Introduction

In recent years, the rapid advancement of 3D sensing technology such as LIDAR scanning has increased the availability and use of 3D point clouds for various applications, such as urban planning, autonomous vehicles, virtual reality, and heritage site virtual preservation. A 3D point cloud represents a scene as a collection of points in three-dimensional space, where each point corresponds to the surface of an object in the scene, described by its spatial coordinates (x, y, z) and often augmented with additional attributes like colour and intensity. However, these point clouds typically contain a mix of useful and irrelevant data points, such as points corresponding to buildings, vegetation, and other structures, as well as noise and outliers. Effective segmentation and classification of these points are crucial for isolating and analysing specific structures within the scene, such as buildings, while discarding irrelevant data.

This project aims to develop a method for segmenting single-view 3D point clouds using machine learning techniques. Instead of relying on deep learning approaches, which demand extensive training data and computational resources, this project will explore non-deep learning methods, specifically Random Forest and Support Vector Machine (SVM) classifiers. These classifiers will be trained using geometric features extracted from the point clouds, such as eigenvalues, planarity, and sphericity, generated using CloudCompare, a widely-used 3D point cloud processing software.

The goal is to classify each point in the point cloud into one of several predefined categories, such as buildings, cars, or natural terrain, and assess the accuracy of the classifications. By evaluating the performance and accuracy of Random Forrest and SVM on the classifiers, this project seeks to identify effective strategies for point cloud segmentation that balance accuracy and computational efficiency.

Preliminary results indicate that non-deep learning classifiers, when trained on carefully selected geometric features, can achieve satisfactory classification accuracy for single-view 3D point clouds, offering a viable alternative to more resource-intensive deep learning approaches.

System

Design Overview

The primary goal of this project is to segment and classify single-view 3D point clouds using non-deep machine learning techniques. The system is designed to handle large point cloud datasets, extract meaningful geometric features, and train classifiers to accurately label each point within the cloud.

The system is structured into several key components:

1. **Model Training**: Two classifiers, Random Forest and Support Vector Machine (SVM), were trained using the pre-processed and concatenated feature data. The models were

trained on concatenated training data by separating the features into X_train and the labels into y_train. This was compared to the features X_test and labels y_test.

2. **Model Testing and Evaluation**: The trained models are applied to the test set, and their performance is evaluated using metrics such as accuracy, precision, recall, and Intersection over Union (IoU).

3. **Output Generation**: The final classified point clouds are saved in a format compatible with CloudCompare, allowing for further visualization and analysis. Each point is represented by its coordinates and RGB colour, corresponding to its class label. The colour map for the classified labels is as follows:

```
color_map = {
    0: (0, 0, 255),     # Blue
    1: (0, 128, 255),   # Light Blue
    2: (0, 255, 0),     # Green
    3: (128, 255, 0),   # Yellow-Green
    4: (255, 255, 0),   # Yellow
    5: (255, 165, 0),   # Orange
    6: (255, 128, 128), # Light Red
    7: (255, 0, 0)      # Red
}
```

Software and Tools

- **Python 3**: The core of the system is implemented in Python, leveraging its libraries for machine learning and data processing.
- **Scikit-learn**: Used for implementing the Random Forest and SVM classifiers, as well as for cross-validation and hyperparameter tuning.
- **CloudCompare**: Employed for feature extraction from the 3D point clouds. This tool allows for the calculation of various geometric descriptors essential for classification.
- **NumPy and Pandas**: Utilized for data manipulation, normalization, and management of large datasets.

Assumptions and Limitations

- **Feature Selection**: The choice of geometric features is based on their relevance to the classification task. It is assumed that the selected features are sufficient to distinguish between the different classes in the point cloud.
- **Single-View Point Clouds**: The system is designed for single-view point clouds, which may not capture all surfaces in a scene. This limitation is inherent to the dataset and may affect classification accuracy.
- **Non-Deep Learning Approach**: The system uses non-deep learning classifiers, which are less computationally intensive but may not achieve the same level of accuracy as deep learning models on complex datasets.

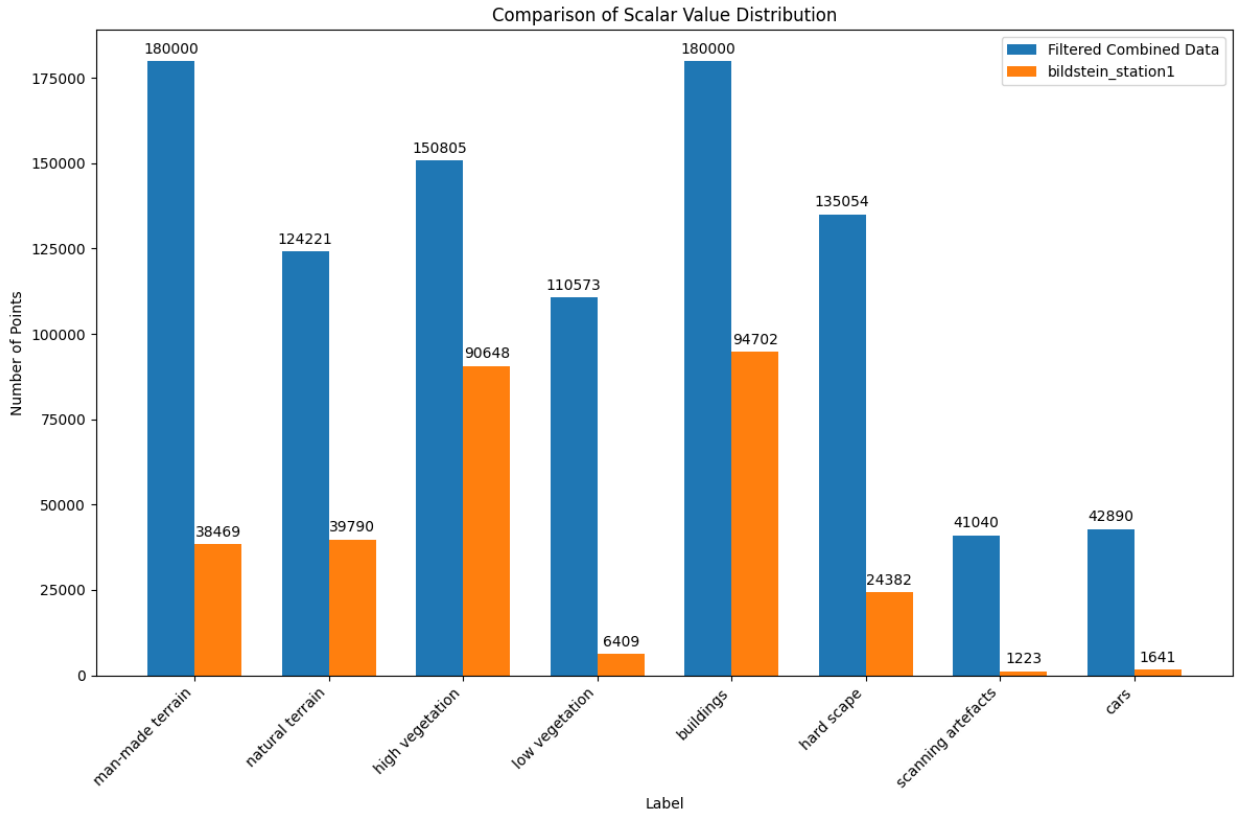
Unimplemented Features

- **Multi-Resolution Feature Extraction**: Although the system currently extracts features at a single resolution, there is potential to improve classification accuracy by implementing multi-resolution feature extraction. This would involve subsampling the point clouds and concatenating features from multiple scales.
- **Additional Classifiers**: The system focuses on Random Forest and SVM classifiers. Implementing and testing additional classifiers, such as k-Nearest Neighbors or Gradient Boosting, could provide further insights into the performance of different algorithms.

Analysis and Discussions

1. **Data Preprocessing**: the data was acquired from the semantic-8 3D labelled data set - http://semantic3d.net/ and split into training and test data. The data contained an additional label 0 for unlabelled points which needed to be removed to ensure the data was clean of unnecessary information. Additionally, the labelled datasets were concatenated with the 3D point clouds.
2. **Subsampling**: once the data was pre-processed and cleaned from unlabelled points and concatenated with the labelling data, the data was subsampled to ensure a balance between the efficiency and accuracy of the models. This involved a spatial methodology of subsampling. Other methods for subsampling include random, random % and octree, however, these methods do not account for surrounding neighbour points and would be detrimental to computing geometric. A minimum space of 0.1 was chosen for spatial subsampling as this distance ensured that sufficient points remained for the neighbourhood geometric feature calculations. The subsampling was performed across the entire training and test set using CloudCompare.
3. **Feature Extraction**: The geometric features for each point in the point cloud were computed using CloudCompare. These features include eigenvalues, omnivariance, anisotropy, planarity, sphericity, and others, which capture the local geometry around each point. A 0.5 distance for the local neighbourhood radius was chosen. Feature extraction was performed manually for each of the training and testing datasets. However, CloudCompare facilitated an efficient workflow, allowing geometric features from multiple datasets to be computed sequentially after they were loaded and subsampled.
4. **Data Concatenation**: The extracted features, along with their corresponding class labels, were applied to the test and training sets. To ensure a balance of computational efficiency and accuracy for Random Forest and SVM, the training set needed to be concatenated. This involved subsampling from each of the training sets once the geometric features were computed maintaining consistency of the local neighbourhood radius. A Python program was written with the Pandas library to loop through each of the computed training files and concatenate the first 20,000 rows containing each label.

This ensured an even distribution of the number of labels over the entire dataset. Additionally, Nan values (values in which the local neighbourhood radius for the feature calculations contained no neighbours) were removed to ensure data integrity. The total combined data across all training sets came to 964,000 points. The following table compares the number of points per label of the concatenated set are in blue vs the bildstein_station1 in orange.



Training and Evaluation

Two machine learning classifiers, Random Forest and Support Vector Machine (SVM), were trained using the preprocessed feature data. The performance of each classifier was evaluated on different test sets using the following metrics:

- **Accuracy**: measures the overall correctness of the model's predictions, considering both positive and negative classes.
- **Precision**: focuses on the accuracy of the positive predictions alone, ignoring how well the model performs on the negative class.
- **Recall**: The proportion of true positive classifications among all actual points of a particular class.
- **Intersection over Union (IoU)**: A measure of overlap between the predicted and actual class labels, computed for each class.

- **F1 Score**: The mean of precision and recall, providing a balance between the two, is especially useful when you need to account for both false positives and false negatives.
- **Support**: The number of actual occurrences of each class in the dataset, indicating how many times the true labels for each class appear.
- **Macro Average**: The average of a metric computed independently for each class without considering class imbalance, treating all classes equally.
- **Weighted Average**: The average of a metric weighted by the number of true instances (support) for each class, taking class imbalance into account.

Classification metrics for concatenated dataset Random Forest

Several tests were conducted on the Random Forest classifier to evaluate how the n_estimators parameter, which determines the number of trees in the forest, impacts the model's accuracy and speed. The table below illustrates the effect of increasing the number of trees on accuracy for the concatenated test set compared to bildstein_station5.

N_Estimators	Accuracy on Training Set	Accuracy on Testing Set
2	0.827	0.586
5	0.948	0.602
10	0.982	0.623
20	0.996	0.635
50	0.9997	0.645
100	0.99992	0.648

Increasing the number of estimators in the Random Forest classifier improves both training and testing accuracy. While training accuracy quickly approaches 100%, testing accuracy shows more gradual improvement, reaching 64.8% with 100 estimators. The marginal gains beyond 20 estimators suggest diminishing returns, indicating a potential balance between model complexity and generalization, with a slight risk of overfitting as the model becomes nearly perfect on the training data.

The following table shows the classification metrics for Random Forest with n_Estimators = 100. The concatenated training set was compared to bildstein_station5 for this example.

Class	Precision	Recall	F1-Score	IoU	Support
1.0	0.72	0.82	0.77	0.6205	65,689
2.0	0.83	0.57	0.68	0.5087	107,855
3.0	0.65	0.66	0.66	0.4868	92,127
4.0	0.20	0.24	0.22	0.1347	33,784
5.0	0.82	0.76	0.79	0.6491	136,404
6.0	0.10	0.21	0.13	0.0523	8,480
7.0	0.01	0.23	0.02	0.0158	219
8.0	0.04	0.03	0.03	0.0129	3,036
Macro Avg	0.42	0.44	0.41	0.3099	447,594
Weighted Avg	0.71	0.65	0.67	0.5229	447,594
Accuracy (Training Set)	0.99992				
Accuracy (Testing Set)	0.65				

The results show that the classifier performs reasonably well, with a testing accuracy of 65% and strong performance on classes with larger support, such as classes 1: man-mad terrain and 5: buildings, which achieved F1-scores of 0.77 and 0.79, respectively. However, the model struggles with classes that have fewer instances, particularly class 8: cars, where the F1-score is only 0.03. The high training accuracy of nearly 100% suggests overfitting, where the model perfectly fits the training data but generalizes less effectively to new data. The IoU scores further confirm this, with higher IoU values for the dominant classes but significantly lower scores for minority classes. This highlights the model's difficulty in handling imbalanced datasets, and suggests that additional strategies, such as data augmentation or balancing techniques, may be needed to improve generalization and performance across all classes.

The following table shows the classification metrics for Random Forest with n_Estimators = 20. The concatenated training set was compared to the entirety of the test sets for this example.

Dataset Name	**Testing Accuracy**
bildstein_station5	0.6482
domfountain_station2	0.6977
neugasse_station1	0.5394
sg27_station4	0.6289
sg28_station4	0.7034
untermaederbrunnen_station3	0.6058

The testing accuracy across different datasets shows variability, with accuracy ranging from 53.94% to 70.34%. The highest accuracy is observed in sg28_station4 at 70.34%, while neugasse_station1 has the lowest at 53.94%. This variation suggests that the classifier's performance is influenced by the specific characteristics of each dataset, such as point cloud density, noise levels, and class distribution. The results indicate that while the model performs well on some datasets, it struggles with others, highlighting the importance of dataset-specific factors in model performance

Classification metrics for concatenated dataset Support Vector Machine

Due to limited time and computational complexity for SMV, only one evaluation of the metrics was able to be completed. Performing evaluation found that SVM took upwards of 6 hours on the training set to perform the metrics calculations. This is in comparison to Random Forest which took less than 1 min for the entire test set. The following table shows the SMV classification metrics from the concatenated training set against bildstein_station5.

Class	Precision	Recall	F1-Score	IoU	Support
1.0	0.52	0.97	0.67	0.5095	65,563
2.0	0.95	0.29	0.44	0.2836	107,450
3.0	0.64	0.82	0.72	0.5615	90,134
4.0	0.29	0.16	0.20	0.1140	33,702
5.0	0.80	0.86	0.83	0.7071	135,909
6.0	0.11	0.11	0.11	0.0575	8,448
7.0	0.00	0.00	0.00	0.0000	209
8.0	0.00	0.00	0.00	0.0000	3,023
Macro Avg	0.41	0.40	0.37	0.2791	444,438
Weighted Avg	0.70	0.66	0.62	0.5101	444,438
Accuracy (Training Set)	**0.54**				
Accuracy (Testing Set)	**0.66**				

The comparison between the SVM and Random Forest classifiers reveals distinct performance characteristics. The SVM achieved a higher overall testing accuracy of 66%, compared to the Random Forest model 65% for bildstein_station5. However, due to limited computational time, SVM was not evaluated against all testing sets and Random Forest proved capable with a high testing accuracy of 70% for sg28_station4 in a fraction of the time. The SVM exhibited stronger precision for dominant classes like class 2, with a precision of 0.95, but struggled with recall, especially in minority classes, leading to lower IoU scores for these classes. The Random Forest, while slightly better in terms of generalization (reflected in the weighted IoU of 0.5229), also displayed difficulties with minority classes, but managed more balanced F1-scores across classes. Both models show a trend of high precision but lower recall for certain classes, suggesting a need for improved recall, particularly in underrepresented classes, to enhance overall IoU and balance in classification performance.

Visual Results

The classified point clouds were visualized by assigning colours to each class label as described in the design overview section after the class labels were predicted using Random Forest. Figure 1 shows visual results for the computed class labels and Figure 2 shows the visual results for the original class labels

Upon visual inspection, the Random Forest classifier produced clean and distinct boundaries between classes, particularly between buildings and ground. However, it had difficulties in the classification of label 8:cars has demonstrated a low precision score of 0.04.

Figure 1

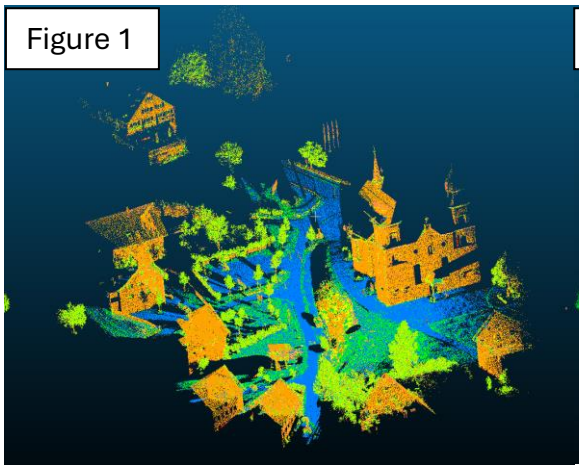
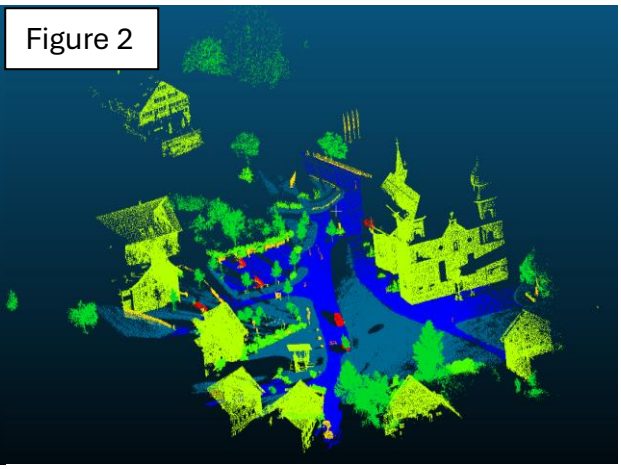


Figure 2



Discussion

The results demonstrate that non-deep machine learning techniques can effectively segment and classify 3D point clouds, particularly when using well-engineered geometric features. The Random Forest classifier's computational performance was drastically superior to the SVM, likely due to its ability to handle a larger feature space. However, SVM produced an accuracy score of 1% higher than that of Random Forest.

One challenge encountered during the project was the computational demand of processing large point clouds. While the classifiers themselves are relatively lightweight compared to deep learning models, the feature extraction and data handling processes required significant memory and processing time. This highlights a potential area for optimization in future work, particularly in the preprocessing pipeline.

Additionally, the visual inspection of classified point clouds revealed that certain classes, such as cars, were more prone to misclassification. This is likely due to the lack of support within these classes, suggesting that additional or more sophisticated features and further normalization with equal distribution of the class labels of the training set might be necessary to improve classification accuracy in these areas.

Conclusion

This project set out to explore the effectiveness of non-deep machine learning techniques, specifically Random Forest and Support Vector Machine (SVM) classifiers, for segmenting and classifying single-view 3D point clouds. Through rigorous experimentation, it was demonstrated that these non-deep learning methods can achieve satisfactory classification accuracy, particularly when leveraging well-engineered geometric features.

The Random Forest classifier, while computationally more efficient, provided a balanced performance across classes, though it struggled with minority classes such as cars. On the other hand, SVM, despite requiring significantly more computational resources, achieved a slightly higher overall accuracy. However, its limited recall and lower IoU scores for minority classes suggest that it may not generalize as effectively as Random Forest.

Key challenges included handling the computational demands of processing large datasets and addressing the class imbalance within the data. These factors underscore the importance of dataset-specific optimizations, such as better feature selection and data augmentation, which could further enhance model performance. Additionally, the reliance on single-view point clouds was a notable limitation, potentially leading to incomplete scene representation.

In conclusion, while both classifiers demonstrated potential, the Random Forest emerged as a more practical choice for this application, particularly given its balance between accuracy, computational efficiency, and ability to handle a larger feature space.

Future work could enhance the current system by integrating multi-view point clouds for a more complete scene representation, improving classification accuracy through additional context. Implementing multi-resolution feature extraction would allow the model to capture both fine and coarse geometric details, enhancing its ability to distinguish between classes. Addressing class imbalance with advanced data augmentation, synthetic data generation, or transfer learning could improve generalization, especially for minority classes. Exploring ensemble methods or boosting techniques like Gradient Boosting Machines (GBM) might further enhance performance. Additionally, automating the feature extraction and preprocessing pipelines could reduce computational demands, making the system more scalable and efficient for larger datasets. These improvements would contribute to a more robust and adaptable point cloud classification system.

User Manual

This system segments and classifies single-view 3D point clouds using Random Forest and SVM classifiers. Ensure Python 3 is installed along with the required libraries:

pip install pandas scikit-learn joblib numpy tqdm matplotlib

Data Files: The system requires 3D point cloud data files and corresponding label files in .txt format.

Processing: Run Processing.py to filter and subsample point cloud data, removing unlabelled points and balancing the dataset. The processed data will be saved in the filtered training directory. You can visualize data distribution by running the plotHist function in the same script.

Training and Testing: Use Models.py to train the Random Forest or SVM classifiers. The script reads the processed data, trains the model, and saves it in the current directory. It also evaluates the model, providing metrics such as accuracy, precision, recall, F1-score, and IoU.

Classified Point Clouds: After training, the system predicts labels for the test set and generates a classified point cloud with color-coded labels, saved as classified_points.txt in the current directory.

You can customize parameters, such as the number of estimators for Random Forest or SVM settings, directly in Models.py. Adjust the color map for classified points in the create_pointCloud function.

Ensure data files are correctly formatted and placed, and all required libraries are installed. If SVM training is slow, consider reducing the dataset size. For more details, refer to the comments in the scripts.