个人资料

heybiiiiii

关注 发私信

访问： 385506次
积分： 5319
等级： BLOG 6
排名： 第5829名

原创： 164篇 转载： 2篇
译文： 0篇 评论： 111条

文章搜索

文章分类

Hadoop (43)
Datamining (6)
Spark (19)
Docker (13)
.NET (34)
JS (4)
设计模式 (1)
Others (11)
Linux (5)
Unity3D (1)
软件测试 (10)
HBase (11)
Kafka (1)
Scala (2)
Storm (3)
概率论与数理统计 (0)
线性代数 (0)
Drools (2)

文章存档

2016年09月 (1)
2016年07月 (6)

[置顶] KMeans算法检测网络异常入侵

标签： 算法 KMeans Spark SparkMllib
2016-05-09 17:09 4303人阅读 评论(1)

分类：
Spark (18)

非监督学习技术

在决策树算法预测森林植被中
我们可以体会到属于**监督学习**的分类和回归技术的强大，可以预测“即将发生”的事情
使用监督学习技术有一个**很关键的前提：**需要大量的数据对模型进行训练，模型能够从已知的数据中学习规律进而预测未知的数据

然而在某些场景下，并不是都能提供监督学习所需要的样本数据来训练模型，有可能只能给出部分正确的输出，甚至一个输出都没有
这种情况下，监督学习的技术就不能够使用了

此时，对应监督学习，另一种非监督学习技术就可以排上用场了

异常检查

顾名思义，异常检测就是要找出不同寻常的情况，**异常是一种未知的情况**，也就是说，无论何时何地，我们都无法归纳总结出所有的异常分类
如果可以，那么使用监督学习技术可以轻易的将网站的每个访问划分为“正常”或者“异常”
举个例子来说，我们永远不知道黑客有什么新的技术手段可以入侵你的网站系统，即使今天你有所有已知的黑客手段，但是谁知道明天又会有新的漏洞被黑客利用？
所以说，当一个访问请求被处理的时候，如果使用监督学习技术，恰好这是一个异常访问的请求，又恰好这是一种全新的异常类别
此时监督学习技术就会束手无策

在这种场景下，使用非监督学习技术可以有效的解决这个问题，通过学习，它们能够知道什么是正常的输入
从而能够判别出新数据和历史数据的差别，注意，**这里的有差别并不意味着该数据就是异常数据，只是说它和历史的正常数据有差异，值得进一步调查**

所以，**非监督学习并不是要将数据精确地划分到哪个类别中，而是对历史数据进行对比分析找出差异**

2016年05月 (10)
2016年03月 (7)
2016年02月 (3)

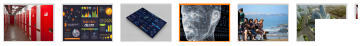
展开

阅读排行

Kubernetes用户指南（一）--... (23766)
Hadoop/Spark相关面试问题... (15076)
Kubernetes用户指南（四）--... (11828)

可视化数据

人脸识别算法



推荐文章

* 【2017年11月27日】CSDN博客更新周报
* 【CSDN】邀请您来GitChat赚钱啦！
* 【GitChat】精选——JavaScript进阶指南
* 改做人工智能之前，90%的人都没能给自己定位
* TensorFlow 人脸识别网络与对抗网络搭建
* Vue 移动端项目生产环境优化
* 面试必考的计算机网络知识点梳理

最新评论

Kubernetes用户指南（一）--快速开始、... 专业大数据 ：交流学习java架构大数据，加群460570824

CDH5.3配置Kerberos+LDAP+Sentry记录 x6696 ：非常有用 解决了几个问题 多泄

搭建Drools开发环境 木子-轩 ：@qq_15951857:<?xml version="1.0" enco...

MapReduce中的分布式缓存使用 大数据中的大叔 ：代码部分，测试过了，伪分布环境下测试不通过，完全分布环境下测试通过。我们传进去的Distribute...

在Docker中从头部署自己的Spark集群 2551 ：docker commit {containerId}#会返回一个iddocker tag {id}...

asp.net权限控制的方式 woshisunzewei ：http://122.112.248.118:8082/Account/Loginhttp://12...

KMeans均值聚类

聚类是最有名的非监督学习技术，它试图找到数据中的自然群组
一群特征相似而又与其他数据不同的数据点往往代表某种意义，从而将这些数据点划分为一个族群
聚类算法就是要将所有数据中的相似数据划分到同一个族群中
在网络的异常检测中，聚类算法是十分合适的，通过其将所有访问请求划分为一个个族群，将正常和异常的访问隔离开
我们可以进一步在异常的族群中分析这些数据是否属于网络入侵

K均值聚类是运行的最广泛的聚类算法，根据人为定义的一个k值，该算法会将数据聚类为k个族群
关于k值的确定需要结合业务场景和数据特性，并在反复的实验中得到一个最优
KMeans聚类的详细说明可以参考：
mahout运行测试与数据挖掘算法之聚类分析（一）kmeans算法解析

程序开发

数据集

案例中使用的是KDD Cup1999的数据集，可以在这里下载
进入下载页面后可以看到有很多数据集，本篇只用到kddcup.data.*数据文件，一个是完整数据集，解压缩之后有743M，一个是10%的数据集，解压缩之后只有45M
可以先使用10%的数据集进行代码测试，得到一个比较好的结果之后再运用到全部数据集中

数据集中的每一行代表一个网络请求，描述了该请求的所有信息，在决策树算法预测森林植被中
我们知道特征分为数值型和类别型，该数据集中也包含了这两种特征
每行数据的最后一个特征是目标特征，例如大部分请求被标记为normal表示正常访问
还有其他各种异常标记
正如之前讨论的，我们完全可以使用特征向量+目标特征的形式使用监督学习技术来训练模型进行预测
但是如果出现一个异常请求不在所有的目标类别中，这不糟糕了~

所以为了找出“未知的攻击”，我们先不在算法中使用这些目标特征

聚类的初步尝试

将要使用的数据集上传到HDFS之后，我们先来看看这些数据的基本信息：

```
val conf = new SparkConf().setAppName("KMeans")
val sc = new SparkContext(conf)
//读取数据
val rawData = sc.textFile("/spark_data/ch05/kddcup.data")
//根据类别查看统计信息,各个类别下有多少数据
//根据",",分割,只保留最后的类别
val catStatsData = rawData.map(_.split(",").last)
  //对类别的数目进行统计,并根据统计的数量从小打到排序
  .countByValue().toSeq.sortBy(_._2)
  //转换为从大到小排序
  .reverse
catStatsData.foreach(println)
```

运行结果：

```
(smurf.,280790)
(neptune.,107201)
(normal.,97278)
(back.,2203)
(satan.,1589)
(ipsweep.,1247)
(portsweep.,1040)
(warezclient.,1020)
(teardrop.,979)
(pod.,264)
(nmap.,231)
(guess_passwd.,53)
(buffer_overflow.,30)
(land.,21)
(warezmaster.,20)
(imap.,12)
(rootkit.,10)
(loadmodule.,9)
(ftp_write.,8)
(multihop.,7)
(phf.,4)
(perl.,3)
(spy.,2)
```

总共有23种不同类型，其中smurf和neptune的攻击类型最多，竟然比normal的正常访问还要多

之前提到的，由于样本数据中包含有类别型特征（下标为1，2，3和最后一个目标特征）但是KMeans算法要求特征都要为数值型，我们首先不对这些类别型特征做处理，直接跳过：

```
val labelsAndData = rawData.map { line =>
  //buffer是一个可变列表
  val buffer = line.split(",").toBuffer
  //下标1-3的元素
  buffer.remove(1, 3)
  //最后一个元素为label
  val label = buffer.remove(buffer.length - 1)
  //转换为Vector
  val vector = Vectors.dense(buffer.map(_.toDouble).toArray)
  (label, vector)
}
//数据只用到values部分
val data=labelsAndData.values.cache()
```

拿到可以进行模型训练的数据之后就可以编写聚类的代码了：

```
//训练模型
val kmeans = new KMeans()
val model = kmeans.run(data)
//输出所有聚类中心
model.clusterCenters.foreach(println)
```

输出结果为两个向量：

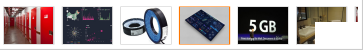
```
[47.979395571029514,1622.078830816566,868.5341828266062,4.453261001578883E-5,0.006432937937735

[2.0,6.9337564E8,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,57.0,3.0,
```

现在我们用这个模型来为每个数据分配族群，并输出每个聚类中心有哪些类别，各有多少个数据：

```
//输出每个聚类中心有哪些类别各有多少个数据
val clusterLabelCount = labelsAndData.map { case (label, datum) =>
  //为样本数据划分聚类中心
  val cluster = model.predict(datum)
  //返回数据的中心和类别二元组
  (cluster, label)
}.countByValue()
//排序之后格式化输出
clusterLabelCount.toSeq.sorted.foreach { case ((cluster, label), count) =>
  println(f"$cluster%1s$label%1s$count")
}
```


可视化数据



人脸识别算法

由于我们没有人为设置k的值，程序就自作主张设置了k=2，但是我们知道，数 3
个，因此这个模型肯定是错误的
可以看到输出的结果为：

```
0   back.           2203
0   buffer_overflow.    30
0   ftp_write.          8
0   guess_passwd.       53
0   imap.               12
0   ipsweep.            1247
0   land.               21
0   loadmodule.         9
0   multihop.           7
0   neptune.            107201
0   nmap.               231
0   normal.             97278
0   perl.               3
0   phf.                4
0   pod.                264
0   portsweep.          1039
0   rootkit.            10
0   satan.              1589
0   smurf.              280790
0   spy.                2
0   teardrop.           979
0   warezclient.        1020
0   warezmaster.        20
1   portsweep.          1
```

族群1只有一个数据点

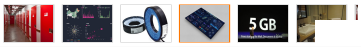
K值的选择

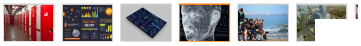
k的值到底设置为多少比较合适呢？
在示例的样本数据中，有23个类别，那么意味着k至少等于23
通常情况下我们要通过多次尝试才能找到最好的k值

那么什么样的k值才是“最好”的呢？
如果每个数据点都紧靠最近的质心，那么这个聚类是较优的

为了能够判断每个数据点到质心的距离，我们可以编写以下函数来判断：

可视化数据





人脸识别算法


```scala
/**
  * 计算两个向量之间的距离
  *
  * @param a 向量1
  * @param b 向量2
  *          欧式距离:空间上两个点的距离=两个向量相应元素的差的平方和的平方根
  **/
def distance(a: Vector, b: Vector) = {
  //求平方根
  math.sqrt(
    //将两个向量合并
    a.toArray.zip(b.toArray)
      //两个向量中的每个值相减
      .map(d => d._1 - d._2)
      //相间的值平方
      .map(d => d * d)
      //之后相加
      .sum)
}

/**
  * 计算数据点到聚类中心质心的距离
  *
  * @param datum 数据点
  * @param model kmeans模型
  **/
def distToCentrolid(datum: Vector, model: KMeansModel) = {
  //得到该数据点的聚类中心
  val cluster = model.predict(datum)
  //得到该聚类中心的质心
  val centrolid = model.clusterCenters(cluster)
  //计算距离
  distance(centrolid, datum)
}

/**
  * 根据各个数据点到该数据点聚类中心质心的距离来判断该模型优劣
  * @param data 样本数据
  * @param k k值
  * */
def clusteringScore(data: RDD[Vector], k: Int) = {
  val kmeans = new KMeans()
  //设置k值
  kmeans.setK(k)
  val model = kmeans.run(data)
  //计算样本数据到其各自质心的记录的平均值
  data.map { datum =>
    distToCentrolid(datum, model)
  }.mean()
}
```

有了判断模型优劣的标准之后，就可以通过取不同的k值来观察模型：

```scala
//取不同k值观察模型优劣
(5 to 40 by 5).map { k =>
  (k, clusteringScore(data, k))
}.foreach(println)
```

输出结果如下：

```
(5, 1779.3473960726312)
(10, 1054.5660956505587)
(15, 998.6026754769782)
(20, 438.0714456623944)
(25, 386.70458251397577)
(30, 329.4472646112194)
(35, 644.939843705805)
(40, 221.3547720824891)
```

可以看到，平均距离随着k的增大而降低
这点是毫无疑问的，因为随着族群点的增加，数据点离最近的质心肯定更近，当族群点等于数据量的时候平均距离为0，每个数据点都是自己构成的质心

而一个很奇怪的现象是，k=35时的距离竟然比k=30的时候要大
这是**因为KMeans的迭代过程是从一个随机点开始的，因此可能收敛于一个局部最小值**
k=35的情况可能是随机初始的质心造成的，也可能是由于算法在达到局部最小值之前就结束了
为了解决这个可能存在的问题，我们可以通过**对固定的k值多次聚类**，每次都随机不同的初始质心，然后在其中**选择最优的**
Spark Mllib提供了设置KMeans运行次数的方法，在clusteringSore函数中加入：

```
//设置该k值的聚类次数
kmeans.setRuns(10)
//设置迭代过程中,质心的最小移动值,默认为1.0e-4
kmeans.setEpsilon(1.0e-6)
```

setEpsilon设置迭代过程中,质心的最小移动值，移动值越小使得迭代的时间越多，聚类的结果更优化
现在我们重新选择k值进行评估：

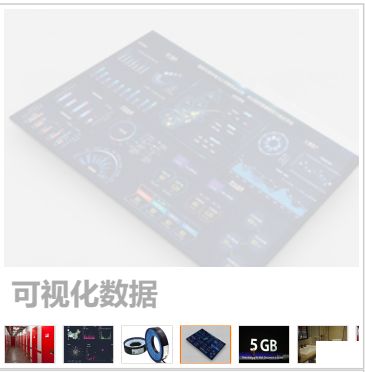
```
(50 to 130 by 10).map { k =>
  (k, clusteringScore(data, k))
}.foreach(println)
```

输出结果：

```
(50, 186.72154668205906)
(60, 144.68385906795461)
(70, 119.93958735623515)
(80, 109.46520683932486)
(90, 94.08809036196241)
(100, 76.37377878951273)
(110, 74.78271035271912)
(120, 71.515517236215)
(130, 65.13241545688685)
```

这个时候随着k的增大，平均距离在持续减小
但是**这个减少的幅度是有临界点的**，当k值超过这个临界点，即使继续增大，也不会显著地降低距离
这个临界点就是我们要找的最好的k值

从输出结果中可以看到，k值应该取130
我们再次打印出每个族群包含的类别和个数信息：




```
0   neptune.    48517
0   nmap.   93
0   normal. 2974
0   portsweep.  904
0   rootkit.    3
0   satan.  222
0   teardrop.   865
0   warezmaster.    1
1   portsweep.  1
2   warezclient.    59
3   multihop.   1
3   normal. 1
4   normal. 1
5   normal. 310
6   normal. 1
6   warezmaster.    15
7   normal. 22
8   normal. 713
9   normal. 19
10  normal. 1
11  normal. 6
12  normal. 1
13  normal. 2
14  back.   2155
15  normal. 1
16  normal. 102
17  normal. 17
18  smurf.  227840
19  normal. 1
...
...
117 warezmaster.    1
118 normal. 3
119 normal. 2
120 back.   9
120 normal. 23
121 back.   1
121 normal. 362
122 multihop.   1
122 normal. 5174
122 smurf.  177
122 warezclient.    31
123 normal. 1242
124 normal. 351
125 back.   18
125 normal. 5
126 normal. 2
127 normal. 1
128 multihop.   1
128 normal. 376
128 rootkit.    1
129 normal. 7
```

结果比第一次好上很多了

现在可以使用这个模型对全体数据进行聚类了，使用这个模型可以将数据中离质心最远的点找出来

将这个点到质心的距离设置为阈值

当有新的数据进来时，**判断这个数据到其质心的距离是否超过这个阈值**

超过就发出警报进行异常检车

总结

在本篇中，样本数据的类别型特征被直接跳过，但是在实际场景中是不能这么做的
正确的做法应该是讲类别型特征转换为数值型特征来训练，得到的模型和之前讨论过的将不太一样
但是模型模型的训练和寻找最优k值的过程是一致的

KMeans训练出来的模型还可以和Spark Streaming相结合，搭建出实时的网络流量异常预警系统

Github源码地址

作者：@小黑

- 上一篇 MapReduce的类型与格式
- 下一篇 MapReduce高级特性

查看评论

qq_22245863　1楼 2017-02-08 10:37发表

val label = buffer.remove(buffer.length - 1) ? remove是您自己写的函数吗？

可视化数据

人脸识别算法