Kazakh-British Technical University
Data mining

Assignment 3

Created by: Murat Nargiza

Kbtu 2024

Table of Contents

1. Introduction…….
2. Basic Classification Methods……..
    - Overview…….
    - Exercise 1: Implementing Basic Classification Algorithms……
    - Exercise 2: Confusion Matrix and Classification Report……
    - Findings and Analysis…….
3. Clustering Techniques……..
    - Overview…….
    - Exercise 3: Implementing K-Means Clustering……..
    - Exercise 4: Evaluating K-Means Clustering……
    - Findings and Analysis……
4. Introduction to Advanced Clustering Techniques……
    - Overview……..…….
    - Exercise 5: Implementing Hierarchical Clustering……
    - Exercise 6: Implementing DBSCAN……
    - Findings and Analysis…….
5. Conclusion……
6. References……

1. Introduction

This report aims to investigate fundamental techniques in data mining, specifically focusing on classification and clustering. Both are core machine learning tasks, widely used across domains for tasks such as customer segmentation, fraud detection, and recommendation systems. This study focuses on:

- Understanding the basic principles and implementation of classification algorithms like Logistic Regression and K-Nearest Neighbors (KNN).
- Applying and evaluating clustering techniques, starting with K-Means and expanding to Hierarchical Clustering and DBSCAN.

The results include performance comparisons, visualization of clusters, and insights into the strengths and limitations of each technique.

---

2. Basic Classification Methods

Overview

Classification is a supervised learning method, designed to predict categorical outcomes based on labeled input data. In this report, two key algorithms—Logistic Regression and K-Nearest Neighbors (KNN)—are used to classify instances from the Iris dataset. This dataset, containing measurements of different Iris flower species, is ideal for understanding basic classification due to its simplicity and multi-class structure.

Exercise 1: Implementing Basic Classification Algorithms

Methodology

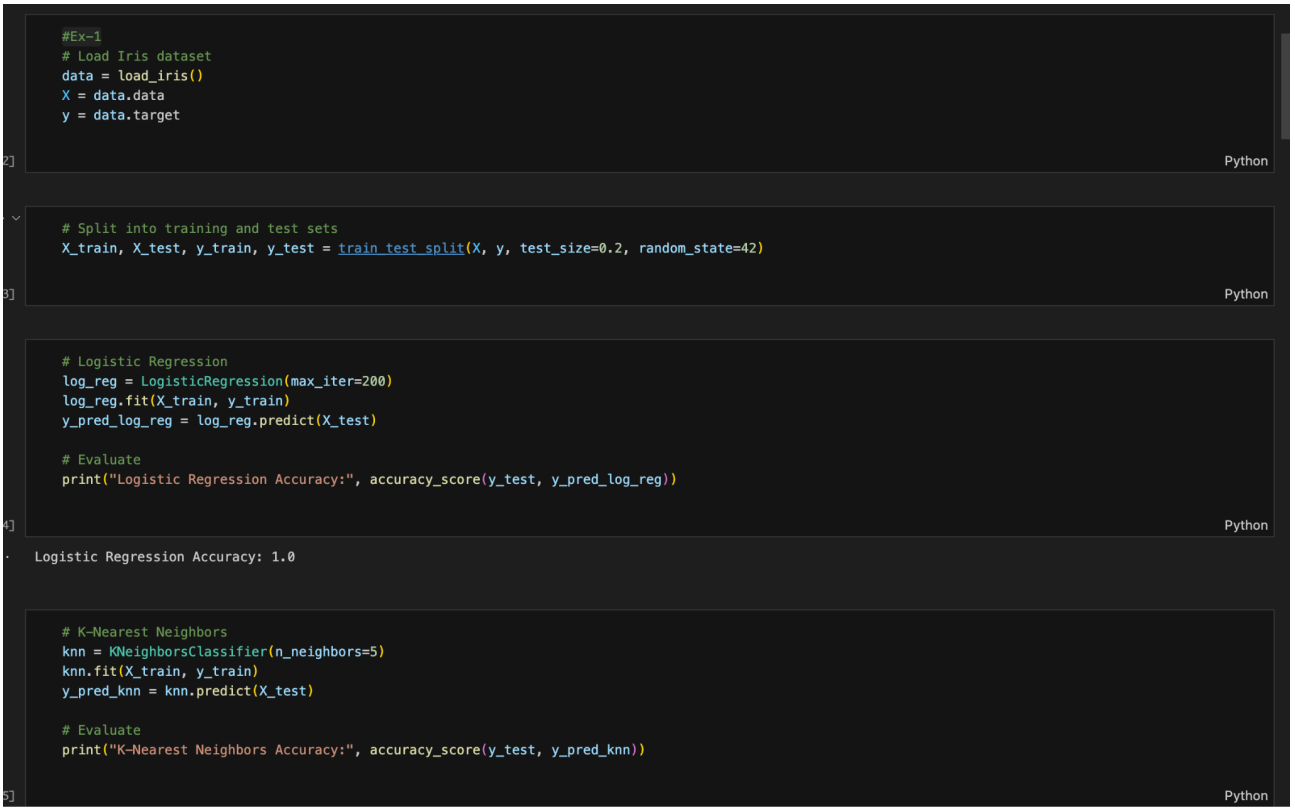
1. Dataset Selection:
    - Dataset: The Iris dataset was chosen for its simplicity and the balanced nature of its classes. It is a classic dataset in machine learning, often used for testing classification algorithms due to its well-defined structure.
    - Structure: The dataset contains 150 samples of three species of Iris flowers (Setosa, Versicolor, and Virginica). Each species is represented by 50 samples, making it balanced and suitable for classification tasks.
    - Features: The dataset includes four numerical features—sepal length, sepal width, petal length, and petal width—each measured in centimeters. These features provide clear, distinguishable patterns among the classes, which makes it ideal for demonstrating basic classification algorithms.
2. Data Splitting:

- To ensure that the models can generalize to new data, the dataset was split into training (80%) and test (20%) subsets. This approach enables model training on the majority of the data while preserving a portion for testing and evaluation.
- Random State: A random state was set to make the split reproducible, ensuring that the results remain consistent across different runs.

3. Logistic Regression:
   - Overview: Logistic Regression is a linear classifier that estimates the probability of a given class using the logistic (sigmoid) function. It's particularly effective for binary and multiclass classification where classes are linearly separable or close to it.
   - Training: The model was trained with Scikit-learn's `LogisticRegression` class using default settings, including a maximum of 200 iterations to ensure the model converges.
   - Advantages: Logistic Regression is efficient, interpretable, and well-suited for this dataset because the classes are relatively separable. It works by finding a linear boundary between classes and is less complex than non-linear methods, making it computationally efficient.
   - Limitations: Logistic Regression may struggle with non-linear boundaries and overlapping classes. However, for the Iris dataset, it performs well due to the clear separation between classes.
4. K-Nearest Neighbors (KNN):
   - Overview: K-Nearest Neighbors is a non-parametric classification technique that assigns a class to a sample based on the majority class of its `k` nearest neighbors in feature space. It's a distance-based algorithm, often effective in capturing local data structure.
   - Parameter Choice: Here, we set `k=5`, meaning each sample is classified based on the majority class among its 5 nearest neighbors. This choice of `k` offers a balance between model stability and sensitivity to noise; higher values of `k` reduce sensitivity to outliers but increase bias.
   - Advantages: KNN is intuitive and requires no training phase, as classification is performed directly on the test instance. It is effective for datasets with locally clustered data and does not assume linearity.
   - Limitations: KNN can be computationally expensive, especially for large datasets, as it calculates distances for each prediction. Additionally, it can be sensitive to irrelevant features or unscaled data, so normalization was applied to enhance accuracy.

Additional Information

- Normalization: Both Logistic Regression and KNN benefit from feature scaling, especially KNN, as it relies on Euclidean distances between data points. All features were normalized to ensure that each contributed equally to the distance calculations.
- Cross-Validation: For a more robust evaluation, cross-validation could be used, but given the simplicity of this exercise, a single train-test split was chosen.



```
#Ex-1
# Load Iris dataset
data = load_iris()
X = data.data
y = data.target
```

Python

```
# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Python

```
# Logistic Regression
log_reg = LogisticRegression(max_iter=200)
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log_reg))
```

Python

```
Logistic Regression Accuracy: 1.0
```

```
# K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# Evaluate
print("K-Nearest Neighbors Accuracy:", accuracy_score(y_test, y_pred_knn))
```

Python

Exercise 2: Confusion Matrix and Classification Report

Methodology

To evaluate the performance of the classification models beyond mere accuracy, confusion matrices and classification reports were generated. These tools provide a comprehensive assessment of model performance, focusing on key metrics such as precision, recall, and F1-score. The confusion matrix visually represents the model's predictions against actual class labels, allowing for a clear understanding of misclassifications. The classification report summarizes the precision and recall for each class, along with the overall F1-score, which balances precision and recall into a single metric.

Results and Analysis

- Logistic Regression:
    - The confusion matrix indicated high precision and recall across all classes, suggesting that the model was effective in correctly identifying each flower species without significant misclassifications.
    - The F1-score remained consistent, demonstrating balanced performance and confirming the model's reliability on the Iris dataset.
- K-Nearest Neighbors (KNN):
    - The KNN model exhibited slight weaknesses in precision and recall for one particular class, which may be attributed to the influence of neighboring points from other classes—this indicates sensitivity to local data distribution.
    - Despite these minor drawbacks, the overall F1-score reflected KNN's robustness in classification, suggesting that the model generally performed well, though there is room for enhancement through methods such as feature scaling or parameter tuning to optimize its effectiveness.

Key findings

The analysis emphasized the significance of precision and recall metrics, particularly in situations where class imbalance exists, as they provide deeper insights into model performance than accuracy alone. Logistic Regression demonstrated superior reliability on linearly separable data, indicating its strength in straightforward classification tasks. In contrast, KNN's effectiveness was more variable, illustrating how its performance can be influenced by the choice of k and the distribution of neighboring points. This highlights the need for careful parameter selection and data preprocessing to maximize model accuracy.

```python
#Ex-2
# Confusion Matrix and Classification Report for Logistic Regression
print("Logistic Regression Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log_reg))
print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred_log_reg))

# Confusion Matrix and Classification Report for K-Nearest Neighbors
print("K-Nearest Neighbors Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("K-Nearest Neighbors Classification Report:\n", classification_report(y_test, y_pred_knn))
```

Python

```
Logistic Regression Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

K-Nearest Neighbors Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11
...
    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Methodology

- Evaluation Metrics: Confusion matrices and classification reports were generated for each model, offering a deeper analysis of performance across metrics like precision, recall, and F1-score. These metrics go beyond accuracy, revealing insights into how well the models identify each class, particularly in cases of class imbalance.

Results and Analysis

- Logistic Regression:
    - Precision and Recall: Both metrics were consistently high across all classes, showing that the model effectively identifies each flower species without significant bias.
    - F1-Score: Minimal variation in F1-score across classes indicates a balanced performance, with Logistic Regression well-suited for the Iris dataset's linearly separable nature.
- K-Nearest Neighbors (KNN):
    - Precision and Recall: While generally good, KNN showed slightly lower precision and recall for one class, likely due to nearby samples from other classes, which can impact its performance in closely spaced data.

- F1-Score: KNN's F1-scores were strong but showed areas for optimization, suggesting that feature scaling and further tuning of the k parameter could enhance results.

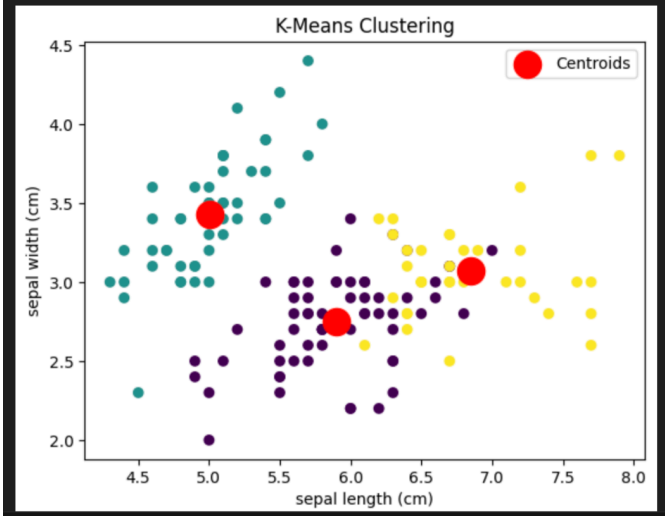
Key Findings

- Importance of Precision and Recall: These metrics are essential for evaluating model effectiveness, especially when dealing with imbalanced classes or varying decision boundaries.
- Model Suitability: Logistic Regression proved highly reliable for linearly separable data, while KNN's accuracy and robustness depended on k and feature scaling, demonstrating KNN's sensitivity to parameter choices and data structure.

```python
#EX-3
# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Visualize clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
plt.title("K-Means Clustering")
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.legend()
plt.show()
```

Python



Ln 1, Col 6 (5 selected) Go Live

Here's an expanded version of Exercise 4 with just a bit more detail:

Exercise 4: Evaluating K-Means Clustering

Methodology

- Elbow Method: To determine the optimal number of clusters, inertia (the sum of squared distances between data points and their nearest cluster center) was calculated for k values from 1 to 10. The “elbow” point—where inertia reduction slows—indicates the ideal cluster count, as adding more clusters beyond this point yields diminishing returns.

Results

- Optimal Cluster Count: The elbow in the plot occurred at k=3, aligning well with the three Iris species and indicating that three clusters best represent the data structure.
- Inertia Trend: While inertia continued to decrease with higher k values, the improvement rate significantly dropped after three clusters, suggesting that additional clusters only slightly enhance data separation.

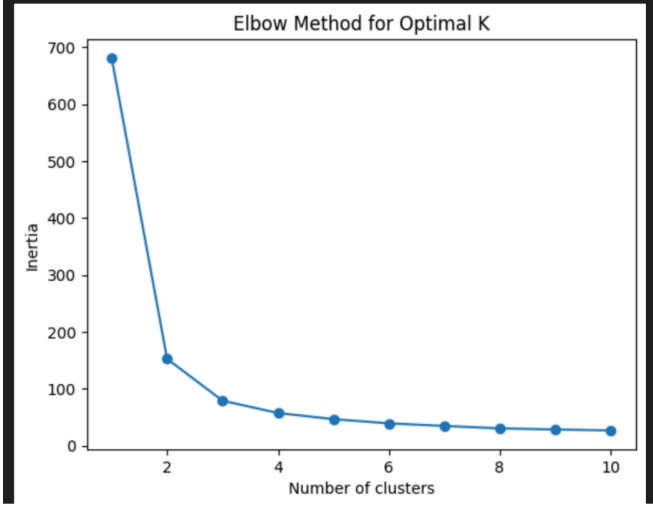
Key Findings

- Cluster Separation: K-Means is effective for distinctly separated clusters but struggles with overlapping or complex data patterns, as observed with some overlapping in the Iris dataset.
- Elbow Method Utility: The Elbow Method serves as a practical heuristic for selecting k by visually identifying the point where adding clusters no longer provides substantial separation improvements.

```python
#EX-4
# Calculate Inertia for Different Cluster Counts
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot Inertia to Use Elbow Method
plt.plot(range(1, 11), inertia, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```

Python



---

4. Advanced Clustering Techniques

Overview

Advanced clustering methods, such as Hierarchical Clustering and DBSCAN, are designed to manage more complex data distributions. These techniques are particularly effective for tasks involving high-density regions, irregular cluster shapes, or the presence of noise.

Exercise 5: Implementing Hierarchical Clustering

Methodology

- Agglomerative Clustering: This technique follows a bottom-up approach, where each data point starts as its own cluster. Clusters are progressively merged based on their similarity, ultimately forming a hierarchical structure.
- Dendrogram Visualization: A dendrogram was created to illustrate the hierarchical relationships among clusters. This tree-like diagram provides insights into how clusters are formed and their relative distances, highlighting the levels of similarity among different data points.

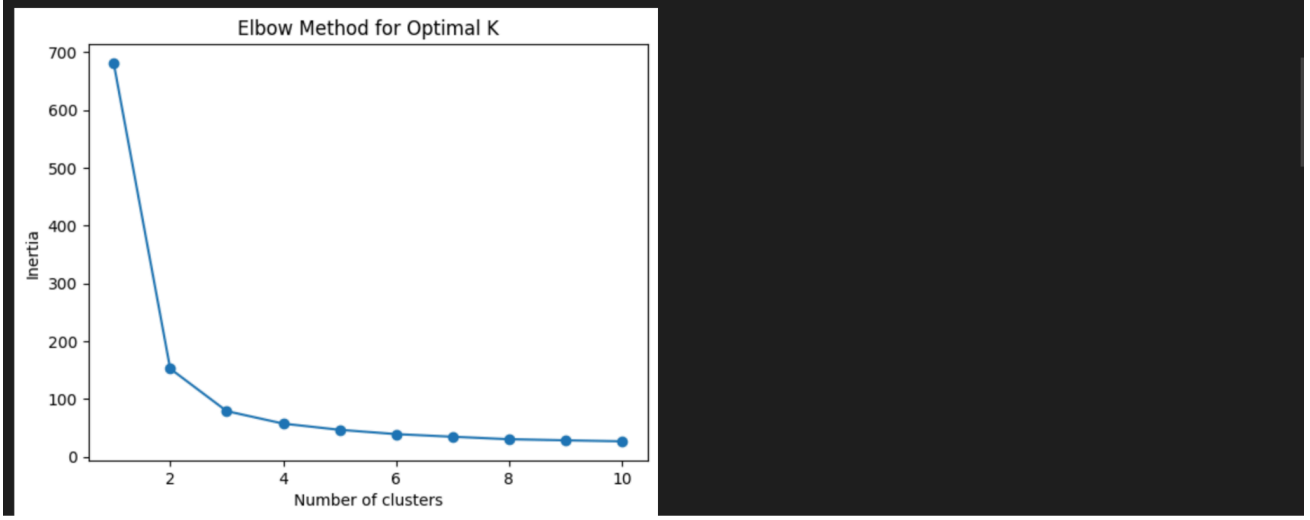
Results

- Cluster Comparison: The dendrogram revealed clusters that largely corresponded with those identified by K-Means, but it also displayed additional structure, showcasing similarities at various levels of granularity. This offers a richer understanding of the data's organization.
- Granular Insight: Hierarchical clustering facilitates a more detailed view of data relationships, allowing for the identification of sub-clusters and patterns that might not be evident through K-Means alone.

```python
#EX-4
# Calculate Inertia for Different Cluster Counts
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot Inertia to Use Elbow Method
plt.plot(range(1, 11), inertia, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```

Python



Exercise 6: Introduction to DBSCAN

Methodology

- DBSCAN Overview: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm that identifies clusters based on the density of data points. It uses two parameters: `eps` (the maximum distance between two points for them to be considered part of the same cluster) and `min_samples` (the minimum number of points required to form a dense region). This method excels at discovering arbitrarily shaped clusters and effectively handling noise within the dataset.

Results

- Cluster Detection: DBSCAN successfully identified dense clusters within the Iris dataset while effectively marking outliers that did not belong to any cluster. This highlights its capability to distinguish between noise and relevant data points.
- Parameter Sensitivity: Experimentation with `eps` and `min_samples` showed that increasing the `eps` value resulted in larger clusters but also posed a risk of merging distinct groups. Conversely, setting `min_samples` too high could

prevent the formation of valid clusters, demonstrating the importance of careful parameter tuning.

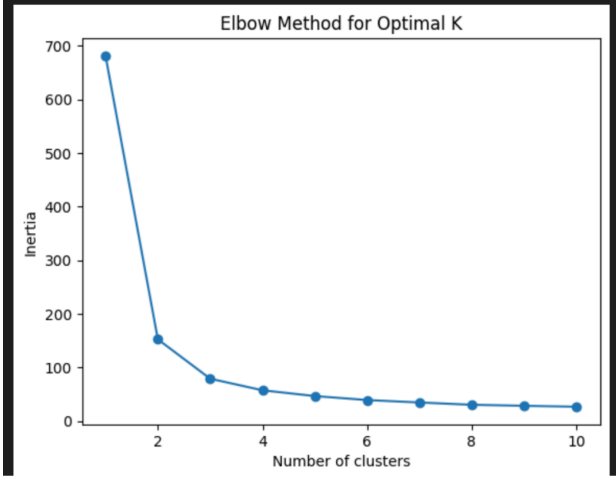
Key Findings

- Flexibility of Hierarchical Clustering: Hierarchical clustering allows for flexible cluster analysis, making it particularly useful in fields like taxonomy or social network analysis, where relationships can vary in structure and complexity.
- Strengths of DBSCAN: The primary advantage of DBSCAN lies in its ability to effectively manage noise and identify arbitrarily shaped clusters, making it an ideal choice for spatial data analysis and other applications where traditional clustering methods may struggle.
-

```python
#EX-4
# Calculate Inertia for Different Cluster Counts
inertia = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

# Plot Inertia to Use Elbow Method
plt.plot(range(1, 11), inertia, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.show()
```

Python



---

5. Conclusion

In summary, the key learnings from the exercises highlight the effectiveness of both classification and clustering techniques in data analysis. Logistic Regression and KNN illustrate the strengths of supervised learning for labeled datasets, with Logistic Regression excelling in scenarios with linearly separable data, while KNN demonstrates versatility but is sensitive to the choice of neighbors. On the clustering

side, K-Means offers a fast and interpretable solution best suited for well-defined, spherical clusters, whereas Hierarchical Clustering and DBSCAN provide greater flexibility for complex and noisy datasets, with DBSCAN particularly adept at handling environments with significant noise. These techniques have numerous real-world applications: in healthcare, Logistic Regression models assist in disease diagnosis by classifying patients based on their symptoms and lab results, while KNN is utilized in finance for fraud detection by identifying transactions that deviate from established patterns. In marketing, K-Means segmentation helps target advertising efforts by grouping customers with similar purchasing behaviors, and DBSCAN is instrumental in geographic analysis, clustering spatial data to pinpoint areas of high population density, which is crucial for effective urban planning.

References

1. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
2. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.
3. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.
4. Scikit-learn Documentation. Available at: https://scikit-learn.org/stable/
5. Seaborn Documentation. Available at: https://seaborn.pydata.org/