NPM3D

Mini challenge

Aubin Tchoï

Semantic classification of 3D point clouds

Overview of the method. The method implemented here can be divided into three parts, starting with a part of feature engineering where local descriptors are derived from a principal component analysis. The features designed at this stage are sensible to the choice of the neighborhoods with two main drawbacks: using k-nearest neighbors does not allow for consistent geometrical meaning, but the use of spherical neighborhood can be found to be impractical when the density of the point cloud is uneven. The implementation of multiscale neighborhoods on subsampled point clouds addresses these issues, the idea is borrowed from H. Thomas, J.E. Deschaud, B. Marcotegui, F. Goulette, Y. Le Gall. Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods. The features computed on the different scales are then used to train a random forest, which allows us to obtain a first coarse segmentation of the point cloud. The last step in the pipeline is a smoothing algorithm, that operates by depriving the point cloud from its ground in order to disconnect the various items in the scene, and then applies region growing to identify proximity clusters in the remaining point cloud and aggregates the votes in each cluster to create a smooth area.

1 Feature engineering

This part is an extension of the practical work on the descriptors (TP3). We define point-wise neighborhoods using a KDTree and then extract the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \in \mathbb{R}$ and normalized eigenvectors $e_1, e_2, e_3 \in \mathbb{R}^3$ of the covariance matrices of each neighborhood to build features on top of them. The features adopted are described in Table 1.

To these features we add the absolute and vertical moments, that account for point dispersion around each axis in a very straightforward way, for a total of 21 features. If we denote by $\mathcal{N}$ the neighborhood and p_0 the point the features are computed on, the absolute and vertical moments are defined as following:

- Absolute moment (for $k \in \{1, 2\}$ and $i \in \{1, 2, 3\}$): $\frac{1}{|\mathcal{N}|} |\sum_{p \in \mathcal{N}} \langle p - p_0, e_i \rangle^k|$
- Vertical moment (for $k \in \{1, 2\}$): $\frac{1}{|\mathcal{N}|} |\sum_{p \in \mathcal{N}} \langle p - p_0, e_z \rangle^k|$

Feature names in the source code	Formal definition
eigensum	$\sum_i \lambda_i$
eigen_square_sum	$\sum_i \lambda_i^2$
omnivariance	$\prod_i \lambda_i$
eigenentropy	$-\sum_i \lambda_i \log(\lambda_i)$
linearity	$1 - \frac{\lambda_2}{\lambda_1}$
planarity	$1 - \frac{\lambda_2 - \lambda_3}{\lambda_1}$
sphericity	$\frac{\lambda_3}{\lambda_1}$
curvature_change	$\frac{\lambda_3}{\sum_i \lambda_i}$
verticality	$\lvert 1 - 2\frac{\text{angle}(e_3, e_z)}{\pi} \rvert = 2\frac{\arcsin \lvert\langle e_3, e_z\rangle\rvert}{\pi}$
lin_verticality	$\lvert 1 - 2\frac{\text{angle}(e_1, e_z)}{\pi} \rvert = 2\frac{\arcsin \lvert\langle e_1, e_z\rangle\rvert}{\pi}$
horizontalityx	$\lvert 1 - 2\frac{\text{angle}(e_3, e_y)}{\pi} \rvert = 2\frac{\arcsin \lvert\langle e_3, e_y\rangle\rvert}{\pi}$
horizontalityy	$\lvert 1 - 2\frac{\text{angle}(e_3, e_x)}{\pi} \rvert = 2\frac{\arcsin \lvert\langle e_3, e_x\rangle\rvert}{\pi}$
neighborhood_size	$\lvert\mathcal{N}\rvert$

Table 1: Features used

The absolute moments describe the dispersion of the neighborhood around each of the three directions given by the eigenvectors, and the vertical moments the dispersion along the vertical axis, which plays a specific role because of the gravity and the nature of the scene. For instance, using rotation invariant features such as the planarity or the absolute moments it is hard to distinguish a wall from the ground, whereas points on the ground would have a much higher verticality than points on a wall.

The two last features in the table differentiate the x-axis from the y-axis, and were closely monitored to avoid overfitting on the training set. For instance, if the training set only contains scenes that describe alleys that are directed towards the same direction, the walls would have the same orientation most of the time. This could wrongly lead the model to learn a correlation between either the `horizontalityx` or the `horizontalityy` and the label 'building'.

2 Learning part

By training scikit-learn's RandomForestClassifier on these features without any additional parameter tuning we reach an average IoU of 35.74% on the test set.

2.1 Model choice

Various tree-based models were benchmarked on this problem: scikit-learn's RandomForestClassifier, Microsoft's LightGBM, and the library XGBoost. The models were evaluated after each step of the implementation of the method, and the best performing model on the complete method was LightGBM.

The function that we are trying to learn in this problem is highly non-smooth, which is why this method opts for tree-based models rather than deep neural networks. Not using deep learning was a deliberate choice in this mini challenge, mainly because I did not want to deal with resource limitations and wanted to explore white-box approaches specifically. In the source code you will find a notebook that contains an implementation of PointNet with the additional block used for segmentation. The model unfortunately does not run on Google Colab's free plan without exceeding either the memory limit or the time limit.

2.2 Model evaluation

To evaluate the model, the main metrics used were the Jaccard score (IoU) and the confusion matrix, which brings a rich information that highlights the unbalance between each class.

To divide the training data into training set and evaluation set, two approaches were explored: splitting the data randomly or using one of the three point clouds for evaluation. The latter option allows for enriched analysis by visualizing the classification obtained using CloudCompare, which was very beneficial when trying to explain the behavior of the method at each step of the implementation.

3 Multiscale neighborhoods

This part is a re-implementation from scratch (possibly with some details missing) of the method described in H. Thomas, J.E. Deschaud, B. Marcotegui, F. Goulette, Y. Le Gall. Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods. The method consists in computing the features with spherical neighborhoods of increasing radii on point clouds subsampled with voxels of increasing sizes. More precisely, for each scale $s \in \{0, \cdot, S-1\}$, the neighborhood radius is equal to $r_s = r_0 * \varphi^s$ and the cloud is subsampled with a grid size of r_s/ρ. The parameters used for the experiments are: $r_0 = 0.1$ m, $\rho = 5$, $\varphi = 2$, $S = 8$. The number of features is multiplied by S at the end of this part, for a total of 168 features.

This approach yields an average IoU of 52.5% on the test set. The feature importance analysis shows that the features computed on some scales are overall more discriminating than on some other scales and that there are some features that only intervene at a specific range of scales, with the example of the curvature change that is most meaningful on the biggest scale. We also observe that the neighborhood size is the most prominent feature at almost all scales and that the two features `horizontalityx` and `horizontalityy` seem to play similar roles at every scale.

4 Smoothing algorithm

The motivation behind the part that follows comes from the following observation: the segmentation obtained at the end of the previous phase performs well on regular shapes like the ground or the buildings but performs poorly on the vegetation and on cars. When looking more closely into it, we can observe that points in trees have very noisy features and are classified with various labels. The purpose of this section is to design an algorithm to smooth the labels in such areas.

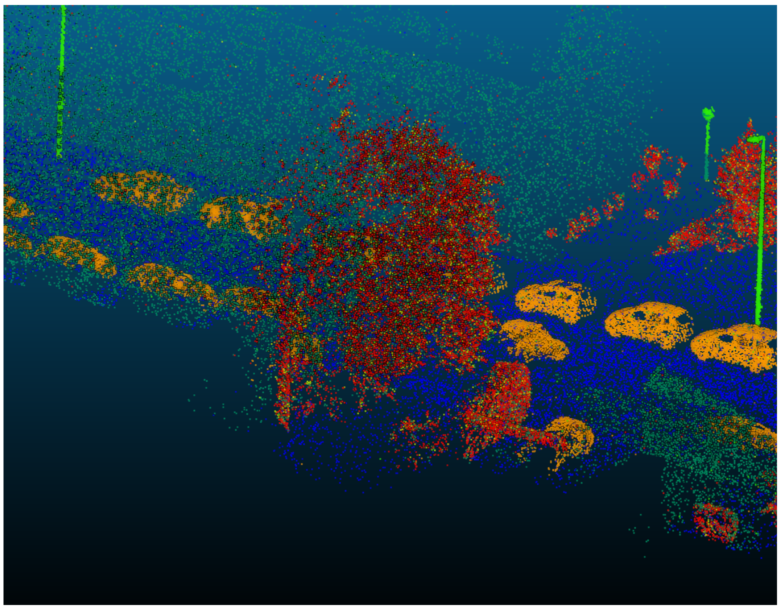

Figure 1: Classification of the validation point cloud

The part that follows is arguably specific to the type of scenes considered. It is based on the assumption that we are dealing with urban scenes, generalizes directly to most outdoor scenes but is probably not fit for indoor scenes. Even though the method loses here in versatility, one could argue that it already lost some with the use of feature dependent on absolute directions like the verticality.

4.1 Ground removal

The first step of the smoothing algorithm consists in performing a coarse segmentation of the point cloud before aggregating the labels in each cluster of the segmentation. To do so, a possible general approach would rely on the K-means algorithm. This algorithm seems highly unreliable for this particular task and requires finding a good value for k, which is specific to each point cloud. A more intuitive method would consist in identifying the connected components in the point cloud by defining a maximum distance for two points to belong to the same connected component. With an adequate value for this maximum distance we can partially retrieve each item identified as a car, a pole, a pedestrian, and so on. Since the ground links all the items in the scene, we have to remove the ground for this operation to be possible.

For this step to be fast I chose to simply rely on the previous classification to identify the ground. This problem could be seen as a 2-class classification problem and lead to more sophisticated methods, but with an accuracy of 97.7% for points on the ground the estimation seems reliable enough.

4.2 Region growing

To identify connected components, a region growing algorithm can be applied as following: seeds are selected randomly in the point cloud deprived from the ground and from each seed a connected component is built by selecting spherical neighborhoods iteratively. The maximum distance between two points of distinct connected components is thus equal to the radius of the neighborhood search, and can be chosen larger for smoother segmentations. The figures below show the results obtained with a radius of 0.2 m.

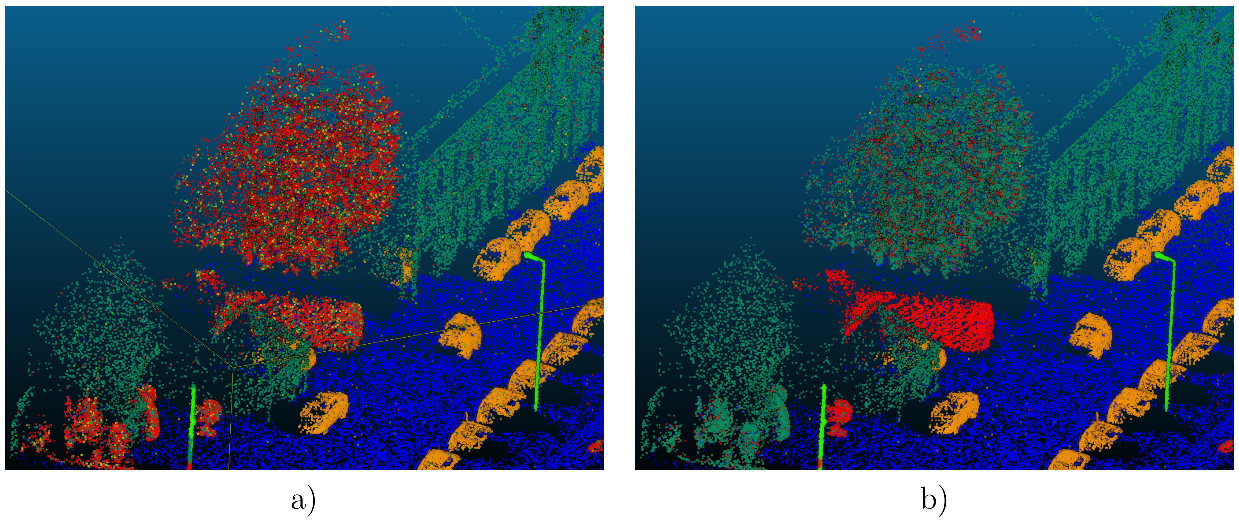

Figure 2: Classification obtained a) before smoothing b) after smoothing

The smoothing algorithm performs well on the smaller bush under the tree but seems to identify the tree as a building and still shows some disparity in the labelling of the tree. The first issue comes from the fact that the building is almost in contact with the tree and the two are therefore identified as a single connected component. The second issue is hard to address; increasing the radius of the search could solve this issue but empirically seems to be dubious as it creates many other occurrences of the first issue where various elements are blended altogether.

A region criterion was added to deal with this issue: to an existing connected component, points will only be added if they present similar features. To choose the feature that will take part in this comparison I studied the means and standard deviations of each feature for each class. To simplify the problem I aggregated the features over the scales. I focused on the specific example of vegetation blending with its surrounding to choose the features that identified best this class: the omnivariance, the planarity, the neighborhood sizes and the squared moments over the x and y axes.

In the implementation, thresholds have to be set for each feature to retain the following points: $\{p \in \mathcal{N}, \forall f \in \mathcal{F}, |f(p) - f(p_0)| < \eta_f\}$. The thresholds were initially set to be equal to the standard deviation of the feature on the class 'vegetation', and later fine-tuned with a cross validation.

In practice this method turned out to be highly sensitive to the choice of the seed and quite inconsistent. A direct improvement was to replace $f(p_0)$ with a value representative of the whole region already identified. The current implementation retains the mean of the feature over the region, and the thresholds are also adapted with a multiplication by the biased standard deviation of the feature over the region (to avoid an issue where the regions would not grow beyond 1 point the standard deviation is clipped to be higher than 1).

4.3 Label aggregation

When semantically coherent regions are identified, the labels within each region are aggregated by electing the argmax of the ponderated label frequencies. Each region is thus filled with the same label, which incentivizes the use of relatively small radii when extracting the regions. This aggregation gives a window to check the regularity of the regions extracted. We can indeed take the ponderated frequencies, compute their softmax and make sure that one of the values is much higher than the others. The weights can be learned by comparing the values of these scores on all the regions and finding the weights that would lead to the best possible prediction on all the regions.

Some cases of relatively balanced scores caught my attention: some of the scenes contain bushes or small trees that are put on top of a tree pot, which is also labelled as part of the vegetation in the training set. On these example the method would predict the label "building" for the tree pot and "vegetation" for the bush, which is an arguable choice. Unfortunately, the smoothing algorithm would on occasions either label the whole as a building, which lowers the score, or as vegetation. This example shows that intricate details on the ground truth can be hard to interpret with a semantic classification, with potential errors further propagated by the smoothing algorithm.

The method with the smoothing algorithm yields an average IoU of 75.1% on the test set. The worst performing classes are the pedestrians, who are not detected at all, and the cars, often mistaken for pedestrians. Having a dataset with a more balanced proportion of pedestrians would help, it is also possible that the method would have better performances on a point cloud sampled with a finer resolution.

4.4 Context of the experiments

The experiments were carried out on a MacBook Air laptop equipped with an Apple M1 chip and 8GB of RAM. The whole method ran for a total of 3359 seconds, with most notably 1351 seconds on the cloud subsampling and the feature computation, 422 seconds on the model training and 1368 seconds on the smoothing algorithm. It does not rely on GPU acceleration and is entirely implemented with Python with a heavy reliance on NumPy for vectorization.

5 Possible extensions

The region growing algorithm could probably be extended to recycle the subsampled point clouds in a way that would be very similar to the multiscale neighborhoods. Each subsampled point cloud would lead to different clusterizations, that could be combined to create a more stable one. A first and more direct extension of the current method could also rely on the subsample features instead of the aggregated features, or at least on an aggregation that would be smarter than a simple sum.

Another possible improvement would lie in the choice of the spheres in the multiscale neighborhoods. The current implementation follows the law: $r_s = r_0 * \varphi^s$, but other forms could be put to the test.

Many parameters can be fine-tuned in this method, the parameters r_0, ρ, φ and S of the multiscale neighborhoods, the thresholds η_f and the radius used in the region growing algorithm, the weights in the label aggregation and finally, the model parameters. Three cross validations are currently performed separately on the radius in the region growing algorithm, the thresholds η_f and on the parameters of the model but there is no cross validation that involves all the parameters at once. It would also be more beneficial to train these parameters on datasets with more diversity, as some of them can quickly overfit on the specific patterns or specificities of each dataset. Potential unexpected behaviors like the one with the tree pot could also occur, which refrains us from trying to overly fine-tune the method.

The way the label aggregation is carried out is also somewhat unsatisfying, I believe that either the labels should take part in the region growing algorithm, or that the aggregation should allow for subdivisions of the identified regions instead of dissociating the two processes.