Machine Learning and Neural Computation Coursework I

Peter Sarvari - 18/11/2017

00987075

Ps5714@ic.ac.uk

MEng Bioengineering

Introduction

General introduction

Predicting one-bedroom monthly house rents from location is not an easy task: accurate prediction is not expected, since there are many factors contributing to the price other than location and number of rooms. For example, furniture, condition of the building, facilities within the building (e.g. reception, gym) to mention a few. Hence, I decided to make some assumptions to exclude luxurious houses (price>2000 pounds per months) and also houses, which seem too well-priced to be true (<300 pounds per months).

The task is to build a regression model to predict the prices according to location. A linear model does not seem appropriate, since we do not expect housing prices to increase with increasing latitude/longitude. I implemented a preliminary analysis to understand how important location is in house rents: for both latitude and longitude, distance from its respective mean has been calculated. The correlation of the price is -0.4501 and -0.4316 with respect to the latitude and longitude mean distances. This is what we expect: as we move away from the centre (mean), the price seems to drop. This result is good enough to put some effort into building a price predictor.

Introduction to methods

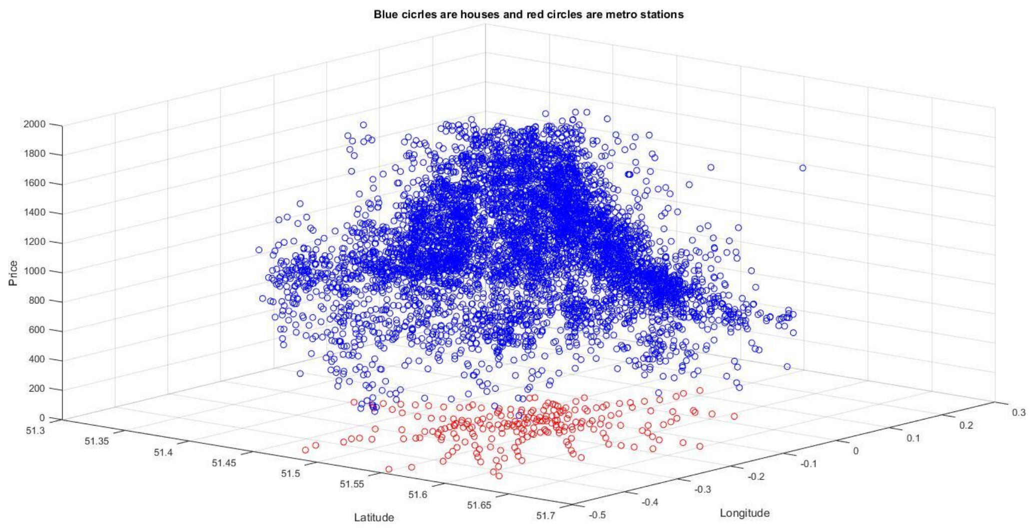
With regards the base functions of my predictor model, I decided on the not normalized Gaussian functions with different means and uniform covariances. The reason is the following: people value flats more that are close to some landmark, e.g. museums (Trafalgar square and South Kensington area), workplaces (Bank and Canary Wharf Area) or entertainment (Soho, Camden). As long as the flat is relatively close to these places, we expect the price to drop with the Euclidean distance. When the flat is far, however, the distance from a far landmark does not affect its price (people would rather go to the local pub on a Friday night than travel one hour to Soho). This qualitative relationship is very well modelled by the negative exponential in the Gaussian. The maximum value is expected when we are closest to a landmark, so it makes sense to use the landmark locations as the means of the Gaussians. Then, as we move along, the value associated with the proximity of the landmark drops exponentially, until the landmark becomes too far and the distance to it does not matter anymore.

The first issue is that we do not know the distance after which the proximity of the landmark does not matter anymore, so we should aim for learning it from the data (from here on we will refer to this distance as landmark limiting distance, LLD). It is controlled it by the sigma parameter inside the argument of the exponential: if the “sigma is big”, the LLD is big, if the “sigma is small”, the LLD is small. The reason I used the parenthesis is that sigma is a 2-by-2 matrix and I have to define what I mean by big and small in that case. First of all, I only consider diagonal matrices of the form $\begin{pmatrix} a & 0 \\ 0 & a \end{pmatrix}$. This is simply because, as I mentioned above, we expect the price to drop with the Euclidean distance from the landmark. According to the definition of the 2D Gaussian, if we use this sigma (note that the inverse of sigma is inside the exponential), given a row vector of latitude and longitude of a house, $(x \quad y)$, the resulting value inside the exponential will be: $\frac{1}{a} * (x - c_1)^2 + \frac{1}{a} * (y - c_2)^2 = \frac{1}{a} * dist\big((x, y), (c_1, c_2)\big)^2$, where c_1 and c_2 are the latitude and longitude coordinates, where the landmark is centred and $dist$ is the normal Euclidean distance between two points in the 2D space.

The second issue is that we do not know the landmarks. However, we know that (in London) there is usually a tube station next to every landmark and also in the centre of suburban areas. Since we have been given the coordinates of London tube stations, we will use them as the landmarks.

Methods

First of all, let me visualize the data we have:



This confirms three things: the price indeed seems to drop as we move away from the centre, in fact, in a nonlinear manner. The variance of the data is big, as we expected, since location is not the only factor determining the price of the house. Third, we notice that the tube stations cover well the locations corresponding to the houses in our dataset, which supports the decision to use them as the centre of the Gaussian base functions.

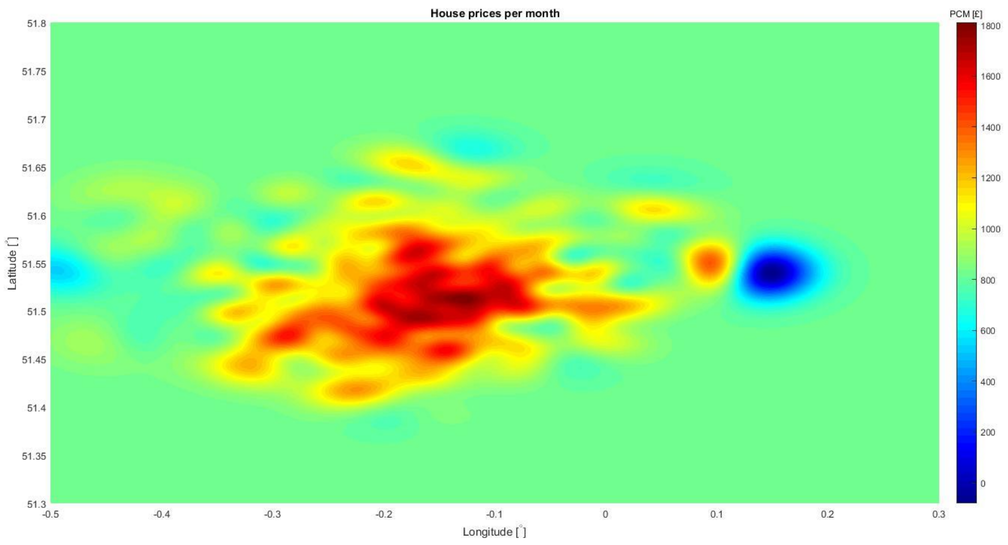
Now, we have to learn weights from the data. These weights tell us how important each landmark (tube station) is. Also, we need to learn hyper parameters from the data: such parameters are a in the covariance matrix (will be referred to as covariance scale from now on) in the Gaussian base functions (see above) and lambda, the regularization parameter. To this end we use equation (3.28) from the book by Bishop to find the weights, and use 10-fold cross-validation with initial shuffle to find the best value of the hyper parameters. Note that in each iteration, we divide the data into 3 portions (train, cross-validation and test) and the best combination of the hyper parameters is defined to be the one that on average results in the smallest mean squared error in the predictions for the cross-validation sets. The results are the average of the mean squared error in the predictions for the test sets. We also take care that throughout the iterations, the same fold is never chosen for cross-validation set one and test set the other time. This is to assure that the data used to tune the hyper parameters are always distinct to the data used to evaluate the model. This is a method that was developed by me and makes the result less dependent on the dataset than the methods used by most people (e.g. http://cs231n.github.io/neural-networks-3/#hyper) since there are different validation and test sets. To justify the regularization, we have to make sure that the expected value for the weights are zero (Gaussian prior on the weighs). So, ideally, we would need to normalize the prices (such that the average price is zero meaning that some weights would need to be negative). In this case, we skipped this step. The result is detailed below. Since according to the specifications, the testRegressor.m script cannot have more than two arguments, I chose lambda = 0 and covariance scale = 0.1 in my final, submitted script. Please note that extra care has been taken to vectorise the Matlab code and get rid of for loops in order to increase performance!

Results

Tested lambdas: 0, 10^{-7}, 10^{-6}, and 10^{-5}
Tested covariance scales: 0.05, 0.1, and 0.25

Best lambda	Best covariance scale	Test MSE	Cross-validation MSE	Train MSE
10^{-7}	0.1	$5.4787*10^4$	$5.3083*10^4$	$5.0285*10^4$

The final reported result is $\sqrt{54787} = 234$ root mean squared error, which is *65%* of the standard deviation of the prices (360). The model heat map for the predicted prices with the submitted script is below.



The heat map looks reasonable: close to landmarks prices are higher and if the landmark is in the centre, the prices are even higher (weight and hence the importance of the landmark is bigger). There are some signs of overfitting in regions where we did not have enough data: for example, it is highly unlikely that around 0.15 longitude and 51.55 latitude the expected price is zero. Apart from that, at other unseen regions, the model makes a reasonable prediction (around 800), which is effectively the bias parameter.

Personalized Tube

Let's suppose I want to live near Lambeth North and I would like to estimate my monthly rent costs on a one-bedroom flat. Simply using "testRegressor([51.4989 -0.11216], param)" (param obtained from the script containing the data cleaning and reference to testRegressor – see Appendix) results in 1589 pounds per months. No wonder about its high price: it is close to the Thames, museums, colleges and Waterloo. A quick google search for "rent flat near Lambeth North" gives some idea how accurate our prediction is: the first 5 one-bedroom flats that came up have an average price of 1710 pounds, so our prediction is very reasonable.

Appendix

Coursework_main.m

```
%% ======= Part 1: Clear workspace, load data ========

clc, clear
close all

load('london.mat')

%% ======= Part 2: Data cleaning ========
X = Prices.location;
target = Prices.rent;

%Assumption on the location of the useful data
target = target(X(:,1)<52); %IT IS VERY VERY IMPORTANT TO DO THIS FIRST
BECAUSE IF I OVERWRITE X, I WILL ALWAYS DELETE THE LAST SAMPLES OF TARGET
X = X(X(:,1)<52,:);
target = target(X(:,1)>51);
X = X(X(:,1)>51,:);
target = target(X(:,2)>-0.5);
X = X(X(:,2)>-0.5,:);
target = target(X(:,2)<0.4);
X = X(X(:,2)<0.4,:);

%Getting rid of multiple entries
out = unique([X, target],'rows');
X = out(:,1:2);
target = out(:,3);

%Getting rid of conflicting entries
if ~isequal(X,unique(X, 'rows'))
    [X,I,~] = unique(X, 'rows', 'first'); %could have taken the average of
conflicting prices of samples, but we have enough (N>>D)
    target = target(I);
end

%Assumption on the prices of the useful data
X = X(target<2000,:);
target = target(target<2000);
X = X(target>300,:);
target = target(target>300);
y = target;


%Simple anomaly detection to further filter out the outliers
X_mean = mean(X,1);
covar = 1/(size(X,1))*(bsxfun(@minus, X', X_mean')*bsxfun(@minus, X,
X_mean));
probab = @(X_in)multivariateGaussian(X_in, X_mean, covar);
prob_vals = probab(X);
y = y(prob_vals>3*10^-2,:); %Arbitrary limit, 3 samples dropped
X = X(prob_vals>3*10^-2,:);

%Normalization - NOT USED
% avg = mean(X,1);
% standard = std(X,1);
% X = bsxfun(@minus, X, avg);
% X = bsxfun(@rdivide, X, standard);

%3D plotting of cleaned samples
scatter3(X(:,1), X(:,2), y, 'b');
```

```
hold on

%Tube location cleaning
%Same assumption on location as before
metro = Tube.location;
metro = metro(metro(:,1)<52,:);
metro = metro(metro(:,1)>51,:);
metro = metro(metro(:,2)>-0.5,:);
metro = metro(metro(:,2)<0.4,:);

%Normalization of metro stations - NOT USED
% metro = bsxfun(@minus, metro, avg);
% metro = bsxfun(@rdivide, metro, standard);

%GET RID OF TUBE STATIONS WITHOUT HOUSES CLOSE
res_matrix = bsxfun(@plus, sum(X.^2, 2), bsxfun(@minus, (sum(metro.^2,
2))', 2*X*metro'));
%I want (SAMPLEi - MUx)^2, I calculate SAMPLEi^2+MUx^2-2*SAMPLEi*MUx, which
is the same but can be vectorized
%res_matrix is the distance how far sample i is from tube station x
[~, ix] = min(res_matrix, [], 2); %which tube station is closest to each
sample
threshold = 5; %GET RID OF TUBE STATIONS IF ONLY 5 HOUSES OR LESS ARE THE
CLOSEST TO IT - LESS CHANCE TO OVERFIT
centers = unique(ix);
metro_mus = [];
for i = 1:numel(centers)
    loc = centers(i);
    num = sum(ix==loc);
    if num > threshold
       metro_mus = [metro_mus loc];
    end
end
metro = metro(metro_mus,:);

%2D plotting of tube stations on the same graph
scatter3(metro(:,1), metro(:,2), zeros(size(metro,1),1), 'r');
title('Blue cicrles are houses and red circles are metro stations')
xlabel('Latitude')
ylabel('Longitude')
zlabel('Price')

%save('metro.mat', 'metro')

%% ======= Part 3: Use Cross Validation (K-folds) to choose best lambda
========
% My own method
% See main text for discussion of the method
lambda = [0]; %use to test trainRegressor
%lambda = [0, 10^-7 10^-6 10^-5]; %use to tune hyperparameters
covariance_scale = [0.1]; %use to test trainRegressor
%covariance_scale = [0.05 0.1 0.25]; %use to tune hyperparameters
[A,B] = meshgrid(lambda,covariance_scale);
c=cat(2,A',B');
options=reshape(c,[],2); %all possible combinations of lambda and
covariance_scale
m = size(X,1);
seq = randperm(m); %Shuffle the data
series = 0.1:0.1:0.9; %K = 10 folds
mse_min = 10^6; %Set this big enough
```

```
test_mse = zeros(size(options, 2),
(numel(series)+1)/2*(numel(series)+1)/2); %only works if K is even!
train_mse = zeros(size(test_mse));
crossval_mse = zeros(size(test_mse));
crossval_mse_mean = zeros(size(options));
X_K = cell((numel(series)+1), 1);
y_K = cell((numel(series)+1), 1);

%Partition the data into K folds

X_K{1} = X(seq(1:ceil(series(1)*m)), :);
y_K{1} = y(seq(1:ceil(series(1)*m)), :);
for i = 2:numel(series)
  idx = ceil(series(i-1)*m):ceil(series(i)*m);
  X_K{i} = X(seq(idx), :);
  y_K{i} = y(seq(idx), :);
end
X_K{numel(series)+1} = X(seq(ceil(series(end)*m):end), :);
y_K{numel(series)+1} = y(seq(ceil(series(end)*m):end), :);

%This is what the above code effectively does

%X_1 = X(seq(1:ceil(0.10*m)), :);
%X_2 = X(seq(ceil(0.10*m):ceil(0.20*m)), :);
%X_3 = X(seq(ceil(0.20*m):ceil(0.30*m)), :);
%X_4 = X(seq(ceil(0.30*m):ceil(0.40*m)), :);
%X_5 = X(seq(ceil(0.40*m):ceil(0.50*m)), :);
%X_6 = X(seq(ceil(0.50*m):ceil(0.60*m)), :);
%X_7 = X(seq(ceil(0.60*m):ceil(0.70*m)), :);
%X_8 = X(seq(ceil(0.70*m):ceil(0.80*m)), :);
%X_9 = X(seq(ceil(0.80*m):ceil(0.90*m)), :);
%X_10 = X(seq(ceil(0.90*m):end), :);

for i = 1:size(options,1)
  for j = 1:((numel(series)+1)/2) %crossval set index in the mse matrix,
only works if K is even
    X_crossval = X_K{j};
    y_crossval = y_K{j};
    for k = ((numel(series)+1)/2)+1:(numel(series)+1) %test set index in
the mse matrix
        ind = (j-1)*(numel(series)+1)/2+k-((numel(series)+1)/2);
      if k~= j
        X_test = X_K{k};
        y_test = y_K{k};
        folds = 1:(numel(series)+1);
        X_train = vertcat(X_K{folds(folds~=j & folds~=k), :}); %training
data
        %is the 80% that is not chosen for validation or testing
        y_train = vertcat(y_K{folds(folds~=j & folds~=k), :});

        %model = algo(X_train,y_train)
        %use this to tune hyperparameters:
        %param = trainRegressor_crossval(X_train, y_train, options(i,1),
options(i,2));
        %use this to test trainRegressor:
        param = trainRegressor(X_train, y_train);

        %y_pred_train = model(X_train);
        y_pred_train = testRegressor(X_train, param);
        %y_pred_crossval = model(X_crossval);
        y_pred_crossval = testRegressor(X_crossval, param);

        %y_pred_test = model(X_test);
        y_pred_test = testRegressor(X_test, param);

        train_mse(i,ind) = mse(y_train, y_pred_train); %put it in the
relevant location in the mse matrices
        crossval_mse(i,ind) = mse(y_crossval, y_pred_crossval);
        test_mse(i,ind) = mse(y_test, y_pred_test);

      end
    end
  end

  %Choose the best hyperparameters - the combination that on average
  %reduces the MSE in the cross-validation set the most
  crossval_mse_mean(i) = mean(crossval_mse(i, :));
  if crossval_mse_mean(i) < mse_min
    mse_min = crossval_mse_mean(i);
    best_lambda_mse = options(i,1);
    best_covariance_scale_mse = options(i,2);
    test_mse_best_option = mean(test_mse(i, :));
    train_mse_best_option = mean(train_mse(i,:));
    crossval_mse_best_option = mse_min;
  end
i %Reporting which combination of the options the algorithm is working on
end

%% ======= Part 4: Results ========

%Report the relevant results - see Table, main text
best_lambda_mse
best_covariance_scale_mse
test_mse_best_option
train_mse_best_option
crossval_mse_best_option
```

MultivariateGaussian.m (coursework_main.m, cleaning part /anomaly detection/ uses this function)

```
function p = multivariateGaussian(X, mu, Sigma2)
%    p = MULTIVARIATEGAUSSIAN(X, mu, Sigma2) Computes the probability
%    density function of the examples X under the multivariate gaussian
%    distribution with parameters mu and Sigma2. If Sigma2 is a matrix, it
is
%    treated as the covariance matrix. If Sigma2 is a vector, it is treated
%    as the \sigma^2 values of the variances in each dimension (a diagonal
%    covariance matrix)
%    X should have dimension n by k, where n is the number of samples and k
%    is the number of features (base functions)
%    Normalized distribution and vectorized implementation

k = length(mu); %Dimension of the Gaussian

if (size(Sigma2, 2) == 1) || (size(Sigma2, 1) == 1)
    Sigma2 = diag(Sigma2);
end
```

```
X = bsxfun(@minus, X, mu(:)'); %(:) makes mu a column vector so it does not
matter if input is row or column vector
p = (2 * pi) ^ (- k / 2) * det(Sigma2) ^ (-0.5) * ...
    exp(-0.5 * sum(bsxfun(@times, X * pinv(Sigma2), X), 2));

end
```