COMP4040: Data Mining

— Midterm Exam —

Nguyen Trong Nhan
V202000224

TECHNICAL REPORT

Preliminaries

Please install the necessary packages according to the requirements.txt.

1. Pattern Mining

Question 1

When using the Apriori method to mine frequent patterns of length less than 3 in two datasets, I applied different minimal support thresholds tailored to each dataset's characteristics and use case.

For the Groceries Market Basket dataset, I set the minimum support thresholds at 0.0045, 0.01, and 0.02. Here's how the different thresholds impacted performance:

- With a threshold of 0.0045, the memory usage increased by 2.80 MiB, and the execution time was approximately 0.63 seconds.
- At a threshold of 0.01, memory usage was lower at 0.46 MiB, and execution time decreased to about 0.35 seconds.
- Increasing the threshold further to 0.02 reduced memory usage to 0.28 MiB and shortened execution time to around 0.19 seconds.

For the MoviesLen-100K dataset, which contains 100,000 samples, I used higher thresholds of 0.1, 0.2, and 0.5 to avoid excessive memory usage and longer runtimes associated with generating a large number of itemsets at lower thresholds:

- At a threshold of 0.1, memory usage was significantly higher at 41.47 MiB, and execution time was around 6.77 seconds.
- A threshold of 0.2 reduced memory usage to 8.88 MiB and cut down the execution time to approximately 0.30 seconds.

- The highest threshold of 0.5 minimized memory usage effectively to 0.0 MiB, and the quickest execution time observed was 0.12 seconds.

Overall, increasing the minimal support threshold consistently led to decreases in both execution time and memory usage. This negative linear correlation indicates that higher thresholds limit the number of frequent itemsets generated, thereby reducing computational demands.

Question 2

The processing time of the Apriori algorithm is influenced by several factors, including dataset size, the number of unique items or users, data sparsity, and the support threshold.

For the MoviesLen-100K dataset, which contains over 100,000 records:
- Dataset size: 100,000+
- Number of unique users: 247
- Number of unique movies: 1,682
- Sparsity: 0.937 (considered sparse)

Despite having fewer rows than the Groceries dataset, MoviesLen-100K may require more processing time due to its larger number of unique items (movies), which increases the complexity of transactions. Additionally, the algorithm's efficiency in handling sparse data can impact processing time.

For the Groceries dataset:
- Dataset size: 14,963
- Number of unique transactions: 14,963
- Number of unique items: 167
- Sparsity: 0.985 (highly sparse)

While the Groceries dataset has fewer unique items compared to MoviesLen-100K, it has a higher dataset sparsity, which can affect processing time differently.

When setting support thresholds for Apriori:
- Lower thresholds generate more frequent itemsets, potentially increasing processing time.
- Higher thresholds limit the number of frequent itemsets generated, potentially reducing processing time.

In summary, the processing time of the Apriori algorithm can vary based on dataset characteristics such as size, sparsity, and the number of unique items or users, as well as the chosen support threshold.

Question 3

For the Groceries dataset, I proposed using smaller support thresholds {0.01, 0.02, 0.03} due to the dataset's manageable size and the desire to explore insights while considering the computational resources. Here's the comparison of execution times for Apriori and FP-growth at these thresholds:

- Minimal support of 0.01:
  - Apriori: 0.238 seconds
  - FP-growth: 0.388 seconds

- Minimal support of 0.02:
  - Apriori: 0.081 seconds
  - FP-growth: 0.152 seconds

- Minimal support of 0.05:
  - Apriori: 0.010 seconds
  - FP-growth: [No data provided]

For the MoviesLen-100K dataset, I suggested using larger support thresholds {0.1, 0.2, 0.3} to mitigate the computational burden on slower machines and to draw comparative insights into algorithmic performance. Here are the execution times for Apriori and FP-growth at these thresholds:

- Minimal support of 0.1:
  - Apriori: 6.697 seconds
  - FP-growth: 171.114 seconds

- Minimal support of 0.2:
  - Apriori: 0.190 seconds
  - FP-growth: 1.615 seconds

- Minimal support of 0.5:
  - Apriori: 0.007 seconds
  - FP-growth: 0.027 seconds

Across all tested thresholds and datasets, Apriori consistently outperformed FP-growth in terms of execution time. As the minimum support threshold decreased, both algorithms generally experienced increases in execution time, with FP-growth exhibiting longer times for generating frequent itemsets compared to Apriori. These findings suggest that Apriori is more efficient, particularly for smaller support thresholds and datasets.

2. Classification

Question 1

```
Accuracy: 0.8507371007371007
Confusion Matrix:
[[3484  291]
 [ 438  671]]
```

The confusion matrix shows your model has an accuracy of 85.07%, which may be acceptable depending on the application. The model effectively identifies individuals not earning above $50,000, but the number of False Negatives (438) is relatively high, indicating it misses many individuals who do earn more than $50,000. The performance's acceptability depends on the importance of accurately identifying all high earners and the consequences of misclassification in your specific use case. More improvement might be needed if minimizing False Negatives is critical.

Question 2

```
Accuracy: 0.8531941031941032
Confusion Matrix:
[[3531  244]
 [ 473  636]]
```

Results discussion

The non-linear model slightly improved in overall accuracy and reduced False Positives compared to the linear model. However, it identified fewer True Positives and increased False Negatives, which might be critical depending on the application.

Performance comparison

The non-linear model shows a marginal increase in accuracy (85.32% vs. 85.07%). It reduces False Positives but increases False Negatives and decreases True Positives, suggesting a trade-off that may not be beneficial for all applications.

Performance acceptability

The slight improvement in accuracy and reduction in False Positives may not compensate for the increased False Negatives, especially in contexts where missing high earners is costly. The model's acceptability depends on the specific needs and consequences of misclassification in the intended application. Further adjustments or different modelling approaches may be necessary.

Question 3

Identifying the Problem:

1. Imbalanced Data: The dataset likely has fewer records for individuals earning over $50,000, leading to poor model performance in identifying this minority class.

2. Feature Representation: The mix of continuous and categorical variables may not be optimized, impacting model accuracy.

3. Missing Values: The handling of missing values could significantly affect model performance.

Proposed solutions:

1. Addressing Imbalanced Data:
   - Resampling Techniques: Oversample the minority class or adjust class weights during training.

2. Improving Feature Representation:
   - Feature Engineering: Create new features capturing patterns more effectively.
   - Encoding Technique: Properly encode categorical variables and normalize continuous ones.

3. Handling Missing Values:
   - Imputation: Replace missing values with imputed ones or use indicator variables to denote missing data.

Explanation: These solutions aim to mitigate specific challenges observed in the dataset. Balancing data and enhancing feature representation can lead to more accurate predictions, particularly crucial for datasets with imbalanced classes and varied feature types.

Question 4

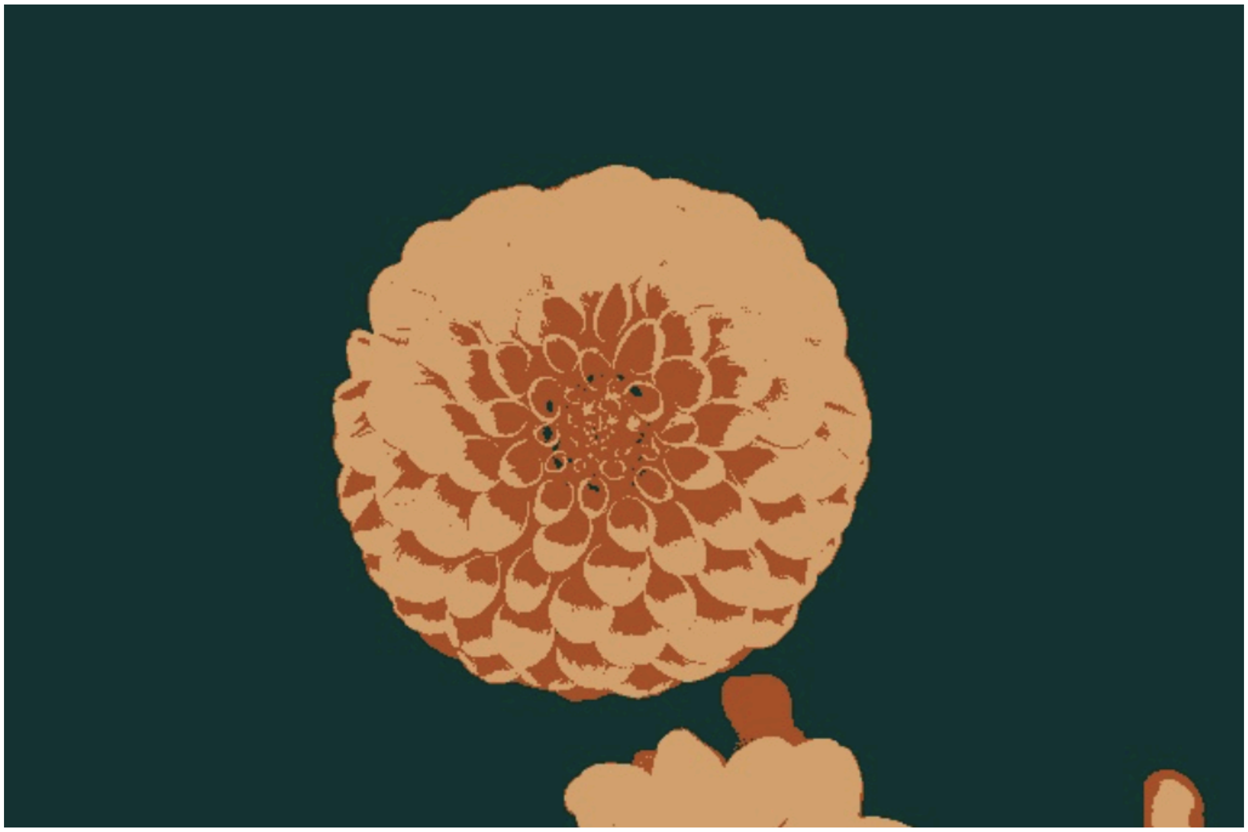
We can use Z-Scores as a method to identify outliers. This technique involves calculating the standard deviation and mean of the data, and then identifying data points that lie several standard deviations away from the mean. Typically, data points with a Z-score greater than 3 or less than -3 are considered outliers. This method is effective for datasets with a Gaussian distribution and helps in pinpointing anomalies that could skew the results of your analysis or model.

3. Clustering

Question 1

Clustering algorithms group similar pixels in an image. By assigning pixels with similar characteristics to the same cluster, clustering can effectively segment an image into regions corresponding to different semantic parts. This is achieved by defining similarity measures between pixels, clustering them accordingly, and then assigning each pixel to its corresponding cluster.

Question 2



This is the result with the number of clusters set to 3. Since it is clear that there are three main colors in the image, we can clearly segment the background, flower, and flower center using this number.

Question 3

The DBSCAN algorithm exhibits a longer runtime compared to alternative methods. For detailed insights and corresponding results, kindly refer to the provided code section.