Data and Information Quality Project

DQ: Accuracy, Dimensionality

ML task: Clustering

Project ID: 43

Luca Andrulli (10868093), Alessandro Spezzapria (10679139)

January 24, 2024

In this Data and Information Quality project, our focus centred on two critical dimensions of data quality: **accuracy** and **dimensionality**. Our machine-learning task was **clustering**. This report elaborates on our setup choices, particularly delving into the implementation details of the data collection phase and the pollution function designed for accuracy assessment.

In the subsequent sections, we provide insights into our pipeline implementation, elucidating the construction of the data preparation phase. Detailed exploration is conducted on how data analysis is executed for both the polluted dataset and the cleaned counterpart.

The latter part of the report is dedicated to presenting the results obtained from our experiments, and we strive to interpret and understand the implications of these results.

1 Setup choices

1.1 Data collection for Accuracy

Accuracy refers to the correctness and reliability of the information gathered or processed in the context of data collection and analysis. In the given scenario, the first step involves creating a dataset with 1000 samples and 4 features using a function provided by the professor. The subsequent decision to implement a polluting function called `make_outliers` is driven by the need to introduce controlled anomalies into the dataset.

The function allows controlled introduction of anomalies, facilitating robustness testing of data analysis methods and assessing the impact of outliers on model performance. The function adds outliers to a DataFrame by generating random values that deviate from the dataset's usual pattern. The idea is to create a duplicate DataFrame to avoid direct changes, calculate the number of outliers based on a certain percentage, and insert these data within the DataFrame at random indices and features. The outliers were generated from a randomly created array with values ranging from 0 to 1. To deviate from the initial DataFrame by a defined distance, the data that will be inserted in a specified location (given by the index and feature mentioned earlier) is multiplied by the distance and the median relative to the feature where the data will be inserted. The median was chosen over the mean to avoid bias.

```
def make_outliers(df, outlier_percentage, outlier_distance):
  df_copy = df.copy()
  n_samples = df_copy.shape[0]
  n_features = df_copy.shape[1]

  num_outliers = int(n_samples*outlier_percentage / 100)
  outliers = np.random.rand(num_outliers)
  polluted_indices_casual = np.random.choice(n_samples,
                                             size = num_outliers,
                                             replace=False)
  for i in range(num_outliers):
    feature_to_pollute = np.random.choice(n_features)
    df_copy.iloc[polluted_indices_casual[i]][feature_to_pollute] =
        outliers[i] * outlier_distance *
                           df_copy.median()[feature_to_pollute]
  return df_copy
```

1.2 Data collection for Dimensionality

In the Dimensionality section, we delved into the exploration of dataset dimensionality through two distinct approaches.

Firstly, we utilized the function `make_dataset_for_clustering`, which allowed us to manipulate the number of samples while keeping the number of features constant at 4. Specifically, we dynamically adjusted the dataset size by randomly generating more samples ranging from 100 to 1000. This method provided insights into how varying the number of samples influences clustering analyses while maintaining a consistent number of features.

Secondly, we extended our exploration using the same function, `make_dataset_for_clustering`. However, in this iteration, our focus shifted towards modifying the number of features while keeping the sample size constant at 1000. By altering the dimensionality in terms of features, we aimed to understand the impact of feature count on clustering outcomes and overall dataset characteristics.

These two perspectives on dimensionality variation — changing the number of samples and adjusting the number of features — contribute to a comprehensive understanding of how dataset dimensions influence clustering results and provide valuable insights for subsequent analyses.

1.3 Outlier Detection Evaluation

In this section, our focus is on data preparation for accuracy assessment, with a specific emphasis on outlier detection.

Given the inherent diversity and nuances associated with the detection of outliers, we conducted a thorough exploration of three distinct algorithms: Z-score, Local Outlier Factor (LOF), and k-Nearest Neighbors (KNN).

The initial algorithm, **Z-score**, represents a statistical approach. Two variations were implemented: the conventional Z-score, calculated through mean and standard deviation, which gauges the number of standard deviations a data point deviates from the mean $Z = \left|\frac{\text{observation}-\text{mean}}{\text{std}}\right|$. The fine-tuning of the threshold parameter in the `ZS(data, threshold)` function was performed through an iterative process to optimize performance. The second implementation, the **Robust Z-score**, was introduced to mitigate the impact of outliers by incorporating the median and median absolute deviation (MAD) in its calculations $Z = \frac{|x-\text{median}(x)|}{\text{mad}(x)}$.

The second method employed was the **Local Outlier Factor (LOF)**, a distance-based algorithm. LOF computes the local density of each data point by contrasting it with the

density of its neighbours. Outliers are distinguished by having substantially lower local density than their neighbours. The LOF algorithm determines the LOF of a data point by comparing its local density to the average local density of its neighbours.

The implementation of the above methods closely follows the class model, featuring an additional functionality such as the `find_index(values, array)` function to retrieve the row index of specific elements.

The third and last method employed is the **K-Neares Neighbours (KNN)** a valuable algorithm for outlier detection due to its adaptability, simplicity, and versatility. By assessing data points based on distances, KNN can identify outliers in both local and global contexts, making it applicable to diverse data distributions. The implementation is encapsulated within the `KNN_outlierdetect(df)` function. The output is, as before, the index of the outliers found.

In the accuracy assessment data preparation phase, we evaluated Z-score, Local Outlier Factor (LOF), and K-Nearest Neighbors (KNN) for outlier detection. Both Z-score and LOF employed statistical and distance-based approaches, respectively. However, our analysis revealed that KNN consistently outperformed the others, emerging as the preferred algorithm in our project.

1.4 OPTICS algorithm modifications

During testing, an error surfaced in the OPTICS algorithm when there was an insufficient number of clusters for performance calculation. To resolve this issue, modifications were made to the code within the `D_data_analysis.py` module. Specifically, in cases where the algorithm failed to calculate the silhouette score due to the absence of clusters, the code was adjusted to set the silhouette score to -1 as a handling measure.

1.5 Other setup choices

To ensure consistent results, a seed of value 2023 was set for the random value-generating functions used in this case study. This also ensures that outliers are created consistently.

2 Pipeline Implementation

2.1 Accuracy

The initial phase involved defining distances and percentages to conduct 10 experiments, each focusing on a distinct distance. Utilizing a for loop, the experiments compared 10 different DataFrames, each injected with varying percentages of outliers, ranging from 5% to 50% of the total number of samples. The cluster algorithms tested included KMeans, Agglomerative, Spectral, OPTICS, and BIRCH. For each experiment, the dataset was created using the `make_dataset_for_clustering` function with parameters set to 1000 for samples and 4 for features. Outliers were injected using the `make_outliers` function. The silhouette of the dataset was then calculated to assess cluster consistency and speed. After identifying outliers with the `KNN_outlierdetect` function, they were removed, and the analysis was repeated, calculating silhouette and speed. The resulting graphs depicted injection rates on the X-axis, with silhouette values in the first case and speed in the second on the Y-axis.

2.2 Dimensionality

In addressing dimensionality, we conducted a total of 20 experiments – 10 by varying the number of samples and another 10 by varying the number of features, as previously outlined. Employing a for loop, we generated 10 values for samples ranging from 100 to

1000 and 1 to 15 for features in ascending order, utilizing the `np.random.choice` function with the appropriate parameters. Following this, the procedure remained consistent. It's worth noting that in these experiments, no corrections were made to the DataFrame, as there were no injections of Data Quality (DQ) issues. Consequently, the analysis was executed only once for each experiment.

3 Result

3.1 Accuracy

In the evaluation of results, our attention was directed towards the silhouette graph, a crucial metric in assessing clustering performance. In the accuracy phase, a notable observation emerged during the analysis of the silhouette graph, particularly in the context of varying outlier distances.

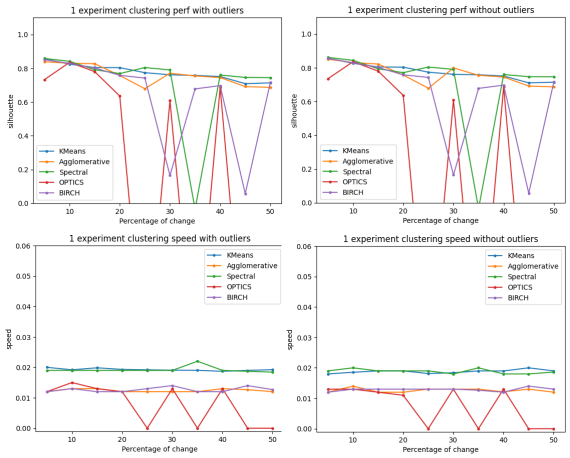


Figure 1: Silhouette for outliers distance of 2

In the initial experiments, conducted with lower outlier distances, the silhouette scores exhibited minimal differences between clustering algorithms applied to the fixed dataset. This is evident in Figure 1, representing the outcomes of the initial round of experiments (as seen before, the data points not seen in the graph are at value -1, we did not represent them in the graph for better visualisation).

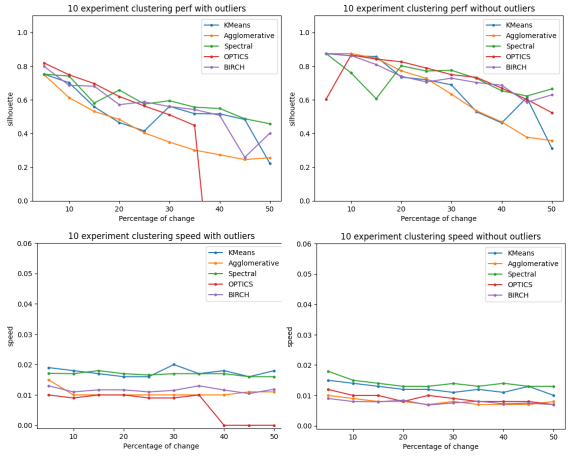


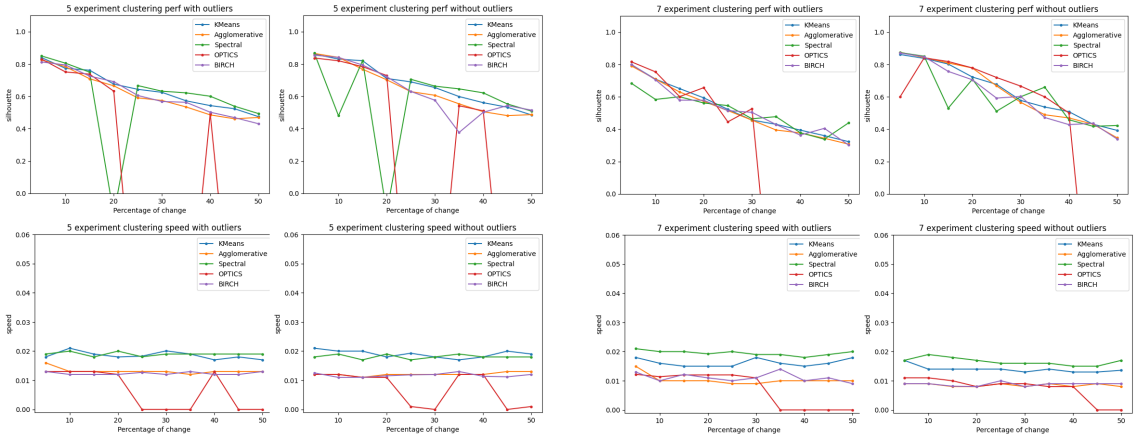
Figure 2: Silhouette for outliers distance of 30

However, as the outlier distances increased, a discernible trend emerged. A noteworthy

improvement in silhouette performance was observed for a lower percentage of outliers. This can be attributed to the fact that, with a smaller percentage of outliers, there are fewer instances to eliminate, resulting in fewer rows being deleted from the dataset (see Figure 2).

Contrastingly, as the percentage of outliers and the outlier distance increased, the outlier detection algorithm demonstrated enhanced efficiency in identifying and deleting more rows. While this is indicative of successful outlier detection, it concurrently led to a substantial reduction in available data for the clustering algorithm. Consequently, the high rate of elimination, coupled with a considerable percentage of outliers, contributed to a lower silhouette score for the clustering algorithm.

A significant enhancement following the outlier detection phase is evident, particularly in the context of larger outlier distances, as reflected in the reduction of misclassified values for the OPTICS algorithm. This improvement is clearly illustrated in Figure 3.



(a) Silhouette for outliers distance of 7

(b) Silhouette for outliers distance of 15

Figure 3

Examining specific instances, such as the one at iteration 5, we observe a notable advancement with a reduction of one misclassified element. This trend gains momentum, as evidenced by iteration 7, where the improvement extends to two misclassified elements. The culmination of these enhancements is vividly illustrated in Figure 2, where a comprehensive improvement is observed across all previously misclassified elements. These findings highlight the pivotal role of outlier detection not only in optimizing clustering outcomes but also in refining the accuracy of specific algorithms, such as OPTICS, by mitigating the influence of outlier-induced misclassifications.

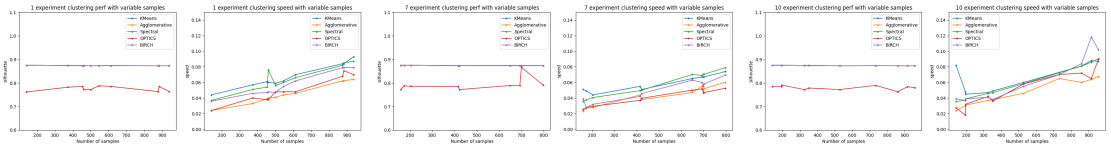
In essence, the trade-off between effective outlier detection and its impact on dataset size played a pivotal role in shaping the overall performance of the clustering algorithm. This nuanced relationship underscores the importance of carefully considering outlier handling policies, especially in scenarios where high percentages of outliers are present, to ensure optimal clustering outcomes.

3.2 Dimensionality

In the dimensionality analysis, a notable disparity emerged between the impact of increasing samples and increasing the number of features.

When the number of samples was increased, a substantial change in the execution speed

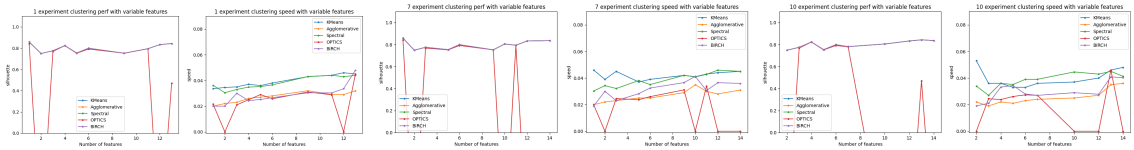
of the examples was observed in relation to the sample size. However, the silhouette graph showed no significant variation, except for the OPTICS algorithm, which exhibited different values in each instance. These changes can be seen in Figure 4.



(a) Experiment 1 (b) Experiment 7 (c) Experiment 10

Figure 4: Sample change

Conversely, when the number of features was increased, the speed of the algorithm did not exhibit a linear increase proportional to the number of features. Instead, the speed appeared more consistent over time. Moreover, there was a discernible decrease in the silhouette value, indicating less accurate clustering with the rise in the number of features. A remarkable difference was observed in the performance of the OPTICS algorithm, which demonstrated an inability to perform well, especially at higher numbers of features. These changes can be seen in Figure 5.



(a) Experiment 1 (b) Experiment 7 (c) Experiment 10

Figure 5: Features change

These findings underscore the sensitivity of clustering algorithms to changes in dataset characteristics, especially concerning the dimensions of the data.