Linear regression using gradient descent

Problem representation:

The features that we use are as following and they are stored in a csv file :

Table XXX Features for Linear Regression

|  |  |
| --- | --- |
| Features | Description |
| Gender | Categorical;  Encoded as 1 or 0:  1: Male  0: Female |
| MarathonTime | Continuous;  It is average race time for one marathon; Encoded as float numbers; |
| marathonTimeAged | It is the product of MarathonTime and Age |
| TimeSqr | It is the square of MarathonTime |
| Age | Continuous;  Encoded as discrete integers |
| Experience | Categorical;  Whether the person has previous full marathon experience or not;  1: yes  0: no |

By multiplying and combining some features, we intend to reveal more relationships between features and outputs, and obtain a better hypothesis. During some unofficial trials, using each raw feature individually makes the training process very difficult. Often a convergence can not be obtained. Therefore, by doing trials that combine features and create features such as TimeSqr, we allow the classifier to detect more relationships between the inputs and outputs and get a converged result. We also make the following decisions: for the people without age information or without marathon experience, we replace their age and their race time information with the average values for all the dataset in order to reduce the impact on the training process. The marathon time is obtained by dividing the total race time (min) by the total number of full marathons that each individual has participated in. For the people who have several marathon experiences, their average result is stored. We divide marathonTime by 100 in csv file. The resulting marathonTimeAged, timeSqr and age are also divided by 100 again in our code in order to scale down our inputs and accelerate the training process since we set our initial weight parameters randomly close to 0 at the beginning.

Training method:

We choose the gradient method since it allows us to track the process more easily. The closed form solution involves a matrix inversion that may be applied on an invertible matrix. In order to avoid spending time budget on investigating the singular matrix problem, we select the gradient descent method. For this method, the hyper parameter to be set is the learning rate. So, based on Robbins-Monroe conditions, we choose a Robbins-Monroe sequence to ensure the convergence of the weights to a local minimum, which is desirable for linear regression method:

learning rate = 1/(k+1) for k=1,2,3….T

The gradient descent iterative process will keep going until the difference between the present and the previous weight matrices and the training error both go below certain threshold values. Sum of squared error formula is utilized to calculate the above two values. The thresholds are set by us while considering acceptable run-time cost and accuracy after several trials.

The initial weights are assigned to some random small floating numbers. For splitting the data, we use K-Fold Cross-Validation in order to have a better estimation of error. We use 31-Fold cross-validation to make data manipulation easier: we have 8711 examples and this number can only be divided by 31 in order to include everyone in the training and testing sets.

Results:

The evaluation process that we use is based on the K-Fold Cross-Validation. For each round of our cross validation, we use our training set to get an estimated set of weight parameters. Then we plug the parameters to obtain a training set error and a testing set/validation error. A sum of squared difference is calculated for each error and the following table shows the errors generated during our training process:

Table XXX Training and Validation Error for K-Fold Cross-Validation

|  |  |  |
| --- | --- | --- |
| Index | Err(train) | Err(validation) |
| 0 | 70.39 | 2.26 |
| 1 | 26.82 | 0.91 |
| 2 | 13.5 | 0.42 |
| 3 | 8.36 | 0.31 |
| 4 | 7.97 | 0.3 |
| 5 | 5.58 | 0.2 |
| 6 | 7.65 | 0.24 |
| 7 | 4.07 | 0.15 |
| 8 | 3.91 | 0.14 |
| 9 | 5.01 | 0.16 |
| 10 | 3.08 | 0.09 |
| 11 | 2.97 | 0.1 |
| 12 | 3.64 | 0.1 |
| 13 | 2.41 | 0.07 |
| 14 | 2.34 | 0.08 |
| 15 | 3.58 | 0.11 |
| 16 | 1.95 | 0.05 |
| 17 | 1.89 | 0.05 |
| 18 | 2.27 | 0.08 |
| 19 | 1.63 | 0.06 |
| 20 | 5.92 | 0.2 |
| 21 | 1.32 | 0.04 |
| 22 | 1.31 | 0.05 |
| 23 | 2.95 | 0.11 |
| 24 | 1.09 | 0.03 |
| 25 | 1.08 | 0.03 |
| 26 | 2.5 | 0.08 |
| 27 | 0.91 | 0.03 |
| 28 | 0.9 | 0.03 |
| 29 | 2.43 | 0.1 |
| 30 | 0.75 | 0.03 |
| Average: | 6.46 | 0.21 |

The results from K-Fold Cross Validation allows to get a better estimation of error for our model. So, the average validation error which is 0.21 is considered as predicted validation error.

We retrain the data using the whole dataset as training set in order to get the weight parameters and our hypothesis function, which is :

(0.0249, 0.0493, 0.8269, 0.1803, 0.0305, 0.0631, 0.2205)

This is the parameters that we use to generate our predicted results (refer to our prediction file). After comparing our prediction with the given data, our final training error is 69.37. The sum of marathon time of all dataset is 19976.89, so this equals an error of 0.35%. This error is considered acceptable, given the large number of examples.