Report Segmenting Single-view 3D Point Clouds with Machine Learning

Introduction

In this project, we aim to automate the classification and segmentation of points in 3D point clouds using the Machine Learning Classification techniques namely Random Forest, SVM and K-Nearest Neighbor. Using laser scanners detailed scenes of surfaces are captured and the information is stored in the form of points x,y,z and may also include the colour component red, green and blue. These colours are used to illustrate the label or class of a given point.

In this experiment we will leverage the properties of the surrounding area of a point, to generate classification models that will label each point in the 3D point cloud. These properties are called geometric features. The label keys are 1: man-made terrain, 2: natural terrain, 3: high vegetation, 4: low vegetation, 5: buildings, 6: hardscape, 7: scanning artefacts, 8: cars.

We test the accuracy of the models and output binaries to be plotted on CloudCompare.

Systems and Hardware

I used Cloudcompare software and Cloudcompy python libraries. The python code is run on Jupyter Notebook installed in a Cloudcompy Python environment. This allows for the manipulation of cloud binaries from within Python 3.9.7 version. My workstation is using Linux Debian Release 11 SID with 12 gig ram and core i5 processor.
Due to the limited processing power and memory available to me, my main emphasis was on sampling large data to get a good representation of the class Labels and leverage balancing in order to improve the training and scalability. I did not use the whole range of data files in training for this reason and used 3 files , bildstein_station1_xyz_intensity_rgb, bildstein_station3_xyz_intensity_rgb and untermaederbrunnen_station1_xyz_intensity_rgb.

Dataset

The preparation of the datasets proved challenging due to the limited memory on hand. My initial approach was to use chunking of the dataset and have multiple parts of the data. I used Linux scripts to split the large files and append tags to show the order of the files. This approach did not scale well as certain files were 17 gigabytes in size, and the overhead of maintaining the processed chunks created inefficiencies. I choose to use the data that would fit in the memory. This was especially necessary for the SVM model which has no incremental training ability. As such it would need to be trained using one batch.

Cleaning and Formatting

The data was well formatted and correct. The only task needed was to remove the rows with the Label class of 0, which represents unclassified. This would not aid in the process of classification and we dropped all rows. To achieve this I combined each dataset with the respective label file and iteratively searched for 0 Label rows. The data included r,g,b and intensity columns. These did not add to the geometric computation done in CloudCompare, and would not be used in the testing of the models as such these columns were dropped as well. The file size dropped by 30 % improving the challenge of memory space.

Balancing
As this is a multiclass classification problem, an issue of class balance comes up when I split the larger files. The dataset chunks often did not have a whole spectrum of classes, and if they did they were an uneven distribution across the different classes. An unbalanced dataset leads to poor training results. I chose two alternatives for correcting this problem, by using resampling. The first is undersampling which simply deletes entries from the classes which outnumber the smallest class group until there is an even number. Oversampling creates new entries in the underrepresented classes until the numbers are even. Undersample is used to make the training set even more compact.

Dataset selection has two main benefits speeding up the SVM training and reducing the size of training data. There are different methods to reduce the cardinality of a training set namely Data Geometry Analysis methods, neighbourhood analysis, evolutionary, active learning and random sampling methods. I chose to use the simpler option of Sampling. Due to the limitations of SVM which is not easily able to carry out partial fitting the alternative is using a voting approach based on multiple models.

Feature Generation
The next step was to compute the features for each point, namely Sum of eigenvalues, Omnivarianc, Eigenentropy, Anisotropy, Planarity, Linearity, PCA 1, PCA 2, PCA 3, Surface variation, Sphericity, Verticality, 1st eigenvalue, 2nd eigenvalue and 3rd eigenvalue.
My initial approach to feature generation was to use command line commands and then scripts to apply to a folder of files. This approach created wastage when saving the 9 different versions of the binary file. It also proved unreliable even when given a long time to run. In an effort to save time and computational resources I decided to balance the dataset before computing the features. Eliminating any extra rows which would be discarded if balancing would be done after feature generation.

Normalisation
I applied normalisation to the features vector. The reason for this is to make all the fields in the feature vector the same scale thus leading to a better performing model and better stability across different values. Steps are feature calculation, normalisation, SVM training set selection and RDF classification and optimization and evaluation.
Before one creates a sample the distribution of the values is important so as to know the sample represents a balanced representation of the data. The important field is the Labels column which will be the target of the Machine Learning models.
After noticing the time it takes to generate features, I decided to balance the dataset before computing the features. This will save computation to only the rows which will be used in the training stages.

To improve the performance of the models I used hyperparameter tunning with Grid Search on a sample of the data. With these best parameters, I used Stratified Cross Validation to train the model. This approach maintains the rations of the different classes in the target fill.

Results
After the training, the test datasets were prepared in a similar way to the training dataset. The same considerations were made to only consider files which would fit into the computer resources.

Random Forest	Parameters	Accuracy
GridSearch	n_estimators': 80, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 90, 'bootstrap': True	
No tunning	Default	
SVM	Parameters	Accuracy
GridSearch	'C': 100, 'gamma': 0.1, 'kernel': 'rbf'} SVC(C=100, gamma=0.1	
No tunning	Default	
KNN	Parameters	Accuracy
GridSearch	'leaf_size': 1, 'metric': 'chebyshev', 'n_neighbors': 1, 'p': 1, 'weights': 'uniform’	
No tunning	Default	

Initial trials showed high accuracy using split test parameters however when I applied the trained models to the Test dataset, the models performed poorly. This justified my approach of using hyperparameter tuning and cross-validation. The reason may be due to overfitting as the data was highly suited for the type of scene or local the training data was based on. To counter this I combined the balanced datasets into a single data frame for training.
For the purposes of graphical representation, I sampled the test data so as to have an adequate image scale after colour labelling.
Computation limits have affected the coverage I was able to apply in the evaluation.
My source code has two ipynb files, Complete and ‘Evaluation and Testing’.
The Complete file is to be run first and shows how to execute the methods in the order needed. The details are specified in the comments. After completing the execution of the code, the output of the code will be models which will be saved into the same folder.
The Evaluation and Testing code may then be run to prepare the test file for testing using the models generated previously.
Conclusion
The segmentation of Point Clouds is a task that is left to people to carry out, and these labelled sets are known as ground truths. By breaking up the task into point classification we apply classical Machine Learning Classification techniques to this problem space. Due to large file sizes scaling of data was necessary. Further to that the uneven distribution of class Labels in these datasets needed to be corrected to ensure a balanced dataset to train the

models. I used CloudCompy a Jupyter-notebook library which allowed for the manipulation of cloud binaries from within Python, although still limited to the size of the ram and processing power on the machine. Linux command line scripts proved to be untidy to scale to many operations and files.

Dependencies and Libraries

Utilities

```
import pandas as pd
import numpy as np
import os
import gc
import math
import sys
import time
import matplotlib.pyplot as plt
```

Model handling

```
import joblib
```

Balancing

```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
```

Cloudcompy

```
from cloudComPy import CloudSamplingTools
import cloudComPy as cc;
```

Sklearn

```
from sklearn.ensemble import BaggingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestRegressor as rfr
from sklearn import svm
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.utils import shuffle
from sklearn import svm
```