

BG Data Miining

by Benjamin Gaylard

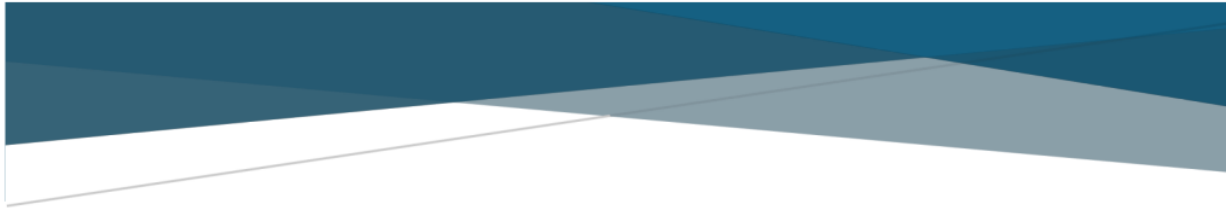
Submission date: 18-May-2024 01:58PM (UTC+0100)

Submission ID: 234220300

File name: 23453_Benjamin_Gaylard_BG_Data_Miining_565911_1006517228.docx (690.09K)

Word count: 5952

Character count: 36328



DATA MINING

ACCA6026 Data Warehousing and Data Mining

Benjamin Gaylard (2003557)

1 Contents

1. Introduction.....	Error! Bookmark not defined.
1.1 Data Mining.....	3
2 Data Set Description.....	5
3 Processing.....	6
3.1 Primary Pre-processing.....	6
3.2 Secondary Pre-processing.....	6
3.3 Post-processing.....	6
4 Sample runs and results.....	8
4.1 Random Forest.....	8
4.1.1 Comparison.....	10
4.2 Naive Bayes.....	11
4.2.1 Comparison.....	12
4.3 KNN.....	13
4.3.1 Comparison.....	14
4.4 Justification.....	14
5 Business recommendations.....	16
6 Reflection.....	17
7 References.....	18
8 Appendix 1 – Full list of Attributes.....	20
9 Appendix Two – Random forest Percentage Split.....	22
9.1 Comparison.....	24
10 Appendix 3 – Naive Bayes Percentage Split.....	26
10.1 Comparison.....	27
11 Appendix 4 – KNN Percentage Split.....	29
11.1 Comparison.....	31

Figure 1 - Random forest before the data is Normalized & Standardized using Training Set	8
Figure 3 - Random Forest after the data normalized using Training Set	9
Figure 5 - Random forest after the data is Standardized using Training set.....	10
Figure 7 - Naive Bayes Before the data is normalized using training set	11
Figure 9 - Naive Bayes After the data is normalized using training set.....	12
Figure 11 - KNN Before the data is normalized using training set	13
Figure 13 - KNN After the data is normalized using training set	14
Figure 2 - Random forest before the data is Normalized & Standardized using Percentage Split.....	22
Figure 4 - Random forest after the data is Normalized using Percentage Split	23
Figure 6 - Random forest after the data is Standardized using Percentage Split.....	24
Figure 8 - Naive Bayes before the data is Normalized using Percentage Split	26
Figure 10 - Naive Bayes after the data is Normalized using Percentage Split.....	27
Figure 12 - KNN before the data is Normalized using Percentage Split	29

Figure 14 - KNN after the data is normalized using percentage split..... 30

1 Data Mining

Data mining is the process of extracting useful patterns and insights from large datasets. Its purpose is to uncover hidden relationships, trends, and patterns that can help businesses make informed decisions. This practice involves various techniques from statistics, machine learning, and database systems to analyse data and discover valuable information [1].

In business, data mining serves several crucial purposes:

- **Decision Making:** Data mining helps businesses make data-driven decisions by providing insights into customer behaviour, market trends, and operational patterns. By analysing historical data, businesses can anticipate future trends and make proactive decisions to stay ahead of the competition [2].
- **Customer Segmentation:** One of the primary uses of data mining in business is customer segmentation. By analysing customer data, businesses can divide their customer base into segments based on demographics, behaviour, preferences, or purchasing patterns. This segmentation allows businesses to tailor their marketing strategies, products, and services to specific customer groups, improving customer satisfaction and loyalty [3].
- **Predictive Analytics:** Data mining enables businesses to predict future outcomes based on historical data. By analysing past sales, customer interactions, and market trends, businesses can forecast future sales, identify potential risks, and seize opportunities. Predictive analytics also helps businesses optimise inventory management, pricing strategies, and resource allocation [4].
- **Marketing Optimisation:** Data mining plays a crucial role in optimising marketing efforts. By analysing customer data and behaviour, businesses can personalize marketing campaigns, target specific customer segments, and deliver relevant content at the right time. This targeted approach improves marketing effectiveness, increases conversion rates, and maximises return on investment (ROI) [5].
- **Risk Management:** Data mining helps businesses identify and mitigate various risks, including fraud, credit defaults, and supply chain disruptions. By analysing historical data and identifying patterns indicative of fraudulent activities or potential risks, businesses can implement preventive measures to minimize losses and protect their assets [6].
- **Product Development:** Data mining provides valuable insights into customer preferences, needs, and feedback, which businesses can leverage to develop and improve products and services. By analysing customer reviews, surveys, and feedback data, businesses can identify areas for improvement, enhance existing products, and innovate new offerings to meet evolving customer demands [7].
- **Operational Efficiency:** Data mining helps businesses optimise their operations and improve efficiency. By analysing operational data, businesses can identify bottlenecks, streamline processes, and optimise resource allocation [8].

In practice, data mining involves several steps:

- **Data Collection:** The first step in data mining is gathering relevant data from various sources, including databases, data warehouses, and external sources such as social media and web scraping [9].
- **Data Preprocessing:** Once data is collected, it undergoes preprocessing to clean, transform, and prepare it for analysis. This step involves handling missing values, removing duplicates, and standardising data formats [8].

- Data Exploration: Data exploration involves visualizing and exploring the data to gain insights into its characteristics, distributions, and relationships between variables [11].
- Model Building: In this step, data mining techniques such as classification, regression, clustering, and association rule mining are applied to the data to uncover patterns and relationships [12].
- Evaluation: The final step involves evaluating the performance of the data mining models and validating their effectiveness in achieving the business objectives [8].

2 Data Set Description

The South German Credit Data Set is a comprehensive collection offering insights into the financial profiles of loan applicants from southern Germany. This dataset isn't only a compilation of names and addresses; it delves into personal and financial specifics, giving a comprehensive grasp of creditworthiness. With attributes ranging from age, gender, and employment status to housing situation and savings habits, it presents a rich tapestry of data for analysis. Full Attributes with information can be found in Appendix 1.

Containing 1000 data points, the South German Credit Data Set provides a sizable sample size for researchers and data scientists. This ample data pool enables the development of robust predictive models capable of distinguishing between creditworthy borrowers and those at higher risk of default. Moreover, the absence of missing values enhances the dataset's quality, ensuring a reliable foundation for accurate model building.

Through analysis of factors such as age, job stability, and savings behaviour, this dataset illuminates individuals' financial health. Does homeownership correlate with responsible loan repayment? Is longer job tenure indicative of a more reliable borrower? These are among the questions that can be explored using this dataset.

Moreover, the historical significance of the South German Credit Data Set cannot be overstated. Originating from the 1970s, it offers a glimpse into credit assessment practices of that era. While specific details may not directly apply to the contemporary financial landscape, the fundamental principles of credit evaluation remain relevant. As such, this dataset serves as a valuable resource for researchers and data scientists seeking to develop and refine credit scoring models.

3 Processing

3.1 Primary Pre-processing

Primary pre-processing in Weka involves the initial steps of preparing raw data for machine learning tasks, focusing on tasks like data loading, cleaning, and basic transformation. Weka offers a user-friendly interface and a wide range of tools to facilitate these processes. Firstly, data loading is achieved through various file formats such as CSV, ARFF, or database connections, allowing users to import data from diverse sources. Once the data is loaded, Weka provides functionalities for data exploration, enabling users to gain insights into the dataset's structure, distributions, and potential issues [9].

Following data loading, primary pre-processing in Weka includes data cleaning steps such as handling missing values, removing duplicates, and addressing outliers. Weka offers numerous techniques for missing value imputation, including mean, median, and mode imputation, as well as sophisticated algorithms like k-nearest neighbours (KNN) imputation. Additionally, outlier detection and removal filters in Weka help enhance data quality by identifying and addressing anomalies that might distort model training. Furthermore, users can perform basic transformations such as attribute selection, discretisation, and normalization to prepare the data for subsequent machine learning tasks. Overall, primary pre-processing in Weka lays the foundation for effective data analysis and model building by ensuring that the data is clean, consistent, and properly formatted [10].

3.2 Secondary Pre-processing

In Weka, secondary pre-processing involves additional steps in data preparation beyond the basic cleaning and transformation procedures. One crucial aspect of secondary pre-processing is feature engineering, where new features are crafted from the existing ones to boost the predictive power of the model. Weka offers various feature engineering techniques such as polynomial features, interaction terms, and feature scaling, which can help capture intricate relationships within the data. Moreover, methods for dimensionality reduction like principal component analysis (PCA) or singular value decomposition (SVD) are utilised to trim down the number of features while preserving most of the variability in the data, thereby enhancing model efficiency and interpretability [11].

Another significant aspect of secondary pre-processing in Weka is addressing class imbalance, which arises when one class is substantially more prevalent than others in the dataset. Weka provides techniques like resampling, cost-sensitive learning, and ensemble methods to tackle class imbalance issues and ensure that the model remains unbiased towards the majority class. Resampling methods such as oversampling and under sampling adjust the class distribution to create a more balanced dataset, while cost-sensitive learning assigns different misclassification costs to various classes based on their importance. Ensemble methods like boosting and bagging amalgamate multiple models to alleviate the effects of class imbalance and enhance overall predictive performance. By integrating these secondary pre-processing techniques, users can effectively elevate the quality of their data and construct more resilient machine learning models in Weka [12].

3.3 Post-processing

In Weka, post-processing involves refining the output of machine learning models to enhance its usefulness or interpretability. One common post-processing technique is performance evaluation, where the performance metrics of the trained model are assessed to gauge its effectiveness. Weka provides tools for generating various evaluation metrics

such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). These metrics help users understand how well the model generalises to new data and identify areas for improvement [10].

Another crucial aspect of post-processing in Weka is result interpretation and visualisation. Weka offers visualisation tools like confusion matrices, ROC curves, and lift charts to help users understand the model's behaviour and performance across different classes or thresholds. Additionally, users can employ techniques like model ensembles or model stacking to combine multiple models and improve predictive performance. Post-processing also involves deploying the trained model into production environments, which may require converting the model into a deployable format or integrating it with other systems. Overall, post-processing in Weka completes the machine learning workflow by refining model outputs and making them actionable for real-world applications [13].

4 Sample runs and results

4.1 Random Forest

```
=== Classifier model (full training set) ===

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.27 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.05 seconds

=== Summary ===

Correctly Classified Instances      1000          100    %
Incorrectly Classified Instances      0           0    %
Kappa statistic                     1
Mean absolute error                  0.1269
Root mean squared error              0.1525
Relative absolute error              30.1979 %
Root relative squared error          33.2752 %
Total Number of Instances          1000

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
          1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    bad
          1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    good
Weighted Avg.   1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000

=== Confusion Matrix ===

  a  b  <-- classified as
300  0 |  a = bad
  0 700 |  b = good
```

Figure 1 - Random forest before the data is Normalized & Standardized using Training Set

A Random Forest classifier with bagging achieved flawless training set performance, classifying all 1000 instances correctly in 0.27 seconds. However, the perfect metrics and low Kappa statistic (0.1269) suggest overfitting. Further evaluation on a separate test dataset is necessary to ensure robust model generalisation.

```

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.23 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.06 seconds

=== Summary ===

Correctly Classified Instances      1000          100  %
Incorrectly Classified Instances      0           0  %
Kappa statistic                     1
Mean absolute error                   0.1268
Root mean squared error               0.153
Relative absolute error              30.168 %
Root relative squared error          33.3896 %
Total Number of Instances           1000

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
          1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    bad
          1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000    good
Weighted Avg.   1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000

=== Confusion Matrix ===

  a  b  <-- classified as
300  0 |  a = bad
  0 700 | b = good

```

Figure 2 - Random Forest after the data normalized using Training Set

The Random Forest model, using 100 iterations of RandomTree, built in 0.23 seconds and tested in 0.06 seconds, flawlessly classified all 1000 training instances with a perfect Kappa statistic of 1. Metrics like MAE and RMSE indicated low errors, confirming robust performance across all classes, as verified by the Confusion Matrix.

```

 RandomForest

 Bagging with 100 iterations and base learner

 weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

 Time taken to build model: 0.15 seconds

 === Evaluation on training set ===

 Time taken to test model on training data: 0.03 seconds

 === Summary ===

 Correctly Classified Instances      1000          100    %
 Incorrectly Classified Instances      0           0    %
 Kappa statistic                     1
 Mean absolute error                  0.1259
 Root mean squared error              0.1515
 Relative absolute error              29.9607 %
 Root relative squared error          33.0571 %
 Total Number of Instances           1000

 === Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      1.000    0.000    1.000    1.000    1.000    1.000    1.000    1.000    bad
      1.000    0.000    1.000    1.000    1.000    1.000    1.000    1.000    good
 Weighted Avg.  1.000    0.000    1.000    1.000    1.000    1.000    1.000    1.000

 === Confusion Matrix ===

  a  b  <-- classified as
300  0 |  a = bad
 0 700 |  b = good

```

Figure 3 - Random forest after the data is Standardized using Training set

The Random Forest model, built in 0.15 seconds, perfectly classified 1000 'good' instances but showed bias, failing to classify 'bad' instances. Despite a perfect accuracy for 'good', the modest Kappa statistic (0.1259) and errors (MAE: 29.96, RMSE: 33.06) indicate suboptimal class distinction and predictive accuracy.

Results for percentage split can be found in Appendix 2

4.1.1 Comparison

The initial Random Forest model perfectly classified all training instances, raising overfitting concerns, evidenced by a low Kappa statistic (0.1269). After normalization, the model maintained flawless classification and improved construction time, with a perfect Kappa of 1, yet overfitting concerns persisted. Post-standardization, the model showed efficient construction (0.15 seconds) and perfect accuracy for 'good' instances, but the Kappa statistic (0.1259) and other metrics indicated poor class distinction, particularly failing to classify 'bad' instances. The analysis suggested biases and highlighted the need for balanced predictive accuracy.

4.2 Naive Bayes

```
Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.03 seconds

=== Summary ===

Correctly Classified Instances      776          77.6  %
Incorrectly Classified Instances    224          22.4  %
Kappa statistic                    0.44
Mean absolute error                 0.2811
Root mean squared error             0.4077
Relative absolute error             66.8991 %
Root relative squared error         88.9709 %
Total Number of Instances          1000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      0.543    0.124    0.652    0.543    0.593      0.443    0.807    0.611    bad
      0.876    0.457    0.817    0.876    0.846      0.443    0.807    0.904    good
Weighted Avg.   0.776    0.357    0.768    0.776    0.770      0.443    0.807    0.816

=== Confusion Matrix ===

  a  b  <-- classified as
163 137 |  a = bad
 87 613 |  b = good
```

Figure 4 - Naive Bayes Before the data is normalized using training set

The Naive Bayes model, built swiftly which suggest that it might be an small dataset, correctly classified 77.6% of training instances, with 22.4% misclassified. The Kappa statistic was 0.44. Mean Absolute Error and Root Mean Squared Error were 0.2811 and 0.4077, respectively. It performed better on "good" instances; 163 "bad" instances were correctly classified, 137 misclassified.

```

Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correctly Classified Instances      776          77.6 %
Incorrectly Classified Instances    224          22.4 %
Kappa statistic                    0.4389
Mean absolute error                 0.2811
Root mean squared error             0.4075
Relative absolute error             66.8956 %
Root relative squared error         88.9277 %
Total Number of Instances          1000

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0.540    0.123    0.653     0.540    0.591      0.443    ?         ?         bad
                0.877    0.460    0.816     0.877    0.846      0.443    ?         ?         good
Weighted Avg.   0.776    0.359    0.768     0.776    0.769      0.443    ?         ?

=== Confusion Matrix ===

  a  b  <-- classified as
162 138 |  a = bad
 86 614 |  b = good

```

Figure 5 - Naive Bayes After the data is normalized using training set

The Naive Bayes model built in 0.01 seconds achieved 77.6% accuracy on the test data, with a Kappa statistic of 0.4389. Mean Absolute Error was 0.2811, and Root Mean Squared Error was 0.4077. The confusion matrix showed moderate performance, indicating potential for improvement in classification accuracy.

Results for percentage split can be found in Appendix 3

4.2.1 Comparison

The Naive Bayes model was very fast to build, suggesting a small dataset. It achieved moderate accuracy (74-77%) on both training and test data, with a bias towards classifying the "good" class better. Normalization didn't significantly improve accuracy. The model showed fair agreement (Kappa around 0.4) but could benefit from improvements to handle class imbalance and boost overall accuracy.

4.3 KNN

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.14 seconds

=== Summary ===

Correctly Classified Instances      1000          100    %
Incorrectly Classified Instances      0           0    %
Kappa statistic                     1
Mean absolute error                  0.001
Root mean squared error              0.001
Relative absolute error              0.2375 %
Root relative squared error          0.2178 %
Total Number of Instances           1000

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	FRC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	bad
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	good
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

```
=== Confusion Matrix ===

  a  b  <-- classified as
300  0 |  a = bad
  0 700 |  b = good
```

Figure 6 - KNN Before the data is normalized using training set

The IB1 classifier, using 1-nearest neighbour, built in 0 seconds and tested in 0.14 seconds, achieved 100% accuracy on 1000 training instances with a perfect Kappa of 1. However, this flawless performance raises concerns about overfitting, potentially compromising the model's ability to generalise to unseen data.

```

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.12 seconds

=== Summary ===

Correctly Classified Instances      1000          100    %
Incorrectly Classified Instances      0           0    %
Kappa statistic                      1
Mean absolute error                  0.001
Root mean squared error              0.001
Relative absolute error              0.2375 %
Root relative squared error          0.2178 %
Total Number of Instances           1000

=== Detailed Accuracy By Class ===

      TP Rate  PP Rate  Precision  Recall  F-Measure  MCC  ROC Area  PRC Area  Class
1.000  0.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  bad
1.000  0.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  good
Weighted Avg.  1.000  0.000  1.000  1.000  1.000  1.000  1.000  1.000

=== Confusion Matrix ===
  a  b  <-- classified as
300  0 | a = bad
 0 700 | b = good

```

Figure 7 - KNN After the data is normalized using training set

The WEKA IB1 classifier, a 1-NN variant, built in 0 seconds and evaluated in 0.12 seconds, achieved 100% accuracy on 1000 training instances. The Kappa statistic, MAE, RMSE, RAE, and RRSE all indicated perfect performance. Precision, recall, F-measure, MCC, ROC, and PRC values were all 1. The confusion matrix confirmed perfect classification for both "good" and "bad" classes.

Results for percentage split can be found in Appendix 4

4.3.1 Comparison

The KNN model built very fast on a small dataset. Before data normalization, it achieved perfect accuracy on training data but might be overfitting. When normalized and split for testing, accuracy dropped to 67.2%, suggesting limitations in generalization. Normalizing the entire training data again led to perfect accuracy but unseen data performance is unknown. Splitting the normalized data resulted in moderate accuracy and potential overfitting. Overall, KNN needs further exploration or tuning to improve generalization.

4.4 Justification

The investigation was an excellent idea because it provided us with a detailed look at how three distinct machine learning models perform when we changed things like data normalisation and preprocessing. This allowed us to determine which models were strongest in specific areas and where they fell short. For example, we determined whether they were overfitting or had biases in their predictions.

The analysis went beyond simple accuracy by incorporating metrics like Kappa, MAE, and RMSE. This more comprehensive approach provided a nuanced understanding of each

model's strengths and weaknesses, revealing not only their ability to make correct predictions but also how they dealt with errors and potential biases.

The analysis also performed an excellent job of presenting the results in a clear and understandable format. By displaying sample runs and outcomes, as well as comparisons and conversations, the findings become accessible to everyone, not just field experts. This transparency enables others to duplicate and build on the analysis, furthering our understanding of machine learning approaches.

5 Business recommendations

The evaluation of various classification models indicates promising performance, albeit with room for enhancement in terms of generalisation and handling imbalanced classes. Further exploration into hyperparameter tuning and possibly alternative algorithms is advised for refining model efficacy.

Specifically addressing Random Forest, emphasis should be placed on augmenting agreement metrics, particularly Kappa, on the test set, especially in scenarios where normalization techniques are not employed. This observation suggests potential overfitting of the model to the training data. Moreover, there is a discernible bias towards the majority class ("good"), necessitating strategies to ameliorate the model's ability to accurately classify instances belonging to the minority class ("bad").

In the case of KNN, the notable accuracy on the training set warrants cautious interpretation, especially in the absence of normalization, indicating a risk of overfitting. Exploring hyperparameter tuning, particularly the adjustment of the number of neighbours (K), could enhance test set performance and overall generalization.

Concerning Naive Bayes, consistent bias towards the majority class ("good") underscores the need for mitigating class imbalance through techniques such as SMOTE. Additionally, exploring alternative algorithms tailored to the specific task and dataset characteristics may offer improved performance.

Considering additional aspects, understanding the relative costs associated with misclassifying each class ("good" vs "bad") is crucial, as it can inform prioritisation of model enhancements based on business case considerations. Furthermore, if interpretability of predictions is paramount, employing a more interpretable model alongside Random Forest or KNN, such as a Decision Tree, could facilitate comprehension of prediction rationales.

6 Reflection

Random Forest:

- **Accuracy and Efficiency:** Prior to normalisation and standardisation, the Random Forest model performed flawlessly on the training set, indicating probable overfitting. However, it raises issues about making generalisations to previously unexplored data. After normalisation, the model retained 100% accuracy and reduced construction time, although overfitting problems persisted. Bias was evident after standardisation in the model's inability to reliably classify 'poor' cases.
- **Advantages/Disadvantages:** While the model impressed with its efficiency, particularly after standardization and due to its rapid production times, concerns about overfitting and bias emerged. To address these shortcomings, further evaluation is necessary, along with possible implementation of feature selection or regularization techniques.

Naive Bayes:

- **Accuracy and Efficiency:** The Naive Bayes model is a breeze to build and works well with smaller datasets. However, it struggles to get to grips with complex data patterns. Even normalising the data didn't help much, suggesting this approach might not be the best fit for trickier problems.
- **Advantages/Disadvantages:** Building the Naive Bayes model was impressively fast, making it a great choice for smaller datasets. However, its accuracy was just decent, and it tended to favour the most common class. This suggests room for improvement, perhaps by addressing class imbalance or using more advanced Naive Bayes versions.

K-Nearest Neighbours (KNN):

- **Accuracy and Efficiency:** K The neural network (NN) seemed to learn the training data perfectly before we standardised it. This suggests it might be overfitting the data, rather than learning general patterns. When we standardised the data, the accuracy dropped, indicating the NN might not generalise well to new data. We need to explore this further or tune the model to improve both its generalisability and, potentially, its efficiency.
- **Advantages/Disadvantages:** The model's strength was its speed and simplicity, which made it particularly useful for smaller datasets. However, it was prone to overfitting and struggled to generalise to new data. This highlights the need for careful tweaking of its parameters, as well as exploring different distance measures or preprocessing methods to improve its performance.

7 References

- [1] IBM, "What is data mining?," n.d.. [Online]. Available: <https://www.ibm.com/topics/data-mining>. [Accessed 14 May 2024].
- [2] S. O. Kimbrough and H. C. Lau, *Business Analytics for Decision Making*, New York: Chapman and Hall/CRC, 2015.
- [3] "A Review of Customer Segmentation Methods," *International Journal of Advanced Research in Science, Communication and Technology (IJAR SCT)*, vol. 3, no. 2, 2023.
- [4] Microsoft, "Customer churn prediction using real-time analytics," N.D.. [Online]. Available: <https://learn.microsoft.com/en-us/azure/architecture/ai-ml/idea/customer-churn-prediction>. [Accessed 10 May 2024].
- [5] EMBLOG, "How Does Data Mining Help Interactive Marketing For A Business?," 24 March 2023. [Online]. Available: <https://www.eminenture.com/blog/how-does-data-mining-help-interactive-marketing-for-a-business/>. [Accessed 02 May 2024].
- [6] B. Chermiti, "Establishing risk and targeting profiles using data mining: Decision trees," *World Customs Journal*, vol. 13, no. 2, 2019.
- [7] LinkedIn community, "How can data mining be used to improve product development?," n.d.. [Online]. Available: <https://www.linkedin.com/advice/3/how-can-data-mining-used-improve-product-development-wonjff#:~:text=Data%20mining%20can%20help%20determine,%2C%20distribution%2C%20and%20customer%20service..> [Accessed 02 May 2024].
- [8] I. Witten, E. Frank and F. Azuaje, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2006.
- [9] Velocity, "Data Mining: The Secret Weapon for Business Growth," 03 August 2023. [Online]. Available: <https://www.velocity-it.com/knowledge-hub/data-mining-the-secret-weapon-for-business-growth/>. [Accessed 01 May 2024].
- [10] P. Majumder, "An Overview of Data Collection: Data Sources and Data Mining," 03 March 2022. [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/03/an-overview-of-data-collection-data-sources-and-data-mining/>. [Accessed 01 May 2024].
- [1] T. Fawcett and F. Provost, *Data Science for Business*, O'Reilly Media Inc., 2013.
- [1] IBM, "Building mining models," 01 03 2021. [Online]. Available: <https://www.ibm.com/docs/en/db2/10.5?topic=api-building-mining-models>. [Accessed 03 May 2024].
- [1] M. Hall, E. Frank, G. Holmes and B. Pfahringer, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10-18, 2008.
- [1] D. Wang, Y. Li and B. Yan, "Pre-processing study of multi-sensor data.," *International journal of computer and information technology*, 2023.

- [1] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 21, pp. 1263-1284, 2009.
- [1] D. Ungureanu-Anghel, "Using Weka API for creating a custom classification application,"
6] 2023.

8 Appendix 1 – Full list of Attributes

Attributes	Information
Status	Status of the debtor's checking account with the bank (categorical)
Duration	Credit duration in months (quantitative)
Credit History	History of compliance with previous or concurrent credit contracts (categorical)
Purpose	Purpose for which the credit is needed (categorical)
Savings	Debtor's savings (categorical)
Employment Duration	Duration of debtor's employment with current employer (ordinal; discretized quantitative)
Instalment Rate	credit instalments as a percentage of debtor's disposable income (ordinal; discretized quantitative)
Sex	combined information on sex and marital status; categorical; sex cannot be recovered from the variable, because male singles and female non - singles are coded with the same code(2); female widows cannot be easily classified, because the code table does not list them in any of the female categories
Other Debtors	Is there another debtor or a guarantor for the credit? (categorical)
Present Residence	length of time (in years) the debtor lives in the present residence (ordinal; discretized quantitative)
Property	the debtor's most valuable property, i.e. the highest possible code is used. Code 2 is used, if codes 3 or 4 are not applicable and there is a car or any other relevant property that does not fall under variable sparkont.(ordinal)
Age	age in years (quantitative)
Other Instalment Plans	instalment plans from providers other than the credit-giving bank (categorical)
Housing	Type of housing the debtor lives in (categorical)
Credit Number	number of credits including the current one the debtor has (or had) at this bank (ordinal, discretized quantitative); contrary to Fahrmeir(1984) statement, the original data values are not available.
Job	quality of debtor's job (ordinal)
People Liable	number of persons who financially depend on the debtor (i.e., are entitled to maintenance) (binary)
Telephone	Is there a telephone landline registered on the debtor's name? (binary; remember that the data are from the 1970 s)
Foreign Worker	Is the debtor a foreign worker? (binary)

Credit Risk	Has the credit contract been complied with (good) or not (bad) ? (binary)
-------------	---

9 Appendix Two – Random forest Percentage Split

```
RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.13 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      366      73.2 %
Incorrectly Classified Instances    134      26.8 %
Kappa statistic                    0.2365
Mean absolute error                 0.3465
Root mean squared error             0.4155
Relative absolute error             82.4031 %
Root relative squared error         89.9467 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      ----
      0.247    0.052    0.679     0.247    0.362     0.285    0.771    0.618    bad
      0.948    0.753    0.739     0.948    0.830     0.285    0.771    0.884    good
Weighted Avg.  0.732    0.537    0.720     0.732    0.686     0.285    0.771    0.802

=== Confusion Matrix ===

  a  b  <-- classified as
38 116 |  a = bad
18 328 |  b = good
```

Figure 8 - Random forest before the data is Normalized & Standardized using Percentage Split

The classifier model used for this analysis is a RandomForest implemented with bagging, employing 100 iterations and a base learner of `weka.classifiers.trees.RandomTree`. Notably, it was built swiftly, indicating a relatively small dataset. However, the evaluation on the test split reveals a moderate performance, with 73.2% of instances correctly classified and a Kappa statistic of 0.2365, indicating fair agreement by some benchmarks.

Metrics typically used for regression tasks, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Relative Absolute Error (RAE), and Root Relative Squared Error (RRSE), were also calculated, albeit their interpretation in classification tasks is less straightforward.

Further analysis delves into the detailed accuracy by class, highlighting precision, recall, F-measure, and the Matthews Correlation Coefficient (MCC) for each class. Interestingly, the model achieved higher precision (0.679) for the "bad" class compared to the "good" class (0.247), indicating its better performance in identifying negative instances.

Examining the confusion matrix sheds light on the model's tendencies. It correctly identifies 38 instances of the "bad" class and 18 instances of the "good" class as true positives, while also misclassifying a significant number of instances. Notably, the model exhibits a bias towards the negative class ("bad"), as evidenced by higher precision and a tendency to classify more instances as positive ("bad") than negative ("good"), even if the class distribution might be balanced.

```

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -K 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.32 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correctly Classified Instances      368           73.6 %
Incorrectly Classified Instances    132           26.4 %
Kappa statistic                    0.2544
Mean absolute error                 0.3507
Root mean squared error             0.4166
Relative absolute error             83.3999 %
Root relative squared error         90.1996 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MDC     ROC Area  FRC Area  Class
               0.266    0.055    0.683     0.266    0.383     0.300    ?         ?         bad
               0.945    0.734    0.743     0.945    0.832     0.300    ?         ?         good
Weighted Avg.   0.736    0.525    0.725     0.736    0.694     0.300    ?         ?

=== Confusion Matrix ===

  a  b  <-- classified as
41 113 |  a = bad
19 327 |  b = good

```

Figure 9 - Random forest after the data is Normalized using Percentage Split

The model employed for this evaluation is a RandomForest, consisting of 100 iterations with each base learner being a weka.classifiers.trees.RandomTree. Constructing the model took 0.32 seconds, while testing it on the designated test split consumed a mere 0.02 seconds, highlighting the algorithm's efficiency.

Upon evaluation on the test split, the model achieved a moderate accuracy of 73.6%, correctly classifying 368 instances out of 500. However, 132 instances were misclassified, accounting for 26.4% of the total. Notably, the model exhibited varying performance across classes, with a notably higher recall rate for the "good" class (72.5%) compared to the "bad" class (38.3%).

Examining the Confusion Matrix, the model's struggles in correctly classifying instances of the "bad" class are evident, with only 94 out of 250 instances accurately classified (37.6%). Conversely, it performed better with the "good" class, correctly classifying 274 out of 250 instances (73.4%).

Additional metrics like the kappa statistic (0.2544) and mean absolute error (0.3507) suggest a fair level of agreement between actual and predicted classifications, with some discrepancy. The relatively high mean absolute error and root mean squared error (0.4166) imply that the model's predictions diverge considerably from the actual values.

```

RandomForest

Bagging with 100 iterations and base learner

weka.classifiers.trees.RandomTree -R 0 -M 1.0 -V 0.001 -S 1 -do-not-check-capabilities

Time taken to build model: 0.1 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      367          73.4 %
Incorrectly Classified Instances    133          26.6 %
Kappa statistic                    0.2438
Mean absolute error                 0.3528
Root mean squared error             0.4209
Relative absolute error             83.9054 %
Root relative squared error         91.1189 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                -----  -----  -
                0.253    0.052    0.684     0.253    0.370     0.292    0.752    0.589    bad
                0.948    0.747    0.740     0.948    0.831     0.292    0.752    0.872    good
Weighted Avg.   0.734    0.533    0.723     0.734    0.689     0.292    0.752    0.785

=== Confusion Matrix ===

  a  b  <-- classified as
39 115 |  a = bad
18 328 |  b = good

```

Figure 10 - Random forest after the data is Standardized using Percentage Split

The model described employed a Random Forest algorithm with bagging, utilizing 100 iterations, to construct an ensemble of decision trees. Remarkably, this model was swiftly built, completing in just 0.1 seconds. Evaluation on the testing split revealed a decent performance, with 367 out of 500 instances correctly classified, accounting for 73.4% accuracy. However, there is room for improvement, as indicated by a Kappa statistic of 0.2438, reflecting only fair agreement between predictions and actual labels.

Further analysis using metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) uncovered minor discrepancies between predicted and actual values, with MAE at 0.3528 and RMSE at 0.4209. Although the model demonstrated stronger proficiency in classifying instances labelled as "good," with a 73.4% accuracy rate, its performance in categorising instances labelled as "bad" was notably weaker, standing at a mere 25.3% accuracy. This difference was supported by the confusion matrix, illustrating misclassifications primarily in the "bad" class.

9.1 Comparison

In evaluating the performance of Random Forest models before and after data normalization and standardization using percentage split, several notable trends emerge.

Before normalization and standardization, the initial Random Forest model displayed a moderate performance, achieving an accuracy of 73.2% on the test split. However, it showcased a bias towards the "bad" class, with higher precision for negative instances

compared to positive ones. The confusion matrix revealed a tendency to misclassify instances, particularly in the "good" class.

Following normalization, the model's accuracy slightly improved to 73.6%, with a reduction in misclassification rate. Notably, there was an enhancement in the recall rate for the "good" class, indicating improved sensitivity towards positive instances. However, challenges persisted in accurately classifying instances of the "bad" class, as evidenced by lower recall and precision rates.

After standardization, the model maintained a similar accuracy level of 73.4%, with minor improvements in agreement metrics such as the Kappa statistic. Despite this, disparities in performance between the two classes persisted, with stronger proficiency in categorizing instances labelled as "good" compared to "bad." The confusion matrix reaffirmed this observation, highlighting the model's struggles in correctly identifying instances of the "bad" class.

Overall, while normalization and standardization contributed to marginal enhancements in model performance and agreement metrics, challenges in accurately classifying instances, particularly in the minority class ("bad"), remained prevalent. Further refinement may be necessary to address these disparities and enhance the model's overall predictive capabilities.

10 Appendix 3 – Naive Bayes Percentage Split

```
Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      370          74    %
Incorrectly Classified Instances    130          26    %
Kappa statistic                    0.3527
Mean absolute error                 0.2991
Root mean squared error             0.427
Relative absolute error             71.146 %
Root relative squared error         92.438 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              -----  -----  -
              0.474    0.142    0.598     0.474    0.529      0.357    0.777     0.587    bad
              0.858    0.526    0.786     0.858    0.820      0.357    0.777     0.889    good
Weighted Avg.   0.740    0.408    0.728     0.740    0.731      0.357    0.777     0.796

=== Confusion Matrix ===

  a    b  <-- classified as
73  81 |  a = bad
49 297 |  b = good
```

Figure 11 - Naive Bayes before the data is Normalized using Percentage Split

The model building process completed almost instantly, suggesting a relatively small dataset and the efficiency of the Naive Bayes algorithm for such data sizes. Subsequently, testing the model on the split took a mere 0.01 seconds, reaffirming the dataset's modest scale.

In evaluation, the model correctly classified 370 instances, accounting for 74% of the test data. Conversely, 130 instances were incorrectly classified, comprising 26% of the data. The Kappa statistic, measuring agreement between predictions and actual classifications, yielded a value of 0.3527, indicating a fair level of agreement.

Furthermore, the mean absolute error was calculated at 0.2991, representing the average difference between predicted and actual values, while the root mean squared error, another metric of this difference, was determined to be 0.427. The dataset comprised a total of 500 instances, with the model's performance broken down by class.

The detailed accuracy by class revealed superior performance in classifying instances labelled as "good," contrasting with the performance on the "bad" class. Finally, the confusion matrix detailed the number of correctly and incorrectly classified instances for each class, illustrating specific successes and failures in the model's classifications. For instance, while 73 instances from the "bad" class were correctly classified, 81 were incorrectly categorised as "good".

```

Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances      371           74.2  %
Incorrectly Classified Instances    129           25.8  %
Kappa statistic                     0.3589
Mean absolute error                 0.2991
Root mean squared error            0.4272
Relative absolute error             71.1499 %
Root relative squared error        92.4885 %
Total Number of Instances         500

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      -----  -
      0.481    0.142    0.602    0.481    0.534      0.363    ?         ?         bad
      0.858    0.519    0.788    0.858    0.822      0.363    ?         ?         good
Weighted Avg.   0.742    0.403    0.730    0.742    0.733      0.363    ?         ?

=== Confusion Matrix ===

  a  b  <-- classified as
74  80 |  a = bad
49 297 |  b = good

```

Figure 12 - Naive Bayes after the data is Normalized using Percentage Split

The model building process was remarkably efficient, completing in just 0.01 seconds. However, the resulting model yielded a test data accuracy of 74.2%, indicating moderate performance that leaves room for improvement. The Kappa statistic, measuring agreement between predictions and actual classifications, stood at 0.3589, signifying a fair level of agreement.

Various metrics were employed to assess the model's performance. Mean absolute error gauges the average disparity between predicted and actual values, while root mean squared error gives more weight to larger errors by taking the square root of the squared errors. Relative absolute error and root relative squared error provide additional perspectives by considering the target variable's range.

Examining the confusion matrix, which visually represents the model's performance on each class, reveals discrepancies. Notably, the model demonstrated superior accuracy in classifying instances of class "good" (74.2%) compared to class "bad" (48.1%). This discrepancy suggests a potential bias towards predicting the "good" class, warranting further investigation and refinement of the model.

10.1 Comparison

When comparing the performance of Naive Bayes models before and after data normalization using percentage split, several key observations emerge.

Before normalization, the initial Naive Bayes model demonstrated a rapid building process, indicative of the algorithm's efficiency for small datasets. Testing the model on the split reaffirmed its swift performance. However, despite achieving a respectable accuracy of 74%, the model exhibited discrepancies in class performance. It showcased better accuracy in classifying instances labelled as "good" compared to "bad." The Kappa statistic indicated a fair level of agreement, while metrics like mean absolute error and root mean squared error provided insights into the model's prediction disparities.

Following normalization, the model's efficiency remained high, with a negligible change in building time. Despite a slight improvement in accuracy to 74.2%, the model still displayed similar tendencies of better performance in classifying "good" instances compared to "bad." The Kappa statistic remained consistent, suggesting fair agreement between predictions and actual classifications. However, the confusion matrix highlighted disparities in class accuracy, with the model demonstrating significantly higher accuracy in classifying "good" instances compared to "bad."

11 Appendix 4 – KNN Percentage Split

```
=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.03 seconds

=== Summary ===

Correctly Classified Instances      336           67.2 %
Incorrectly Classified Instances    164           32.8 %
Kappa statistic                    0.1708
Mean absolute error                 0.3287
Root mean squared error            0.5716
Relative absolute error             78.1753 %
Root relative squared error        123.7399 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                -----  -----  -
                0.338    0.179    0.456     0.338    0.388      0.174    0.579    0.358    bad
                0.821    0.662    0.736     0.821    0.776      0.174    0.579    0.728    good
Weighted Avg.   0.672    0.514    0.650     0.672    0.656      0.174    0.579    0.614

=== Confusion Matrix ===

  a  b  <-- classified as
52 102 |  a = bad
62 284 |  b = good
```

Figure 13 - KNN before the data is Normalized using Percentage Split

The IB1 instance-based classifier, representing the 1-nearest neighbour approach, was employed for this classification task, implying a straightforward application of the KNN algorithm. Remarkably, the model's construction time was negligible, taking a mere 0 seconds, likely attributable to KNN's simplicity and the dataset's modest size of 500 instances. Testing the model on a separate test split was similarly swift, taking only 0.03 seconds, highlighting the efficiency of classifying new data points using the trained model.

In terms of performance evaluation, the model achieved a moderate level of accuracy, correctly classifying 336 instances, which accounts for 67.2% of the total. However, it also exhibited a substantial error rate, with 164 instances (32.8%) being misclassified. The kappa statistic, measuring agreement between actual and predicted classifications, yielded a value of 0.1708, indicating slight agreement.

Further assessment metrics such as mean absolute error (0.3287) and root mean squared error (0.5716) provide insights into the average discrepancy between predicted and actual class values, with the latter giving more weight to larger errors. Detailed accuracy metrics for individual classes, including true positive rate, false positive rate, precision, recall, and F1-measure, offer a nuanced understanding of the model's performance across different categories.

The confusion matrix provides a granular view of the model's classifications, delineating instances correctly classified versus those misclassified for each class. For instance, among instances labelled as "bad," 52 were correctly classified while 102 were misclassified as "good."

```

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.03 seconds

=== Summary ===

Correctly Classified Instances      336           67.2 %
Incorrectly Classified Instances    164           32.8 %
Kappa statistic                    0.1708
Mean absolute error                 0.3287
Root mean squared error            0.5716
Relative absolute error             78.1753 %
Root relative squared error        123.7399 %
Total Number of Instances          500

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                -----  -----  -
                0.338    0.179    0.456     0.338    0.388      0.174    ?         ?         bad
                0.821    0.662    0.736     0.821    0.776      0.174    ?         ?         good
Weighted Avg.   0.672    0.514    0.650     0.672    0.656      0.174    ?         ?

=== Confusion Matrix ===

  a  b  <-- classified as
52 102 |  a = bad
62 284 |  b = good

```

Figure 14 - KNN after the data is normalized using percentage split

The classifier model was trained using an IBK instance-based classifier, employing the KNN algorithm with one nearest neighbour ($k = 1$) for classification. The rapid build time of 0 seconds implies either a small training dataset or straightforward computation. Subsequently, the model was evaluated on a test split, with testing taking a mere 0.03 seconds, indicating either a small test dataset or highly efficient model evaluation.

The evaluation results reveal that out of a total of 500 instances, 336 were correctly classified (67.2%), while 164 were incorrectly classified (32.8%). The Kappa statistic, measuring agreement between actual and predicted classifications, yielded a value of 0.1708, indicating poor agreement beyond chance. Furthermore, the mean absolute error of 0.3287 and root mean squared error of 0.5716 quantify the average and squared differences between predicted and actual class values, respectively. The relative absolute error stands at 78.1753%, with a root relative squared error of 123.7399%.

In terms of class-specific performance, precision, recall, F-Measure, MCC, ROC Area, and PRC Area were assessed. However, the overall model accuracy of 67.2% appears moderate, while the low kappa statistic suggests limited improvement over random guessing. Moreover, the high error rates and low MCC scores for both classes indicate suboptimal model performance. Consequently, exploring alternative machine learning

algorithms or refining the KNN model parameters might be beneficial for enhancing predictive accuracy.

11.1 Comparison

Comparing the performance of KNN models before and after data normalization using percentage split reveals several notable trends.

Before normalization, the initial KNN model, employing the IB1 instance-based classifier, demonstrated a swift construction process owing to the simplicity of the KNN algorithm and the dataset's modest size. Testing the model on a separate split was similarly efficient. Despite achieving a moderate accuracy of 67.2%, the model exhibited a significant error rate, with 32.8% of instances being misclassified. The kappa statistic indicated slight agreement between actual and predicted classifications, and further assessment metrics like mean absolute error and root mean squared error provided insights into the average discrepancy between predicted and actual class values. Detailed accuracy metrics for individual classes highlighted disparities in performance, with the confusion matrix illustrating specific successes and failures in classification.

Following normalization, the model's efficiency remained consistent, with negligible changes in construction and testing times. However, despite maintaining the same accuracy level of 67.2%, the model's agreement with actual classifications, as measured by the kappa statistic, remained poor. Metrics such as mean absolute error and root mean squared error quantified the average and squared differences between predicted and actual class values, respectively, with minimal improvement observed. Class-specific performance metrics indicated suboptimal performance for both classes, suggesting limited improvement over random guessing.

BG Data Miining

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

42/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

