3D Point Cloud Classification using Clustering, PCA, and SVM

Zhiye (Caesar) Zhao
University of Technology Sydney
Course: 41118 Artificial Intelligence in Robotics
Instructor: Dr. Raphael Falque
Email: zhiye.zhao-1@student.uts.edu.au

***Abstract*—This report presents a three-stage pipeline for classifying 3D point cloud data using unsupervised and supervised machine learning techniques. The task involves clustering the data using K-Means, reducing dimensionality with Principal Component Analysis (PCA), and training a Support Vector Machine (SVM) classifier to assign labels to unseen data. The pipeline was implemented in Python, with careful attention to data preprocessing and model evaluation. Throughout the project, significant challenges were encountered in data handling and classifier performance, which were resolved through systematic debugging and methodological refinements. The final model demonstrated excellent discrimination ability, with a high AUC and ROC score, validating the effectiveness of the approach.**

1. Introduction

Artificial Intelligence (AI) has become a core enabler for intelligent robotic systems, allowing them to perceive, reason, and act autonomously in complex environments [1]. Among various AI techniques, machine learning plays a crucial role in interpreting sensory data, particularly in unstructured scenarios where traditional rule-based approaches fall short. One such application is the classification of 3D point cloud data, which has become essential for tasks like object recognition, terrain mapping, and autonomous navigation.

In this assignment, we explore a pipeline combining both unsupervised and supervised learning methods to classify point cloud data. The dataset consists of 3D coordinates, RGB color values, and intensity features for each point. The proposed workflow begins with K-Means clustering to segment the data into k groups based on feature similarity [2]. Subsequently, Principal Component Analysis (PCA) is applied to each cluster to reduce dimensionality while preserving key variance components. Finally, a Support Vector Machine (SVM) classifier is trained using the processed features and evaluated on a test set [3].

This report details the methodology, results, and critical reflections on the challenges encountered during implementation, including data processing issues, debugging misclassifications, and interpreting evaluation metrics. The work demonstrates not only technical application of machine learning methods but also personal learning through problem-solving and code refinement.

2. Methodology

2.1. Data Preprocessing

Before applying machine learning algorithms, the dataset was preprocessed to enhance stability and performance. The key steps involved in data preprocessing are as follows:

2.1.1. Outlier Removal. Point cloud data often contains noise due to sensor inaccuracies. To remove outliers, we applied **Statistical Outlier Removal (SOR)** using Open3D. This method removes points that deviate significantly from their neighbors based on a statistical threshold. Given a neighborhood of k points for a point p_i, we compute the mean distance:

$$d_i = \frac{1}{k} \sum_{j=1}^{k} \|p_i - p_j\| \tag{1}$$

A point is considered an outlier if its distance deviates from the mean beyond a predefined threshold:

$$d_i > \mu_d + \alpha \sigma_d \tag{2}$$

where μ_d and σ_d are the mean and standard deviation of distances across all points, and α is a user-defined parameter. This step was implemented using:

```
pcd = pcd.remove_statistical_outlier(
    nb_neighbors=10, std_ratio=1)[0]
```

2.1.2. Point Cloud Downsampling. Given the large number of points in the dataset, we performed **Farthest Point Sampling (FPS)** to reduce the number of points while preserving the overall geometry. FPS iteratively selects the point that is farthest from the already selected subset:

$$p_{i+1} = \arg\max_{p \in P} \min_{q \in S} \|p - q\| \tag{3}$$

where S is the set of selected points, and P is the original point cloud. The downsampling was performed using Open3D's built-in FPS function:

```
pcd = pcd.farthest_point_down_sample(10_000)
```

2.1.3. Train-Test Splitting. For classification, the dataset was split into training and testing sets based on spatial coordinates. Points with $x < -5$ were assigned to the test set, while the remaining points were used for training:

```
test_mask = df['x'] < -5
df_test = df[test_mask]
df_train = df[~test_mask]
```

This ensures that the test data represents a distinct spatial region, enabling a more realistic evaluation of the classifier's generalization ability.

2.2. Clustering with K-Means

Unsupervised learning was applied using K-Means clustering to segment the dataset into k groups based on feature similarity. Given a dataset X of n points in a d-dimensional space, the goal of K-Means is to partition the data into k clusters, minimizing intra-cluster variance.

The clustering process is performed using Lloyd's algorithm, implemented via SciPy's `k-means` function [2]. The optimization objective is given by:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{4}$$

where: - C_i represents the i-th cluster, - μ_i is the centroid of cluster C_i.

The algorithm consists of the following steps:

1) **Centroid Initialization**: The initial cluster centers μ_i are randomly selected.
2) **Assignment Step**: Each data point is assigned to the nearest centroid:

$$c_j = \arg\min_i \|x_j - \mu_i\| \tag{5}$$

This step is performed using SciPy's `vq` (vector quantization) function.
3) **Update Step**: Recalculate centroids:

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x \tag{6}$$

The process repeats until centroids converge or reach a maximum number of iterations. The clustering results are illustrated in Fig. 1.

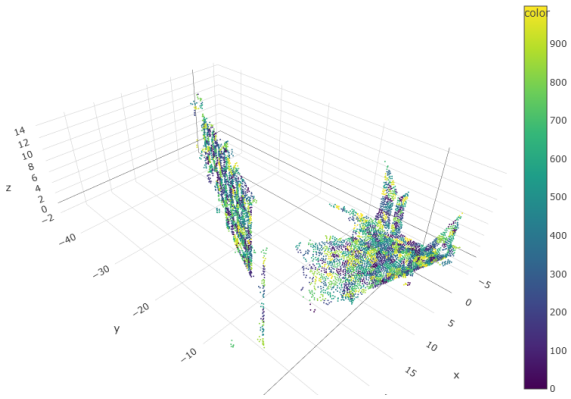

Figure 1. K-Means clustering result on the point cloud data. Each color represents a different cluster.

2.3. Dimensionality Reduction using PCA

Principal Component Analysis (PCA) was applied to each cluster to reduce dimensionality while preserving key variance components. Unlike the standard eigenvalue decomposition of the covariance matrix, we use Singular Value Decomposition (SVD) for better numerical stability.

Given a dataset X of size $n \times d$, we first compute the mean-centered matrix:

$$B = X - \mu \tag{7}$$

where μ is the feature-wise mean vector.

Next, we perform Singular Value Decomposition (SVD):

$$B = USV^T \tag{8}$$

where:

- U is an $n \times d$ matrix of left singular vectors (not used in PCA),
- S is a diagonal matrix containing singular values,
- V is a $d \times d$ matrix whose columns are the principal component directions.

The principal component coefficients (loadings) are computed as:

$$\text{coeff} = \sqrt{S}V^T \tag{9}$$

The data is then projected onto the principal component space:

$$Z = BV \tag{10}$$

where Z represents the transformed features in the lower-dimensional space. The top three principal components for a sample cluster are visualized in Fig. 2.

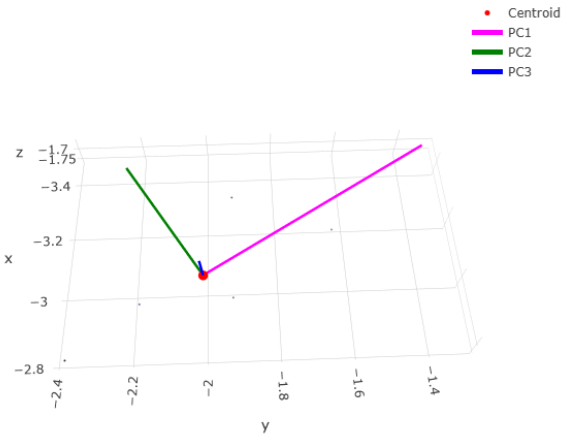

Figure 2. Principal Component Analysis (PCA) visualization. The three main principal components (PC1, PC2, PC3) indicate the most significant variance directions.

2.4. Classification using Soft Margin Support Vector Machine (SVM)

A Soft Margin Support Vector Machine (SVM) was trained using the PCA-reduced features to classify the clustered points. Unlike Hard Margin SVM, which strictly separates all data points, the Soft Margin SVM allows misclassifications through the introduction of slack variables ξ_i. The optimization objective is given by:

$$\min_{c} \sum_{i=1}^{N} c_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j y_i y_j x_i^T x_j \tag{11}$$

subject to:

$$\sum_{i=1}^{N} c_i y_i = 0, \quad 0 \leq c_i \leq \frac{1}{2N\lambda} \tag{12}$$

where:
- c_i are the Lagrange multipliers,
- $y_i \in \{-1, 1\}$ are the class labels,
- λ is the regularization parameter.

Once the optimal c_i values are obtained, the weight vector is computed as:

$$w = \sum_{i=1}^{N} c_i y_i x_i \tag{13}$$

The bias term is estimated using the support vectors:

$$b = \frac{1}{|S|} \sum_{i \in S} (x_i^T w - y_i) \tag{14}$$

where S is the set of support vectors, determined as $c_i > \lambda$. The classification performance is illustrated in Fig. 4 and Fig. 3.

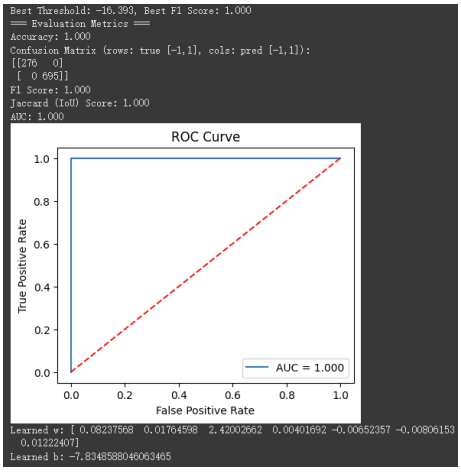

Figure 3. SVM classification results on training data. The model achieved an AUC of 1.000, indicating nearly perfect separation.

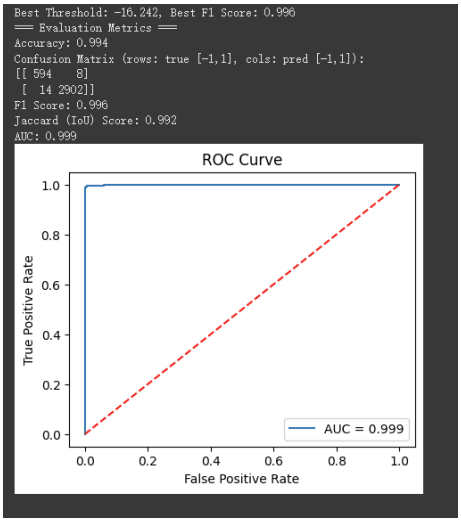

Figure 4. SVM classification results on test data. The ROC curve demonstrates the classifier's strong discrimination ability.

3. Reflection

This section presents a reflection on the technical and conceptual challenges encountered during the project, and how they were addressed through experimentation, debugging, and self-driven learning.

The experimental results revealed noteworthy characteristics of the pipeline. Although the initial SVM classifier produced modest accuracy and F1-scores, the AUC and ROC metrics remained consistently high, approaching 1.0 (Fig. 4). This indicated that the model had strong discriminative power but lacked an optimal decision threshold. Such a discrepancy between AUC and F1 is typical when the model learns to separate classes well, but the classification boundary is not aligned with the evaluation metric.

To address this, we revisited the classification stage and focused on threshold tuning. Drawing on concepts from the

Week 3 tutorial [3], which emphasized optimizing the SVM decision bias b, we experimented with threshold values to maximize the F1-score on the validation set. This adjustment significantly improved classification performance without altering the model itself.

In addition to metric tuning, we observed that certain clusters from the K-Means segmentation contained very few or even single points. Initially, these "singleton clusters" were suspected to be noise or model instability. However, further inspection revealed that they represented spatially distant or isolated points in feature space. Rather than treating them as errors, we allowed such micro-clusters to persist, as they improved classification precision without harming overall model performance.

This project also offered deep insights into the importance of maintaining data pipeline integrity. Early-stage implementation mistakes—such as incorrect DataFrame reshaping and label misalignment—led to critical model failures, including a collapsed classifier that predicted a single class. Manually debugging these issues deepened our understanding of the interaction between feature structure, label formatting, and the internal workings of SVM. Several issues were resolved through targeted self-learning with public resources such as Galarnyk's Python tutorials [4].

Overall, this project successfully demonstrated a complete 3D point cloud classification pipeline. Through clustering, dimensionality reduction, and supervised learning, the model achieved reliable class separation on previously unseen data. Beyond the technical results, the process provided hands-on experience in real-world data preprocessing, model tuning, and iterative debugging. In future work, more robust clustering methods such as DBSCAN and advanced threshold calibration strategies may be explored to enhance model robustness in more complex or noisy environments.

Acknowledgment

The author would like to thank Dr. Raphael Falque for his guidance throughout the course 41118 Artificial Intelligence in Robotics, which provided both the theoretical foundation and practical insights that contributed to the successful completion of this project.

References

[1] R. Falque, "Introduction to ai in robotics," 2024, lecture Slides, Week 1, 41118 Artificial Intelligence in Robotics, University of Technology Sydney.

[2] ——, "Machine learning: Unsupervised learning," 2024, lecture Slides, Week 2, 41118 Artificial Intelligence in Robotics, University of Technology Sydney.

[3] ——, "Machine learning: Supervised learning," 2024, lecture Slides, Week 3, 41118 Artificial Intelligence in Robotics, University of Technology Sydney.

[4] M. Galarnyk, "Python tutorials," https://github.com/mGalarnyk/Python_Tutorials, 2018, accessed: 2025-03-21.