# 机器学习01【机器学习】

# 主要内容

1. Anconda安装使用
2. python开发spark
3. 线性回归算法原理
4. 线性回归算法案例

# 学习目标

|  |  |  |
| --- | --- | --- |
| 节数 | 知识点 | 要求 |
| 第一节 （pySpark环境准备） | window配置python环境变量 | 掌握 |
| window配置Spark环境变量 | 掌握 |
| python中安装py4j模块 | 掌握 |
| python中安装pyspark模块 | 掌握 |
| 第二节（pySpark代码开发） | pySpark开发工具介绍 | 掌握 |
| python wordCount | 掌握 |
| python开发Spark原理 | 掌握 |
| python开发Spark注意问题 | 掌握 |
| 第三节（pySpark案例） | 统计PV,UV | 掌握 |
| 统计除了某个地区以外的UV | 掌握 |
| 统计每个网站最活跃的top2地区 | 掌握 |
| 统计每个网站最热门的操作 | 掌握 |
| 统计每个网站下最活跃的top3用户 | 掌握 |
| 第四节（线性回归算法） | 回归概念 | 掌握 |
| 一元线性回归 | 掌握 |
| 多元线性回归 | 掌握 |
| 线性回归原理 | 掌握 |
| 第五节（线性回归案例） | 梯度下降方式调节 | 掌握 |
| 模型过拟合 | 掌握 |
| 线性回归案例 | 掌握 |

# pySpark环境准备

1).window配置python环境变量

window安装python，配置python环境变量。安装python后,在环境变量path中加入安装的路径，cmd中输入python，检验python是否安装成功。



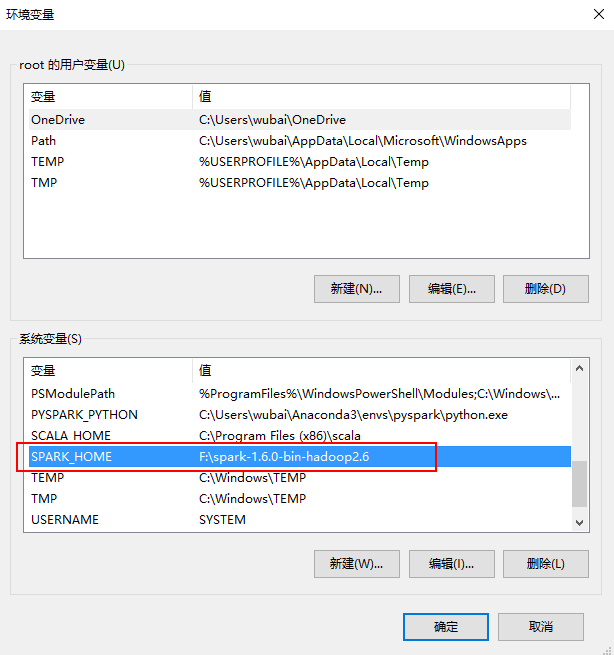
注意：如果使用的是anaconda安装的python环境，建议安装python3.5.x版本，这个版本和spark1.6兼容。如何在anaconda中安装python3.5.x版本的python环境？参考文档：***“Anaconda安装及使用.docx”***

2).window Spark环境变量。

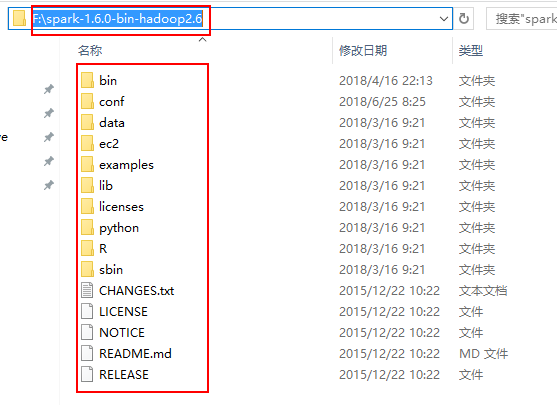
window中配置Spark运行环境及Spark环境变量。

a).解压spark的安装包到磁盘目录

b).在环境变量中配置SPARK\_HOME指定解压的路径,配置环境变量。



注意：解压spark安装包后，配置环境变量，一定配置到含有bin的外层。

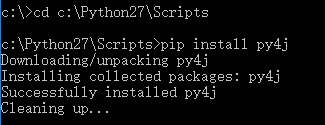


3).python中安装py4j模块。

在python环境中安装py4j模块(python调用java API的中间通信模块)

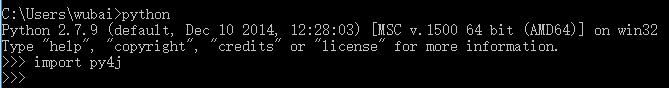
进入C:\Python27\Scripts使用pip install py4j：

pip install py4j 注：卸载python 模块：pip uninstall py4j



或者，将解压的spark安装包中的F:\spark-1.6.0-bin-hadoop2.6\python\lib\py4j-0.9-src\py4j拷贝C:\Python27\Lib\site-packages中。

验证py4j是否安装成功：进入python ,import py4j



注意：如果使用的是anaconda安装的python3.5.x的环境，之后使用这个python3.5.x环境，一定要将py4j模块放在安装的python3.5.x的目录，即：anaconda目录\envs\Python35【安装python3.5.x版本取的名称】\Lib\site-packages\中。

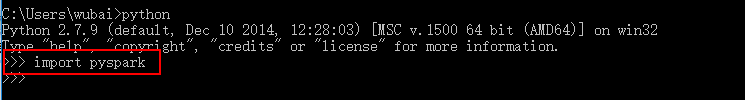
4).在python中安装pyspark模块

使用pip安装pyspark：

pip install pyspark ,会安装最新的版本的pyspark。

或者，将解压的spark安装包中的F:\spark-1.6.0-bin-hadoop2.6\python\pyspark拷贝到C:\Python27\Lib\site-packages中，验证pyspark模块是否安装成功：

进入cmd,输入python,导入pyspark模块，如果没错即安装成功。



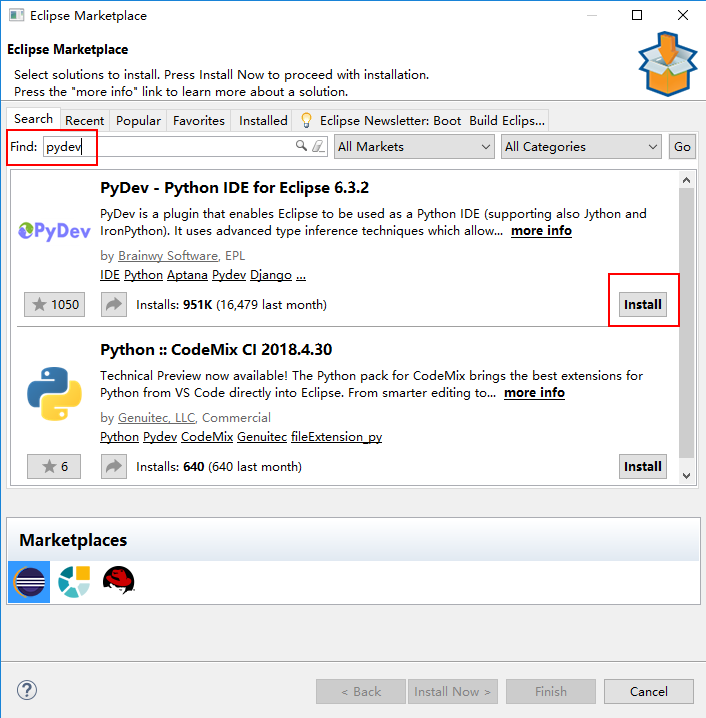
注意：如果使用的是anaconda安装的python3.5.x的环境，之后使用这个python3.5.x环境，一定要将pyspark模块放在安装的python3.5.x的目录，即：anaconda目录\envs\Python35【安装python3.5.x版本取的名称】\Lib\site-packages\中。

# pySpark代码开发

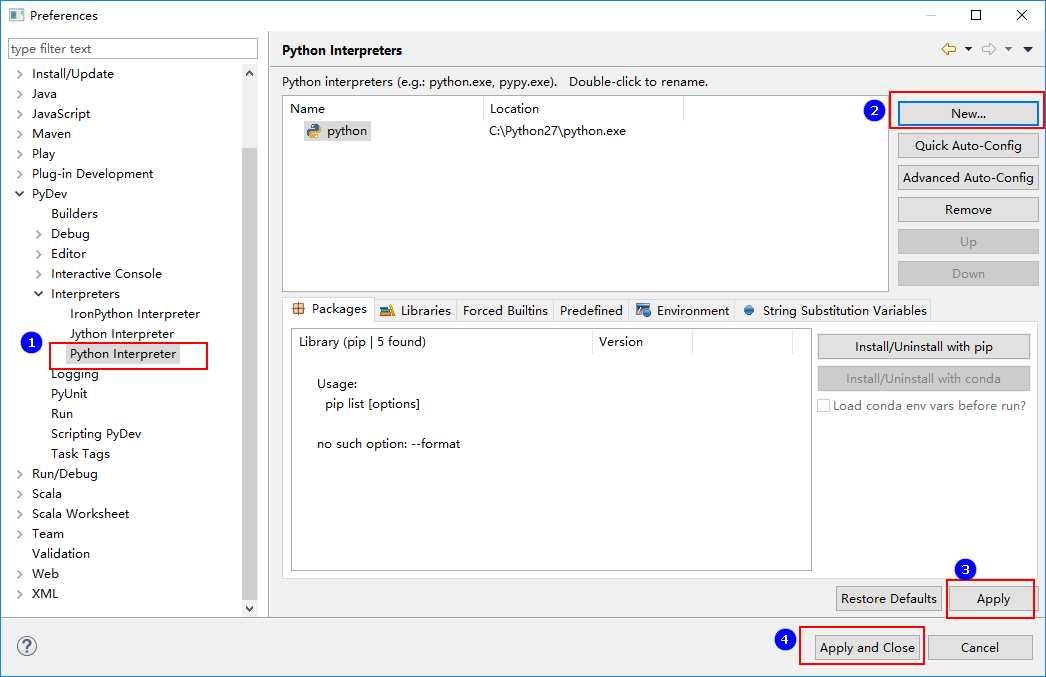
1. eclipse开发pySpark程序

在eclipse中开发pySpark程序，需要安装pydev插件。

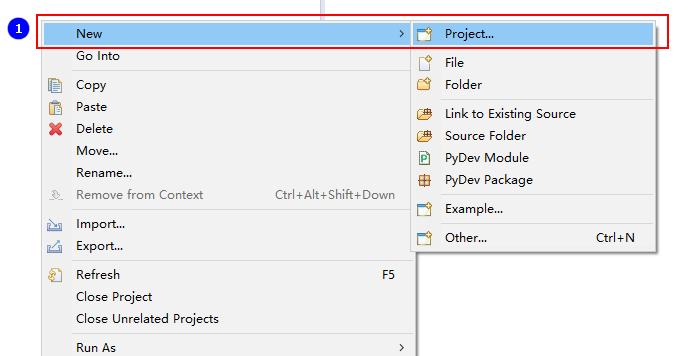
1).eclipse安装python插件,安装完成后重启。

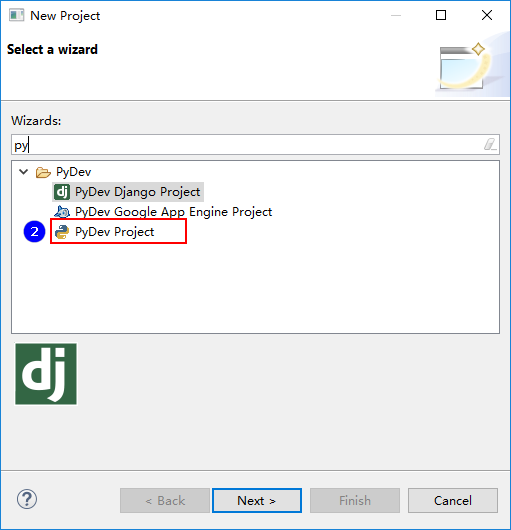


2). 在window--->preferences中找到python interpreter配置安装python的路径：



3).新建python项目：





1. pyCharm开发pySpark程序

pyCharm是专为开发python的工具，在pyCharm中直接可以编写pySpark代码。

1. IDEA开发pySpark程序

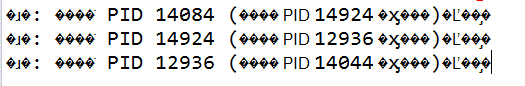
IDEA中开发pySpark代码需要安装python插件。

1. python wordcount

pySpark WordCount 案例:

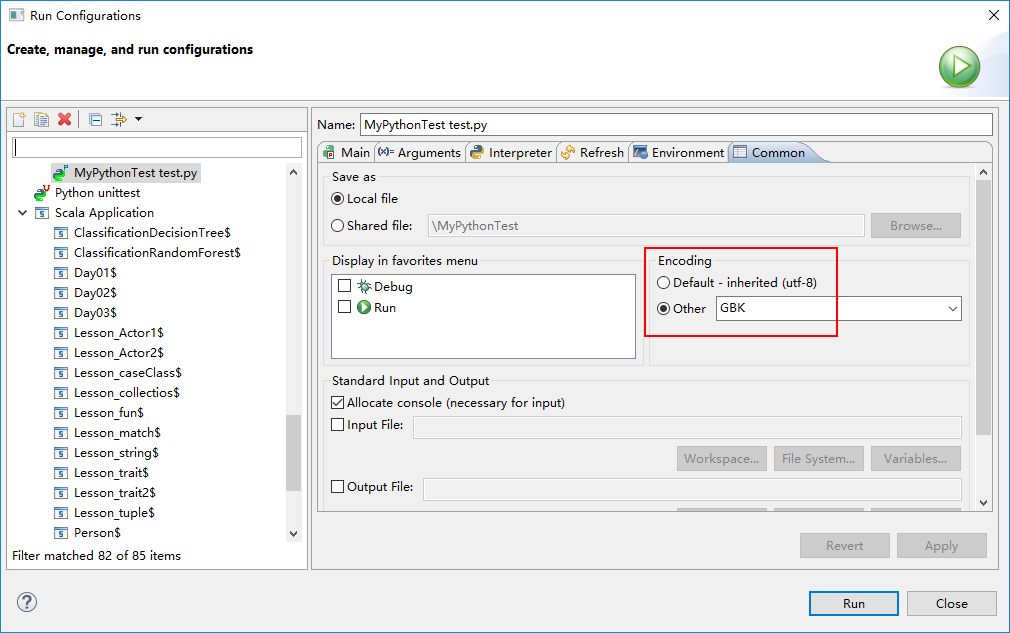
|  |
| --- |
| #coding:utf-8  #从pyspark中导入相应的包  from pyspark import SparkConf  from pyspark import SparkContext  def **show**(x):  print x  if \_\_name\_\_ == *'\_\_main\_\_'*:  conf = SparkConf()  conf.setAppName(*"wordcount"*)  conf.setMaster(*"local"*)  sc = SparkContext(conf=conf)  lines = sc.textFile(*"../../data/words"*, 2)  print *"lines rdd partition length = %d"*%(lines.getNumPartitions())  words = lines.flatMap(lambda line:line.split(*" "*), True)  pairWords = words.map(lambda word : (word,1),True)  result = pairWords.reduceByKey(lambda v1,v2:v1+v2, 3)  print *"result rdd partition length = %d"*%(result.getNumPartitions())  result.foreach(lambda t :show(t))  #将结果保存到文件  result.saveAsTextFile(*"../../data/wc-result"*) |

eclipse console 控制台乱码问题：



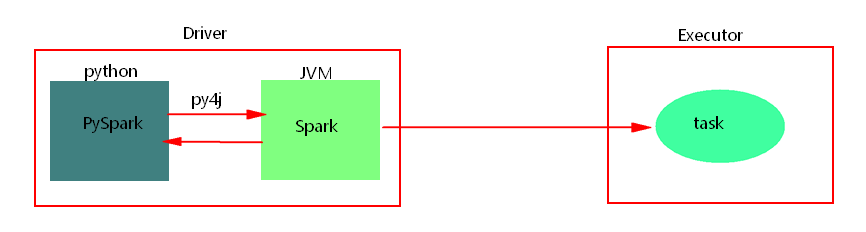
eclipse控制台只支持GBK编码。运行时需要修改编码，运行python文件时，

右键->Run As->Run Configurations->Common->Encoding 改为GBK,没有GBK就直接输入运行。



1. python开发Spark原理

使用python api编写pyspark代码提交运行时，为了不破坏spark原有的运行架构，会将写好的代码首先在python解析器中运行(cpython)，Spark代码归根结底是运行在JVM中的，这里python借助Py4j实现Python和Java的交互,即通过Py4j将pyspark代码“解析”到JVM中去运行。例如，在pyspark代码中实例化一个SparkContext对象，那么通过py4j最终在JVM中会创建scala的SparkContext对象及后期对象的调用、在JVM中数据处理消息的日志会返回到python进程中、如果在代码中会回收大量结果数据到Driver端中，也会通过socket通信返回到python进程中。这样在python进程和JVM进程之间就有大量通信。



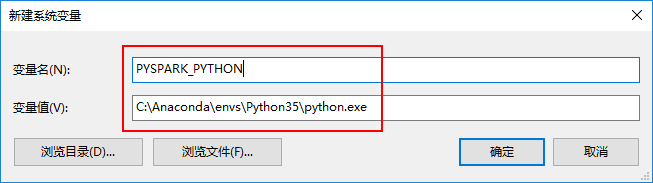
python开发spark，需要进行大量的进程间的通信，如果通信量过大，会出现“socket write error”错误，应尽量少使用回收数据类算子，也可以调节回收日志的级别，降低进程之间的通信。

1. python开发Spark问题
2. 关于安装路径

jdk，anaconda及python的安装路径中不能有空格和中文。

1. 指定spark使用的python版本

如果使用的anaconda更换了python3.5.x版本，之后在开发工具中指定了python解析器为3.5.x版本之后，运行python spark 代码时spark默认的使用的python版本使环境变量中指定的版本。会导致与指定的python解析器的python版本不一致。这时需要在环境变量中指定下PYSPARK\_PYTHON环境变量即可，值为指定的python3.5.x python解析器路径。如：



# pySpark案例

1. 统计PV,UV
   1. if \_\_name\_\_ == '\_\_main\_\_':
   2. conf = SparkConf()
   3. conf.setMaster("local")
   4. conf.setAppName("test")
   5. sc = SparkContext(conf=conf)
   6. #pv
   7. sc.textFile("./pvuv").map(lambda line:(line.split("\t")[4],1)).reduceByKey(lambda v1,v2:v1+v2).sortBy(lambda tp:tp[1],ascending=False).foreach(print)
   8. #uv
   9. sc.textFile("./pvuv").map(lambda line:line.split("\t")[1]+"\_"+line.split("\t")[4]).distinct().map(lambda one:(one.split("\_")[1],1)).reduceByKey(lambda v1,v2:v1+v2).sortBy(lambda tp:tp[1],ascending=False).foreach(print)
2. 统计除了某个地区外的UV
   1. if \_\_name\_\_ == '\_\_main\_\_':
   2. conf = SparkConf()
   3. conf.setMaster("local")
   4. conf.setAppName("test")
   5. sc = SparkContext(conf=conf)
   6. #uv
   7. sc.textFile("./pvuv").filter(lambda line:line.split("\t")[3]=='beijing').map(lambda line:line.split("\t")[1]+"\_"+line.split("\t")[4]).distinct().map(lambda one:(one.split("\_")[1],1)).reduceByKey(lambda v1,v2:v1+v2).sortBy(lambda tp:tp[1],ascending=False).foreach(print)
3. 统计每个网站最活跃的top2地区
   1. def get\_top2\_local(one):
   2. site = one[0]
   3. local\_iterable = one[1]
   4. local\_dic = {}
   5. for local in local\_iterable:
   6. if local in local\_dic:
   7. local\_dic[local] += 1
   8. else:
   9. local\_dic[local] = 1
   10. sorted\_list = sorted(local\_dic.items(),key = lambda x:x[1],reverse= True)
   11. return\_list = []
   12. if(len(sorted\_list)>=2):
   13. for i in range(0,2):
   14. return\_list.append(sorted\_list[i])
   15. else:
   16. return\_list = sorted\_list
   17. return return\_list
   18. if \_\_name\_\_ == '\_\_main\_\_':
   19. conf = SparkConf()
   20. conf.setMaster("local")
   21. conf.setAppName("test")
   22. sc = SparkContext(conf=conf)
   23. #统计每个网站最活跃的top2地区
   24. lines = sc.textFile("./pvuv")
   25. site\_local = lines.map(lambda line:(line.split("\t")[4],line.split("\t")[3]))
   26. site\_localIterable = site\_local.groupByKey()
   27. sorted\_result = site\_localIterable.map(lambda one:get\_top2\_local(one))
   28. sorted\_result.foreach(print)
4. 统计每个网站最热门的操作
   1. def get\_hot\_operator(one):
   2. site = one[0]
   3. operator\_iterable = one[1]
   4. operator\_dic = {}
   5. for operator in operator\_iterable:
   6. if operator in operator\_dic:
   7. operator\_dic[operator] += 1
   8. else:
   9. operator\_dic[operator] = 1
   10. sorted\_list = sorted(operator\_dic.items(),key = lambda x:x[1],reverse= True)
   11. return\_list = []
   12. if(len(sorted\_list)>=2):
   13. for i in range(0,1):
   14. return\_list.append(sorted\_list[i])
   15. else:
   16. return\_list = sorted\_list
   17. return return\_list
   18. if \_\_name\_\_ == '\_\_main\_\_':
   19. conf = SparkConf()
   20. conf.setMaster("local")
   21. conf.setAppName("test")
   22. sc = SparkContext(conf=conf)
   23. #统计每个网站最热门的操作
   24. lines = sc.textFile("./pvuv")
   25. site\_operator = lines.map(lambda line:(line.split("\t")[4],line.split("\t")[5]))
   26. site\_operatorIterable = site\_operator.groupByKey()
   27. sorted\_result = site\_operatorIterable.map(lambda one:get\_hot\_operator(one))
   28. sorted\_result.foreach(print)
5. 统计每个网站下最活跃的top3用户
   1. def get\_uid\_site\_count(one):
   2. uid = one[0]
   3. site\_iterable = one[1]
   4. site\_dic = {}
   5. for site in site\_iterable:
   6. if site in site\_dic:
   7. site\_dic[site] += 1
   8. else:
   9. site\_dic[site] = 1
   10. return\_list = []
   11. for site,count in site\_dic.items():
   12. return\_list.append((site,(uid,count)))
   13. return return\_list
   14. def get\_top3\_uid(one):
   15. site = one[0]
   16. uid\_count\_iterable = one[1]
   17. top3\_uid = ['','','']
   18. for tp in uid\_count\_iterable:
   19. uid = tp[0]
   20. count = tp[1]
   21. for i in range(0,len(top3\_uid)):
   22. if(top3\_uid[i]==''):
   23. top3\_uid[i] = tp
   24. break
   25. elif(count > top3\_uid[i][1]):
   26. for j in range(2,i,-1):
   27. top3\_uid[j] = top3\_uid[j-1]
   28. top3\_uid[i] = tp
   29. break
   30. return top3\_uid
   31. if \_\_name\_\_ == '\_\_main\_\_':
   32. conf = SparkConf()
   33. conf.setMaster("local")
   34. conf.setAppName("test")
   35. sc = SparkContext(conf=conf)
   36. #统计每个网站最活跃的top3用户
   37. lines = sc.textFile("./pvuv")
   38. uid\_site = lines.map(lambda line:(line.split("\t")[2],line.split("\t")[4]))
   39. uid\_siteIterable = uid\_site.groupByKey()
   40. uid\_site\_count = uid\_siteIterable.flatMap(lambda one:get\_uid\_site\_count(one))
   41. top3\_uid\_info = uid\_site\_count.groupByKey().map(lambda one:get\_top3\_uid(one))
   42. top3\_uid\_info.foreach(print)

# 线性回归算法

**什么是回归**？

从大量的函数结果和自变量反推回函数表达式的过程就是回归。线性回归是利用数理统计中回归分析来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。

**一元线性回归：**

只包括一个自变量（）和一个因变量（），且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。公式：

**多元线性回归：**

如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。公式：



对于一元线性回归公式来说，如果给定两组x,y的值，就可以求得和的值。例如：给定两组x，y的值x=3,y=5和x=6,y=8，那么可知如下：

5=+\*3 8=+\*6

可以求得=2,=1。两点确定一条直线，可以确定和的值。我们可以叫做一元线性回归公式的**截距**，可以叫做一元线性回归公式的**斜率**。

假设某个医院住院人数与医院的床位个数呈线性关系，即：

住院人数 = +\*医院床位数

在这里，如果想求得住院的人数，就要知道，的值，这里的叫做截距，叫做**权重**,“医院床位数”可以理解成影响住院人数的一个**维度**。通过给出两组数据，就可以得到截距和权重的值。

对于一元线性回归来说，如果现在有很多呈线性关系离散的点，假设现在需要找到一个线性回归公式来表示这些点自变量和因变量的关系，每两个点之间都能确定一条直线，如何找到一条直线来表示这些点的关系呢？

**如何确定这条直线？**

每一个离散的点在自变量处得到的的值记为，对应的，需要找到的这条直线在自变量处得到的的值记为，当所有的到对应的的距离差平方累加起来最小，我们就可以认为这条直线比较完美。也就是当平均误差公式：error= 最小时这条直线比较完美，其中，

代表一系列的真实离散点的y值，

代表一系列得到的直线在对应的x处的y值，一元线性回归中=。

m代表有m个离散点，前面的 是为了方便求导加上的。

 表示的是一系列权重,,….

这个公式也叫做**最小二乘法误差公式**。error的值如果是0说明确定的这条直线穿过了所有的点，对于离散分散的一组点，error的值不可能为0。

**确定权重的值（确定模型）**

当误差公式:



取得最小值时，能确定一条完美的直线，也就是确定一组 值与自变量的关系，拟合出来的一条直线能尽可能的通过更多的点。确定了error的最小值，也就有可能确定一组值。

**如何确定error的最小值？**

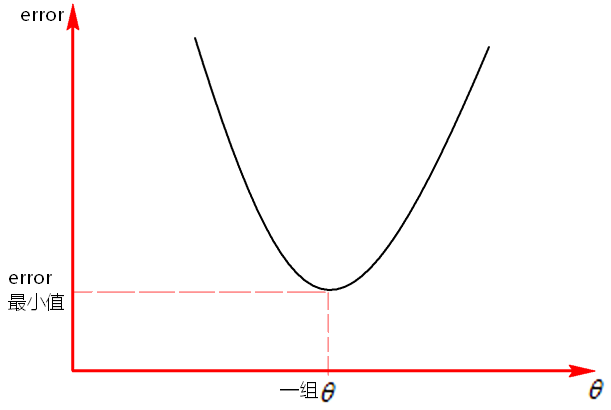
如果将error函数看成自变量关于error的函数，那么可以转换为：



如果将一组权重看成一列向量，表示一列维度组成的向量，就是这组向量的转置，

就是，那么现在的问题就是如何找到一组，使得error的值最小。

我们可以看出error函数的关系图像大概是一个开口向上的图像，那么当error的导数为0时，此时能得到一组的值对应的error的值最小。



通过求导的方式，理论上可以确定error的最小值，但是由于是一组数据，无法确定error最小下对应的这一组权重 。

正向的求导不能得到一组权重值，就不能确定线性回归的公式。那么可以根据数据集来穷举法反向的试，来确定线性回归的公式：

**如何反向穷举求得线性回归的公式（训练线性回归模型）？**

已知有线性回归关系的训练数据

已知线性回归的公式：



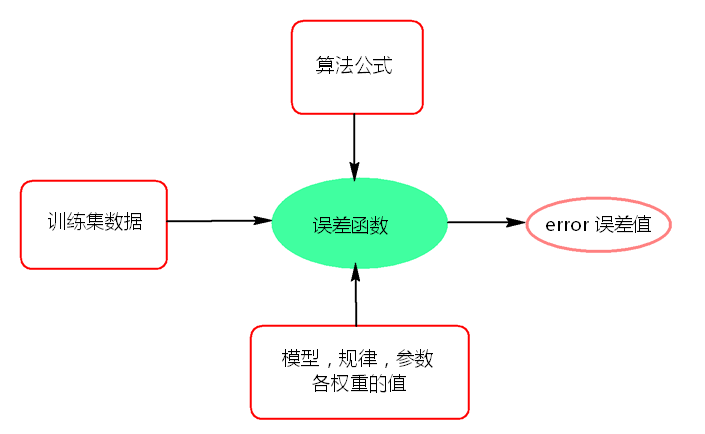
已知误差函数公式：

error=

要求得在error最小下的一组权重值，以便确定线性回归的公式。我们要确定的这组就是要确定一个模型。确定模型就是确定一组参数，就是确定这组的值。

穷举之前，会有一个初始模型，一般是随机数，假设=1，=1，=1…,有了初始模型那么针对每一条训练数据都能求得一个真实值与模型评估值的平方差，也就是针对一组测试数据，能得到在这组初始模型下error误差函数的值。那么得到的error值是不是最小值？需要不断调节初始模型参数，需要在训练模型时，指定调节初始模型，，…的调节步长step（也叫学习率）,迭代去求出每次调节模型后的error值，由于error的值不可能绝对的达到0（如果达到0说明训练出来的模型对应的线性回归公式完全的将所有训练数据集的点穿过），在训练模型之前指定一个最小error值，即：当迭代求出模型的error误差值小于指定的error值就停止迭代，那么当前这组值就是确定好的模型。

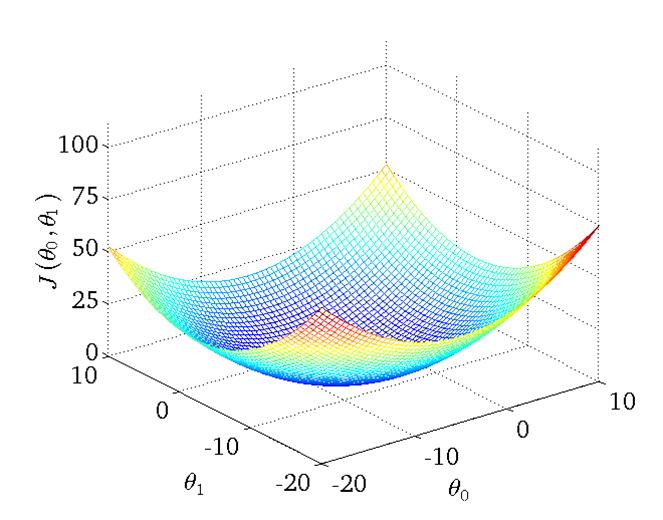
如果每次迭代求得模型对应的error大于指定的最小error值，要继续调节这组值，一直迭代到模型的error值小于指定的error值。



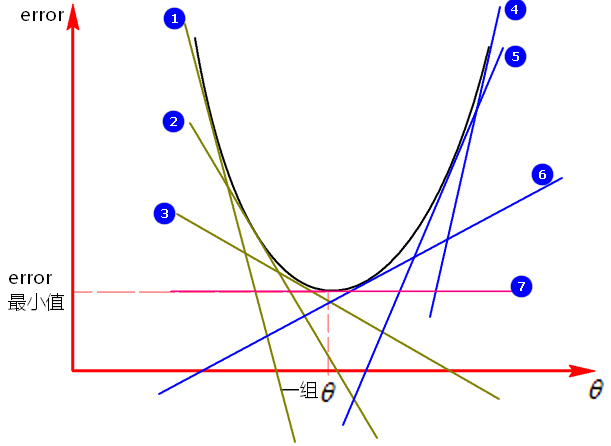
# 线性回归案例

1. 梯度下降法调节参数

假设和error的函数关系中，中只有一个维度，那么这个关系其实就是和error的关系，反应图上就是一维平面关系，如果中有两个维度和，那么这个反映到图上就是个三维空间关系，以此类推。



如果中只有一个维度，如下图：



图中①，②，③，④，⑤，⑥，⑦都是图像的切线（斜率），在切点处可以求得对应的error值，如何调节模型得到一组 值可以使得到的error值更小，如图，沿着①->②->③方向调节和沿着④->⑤->⑥方向调节，可以使error值更小，这种沿着斜率绝对值减少的方向,沿着梯度的负方向每次迭代调节的方法就叫做**梯度下降法**。

梯度下降是迭代法的一种,可以用于求解最小二乘问题，在求解error函数的最小值时，可以通过梯度下降法来一步步的迭代求解，得到最小化的error函数和模型参数值。

每次按照步长(学习率)来调节值，迭代求出error的最小值，这里的error不可能有一个固定的最小值，只会向着最小值的方向收敛。

判断模型error误差值收敛，也就是停止迭代的两种方式：

1. 指定一个error值，当迭代处理过程中得到的error值小于指定的值时，停止迭代。
2. 设置迭代次数，当迭代次数达到设置的次数时，停止迭代。

迭代周期：从调整参数开始到计算出error值判断是否大于用户指定的error是一个迭代周期。

当步长比较小时，迭代次数多，训练模型的时间比较长，当步长比较大时，有可能迭代的次数也比较多，训练模型的时间也会相对比较长，需要找到一个合适的步长。

1. 模型过拟合

训练模型都会将数据集分为两部分，一般会将0.8比例的数据集作为训练集，将0.2比例的数据集作为测试集，来训练模型。模型过拟合就是训练出来的模型在训练集上表现很好，但是在测试集上表现较差的一种现象，也就是模型对已有的训练集数据拟合的非常好（误差值等于0），对于测试集数据拟合的非常差，模型的泛化能力比较差。

**如何判断模型发生过拟合？**

训练出模型后，可以在训练集中测试下模型的正确率，在测试集中测试下模型的正确率，如果两者差别很大（测试集正确率小，训练集正确率大），那么模型就有可能发生了过拟合。

1. Spark Mllib 线性回归案例

|  |
| --- |
| **object** LinearRegression {  **def** main(args: Array[*String*]) {  // 构建Spark对象  **val** conf = **new** SparkConf().setAppName("LinearRegressionWithSGD").setMaster("local")  **val** sc = **new** SparkContext(conf)  Logger.getRootLogger.setLevel(Level.WARN)  // sc.setLogLevel("WARN")  //读取样本数据  **val** data\_path1 = "lpsa.data"  **val** data = sc.textFile(data\_path1)  **val** examples = data.map { line =>  **val** parts = line.split(',')  **val** y = parts(0)  **val** xs = parts(1)  **LabeledPoint**(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(\_.toDouble)))  }.cache()  **val** train2TestData = examples.randomSplit(Array(0.8, 0.2), 1)  /\*  \* 迭代次数  \* 训练一个多元线性回归模型收敛（停止迭代）条件：  \* 1、error值小于用户指定的error值  \* 2、达到一定的迭代次数  \*/  **val** numIterations = 100  //在每次迭代的过程中 梯度下降算法的下降步长大小 0.1 0.2 0.3 0.4  **val** stepSize = 1    **val** miniBatchFraction = 1  **val** lrs = **new** LinearRegressionWithSGD()  //让训练出来的模型有w0参数，就是由截距  lrs.setIntercept(**true**)  //设置步长  lrs.optimizer.setStepSize(stepSize)  //设置迭代次数  lrs.optimizer.setNumIterations(numIterations)  //每一次下山后，是否计算所有样本的误差值,1代表所有样本,默认就是1.0  lrs.optimizer.setMiniBatchFraction(miniBatchFraction)  **val** model = lrs.run(train2TestData(0))  println(model.weights)  println(model.intercept)  // 对样本进行测试  **val** prediction = model.predict(train2TestData(1).map(\_.features))  **val** predictionAndLabel = prediction.zip(train2TestData(1).map(\_.label))    **val** print\_predict = predictionAndLabel.take(20)  println("prediction" + "\t" + "label")  **for** (i <- 0 to print\_predict.length - 1) {  println(print\_predict(i).\_1 + "\t" + print\_predict(i).\_2)  }    // 计算测试集平均误差  **val** loss = predictionAndLabel.map {  **case** (p, v) =>  **val** err = p - v  Math.abs(err)  }.reduce(\_ + \_)  **val** error = loss / train2TestData(1).count  println(s"Test RMSE = " + error)  // 模型保存  **val** ModelPath = "model"  model.save(sc, ModelPath)  // val sameModel = LinearRegressionModel.load(sc, ModelPath)  sc.stop()  }  } |

# 本节作业

1. 完成pyspark环境配置。
2. pySpark代码开发。
3. 线性回归原理。