龙猫先生

5-Spark高级数据分析-第五章 基于K均值聚类的网络流量异常检测

据我们所知，有‘已知的已知’，有些事，我们知道我们知道；我们也知道，有‘已知的未知’，也就是说，有些事，我们现在知道我们不知道。但是，同样存在‘不知的不知’—有些事，我们不知道我们不知道。

上一章中分类和回归都属于监督学习。当目标值是未知时，需要使用非监督学习，非监督学习不会学习如何预测目标值。但是，它可以学习数据的结构并找出相似输入的群组，或者学习哪些输入类型可能出现，哪些类型不可能出现。

5.1 异常检测

异常检测常用于检测欺诈、网络攻击、服务器及传感设备故障。在这些应用中，我们要能够找出以前从未见过的新型异常，如新欺诈方式、新入侵方法或新服务器故障模式。

5.2 K均值聚类

聚类是最有名的非监督学习算法，K均值聚类是应用最广泛的聚类算法。它试图在数据集中找出k个簇群。在K均值算法中数据点相互距离一般采用欧氏距离。

在K均值算法中簇群其实是一个点，即组成该簇的所有点的中信。数据点其实就是由所有数值型特征组成的特征向量，简称向量。

簇群的中心称为质心，它是簇群中所有点的算术平均值，因此算法取名K均值。算法开始时选择一些数据点作为簇群的质心。然后把每个数据点分配给最近的质心。接着对每个簇计算该簇所有数据点的平均值，并将其作为该簇的新质心。然后不断重复这个过程。

5.3 网络入侵

统计对各个端口在短时间内被远程访问的次数，就可以得到一个特征，该特征可以很好地预测端口扫描攻击。检测网络入侵是要找到与以往见过的连接不通的连接。K均值可根据每个网络连接的统计属性进行聚类，结果簇定义了历史连接类型，帮我们界定了正常的连接的区域。任何在区域之外的点都是不正常的。

5.4 KDD Cup 1999数据集

KDD Cup是数据挖掘竞赛，由ACM特别兴趣小组举办。1999年主题为网络入侵。
数据下载地址：http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
百度云：http://pan.baidu.com/s/1cFqnRS
数据集大小为108，每个连接信息包括发送的字节数、登录次数、TCP错误数等。数据集为CSV格式，每个连接占一行，包括38个特征。
我们关心的问题是找到“未知”的攻击。

5.5 初步尝试聚类

加载数据并查看有哪些类别标号及每类样本有多少：

Scala：

```
1 val rawData = sc.textFile("D:/Workspace/AnalysisWithSpark/src/main/java/advanced/chapter5/kddcu
2 rawData.map(_.split(',').last).countByValue().toSeq.sortBy(_._2).reverse.foreach(println)
```

Java：

```
//初始化SparkConf
SparkConf sc = new SparkConf().setMaster("local").setAppName("AnomalyDetectionInNetworkTraffic");
System.setProperty("hadoop.home.dir", "D:/Tools/hadoop-2.6.4");
JavaSparkContext jsc = new JavaSparkContext(sc);

//读入数据
JavaRDD<String> rawData =jsc.textFile("src/main/java/advanced/chapter5/kddcup.data/kddcup.data.corrected");

//查看有哪些类别标号及每类样本有多少
ArrayList<Entry<String, Long>> lineList = new ArrayList<>(rawData.map(line -> line.split(",")[line.split(",").length-1]).countByValue().entrySet());
Collections.sort(lineList, (m1, m2) -> m2.getValue().intValue()-m1.getValue().intValue());
lineList.forEach(line -> System.out.println(line.getKey() + "," + line.getValue()));
```

结果：

```
smurf.,2807886
neptune.,1072017
normal.,972781
satan.,15892
ipsweep.,12481
portsweep.,10413
nmap.,2316
back.,2203
warezclient.,1020
teardrop.,979
pod.,264
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
```

看来用Scala一行能写完的代码用Java还是比较麻烦的。

下面将CSV格式的行拆成列，删除下标从1开始的三个类别型列和最后的标号列。

Scala：

```
import org.apache.spark.mllib.linalg._
val labelsAndData = rawData.map { line =>
    val buffer = line.split(',').toBuffer
```

2013年10月 (1)
2013年8月 (1)
2013年7月 (1)
2013年6月 (1)
2013年5月 (4)
2013年4月 (5)
2013年3月 (4)
2013年2月 (3)
2013年1月 (9)
2012年12月 (23)
2012年11月 (3)
2012年10月 (13)
2012年9月 (16)
2012年7月 (20)
2012年6月 (12)
2012年5月 (18)

最新评论

1. Re:2-Spark高级数据分析-第二章 用Scala和Spark进行数据分析
不错，正在学习中
--loulley

2. Re:2-Spark高级数据分析-第二章 用Scala和Spark进行数据分析
基于Python Spark的大数据分析
课程观看地址：
xuetuwuyoucom/course/173
课程来自学途无忧网：
wwwxuetuwuyoucom
--学习童

3. Re:VS2008 f:/dd/vctools/vc7libs/s误调试
@龙猫先生好像没用啊，我只是在弹出的对话框中 加了一个黑色显示视频区域。...
--Riven丶

4. Re:VS2008 f:/dd/vctools/vc7libs/s误调试
@Riven丶在出问题的控件中增加一个DrawItem函数Overwrite：重写...
--龙猫先生

5. Re:VS2008 f:/dd/vctools/vc7libs/s误调试
大哥 这怎么重写啊
--Riven丶

阅读排行榜

1. 2-Spark高级数据分析-第二章 用Scala和Spark进行数据分析(4971)
2. Sigar使用(3518)
3. Mysql使用mysqldump按时间导出时的一个注意事项(2174)
4. 4-Spark高级数据分析-第四章 用决策树算法预测森林植被(1891)
5. 3-Spark高级数据分析-第三章 音乐推荐和Audioscrobbler数据集(1331)
6. 5-Spark高级数据分析-第五章 基于K均值聚类的网络流量异常检测(1069)
7. 使用SQL语法来查询Elasticsearch：Elasticsearch-SQL插件(1004)
8. 1-Spark高级数据分析-第一章 大数据分析(368)
9. 利用powerdesigner反向数据库结构，生成ER图(299)
10. 0-Spark高级数据分析-读书笔记(270)

```
    buffer.remove(1, 3)
    val label = buffer.remove(buffer.length-1)
    val vector = Vectors.dense(buffer.map(_.toDouble).toArray)
    (label,vector)
}
val data = labelsAndData.values.cache()
```

Java：

```
//删除下标从1开始的三个类别型列和最后的标号列
JavaRDD<Tuple2<String, Vector>> labelsAndData = rawData.map(line -> {
    String[] lineArrya = line.split(",");
    double[] vectorDouble = new double[lineArrya.length-4];
    for (int i = 0, j=0; i < lineArrya.length; i++) {
        if(i==1 || i==2 || i==3 || i==lineArrya.length-1) {
            continue;
        }
        vectorDouble[j] = Double.parseDouble(lineArrya[i]);
        j++;
    }
    String label = lineArrya[lineArrya.length-1];
    Vector vector = Vectors.dense(vectorDouble);
    return new Tuple2<String, Vector>(label,vector);
});

RDD<Vector> data = JavaRDD.toRDD(labelsAndData.map(f -> f._2));
```

对数据进行聚类

Scala：

```
import org.apache.spark.mllib.clustering._
val kmeans = new KMeans()
val model = kmeans.run(data)
model.clusterCenters.foreach(println)
```

Java：

```
//聚类
KMeans kmeans = new KMeans();
KMeansModel model = kmeans.run(data);

//聚类结果
Arrays.asList(model.clusterCenters()).forEach(v -> System.out.println(v.toJson()));
```

结果：

{"type":1,"values":

[48.34019491959669,1834.6215497618625,826.2031900016945,5.7161172049003456

E-6,6.487793027561892E-4,7.961734678254053E-

6,0.012437658596734055,3.205108575604837E-5,0.14352904910348827,0.00808830584493399,6.818511237273984E-5,3.6746467745787934E-5,0.012934960793560386,0.0011887482315762398,7.430952366370449E-5,0.0010211435092468404,0.0,4.082940860643104E-7,8.351655530445469E-4,334.9735084506668,295.26714620807076,0.17797031701994304,0.17803698940272675,0.05766489875327384,0.05772990937912762,0.7898841322627527,0.021179610609915762,0.02826081009629794,232.98107822302248,189.21428335201279,0.753713389800417,0.030710978823818437,0.6050519309247937,0.006464107887632785,0.1780911843182427,0.17788589813471198,0.05792761150001037,0.057659221424000437]}

{"type":1,"values":[10999.0,0.0,1.309937401E9,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,1.0,1.0,1.0,0.0,0.0,0.0,255.0,1.0,0.0,0.0,0.65,1.0,0.0,0.0,0.0,0.0,1.0,1.0]}

程序输出两个向量，代表K均值将数据聚类成k=2个簇。对本章的数据集，我们知道连接的类型有23个，因此程序肯定没能准确刻画出数据中的不同群组。

查看两个簇中分别包含哪些类型的样本。

Scala：

```
val clusterLabelCount = labelsAndData.map { case (label,datum) =>
    val cluster = model.predict(datum)
    (cluster,label)
}.countByValue
clusterLabelCount.toSeq.sorted.foreach {
    case ((cluster,label),count) =>
        println(f"$cluster%1s$label%18s$count%8s")
}
```

Java：

```
1 ArrayList<Entry<Tuple2<Integer, String>, Long>> clusterLabelCount = new
ArrayList<Entry<Tuple2<Integer, String>, Long>>(labelsAndData.map( v -> {
2     int cluster = model.predict(v._2);
3     return new Tuple2<Integer, String>(cluster, v._1);
4 }).countByValue().entrySet());
5
6 Collections.sort(clusterLabelCount, (m1, m2) -> m2.getKey()._1-m1.getKey()._1);
7 clusterLabelCount.forEach(t -> System.out.println(t.getKey()._1 +"\t"+ t.getKey()._2
+"\t\t"+ t.getValue()));
```

结果：

```
1 portsweep. 1
0 portsweep. 10412
0 rootkit. 10
0 buffer_overflow. 30
0 phf. 4
0 pod. 264
```

```
0 perl. 3
0 spy. 2
0 ftp_write. 8
0 nmap. 2316
0 ipsweep. 12481
0 imap. 12
0 warezmaster. 20
0 satan. 15892
0 teardrop. 979
0 smurf. 2807886
0 neptune. 1072017
0 loadmodule. 9
0 guess_passwd. 53
0 normal. 972781
0 land. 21
0 multihop. 7
0 warezclient. 1020
0 back. 2203
```

结果显示聚类根本没有任何作用。簇1只有一个数据点！

5.6 K的选择

计算两点距离函数：

Scala：

```
def distance(a: Vector, b: Vector) =
    math.sqrt(a.toArray.zip(b.toArray).
        map(p => p._1 - p._2).map(d => d * d).sum)
```

Java：

```
public static double distance(Vector a, Vector b){
    double[] aArray = a.toArray();
    double[] bArray = b.toArray();
    ArrayList<Tuple2<Double, Double>> ab = new ArrayList<Tuple2<Double, Double>>();
    for (int i = 0; i < a.toArray().length; i++) {
        ab.add(new Tuple2<Double, Double>(aArray[i],bArray[i]));
    }
    return Math.sqrt(ab.stream().map(x -> x._1-x._2).map(d -> d*d).reduce((r,e) -> r=
r+e).get());
}
```

计算数据点到簇质心距离函数：

Scala：

```
def distToCentroid(datum: Vector, model: KMeansModel) = {
    val cluster = model.predict(datum)
    val centroid = model.clusterCenters(cluster)
    distance(centroid, datum)
}
```

Java：

```
public static double distToCentroid(Vector datum, KMeansModel model) {
    int cluster = model.predict(datum);
     Vector[] centroid = model.clusterCenters();
     return distance(centroid[cluster], datum);
}
```

给定k值的模型的平均质心距离函数：

Scala：

```
import org.apache.spark.rdd._
def clusteringScore(data: RDD[Vector], k: Int) = {
    val kmeans = new KMeans()
    kmeans.setK(k)
    val model = kmeans.run(data)
    data.map(datum => distToCentroid(datum, model)).mean()
}
```

Java：

```
public static double clusteringScore(JavaRDD<Vector> data, int k) {
    KMeans kmeans = new KMeans();
    kmeans.setK(k);
    KMeansModel model = kmeans.run(JavaRDD.toRDD(data));
    return data.mapToDouble(datum -> distToCentroid(datum, model)).stats().mean();
}
```

对K从5到40进行评估：

Scala：

```
(5 to 40 by 5).map(k => (k, clusteringScore(data, k))).foreach(println)
```

Java：

```
List<Double> list = Arrays.asList(new Integer[]{1, 2, 3, 4, 5, 6, 7, 8}).stream().map(k -> clusteringScore(labelsAndData.map(f -> f._2), k*5)).collect(Collectors.toList()); list.forEach(System.out::println);
```

要算很久，结果：

1938.8583418059206

1686.4806829850777

1440.0646239087368

1305.763038353858

964.3070891182899

878.7358671386651

571.8923560384558

745.7857049862099

5.11 聚类实战

偷懒了，中间的那些和R相关还有标准化的没有写。

取k=150，聚类结果如下：

```
149 normal. 4
148 warezclient. 590
148 guess_passwd. 52
148 nmap. 1472
148 portsweep. 378
148 imap. 9
148 ftp_write. 2
…..
97 warezclient. 275
96 normal. 3
95 normal. 1
94 normal. 126
93 normal. 47
92 normal. 52196
92 loadmodule. 1
92 satan. 1
92 buffer_overflow.3
92 guess_passwd. 1
91 normal. 1
90 normal. 3
89 normal. 6
88 normal. 12388
…..
16 normal. 1
15 normal. 11
14 normal. 68
13 normal. 232
12 normal. 1
11 portsweep. 1
10 portsweep. 1
9 warezclient. 59
9 normal. 1
8 normal. 1
7 normal. 1
6 portsweep. 1
5 portsweep. 1
4 portsweep. 1
3 portsweep. 2
2 portsweep. 1
1 portsweep. 1
0 smurf. 527579
0 normal. 345
```

作为示例，我们在原始数据上进行异常检查：

Scala：

```
val model = ...
val originalAndData = ...
val anomalies = originalAndData.filter { case (original, datum) =>
    val normalized = normalizeFunction(datum)
    distToCentroid(normalized, model) > threshold
```

```
6 }.keys
```

Java：

```
1         KMeans kmeansF = new KMeans();
2         kmeansF.setK(150);
3         KMeansModel modelF = kmeansF.run(data);
4
5         System.out.println("json:---------");
6         Arrays.asList(modelF.clusterCenters()).forEach(v ->
System.out.println(v.toJson()));
7
8         ArrayList<Entry<Tuple2<Integer, String>, Long>> clusterLabelCountF = new
ArrayList<Entry<Tuple2<Integer, String>, Long>>(labelsAndData.map( v -> {
9             int cluster = modelF.predict(v._2);
10            return new Tuple2<Integer, String>(cluster, v._1);
11        }).countByValue().entrySet());
12
13        Collections.sort(clusterLabelCountF, (m1, m2) -> m2.getKey()._1-
m1.getKey()._1);
14        clusterLabelCountF.forEach(t -> System.out.println(t.getKey()._1 +"\t"+
t.getKey()._2 +"\t\t"+ t.getValue()));
15
16        //距离中心最远的第100个点的距离
17        JavaDoubleRDD distances = labelsAndData.map(f -> f._2).mapToDouble(datum ->
distToCentroid(datum, modelF));
18        Double threshold = distances.top(100).get(99);
19
20        JavaRDD<Tuple2<String, Vector>> result = labelsAndData.filter(t ->
distToCentroid(t._2, modelF) > threshold);
21        System.out.println("result:---------");
22        result.foreach(f -> System.out.println(f._2));
```

结果如下：

[2.0,222.0,1703110.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,73.0,255.0,1.0,0.0,0.0,0.01,0.03,0.0,0.0,0.0,0.0]

[10.0,194.0,954639.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,255.0,255.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]

[43.0,528.0,1564759.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,94.0,10.0,0.11,0.76,0.01,0.0,0.0,0.0,0.0,0.7,0.1]

[24.0,333.0,1462897.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,2.0,2.0,1.0,0.0,0.0,0.5,0.0,0.0,0.0,0.0,0.0,0.0]

[60.0,885.0,1581712.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,30.0,8.0,0.27,0.1,0.03,0.0,0.0,0.0,0.0,0.0,0.0]

[65.0,693.0,2391949.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,75.0,16.0,0.21,0.05,0.01,0.0,0.0,0.0,0.0,0.0,0.0]

```
[60.0,854.0,1519233.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,113.0,34.0,0.3,0.04,0.01,0
.0,0.0,0.0,0.0,0.0,0.0]
[107.0,585.0,2661605.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,171.0,47.0,0.27,0.02,0.01
,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
```

……

……

5.12 小结

可以改成StreamingKmeans，它会根据增量对簇进行更新。官方文档中也只有用Scala写的代码，如果需要找Java的话，可以参考我的另外一个项目中的代码：https://github.com/jiangpz/LearnSpark/blob/master/src/main/java/mllib/StreamingKmeansExample.java

标签：Spark

« 上一篇：4-Spark高级数据分析-第四章 用决策树算法预测森林植被
» 下一篇：Mysql使用mysqldump按时间导出时的一个注意事项

posted on 2016-08-24 17:14 龙猫先生 阅读(1069) 评论(0) 编辑 收藏

» 更多知识库文章...