On identifying the compensation for vaguely followed trips

Data Mining Project

Mattia Nardon
University of Trento
mattia.nardon@studenti.unitn.it

Giovanni Scialla
University of Trento
giovanni.scialla@studenti.unitn.it

Eric Suardi
University of Trento
eric.suardi@studenti.unitn.it

1 Introduction

In the dynamic landscape of logistics and transportation, optimizing routes for the efficient movement of merchandise is a critical undertaking for companies seeking to streamline operations and reduce costs. This project addresses the challenges faced by a logistics company responsible for transporting various goods between cities using trucks. The company encounters a common discrepancy between the planned routes (referred to as standard routes) and the actual routes executed by truck drivers. This disparity arises due to drivers' personal preferences, loading variations, and occasional deviations from the standard routes assigned. However, this is just a specific case of a more general problem, which is analyzing and exploiting enormous datasets in order to improve the efficiency of a company. This means that the solution is not bound to this application but it can be adapted to others that are similar in nature, for example: in restaurant chains where proposed menus are not always aligned with what the customers want, to help in the replication of financial portfolios by leveraging the discrepancies between big financial institutions to obtain the best amount of assets, etc.

Our system is designed to align with the company's objectives with those of the drivers, providing actionable solutions for the logistics challenges. Specifically, our methodology yields:

1. Strategic recommendations for the optimization of standard routes, by minimizing the gap between planned and executed routes.
2. Personalized lists of standard routes for each driver, intelligently ordered to reduce the likelihood of deviations. The arrangement ensures a more effective alignment with drivers' preferences.
3. Ideal standard routes for each driver, offering a proactive alternative solution to match the ideal drivers' schedules.

The primary difficulties posed by this problem include:

- Different representation between the cities and merchandise. Selecting and using a specific vectorization for defining with numbers the sequences of cities in the routes, and the same for defining the merchandise and their quantities will pose a problem into defining a correct measure of similarity between different routes.
- Dealing with discrete objects will generate a Non-Euclidean search space of the problem where the classical notion of distance between points in that space will not work.
- Scalability with "Big Data"; in order to make the algorithm scalable with a huge amount of data, an extra effort into optimizing calculations between vectors is needed.

In this work, we present methods that effectively tackle these major challenges, producing a solution that demonstrates to improve the company's needs. In summary, our solution transforms the datasets into numerical values, the cities are encoded using a modified version of K-Shingles, while the merchandise using quantity-hot encoding. The numerical dataset, which hybridizes distance metrics for cities and merchandise, is used to compute a distance matrix between pairs of routes in an efficient and parallelized approach, combining Minhashing and LSH to reduce computations with the speed of NumPy vectorized operations and Python code compilation in C code and parallelization with Numba to exploit all available resources. This method guarantees a reliable and scalable solution for the logistics company's route optimization requirements. The distance matrix is fed into a clustering technique called HDBSCAN, and the resulting clusters are utilized to uncover patterns and select the optimum set of recommended standard routes by locating the routes that minimize cluster divergence.

By analyzing and matching each historical route to the most similar standard route and sequentially identifying the most confident routes based on a score that accounts for the confidence and differences between each actual route, we are able to learn and identify each driver's preferred standard routes where the discrepancy is minimized.

Using previously collected information, a similar method is used to forecast the route that best suits each driver's preferences and create their perfect standard route, which aims to position and summarize their inclination.

We have proven that our method on average has improvements for every task. In Task 1, our approach exhibits a mean improvement of approximately 45%. For Task 2, the top 1 standard route demonstrates a remarkable mean improvement of 68.8%. Finally, in the last task, we observe an average improvement of -4.88% in divergence. The meaning of these improvements will be explained later on. These task-specific improvements highlight the adaptability and effectiveness of our method in addressing diverse challenges.

2 Related Works

Our strategy integrates a number of ideas, methodologies, and algorithms. We used the methods and techniques of reasoning that were covered in the course lectures to get past obstacles that arose throughout the development process. We will provide a brief summary of the most promising methods we investigated during our experimentation phase in the parts that follow. We shall emphasize the main techniques that make up our solution among them.

2.1 Cosine Similarity

Cosine similarity is a measure used to determine the cosine of the angle between two non-zero vectors in an inner product space. In the context of data analysis, it is commonly employed to assess the similarity between documents or vectors in a high-dimensional space.

2.2 Jaccard Distance

Jaccard distance quantifies the dissimilarity between two sets, by 1 minus the Jaccard similarity, which is the ratio of the size of their intersection by the size of their union. It is frequently utilized in various fields, such as information retrieval and bioinformatics, to compare the dissimilarity between sets of data.

2.3 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group, called a cluster, are more similar, in some specific sense defined by a domain expert, to each other than to those in other clusters.

2.4 K-Means

K-means is a clustering algorithm that partitions data into K clusters based on similarity. It iteratively assigns data points to the nearest centroid and recalculates centroids until convergence.

2.5 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is an extension of DBSCAN (Density-Based Spatial Clustering of Applications with Noise) incorporating a hierarchical approach that captures clusters at different levels of granularity, designed for clustering even in non-Euclidean spaces where the number of clusters is automatically determined. HDBSCAN leverages multiple density-based clustering runs with different parameters to enhance robustness and flexibility in identifying clusters.

2.6 K-Shingles

Shingling is an effective way to represent data as sets. A k-shingle for a document is any substring of length k found within the document. Then, it is possible to associate with each document the set of k-shingles that appear one or more times within that document.

2.7 Minhashing

Minhashing is a technique used for quickly estimating the Jaccard similarity between two sets. It is particularly useful in scenarios where traditional methods might be computationally expensive. Minhashing works by using hash functions, used as permutations of the items, on the elements of a set and selecting the minimum hash value as the representative for that set. The similarity between two sets is approximated by the fraction of hash values that match. This allows to efficiently identify similarities among documents and improve the overall performance of the solution.

2.8 Singular Value Decomposition - SVD

SVD is a matrix factorization technique that decomposes a matrix into three other matrices, capturing the underlying structure and relationships within the data. In the context of data analysis, SVD is commonly used for dimensionality reduction, noise reduction, and extracting important features.

2.9 Locality-Sensitive Hashing - LSH

Locality-Sensitive Hashing (LSH) is a method for hashing input data in a way that similar items map to the same "buckets" with high probability. LSH is particularly useful in approximate nearest neighbor search and clustering applications. It helps reduce the search space, making it faster to identify potential matches. By using LSH, it is possible to quickly identify candidate clusters, reducing the computational burden and improving the scalability of the solution.

2.10 Sparse Matrix

Sparse matrices are a type of matrix where the majority of elements are zero. In scenarios where the data exhibits sparsity, such as in high-dimensional spaces or large datasets with many missing values, sparse matrices offer a significant advantage in terms of memory reduction. By employing sparse matrices, not only it is possible to save memory resources but also speed up various matrix operations. This choice is instrumental in enhancing the overall performance and scalability of the solution, particularly in scenarios where the input data exhibit sparsity.

2.11 Numba

Numba is a powerful just-in-time (JIT) compiler designed for Python, specializing in transforming high-level numerical code into optimized machine code. By dynamically compiling functions at runtime, Numba significantly enhances the computational performance of numerical algorithms, making it a valuable tool for accelerating scientific and data-intensive applications in Python.

3 Solution

For clarity, before diving into the details of every single task, we will focus on explaining some of the main techniques that we used to obtain an effective representation of our problem statement and an efficient way to manipulate data. The implementations explained below are more or less common in all the tasks and the workflow pipeline to obtain the Similarity/Distance Matrix is as follows:

1. Represent routes with K-Shingles and Quantity-hot Encoding
2. Vectorization of the Routes using a binary matrix
3. MinHashing for dimensionality reduction
4. Locality-Sensitive Hashing (LSH) to further reduce dimensionality
5. Defining a specific multi-domain Similarity/Distance metric
6. Sparse matrices to reduce memory consumption

3.0.1 K-Shingles and Quantity-Hot Encoding.

To represent travel routes, we designed two different encodings. The cities are encoded into a K-shingle representation. We first categorize cities into unique categorical labels, so each city is assigned an index, and then we produce a unique shingle for each subsequence of K cities present in the data, as a list of K indices. The subsequence follows the same order of the cities in the trips, like a K sliding window, so assuming that a city cannot appear twice in the same route, we are also able to indirectly encode the order of the followed cities. A hash function is then applied to hash the K-shingles into a single representative hash value. We use the Python *hash*() function and we pass the indices of a K-Shingle concatenated as strings separated by commas. The K-shingles present in a route define its set representation. Together, these functions facilitate the transformation of route information into a condensed, standard length, and hashable format for further analysis and efficient similarity comparisons. The number K of shingles is defined in an adaptive way (by default K=3), so that if the dataset is too large or the routes have small trips, the value of K will decrease to 2. It is also possible to set it manually as a parameter when executing the program if the system determines that it won't create enormous sets. The merchandise is instead represented using Quantity-Hot Encoding. To every item, we assign an incremental index. The number of unique items is known from the data so that we can create a vector of the same length for every route where each element contains the sum of the quantities of a specific item in all trips. The whole vector is then normalized dividing by the maximum quantity multiplied by the length of the longest route in terms of trips. We assume that the maximum quantity is the amount of quantity that every item must not exceed in a single trip, and we determine it by the data. This allows us to also have clues about the merchandise carried in the routes in a standard-size vector, at the cost of losing some granular information like the specific trips where the items were transported. The Algorithm 1 shows the implementation.

Before this methodology, we also tried different representations for cities and merchandise: we first vectorized every trip in a stand-alone fashion where the cities were binarized into sets without shingling and the merchandise was a quantity-hot encoded without normalization. These two vectors encoded a single trip, and a route was a concatenation of all the trips. However, this method has inherent problems that produce worse solutions because most routes have a different number of trips so there is no standard for the length of the representations, and padding with zeros the shorter routes still has the problem that for example two identical routes but differing only on one or more trips in the middle messes up the order of all the subsequent trips. All these issues combined made us discard this alternative.

Algorithm 1 Create Shingles

```
1: function CreateShingles($ruotes, k$, Cities, Items, maxQuantity, longestRoute)
2:     Initialize $shingles$ list
3:     Initialize empty lists $citiesInRoute$ and $merchInRoute$
4:     for each $i$, $route$ in $routes$ do
5:         for each $trip$ in $ruote$ do
6:             Save the cities in $citiesInRoute$
7:             for each $merch$ in $trip['merchandise']$ do
8:                 Save merch and add quantity in $merchInRoute$
9:             end for
10:        end for
11:        if len($ruote$) > 0 then
12:            $merchInRoute \leftarrow \frac{merchInRoute}{\text{maxQuantity} \times \text{longestRuote}}$
13:        end if
14:        Initialize empty list $hashedShingles$
15:        for every k-sliding window in $citiesInRoute$ do
16:            $hashed \leftarrow$ Hash the shingles for the trip
17:            Append $hashed$ to $hashedShingles$
18:        end for
19:        Append $[i, hashedShingles, merchInRoute]$ to $shingles$
20:    end for
21:    return $shingles$
22: end function
```

3.0.2 Vectorization as Binary Matrix.

Once we have a collection of K-Shingles encoded route sets we can store them into a binary matrix, where each row represents a route set, each column represents a unique shingle, and the matrix elements indicate the presence (1) or absence (0) of each shingle in the corresponding route set. The merchandise is stored similarly in another matrix. The process has been done separately for both the Standard routes and Actual routes.

3.0.3 MinHashing.

Once we have the binary matrices it's time to choose the number of hashing functions to use in MinHashing. This number is decided automatically since it scales logarithmically with the binary matrix dimensionality. First, we create an empty matrix with the same number of rows and as many columns as the number of hashing functions. Then, we generate randomly hash functions that act as proxies to permutations of the position of the elements, and we efficiently apply them to all the 1s inside each row and store the minimum value (so the first 1 found after the permutation) in the corresponding slot of the signature matrix. We carefully used mostly vectorized operations of NumPy that dramatically improved the speed of computations. The Algorithm 2 shows our implementation.

Algorithm 2 Function for MinHashing

```
1:  function HashCode(numHashes, nCol)
2:      coeffA ← $Random$(numHashes)
3:      coeffB ← $Random$(numHashes)
4:      x ← $Range$(nCol)
5:      hash_code ← ((coeffA ∗ x) + coeffB)%nCol
6:      return hash_code
7:  end function
8:  function MinHashing(matrix, numHashes)
9:      hash_code ← HashCode(numHashes, matrix.shape[1])
10:     signature ← empty()
11:     for row in matrix.rows do
12:         indices ← $Where$(matrix == 1)
13:         signature[row] ← $Zeros$((numHashes)
14:         get_hashes ← hash_code[:, indices]
15:         row_signature ← $Min$(get_hashes)
16:         signature[row] ← row_signature
17:     end for
18:     return signature
19: end function
```

3.0.4 Dimensionality Reduction - Locality-Sensitive Hashing.

We perform Locality-Sensitive Hashing (LSH) on the MinHash matrix to efficiently identify candidate pairs of rows with similar MinHash signatures. The function explained in the Algorithm 3 utilizes a hashing technique to group similar items together in buckets for each band and efficiently narrow down the search space for similarity. It is particularly effective for approximate nearest-neighbor search in high-dimensional spaces. In this context, the MinHash matrix represents the similarity of items, and LSH efficiently identifies candidate pairs with similar characteristics. The approach is suitable since the large datasets make exhaustive pairwise comparisons computationally prohibitive. The choice of bands and columns per band influences the trade-off between sensitivity and specificity in identifying similar pairs. In most implementations, there is a threshold parameter to decide how the balance between the two, but we automatically determine the threshold based on the number of routes present in the dataset to make it scalable and adaptive, starting from near 0 for small datasets to almost 0.9 for bigger ones. However, since the bands must divide perfectly the total columns, we have to adapt the threshold according to the bands and columns per band using the formula: $t = \frac{1}{b}^{\frac{1}{c}}$, where t is the threshold, b is the number of bands, and c is the number of columns, which are determined so that they best approximate the desired threshold. In the implementation of the algorithm, we provide an efficient way to narrow down the search space for further analysis of potentially similar items. The specific pipeline of the code is as follows:

1. Compute the Minhash signature matrix.
2. Divide the columns into bands.
3. Generate hash functions for hashing the bands of the MinHash matrix.
4. Compute hash values for each band of MinHash signatures
5. Update the signature matrix with the computed hash values. Each column in the updated signature matrix corresponds to a band of hash values for a specific row in the MinHash matrix.
6. Find candidate pairs by comparing the similarity of the rows of the updated signature matrix within each band. If the similarity is above a specified threshold, add the pair to the list of candidate pairs.
7. Return the list of candidate pairs that are likely to have similar items based on the MinHash signatures.

In contrast to LSH, we also tried to reduce dimensionality with SVD. But, since we opted to work with a set representation we found that LSH would lead to better results in terms of quality and velocity. In summary, LSH is about efficient similarity search, while SVD is about decomposing a matrix to understand its structure and reduce dimensionality.

3.0.5 Multi-domain Similarity/Distance Metric.

To obtain an effective LSH we had to define an appropriate Distance Metric that can deal with the duality of our problem statement. To do so, we propose a new metric that weights the contributions of both the distance matrix of cities and of merchandise. Algorithm 3 explains the procedure. Three hyperparameters allow us to decide before the execution how to compute and weight the two contributions, as explained below:

- *Metric*: this parameter defines which metric should be used to calculate the distance between different items in the merchandise subset, since the representation in quantity-hot encoding is not canonical, the three possibilities are:
  - *Cosine Distance*
  - *Jaccard Distance*
  - *Euclidean Distance (L2)*

  This parameter does not affect the computation of the distance between the cities since the Jaccard Distance makes more sense for this kind of data.
- *Alpha*: this parameter defines how much importance should be given to the city route distance matrix with respect to the merchandise distance matrix when weighting the contributions (Alpha in [0,1]). It is used in the next parameter as the weight.
- *Fusion*: how to combine the cities distance matrix and the merchandise distance matrix. There are two possibilities:
  - *Weighted Mean*
  - *Weighted Product*

We found that the best combination of hyperparameters is using the Jaccard Distance also for the merchandise, where the intersection is the minimum value element-wise and the union is the maximum value element-wise, the best Alpha is 0.5 and the best fusion method is the Weighted Mean.

Algorithm 3 Compute Multi-Domain Similarity

```
1: function SIMILARITY(Route, Merch, metric, alpha, Fusion)
2:     for $i$ in 0 to $n-1$ do
3:         $intersection \leftarrow Intersection$(Route)
4:         $union \leftarrow Union$(Route)
5:         $routeDistance \leftarrow 1 - \frac{intersection}{union}$
6:         $merchDistance \leftarrow Distance$(metric, Merch)
7:         $similarity[i] \leftarrow Fusion$(alpha, routeDistance, merchDistance)
8:     end for
9:     return $similarity$
10: end function
```

Parallelization and compiling with *C* have been used during the similarity calculation to enhance computational speed.

3.0.6 Sparse Matrices.

The distance matrices are stored as sparse matrices. Sparse matrices are particularly beneficial when dealing with large matrices where a significant portion of the elements is zero. Sparse representations significantly reduce memory requirements. We used the implementations of SciPy. The final route embedding space is shown in Figure 1. The Plotting has been obtained with t-distributed Stochastic Neighbor Embedding (t-SNE) transforming the distance matrix in 3-D. t-SNE is a dimensionality reduction technique that aims to visually represent high-dimensional data in a lower-dimensional space. It focuses on preserving pairwise similarities between data points, emphasizing the preservation of local structures. This helps us to have visual support for the results.

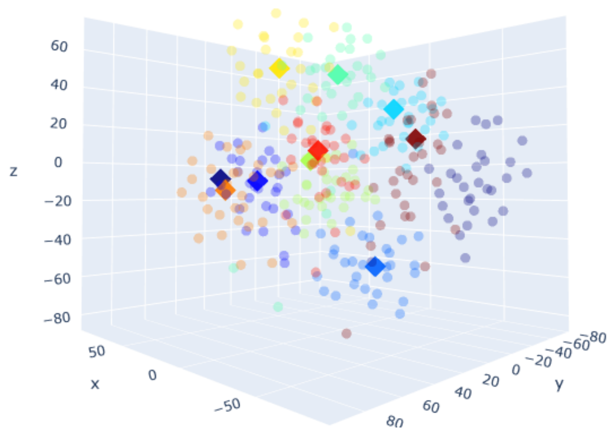

Figure 1. t-SNE 3D representation of the route embedding space; The bright diamonds are the standard routes while the circular points are all the actual routes related to their respective standard route with the same color

3.1 Task 1: Generate Recommendations of Standard Routes

To address the first task, which involves generating recommendations for standard routes, we leveraged the HDBSCAN technique. The methodology employed is outlined in the following sections.

3.1.1 Similarity Computation: First, the process starts by computing the distance between pairs of actual routes. To achieve this, we utilized MinHashing to reduce the dimensionality of route vectors, coupled with Locality-Sensitive Hashing (LSH) to efficiently identify pairs of actual routes for comparison. This approach significantly reduces the computational burden by selectively comparing routes that exhibit a high likelihood of similarity.

3.1.2 Clustering with HDBSCAN:. With the distance scores computed, we applied HDBSCAN to the entire set of actual routes to form clusters. HDBSCAN, being a density-based clustering algorithm, is well-suited for identifying clusters of varying shapes and densities within the data. We used the implementation of Scikit-Learn, however, we encountered many problems related to the library when feeding sparse matrices. After many trials and online research, this forced us to compute the distances between all the pairs returned by LSH, instead of the only necessary ones. This limited by a big margin the capabilities of our solution, by reducing the scalability of the system. We also tried other clustering techniques like DBSCAN and K-Means but the former produced brittle results since it relies mainly on the *eps* parameter but the data are never the same. The latter instead is designed to work in Euclidean spaces, and it is bound to find only K cluster which is fixed and does not allow the solution to find an optimal number of clusters. For us, the system must be able to find patterns in the data, thus if two standard routes produce similar actual routes, this means that the two can be merged into a single standard route.

3.1.3 Identification of Clustroids:

For each cluster generated by HDBSCAN, we determine the clustroids, representing the most central and representative route within that cluster. The clustroid serves as a key reference point for the cluster, capturing its essential characteristics. We define the clustroids as the point in the cluster that minimizes the distance between all the other points in its cluster. We think that the clustroids are a good estimate of the standard routes to recommend since according to the law of big numbers they will minimize the divergence between the goals of the company and the drivers. We did not consider centroids since the space is non-Euclidean and generating a new point not existing in the data is not a wise idea.

By employing this comprehensive approach, we not only identify clusters of similar actual routes, independently of the original standard routes but also provide recommendations for standard routes by selecting the clustroids of these clusters. The HDBSCAN technique, in conjunction with MinHashing and LSH, facilitates an efficient and effective solution for the generation of standard route recommendations.

3.2 Task 2: Generating Personalized Lists of Standard Routes for Each Driver

To create personalized lists of standard routes for each driver, sorted to minimize route diversion, we employ a systematic approach that combines route similarity computation and scoring.

3.2.1 Similarity Computation:

For each driver, we calculate the similarity between each of their actual routes and all standard routes, independently of the original assigned standard route. We leverage all the standard routes to identify if there are routes in the standard set that closely resemble the actual routes. For example, if a driver performs a standard route multiple times in a way that is more similar to another standard route, we consider this as a preference of the driver, so it would be better to assign them the other standard route. Similar to our previous approach, we employ MinHashing, Locality-Sensitive Hashing (LSH), and the previously described similarity metric to quantify the likeness between routes.

3.2.2 Score Computation:

For every actual route, we select the standard route with the highest similarity confidence. We calculate the mean similarity value of each pair of actual routes assigned to the same standard route and then multiply it by the number of occurrences, called support, for that standard route. This results in a composite score that considers both similarity and support. We think that in the top 1 of a driver's list, there should be the standard route which is the most similar and frequent according to the driver's history, since it probably best suits their preferences.

Algorithm 4 Score Computation

```
1: for each driver do
2:     for each $actual\_route$ in driver's history do
3:         $selected\_std\_route \leftarrow standard\_route$ with the highest similarity for that $actualroute$ and save the confidence
4:     end for
5:     for each $standard\_route$ in $selected\_std\_route$ do
6:         $mean\_sim\_std \leftarrow$ mean value for $standard\_route$
7:         $score \leftarrow mean\_sim\_std \times$ num_occurrences($standard\_route$)
8:     end for
9: end for
```

3.2.3 Ranking and Selection:

Once the scores are computed, we sort them in descending order to identify the best-performing standard routes for each driver. We select the top five standard routes as personalized recommendations.

By following this approach, we ensure that the selected standard routes for each driver not only exhibit high similarity to their historical routes but also consider the number of occurrences, providing a robust solution to minimize route diversion. We want to emphasize the fact that the best standard route for a driver could be one that was never assigned to them, in the case that data supports this event.

3.3 Task 3: Driver's Ideal Route

To address this task, standard routes are no longer necessary, as the goal is to derive a generalized ideal standard route for each driver that minimizes divergence among the vaguely followed ones. The proposed solution primarily relies on clustering, again leveraging HDBSCAN, and the computation of intra- and inter-cluster distances.

3.3.1 Clustering for each driver:

Contrary to the first task, we perform the clustering once for each driver's data so that we can have the clusters that describe the driver's history, free from any linkage to standard routes. That is because we don't want any interference from the original standard routes to derive the ideal one, since they can introduce unwanted biases, which will help to describe and classify the "personality" of each driver. This will be useful for the company since it can help to distinguish different preferences, facilitate the assignments of routes, and improve the company's efficiency.

3.3.2 Inter-cluster distance: To choose the cluster that will have the least divergence, two metrics must be derived from the Sparse Distance Matrix. The first one is to minimize the "inter-cluster" distance so that we can find the cluster with which the driver is more confident. We define the inter-cluster distance as the confidence of a cluster, defined as the mean similarity of the points in the cluster multiplied by the size of the cluster. The number of clustered samples N is needed to give more credit to clusters with much more evidence. For example, a driver who performs a standard route just once with high confidence would dominate all the other clusters.

3.3.3 Intra-cluster distance: After picking the right cluster we have an idea of what the driver likes to follow better, now it's time to pick as ideal route the one that minimizes the intra-cluster distance, so the actual route that minimizes the distance between all the others in the same cluster. In this way, we are accounting for his preferred schedule modifications. For example, if the driver likes to stop by Trento every time he does a certain route, we would see this pattern in our embedding space as points very close to each other. The one that would minimize this distance would also account for those preferences.

The algorithmic details of the Intra-cluster and Inter-cluster distance are explained in the pseudo-code Algorithm 5.

Algorithm 5 Inter-Cluster and Intra-Cluster Distance

```
1:  function InterClusterDistance(distancesCluster, N)
2:      mean_distances ← []
3:      for point in distances do
4:          Append to mean_distances the mean distance with other points in same cluster
5:      end for
6:      return (1 − Mean(mean_distances)) × N
7:  end function
8:  function IntraClusterDistance(distances)
9:      mean_distances ← []
10:     for point in distances do
11:         Append to mean_distances the mean distance with other points in the same cluster
12:     end for
13:     return mean_distances
14: end function
```

By weighting the similarity with the number N of total observations found within that cluster, we are effectively selecting clusters not only based purely on distance but also on how much evidence we have about that route being completed in that way. As shown in Figure 2, a route that has been completed plenty of times almost perfectly should be preferred over a route completed very few times, even if perfectly.

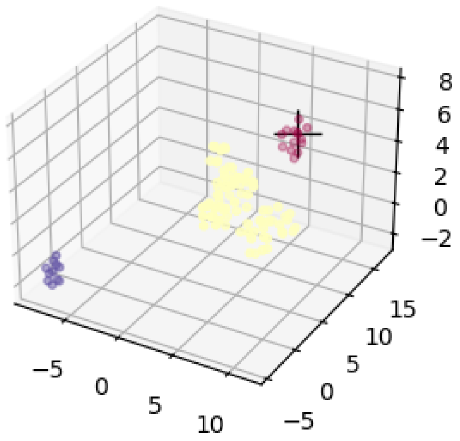

Figure 2. In this example we can see how the red clustered route is being chosen against the more dense purple one because it had more evidence supporting it. The black cross indicates the chosen ideal route that minimizes the intra-cluster distance within the red cluster

4 Experimental evaluation

This section outlines the comprehensive set of experiments conducted to evaluate the effectiveness and performance of the proposed solution.

4.1 Dataset Generation

To assess the quality of our solutions, we implemented some functions to generate large amount of data with a random distribution. First of all, we did a bit of web scraping to obtain a list of possible Italian cities and random items. The algorithm for generating Standard and Actual routes has 140 possible cities and 230 possible items to choose from, while the number of drivers could be set as please. We defined also some parameters to control the randomness and dimensionality of our dataset. In specific we have:

- N_Standard: This parameter controls the number of standard routes that must be generated.
- Divergence: This parameter controls the range of dispersion of the actual routes from the standard routes. It ranges from 0 (less divergence) to 1 (the highest divergence). We define the divergence as possible events happening with the same probability of this parameter, and they are: adding/removing cities, adding/removing/changing the quantities of the items.
- Forward Expansion: This parameter controls the number of actual routes that are generated from each single standard route.

An example of the generated data is shown in Figure 1. To comprehensively evaluate the efficiency and quality of our proposed solutions, we generated three distinct datasets for our experiments, each serving a specific purpose. The datasets were designed with varying sizes to strike a balance between computational efficiency during development and the need for standardized datasets for fair comparisons between different solutions. Specifically, we created datasets of three sizes:

- Small: Approximately 1000 routes
- Medium: Approximately 10,000 routes
- Large: Approximately 50,000 routes

The small and medium datasets allowed for faster computation during the iterative phases of our work, facilitating quick prototyping and testing. Standardizing these datasets ensured that any observed differences in performance across solutions were not merely artifacts of dataset size but rather reflective of the algorithms' inherent capabilities. Finally, the large dataset, with its increased complexity and scale, served as the ultimate testbed for our finalized solutions. This staged approach to dataset sizes offers a thorough understanding of the robustness, scalability, and performance of our solutions in a variety of real-world settings.

4.2 MinHashing for Dimensionality Reduction

In this part we will describe the experiments conducted to assess the efficacy and adaptability of MinHashing for dimensionality reduction, specifically focusing on its performance concerning varying vector dimensions.

4.2.1 Experiment Design. To evaluate the dimensionality reduction capabilities of MinHashing, we conducted experiments with different original vector sizes. The primary goal was to observe how well the method adapts to varying dimensions and to quantify the impact on similarity metrics.

We employed MinHashing to reduce the dimensionality of vectors, allowing for a comparison between the original and reduced representations. Multiple cases were considered and the Table 1 summarizes the percentage of mean difference of the values of similarity between the actual routes and the standard routes for different experiments with varying original and MinHashed vector sizes. In simple terms, it is how close the Minhash similarity is to the original similarity, expressed in percentages where 100% is perfectly accurate.

The experiments conducted affirm that MinHashing is capable of reducing the dimensionality of vectors while preserving similarity, as evidenced by the Table 1 where the mean difference is consistently small even for higher dimensions of vectors. This showcases the effectiveness of MinHashing as a viable technique for applications where efficient dimensionality reduction is crucial. The adaptive nature of MinHashing, reflected in the Table 1, reveals that our approach of dynamically adjusting the number of permutations for the MinHashed vector contributes to further reducing the mean difference. This adaptability underscores the versatility of MinHashing in maintaining similarity across varying vector dimensions.

4.3 LSH for Computation Reduction

To evaluate the efficiency of Locality Sensitive Hashing (LSH) in reducing computations and speeding up the process, experiments were conducted with different dataset sizes. The primary goal was to observe how well the method accelerates computations.

The Table 2 summarizes the mean results obtained from multiple runs with varying dataset sizes and different values for the threshold.

The results demonstrate the effectiveness of LSH in achieving the project's goals of reducing computations and speeding up the overall process. The observed trends across different dataset sizes highlight the scalability and efficiency of LSH, showcasing its potential for real-world applications where computational efficiency is important.

4.3.1 Heuristic Threshold Findings: Through these experiments, valuable heuristic insights were gained regarding the selection of appropriate thresholds based on dataset sizes. It was observed that for smaller datasets, a more permissive threshold could be beneficial to capture a wider range of similarities, while larger datasets might benefit from a more restrictive threshold to focus on stronger similarities. This heuristic approach allows for the customization of LSH parameters based on the specific characteristics of the dataset, providing a practical guide for optimizing performance in diverse scenarios.

4.4 Task 1

The first task of our project involves creating strategic recommendations to optimize standard routes, aiming to minimize the gap between planned and executed routes. To validate the correctness of our solution, we compare the mean distance, standard deviation, and coefficient of variation (ratio of standard deviation and mean) between the original standard routes and the recommended routes from their assigned actual routes. The results obtained from multiple runs using datasets of different sizes are summarized in Table 3.

The results demonstrate the effectiveness of our recommendations in optimizing standard routes, with improvements in mean distance, standard deviation, and coefficient of variation (standard deviation divided by mean) across various dataset sizes.

In Figure 3 it is shown a qualitative t-SNE visualization of standard routes (diamonds) and our recommended standard routes (crosses), in lower dimensions.

4.5 Task 2

The assigned task involved generating personalized lists of standard routes for individual drivers. These lists were intelligently ordered to minimize the probability of deviations, thereby enhancing alignment with the specific preferences of each driver.

Original Size	MinHashed Size	Mean Difference of Similarity (%)
867	200	97.65
	216	97.74
3936	200	98.76
	300	99.20
20462	200	99.55
	400	99.51
21540	200	99.15
	400	99.59

Table 1. Mean Difference of Similarity for Different Experiments, it is the percentage of how close the Minhash similarity is to the original similarity

Dataset Size	Mean Time w/o LSH (s)	LSH Threshold	Mean Time w/ LSH (s)	Reduction of computations (%)
Small (1000)	6.57	0.2	1.51	77.88
		0.5	0.468	98.94
Medium (10000)	1035	0.2	162	68.1
		0.3	71	87
		0.5	19.9	97
Large (50000)	1500+*	0.5	627	95

Table 2. Mean Results of LSH Experiments with Different Dataset Sizes, *the experiment was stopped due to excessive computation time. The provided time is an estimate, and the actual time might have been longer if the experiment had completed.

Large	Std. Routes	Clustroids	Improv. (%)
Mean Distance	0.271	0.140	93.26
Std. Distance	0.119	0.060	98.88
Variation	0.441	0.428	2.91
Medium	**Std. Routes**	**Clustroids**	**Improv. (%)**
Mean Distance	0.243	0.209	16.32
Std. Distance	0.029	0.020	46.58
Variation	0.122	0.100	24.94
Small	**Std. Routes**	**Clustroids**	**Improv. (%)**
Mean Distance	0.305	0.246	27.91
Std. Distance	0.100	0.017	345.46
Variation	0.328	0.072	240.30

Table 3. TASK 1 RESULTS: These results show the mean distance, the standard deviation distance, and the coefficient of variation averaged over multiple runs using different dataset instances

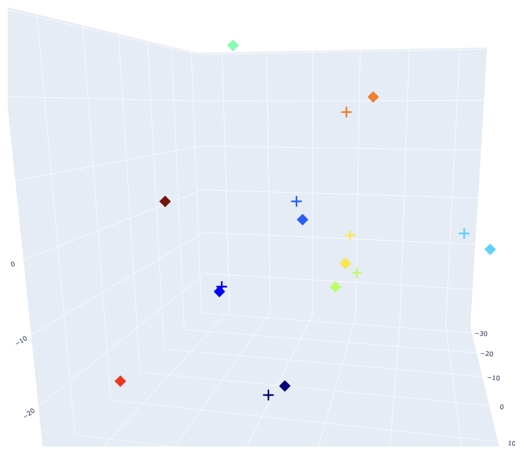

Figure 3. TASK 1: t-SNE visualization of standard routes (diamonds) and our recommended standard routes (crosses)

4.5.1 Methodology. To assess the effectiveness of our approach, we conducted a comprehensive analysis comparing the mean score of drivers (which is the mean divergence of the driver based on historical data, or in simpler terms, the mean divergence of the actual routes per original standard routes) on all standard routes against those on the standard routes identified as the top 5 by our algorithm. This evaluation aimed to measure the degree of improvement in terms of divergence achieved by selecting our top 5 routes in comparison to randomly picking a standard route.

4.5.2 Results. The results aggregated over multiple runs, are presented in the Table 4. The mean scores were averaged across all possible drivers, providing insights into the overall enhancement achieved by opting for our top 5 routes relative to other combinations.

	Mean Improvement (%)
Top1	68.8
Top2	61.5
Top3	56.9
Top4	52.9
Top5	48.9

Table 4. TASK 2 RESULTS: Mean average improvement of the scores by selecting top 5 routes over randomly picking standard routes. *Note: The values in the "Mean Improvement" column represent the percentage improvement in mean scores (divergence reduction) obtained by selecting that ruote from our top 5 routes against picking randomly a standard route.*

drivers (10)	A	B	C	...	Average
Mean Distance	0.674	0.665	0.659		0.588
Best Distance	0.673	0.664	0.648		0.566
Divergence	-0.14%	-0.04%	-1.63%		-4.88%

Table 5. TASK 3 RESULTS: Grade of Dispersion over all possible drivers. A negative value of divergence means that our chosen route is more similar to the driver's preferences.

4.6 Task 3

To confirm the quality of clustering we compared the mean intra-cluster distances of each cluster's best point with the best cluster's point that we picked. This measure of divergence is averaged again over all the possible drivers to obtain an idea of what is the grade of dispersion in picking our ideal route relative to all the other possible ones. Some results over multiple runs are shown in the Table 5. A negative value of divergence means that our chosen route is more similar to the driver's preferences. To visualize the resulting clusters along with the Ideal Driver's route we used the *Uniform Manifold Approximation and Projection (UMAP)* representation. UMAP is a dimension reduction technique that can be used for visualization similarly to t-SNE, but also for general non-linear dimension reduction. A 2D and 3D space of the algorithm results are shown in the figure 4 and 5.

4.7 Limits of our system

While our data mining project has demonstrated good performance across a range of datasets, it is essential to acknowledge certain limitations. The egmMain implementation, despite its efficiency, exhibits a constraint when dealing with datasets larger than 50,000 routes. Beyond this threshold, the computational demands become substantial, impacting the processing speed and, in some cases, leading to resource constraints. However, the main bottleneck are limitations of the implementation of HDBSCAN with sparse matrices that forced us to compute quadratic pairwise distances on the subset of the LSH pairs, instead of linearly the only necessary pairs.

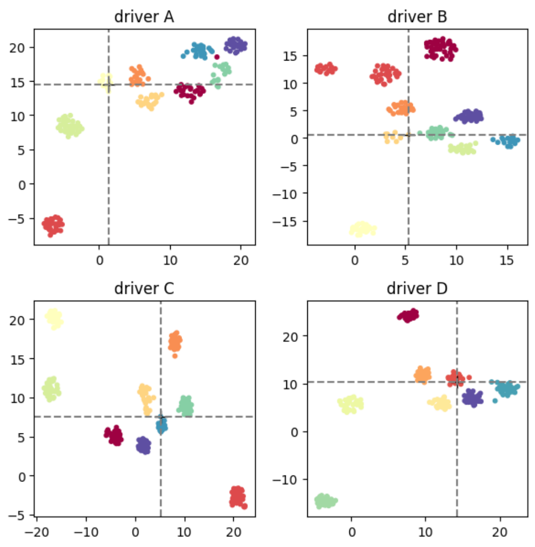

Figure 4. UMAP 2D representation of the HDBSCAN clustering for the first 4 drivers. the selected point is the Ideal Route which, if performed, each driver will have the less divergence with

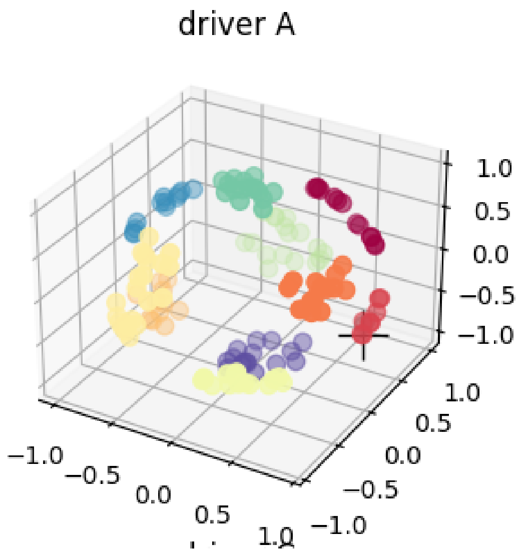

Figure 5. UMAP 3D Representation of the ideal route that driver A would have the less divergence.

On the other hand, the faster variant, egmMainFast, while optimized for speed, reveals its own limit when confronted with datasets exceeding 5,000 routes. The trade-off for speed in this version imposes a restriction on the size of datasets it can effectively handle.

The average timings for the egmMain implementation are calculated across various machines; however, it is important to note that variations may arise. For the Small dataset, it takes approximately 20 seconds, for the Medium approximately 600 seconds, and for the Large approximately 1000 seconds.

These constraints, although present, provide valuable insights for future improvements, potentially guiding enhancements for scalability in subsequent versions of the project.

5 Conclusion

In this work, we researched, analyzed, and implemented various optimization techniques of Data Mining to devise an efficient solution in such a way that it can handle big amounts of data, that are also sparse, discrete, and defined in a non-euclidean space. The main advantages of our solutions are:

- An efficient way of computing the distance matrix between different routes using dimensionality reduction techniques such as K-Shingles, MinHashing, LSH and Sparse matrices.
- A novel notion of distance that can work with the dual-domain of our problem statement.
- A clustering-based approach, leveraging HDBSCAN, to find patterns about the preferred schedules of the company's drivers.
- Optimizations in terms of computation using fast vectorized operations and compiled functions to speed up even more the execution time.

Our evaluation process proved that our solution produces on average improvements for the company on datasets generated without biases.