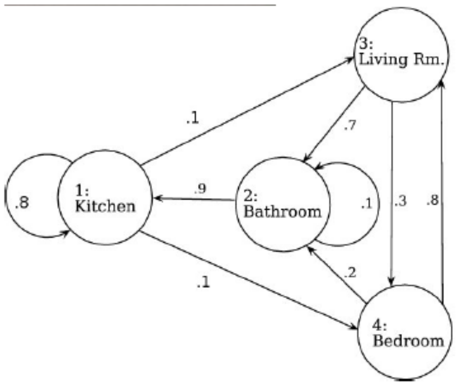
CS 4758/6758 Robot Learning: Homework 2

Due February 23 at 5:00 PM

1 Markov Chains (25 points)



A ground robot is let loose in an apartment with four rooms and begins to randomly wander. After each minute passes, there is a chance it moves to a different room (assume the robot doesn't change rooms more than once a minute). So, for instance, if the robot starts in the kitchen, one minute later there is a .1 probability it moves to the living room, a .1 probability it goes to the bedroom, and .8 chance it remains in the kitchen. Using the information given in in the figue above, answer the following:

A) If the robot starts at t = 0 in the kitchen, what is the probability it is in the bedroom at t = 3? (5 points)

B) Suppose instead the robot begins at time t = 0 in a random room given by the probability distribution p0 . (p0 is a four-dimensional vector; the first element of p_0 is the probability of starting in the first room, etc.). Then similar distributions $p_1, p_2, ...$ are defined for all subsequent t. Give a general expression for p_t . (Hint: Use 4x4 transition matrices at each time.) (5 points)

C) Evaluate $lim_{t->\infty} p_t$. Does it depend on p_0 ? (5 points)

Now let us suppose that the robot gets wet and useless in the bathroom, and cannot return back. So let us modify our transition matrix. (Let us number the rooms such: kitchen:0, livingroom:1, bed:2, bathroom:3.)

$$\begin{bmatrix} .8 & 0 & 0 & 0 \\ .1 & 0 & .8 & 0 \\ .1 & .3 & 0 & 0 \\ 0 & .7 & .2 & 1 \end{bmatrix}$$

D) Starting from kitchen, what is the probability that the robot will stay useful for 2 mins? (5 points)

E) What would $lim_{t->\infty}p$ be for the rooms? (5 points)

2 Hands-on: Classifying Point Clouds (75 points)

In this exercise you will produce a classifier that can recognize objects from their Kinect point clouds. No color information or vision features are used–the classifier will rely on the object geometry alone. We will provide you with sets of training and test data, as well as some example code for working with PCL. For simplicity the data only includes two kinds of objects: scissors and staplers. These were captured at different angles and lighting conditions. All background pixels not belonging to the objects in question have been segmented out of the set. The objects in the test set are different from the objects in the training set–three staplers and three pairs of scissors were used for the training set, and a different pair of scissors and stapler were used for the test set. So the classifier will have to generalize to be successful. Note that while this task is pretty phony, the techniques used herein are not; you may wish to adapt them for your project.

Part 0 – Installation

The first thing you need to do is setup PCL on your computer. Fortunately this is easier than installing ROS.

1. To get PCL itself, go to pointclouds.org/downloads/, then choose your operating system under "Using prebuilt binaries". Follow the instructions from there.
2. To compile new code with PCL you also need CMake. On Linux, you can just grab it from a repo (e.g. sudo apt-get install cmake); there are Windows and Mac installers available from cmake.org.
3. Finally, if you don't have one yet, you need a C++ compiler. On Windows install Visual C++ (the express version is free), for Macs Xcode, and g++ on Linux (sudo apt-get install build-essential).
4. To make sure everything works, try building the starter code. On Linux or Mac open up a terminal in the folder where you extracted the starter files and type

   ```
   cmake .
   make
   ```

   For Windows you will need to use the CMake GUI

Part 1 – Calculating Local Features

(a) The sample code includes the program showPcd.cpp. Open up one of the point clouds with it by typing ./showPcd <filename.pcd>. Take a screenshot and turn it in. (10 points)

(b) We've also supplied the program calcFeatures.cpp. This program will compute surface normals for a point cloud. Invoke it with the command ./calcFeatures <scale_factor> <filename.pcd> <anything you want>. You need to choose a scale factor when estimating the normals. The scale factor should be between a number between .1 and 10 (in cm); you may want to try .1 and 10 to see what a bad set of normals looks like. Pick an appropriate value for the scale factor, and report it here. Also turn in another screenshot. (10 points)

As you might surmise from the name, calcFeatures also computes FPFH features for each point in the cloud and writes them to a file with the extension .features. Before continuing, you need to compute these features for all of the provided point clouds. Be sure to use the scale factor you chose in part (b). (You may want to edit calcFeatures.cpp or write a shell script / batch file. Also, if you leave out the third argument when running calcFeatures, you can prevent the viewer window from launching.)

Part 2 – Feature Aggregation

You've now managed to transform a point cloud into a cloud of local shape descriptors. But the classifier we plan to use–logistic regression–works on feature vectors, not feature clouds. So we need to summarize the FPFH features into one vector per example. At a high level, the approach we will take is:

1. Cluster all the features in the training set
2. Use the cluster centroids as a 'codebook' that contains representatives for all the training features
3. Measure the similarity between each feature and those in the codebook.
4. For each example, average those similarities to produce a single vector.

We will address (and you will implement) each of these steps in turn.

(c) Clustering takes a set of data and divides it into groups of nearby points. There are many reasons why you might want to cluster data; in this case we are using it to find a small set of points that will serve as representatives for the hundreds of thousands in our training set. Specifically, we will use the centroids (centers) of each cluster as these representatives; the hope is that each training point will be similar to at least one of them. The exact algorithm you should use is called k-means. Do not the write the code for it yourself–use an existing implementation. For example MATLAB has a function 'kmeans', Python has a 'kmeans' function in its SciPy library, etc. You have to choose the number of clusters (k) you want ahead of time, so pick 128. The exact steps you should take are:

1. Pool all the local *training* features into one set. Measure the mean μ_i and standard deviation σ_i for each of the 33 feature dimensions.
2. Randomly select 1/10th of the training features. You should only cluster this subset–using the entire set of features might take hours.
3. Normalize the features by subtracting the mean and dividing by the standard deviation. That is, replace the ith entry of feature x with $\frac{1}{\sigma_i}(x_i - \mu_i)$. This scales the features so that all the dimensions have a variance of 1. In general, normalization is a good idea when using geometrically-motivated algorithms (not only k-means but also k-NN, SVM, etc.).
4. Run k-means on the transformed features, with the number of clusters k = 128. If you need to specify a distance metric use the Euclidean distance.
5. Save μ, σ and the resulting cluster centroids for the next steps.
6. Turn in the code you used for this part. (10 points)

(d) Now, having found a codebook, we can generate vectors for the classifier. Map each point cloud to a 128-dimensional vector. Each of its entries should be the average distance between the normalized FPFH features (from that point cloud) and one of the centroids. Attach the code you used for this step. (10 points)

(e) Why did we normalize the training data before clustering? (Hint: What happens if one of the dimensions has a much bigger variance than the others?) (5 points)

(f) Conceptually, part (d) could be described as encoding each local feature as the distance between it and the codebook entries, then averaging the encoded features over each point cloud. What is the advantage of the encoding step? Why not just average the original features? (5 points)

Part 3 – Classification

(g) Using the preceding, distill all of the training point clouds into a set of feature vectors, and do likewise for the test set. Make matching sets of labels using 1 for staplers and 0 for scissors or vice versa (this should be easy!). Train a logistic regression classifier on your training set and evaluate it on the test set. You can use 'mnrfit' in MATLAB, 'logistic_regression' in Octave, 'mlogit' in R, or anything else you prefer. Report the classifier's error rate (the fraction of the test set it misclassified). Also name the implementation of logistic regression you used. (25 points)

Hopefully, you now can teach a robot the difference between scissors and staplers.