Wall/Clutter Classification

Introduction

This project is working on the classification of clutters and walls. Here, walls are defined as the boundary of rooms, while clutters are defined as anything else (like tables, chairs, etc.). We have already manually labeled around 500 maps, which contain about 4000 regions in total. There is no guarantee that our labeled regions are definitely ground truth, so feel free to label your own dataset :)

This document is providing an introduction about what has been done and what has not for this project. The following parts will show the readers how to find and use some already implemented tools and resource. If you have further questions about this document, feel free to contact f.zhou@caltech.edu.

I. Label The Data

1. Automatically label the data with the intuitive classifier
   The manual test is implemented in the directory '/ersp-navigation/src/ersp-navigation/room-segmentation/manual_tests/test_classify'. Using this test, you can read a .pgm file, automatically do the classification, and then write the classification result in a .json file. To complete this manual test, go to line 38 of the main.cpp file and replace the text "## SAVE THE FILE TO THIS PATH ##" with the directory you want to save the .json file. Then, you can build this manual test after setting up the sconscript file, and run the following command on terminal:
   variant-dir/debug/ersp-navigation/src/ersp-navigation/room-segmentation/manual_tests/test_pixeltools/test-pixeltools/test-classify -f 'THE PATH OF .pgm FILE'

2. Manually label the data on roboscope
   Open roboscope, load the pgm file, and label them as either wall or clutter. After applying the changes, save the result to the target directory. The result is a .json file with the same structure as the automatically generated ones. Currently, we have a labeled dataset of 501 maps (out of 3k_pgms) in the folder 'dataset_labeled'. Once again, there is no guarantee that those labeled regions are definitely ground truth or even clean enough. We can only say our current classifiers can work on this dataset.

3. How to read the .json file
   The files in the folder 'dataset_labeled' are not very readable to human beings, as we have updated those files with python to fix some problems and the order of the lists is messed up. However, it does not matter for python to read those .json files. For training, only the "train_clutter_poly" data are used. Please follow the test.py file or other Python file to parse the .json file if needed. Theoretically, it is possible to reconstruct all the regions from .json file, but we do not have the tool to do it now.

II. Classifiers

1. Installation of toolkits
   We currently test the classifier on Python2. You may want to install sklearn and xgboost toolkits before running the current classifier. Following those instructions to install them:
   http://scikit-learn.org/stable/install.html
   http://xgboost.readthedocs.io/en/latest/build.html

2. Features
   We gathered some features from the regions and use them as the input to the classifier (they are already in the .json file).

id	Feature	id	Feature
0	area	12	is_free
1	xmin	13	is_interior
2	xmax	14	compactness
3	ymin	15	long_line_length
4	ymax	16	n_long_lines
5	centroid_x	17	n_free_pixels
6	centroid_y	18	n_unexplored_pixels
7	majorAxisLength	19	n_neighbors
8	minorAxisLength	20	ratio_of_simplification
9	orientation	21	n_doors
10	eccentricity	22	total_door_length
11	solidity		

   The computation of most features and be found in the following .cpp files:
   ersp-navigation/src/ersp-navigation/room-segmentation/regionWshed.cpp

3. Classifiers
   We have tried 3 classifiers in Python: SVM (with radial basis function kernel), decision tree and XGBoost. To test the classifier, just move the .json file from 'data_labeled' to either 'train' (for training) or 'test' (for testing) and run the Python program in the same directory.

   For SVM, the corresponding Python file is 'clf_DT.py'. There are mainly two parameters: C and gamma. They are for the misclassification cost and kernel coefficient, respectively. The 'rbf' kernel is used here. Feel free to try 'linear' or 'poly' kernel if you want, but my experience is that other kernels are much much slower than 'rbf'. I gave some tentative parameters for the current version based on some previous training and testing, which can give a relatively smaller cross validation error (same for other classifiers).

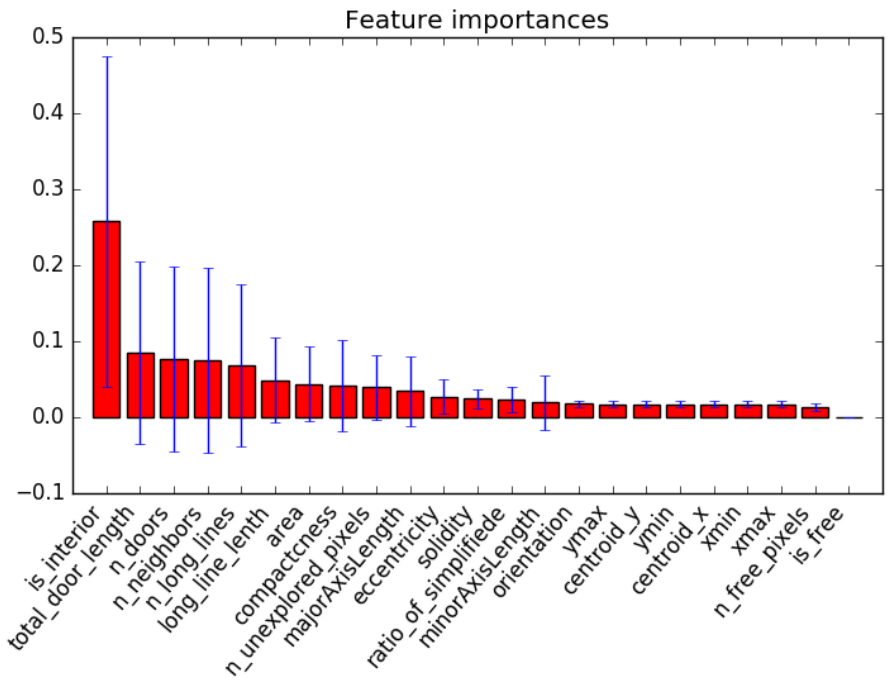
   For decision tree, we only tune the parameter 'max depth'. We use the gini index as the criteria for tree splitting in this version. Based on my experience, entropy and gini index do not have too much difference in their performance. It appears that the optimal max depth for the decision tree is around 4 and we do not have too much overfitting in this model as it is still a very simple one.

For XGBoost, the parameter space has a very high dimension, and we only tuned 3 parameters here: gamma, eta and max depth. Their meanings can be found in:
https://github.com/dmlc/xgboost/blob/master/doc/parameter.md
This program will also give a list of all the mistakes made for the testing dataset in the format of filename/region id/region type. There might still be some overfitting, so tuning more parameters to avoid overfitting might be helpful.

III. Results

1. Feature importance

Calculated from ‘feature_importance.py’.



It might be a good idea to use this graph to reduce some unimportant features.

2. Performance for Different Algorithms:

Testing error rate (cross validation error rate)

Case id	# Training/# Testing	Intuitive Classifier	SVM	Decision Tree	XGBoost
1	3301/890	10.22%	10% (8.91%)	7.42% (10.66%)	7.79% (7.64%)
2	3547/644	13.82%	12.73%(8.43%)	11.80% (9.44%)	12.88%(7.58%)
3	3511/680	13.38%	11.76% (8.69%)	11.03% (9.40%)	10.14%(7.32%)

IV. Conclusions

We believe that the current classifier has been able to classify the obstacles with a relatively lower error rate, compared to the intuitive classifier. However, there is still some gap between the testing error and the cross validation error rate. In other words, overfitting might still exist. For features, it appears that the context information is very important, and then are the shape information. The coordinates information is not very useful here. In practice, we might still need more context information to further improve the performance.