
9	50758.00	0.3350	325.19
10	45705.27	0.3366	323.41

The k-Means algorithm was evaluated for values of k ranging from 2 to 10 using three key metrics:

- **Inertia**, which measures the sum of squared distances between points and their cluster centroids, decreased with increasing k, as expected. However, beyond $k = 4$, the reduction in inertia became less significant, indicating diminishing returns in terms of tighter clustering.
- The **Silhouette score**, which assesses how distinct the clusters are, peaked at $k = 2$ with a score of 0.5198, suggesting that the two clusters were the most well-separated and compact. As k increased, the silhouette score declined, indicating that clusters became more fragmented and less clearly defined.
- The **Calinski-Harabasz score**, which evaluates the ratio of between-cluster to within-cluster variance, also reached its highest value at $k = 2$, reinforcing that the two-cluster configuration offered the best separation.

The results show that 2 clusters (k = 2) give the best clustering performance based on the metrics used (such as silhouette score and Calinski-Harabasz score). These scores suggest that the data points are well grouped and separated when divided into two clusters. However, using only two clusters might be too simplified for this dataset. In reality, the data could be more complex, with more than two meaningful groups, which could lead to valuable insights being missed. Therefore, to ensure that the choice of two clusters is appropriate, it could help to:

- Consult experts or domain knowledge related to the data (like wine quality or characteristics) to see if it's reasonable to divide the data into two clusters.
- Use the quality variable (which wasn't used in clustering) to evaluate whether the clusters align well with the actual quality ratings of the items, to verify the practical usefulness of having just two clusters.

b. Limitation of k-Means

Despite the insights provided by the k-Means algorithm, several limitations were observed during this analysis:

- **Sensitivity to Initialization:** The k-Means algorithm is known to be sensitive to the initial placement of centroids, which can result in different clusters on different runs. This was mitigated in this case by setting *n_init=10*, which runs the algorithm 10 times with different centroid initializations and selects the best result. However, this does not fully eliminate the possibility of suboptimal solutions.
- **Assumption of Spherical Clusters:** k-Means assumes that clusters are spherical and have similar sizes. In datasets with complex structures or varying densities, this assumption may not hold, resulting in poorly defined clusters. This limitation was not directly addressed in the current analysis, and it's possible that some clusters could be non-spherical or of varying densities, leading to suboptimal performance.
- **Predefined Number of Clusters:** The need to specify the number of clusters (k) beforehand can be problematic, especially if the true number of clusters is unknown. While methods such as the elbow method and silhouette analysis can help estimate k, they are not always reliable, particularly in datasets with complex or overlapping clusters.

c. Possible Solutions

To address these limitations, several alternatives and improvements can be considered:

- **k-Means++ Initialization:** This improved version of k-Means enhances the initialization of centroids by spreading them out more strategically, reducing the chances of suboptimal solutions and improving clustering performance.
- **Hierarchical Clustering:** This technique could be employed to generate a dendrogram, which visually represents how data points group together at various levels. The dendrogram can then help estimate the number of clusters before applying k-Means.
- **Gaussian Mixture Models (GMMs):** Unlike k-Means, which assumes spherical clusters, GMMs allow for elliptical clusters and better handle clusters of varying sizes and densities, making them more flexible in identifying complex structures in data.

4. DBSCAN Clustering

In applying the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, two key parameters were used: **epsilon (ε)**, which defines the radius around a point for it to be considered part of a cluster, and **min_samples**, the minimum number of points required to form a dense region. DBSCAN was evaluated using these parameters and its performance was assessed in terms of the **number of clusters**, **noise points**, and the **silhouette score**.

a. Results of DBSCAN

The performance of DBSCAN across various combinations of epsilon (ε) and minimum samples (min_samples) reveals important insights into the clustering structure of the dataset.

- **Low Silhouette Scores**: The highest silhouette score of **0.0547** is observed for $\epsilon = 1.0$ and *min_samples = 1*, which is a relatively poor result. This score indicates that the clusters formed are weakly defined and lack clear separation. In most configurations, the silhouette score is either negative or very close to zero, suggesting that the clusters overlap, and the data points within the clusters are not tightly grouped.
- **Effect of Epsilon (ε)**: As ε increases from **0.1** to **1.0**, the silhouette score slightly improves for **min_samples = 1**, but only marginally. The small improvement in the silhouette score, from **0.027** at $\epsilon = 0.1$ to **0.0547** at $\epsilon = 1.0$, suggests that increasing ε helps DBSCAN identify more spread-out clusters, but these clusters are still not well-separated. For higher values of **min_samples**, increasing ε results in either all points being classified as noise or the formation of a single, undifferentiated cluster.
- **Effect of min_samples**: A higher **min_samples** value generally leads to worse results. For example, at $\epsilon = 0.1$ and *min_samples = 2*, the silhouette score drops to **-0.2067**, indicating poor clustering and potential misclassification of points. As **min_samples** increases, DBSCAN frequently collapses all points into a single cluster or labels them as noise, particularly at smaller ε values. This reflects a difficulty in finding meaningful clusters in the data.
- **Single Cluster or All Noise**: For **min_samples** values of 3 or more, and a wide range of ε values (from 0.1 to 0.6), the algorithm often returns a result where all points either belong to a single large cluster or are labeled as noise. This indicates that the density-based nature of DBSCAN struggles with this dataset, which may not have a clear cluster structure based on point density.

b. Limitations of DBSCAN

Parameter Sensitivity:

- DBSCAN relies heavily on choosing appropriate values for ε (epsilon) and min_samples. If these parameters are not well-tuned, DBSCAN may fail to identify meaningful clusters or label most points as noise. This is evident in cases where the algorithm collapses into a single cluster or identifies no clusters at all.
- Selecting a too-small ε can lead to fragmented clusters or an overabundance of noise points, while a too-large ε may merge distinct clusters into one.

Struggles with Clusters of Varying Densities: DBSCAN is designed to detect clusters based on density. However, if the dataset contains clusters of different densities, DBSCAN may underperform. It tends to either split denser clusters or merge sparser clusters into larger ones.

Curse of Dimensionality: In high-dimensional data, distance-based methods like DBSCAN suffer because the concept of density loses meaning. With increasing dimensions, the distances between points become more similar, making it difficult for DBSCAN to detect dense regions or form clusters.

Limited Scalability: DBSCAN has a time complexity of O(n^2), which makes it inefficient for large datasets. While improvements like indexing structures (e.g., k-d trees) can help, DBSCAN still struggles to handle very large datasets in comparison to other algorithms such as k-Means.

c. Possible Solutions

- **Parameter Tuning:** Using methods like the k-distance plot helped select an eps value, but more sophisticated techniques like grid search or cross-validation might be necessary.
- **Alternative Clustering Algorithms:** For datasets with non-globular or overlapping clusters, algorithms like HDBSCAN (an extension of DBSCAN) or Gaussian Mixture Models (GMMs) might perform better.

5. Comparison

Criteria	k-Means	DBSCAN
Cluster Shape	Assumes spherical clusters of equal size.	Can detect clusters of arbitrary shapes.
Number of Clusters	Must be pre-specified.	Automatically determines based on data.
Handling Outliers	Sensitive to outliers, which can skew results.	Robust to outliers; labels them as noise.
Parameter Sensitivity	Sensitive to the initial centroids and k.	Sensitive to ε (epsilon) and min_samples.
Performance in High Dimensions	Performs better than DBSCAN but can still suffer from the curse of dimensionality.	Performs poorly in high-dimensional spaces.
Scalability	More scalable with time complexity O(n.k.d)	Less scalable with time complexity O(n^2); slower for large datasets.
Cluster Density	Assumes clusters are of equal density.	Works well with clusters of varying densities.
Data Assumptions	Works best with globular, evenly spaced clusters.	No assumptions on cluster shapes or density.
Use Cases	Suitable for large datasets with predefined cluster numbers.	Suitable for noisy datasets and discovering clusters with irregular shapes.
Initialization	Relies on random initialization of centroids.	No initialization is required.
Interpretability	Results are easy to interpret with predefined k.	More challenging to interpret, especially for higher noise levels.

In summary, both k-Means and DBSCAN are widely used clustering algorithms, each with unique strengths and limitations that cater to different data characteristics.

k-Means generally performs well when clusters are spherical and well-separated. It tends to yield higher silhouette scores, indicating distinct and compact clusters, especially at k=2 in the previous analysis. However, its sensitivity to outliers and reliance on a predetermined number of clusters can hinder its effectiveness in more complex datasets.

Conversely, DBSCAN is adept at identifying clusters of arbitrary shapes and is resilient to noise. While it produced a maximum silhouette score of 0.0547, indicating minimal separation among clusters in the analyzed datasets, this is relatively low compared to the scores achieved by k-Means. The performance of DBSCAN can be significantly affected by the choice of parameters ε (epsilon) and min_samples, which can complicate its application and may lead to a lack of distinct clusters, as evidenced by many negative silhouette scores in the results.

References

Stats Made Easy. (2022). *Tutorial 8: Regression Modelling I*. https://www.statsmadeasy.com/stats-concepts/8-regression-modelling-i