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:  Average race time for one marathon: continuous, encoded as continuous 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 obtain a more precise prediction and decision boundary. 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 some average values in order to reduce the impact on the training process. The marathon time is obtained by dividing the total race time by the total distance of full marathons that each individual has participated in. So for the people who have several marathon experiences, their average result is stored. We divide marathonTime by 100 in csv file, and then, 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 close to 0 at the beginning.

Training method:

Since we have a large number of examples to train, we abandon the closed form method in order to avoid cumbersome operations which include matrix inversion and multiplication. So, for the gradient descent method, the hyper parameter that we have to 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. The thresholds are set by us while considering run-time cost and accuracy after several unofficial tests.

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 and to get a model with lower variance. We use 31-Fold cross-validation to make data manipulation easier, since in our csv file, we have 8711 examples, which is a lot. And it 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. The following table shows the errors generated during our training process:

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 |
| Training Error | 2.6622 | 21.8363 | 3.8795 | 45.1977 | 6.5629 | 97.9621 |
| Validation Error | 2.5499 | 22.6946 | 3.9758 | 48.4038 | 6.8212 | 102.1572 |

Table XXX Continued

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | 6 | 7 | 8 | 9 | 10 | 11 |
| Training Error | 14.1327 | 2.0449 | 133.6560 | 18.5279 | 2.9344 | 46.8535 |
| Validation Error | 14.3898 | 2.1591 | 138.7656 | 19.1279 | 3.0650 | 47.2985 |

Table XXX Continued

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | 12 | 13 | 14 | 15 | 16 | 17 |
| Training Error | 6.6298 | 104.8010 | 13.7432 | 1.8772 | 127.1287 | 18.5173 |
| Validation Error | 6.7376 | 107.8738 | 14.4866 | 1.9542 | 128.7379 | 18.9375 |

Table XXX Continued

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | 18 | 19 | 20 | 21 | 22 | 23 |
| Training Error | 2.7565 | 41.3072 | 5.9600 | 94.5076 | 12.1975 | 1.5572 |
| Validation Error | 2.8087 | 43.2650 | 6.1866 | 96.9010 | 13.1501 | 1.7231 |

Table XXX Continued

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Training Error | 106.6105 | 14.0496 | 1.8449 | 126.8092 | 18.0090 | 2.2394 | 31.7830 |
| Validation Error | 111.2988 | 14.8244 | 1.9433 | 129.0988 | 18.5426 | 2.4444 | 34.0299 |

The lowest validation error we get is 1.7231 at the 23 th round. Therefore, we keep the weight parameters generated during this round, which is :

(-0.011198, -0.065218, 0.955649, 0.019748, 0.049628, -0.020564, 0.073874)

and we use those parameters to generate our predicted results (refer to our prediction file). After comparing our prediction with the given data, our final validation error is 50.3845. This error is considered acceptable, given the large number of examples.

// README

Get the dataset by downloading the csv file via the following URL link:

To test the code, run the code in any Python environment with numpy and pandas packages and csv module are installed in advance.

You can put break points at each iteration of K-Fold Cross-Validation in order to track some results, such as training/testing error and index of cross-validation, printed on screen.