洛克-李

关注 发私信

访问： 16625次
积分： 456
等级： 博客 2
排名： 千里之外

原创： 29篇 转载： 1篇
译文： 0篇 评论： 6条

**文章搜索**

**文章分类**

| | |
|---|---|
| python初学 | (3) |
| java | (11) |
| 网络问题 | (1) |
| 机器学习 | (10) |
| Spring框架 | (2) |
| spark | (3) |
| 深度学习 | (2) |

**文章存档**

| | |
|---|---|
| 2018年02月 | (2) |
| 2017年07月 | (1) |
| 2017年06月 | (2) |
| 2017年04月 | (2) |
| 2017年03月 | (2) |

展开

**阅读排行**

| | |
|---|---|
| 14.把eclipse中的web项目打... | (4990) |
| Spark-TimeSeries使用方法 | (3566) |
| 针对中科院java接口的使用方... | (1212) |
| python中简单创建一个类 | (1188) |
| 笔记本电脑遇到wifi搜索不到... | (477) |

# Spark-TimeSeries使用方法

标签： 时间序列 spark timeseries holtwinters arima
2017-04-24 17:41 3571人阅读 评论(3) 收藏 举报

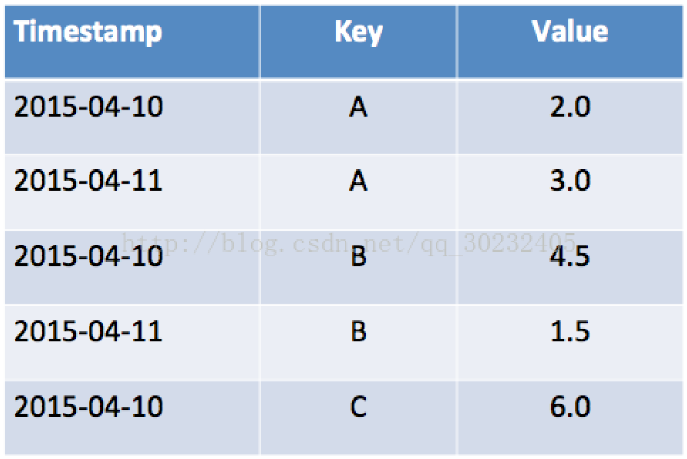
分类：
spark（2）

1.spark里面的库是没有时间序列算法的，但是国外有人已经写好了相应的算法，其github网址是:https://github.com/sryza/spark-timeseries

但基本国内没有太多的资料，所以自己想写一个造福一下后来者。

2.github项目里面的Time-Series Data格式：

（1）假如有如下的数据格式：

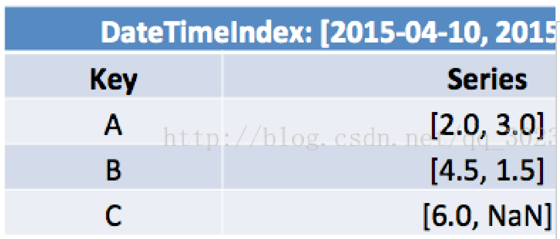


| Timestamp | Key | Value |
|---|---|---|
| 2015-04-10 | A | 2.0 |
| 2015-04-11 | A | 3.0 |
| 2015-04-10 | B | 4.5 |
| 2015-04-11 | B | 1.5 |
| 2015-04-10 | C | 6.0 |

其中timestamp很显然是时间，key我们可以看成是不同公司所测到的一些数据，value就是测到的一些真实数据。

（2）TimeSeriesRDD

要把上面所讲述的数据转换成TimeSeriesRDD格式，其实例如下：



DateTimeIndex: [2015-04-10, 2015

| Key | Series |
|---|---|
| A | [2.0, 3.0] |
| B | [4.5, 1.5] |
| C | [6.0, NaN] |

3.时间序列中的模型。

**最新评论**

Spark-TimeSeries使用方法
qq_20794905 ：你好 在么 有些问题请教

Spark-TimeSeries使用方法
Greed216 ：感谢分享，帮大忙了~~~~

Spark-TimeSeries使用方法
狂人信仰 ：想问下 HoltWinters 是否支持多维values

针对中科院java接口的使用方法和问题
qq_34798140 ：不怎么会玩博客，这个软件在你博客哪里可以下载啊？没找到呢

针对中科院java接口的使用方法和问题
Lei_Gaiceong ：哎呀，你写得太棒了。我好佩服你哦

针对中科院java接口的使用方法和问题
c1261181988 ：NLRIR的路径在lib下。

（1）由于本人用的是该sparkts版本的0.3.0，运行在spark1.6中，其有拥有的时间序列模型不多，有如下：

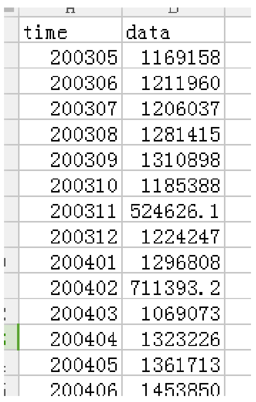
Arima、ARGARCH、EWMA、GARCH、RegressionARIMA

（2）在项目中现在已经更新到了0.4.0，其运行spark版本为2.0以上，如果在1.6以上运行的话则会报错。

相比于0.3.0时间序列模型，0.4.0版本更新了一个算法：holtwinters（这个算法添加上了季节性因素）

## 4.实例

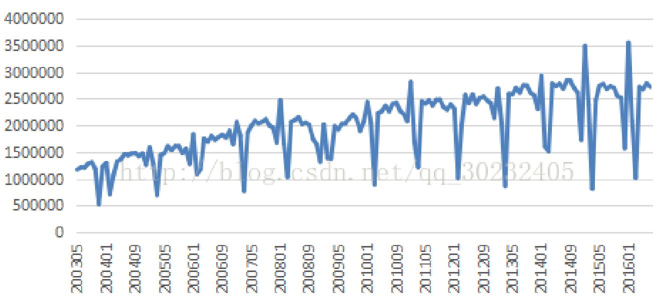
（1）时间序列的训练数据：



| time | data |
|---|---|
| 200305 | 1169158 |
| 200306 | 1211960 |
| 200307 | 1206037 |
| 200308 | 1281415 |
| 200309 | 1310898 |
| 200310 | 1185388 |
| 200311 | 524626.1 |
| 200312 | 1224247 |
| 200401 | 1296808 |
| 200402 | 711393.2 |
| 200403 | 1069073 |
| 200404 | 1323226 |
| 200405 | 1361713 |
| 200406 | 1453850 |

其中time为时间，data为数据。

其图形如下所示：



（2）相比于TimeSeriesRDD格式是少了一个key的格式，所以我在这里先在自己的代码增加一个key的列名。

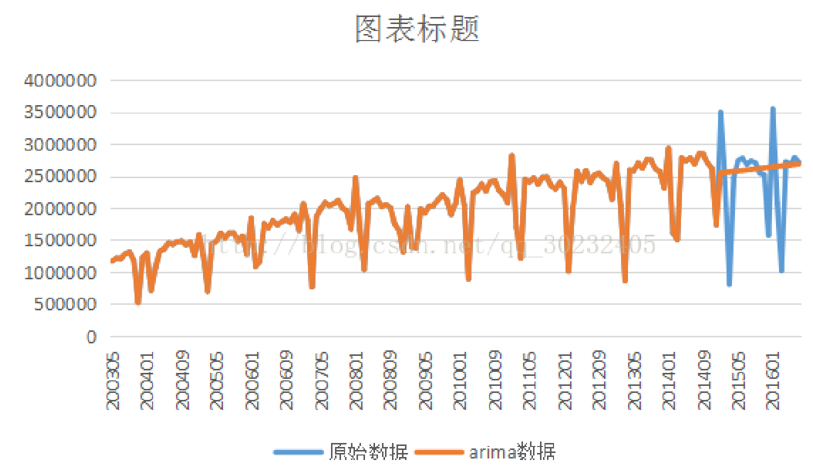
[java]
```java
val timeDataKeyDf=hiveDataDf.withColumn(hiveColumnName(0)+"Key",hiveDataDf(hiveColumnName(
  .select(hiveColumnName(0),hiveColumnName(0)+"Key",hiveColumnName(1))
```

其中hiveDataDf是读出来的dataframe，然后增加了一列

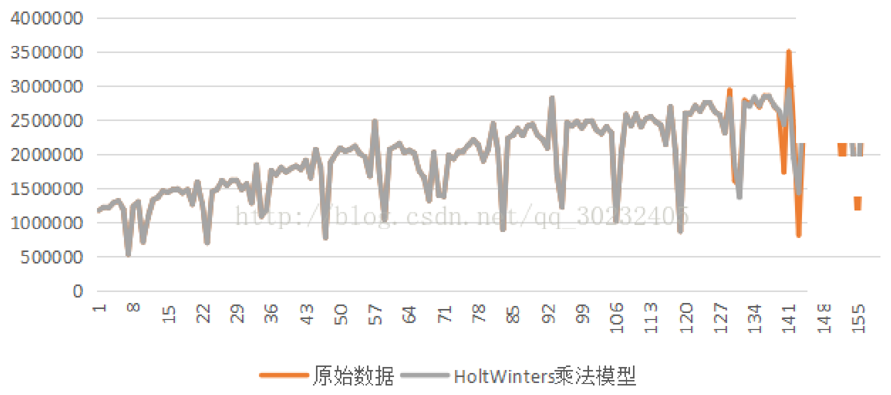
（3）把数据转换成TimeSeriesRDD使，然后建立Arima或者HoltWinters模型，之后进行预测。可到如下图：

Arima的预测图：

关闭



HoltWinters的预测图：



我们从这两个图形上可以看出来，Arima模型没有针对实际的季节性部分来进行预测，所以预测效果比HoltWinters要差。

（4）代码：（详细的项目地址可以看我的csdn_code项目：其地址为：https://code.csdn.net/qq_30232405/spark-timeseries/tree/master）

1）TimeSeriesTrain

```java
[java]
package kingpoint.timeSeries.local

import java.sql.Timestamp
import java.time.{ZoneId, ZonedDateTime}

import com.cloudera.sparkts._
import org.apache.log4j.{Level, Logger}
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types._
import org.apache.spark.sql.{DataFrame, Row, SQLContext}
import org.apache.spark.{SparkConf, SparkContext}

/**
 * 时间序列模型time-series的建立
 * Created by llq on 2017/4/17.
 */
object TimeSeriesTrain {

  /**
   * 把数据中的“time”列转换成固定时间格式：ZonedDateTi
03T10:15:30+01:00 Europe/Paris.）
   * @param timeDataKeyDf
   * @param sqlContext
   * @param hiveColumnName
   * @return zonedDateDataDf
   */
  def timeChangeToDate(timeDataKeyDf:DataFrame,sqlC
{
    var rowRDD:RDD[Row]=sc.parallelize(Seq(Row("")),
    //具体到月份
    if(startTime.length==6){
      rowRDD=timeDataKeyDf.rdd.map{row=>
```

```
    row match{
     case Row(time,key,data)=>{
      val dt = ZonedDateTime.of(time.toString.substring(0,4).toInt,time.toString.sub
      Row(Timestamp.from(dt.toInstant), key.toString, data.toString.toDouble)
     }
    }
   }
  }else if(startTime.length==8){
   //具体到日
   rowRDD=timeDataKeyDf.rdd.map{row=>
    row match{
     case Row(time,key,data)=>{
      val dt = ZonedDateTime.of(time.toString.substring(0,4).toInt,time.toString.sub
      Row(Timestamp.from(dt.toInstant), key.toString, data.toString.toDouble)
     }
    }
   }
  }
  //根据模式字符串生成模式，转化成dataframe格式
  val field=Seq(
   StructField(hiveColumnName(0), TimestampType, true),
   StructField(hiveColumnName(0)+"Key", StringType, true),
   StructField(hiveColumnName(1), DoubleType, true)
  )
  val schema=StructType(field)
  val zonedDateDataDf=sqlContext.createDataFrame(rowRDD,schema)
  return zonedDateDataDf
 }


 /**
  * 总方法调用
  * @param args
  */
 def main(args: Array[String]) {
  /*****环境设置*****/
  //shield the unnecessary log in terminal
  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
  Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

  //set the environment
  System.setProperty("hadoop.home.dir", "D:\\ideaIU\\hadoop-2.2.0-x64-bin\\")
  val conf = new SparkConf().setAppName("kingpoint.timeSeries.local.TimeSeriesTrain").se
  val sc = new SparkContext(conf)
  val sqlContext=new SQLContext(sc)

  /*****参数设置*****/
  //hive中的数据库名字.数据表名
  val databaseTableName="time_series.jxt_electric_month"
  //选择模型(holtwinters或者是arima)
  val modelName="holtwinters"
  //选择要hive的数据表中要处理的time和data列名（输入表中用于训练的列名,必须前面是时间，后面是
data）
  val hiveColumnName=List("time","data")
  //日期的开始和结束，格式为“yyyyMM”或者为“yyyyMMdd”
  val startTime="200305"
  val endTime="201412"
  //预测后面N个值
  val predictedN=19
  //存放的表名字
  val outputTableName="timeseries_outputdate"

  //只有holtWinters才有的参数
  //季节性参数（12或者4）
  val period=12
  //holtWinters选择模型：additive（加法模型）、Multiplicative（乘法模型）
  val holtWintersModelType="Multiplicative"

  /*****读取数据和创建训练数据*****/
//    //read the data form the hive
//    val hiveDataDf=hiveContext.sql("select * from
//      .select(hiveColumnName.head,hiveColumnName.
  val hiveDataDf=sqlContext.load("com.databricks.
> "src/main/resources/data/timeSeriesDate.csv", "he
   .select(hiveColumnName.head,hiveColumnName.ta

  //In hiveDataDF:increase a new column.This colu
  //The reason is:The string column labeling whic
  val timeDataKeyDf=hiveDataDf.withColumn(hiveCol
   .select(hiveColumnName(0),hiveColumnName(0)+"
  val zonedDateDataDf=timeChangeToDate(timeDataKe
```

关闭

```

    /**
     * 创建数据中时间的跨度（Create an daily DateTimeIndex）:开始日期+结束日期+递增数
     * 日期的格式要与数据库中time数据的格式一样
     */
    //参数初始化
    val zone = ZoneId.systemDefault()
    var dtIndex:UniformDateTimeIndex=DateTimeIndex.uniformFromInterval(
      ZonedDateTime.of(2003, 1, 1, 0, 0, 0, 0, zone),
      ZonedDateTime.of(2004, 1, 1, 0, 0, 0, 0, zone),
      new MonthFrequency(1))

    //具体到月份
    if(startTime.length==6) {
      dtIndex = DateTimeIndex.uniformFromInterval(
        ZonedDateTime.of(startTime.substring(0, 4).toInt, startTime.substring(4).toInt, 1,
        ZonedDateTime.of(endTime.substring(0, 4).toInt, endTime.substring(4).toInt, 1, 0,
        new MonthFrequency(1))
    }else if(startTime.length==8){
      //具体到日,则把dtIndex覆盖了
      dtIndex = DateTimeIndex.uniformFromInterval(
        ZonedDateTime.of(startTime.substring(0,4).toInt,startTime.substring(
        ZonedDateTime.of(endTime.substring(0,4).toInt,endTime.substring(4,6)
        new DayFrequency(1))
    }

    //创建训练数据TimeSeriesRDD(key,DenseVector(series))
    val trainTsrdd = TimeSeriesRDD.timeSeriesRDDFromObservations(dtIndex, zonedDateDataDF,
      hiveColumnName(0), hiveColumnName(0)+"Key", hiveColumnName(1))

    /*****建立Modle对象*****/
    val timeSeriesModel=new TimeSeriesModel(predictedN,outputTableName)
    var forecastValue:RDD[String]=sc.parallelize(Seq(""))
    //选择模型
    modelName match{
      case "arima"=>{
        //创建和训练arima模型
        forecastValue=timeSeriesModel.arimaModelTrain(trainTsrdd)
      }
      case "holtwinters"=>{
        //创建和训练HoltWinters模型(季节性模型)
        forecastValue=timeSeriesModel.holtWintersModelTrain(trainTsrdd,period,holtWintersM
      }
      case _=>throw new UnsupportedOperationException("Currently only supports 'ariam' and
    }

    //合并实际值和预测值，并加上日期,形成dataframe(Date,Data)，并保存
    timeSeriesModel.actualForcastDateSaveInHive(trainTsrdd,forecastValue,modelName,predict
  }
}
```

## 2 ) TimeSeriesModel

[java]

```
package kingpoint.timeSeries.local

import java.text.SimpleDateFormat
import java.util.{Calendar, Properties}

import com.cloudera.sparkts.TimeSeriesRDD
import com.cloudera.sparkts.models.ARIMA
import kingpoint.timeSeries.HoltWinters
import org.apache.spark.SparkContext
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.types.{StringType, Stru
import org.apache.spark.sql.{Row, SQLContext, SaveM

import scala.collection.mutable.ArrayBuffer

/**
 * 时间序列模型
 * Created by Administrator on 2017/4/19.
 */
class TimeSeriesModel {

  //预测后面N个值
```

关闭

```
  private var predictedN=1
  //存放的表名字
  private var outputTableName="timeseries_output"

  def this(predictedN:Int,outputTableName:String){
    this()
    this.predictedN=predictedN
    this.outputTableName=outputTableName
  }

  /**
   * Arima模型:
   * 输出其p, d, q参数
   * 输出其预测的predictedN个值
   * @param trainTsrdd
   */
  def arimaModelTrain(trainTsrdd:TimeSeriesRDD[String]): RDD[String] ={
    /***参数设置******/
    val predictedN=this.predictedN

    /***创建arima模型***/
    //创建和训练arima模型.其RDD格式为(ArimaModel,Vector)
    val arimaAndVectorRdd=trainTsrdd.map{line=>
      line match {
        case (key,denseVector)=>
          (ARIMA.autoFit(denseVector),denseVector)
      }
    }

    //参数输出:p,d,q的实际值和其系数值
    val coefficients=arimaAndVectorRdd.map{line=>
      line match{
        case (arimaModel,denseVector)=>{
          (arimaModel.coefficients.mkString(","),
            (arimaModel.p,
             arimaModel.d,
             arimaModel.q))
        }
      }
    }
    coefficients.collect().map{_ match{
      case (coefficients,(p,d,q))=>
        println("coefficients:"+coefficients+"=>"+"(p="+p+",d="+d+",q="+q+")")
    }}

    /***预测出后N个的值*****/
    val forecast = arimaAndVectorRdd.map{row=>
      row match{
        case (arimaModel,denseVector)=>{
          arimaModel.forecast(denseVector, predictedN)
        }
      }
    }
    val forecastValue=forecast.map(_.toArray.mkString(","))

    //取出预测值
    val preditcedValueRdd=forecastValue.map{parts=>
      val partArray=parts.split(",")
      for(i<- partArray.length-predictedN until partArray.length) yield partArray(i)
    }.map(_.toArray.mkString(","))
    preditcedValueRdd.collect().map{row=>
      println("forecast of next "+predictedN+" observations: "+row)
    }
    return preditcedValueRdd
  }

  /**
   *实现HoltWinters模型
   * @param trainTsrdd
   */
  def holtWintersModelTrain(trainTsrdd:TimeSeriesRD
{
    /***参数设置******/
    //往后预测多少个值
    val predictedN=this.predictedN

    /***创建HoltWinters模型***/
    //创建和训练HoltWinters模型.其RDD格式为(HoltWinte
    val holtWintersAndVectorRdd=trainTsrdd.map{line
      line match {
        case (key,denseVector)=>
```

关闭

```
          (HoltWinters.fitModel(denseVector,period,holtWintersModelType),denseVector)
        }
      }

      /***预测出后N个的值*****/
      //构成N个预测值向量，之后导入到holtWinters的forcast方法中
      val predictedArrayBuffer=new ArrayBuffer[Double]()
      var i=0
      while(i<predictedN){
        predictedArrayBuffer+=i
        i=i+1
      }
      val predictedVectors=Vectors.dense(predictedArrayBuffer.toArray)

      //预测
      val forecast = holtWintersAndVectorRdd.map{row=>
        row match{
          case (holtWintersModel,denseVector)=>{
            holtWintersModel.forecast(denseVector, predictedVectors)
          }
        }
      }
      val forecastValue=forecast.map(_.toArray.mkString(","))
      forecastValue.collect().map{row=>
        println("HoltWinters forecast of next "+predictedN+" observations: "+r
      }
      return forecastValue
    }


    /**
     * 批量生成日期（具体到月份的），用来保存
     * @param predictedN
     * @param startTime
     * @param endTime
     */
    def productStartDatePredictDate(predictedN:Int,startTime:String,endTime:String): ArrayBu
  {
      //形成开始start到预测predicted的日期
      var dateArrayBuffer=new ArrayBuffer[String]()
      val dateFormat= new SimpleDateFormat("yyyyMM");
      val cal1 = Calendar.getInstance()
      val cal2 = Calendar.getInstance()

      //设置训练数据中开始和结束日期
      cal1.set(startTime.substring(0,4).toInt,startTime.substring(4).toInt,0)
      cal2.set(endTime.substring(0,4).toInt,endTime.substring(4).toInt,0)

      //开始日期和预测日期的月份差
      val monthDiff = (cal2.getTime.getYear() - cal1.getTime.getYear()) * 12 +
  ( cal2.getTime.getMonth() - cal1.getTime.getMonth())+predictedN
      var iMonth=0
      while(iMonth<=monthDiff){
        //日期加1个月
        cal1.add(Calendar.MONTH, iMonth)
        //保存日期
        dateArrayBuffer+=dateFormat.format(cal1.getTime)
        cal1.set(startTime.substring(0,4).toInt,startTime.substring(4).toInt,0)
        iMonth=iMonth+1
      }
      return dateArrayBuffer
    }

    /**
     * 批量生成日期（具体到日的），用来保存
     * @param predictedN
     * @param startTime
     * @param endTime
     */
    def productStartDayPredictDay(predictedN:Int,star
  {
      //形成开始start到预测predicted的日期
      var dayArrayBuffer=new ArrayBuffer[String]()
      val dateFormat= new SimpleDateFormat("yyyyMMdd"
      val cal1 = Calendar.getInstance()
      val cal2 = Calendar.getInstance()

      //设置训练数据中开始和结束日期
      cal1.set(startTime.substring(0,4).toInt,startTi
  1,startTime.substring(6).toInt)
```

关闭

```
    cal2.set(endTime.substring(0,4).toInt,endTime.substring(4,6).toInt-1,endTime.substring(6).toInt)

    //开始日期和预测日期的月份差
    val dayDiff = (cal2.getTimeInMillis-cal1.getTimeInMillis)/ (1000 * 60 * 60 * 24)+predictedN
    var iDay=0
    while(iDay<=dayDiff){
      //日期加1天
      cal1.add(Calendar.DATE, iDay)
      //保存日期
      dayArrayBuffer+=dateFormat.format(cal1.getTime)
      cal1.set(startTime.substring(0,4).toInt,startTime.substring(4,6).toInt-1,startTime.substring(6).toInt)
      iDay=iDay+1
    }

    return dayArrayBuffer
  }

  /**
   * 合并实际值和预测值，并加上日期,形成dataframe(Date,Data)
   * 并保存在hive中
   * @param trainTsrdd      从hive中读取的数据
   * @param forecastValue   预测出来的数据（分为arima和holtwinters预测的）
   * @param modelName       选择哪个模型名字（arima和holtwinters）
   * @param predictedN      预测多少个值
   * @param startTime       开始日期
   * @param endTime         结束日期
   * @param sc
   * @param hiveColumnName  选择的列名字
   * @param sqlContext
   */
  def actualForcastDateSaveInHive(trainTsrdd:TimeSeriesRDD[String],forecastValue:RDD[Strin
{

    //在真实值后面追加预测值
    val actualAndForcastArray=trainTsrdd.map{line=>
      line match {
        case (key,denseVector)=>
          denseVector.toArray.mkString(",")
      }
    }.union(forecastValue).collect()
    val actualAndForcastSting=
(actualAndForcastArray(0).toString+","+actualAndForcastArray(1).toString).split(",").map(d
(data))
    val actualAndForcastRdd=sc.parallelize(actualAndForcastSting)

    //获取日期，并转换成rdd
    var dateArray:ArrayBuffer[String]=new ArrayBuffer[String]()
    if(startTime.length==6){
      dateArray=productStartDatePredictDate(predictedN,startTime,endTime)
    }else if(startTime.length==8){
      dateArray=productStartDayPredictDay(predictedN,startTime,endTime)
    }
    val dateRdd=sc.parallelize(dateArray.toArray.mkString(",").split(",").map(date=>(date)))

    //合并日期和数据值,形成RDD[Row]
    val dateDataRdd=dateRdd.zip(actualAndForcastRdd).map{
      _ match {
        case (date,data)=>Row(date,data)
      }
    }

    //把dateData转换成dataframe
    val schemaString=hiveColumnName.mkString(" ")
    val schema=StructType(schemaString.split(" ")
      .map(fieldName=>StructField(fieldName,StringT
    val dateDataDf=sqlContext.createDataFrame(dateD

    //加载驱动
    Class.forName("com.mysql.jdbc.Driver")
    //设置用户名和密码
    val prop = new Properties()
    prop.setProperty("user","root")
    prop.setProperty("password","86914381")
    var sqlCommand=""
    //命名表格名字
    dateDataDf.registerTempTable("dateDataDf")
    //编写sql语句
```

关闭

```
    sqlCommand="select * from dateDataDf"
    // 调用DataFrameWriter将数据写入mysql（表可以不存在）
    sqlContext.sql(sqlCommand).write.mode(SaveMode.Append).jdbc("jdbc:mysql://localhost:33

  }
}
```

## 3 ) HoltWinters

[java]

```
/**
 * Copyright (c) 2015, Cloudera, Inc. All Rights Reserved.
 *
 * Cloudera, Inc. licenses this file to you under the Apache License,
 * Version 2.0 (the "License"). You may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * This software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 * CONDITIONS OF ANY KIND, either express or implied. See the License for
 * the specific language governing permissions and limitations under the
 * License.
 */

//package com.cloudera.sparkts.models
package kingpoint.timeSeries

import org.apache.commons.math3.analysis.MultivariateFunction
import org.apache.spark.mllib.linalg._
import org.apache.commons.math3.optim.MaxIter
import org.apache.commons.math3.optim.nonlinear.scalar.ObjectiveFunction
import org.apache.commons.math3.optim.MaxEval
import org.apache.commons.math3.optim.SimpleBounds
import org.apache.commons.math3.optim.nonlinear.scalar.noderiv.BOBYQAOptimizer
import org.apache.commons.math3.optim.InitialGuess
import org.apache.commons.math3.optim.nonlinear.scalar.GoalType

/**
 * Triple exponential smoothing takes into account seasonal changes as well as trends.
 * Seasonality is defined to be the tendency of time-series data to exhibit behavior that repeats
 * itself every L periods, much like any harmonic function.
 *
 * The Holt-Winters method is a popular and effective approach to forecasting seasonal time series
 *
 * See https://en.wikipedia.org/wiki/Exponential_smoothing#Triple_exponential_smoothing
 * for more information on Triple Exponential Smoothing
 * See https://www.otexts.org/fpp/7/5 and
 * https://stat.ethz.ch/R-manual/R-devel/library/stats/html/HoltWinters.html
 * for more information on Holt Winter Method.
 */
object HoltWinters {

  /**
   * Fit HoltWinter model to a given time series. Holt Winter Model has three parameters
   * level, trend and season component of time series.
   * We use BOBYQA optimizer which is used to calculate minimum of a function with
   * bounded constraints and without using derivatives.
   * See http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf for more details.
   *
   * @param ts Time Series for which we want to fit
   * @param period Seasonality of data i.e  period
   * @param modelType Two variations differ in the
   *    Additive method is preferred when seasonal
   *    Multiplicative method is preferred when the
   *    proportional to the level of the series.
   * @param method: Currently only BOBYQA is suppor
   */
  def fitModel(ts: Vector, period: Int, modelType:
  : HoltWintersModel = {
    method match {
      case "BOBYQA" => fitModelWithBOBYQA(ts, perio
      case _ => throw new UnsupportedOperationExcep
    }
  }
```

关闭

```scala

  def fitModelWithBOBYQA(ts: Vector, period: Int, modelType:String): HoltWintersModel = {
    val optimizer = new BOBYQAOptimizer(7)
    val objectiveFunction = new ObjectiveFunction(new MultivariateFunction() {
      def value(params: Array[Double]): Double = {
        new HoltWintersModel(modelType, period, params(0), params(1), params(2)).sse(ts)
      }
    })

    // The starting guesses in R's stats:HoltWinters
    val initGuess = new InitialGuess(Array(0.3, 0.1, 0.1))
    val maxIter = new MaxIter(30000)
    val maxEval = new MaxEval(30000)
    val goal = GoalType.MINIMIZE
    val bounds = new SimpleBounds(Array(0.0, 0.0, 0.0), Array(1.0, 1.0, 1.0))
    val optimal = optimizer.optimize(objectiveFunction, goal, bounds,initGuess, maxIter, m
    val params = optimal.getPoint
    new HoltWintersModel(modelType, period, params(0), params(1), params(2))
  }
}

class HoltWintersModel(
    val modelType: String,
    val period: Int,
    val alpha: Double,
    val beta: Double,
    val gamma: Double) extends TimeSeriesModel {

  if (!modelType.equalsIgnoreCase("additive") && !modelType.equalsIgnoreCase(
    throw new IllegalArgumentException("Invalid model type: " + modelType)
  }
  val additive = modelType.equalsIgnoreCase("additive")

  /**
   * Calculates sum of squared errors, used to estimate the alpha and beta parameters
   *
   * @param ts A time series for which we want to calculate the SSE, given the current par
   * @return SSE
   */
  def sse(ts: Vector): Double = {
    val n = ts.size
    val smoothed = new DenseVector(Array.fill(n)(0.0))
    addTimeDependentEffects(ts, smoothed)

    var error = 0.0
    var sqrErrors = 0.0

    // We predict only from period by using the first period - 1 elements.
    for(i <- period to (n - 1)) {
      error = ts(i) - smoothed(i)
      sqrErrors += error * error
    }

    sqrErrors
  }

  /**
   * {@inheritDoc}
   */
  def removeTimeDependentEffects(ts: Vector, dest: Vector = null): Vector = {
    throw new UnsupportedOperationException("not yet implemented")
  }

  /**
   * {@inheritDoc}
   */
  def addTimeDependentEffects(ts: Vector, dest: Vec
    val destArr = dest.toArray
    val fitted = getHoltWintersComponents(ts)._1
    for (i <- 0 to (dest.size - 1)) {
      destArr(i) = fitted(i)
    }
    dest
  }

  /**
   * Final prediction Value is sum of level trend a
   * But in R's stats:HoltWinters additional weight
   *
   * @param ts
   * @param dest
```

关闭

```scala
 */
def forecast(ts: Vector, dest: Vector): Vector = {
  val destArr = dest.toArray
  val (_, level, trend, season) = getHoltWintersComponents(ts)
  val n = ts.size

  val finalLevel = level(n - period)
  val finalTrend = trend(n - period)
  val finalSeason = new Array[Double](period)

  for (i <- 0 until period) {
    finalSeason(i) = season(i + n - period)
  }

  for (i <- 0 until dest.size) {
    destArr(i) = if (additive) {
      (finalLevel + (i + 1) * finalTrend) + finalSeason(i % period)
    } else {
      (finalLevel + (i + 1) * finalTrend) * finalSeason(i % period)
    }
  }
  dest
}

/**
 * Start from the intial parameters and then iterate to find the final parameters
 * using the equations of HoltWinter Method.
 * See https://www.otexts.org/fpp/7/5 and
 * https://stat.ethz.ch/R-manual/R-devel/library/stats/html/HoltWinters.htm
 * for more information on Holt Winter Method equations.
 *
 * @param ts A time series for which we want the HoltWinter parameters leve
 * @return (level trend season). Final vectors of level trend and season are returned.
 */
def getHoltWintersComponents(ts: Vector): (Vector, Vector, Vector, Vector) = {
  val n = ts.size
  require(n >= 2, "Requires length of at least 2")

  val dest = new Array[Double](n)

  val level = new Array[Double](n)
  val trend = new Array[Double](n)
  val season = new Array[Double](n)

  val (initLevel, initTrend, initSeason) = initHoltWinters(ts)
  level(0) = initLevel
  trend(0) = initTrend
  for (i <- 0 until initSeason.size){
    season(i) = initSeason(i)
  }

  for (i <- 0 to (n - period - 1)) {
    dest(i + period) = level(i) + trend(i)

    // Add the seasonal factor for additive and multiply for multiplicative model.
    if (additive) {
      dest(i + period) += season(i)
    } else {
      dest(i + period) *= season(i)
    }

    val levelWeight = if (additive) {
      ts(i + period) - season(i)
    } else {
      ts(i + period) / season(i)
    }

    level(i + 1) = alpha * levelWeight + (1 - alp

    trend(i + 1) = beta * (level(i + 1) - level(i

    val seasonWeight = if (additive) {
      ts(i + period) - level(i + 1)
    } else {
      ts(i + period) / level(i + 1)
    }
    season(i + period) = gamma * seasonWeight + (
  }

  (Vectors.dense(dest), Vectors.dense(level), Vec
}
```

关闭

```

  def getKernel(): (Array[Double]) = {
    if (period % 2 == 0){
      val kernel = Array.fill(period + 1)(1.0 / period)
      kernel(0) = 0.5 / period
      kernel(period) = 0.5 / period
      kernel
    } else {
      Array.fill(period)(1.0 / period)
    }
  }

  /**
   * Function to calculate the Weighted moving average/convolution using above kernel/weig
   * for input data.
   * See http://robjhyndman.com/papers/movingaverage.pdf for more information
   * @param inData Series on which you want to do moving average
   * @param kernel Weight vector for weighted moving average
   */
  def convolve(inData: Array[Double], kernel: Array[Double]): (Array[Double]
    val kernelSize = kernel.size
    val dataSize = inData.size

    val outData = new Array[Double](dataSize - kernelSize + 1)

    var end = 0
    while (end <= (dataSize - kernelSize)) {
      var sum = 0.0
      for (i <- 0 until kernelSize) {
        sum += kernel(i) * inData(end + i)
      }

      outData(end) = sum
      end += 1
    }

    outData
  }

  /**
   * Function to get the initial level, trend and season using method suggested in
   * http://robjhyndman.com/hyndsight/hw-initialization/
   * @param ts
   */
  def initHoltWinters(ts: Vector): (Double, Double, Array[Double]) = {
    val arrTs = ts.toArray

    // Decompose a window of time series into level trend and seasonal using convolution
    val kernel = getKernel()
    val kernelSize = kernel.size
    val trend = convolve(arrTs.take(period * 2), kernel)

    // Remove the trend from time series. Subtract for additive and divide for multiplicat
    val n = (kernelSize -1) / 2
    val removeTrend = arrTs.take(period * 2).zip(
      Array.fill(n)(0.0) ++ trend ++ Array.fill(n)(0.0)).map{
      case (a, t) =>
        if (t != 0){
          if (additive) {
            (a - t)
          } else {
            (a / t)
          }
        }  else{
          0
        }
    }

    // seasonal mean is sum of mean of all season v
    val seasonalMean = removeTrend.splitAt(period).
      if (prevx == 0 || x == 0) (x + prevx) else (x
    }

    val meanOfFigures = seasonalMean.sum / period

    // The seasonal mean is then centered and remov
    // Subtract for additive and divide for multipl
    val initSeason = if (additive) {
      seasonalMean.map(_ - meanOfFigures )
    } else {
      seasonalMean.map(_ / meanOfFigures )
```

关闭

```
    }

    // Do Simple Linear Regression to find the initial level and trend
    val indices = 1 to trend.size
    val xbar = (indices.sum: Double) / indices.size
    val ybar = trend.sum / trend.size

    val xxbar = indices.map( x => (x - xbar) * (x - xbar) ).sum
    val xybar = indices.zip(trend).map {
      case (x, y) => (x - xbar) * (y - ybar)
    }.sum

    val initTrend = xybar / xxbar
    val initLevel = ybar - (initTrend * xbar)

    (initLevel, initTrend, initSeason)
  }
}
```

- 上一篇 Spark中遇到的一些问题和相应的解决办法
- 下一篇 spark-BigDl:深度学习之神经网络编写

踩
1

您还没有登录,请[登录]或[注册]

**查看评论**

qq_20794905 3楼 2017-08-15 14:06发表
你好 在么 有些问题请教

Greed216 2楼 2017-07-06 11:44发表
感谢分享，帮大忙了~~~~

狂人信仰 1楼 2017-05-22 17:31发表
想问下 HoltWinters 是否支持多维values

**时间序列分析--ARIMA模型** u01352
http://blog.csdn.net/u010414589/article/details/49622625
相关性没有要求。但是，如果你想使用指数平滑法计算出预测区
须是服从零均值、 方差不变的正态分布。即使指数平滑法...

**基于spark用线性回归（linear regression)**
ubuntu+spark+scala实现线性回归（linear regressio wtt5
n）算法（代码+数据）

关闭

**ARIMA模型** u010159842 2017-06-07 10:16 833

ARIMA模型 自回归移动平均模型(Autoregressive Integrated Moving Average Model,简记ARIMA) 目录 [隐藏] 1 什么是ARIMA模型?2 ARIMA模型的基本思想3 ARIMA模型预测的基本程序4 相关链...

**时间序列分析** webzjuyujun 2015-12-21 18:12 3898

时间序列,自相关性,arma,arima

**spark移动平均** iDuanyingjie 2017-02-27 14:55 414

需求：计算在一个特定时间窗口内各个日期不同股票代码的不同收盘价的移动平均数 “移动” 着新的时间序列数据的到来，要不断的重新计算这个平均值，由于会删除最早的值同时增加 均值会相应的 “移动” 股票代码 时间 收盘价 移动平均 Apple 2015-12-11 ...

**Spark的使用方法（二）** qingqing7 2017-12-05 13:52 102

Spark使用过程中报错汇总报错1： ValueError: Cannot run multiple SparkContexts at once; existing SparkContext(app=PySparkShell, master=local[*]) 原因：出现这个错误是因为之前已经启...

**spark sql 获取系统时间，计算时间后结果浮点型转整型** feloxx 2017-05-31 13:38 2785

spark sql 获取当前系统时间 org.apache.spark.sql.AnalysisException: Undefined function: 'getdate'

**Spark中遇到的一些问题和相应的解决办法** qq_30232405 2017-04-11 16:13 469

前言：最近在公司实习，用到了spark的应用。说真的很多spark出现的问题在百度上基本搜索不到相应的结果，还是在google上容易搜到解决办法。（安利一下翻墙插件：xx-net）1.在spark中读取csv文件 介绍如何使用Spark 1.3+的外部数据源接口来自定义CSV输入格...

**基于WEKA实现时间序列的预测** sparkexpert 2016-04-07 23:02 3834

时间序列预测是根据客观事物发展的规律性，运用历史数据来推测未来的发展趋势。 时序预测是一项应用非常广的技术，如股票预测，天气预测等。 然而时序预测也是一项比较难的地方，主要是短期预测可能还比较准，而对一段时间的预测则会比较难。 在学习时序预测过程中，先看了WEKA的功能...

**python利用LSTM进行时间序列分析预测** a81982

关键词：python、Keras、LSTM、Time-Series-Prediction 关于理论部分，可以参考这两篇文章（RNN、LSTM），本文主要从数据、代码角度，利用LSTM 数据（只列出了18行）1455.219971 1399.420044...

关闭

**python+ARIMA 进行时间序列处理** wtt561111 2016-05-20 15:01 8918

利用ARIMA进行时间序列处理

**JFreeChart学习(一)-JfreeChart中的timeSeries(时序图)使用示例**

转自：http://www.open-open.com/lib/view/ope lvyuan30276 2016-07-13 18:19 2744
n1400202235004.html 生成时序图： JFreeChart c
hart = ChartFactory.createTimeSeriesChart( St...

**pandas小记：pandas时间序列分析和处理Timeseries**

http://blog.csdn.net/pipisorry/article/details/52209 pipisorry 2016-08-15 15:3
377pandas 最基本的时间序列类型就是以时间戳（Tim
eStamp）为 index 元素的 Series 类型。其它时间序列处理相关的包[P4J 0.6: Periodic ligh

**jfreechart采用TimeSeriesChart并更改热点内容**

/** * */ package com.huaxia.bank.test; import java.awt. cuiran 2012-03-12 14:01 4958
Color; import java.awt.Dimension; import java.awt.Fon
t; import java.awt.Toolkit; i...

**spark sql 处理时间类型** feloxx 2017-05-18 16:31 5971

spark sql 处理时间类型 1、时间类型应用 2、时间类型运算 3、时间类型小时级别运算

**ARIMA模型** fenghuangdesire 2015-06-02 09:09 10161

时间序列分析分为两大类：频域分析和时域分析。频域分析也称为谱分析，是一种非常有用的纵向数据分析方法。时域分析主要关心从序列值之间的相关关系对时间序列发展规律。 在时域分析里，生成时间序列数据的随机过程按照统计规律的特征是否随着时间变化而变化分为两类，如果随机过程的特征随着时间变化，如GDP的时间序...

**时间序列的分析和预测ARIMA** u014281392 2017-08-25 21:47 1016

分析的数据来自一个kaggle的比赛数据，是一组维基百科页面的浏览量数据，对数据进行简单的分析和处理，预测未来的流量．数据包含部分网页从2015年７月１日到２
据，数据有存在缺失，网页的类型包含多个语种． 下面是数据的

**基于Spark技术实现大规模时间序列异常检测**

最近一直忙于异常检测项目的上线，一直没有时间来更 an
新博客，该系统已经在大规模时间序列场景稳定运行1个
多月，简单总结一下。 达到的目标，通过Spark对3万个服务器进
指标对应一个时间序列，模型全量训练用21个Core耗

**Holt-Winters-季节性预测算法** u013704227 2017-08-17 15:57 950

参考 Holt-Winters seasonal method Holt Winter 指数平滑模型

**python+神经网络实现时间序列预测** wtt561111 2017-03-26 15:31 6424

利用python语言进行时间序列处理

关闭