CM3015

Machine Learning

Midterm Coursework Report

By Hristo Stantchev

California Housing Model

Abstract:

The California Housing Model is a predictive model utilising multiple linear regression with gradient descent. It aims to develop a machine learning model which accurately predicts housing prices by derived data from a popular dataset on California Housing Prices. The model was relatively successful with 63.36% accuracy after significant work on data type and preprocessing issues.

Introduction:

The California Housing Model is a predictive model utilising multiple linear regression with gradient descent. It uses a dataset on California housing prices from a 1990 U.S. census. The model aims at accurately estimating house values based on easy to obtain variables – median income and location. Successful predictive models in regards to pricing could greatly automate or enhance services in the real estate industry for homeowners, workers, and end consumers as accurate price estimations could greatly benefit the field both for buyers (to know the realistic worth of properties) and sellers (listings close to estimated prices could appear more trustworthy).

- The dataset:
  (available in the scikit-learn datasets): 20640 entries across 8 features:
  - MedInc - block group median income
  - HouseAge - median block group house age
  - AveRooms - average rooms/household
  - AveBedrms - average bedrooms/household
  - Population - block group population
  - AveOccup - average members/household
  - Latitude - block group latitude
  - Longitude - block group longitude
  - MedHouseVal – the **target**, median house value in hundreds of thousands of dollars
- Problems:
  - Dataset contains several values that are unreliable and may lead to overfitting. This is especially noticeable due to the greatly uneven distribution of homes.
  - Simple Linear Regression models have had only slight successes by using median income as the explanatory value.
- Dataset Reference: Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291-297
- More information on the dataset here

Background:

The algorithm works upon on a dataset that has already been preprocessed. This includes creating x and y matrices for training and testing as well as trimming the number of features that are irrelevant or harmful to results.

It makes use of 2 functions:

1. Cost: return (1/(2 * n)) * np.sum(np.square(y_pred - y))
    a. Variables:
        i. n - size of y. In other words, integer holding the length of the target array
        ii. y_pred – the predicted results of this iteration. An array holding the output values
        iii. y – expected results of the dataset, the target array
    b. Logic:
        i. y is subtracted from y_pred in order to find the difference from expected to actual output
        ii. The resulting array is then squared in order to account for the multiple explanatory variables.
        iii. Then each value in the array of squared values is summed to derive the cost number
        iv. Finally, the resulting value is divided by 2n to get the average number (2 is for convenience). This gets the mean squared error value.
2. Gradient Descent: return (1/n)*np.dot(x.T,y_pred - y)
    a. Note: This is not full gradient descent, just the means of getting the derivative theta
    b. Variables:
        i. n - size of y. In other words, integer holding the length of the target array
        ii. x – the training set. In other words, 2d array where rows = entries, columns = features
        iii. y_pred – the predicted results of this iteration. An array holding the output values
        iv. y – expected results of the dataset, the target array
    c. Logic:
        i. Firstly – y is subtracted from y_pred in order to find the difference from expected to actual output
        ii. The resulting array alongside the transposition of x are put in a dot function in order to get the dot product of the output difference across the training set
        iii. That is finally divided by n in order to get the derivative theta and the value is returned.

Finally, both are used in the MLR (Multiple Linear Regression) function:

```
def MLR(x,y,epochs,learn_rate):
    n = len(y)
    theta = np.zeros((len(x[0]),1))
    costs = []

    for i in range(epochs):
        y_pred = np.dot(x,theta)
        cost = get_cost(n,y,y_pred)

        theta_gd = grad_desc(n,x,y,y_pred)
        theta = theta - (learn_rate*theta_gd)
        costs.append(cost)

        if(i%(epochs*0.1)==0):
            print(cost)

    return theta,costs
```

Parameters:

1. x – the training set x_train
2. y – the target values of the training set y_train
3. epochs – the number of iterations for the algorithm
4. learn_rate – alpha value, rate of which model learns

Variables:

1. n – size of y. In other words, integer holding the length of the target array
2. theta – 2d array of zeros of size (feature count,1)
3. y_pred – the predicted results of this iteration. An array holding the output values
4. theta_gd – derivative theta achieved by the return value of the gradient descent function
5. costs – an array of length (epochs) holding each cost value

Logic:

1. Firstly n, theta and costs are created
2. Then for each epoch:

    a. y_pred is initialised as an array of the dot product of x and theta. This derives the predictions. On the first iteration all y_pred values will be 0.
    b. Cost is calculated using the cost function. It is used later on to demonstrate the gradient curve approaching the local minimum.
    c. theta_gd is initialised with the gradient descent function.
    d. the theta for the next iteration is calculated by subtracting the product of theta_gd and the learning rate from itself. This is the main moving part of the code.
    e. the current cost is added to the costs array and printed (for user convenience) every 10% of the completion of all iterations.
3. Finally, the function returns theta and the costs array.

Methodology:

The work can be separated into 3 logical parts – dataset exploration, model design and Implementation, and Testing

1. Dataset Exploration:
    a. The dataset was loaded initially as a pandas data frame
    b. The data frame was then graphed into a histogram with all features and the target
    c. It was checked for inconsistencies – there were none
    d. Several features of the dataset had extreme differences between entries (Average rooms, bedrooms, occupation features, and the population feature). This caused overfitting. To avoid that they were preprocessed by applying the numpy logarithmic function and adding + 1 to their value. This would make them resemble more of a Gaussian bell curve.
    e. The features still had extreme disparities and caused overfitting was still an issue, therefore they were removed the X_train and X_test arrays.
    f. Finally, X_train and X_test had a column of values = 1 added in order to provide the x0theta0 value for the model.
2. Model Design and Implementation:
    a. Due to the target requiring a predictive model – linear regression was chosen.
    b. The model was effective by only using average income, however in order to increase its robustness, multiple linear regression was chosen. This would allow multiple explanatory variables as input at the cost of more processing time.
    c. The cost function of the model is the standard way of deriving the mean squared error of an iteration of multiple linear regression.
    d. Gradient descent was chosen as it helps to significantly reduce the number of iterations required for cost to reach its local minimum.
3. Testing:
    a. There were issues in the way y_train and y_test were derived. Their shape was (n,) instead of (n,1) which made any use of the dot numpy function return arrays of shape (n,n). In order to fix that fatal error, both y_train and y_test were reshaped.
    b. After issues were cleared, y_pred was made using the theta of the model and X_test, y_test.
    c. A margin of error was calculated in order to display the accuracy of the model when working with unfamiliar data.

d. All further testing revolved around finding the right values for the amount iterations and the learning rate (alpha).
e. Testing was done when results were most satisfactory (max accuracy was 63.36%).

Results:

The dataset consisted of 20640 entries. 20% of them were reserved for testing and 80% used in training. The use of the multiple linear regression with gradient descent model using 4 of the 8 features as input managed to yield a 63.36% success rate for 10 000 iterations at a learning rate of $45*10^{-7}$.

Evaluation:

1. Strengths:
    a. The model is incredibly time and space-efficient. Execution time was under half a second using no more than 80MB of RAM for 10 000 iterations.
    b. The model can have descent accuracy using only average income, house age, longitude and latitude as input.
    c. As a first machine learning project, it makes use of several techniques taught within the course and is well-documented.
2. Weaknesses:
    a. 63.36% accuracy is not satisfactory to be used in any production code.
    b. Further research and knowledge could have led to more efficient algorithms and/or model for the task at hand.
    c. Could definitely be plotted better, such as using heat maps or multiple spreads using a library such as Seaborn.

Conclusions:

Overall, average income, house age, longitude and latitude are not the most efficient predictors of house prices. The trained on data from California could easily interpret westwards longitude to equal more expensive properties as California has the densest population on its coastal cities. This leads to 2 main conclusions – commercial scale predictive models on property pricing need more nuanced features to work such as – inflation, country GDP, etc. Training sets should also be more varied and while linear regression poses a higher than 50% accuracy rate, solutions with other more complex predictive models could pose better results.

References:

Dataset details and credits, Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions, Statistics and Probability Letters, 33 (1997) 291-297:
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html