MSc Human and Biological Robotics

Author:
Edward McLaughlin

Lecturer:
Dr. Aldo *Faisal*

Machine Learning and Neural Computation: London rent prices

Imperial College London

Department of Bioengineering

April 9, 2018

1 Question 1: Raw Data

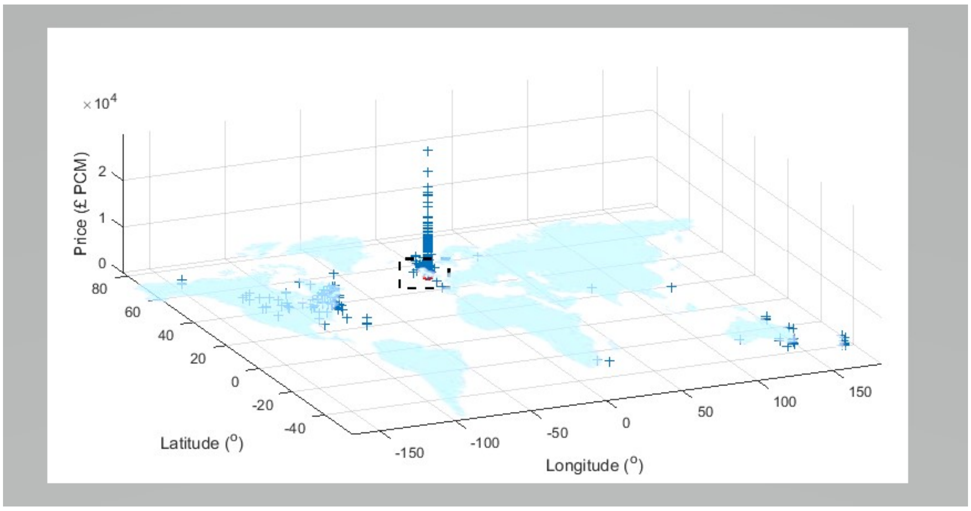


Figure 1.1: Plot of raw data with world map superimposed onto image.

Figure 1.1 shows the raw data for the property prices given with a world map superimposed onto the data with reasonable accuracy. Many properties do not lie in London. As a result, the raw data had to be filtered. The data was filtered in two ways - by location and by price. London was taken to be between 51.30^o and 51.69^o in latitude and between -0.51^o and 0.20^o in longitude. Reasonable house prices were taken as £800-3500/PCM since prices outside of this range could be mistakes from the automated data collection system. The mean rent price of the remaining properties is £1,577 PCM. This filtering is done using the function 'DataFilter' as seen in the Appendix.

2 Question 2: Training Regressor

The training regressor was designed to work in three steps, first to set out Gaussian Distributions (GDs) in a grid and secondly to place a GD in the middle of London, encompassing all properties and thirdly to perform a k-mean regression for optimise the locations of the GD for modelling the data.

The initial grid is laid out by selecting the number of segments to split London up in to. The number of longitudinal and latitudinal divisions is then calculated to create the desired grid within the confines of London as defined in the DataFilter function. Once the grid has been defined, GDs are placed at the centre of each box as well as on the intersection of the gridlines (see Figure 2.1 (left)). Next, all of the GDs which don't encompass more than x properties (x was taken as 2000/gridboxes through trial and error) within half the distance between itself and the adjacent GDs are deleted. This results in a grid of GDs which lay only within London (see Figure 2.1 (right)). Placing a grid in this manner means that increasing the number of grid boxes will not increase the area which the GDs cover but will increase the density of GDs within London.

One large GD to encompass all of London from the weighted mean of the properties' positions was superimposed on the clusters to add an estimate of the overall trend of house prices in London on a macro-scale larger than the cluster grid.

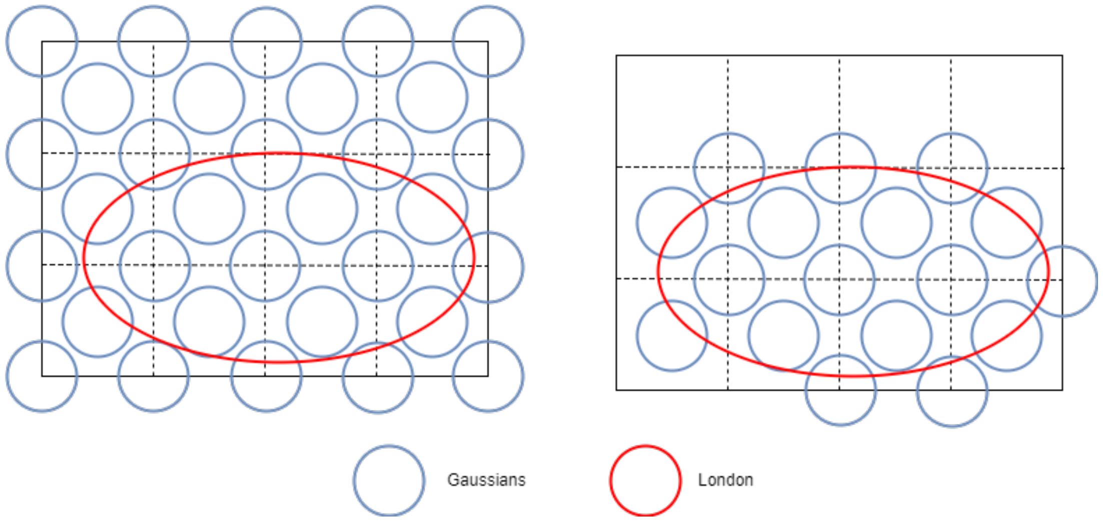


Figure 2.1: Layout design of the initial Gaussian grid.

Finally, a k-means regression is performed on the longitudinal and latitudinal coordinates of the properties to shift the GDs' mean position to optimal positions. The mean values of the GDs were calculated by applying normalised property prices as weights to the property coordinates. Implementing this weighted-mean position resulted in the mean position of the GDs to be affected by both the density of properties as well as their prices. This is necessary as we want the peaks to be biased towards the highest property prices, not necessarily the areas with the highest density of properties.

The gaussian basis functions are then given by $\phi = e^{-(x-\mu)\Sigma^{-1}(x-\mu)^T}$ where μ and Σ are the gaussian mean and covariance matrix respectively and x is the property positions. The weights, w for the basis functions are thus given by $w = \Phi^{\dagger}P$, where Φ is comprised of the individual basis functions, ϕ, and P is the vector of property prices.

3 Question 3: Test Regressor

Table 1: Algorithm results using 5 folds cross validation.

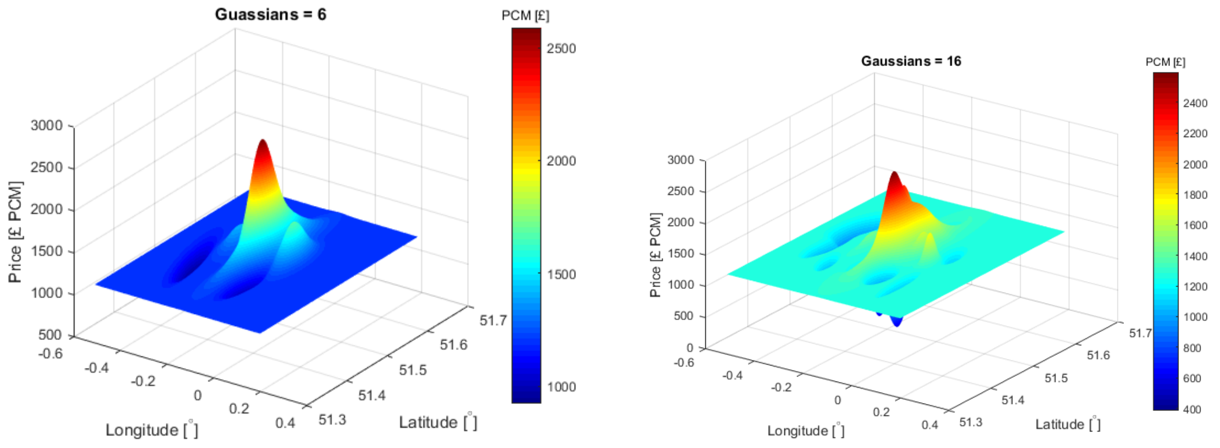
Grid Boxes	Gaussians	Max. Test RMSE (£)	StdDev Test RMSE (£)	Av. (Test-Train) RMSE (% Diff.)	Time Elapsed (s)
4	6	452	6.1	0.34	73
6	8	454	11.8	-0.79	249
8	11	471	9.1	0.39	566
9	12	450	7,1	-0.86	830
12	16	472	13.7	-1.03	1132

Table 1 shows the results for a five fold cross-correlation for various sized grids. Similar tables were produced for other numbers of folds and conveyed similar results. The highest Root Mean Squared Error (RMSE) was chosen as the most conservative result from the simulation. It can be seen that increasing the number of GDs makes a negligible difference to performance, with a RMSE for the test data at around £450 PCM - around a third of the mean rent price of the *filtered properties*. It is difficult to achieve better results than this due to the larger variation in prices compared with position and density of properties. That being said the standard deviation of the RSMEs across the 5 folds increases with the increasing number of GDs and the simulation time is considerably longer.

The difference between the RMSE for the test and training data is also given in Table 1. A high correlation in these values signifies a solution which generalises well i.e. the fit seen on the training data is a good predictor of the fit that will be seen on other sets of data.

Since increasing the number of GDs did not have a large positive effect on the results and was computationally expensive, four grid boxes were used to predict the price in the *faux* home tube station.

4 Question 4: Heat Map



(a) (b)

Figure 4.1: Heat maps for (a) 6 and (b) 16 Gaussians.

The heat map axes had to be adjusted to consider London only. The heat map with fewer GDs looks a lot smoother and continuous. This is because as more GDs are added, the clustering makes them narrower, reducing their σ, as they centre on areas with lots of properties. The areas with few or no data points remain flat, close to the mean property price. The result of this is tall, slender GDs. There is certainly a relationship between the resolution of the meshgrid for the plot and the variance of the GDs which dictates the smoothness of the plot.

5 Question 5: Home Tube Stop

The predicted price for the home tube stop (Kings Cross Thames Link - [51.5308, -0.12044]) is £1,199 PCM. The prices for properties at this location in the given data set are as follows: £1,262, £1,248, £5,408, £1,240, £1,262, £3,312 and £1,196 PCM. Thus this estimate slightly undervalues the London housing market. This might be preferable to over-fitting a solution to predict the high prices with good accuracy which would require many more GDs with much larger values for w - a solution which might not generalise well. For general interest, the rent prices of the first five single room properties given on a web search are £1,408, £2,275, £1,473, £1,690 and £1,560 PCM. Evidently, there has been inflation since the dataset was taken.

Appendix

Main Script

```
clc
clear all
close all

tic

% Load properties data and determine number of test data points.
load('london.mat')

% Filter the Data, allowing only realistically priced properties in London
% to be considered
[Prices] = DataFilter(Prices, Tube);

% Locate personalised tube Stop
[TubeStop, TubeCoor] = personalisedTube(1092693);

% Define parameters for clustering and cross validation
testFolds = 6;
gridBoxes = 6;

% Create test and train data by taking some data
[paramloc] = TestTrainData(Prices.locationGood, Prices.rentGood, gridBoxes);

%Cross Validation
for i = 1:testFolds-1

    %% Train the algorythm
    [param] = trainRegressor(paramloc.trainlocation{i}(:,:), paramloc.trainrent{i}(:,:));

    % Training RMSE
    [valTrain] = testRegressor(paramloc.trainlocation{i}(:,:), param);
    RMSETrain(i) = sqrt(mean((valTrain - paramloc.trainrent{i}(:,1)).^2))

    %% Test results
    [val] = testRegressor(paramloc.testlocation{i}(:,:), param);

    %Calculate RMSE and update paramGood with params giving lowest
    %RMSE
    RMSETest(i) = sqrt(mean((val - paramloc.testrent{i}(:,1)).^2))
    [RMSEmax, RMSEindex] = max(RMSETest);
    if RMSEindex == length(RMSETest)
        paramGood = [];
        paramGood = param;
    end

end
toc
% Heat map
 heatmapRent(@testRegressor, paramGood)
```

Function: DataFilter

```
function [Prices] = DataFilter(Prices, Tube)

% Filter the locations by longitude and latitude of London
Prices_location = Prices.location(( 51.30 < Prices.location(:,1) & Prices.location(:,1) < 51.69 &...
                    -0.510 < Prices.location(:,2) & Prices.location(:,2) < 0.2),:);
Prices_rent = Prices.rent(( 51.30 < Prices.location(:,1) & Prices.location(:,1) < 51.69 &...
                    -0.510 < Prices.location(:,2) & Prices.location(:,2) < 0.2),:);

% Filter out the extreme prices to remove outliers
muRent = mean(Prices_rent);
stdRent = std(Prices_rent);
for i=1:1:length(Prices_location(:,1))
    if abs((Prices_rent(i,1)) > 3500) || (abs(Prices_rent(i,1)) < 800)
        Prices_location(i,1) = 0;
    end
end
Prices.locationGood = Prices_location((Prices_location(:,1) > 0),:);
Prices.rentGood = Prices_rent((Prices_location(:,1) > 0),:);

% Plot properties and tube map
figure(1)
plot3(Prices.locationGood(:,2),Prices.locationGood(:,1),Prices.rentGood(:,1),'bo')
hold on
tubeheight(1:length(Tube.location(:,1)),1:length(Tube.location(:,1))) = 1;
plot3(Tube.location(:,2), Tube.location(:,1), tubeheight(:,:), 'ko')
grid on
xlabel('Longitude')
ylabel('Latitude')
zlabel('Price')

end
```

Function: TestTrainData

```
function [param] = TestTrainData(in, out, gridBoxes)
testFolds = 6;

% Find location of furthest out properties
minLat = min(in(:,1));
maxLat = max(in(:,1));
minLong = min(in(:,2));
maxLong = max(in(:,2));

% Create a grid based on the number of gridboxes specified
xy = divisors(gridBoxes);
if mod(sqrt(gridBoxes),1) == 0 %If a square number
        XY = length(xy)/2 + 0.5;
    elseif mod(length(xy),2) == 0 %If number of divisors is even
        XY = length(xy)/2;
    else
        XY = length(xy)/2 - 0.5; %If number of divisors is odd
end
nx = xy(XY);
ny = gridBoxes/nx;

% Create a spread of Gaussians based on the grid
longSplit = linspace(minLong,maxLong,nx);
latSplit = linspace(minLat,maxLat,ny);
n = 1;
m = 1;
% Place a Gaussian on the intersection of each grid line
while m <= nx
    k = 1;
    while k <= ny
            muOut.mu(1,n) = latSplit(k);
            muOut.mu(2,n) = longSplit(m);
            n = n + 1;
        k = k + 1;
    end
    m = m + 1;
end
% Place a Gaussian in the middle of eahc grid box
m = 1;
while m <= nx-1
    k = 1;
    while k <= ny-1
```

```matlab
        muOut.mu(1,n) = (latSplit(k+1)+latSplit(k))/2;
        muOut.mu(2,n) = (longSplit(m+1)+longSplit(m))/2;
        k = k + 1;
        n = n + 1;
    end
    m = m + 1;
end

%Calculate the distant from each property to each gaussian
for i = 1:length(muOut.mu)
    Dist2Gaussian(:,i) = sqrt((in(:,1)-muOut.mu(1,i)).^2 + ((in(:,2)-muOut.mu(2,i)).^2));
end

% Find the closest gaussian to each property
for i = 1:length(in(:,1))
    [minDist(i), minIndex(i)] = min(Dist2Gaussian(i,:));
end

% Allocate properties to each gaussian based on the minumum distances from
% properties to gaussians
for i = 1:length(muOut.mu)
    GaussiansProps = zeros(length(in(minIndex(:) == i ,1)),2);
    GaussiansProps_Rent = zeros(length(in(minIndex(:) == i ,1)),1);
    GaussiansProps(:,1) = in(minIndex(:) == i ,2);
    GaussiansProps(:,2) = in(minIndex(:) == i ,1);
    GaussiansProps_Rent(:) = out(minIndex(:) == i);
    muOut.GaussProps{i}(:,:) = zeros(length(GaussiansProps(:,1)),3);
    muOut.GaussProps{i}(:,:) = [GaussiansProps(:,2),GaussiansProps(:,1),GaussiansProps_Rent(:)];
end

% Split the properties into training and test data
for i = 1:(testFolds-1)
    param.testlocation{i} = [];
    param.testrent{i} = [];
    param.trainlocation{i} = [];
    param.trainrent{i} = [];
    for ii = 1:length(muOut.mu)
        randomIndex = randperm(length(muOut.GaussProps{ii}(:,1)));
        lengthA = round((length(muOut.GaussProps{ii}(:,1))/testFolds));
        % If loop to take care of errors for small value of lengthA
        if lengthA == 1
            dummyVarA = randomIndex(1);
            param.testlocation{i}(1,1:2) = muOut.GaussProps{ii}(
               dummyVarA,1:2);
            param.testrent{i}(1,1) = muOut.GaussProps{ii}(dummyVarA
               ,3);
            dummyVarB = zeros(1,length(randomIndex-lengthA));
            dummyVarB = randomIndex(2:i-1);
            lengthB = length(muOut.GaussProps{ii}(dummyVarB,1));
            param.trainlocation{i}(end+1:end+lengthB,1:2) = muOut.
               GaussProps{ii}(dummyVarB,1:2);
            param.trainrent{i}(end+1:end+lengthB,1) = muOut.
               GaussProps{ii}(dummyVarB,3);
        elseif lengthA == 0
            dummyVarB = 1:length(muOut.GaussProps{ii}(:,1));
            dummyVarB(randomIndex(((i-1)*lengthA+1):i*lengthA)) =
               [];
            lengthB = length(muOut.GaussProps{ii}(dummyVarB,1));
            param.trainlocation{i}(end+1:end+lengthB,1:2) = muOut.
               GaussProps{ii}(dummyVarB,1:2);
            param.trainrent{i}(end+1:end+lengthB,1) = muOut.
               GaussProps{ii}(dummyVarB,3);
        else
            dummyVarA = randomIndex(((i-1)*lengthA+1):i*lengthA);
            param.testlocation{i}(end+1:end+lengthA,1:2) = muOut.
               GaussProps{ii}(dummyVarA,1:2);
            param.testrent{i}(end+1:end+lengthA,1) = muOut.
               GaussProps{ii}(dummyVarA,3);
            dummyVarB = 1:length(muOut.GaussProps{ii}(:,1));
            dummyVarB(randomIndex(((i-1)*lengthA+1):i*lengthA)) =
               [];
            lengthB = length(muOut.GaussProps{ii}(dummyVarB,1));
            param.trainlocation{i}(end+1:end+lengthB,1:2) = muOut.
               GaussProps{ii}(dummyVarB,1:2);
            param.trainrent{i}(end+1:end+lengthB,1) = muOut.
               GaussProps{ii}(dummyVarB,3);
        end
    end
end

end
```