# 机器学习04【机器学习】

# 主要内容

1. 道路拥堵预测
2. 梯度下降法
3. 逻辑回归优化
4. 模型评估

# 学习目标

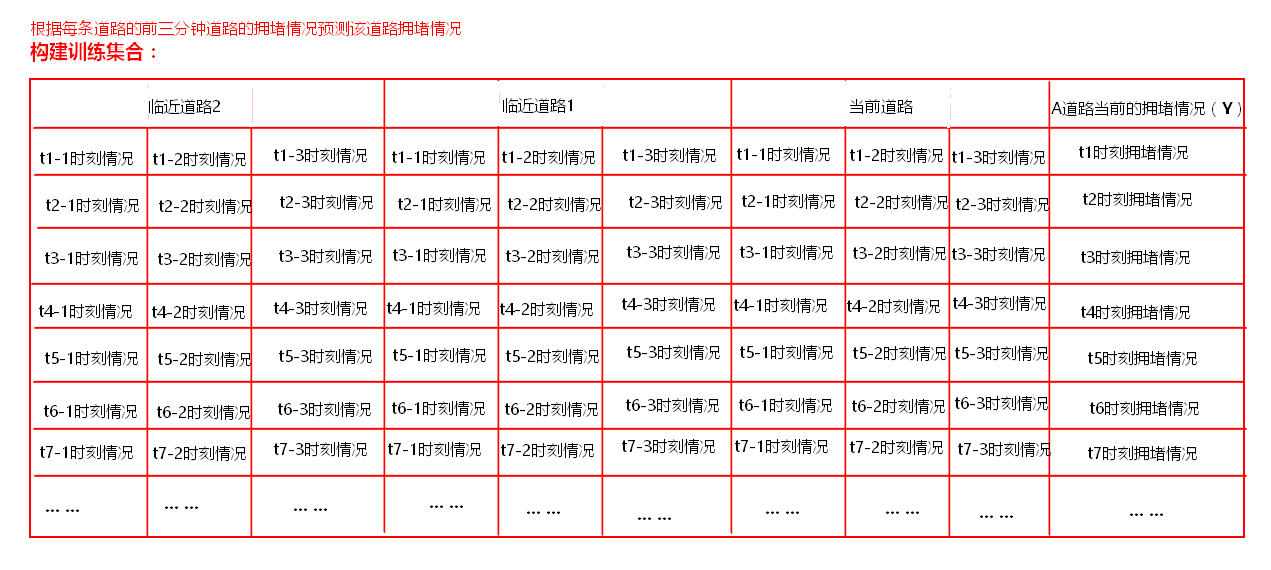
|  |  |  |
| --- | --- | --- |
| 节数 | 知识点 | 要求 |
| 第一节 （道路拥堵分析） | 道路拥堵情况分析 | 掌握 |
| 道路拥堵情况注意点 | 掌握 |
| 第二节（道路拥堵分析代码） | 道路拥堵分析代码 | 掌握 |
| 第三节（梯度下降法） | log函数与求导公式 | 掌握 |
| 梯度下降法 | 掌握 |
| 梯度下降方式案例演示 | 掌握 |
| 第四节（逻辑回归优化01） | 有无截距 | 掌握 |
| 线性不可分问题 | 掌握 |
| 调整分类阈值 | 掌握 |
| 鲁棒性调优 | 掌握 |
| 归一化数据 | 掌握 |
| 调整数据的正负值-均值归一化 | 掌握 |
| 训练方法的选择 | 掌握 |
| 第五节（逻辑回归优化02） | 归一化数据 | 掌握 |
| 调整数据的正负值-均值归一化 | 掌握 |
| 训练方法的选择 | 掌握 |

# 道路拥堵情况预测

1. **构建训练集：**

每条道路的拥堵情况不仅和当前道路前一个时间点拥堵情况有关系，还和与这条道路临近的其他道路的拥堵情况有关。甚至还和昨天当前时间点当前道路是否拥堵有关联。我们可以根据这个规律，构建训练集，预测一条道路拥堵情况。

假设现在要训练一个模型：使用某条道路最近三分钟拥堵的情况，预测该条道路下一分钟的拥堵情况。如何构建训练集？

构建的训练集有什么样的特点，依靠训练集训练的模型就具备什么样的功能。

1. **步骤：**
2. 计算道路每分钟经过的车辆数和速度总和，可以得到道路实时拥堵情况
3. 预测道路的拥堵情况受当前道路附近道路拥堵的情况，受这几个道路过去几分钟道路拥堵的情况。预测道路拥堵情况可以根据附近每条道路和当前道路前3分钟道路拥堵的情况来预测。用附近每条道路和当前道路前3分钟道路的拥堵情况来当做维度。统计这些道路过去5个小时内每分钟的前3分钟拥堵情况构建数据集。
4. 训练逻辑回归模型
5. 保存模型
6. 使用模型预测道路的拥堵情况
7. **道路拥堵预测注意问题：**

注意：提高模型的分类数，会提高模型的抗干扰能力。比如道路拥堵情况就分为两类：“畅通”、“拥堵”，如果模型针对一条本来属于“畅通”分类的数据预测错了，那么预测结果只能就是“拥堵”，那么就发生了质的改变。

如果我们将道路拥堵情况分为四类：“畅通”，“比较畅通”，“比较拥堵”，“拥堵”。如果模型针对一条本来数据“畅通”分类的数据预测错了，那么预测结果错的情况下就不是只有“拥堵”这个情况，有可能是其他三类的一种，也有一定的概率预测分类为“比较畅通”，那么就相当于提高了模型的抗干扰能力。

# 道路拥堵情况代码

1. **实时分析Kafka中数据，将结果存入Redis**
   1. val conf = new SparkConf().setAppName("CarEventCountAnalytics")
   2. conf.set("spark.streaming.kafka.consumer.cache.enabled","false")
   3. conf.setMaster("local[\*]")
   4. val ssc = new StreamingContext(conf, Seconds(5))
   5. val topics = Set("car\_events")
   6. val brokers = "mynode1:9092,mynode2:9092,mynode3:9092"
   7. val kafkaParams = Map[String, Object](
   8. "bootstrap.servers" -> brokers,
   9. "key.deserializer" -> classOf[StringDeserializer],
   10. "value.deserializer" -> classOf[StringDeserializer],
   11. "group.id" -> "predictGroup",//
   12. "auto.offset.reset" -> "earliest",
   13. "enable.auto.commit" -> (false: java.lang.Boolean)//默认是true
   14. )
   15. val dbIndex = 1
   16. // Create a direct stream
   17. val kafkaStream: InputDStream[ConsumerRecord[String, String]] = KafkaUtils.createDirectStream[String, String](
   18. ssc,
   19. PreferConsistent,
   20. Subscribe[String, String](topics, kafkaParams)
   21. )
   22. val events: DStream[JSONObject] = kafkaStream.map(line => {
   23. //JSONObject.fromObject 将string 转换成jsonObject
   24. val data: JSONObject = JSONObject.fromObject(line.value())
   25. println(data)
   26. data
   27. })
   28. val carSpeed : DStream[(String, (Int, Int))]= events.map(jb => (jb.getString("camera\_id"),jb.getInt("speed")))
   29. .mapValues((speed:Int)=>(speed,1))
   30. .reduceByKeyAndWindow((a:Tuple2[Int,Int], b:Tuple2[Int,Int]) => {(a.\_1 + b.\_1, a.\_2 + b.\_2)},Seconds(60),Seconds(60))
   31. carSpeed.foreachRDD(rdd => {
   32. rdd.foreachPartition(partitionOfRecords => {
   33. val jedis = RedisClient.pool.getResource
   34. partitionOfRecords.foreach(pair => {
   35. val camera\_id = pair.\_1
   36. val speedTotal = pair.\_2.\_1
   37. val CarCount = pair.\_2.\_2
   38. val now = Calendar.getInstance().getTime()
   39. // create the date/time formatters
   40. val dayFormat = new SimpleDateFormat("yyyyMMdd")
   41. val minuteFormat = new SimpleDateFormat("HHmm")
   42. val day = dayFormat.format(now)
   43. val time = minuteFormat.format(now)
   44. if(CarCount!=0&&speedTotal!=0){
   45. jedis.select(dbIndex)
   46. jedis.hset(day + "\_" + camera\_id, time , speedTotal + "\_" + CarCount)
   47. }
   48. })
   49. RedisClient.pool.returnResource(jedis)
   50. })
   51. })
   52. /\*\*
   53. \* 异步更新offset
   54. \*/
   55. kafkaStream.foreachRDD { rdd =>
   56. val offsetRanges: Array[OffsetRange] = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
   57. // some time later, after outputs have completed
   58. kafkaStream.asInstanceOf[CanCommitOffsets].commitAsync(offsetRanges)
   59. }
   60. ssc.start()
   61. ssc.awaitTermination()
2. **逻辑回归训练模型**
3. val sparkConf = new SparkConf().setAppName("train traffic model").setMaster("local[\*]")
4. val sc = new SparkContext(sparkConf)
5. // create the date/time formatters
6. val dayFormat = new SimpleDateFormat("yyyyMMdd")
7. val minuteFormat = new SimpleDateFormat("HHmm")
8. def main(args: Array[String]) {
9. // fetch data from redis
10. val jedis = RedisClient.pool.getResource
11. jedis.select(1)
12. // find relative road monitors for specified road
13. val camera\_ids = List("310999003001","310999003102")
14. val camera\_relations:Map[String,Array[String]] = Map[String,Array[String]](
15. "310999003001" -> Array("310999003001","310999003102","310999000106","310999000205","310999007204"),
16. "310999003102" -> Array("310999003001","310999003102","310999000106","310999000205","310999007204")
17. )
18. val temp = camera\_ids.map({ camera\_id =>
19. val hours = 5
20. val nowtimelong = System.currentTimeMillis()
21. val now = new Date(nowtimelong)
22. val day = dayFormat.format(now)//yyyyMMdd
23. val array = camera\_relations.get(camera\_id).get
24. /\*\*
25. \* relations中存储了每一个卡扣在day这一天每一分钟的平均速度
26. \*/
27. val relations: Array[(String, util.Map[String, String])] = array.map({ camera\_id =>
28. // fetch records of one camera for three hours ago
29. val minute\_speed\_car\_map: util.Map[String, String] = jedis.hgetAll(day + "\_'" + camera\_id + "'")
30. (camera\_id, minute\_speed\_car\_map)
31. })
32. // organize above records per minute to train data set format (MLUtils.loadLibSVMFile)
33. val dataSet = ArrayBuffer[LabeledPoint]()
34. // start begin at index 3
35. //Range 从300到1 递减 不包含0
36. for(i <- Range(60\*hours,0,-1)){
37. val features = ArrayBuffer[Double]()
38. val labels = ArrayBuffer[Double]()
39. // get current minute and recent two minutes
40. for(index <- 0 to 2){
41. //当前时刻过去的时间那一分钟
42. val tempOne = nowtimelong - 60 \* 1000 \* (i-index)
43. val d = new Date(tempOne)
44. val tempMinute = minuteFormat.format(d)//HHmm
45. //下一分钟
46. val tempNext = tempOne - 60 \* 1000 \* (-1)
47. val dNext = new Date(tempNext)
48. val tempMinuteNext = minuteFormat.format(dNext)//HHmm
49. for((k,v) <- relations){
50. val map = v //map -- k:HHmm v:Speed\_count
51. if(index == 2 && k == camera\_id){
52. if (map.containsKey(tempMinuteNext)) {
53. val info = map.get(tempMinuteNext).split("\_")
54. val f = info(0).toFloat / info(1).toFloat
55. labels += f
56. }
57. }
58. if (map.containsKey(tempMinute)){
59. val info = map.get(tempMinute).split("\_")
60. val f = info(0).toFloat / info(1).toFloat
61. features += f
62. } else{
63. features += -1.0
64. }
65. }
66. }
67. if(labels.toArray.length == 1 ){
68. //array.head 返回数组第一个元素
69. val label = (labels.toArray).head
70. val record = LabeledPoint(if ((label.toInt/10)<10) (label.toInt/10) else 10.0, Vectors.dense(features.toArray))
71. dataSet += record
72. }
73. }
74. val data: RDD[LabeledPoint] = sc.parallelize(dataSet)
75. // Split data into training (80%) and test (20%).
76. //将data这个RDD随机分成 8:2两个RDD
77. val splits = data.randomSplit(Array(0.8, 0.2))
78. //构建训练集
79. val training = splits(0)
80. /\*\*
81. \* 测试集的重要性：
82. \* 测试模型的准确度，防止模型出现过拟合的问题
83. \*/
84. val test = splits(1)
85. if(!data.isEmpty()){
86. // 训练逻辑回归模型
87. val model = new LogisticRegressionWithLBFGS()
88. .setNumClasses(11)
89. .setIntercept(true)
90. .run(training)
91. // 测试集测试模型
92. val predictionAndLabels = test.map { case LabeledPoint(label, features) =>
93. val prediction = model.predict(features)
94. (prediction, label)
95. }
96. predictionAndLabels.foreach(x=> println("预测类别："+x.\_1+",真实类别："+x.\_2))
97. // Get evaluation metrics. 得到评价指标
98. val metrics: MulticlassMetrics = new MulticlassMetrics(predictionAndLabels)
99. val precision = metrics.accuracy// 准确率
100. println("Precision = " + precision)
101. if(precision > 0.8){
102. val path = "hdfs://mycluster/model/model\_"+camera\_id+"\_"+nowtimelong
103. model.save(sc, path)
104. println("saved model to "+ path)
105. jedis.hset("model", camera\_id , path)
106. }
107. }
108. })
109. RedisClient.pool.returnResource(jedis)
     1. }
110. **加载模型进行道路预测**
111. val sparkConf = new SparkConf().setAppName("predict traffic").setMaster("local[4]")
112. val sc = new SparkContext(sparkConf)
113. sc.setLogLevel("Error")
114. // create the date/time formatters
115. val dayFormat = new SimpleDateFormat("yyyyMMdd")
116. val minuteFormat = new SimpleDateFormat("HHmm")
117. val sdf = new SimpleDateFormat("yyyy-MM-dd\_HH:mm:ss")
118. def main(args: Array[String]) {
119. val input = "xxxx-xx-xx\_17:15:00"
120. val date = sdf.parse(input)//yyyy-MM-dd\_HH:mm:ss
121. val inputTimeLong = date.getTime()
122. val day = dayFormat.format(date)//yyyyMMdd
124. // fetch data from redis
125. val jedis = RedisClient.pool.getResource
126. jedis.select(1)
127. // find relative road monitors for specified road
128. val camera\_ids = List("310999003001", "310999003102")
129. val camera\_relations: Map[String, Array[String]] = Map[String, Array[String]](
130. "310999003001" -> Array("310999003001", "310999003102", "310999000106", "310999000205", "310999007204"),
131. "310999003102" -> Array("310999003001", "310999003102", "310999000106", "310999000205", "310999007204"))
132. val temp = camera\_ids.foreach({ camera\_id =>
133. val array = camera\_relations.get(camera\_id).get
134. val relations: Array[(String, util.Map[String, String])] = array.map({ camera\_id =>
135. // fetch records of one camera for three hours ago
136. (camera\_id, jedis.hgetAll(day + "\_'" + camera\_id + "'"))
137. })
138. // organize above records per minute to train data set format (MLUtils.loadLibSVMFile)
139. val featers = ArrayBuffer[Double]()
140. // get current minute and recent two minutes
141. for (index <- 3 to (1,-1)) {
142. //拿到过去 一分钟，两分钟，过去三分钟的时间戳
143. val tempOne = inputTimeLong - 60 \* 1000 \* index
144. val currentOneTime = new Date(tempOne)
145. //获取输入时间的 "HHmm"
146. val tempMinute = minuteFormat.format(currentOneTime)//"HHmm"
147. println("inputtime ====="+currentOneTime)
148. for ((k, v) <- relations) {
149. val map = v //map : (HHmm,totalSpeed\_total\_carCount)
150. if (map.containsKey(tempMinute)) {
151. val info = map.get(tempMinute).split("\_")
152. val f = info(0).toFloat / info(1).toFloat
153. featers += f
154. } else {
155. featers += -1.0
156. }
157. }
158. }
159. // Run training algorithm to build the model
160. val path = jedis.hget("model", camera\_id)
161. if(path!=null){
162. val model: LogisticRegressionModel = LogisticRegressionModel.load(sc, path)
163. // Compute raw scores on the test set.
164. val prediction = model.predict(Vectors.dense(featers.toArray))
165. println(input + "\t" + camera\_id + "\t" + prediction + "\t")
166. }
167. })
168. RedisClient.pool.returnResource(jedis)
169. }

# 梯度下降法

1. **log函数及求导公式**

参考ppt

1. **梯度下降法**

逻辑回归的公式为：

 其中：

则： 叫做预测函数，一般表示结果取1的概率，那么对于单条样本分类1和0的概率分别为：

，可以整理成如下格式：



**构造损失函数(cost函数)**：

利用最大似然估计推导损失函数，最大似然估计就是利用已知的样本结果，反推能导致这样结果最大概率的一组参数值。举例：假设一个袋子中只装有未知数的黑球和白球，现在有放回的随机抽取球，做了一次测试，这次测试中抽取了10次，结果其中8次抽到了黑球，2次抽到了白球，假设抽到黑球的概率是p,那么抽到8次黑球和2次白球的概率为：，现在要求p是多少？

这里已经有了抽样10次，抽到了8次黑球，2次白球的样本结果，那么导致结果P最大概率下的p就是抽到黑球的概率。这里的思想就是最大似然思想。

根据最大似然估计可知，由已知的样本结果(各个维度的值和分类的类别)，反推导致这样结果出现最大概率的一组参数值()，可以取似然函数为：



两边取对数(单调性不变)，将相乘转为相加：

对数似然函数为： 。最大似然估计就是要求使取最大值时的一组。由于是凸函数，有极大值，可以乘以一个负系数，变成凹函数。得到逻辑回归的损失函数：

 ，现在只要求得最小值时的那组值，就是求得了满足最大似然函数最大的那组最佳值。

**利用梯度下降法求****的最小值**：

要求的的最小值，只要求得满足偏导为零时的那组就可以。















补充：==





通过将来求解方程组，由于参数的数量很多，得到最佳的一组值不大现实。可以使用梯度下降法来求解方程组的值。根据梯度下降法可以得到的更新过程如下：

 ，**代表学习步长**。表示在t+1时刻的一组参数值，表示在t时刻的一组参数值,表示在当前下随着变化而变化的趋势，

 随着的变大而变大

 随着的变大而变小

 随着的变化而不再变化

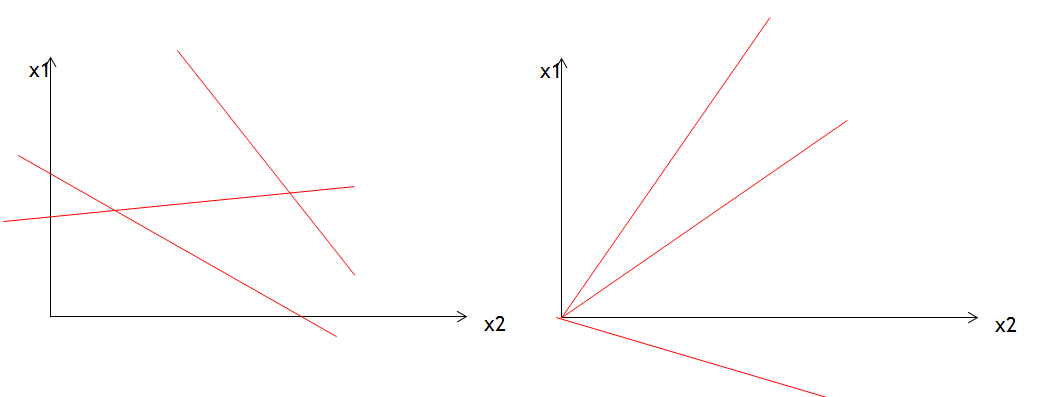
总结：就是根据步长每次按照梯度下降的方向迭代寻找一组最合适的使损失函数最小，这组就是逻辑回归公式中最优的权重值。

1. **梯度下降案例演示**
   1. def h(x):
   2. return w0 + w1\*x[0]+w2\*x[1]
   3. if \_\_name\_\_ == '\_\_main\_\_':
   4. # y=3 + 2 \* (x1) + (x2)
   5. rate = 0.001
   6. x\_train = np.array([[1, 2], [2, 1], [2, 3], [3, 5], [1, 3], [4, 2], [7, 3], [4, 5], [11, 3], [8, 7]])
   7. y\_train = np.array([7, 8, 10, 14, 8, 13, 20, 16, 28, 26])
   8. x\_test = np.array([[1, 4], [2, 2], [2, 5], [5, 3], [1, 5], [4, 1]])
   9. w0 = np.random.normal()
   10. w1 = np.random.normal()
   11. w2 = np.random.normal()
   12. for i in range(10000):
   13. for x, y in zip(x\_train, y\_train):
   14. w0 = w0 - rate\*(h(x)-y)\*1
   15. w1 = w1 - rate \*(h(x)-y)\*x[0]
   16. w2 = w2 - rate\*(h(x)-y)\*x[1]
   17. plt.plot([h(xi) for xi in x\_test])
   18. print("w0导数 = %f ,w1导数 = %f ,w2导数 = %f "%(rate\*(h(x)-y)\*1,rate\*(h(x)-y)\*x[0],rate\*(h(x)-y)\*x[1]))
   19. print(w0)
   20. print(w1)
   21. print(w2)
   22. result = [h(xi) for xi in x\_train]
   23. print(result)
   24. result = [h(xi) for xi in x\_test]
   25. #[9,9,12,16,10,12]
   26. print(result)
   27. plt.show()

# 逻辑回归优化01

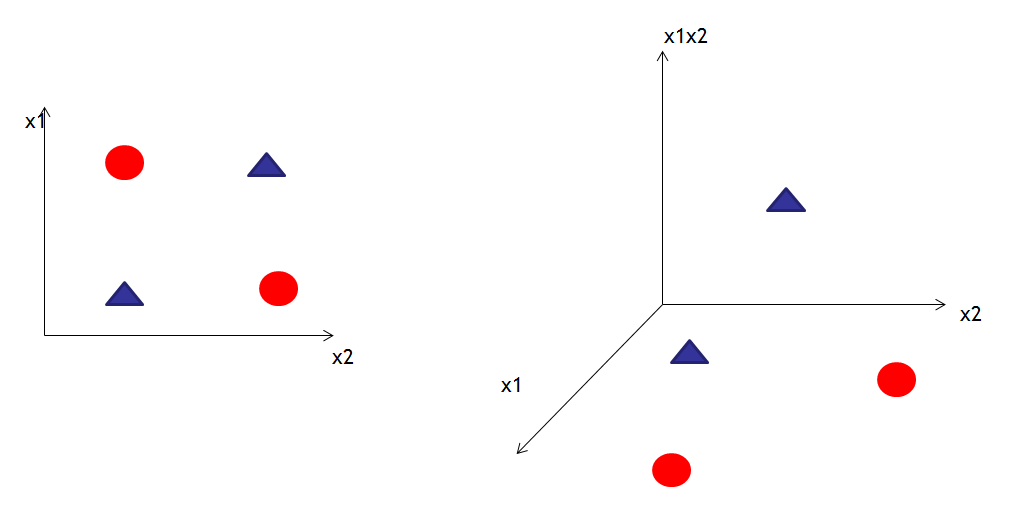
1. 有无截距

对于逻辑回归分类，就是找到z那条直线，不通过原点有截距的直线与通过原点的直线相比，有截距更能将数据分类的彻底。



1. 线性不可分问题

对于线性不可分问题，可以使用升高维度的方式转换成线性可分问题。低维空间的非线性问题在高维空间往往会成为线性问题。



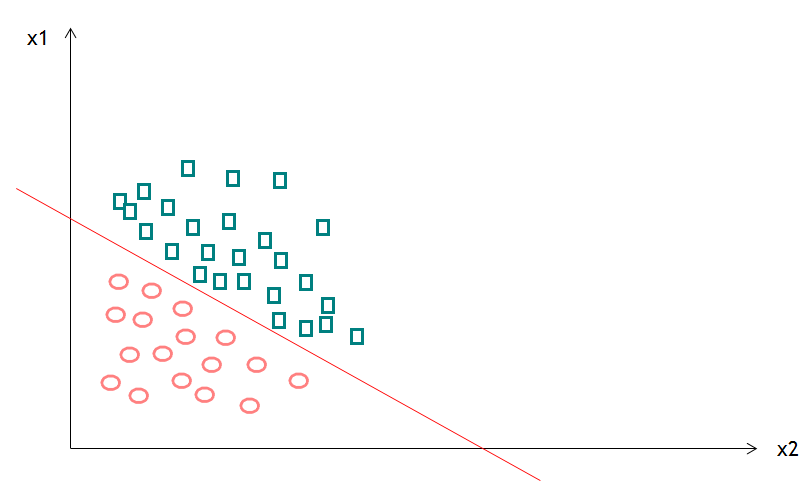
1. 调整分类阈值

在一些特定的场景下，如果按照逻辑回归默认的分类阈值0.5来进行分类的话，可能存在一些潜在的风险，比如，假如使用逻辑回归预测一个病人得癌症的概率是0.49，那么按照0.5的阈值，病人推测出来是没有得癌症的，但是49%的概率得癌症，比例相对来说得癌症的可能性也是很高，那么我们就可以降低分类的阈值，比如将阈值设置为0.3，小于0.3认为不得癌症，大于0.3认为得癌症，这样如果病人真的是癌症患者，规避掉了0.49概率下推断病人是不是癌症的风险。

降低阈值会使逻辑回归整体的正确率下降，错误率增大，但是规避了一些不能接受的风险。

1. 鲁棒性调优

鲁棒是Robust的音译，也就是健壮和强壮的意思,比如说，计算机软件在输入错误、磁盘故障、网络过载或有意攻击情况下，能不死机、不崩溃，就是该软件的鲁棒性，那么算法的鲁棒性就是指这个算法的抗干扰能力强。逻辑回归的决策边界 如下(当维度只有两个时，决策边界是一条直线)：



如下两个模型：

 与 

第一个模型相对于第二个来说各个维度的权重相对来说小，当遇到数据有噪声时，抗干扰的能力强，数据扰动小。比如：当时，第一个模型结果为90.3，第二个模型结果为903。当有数据噪声时，第一个模型结果为94.3，数据扰动为4，第二个模型结果为943，数据扰动为40。

可见模型维度权重小的模型抗干扰能力强，并不是模型维度权重越小，模型越好，如果模型权重很小，很小，那么模型受各个维度值变化的影响非常小，只与参数有关，与其他维度的权重没有关系，所有训练数据和测试数据都会得到相同的结果，那就是欠拟合的状态。如果模型维度权重太大，维度又多，这种模型当数据改变一点，扰动也大，对训练数据拟合的好，当测试数据本身改变一点时，变化也大，反而成了一种过拟合的状态。

无论是欠拟合或者过拟合模型都无法**泛化**(模型能够应用到新样本的能力)到新的样本数据，无法预测准确的值。各个维度的值偏大容易过拟合（**一般过多的维度特征，很少的训练数据集，容易导致过拟合出现**），各个维度的值偏小容易欠拟合，可见维度的权重不能太大，也不能太小，如何得到一组比较合适的权重值？这里就需要使用正则化。

正则化主要是用来解决模型过拟合，实际上，模型参数的权重值越小，通常对应越光滑的函数图像，就不易发生过拟合问题。正则化中将保留模型中的所有的维度，会将各个维度的权重保持尽可能小的状态。有两种正则化：L1正则化和L2正则化，公式如下：

 、 

 正则化的区别：倾向于使所有的w要么取1，要么取0。倾向于使w整体偏小。由此可见，正则化对于参数为0的维度，可以直接舍去，起到降维的作用。

如何应用正则化？简单直观来看可以重写逻辑回归的误差函数为：





其中：叫做正则化参数，一般设置不会超过1，0.3或者0.4最佳，不要太大，，叫做误差函数的惩罚项，就是对模型中参数做限制。

我们训练模型目的是使E尽可能的小，E中的越小代表训练出来的一堆参数能更好的拟合训练数据，代表我们训练的模型越准确，代表我们模型的准确能力。我们希望能够很好的拟合训练数据，但是不能过拟合，那么E后面的或者就是通过正则化项在保证得到的参数值较小，这里或者可以说代表模型的推广能力，也就是抗干扰能力。也就是说在控制两者之间的平衡，可以通过设置保证E尽可能小的情况下，既能保证训练出来的模型很好的拟合训练数据，又能保证得到的模型参数值保证在较小状态，不会发生过拟合。重写误差函数之后，误差就相对来说不大准确，正确率下降，但是相应的牺牲了正确率，提高了推广能力。

# 逻辑回归优化02

1. 归一化数据

|  |  |  |
| --- | --- | --- |
| **老虎数量** | **麻雀数量** | **是否污染** |
| 2 | 50640 | 1 |
| 3 | 55640 | 0 |
| 1 | 62020 | 0 |
| 0 | 54642 | 1 |

如图，要预测某区域是否污染，有两个维度特征：老虎数量和麻雀数量。这两个维度特征的量级不同，会导致训练出来模型中老虎这个特征对应的w参数大，而麻雀数量这个特征对应的w参数小，容易导致参数小的特征对目标函数的影响被覆盖，所以需要对每个特征的数据进行归一化处理，以减少不同量级的特征数据覆盖其他特征对目标函数的影响。

归一化数据可以使各个特征维度对目标函数的影响权重一致，提高迭代的求解的收敛速度。

**最大最小值归一化**：，缺点是抗干扰能力弱，受离群值影响比较大，中间容易没有数据。最大最小值归一化后的数据落在[0,1]之间。假设某个特征下有一组数据：1,2,3,4,5,100那么对数据使用最大最小值归一化后的值为：0,2/99,3/99,4/99,1。中间没有数据，受离群值100的影响大。

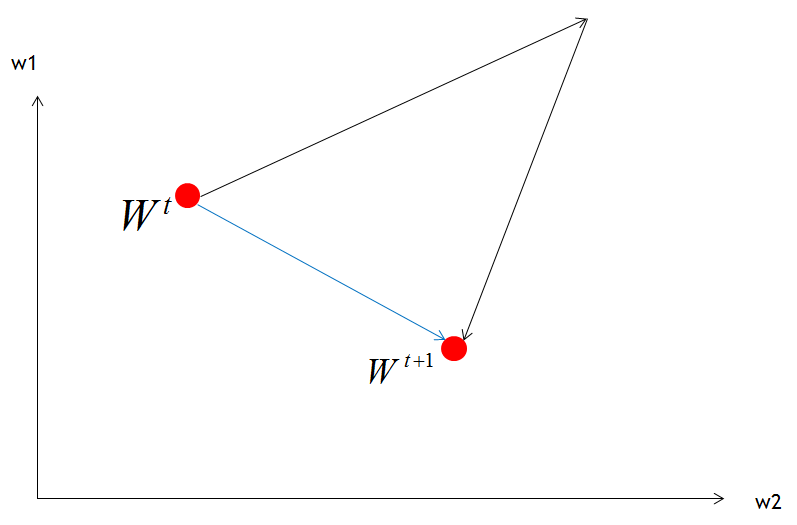
**方差归一化**： ,其中u是样本的均值，是样本的标准差（方差的开方，方差：所有点与均值的差值平方和）。方差归一化抗干扰能力强，和所有数据有关，求标准差需要所有的值介入，若有离群值，会被抑制下来。但是归一化后的数据最终的结果不一定落在0到1之间。

注意：理论上一个模型算法如果拿到训练集所有的特征一起训练模型就要归一化数据。决策树算法可以不归一化数据。

1. 调整数据的正负值-均值归一化

逻辑回归中，梯度下降法迭代求的过程如下：

可以转换写为： ，在步长值确定的情况下，A是一个值，当一组值为同号时，调整w参数只能朝着一个方向调节，要么使 参数朝着正方向调节，要么朝着负方向调节，如下图，假设由调节到，如果w只能朝着一个方向调节需要两步，如果由沿直线调节到只需要一步，但是这样就要向小的方向调节，需要向大的方向调节，这样就有了矛盾，如何解决矛盾？这里就需要使用均值归一化。



均值归一化是将原来的特征值减去这个特征在数据集中的均值，这样就会使x的各个维度取值上有正有负，在迭代求参数时，能减少迭代的次数。

1. 训练方法选择

训练逻辑回归的方法有：SGD和L-BFGS,两者的区别为：

SGD:随机从训练集选取数据训练，不归一化数据，需要专门在外面进行归一化，支持L1,L2正则化，不支持多分类。

L-BFGS:所有的数据都会参与训练，算法融入方差归一化和均值归一化。支持L1,L2正则化，支持多分类。

# 本节作业

1. 掌握道路拥堵情况分析。
2. 理解梯度下降法。
3. 逻辑回归优化。