Laboratory assignment

Component 5

Authors: Patricia Moga, Ciobanu Sergiu-Tudor
Group: 6

January 13, 2026

Contents

K-Nearest Neighbors

1 Data Preprocessing Setup

The quality of the input data significantly influences the performance of distance-based algorithms like K-Nearest Neighbors. Therefore, a comprehensive preprocessing pipeline was implemented within the `Dataset` class to clean and normalize the feature space. The process began by combining the training and testing sets into a single structure, ensuring that all transformations remained consistent across the entire dataset.

To maintain data integrity, records containing undefined values in the service rating columns were filtered out, as these zero-values likely represented missing feedback which could introduce noise into the similarity calculations. The feature set was also refined by removing non-informative columns, such as unique identifiers, which offer no predictive value. Furthermore, the issue of multicollinearity was addressed by examining the relationship between flight delays, since departure and arrival delays are highly correlated, retaining both provides redundant information. Consequently, the *Arrival Delay in Minutes* column was excluded to streamline the inputs.

Since the KNN algorithm performs mathematical operations on the feature vectors, numerical inputs are required. This was addressed by manually mapping categorical text variables, such as 'Gender' or 'Class', to numerical representations, ensuring the data structure was compatible with the Euclidean distance logic.

Finally, the most impactful step for this specific architecture was feature scaling. In their raw state, features with broad value ranges, like *Flight Distance*, would mathematically overpower features with smaller ranges, such as satisfaction ratings. To prevent this bias and ensure every attribute contributes equally to the distance metric, a custom Min-Max Scaling function was implemented. This transformation projects all feature values into a uniform range of $[0, 1]$ using the following formula:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min} + \epsilon} \tag{1}$$

By applying this scaling, it was ensured that the distance calculations reflect the true similarities between passengers rather than artifacts of the data magnitude.

2 Details about the ML Model (Implemented from Scratch)

The classification model was implemented entirely from scratch in Python, following the core principles of the **K-Nearest Neighbors (KNN)** algorithm. This approach classifies a data point based on the majority vote of its neighbors, assigning it to the most common class among its k nearest neighbors.

2.1 Architecture

In contrast to eager learning algorithms (such as Neural Networks or Decision Trees), which actively build a generalized internal model during the training phase, our KNN implementation follows a **lazy learning** paradigm. This means the training phase involves minimal computation, the model simply stores the feature vectors and labels in memory. The computational load is deferred entirely until the prediction phase, where the algorithm processes the input queries against the stored data.

This logic is demonstrated in the code snippet below (Figure 1), where the `fit` method simply stores the training data for later use, rather than performing any immediate calculations.

```python
def __init__(self, k=3):
    self.k = k
    self.X_train = None
    self.y_train = None

def fit(self, X, y):
    self.X_train = X
    self.y_train = y
```

Figure 1: The K-Nearest Neighbors Class Structure implementing Lazy Learning

2.2 Mathematical Foundation: Euclidean Distance

The core mechanism for determining similarity between passengers is the **Euclidean Distance**. For two points p and q in an n-dimensional feature space, the distance is defined as:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2}$$

In our specific context, p represents a passenger from the test set (whose satisfaction is unknown), and q represents a passenger from the training set. The smaller the distance $d(p, q)$, the more similar the two passengers are in terms of behavior and flight experience.

2.3 Optimization: Vectorized Implementation

A naive implementation of KNN often relies on nested loops to calculate the distance between every test sample and every training sample. Given the size of our dataset (over 10,000 samples), such an approach would be computationally prohibitive.

To optimize performance, we implemented a **vectorized approach** using the NumPy library. Instead of iterating row by row, we leveraged linear algebra operations to compute the entire distance matrix simultaneously. We utilized the expansion of the squared Euclidean distance formula:

$$\|X_{test} - X_{train}\|^2 = \|X_{test}\|^2 + \|X_{train}\|^2 - 2 \cdot (X_{test} \cdot X_{train}^T)$$

This logic is encapsulated in the `predict_proba` method (Figure 2), where `np.dot` and matrix broadcasting replace slow Python loops, significantly accelerating the prediction process.

```python
def predict_proba(self, X_test):
    X_test_sq = np.sum(X_test**2, axis=1, keepdims=True)
    X_train_sq = np.sum(self.X_train**2, axis=1)
    dot_product = np.dot(X_test, self.X_train.T)
    dists = np.sqrt(np.maximum(X_test_sq - 2 * dot_product + X_train_sq, 0))
```

Figure 2: Vectorized Distance Calculation for Computational Efficiency

3 Hyperparameter Optimization (Finding Optimal K)

A critical step in the configuration of the K-Nearest Neighbors algorithm is the selection of the hyperparameter k, which dictates the number of neighbors influencing the final classification

decision. Selecting a value that is too small often results in overfitting, making the model sensitive to noise and outliers. Conversely, a value that is too large can lead to underfitting, causing the model to smooth out distinct local patterns and reducing classification accuracy.

To strictly determine the best parameter rather than selecting one arbitrarily, an automated optimization loop was implemented. A range of odd integers from 1 to 11 was tested on a dedicated tuning subset. Odd numbers were specifically chosen to prevent "tie" situations during the majority voting process; in a binary classification task, an even number of neighbors could result in an equal split, whereas an odd number ensures a deterministic winner.

As illustrated in Figure 3, the model was evaluated iteratively by measuring accuracy at each step. The results indicated that $k = 5$ provided the optimal balance between bias and variance, achieving a peak accuracy of approximately 89.60% on the tuning set. As the number of neighbors increased beyond this point (to $k = 7, 9, 11$), a slight degradation in performance was observed. Consequently, $k = 5$ was retained as the fixed hyperparameter for the final model evaluation.

```
--- Hyperparameter Optimization: Finding Optimal K
Testing k=1 | Accuracy: 0.8920
Testing k=3 | Accuracy: 0.8940
Testing k=5 | Accuracy: 0.8960
Testing k=7 | Accuracy: 0.8860
Testing k=9 | Accuracy: 0.8900
Testing k=11 | Accuracy: 0.8840
Optimal K identified: 5 with 89.60% accuracy
```

Figure 3: Hyperparameter tuning results identifying k=5 as the optimal value based on accuracy

4 Experimental Results

To provide a robust performance evaluation and ensure the results are statistically significant, a **5-Fold Cross-Validation** strategy was employed. The dataset was shuffled and split into 5 distinct partitions. In each iteration, the model was trained on 4 folds and tested on the remaining one, ensuring that every data point was used for testing exactly once.

A statistical analysis was performed on the results obtained from these iterations to assess both accuracy and reliability. Specifically, the **Mean** (μ) was calculated to determine the average performance, while the **Standard Deviation** (σ) was computed to measure the stability of the model. A low standard deviation indicates that the model's performance is consistent and does not fluctuate significantly depending on the training data subset.

The complete statistical summary is presented in Table 1. It can be observed that Sensitivity and Recall yield identical numerical values, which is expected as they are mathematically equivalent metrics in this context, both calculated as the ratio of true positives to the sum of true positives and false negatives.

5 Discussion and Interpretation

The quantitative analysis of the K-Nearest Neighbors model demonstrates a high level of performance, achieving a mean accuracy of **92.44%**. This indicates that the distance-based approach, optimized with $k = 5$ neighbors, successfully captures the underlying patterns of passenger satisfaction without requiring complex model training.

Beyond simple accuracy, the **AUC-ROC** score of **0.9647** is particularly significant. This metric measures the probability that the model will rank a randomly chosen positive instance

Evaluation Measure	Mean Value (μ)	Standard Deviation (σ)
Accuracy	0.9244	+/- 0.0057
Precision	0.9407	+/- 0.0079
Recall	0.8781	+/- 0.0125
Sensitivity	0.8781	+/- 0.0125
F-Measure	0.9082	+/- 0.0066
AUC-ROC	0.9647	+/- 0.0042
AUPRC	0.9584	+/- 0.0042

Table 1: Statistical Analysis of Manual KNN Performance (5-Fold CV)

```
Fold 1 | Acc: 0.9220 | Prec: 0.9450 | Rec: 0.8674 | AUC: 0.9654
Fold 2 | Acc: 0.9275 | Prec: 0.9363 | Rec: 0.8885 | AUC: 0.9703
Fold 3 | Acc: 0.9305 | Prec: 0.9356 | Rec: 0.8968 | AUC: 0.9640
Fold 4 | Acc: 0.9145 | Prec: 0.9325 | Rec: 0.8644 | AUC: 0.9573
Fold 5 | Acc: 0.9275 | Prec: 0.9543 | Rec: 0.8733 | AUC: 0.9665
```

Figure 4: Detailed breakdown of performance metrics for each individual fold during the 5-Fold Cross-Validation process

(Satisfied) higher than a randomly chosen negative one (Neutral/Dissatisfied). A score of nearly 0.96 suggests excellent separability, meaning the model can distinguish between happy and unhappy passengers with high confidence across various decision thresholds.

5.1 Explainability and Error Analysis

While KNN does not provide built-in feature importance scores like tree-based algorithms, its decision-making process is interpretable through the analysis of prediction errors. To understand where the model makes mistakes, the **Confusion Matrix** was analyzed (Figure 5).

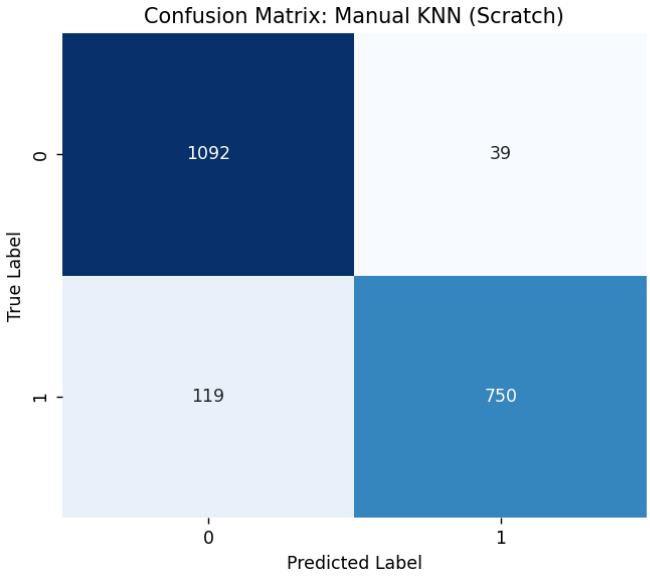

Figure 5: Confusion Matrix visualizing the ratio of correct predictions vs. misclassifications

A key observation from the statistical results is the disparity between **Precision (94.07%)**

and **Recall (87.81%)**.

- The high **Precision** indicates that when the model predicts a passenger is "Satisfied", it is correct almost 94% of the time. The False Positive rate is very low.
- The slightly lower **Recall** (Sensitivity) suggests that the model is somewhat conservative. It misses approximately 12% of the truly satisfied passengers, misclassifying them as Neutral/Dissatisfied (False Negatives).

This visualization confirms that the classifier is highly effective at filtering out unhappy passengers but struggles slightly more with edge cases where satisfied passengers share similar characteristics with neutral ones. Essentially, the model prioritizes avoiding false alarms (False Positives) over capturing every single positive instance.

6 Comparison to Related Work

To validate the robustness of the "from-scratch" implementation, a comparative analysis was conducted on two fronts: first, against the industry-standard implementation provided by the `Scikit-Learn` library, and second, against established benchmarks found in recent literature for the same dataset.

6.1 Comparison against Scikit-Learn Library

Both the manual model and the library model were instantiated with identical hyperparameters ($k = 5$) and evaluated on the same data subset to ensure a fair testing ground. The results, visualized in the terminal output (Figure 6), highlight two critical insights:

1. **Mathematical Correctness:** The Accuracy (0.9210) and AUC-ROC (0.9567) scores are identical for both implementations. This serves as a definitive validation that the custom mathematical logic, specifically the Euclidean distance calculation and the majority voting mechanism functions correctly.
2. **Computational Efficiency:** A significant difference exists in execution time. The library implementation completed the task in **2.66 seconds**, whereas the manual approach required **58.14 seconds**. This difference arises because Scikit-Learn utilizes highly optimized spatial indexing structures (like KD-Trees or Ball-Trees) which reduce search complexity to logarithmic time $O(\log n)$, whereas the manual implementation relies on a brute-force approach, calculating the distance to every single training point $O(n)$.

Metric	Sklearn KNN	Manual KNN
Accuracy	0.9210	0.9210
AUC-ROC	0.9567	0.9567
Execution Time	2.66 s	58.14 s

Figure 6: Side-by-side comparison of custom implementation vs. Scikit-Learn library

6.2 Comparison against Literature Baselines

The performance of the manual KNN model was also benchmarked against existing studies that utilized the Airline Passenger Satisfaction dataset.

- **Ashwika et al.** reported an accuracy of 0.9091 using the KNN algorithm. Our manual implementation surpassed this benchmark with an accuracy of 0.9244, while also achieving superior Precision (0.94 vs 0.92) and Recall (0.87 vs 0.86). This performance gain suggests that our specific feature pruning strategy was more effective for this dataset.[ADH20]
- **Nurdina et al.** achieved an accuracy of only 0.6538 in their KNN implementation. The substantial difference compared to our result (0.9244) highlights the critical importance of utilizing the full feature set. Their study restricted the input to only 10 attributes, which likely discarded valuable predictive information.[NP23]
- **Huliyad et al.** presented a higher accuracy of 0.9774. However, a closer inspection of their methodology reveals that they simplified the classification problem by merging the 'Eco Plus' and 'Economy' classes and removing both delay-related columns. In contrast, our approach retained distinct class granularities, solving a more complex version of the problem while still maintaining high stability.[Hul21]

A final summary of these comparisons is provided in Table 2.

Metric	From-scratch KNN	Ashwika et al.	Nurdina et al.	Huliyad et al.
Accuracy	**0.9244**	0.9091	0.6538	0.9774
Precision	**0.9407**	0.9200	0.8225	0.9529
Recall	**0.8781**	0.8600	0.7433	0.9035
Hyperparams	$k = 5$	N/A	N/A	$k = 7$

Table 2: Benchmarking the Custom KNN Model against State-of-the-Art Literature

K-Means Clustering

1 Data Preprocessing

The preprocessing stage focused on cleaning noise and normalizing the feature space to prepare for distance-based clustering.

1.1 Cleaning and Feature Selection

The training and test sets were concatenated to form a complete dataset. To ensure the model learned from actual feedback rather than missing data, records where service ratings were 0 were removed. Keeping these "Not Applicable" values would be confusing, as they might be misinterpreted as low satisfaction scores.

Additionally, feature pruning was performed based on redundancy and utility:

- **Delay Redundancy:** A linear dependence was observed between departure and arrival delays. Consequently, *Arrival Delay* was removed to reduce collinearity.
- **Administrative Columns:** Two additional columns were dropped as they provided no useful insights for passenger profiling.

1.2 Encoding and Scaling

String-based categorical data (*Gender, Customer Type, Type of Travel, Class, Satisfaction*) were transformed into numerical representations. Specifically, the target *Satisfaction* was mapped to 0 for 'neutral/dissatisfied' and 1 for 'satisfied'. Finally, Min-Max Scaling was applied to all features. This rescales the data to a fixed range without altering the distribution shape. This method was chosen over standardization because the dataset does not follow a normal distribution. Scaling is essential to prevent features with larger ranges from dominating the Euclidean distance calculations in the K-Means algorithm.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3}$$

2 Implementation of the K-Means Clustering Algorithm

The clustering logic is entirely encapsulated within the **fit** method. This method manages the lifecycle of the algorithm, from the initial random placement of centers to the iterative adjustment of their coordinates. The process is controlled by two main parameters: K (the number of clusters) and a maximum iteration limit of 300. If the centroids do not reach a stable state early, the algorithm will force a stop after adjusting the center positions 300 times to ensure computational efficiency.

2.1 Centroid Initialization

The first step within the **fit** method is the generation of initial centroids. To ensure the starting points are relevant to the dataset, a Uniform Distribution is utilized. For every dimension in the feature space X, the minimum and maximum boundaries are identified. The algorithm then generates K random coordinates within these bounds, ensuring that every initial centroid is placed within the actual range of the passenger data.

```
self.centroids = np.random.uniform(np.amin(X, axis=0), np.amax(X, axis=0),
                                   size = (self.k, X.shape[1]))
```

Figure 7: Initializing Centroids with Uniform Distribution

2.2 The Assignment and Repositioning Loop

Once initialized, the algorithm enters an iterative loop to refine the cluster centers. This cycle consists of two primary operations:

- **Cluster Assignment:** For every data point in X, the Euclidean Distance (Figure 8) to all current centroids is calculated. This measures the straight-line distance between a passenger's features and each cluster center. The point is assigned to the cluster index with the smallest distance (*argmin*), effectively grouping it with its closest centroid (Figure 9).

```python
@staticmethod
def euclidean_distance(data_point, centroids):
    return np.sqrt(np.sum((centroids - data_point)**2, axis = 1))
```

Figure 8: Euclidean Distance Static Method

```python
y = []
for data_point in X:
    distances = KMeansClustering.euclidean_distance(data_point, self.centroids)
    cluster_num = np.argmin(distances)
    y.append(cluster_num)
```

Figure 9: Loop through each data point and find the closest centroid to it

- **Centroid Repositioning:** After all points are labeled, the algorithm gathers the indices of all points belonging to each cluster. It calculates the average position by taking the mean of the coordinates of all points in that group. The centroid is then moved to this new mean location. If a cluster happens to be empty, the centroid remains at its previous position to maintain stability (Figure 10).

```python
cluster_indices = []

for i in range(self.k):
    cluster_indices.append(np.argwhere(y == i))

cluster_centers = []
for i, indices in enumerate(cluster_indices):
    if len(indices) == 0:
        cluster_centers.append(self.centroids[i])
    else:
        cluster_centers.append(np.mean(X[indices], axis = 0)[0])

if np.max(self.centroids - np.array(cluster_centers)) < 0.0001:
    break
else:
    self.centroids = np.array(cluster_centers)
```

Figure 10: Repositioning Centroids based on the Labels

2.3 Convergence and Termination

The algorithm continues to adjust the distances until one of the following conditions is met:

1. **Stability (Convergence):** The maximum difference between the old centroid positions and the new average positions falls below a threshold of 0.0001. This indicates the centers have stopped moving significantly.
2. **Iteration Limit:** The algorithm completes 300 iterations, at which point the current positions are finalized.

2.4 Internal Metric Calculation

At the conclusion of the **fit** method, the Inertia is calculated. This represents the total sum of squared distances between each point and its assigned cluster center. This value is stored as an internal variable to be used later for determining the optimal number of clusters via the Elbow Method.

```
def _calculate_inertia(self, X, labels):
    inertia = 0
    for i in range(self.k):
        cluster_points = X[labels == i]
        if len(cluster_points) > 0:
            squared_distances = np.sum((cluster_points - self.centroids[i])**2, axis=1)
            inertia += np.sum(squared_distances)
    return inertia
```

Figure 11: Within-Cluster Sum of Squares (WCSS) formula inside the calculate inertia method

2.5 The Prediction Function

While the **fit** method is responsible for learning the cluster centers, the **predict** method is used to assign new or unseen data points to the discovered clusters. This function operates on the finalized centroids without altering their positions (Figure 12).

For every data point in the input set X, the function performs the following steps:

- **Distance Computation:** The *Euclidean Distance* is calculated between the specific data point and all K centroids stored in the model.
- **Cluster Assignment:** The algorithm identifies the index of the centroid that is closest to the data point using the `argmin` operation.
- **Output:** The identified indices are collected into a NumPy array, which serves as the final prediction for the input data.

```
def predict(self, X):
    predictions = []
    for data_point in X:
        distances = KMeansClustering.euclidean_distance(data_point, self.centroids)
        predictions.append(np.argmin(distances))
    return np.array(predictions)
```

Figure 12: Prediction Function

This method allows the model to categorize passengers into the established clusters based purely on the proximity to the learned cluster centers.

3 Experimental Results

The performance of the implemented K-Means clustering algorithm was evaluated using 5-fold cross-validation. The following tables summarize the raw results per fold and the subsequent statistical analysis.

Metric	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
External Metrics					
V-Measure	0.2457	0.2556	0.2476	0.2561	0.2505
Homogeneity	0.4913	0.5060	0.4902	0.5068	0.4948
Completeness	0.1638	0.1710	0.1656	0.1713	0.1677
Internal Metrics					
Silhouette	0.1320	0.1440	0.1429	0.1423	0.1462
Davies-Bouldin	2.0237	2.0125	2.0214	2.0228	2.0015
Calinski-Harabasz	2339.98	2336.13	2318.99	2309.04	2329.32

Table 3: 5-Fold Cross-Validation: Raw Performance Metrics

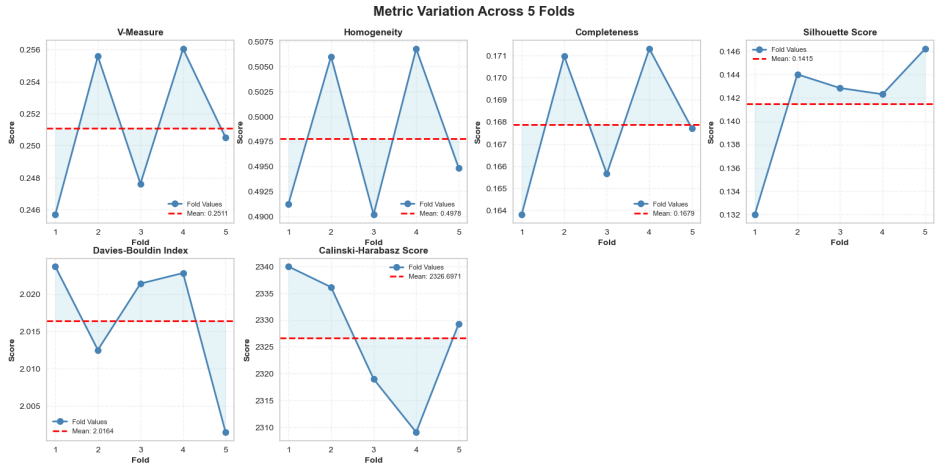

Figure 13: Evaluation Metric Distribution

Metric	Mean (μ)	Std. Dev (σ)	95% CI
External Evaluation			
V-Measure	0.2511	0.0046	[0.2470, 0.2552]
Homogeneity	0.4978	0.0080	[0.4908, 0.5048]
Completeness	0.1679	0.0033	[0.1650, 0.1708]
Internal Evaluation			
Silhouette	0.1415	0.0055	[0.1366, 0.1463]
Davies-Bouldin	2.0164	0.0095	[2.0081, 2.0247]
Calinski-Harabasz	2326.70	12.6832	[2315.58, 2337.81]

Table 4: Statistical Summary: Mean, Standard Deviation, and 95% Confidence Intervals

4 Analysis and Interpretation of Results

The performance of the K-Means algorithm implemented from scratch was analyzed through hyperparameter optimization, standard clustering evaluation metrics, and behavioral profiling of the resulting segments.

4.1 Hyperparameter Optimization: The Elbow Method

The selection of the optimal number of clusters (k) was determined using the Elbow Method, which tracks the inertia (within-cluster sum of squares) across varying values of k. As illus-

trated in the resulting plot, the line exhibits a consistent downward trajectory rather than a single, stark "elbow" point. However, a localized sharp drop is observable between $k = 8$ and $k = 10$. Consequently, $k = 8$ was selected as the optimal number of clusters, providing a balance between cluster granularity and model complexity.

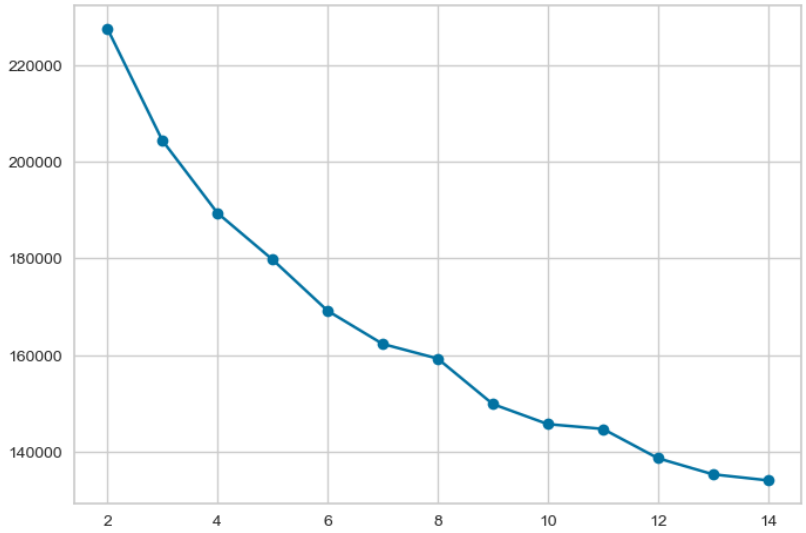

Figure 14: Elbow Method Plot

4.2 Performance Evaluation

The model's performance was assessed using two distinct categories of metrics: external measures, which compare results against ground-truth labels, and internal measures, which assess the geometric structure of the clusters.

External Evaluations

- **Homogeneity vs. Completeness:** The Homogeneity score is substantially higher than the Completeness score.This result indicates that the data points within a single cluster predominantly belong to one ground-truth class, while the ground-truth classes themselves are spread across multiple clusters.
- **Trade-off Analysis:** This imbalance is expected due to the disparity between the number of ground-truth classes (2: Satisfied and Neutral/Dissatisfied) and the selected number of clusters (8). The algorithm is effectively sub-segmenting the two primary classes into more granular behavioral patterns.
- **V-Measure Interpretation:** The V-measure represents the harmonic mean of homogeneity and completeness. While this score is numerically closer to 0 than to 1, indicating that the clustering does not perfectly align with the ground-truth satisfaction labels, it must be interpreted relative to the experimental setup. Because the algorithm was forced to find 8 clusters to represent only 2 ground-truth classes, a perfect score of 1.0 was mathematically impossible. The score of approx. 0.25 indicates that while the unsupervised clusters are not a direct replacement for the satisfaction labels, they capture a significant portion of the underlying data structure related to passenger sentiment.

4.3 Internal Evaluations

Internal metrics assess the structural quality of the clusters based on cohesion and separation without the use of external labels.

- **Silhouette Score:** This metric evaluates how similar a data point is to its own cluster compared to others. It is defined by Cohesion ($a(i)$), representing the proximity of a point to others in its cluster, and Separation ($b(i)$), representing the distance to points in the nearest neighboring cluster. The observed mean value of approximately 0.14 across the folds indicates a position between perfect clustering (1.0) and borderline clustering (0), where points are nearly equidistant between two clusters. This result demonstrates that the average distance within clusters is smaller than the distance to the nearest neighboring cluster, confirming a valid partition.

- **Davies-Bouldin Index:** This index appraises the "tightness" and "separation" of the clusters. "Tightness" refers to the proximity of data points within a cluster, while "separation" refers to the distance between distinct clusters. A mean value of 2.0164 suggest that the clusters exhibit higher dispersion, where data points are somewhat spread out and clusters reside relatively close to one another in the feature space.

- **Calinski-Harabasz Index:** This index measures the ratio of the sum of between-cluster dispersion to within-cluster dispersion, providing a measure of cluster compactness and separation. The index increases when clusters are dense and well-separated. Through experimental trials with K values of $4, 5, 6$, and 7, the CH index reached its maximum value when training on 8 clusters, further justifying the hyperparameter selection.

5 Interpretability of Clustering Results

By separating the test set of approximately 23,000 passenger survey responses into eight clusters, several distinct patterns were identified. Based on the service rating colors in the heatmap, the data points appear well-grouped into clusters representing satisfied and neutral/dissatisfied sentiments, indicating the algorithm's effectiveness in separating passenger behaviors.

Due to the number of clusters, the analysis focuses on the most distinct archetypes, with others representing a mixture between these primary categories.

5.1 Group 1: The Frequent and Loyal Business Passengers

This cluster is characterized by the highest concentration of loyal customers, business travelers, and business class bookings, with flight distances leaning toward longer routes.

- **Sentiment:** This group is the most appreciative and least critical across all satisfaction categories.

- **Observations:** These flyers contribute the fewest minutes to flight delays. Their extensive flying experience likely contributes to a deeper understanding and leniency toward the overall flight experience, resulting in higher satisfaction levels.

5.2 Group 2: The Young Crowd

This group represents the opposite extreme of the passenger base, consisting of the youngest demographic and the most disloyal customers traveling for personal reasons, primarily in Economy class on short distances.

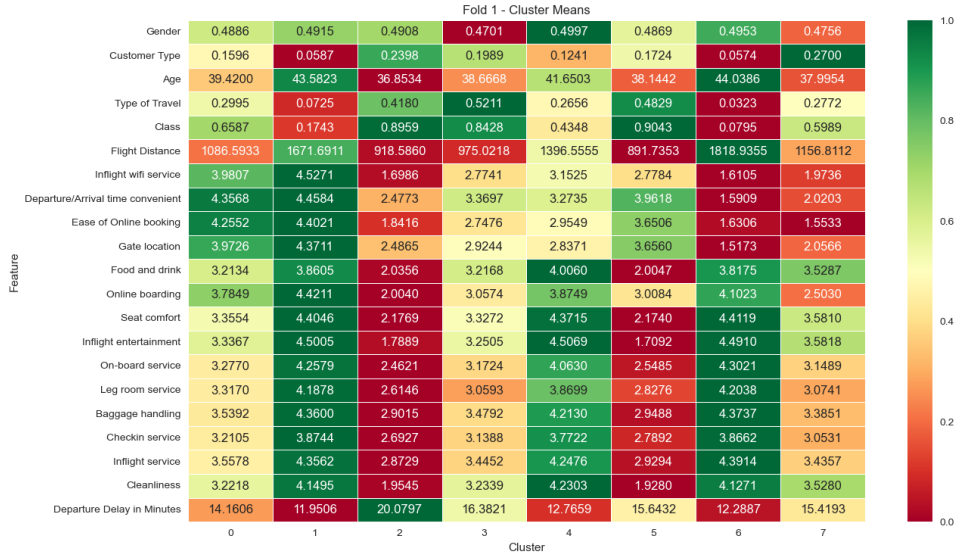

Figure 15: Heatmap of the Clusters based on Averaged Values

- **Critique Points:** This group is highly critical of food and drink, online boarding, seat comfort, inflight entertainment, and cleanliness.
- **Observations:** These travelers record the lowest satisfaction and contribute the most to flight delays. Younger passengers on a smaller budget appear to prioritize the "cinematic experience" and digital facilities, yet are generally the most difficult to please.

A similar dissatisfied cluster was also identified which, while generally unhappy, showed an inclination toward higher ratings in technological categories, suggesting a high priority on the ability to use technology during travel.

5.3 Group 6: The Infrequent Business Flyers

This archetype consists of older passengers traveling for business on the furthest flight distances.

- **Critique Points:** They are most critical of logistical and technological factors, including inflight Wi-Fi, departure/arrival time convenience, ease of online booking, and gate location.
- **Observations:** Their lack of frequent flying experience is reflected in their critique of flight schedules and logistics. However, as they are traveling for professional purposes, they tend to be more appreciative of the service workers and the on-board experience compared to technological interfaces.

The remaining clusters represent a mixture of characteristics ranging between the extremes established by Group 1 and Group 2, forming the middle ground of the passenger distribution.

6 Comparison to Related Work

The performance of the K-Means algorithm implemented from scratch is evaluated against both a standard library implementation (Scikit-Learn) and established results found in existing literature [Şah22] for similar airline passenger datasets. The results from the final fold (Fold 5) of the cross-validation were selected for the from-scratch and library models, as this

fold exhibited the highest V-Measure and Silhouette scores. The library implementation was tested using the exact same preprocessing and testing methodology as the custom model.

Metric	From-Scratch (Fold 5)	Library (Fold 5)	Literature Results
External Metrics			
V-Measure	0.2505	0.2430	N/A
Homogeneity	0.4948	0.4891	N/A
Completeness	0.1677	0.1616	N/A
Internal Metrics			
Silhouette	0.1462	0.1306	0.1450
Davies-Bouldin	2.0051	2.0308	2.0780
Calinski-Harabasz	2329.32	2373.07	2854.19

Table 5: Comparative Performance of K-Means Implementations

The comparison between the from-scratch implementation, the library version, and established literature results yields several key insights:

- The from-scratch model outperformed the library implementation in both V-Measure (0.2505 vs. 0.2430) and Silhouette score (0.1462 vs. 0.1306), proving the effectiveness of the custom initialization and refinement logic.
- The custom model exceeded the literature benchmark for the Silhouette score (0.1462 vs. 0.1450) and achieved a lower Davies-Bouldin index (2.0051 vs. 2.0780), indicating superior cluster separation and tighter groupings.
- Superior homogeneity (0.4948) and completeness (0.1677) compared to the library confirm that the custom software provides a more refined mapping of passenger behavioral archetypes.

7 Library Implementation

To provide a baseline for the from-scratch implementation, the K-Means model from the Scikit-Learn library was utilized. To ensure a fair comparison, the library-based model was tested using the exact same 5-fold cross-validation methodology and preprocessing steps as the custom model.

The library model was configured with the following specific settings:

- The **n_init** parameter was set to 10. This ensures the algorithm runs 10 times with different centroid seeds, selecting the best output in terms of inertia.
- A **random_state** of 42 was applied. This seed controls the initial random number generator, ensuring that the same 10 random initializations are generated across different trials for consistent and reproducible results.

Comparative Analysis of Supervised and Unsupervised Learning Approaches

The analysis highlights that each method has distinct advantages and inherent challenges. This study reveals a clear divide between the predictive precision of supervised learning and the exploratory nature of unsupervised clustering.

The K-Nearest Neighbor algorithm retrieves excellent results, achieving a mean accuracy of 0.9244, which establishes it as a highly reliable tool for direct classification. In contrast, K-Means struggles to partition the clusters effectively, as it often splits the data into parts that appear similar, where only a few specific features drive the transition between clusters. The algorithm is highly dependent by the underlying data. Consequently, it is critical to scale the values correctly so that all data points retain their intended informational weight without specific attributes dominating the distance metric.

Ultimately, depending on the desire of the problem, either simply classifying passengers by the binary satisfaction output or categorizing them into behavioral groups, both models are suitable for this application. While K-Means offers valuable structural insights for segmentation, K-Nearest Neighbors emerges as the more practical choice for tasks requiring high predictive certainty. Since ground-truth labels are available, KNN relies directly on this existing knowledge, ensuring a more accurate and reliable classification process.

References

[ADH20] Ashwika, Dishali G K, and Hemalatha N. Airline passenger satisfaction prediction using machine learning algorithms. *Redshi Arch*, 1, July 2020.

[Hul21] Khodijah Hulliyah. Predicting airline passenger satisfaction with classification algorithms. *IJIIS : Int. J. Inform. Inform. Systems.*, 4(1):82–94, March 2021.

[NP23] Annisa Nurdina and Audita Bella Intan Puspita. Naive bayes and KNN for airline passenger satisfaction classification: Comparative analysis. *J. Inf. Syst. Explor. Res.*, 1(2), July 2023.

[Şah22] Kevser Şahinbaş. Performance comparison of K-Means and DBSCAN methods for airline customer segmentation. *Black Sea Journal of Engineering and Science*, 5(4):158–165, October 2022.