第一周机器学习

> 本文公式需要使用Mathjax，然后令人悲伤的是github不支持Mathjax
> 您可以将这篇md文件pull下来，使用您本地的markdown解析器解析
> 没有必要在公示显示上浪费时间，您也可以下载我本地生成的html用浏览器打开即可

Mathjax开源项目地址

梯度下降法的Python实现

参考代码，自己就一些细节进行优化 https://www.cnblogs.com/focusonepoint/p/6394339.html

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np
from numpy import linalg



def gradientDescent(x, y, theta, m, alpha, maxIteration):
  '''
  使用批处理梯度下降算法计算theta
  '''
  # 得到x的转置
  # 即 x的第一行为x1 第二行为x2 第三行全部初始化为1
  xTrains = x.transpose()
  # theta 是一个列向量
  for i in xrange(0,maxIteration):
    # x矩阵(10*3)与theta(3*1)矩阵相乘
    # hypothesis(i) = x1(i)*theta1(i) + x2(i)*theta2(2) + 1*theta0(i)
    hypothesis = np.dot(x, theta)
    # 作差
    loss = hypothesis - y
    # 当loss的范数在我们的误差允许范围内 就停止循环
    if (linalg.norm(loss) < 1e-5):
      break
    # xTrains (3*10) * loss(10*1) = gradient(3*1)
    # 计算代价函数
    gradient = (1.0/m) * np.dot(xTrains, loss)
    theta = theta - alpha * gradient
  print('the number of iteration is %d' % i);
  return theta


# define the prepared 训练集
# the meaning of column : x1,x2,y
dataSet = np.array([
  [1.1,1.5,2.5],
  [1.3,1.9,3.2],
```

```
  [1.5,2.3,3.9],
  [1.7,2.7,4.6],
  [1.9,3.1,5.3],
  [2.1,3.5,6.0],
  [2.3,3.9,6.7],
  [2.5,4.3,7.4],
  [2.7,4.7,8.1],
  [2.9,5.1,8.8],
])


# print(dataSet)
m,n = np.shape(dataSet)
# print(m,n)
trainData = np.ones((m,n))
# 截取dataSet的前N-1列
trainData[:,:-1] = dataSet[:,:-1]
# 获取dataSet的最后一列
trainLabel = dataSet[:,-1]

# print(m,n)
theta = np.ones(n)
# print(theta)
alpha = 0.001

# the max time of iteration 这个值定义的尽量大(考虑计算机的性能)
maxIteration = 10000000
theta = gradientDescent(trainData, trainLabel, theta, m, alpha, maxIteration)
print('thec value of theta is:')
print(np.round(theta,2))


# a test for the algorithm
x = np.array([
  [3.1, 5.5],
  [3.3, 5.9],
  [3.5, 6.3],
  [3.7, 6.7],
  [3.9, 7.1]
])


# define a predict function used to test
def predict(x,theta):
  m, n = np.shape(x)
  xTest = np.ones((m, n+1))
  xTest[:, :-1] = x
  yPre = np.dot(xTest,theta)
  return yPre

print('the predicted value is')
yP = predict(x, theta)
print(np.round(yP,2))
```

运行结果

```
the number of iteration is 114575
thec value of theta is:
[ 0.71  1.39 -0.38]
the predicted value is
[ 9.5 10.2 10.9 11.6 12.3]
[Finished in 2.2s]
```

优化技巧

- Feature Scaling（特征缩放）
- 多项式回归
    - $$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
    - $$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$
    - $$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}$$
    - 上面的举例只是为了说明，x_i的取值可以不是x的一次多项式，但是这里要注意的是特征缩放在这里显得尤为重要
- α选取技巧
    - 如果J（θ)的值随着θ的取值单调递增或者出现震荡，那么α应该选的小一点

Normal Equation（正规方程法）

思想

$$J\theta(x) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x) - y)^2$$

$$= a\theta^2 + b\theta + c$$

微积分思想：求导后令导数为零解方程可以求出极值点θ
对于θ是一个n维向量的情况，可以利用多元函数取极值的必要条件，即偏导数为0

结论

$$\theta = (X^T X)^{-1} X^T y$$

Note

1. No need to do feature scaling
2. 只适用于线性模型，不适合逻辑回归模型等其他模型
3. the pseudo inverse of matrix
    - redundant features (x中存在线性相关的量)
    - too many features (eg. m <= n 数据个数小于特征参数)

Code

这里我使用上一个梯度下降法的例子作为对比,采用相同的数据对比运行结果

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np

def normalEqation(x, y):
  '''
  使用正规方程算法计算theta
  '''
  # 得到x的转置
  xTrains = x.transpose()
  m,n = np.shape(x)
  # theta 为 n维列向量
  theta = np.linalg.pinv(np.dot(xTrains,x))
  theta = np.dot(theta,xTrains)
  theta = np.dot(theta,y)
  return theta


# define the prepared 训练集
# the meaning of column : x1,x2,y
dataSet = np.array([
  [1.1,1.5,2.5],
  [1.3,1.9,3.2],
  [1.5,2.3,3.9],
  [1.7,2.7,4.6],
  [1.9,3.1,5.3],
  [2.1,3.5,6.0],
  [2.3,3.9,6.7],
  [2.5,4.3,7.4],
  [2.7,4.7,8.1],
  [2.9,5.1,8.8],
])


# print(dataSet)
m,n = np.shape(dataSet)
# print(m,n)
trainData = np.ones((m,n))
# 截取dataSet的前N-1列
trainData[:,:-1] = dataSet[:,:-1]
# 获取dataSet的最后一列
trainLabel = dataSet[:,-1]



theta = normalEqation(trainData, trainLabel)
print('thec value of theta is:')
print(np.round(theta,2))


# a test for the algorithm
x = np.array([
  [3.1, 5.5],
  [3.3, 5.9],
```

```
    [3.5, 6.3],
    [3.7, 6.7],
    [3.9, 7.1]
  ])


# define a predict function used to test
def predict(x,theta):
  m, n = np.shape(x)
  xTest = np.ones((m, n+1))
  xTest[:, :-1] = x
  yPre = np.dot(xTest,theta)
  return yPre

print('the predicted value is')
yP = predict(x, theta)
print(np.round(yP,2))
```

运行结果：

```
thec value of theta is:
[ 0.61  1.45 -0.34]
the predicted value is
[ 9.5 10.2 10.9 11.6 12.3]
[Finished in 0.2s]
```

由此可以知道，在特征矩阵维度不是太大情况下，对于线性回归模型，**normal equation** 是一个优先选用的方法