

时间序列预测法及Spark-TimeSerial实现



raincoffee (/u/a62849f621bf) + 关注

2017.06.22 19:09* 字数 3266 阅读 2234 评论 9 喜欢 4

(/u/a62849f621bf)

时间序列预测法及Spark-Timeserial

时间序列预测法

时间序列预测法(Time Series Forecasting Method)

什么是时间序列预测法？

一种历史资料延伸预测，也称历史引伸预测法。是以时间数列 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%97%B6%E9%97%B4%E6%95%B0%E5%88%97>) 所能反映的社会经济现象的发展过程和规律性，进行引伸外推，预测其发展趋势的方法。

时间序列，也叫时间数列、历史复数或动态数列 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E5%8A%A8%E6%80%81%E6%95%B0%E5%88%97>)。它是将某种统计指标 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%BB%9F%E8%AE%A1%E6%8C%87%E6%A0%87>) 的数值，按时间先后顺序排到所形成的数列。时间序列预测法就是通过编制和分析时间序列，根据时间序列所反映出来的发展过程、方向和趋势，进行类推或延伸，借以预测下一段时间或以后若干年内可能达到的水平。其内容包括：收集与整理某种社会现象的历史资料；对这些资料进行检查鉴别，排成数列；分析时间数列，从中寻找该社会现象随时间变化而变化的规律，得出一定的模式；以此模式去预测该社会现象将来的情况。

时间序列预测法的步骤

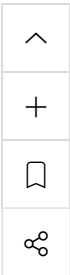
第一步 收集历史资料，加以整理，编成时间序列，并根据时间序列绘成统计图 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%BB%9F%E8%AE%A1%E5%9B%BE>)。时间序列分析通常是把各种可能发生作用的因素进行分类，传统的分类方法是按各种因素的特点或影响效果分为四大类：(1)长期趋势；(2)季节变动；(3)循环变动 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E5%BE%AA%E7%8E%AF%E5%8F%98%E5%8A%A8>)；(4)不规则变动。

第二步 分析时间序列。时间序列中的每一时期的数值都是由许许多多不同的因素同时发生作用后的综合结果。

第三步 求时间序列的长期趋势(T)季节变动(s)和不规则变动(I)的值，并选定近似的数学模式来代表它们。对于数学模式中的诸未知参数，使用合适的技术方法求出其值。

第四步 利用时间序列资料求出长期趋势、季节变动和不规则变动的数学模型后，就可以利用它来预测未来的长期趋势 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E8%B6%8B%E5%8A%BF>) 值T和季节变动值s，在可能的情况下预测不规则变动值I。然后用以下模式计算出未来的时间序列的预测值Y：

加法模式T+S+I=Y



乘法模式 $T \times S \times I = Y$

如果不规则变动的预测值难以求得，就只求长期趋势 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E8%B6%8B%E5%8A%BF>) 和季节变动的预测值，以两者相乘之积或相加之和为时间序列的预测值。如果经济现象本身没有季节变动或不需预测分季分月的资料，则长期趋势的预测值就是时间序列的预测值，即 $T=Y$ 。但要注意这个预测值只反映现象未来的发展趋势，即使很准确的趋势线 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E8%B6%8B%E5%8A%BF%E7%BA%BF>) 在按时间顺序的观察方面所起的作用，本质上也只是是一个平均数 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E5%B9%B3%E5%9D%87%E6%95%B0>) 的作用，实际值将围绕着它上下波动。

时间序列分析基本特征

1. 时间序列分析法是根据过去的变化趋势预测未来的发展,它的前提是假定事物的过去延续到未来。

时间序列分析,正是根据客观事物发展的连续规律性,运用过去的历史数据,通过统计分析,进一步推测未来的发展趋势。事物的过去会延续到未来这个假设前提包含两层含义:一是不会发生突然的跳跃变化,是以相对小的步伐前进;二是过去和当前的现象可能表明现在和将来活动的发展变化趋向。这就决定了在一般情况下,时间序列分析法对于短、近期预测比较显著,但如延伸到更远的将来,就会出现很大的局限性,导致预测值偏离实际较大而使决策失误。

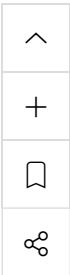
2. 时间序列数据变动存在着规律性与不规律性

时间序列中的每个观察值大小,是影响变化的各种不同因素在同一时刻发生作用的综合结果。从这些影响因素发生作用的大小和方向变化的时间特性来看,这些因素造成的时间序列数据的变动分为四种类型。

- (1)趋势性:某个变量随着时间进展或自变量变化,呈现一种比较缓慢而长期的持续上升、下降、停留的同性质变动趋向,但变动幅度可能不相等。
- (2)周期性:某因素由于外部影响随着自然季节的交替出现高峰与低谷的规律。
- (3)随机性:个别为随机变动,整体呈统计规律。
- (4)综合性:实际变化情况是几种变动的叠加或组合。预测时设法过滤除去不规则变动,突出反映趋势性和周期性变动。

时间序列预测法的分类

时间序列预测法可用于短期预测 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%9F%AD%E6%9C%9F%E9%A2%84%E6%B5%8B>)、中期预测 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E4%B8%AD%E6%9C%9F%E9%A2%84%E6%B5%8B>) 和长期预测 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E9%A2%84%E6%B5%8B>)。根据对资料分析方法的不同,又可分为:简单序时平均数法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%AE%80%E5%8D%95%E5%BA%8F%E6%97%B6%E5%B9%B3%E5%9D%87%E6%95%B0%E6%B3%95>)、加权序时平均数法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/w/index.php?title=%E5%8A%A0%E6%9D%83%E5%BA%8F%E6%97%B6%E5%B9%B3%E5%9D%87%E6%95%B0%E6%B3%95&action=edit>)、移动平均法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%A7%BB%E5%8A%A8%E5%B9%B3%E5%9D%87%E6%B3%95>)、加权移动平均法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E5%8A%A0%E6%9D%83%E7%A7%BB%E5%8A%A8>)



%E5%B9%B3%E5%9D%87%E6%B3%95)、趋势预测法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E8%B6%8B%E5%8A%BF%E9%A2%84%E6%B5%8B%E6%B3%95>)、指数平滑法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%8C%87%E6%95%B0%E5%B9%B3%E6%BB%91%E6%B3%95>)、季节性趋势预测法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E5%AD%A3%E8%8A%82%E6%80%A7%E8%B6%8B%E5%8A%BF%E9%A2%84%E6%B5%8B%E6%B3%95>)、市场寿命周期预测法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/w/index.php?title=%E5%B8%82%E5%9C%BA%E5%AF%BF%E5%91%BD%E5%91%A8%E6%9C%9F%E9%A2%84%E6%B5%8B%E6%B3%95&action=edit>)等。

简单序时平均数法 也称算术平均法 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%AE%97%E6%9C%AF%E5%B9%B3%E5%9D%87%E6%B3%95>)。即把若干历史时期的统计数值作为观察值，求出算术平均数作为下期预测值。这种方法基于下列假设：“过去这样，今后也将这样”，把近期和远期数据等同化和平均化，因此只能适用于事物变化不大的趋势预测。如果事物呈现某种上升或下降的趋势，就不宜采用此法。

加权序时平均数法 就是把各个时期的历史数据按近期和远期影响程度进行加权，求出平均值，作为下期预测值。

简单移动平均法 就是相继移动计算若干时期的算术平均数作为下期预测值。

加权移动平均法 即将简单移动平均数进行加权计算。在确定权数时，近期观察值的权数应该大些，远期观察值的权数应该小些。

上述几种方法虽然简便，能迅速求出预测值，但由于没有考虑整个社会经济发展的新动向和其他因素的影响，所以准确性较差。应根据新的情况，对预测结果作必要的修正。

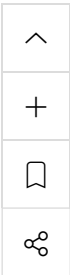
指数平滑法 即根据历史资料的上期实际数和预测值，用指数加权的办法进行预测。此法实质是由内加权移动平均法演变而来的一种方法，优点是只要有上期实际数和上期预测值，就可计算下期的预测值，这样可以节省很多数据和处理数据的时间，减少数据的存储量，方法简便。是国外广泛使用的一种短期预测 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E7%9F%AD%E6%9C%9F%E9%A2%84%E6%B5%8B>)方法。

季节趋势预测法 根据经济事物每年重复出现的周期性季节变动指数，预测其季节性变动趋势。推算季节性指数可采用不同的方法，常用的方法有季(月)别平均法和移动平均法两种：a．季(月)别平均法。就是把各年度的数值分季(或月)加以平均，除以各年季(或月)的总平均数，得出各季(月)指数。这种方法可以用来分析生产、销售 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E9%94%80%E5%94%AE>)、原材料储备、预计资金周转 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E5%91%A8%E8%BD%AC>)需要量等方面的经济事物的季节性变动；b．移动平均法。即应用移动平均数计算比例求典型季节指数。

市场寿命周期预测法 就是对产品市场寿命周期的分析研究。例如对处于成长期的产品预测其销售量，最常用的一种方法就是根据统计资料，按时间序列画成曲线图 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%9B%B2%E7%BA%BF%E5%9B%BE>)，再将曲线外延，即得到未来销售发展趋势。最简单的外延方法是直线外延法，适用于对耐用消费品 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E8%80%90%E7%94%A8%E6%B6%88%E8%B4%B9%E5%93%81>)的预测。这种方法简单、直观、易于掌握。

时间序列预测法

1.逐步自回归(StepAR)模型：StepAR模型是有趋势、季节因素数据的模型类。



2.Winters Method—Additive模型：它是将时势 (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%97%B6%E5%8A%BF>)和乘法季节因素相结合，考虑序列中有规律节波动。

3.ARIMA模型：它是处理带有趋势、季节因平稳随机项数据的模型类[3] (https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%97%B6%E9%97%B4%E5%BA%8F%E5%88%97%E9%A2%84%E6%B5%8B%E6%B3%95#_note-2)。

4.Winters Method—Muhiplicative模型：该方将时间趋势和乘法季节因素相结合，考虑序列规律的季节波动。时间趋势模型可根据该序列律的季节波动对该趋势进行修正。为了能捕捉到季节性，趋势模型包含每个季节的一个季节参季节因子采用乘法季节因子。随机时间序列

$$x_t(t=1,2,\dots,N,N=n)$$

整理汇总历史上各类保险的数据得到逐月的数据，Winters Method-Multiplicative模型表示为

$$x_t = (a + bt)s(t) + \varepsilon_t \quad (1)$$

其中a和b为趋势参数，s(t)为对应于时刻t的这个季节选择的季节参数，修正方程为。

$$a_t = \omega_1 \frac{x_t}{s(t-1)} + (1 - \omega_1)(a_{t-1} + b_{t-1})$$

$$a_t = \omega_1 \frac{x_t}{s(t-1)} + (1 - \omega_1)(a_{t-1} + b_{t-1})$$

,

$$b_t = \omega_2(a_t - a_{t-1}) + (1 - \omega_2)b_{t-1} \quad (2)$$

其中： x_t, a_t, b_t 分别为序列在时刻t的实测值、平滑值和平滑趋势s(t-1)(t)选择在季节因子被修正之前对应于时刻t的季节因子的过去值。

在该修正系统中，趋势多项式在当前周期中总是被中心化，以便在t以后的时间里预报值的趋势多项式的截距参数总是修正后的截距参数 a_t 。向前 τ 个周期的预报值是。

$$x_{t+\tau} = (a_t + b_t\tau)s(t+\tau) \quad (3)$$

当季节在数据中改变时季节参数被修正，它使用季节实测值与预报值比率的平均值。

5.GARCH (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/GARCH>)(ARCH)模型

带自相关扰动的回归模型为。

$$x_t = \xi_t \beta + v_t,$$

$$v_t = \varepsilon_t = \varphi_1 v_{t-1} + \dots + \varphi_n v_{t-n}$$

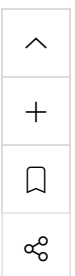
$$v_t = \varepsilon_t = \varphi_1 v_{t-1} + \dots + \varphi_n v_{t-n}$$

,

$$\varepsilon_t = N(0, \sigma^2) \quad (4)$$

其中： x_t 为因变量； ξ_t 为回归因子构成的列向量； β 为结构参数构成的列向量； ε_t 为均值是0、方差是一的独立同分布正态随机变量。

服从GARCH过程的序列 x_t ，对于t时刻X的条件分布记为



$$x_t | \varphi_t \sim 1^N(0, h_t) \quad (5)$$

其中\phi_{t-1}表示时间t-1前的所有可用信息，条件方差。

$$h_t = \tilde{\omega} + \sum_{i=1}^p \alpha_i x_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

$$h_t = \tilde{\omega} + \sum_{i=1}^p \alpha_i x_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

(6)。

其中p≥0,q>0,

$$\tilde{\omega} \geq 0$$

$$\tilde{\omega} \geq 0$$

$$\gamma_j \geq 0$$

$$\gamma_j \geq 0$$

当p=0时，GARCH(p,q)模型退化为ARCH(p)模型，ARCH参数至少要有一个不为0。

GARCH回归模型可写成

$$x_t = \xi_t' \beta + \epsilon_t, \epsilon_t = \sqrt{h_t} e_t$$

$$x_t = \xi_t' \beta + \epsilon_t, \epsilon_t = \sqrt{h_t} e_t$$

$$h_t = \tilde{\omega} + \sum_{i=1}^p \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

$$h_t = \tilde{\omega} + \sum_{i=1}^p \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

$$e_t \sim N(0,1) \quad (7)$$

也可以考虑服从自回归过程的扰动或带有GARCH误差的模型，即AR(n)-GARCH(p,q)。

$$x_t = \xi_t' \beta + v_t$$

$$x_t = \xi_t' \beta + v_t$$

$$v_t = \epsilon_t - \varphi_1 v_{t-1} - \dots - \varphi_n v_{t-n}$$

$$v_t = \epsilon_t - \varphi_1 v_{t-1} - \dots - \varphi_n v_{t-n}$$

$$\epsilon_t = \sqrt{h_t} e_t$$

$$\epsilon_t = \sqrt{h_t} e_t$$

$$h_t = \xi_t' \beta + \sum_{i=1}^p \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

$$h_t = \xi_t' \beta + \sum_{i=1}^p \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^q \gamma_j h_{t-j}$$

(8)

其中三次平滑指数 (HoltWinters) <http://www.dataguru.cn/article-3235-1.html>
(<https://link.jianshu.com?t=http://www.dataguru.cn/article-3235-1.html>)

该部分详细介绍见

<http://wiki.mbalib.com/wiki/%E6%97%B6%E9%97%B4%E5%BA%8F%E5%88%97%E9%A2%84%E6%B5%8B%E6%B3%95> (<https://link.jianshu.com?t=http://wiki.mbalib.com/wiki/%E6%97%B6%E9%97%B4%E5%BA%8F%E5%88%97%E9%A2%84%E6%B5%8B%E6%B3%95>)

Spark-TimeSerial

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/> (<https://link.jianshu.com?t=https://github.com/sryza/>)**spark-timeseries**

spark-timeserial介绍：<https://yq.aliyun.com/articles/70292> (<https://link.jianshu.com?t=https://yq.aliyun.com/articles/70292>)

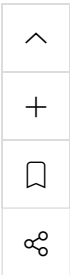
实例：http://blog.csdn.net/qq_30232405/article/details/70622400
(https://link.jianshu.com?t=http://blog.csdn.net/qq_30232405/article/details/70622400)

实际应用。

数据格式（处理过后的）：每5分钟一个值；

	time	key	ct
	2017-01-01 00:00:...	key	2.2054527E7
	2017-01-01 00:05:...	key	2.1454076E7
	2017-01-01 00:10:...	key	2.1401227E7
	2017-01-01 00:15:...	key	2.1246488E7
	2017-01-01 00:20:...	key	2.1128788E7
	2017-01-01 00:25:...	key	2.1079653E7
	2017-01-01 00:30:...	key	2.0890998E7
	2017-01-01 00:35:...	key	2.0583514E7
	2017-01-01 00:40:...	key	2.026607E7
	2017-01-01 00:45:...	key	1.9944556E7
	2017-01-01 00:50:...	key	1.953332E7
	2017-01-01 00:55:...	key	1.9086006E7
	2017-01-01 01:00:...	key	1.8718264E7
	2017-01-01 01:05:...	key	1.8172172E7
	2017-01-01 01:10:...	key	1.7635609E7
	2017-01-01 01:15:...	key	1.708388E7
	2017-01-01 01:20:...	key	1.6584532E7

预测代码：



```

/**
 * Created by ${lyp} on 2017/6/21.
 */

case class PV(time:String,key:String,ct: Double );

object RunModel {
  val starttime="2017-01-01 00:00:00"
  val endtime= "2017-05-31 23:55:00"
  val predictedN=288
  val outputTableName=""
  val modelName="holtwinters"
  val sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
  val hiveColumnName=List("time","key","ct")

  def main(args: Array[String]): Unit = {

    Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    val conf= new SparkConf().setAppName("timeserial").setMaster("local")
    val sc= new SparkContext(conf)
    val sqlContext=new SQLContext(sc)

    import sqlContext.implicits._

    //create dataframe
    val trainData=getData(sc,sqlContext,"src/main/resource/data.csv")
    //val verifyData=getData(sc,sqlContext,"src/main/resource/data2.csv")
    trainData.show()
    //verifyData.show()

    //create DateTimeIndex
    val zone = ZoneId.systemDefault()
    var dtIndex:UniformDateTimeIndex=DateTimeIndex.uniformFromInterval(
      ZonedDateTime.of(2017, 1, 1, 0, 0, 0, zone),
      ZonedDateTime.of(2017, 5, 31, 23, 55, 0, zone),
      new MinuteFrequency(5)
    )

    //create TimeSeriesRDD
    val trainTsrd=TimeSeriesRDD.timeSeriesRDDFromObservations(dtIndex,trainData,hiveColumnName)
    trainTsrd.foreach(println(_))

    //cache
    trainTsrd.cache()

    //add absent value "linear", "nearest", "next", or "previous"
    val filledTrainTsrd=trainTsrd.fill("linear")

    //create model
    val timeSeriesModel= new TimeSeriesModel(predictedN)

    //train model
    val forecast=modelName match {
      case "arima"=>{
        println("begin train")
        val (forecast,coefficients)=timeSeriesModel.arimaModelTrain(filledTrainTsrd)
        forecast
      }
      case "holtwinters"=>{
        //季节性参数（12或者4）
        val period=288*30
        //holtWinters选择模型: additive（加法模型）、Multiplicative（乘法模型）
        val holtWintersModelType="Multiplicative"
        val (forecast,sse)=timeSeriesModel.holtWintersModelTrain(filledTrainTsrd,period,holtWintersModelType)
        forecast
      }
      case _=>throw new UnsupportedOperationException("Currently only supports 'arima' and 'holtwinters'")
    }

    val time=timeSeriesModel.productStartDatePredictDate(predictedN,endtime,endtime)

    forecast.map{
      row=>
        val key=row._1
        val values=row._2.toArray.mkString(",")
        (key,values)
    }.flatMap(row=>row._2.split(",")).saveAsTextFile("src/main/resource/30Multiplicative")
  }
}

```



```
def getTrainData(sqlContext:SQLContext):DataFrame={
  val data=sqlContext.sql(
    s"""
      |select time, 'key' as key, ct from tmp_music.tmp_lyp_nginx_result_ct2 where time
      |""".stripMargin)
  data
}

def getData(sparkContext: SparkContext,sqlContext:SQLContext,path:String):DataFrame={
  val data=sparkContext.textFile(path).map(line=>line.split(",")).map{
    line=>
      val time =sdf.parse(line(0))
      val timestamp= new Timestamp(time.getTime)
      Row(timestamp,line(1),line(2).toDouble)
  }

  val field=Seq(
    StructField(hiveColumnName(0), TimestampType, true),
    StructField(hiveColumnName(1), StringType, true),
    StructField(hiveColumnName(2), DoubleType, true)
  )
  val schema=StructType(field)
  val zonedDateDataDf=sqlContext.createDataFrame(data,schema)
  zonedDateDataDf
}
}
```




```

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

    //预测后面N个值
    private var predictedN=1
    //存放的表名字
    private var outputTableName="timeseries_output"

    def this(predictedN:Int){
        this()
        this.predictedN=predictedN
    }

    case class Coefficient(coefficients: String,p: String,d: String,q:String,logLikelihoodCSS: Double,arc: Double)

    /**
     * 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,Vector)],RDD[(String,Coefficient)])={
        val predictedN=this.predictedN

        //train model
        val arimaAndVectorRDD=trainTsRDD.map{line=>
            line match {
                case (key,denseVector)=>
                    (key,ARIMA.autoFit(denseVector),denseVector)
            }
        }

        /**参数输出:p,d,q的实际值和其系数值、最大似然估计值、aic值**/
        val coefficients=arimaAndVectorRDD.map{line=>
            line match{
                case (key,arimaModel,denseVector)=>{
                    val coefficients=arimaModel.coefficients.mkString(",")
                    val p=arimaModel.p.toString
                    val d=arimaModel.d.toString
                    val q=arimaModel.q.toString
                    val logLikelihoodCSS=arimaModel.logLikelihoodCSS(denseVector).toString
                    val arc=arimaModel.approxAIC(denseVector).toString
                    (key,Coefficient(coefficients,p,d,q,logLikelihoodCSS,arc))
                }
            }
        }

        //print coefficients
        coefficients.collect().map{f=>
            val key=f._1
            val coefficients=f._2
            println(key+" coefficients:"+coefficients.coefficients+"=>"+(p="+coefficients.p"+d="+coefficients.d"+q="+coefficients.q"+logLikelihoodCSS="+coefficients.logLikelihoodCSS+" arc:"+coefficients.arc))
        }

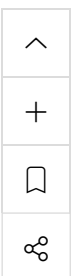
        //predict
        val forecast= arimaAndVectorRDD.map{
            row=>
                val key=row._1
                val model=row._2
                val denseVector=row._3
                (key,model.forecast(denseVector,predictedN))
        }

        //print predict
        val forecastValue=forecast.map{
            _ match{
                case (key,value)=>{
                    val partArray=value.toArray.mkString(",").split(",")
                    var forecastArrayBuffer=new ArrayBuffer[Double]()
                    var i=partArray.length-predictedN
                    while(i<partArray.length){
                        forecastArrayBuffer+=partArray(i).toDouble
                        i=i+1
                    }
                    (key,Vectors.dense(forecastArrayBuffer.toArray))
                }
            }
        }

        println("Arima forecast of next "+predictedN+" observations:")
        forecastValue.foreach(println)

        //return forecastValue & coefficients
        (forecastValue,coefficients)
    }
}

```



```

}

/**
 * 实现HoltWinters模型
 * @param trainTsRDD
 */
def holtWintersModelTrain(trainTsRDD:TimeSeriesRDD[String],period:Int,holtWintersModelType: String) : (ArrayBuffer[Forecast],ArrayBuffer[SSE]) = {

  //set parms
  val predictedN=this.predictedN

  //create model
  val holtWintersAndVectorRDD=trainTsRDD.map{
    row=>
      val key=row._1
      val denseVector=row._2
      //ts: Vector, period: Int, modelType: String = "additive", method: String = "BOBYQA"
      val model=HoltWinters.fitModel(denseVector,period,holtWintersModelType)
      (key,model,denseVector)
  }

  //create dist vector
  val predictedArrayBuffer=new ArrayBuffer[Double]()
  var i=0
  while(i<predictedN){
    predictedArrayBuffer+=i
    i=i+1
  }
  val predictedVectors=Vectors.dense(predictedArrayBuffer.toArray)

  //predict
  val forecast=holtWintersAndVectorRDD.map{
    row=>
      val key=row._1
      val model=row._2
      val denseVector=row._3
      val forecast=model.forecast(denseVector,predictedVectors)
      (key,forecast)
  }

  //print predict
  println("HoltWinters forecast of next "+predictedN+" observations:")
  forecast.foreach(println)

  //sse- to get sum of squared errors
  val sse=holtWintersAndVectorRDD.map{
    row=>
      val key=row._1
      val model=row._2
      val vector=row._3
      (key,model.sse(vector))
  }
  return (forecast,sse)
}

/**
 * 批量生成日期（具体到分钟秒），用来保存
 * 格式为: yyyy-MM-dd HH:mm:ss
 * @param predictedN
 * @param startTime
 * @param endTime
 */
def productStartDatePredictDate(predictedN:Int,startTime:String,endTime:String): ArrayBuffer[String] = {
  //形成开始start到预测predicted的日期
  var dateArrayBuffer=new ArrayBuffer[String]()
  val dateFormat= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  val cal1 = Calendar.getInstance()
  val cal2 = Calendar.getInstance()
  val st=dateFormat.parse(startTime)
  val et=dateFormat.parse(endTime)
  //设置训练数据中开始和结束日期
  cal1.set(st.getYear,st.getMonth,st.getDay,st.getHours,st.getMinutes,st.getSeconds)
  cal2.set(et.getYear,et.getMonth,et.getDay,et.getHours,et.getMinutes,et.getSeconds)
  //间隔差
  val minuteDiff=(cal2.getTimeInMillis-cal1.getTimeInMillis)/(1000 * 60 * 5)+predictedN

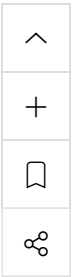
  var iMinuteDiff = 0;
  while(iMinuteDiff<=minuteDiff){
    cal1.add(Calendar.MINUTE,5)
    //保存日期
    dateArrayBuffer+=dateFormat.format(cal1.getTime)
    iMinuteDiff=iMinuteDiff+1;
  }
}

```



```
    }  
    dateArrayBuffer  
  }  
}
```

Holtwinters实现



```

/**
 * 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

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 {

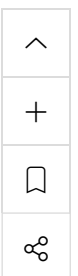
  /**
   * Fit HoltWinter model to a given time series. Holt Winter Model has three parameters
   * level, trend and season component of time series.
   * We use BOBYQA optimizer which is used to calculate minimum of a function with
   * bounded constraints and without using derivatives.
   * See http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf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

  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)
      }
    })
    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, maxEval)
    val params = optimal.getPoint
    new HoltWintersModel(modelType, period, params(0), params(1), params(2))
  }

  // 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

  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("multiplicative"))
        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 parameters
     * @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 sqErrors = 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)
            sqErrors += error * error
        }

        sqErrors
    }

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

    /**
     * {@inheritDoc}
     */
    override def addTimeDependentEffects(ts: Vector, dest: Vector): Vector = {
        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 and season
     * But in R's stats:HoltWinters additional weight is given for trend
     *
     * @param ts
     * @param dest
     */
    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 initial 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.html
* for more information on Holt Winter Method equations.
*
* @param ts A time series for which we want the HoltWinter parameters level,trend and season
* @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 - alpha) * (level(i) + trend(i))

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

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

  (Vectors.dense(dest), Vectors.dense(level), Vectors.dense(trend), Vectors.dense(season))
}

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)
  }
}

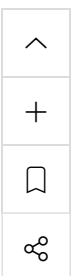
/**
 * Function to calculate the Weighted moving average/convolution using above kernel/weight
 * for input data.
 * See http://robjhyndman.com/papers/movingaverage.pdf for more information
 * @param inData Series on which you want to do moving average
 * @param kernel Weight vector for weighted moving 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 multiplicative
    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 values of that period
    val seasonalMean = removeTrend.splitAt(period).zipped.map { case (prevx, x) =>
      if (prevx == 0 || x == 0) (x + prevx) else (x + prevx) / 2
    }

    val meanOfFigures = seasonalMean.sum / period

    // The seasonal mean is then centered and removed to get season.
    // Subtract for additive and divide for multiplicative.
    val initSeason = if (additive) {
      seasonalMean.map(_ - meanOfFigures)
    } else {
      seasonalMean.map(_ / meanOfFigures)
    }

    // 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

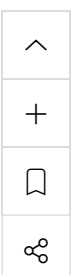
    val xxbar = indices.map( x => (x - xbar) * (x - xbar) ).sum
    val xybar = indices.zip(trend).map {
      case (x, y) => (x - xbar) * (y - ybar)
    }.sum

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

    (initLevel, initTrend, initSeason)
  }
}

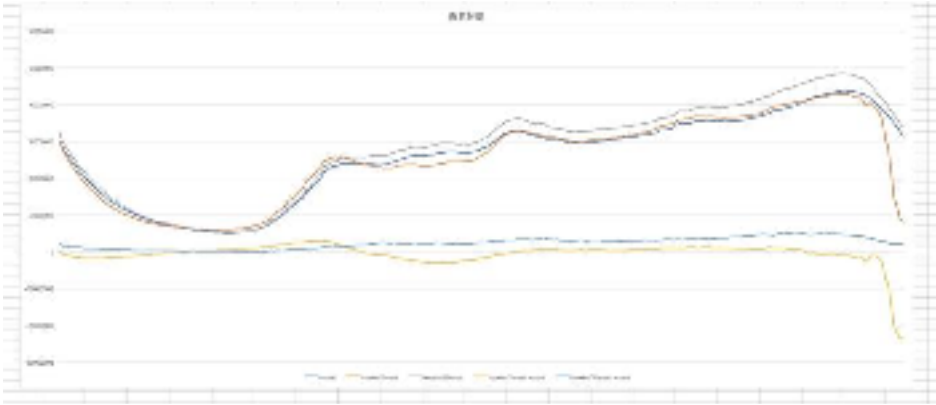
```

预测结果：



	A	B	C	D	E	F	G
1	2017/5/1 0:00 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
2	2017/5/1 0:05 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
3	2017/5/1 0:10 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
4	2017/5/1 0:15 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
5	2017/5/1 0:20 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
6	2017/5/1 0:25 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
7	2017/5/1 0:30 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
8	2017/5/1 0:35 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
9	2017/5/1 0:40 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04
10	2017/5/1 0:45 key	23331723	23331723.04	23331723.04	23331723.04	23331723.04	23331723.04

折线图：



actual：6/1当天的实际值

Hotwind7*MUTI:预测值（周期为一周，乘法）

Hotwind30*MUTI:预测值（周期为一个月，乘法）

其余：预测值与实际值的差值

结论：

按照5分钟为时间间隔。最小的周期性为天。即period=288。这个周期误差较其余两者稍微大一些。选择一周288 * 7 或者一个月288 * 30，效果如图。还可以。但其实这个值怎么选。有点存疑，即使根据sum of squared errors来看。

小礼物走一走，来简书关注我

赞赏支持

Spark (/nb/11733884)

举报文章 © 著作权归作者所有



raincoffee (/u/a62849f621bf) ♂

写了 197452 字，被 68 人关注，获得了 120 个喜欢

(/u/a62849f621bf)

一个人的咖啡


+ 关注


喜欢 | 4





更多分享

(http://cwb.assets.jianshu.io/notes/images/1376029)











写下你的评论...

9条评论

只看作者

按喜欢排序 按时间正序 按时间倒序



长空一雁 (/u/34cd311e461a)

2楼 · 2017.08.07 14:13

(/u/34cd311e461a)
学习了

赞 回复

raincoffee (/u/a62849f621bf) : @长空一雁 (/users/34cd311e461a) 😊

2017.08.07 22:13 回复

添加新评论



e*辉 (/u/6e76ce5d9ab3)

3楼 · 2017.10.26 09:39

(/u/6e76ce5d9ab3)

楼组你好，我现在下载了那个项目，但是存在一些缺包的问题，不懂你是否也遇到过这种情况呢？

赞 回复

raincoffee (/u/a62849f621bf) : @辉e (/users/6e76ce5d9ab3) 貌似没有出现啊

2017.10.26 12:04 回复

e*辉 (/u/6e76ce5d9ab3) : @raincoffee (/users/a62849f621bf) 能否加个qq ? 374488688

2017.10.26 12:18 回复

raincoffee (/u/a62849f621bf) : @辉e (/users/6e76ce5d9ab3) 晚上吧 在工作白天

2017.10.26 12:22 回复

添加新评论 | 还有1条评论，展开查看



e*辉 (/u/6e76ce5d9ab3)

4楼 · 2017.11.01 16:41

(/u/6e76ce5d9ab3)

请问一下，为什么预测的数据感觉严重不准确呢？是我预测的数据量太大的缘故？预测多少的数据量才合适呢？

赞 回复



aiq涛声xs依|日fw (/u/9398c13843b8)

5楼 · 2017.12.20 14:59

(/u/9398c13843b8)

能否加一下qq：1102390121，我的预测值为NaN，什么时候选择HoltWinters中的additive，对数据的输入有什么要求吗

赞 回复

被以下专题收入，发现更多相似内容

+ 收入我的专题



数据科学 (/c/102149797c26?utm_source=desktop&utm_medium=notes-included-collection)



推荐阅读

更多精彩内容 > (/)

8.Countdownlatch (/p/7e43fece0cfb?utm_campaign=maleskine&utm_...

Countdownlatch 此小节介绍几个与锁有关的有用工具。 闭锁 (Latch) 闭锁 (Latch)：一种同步方法，可以延迟线程的进度直到线程到达某个终点状态。通俗的讲就是，一个闭锁相当于一扇大门，在大门打开之...

raincoffee (/u/a62849f621bf?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

Hadoop4-MapReduce2.x-yarn框架 (/p/dc06871dbe4... (/p/dc06871dbe4e?

Hadoop-MapReduce2.x-yarn框架 1.mapreduce1.0的不足 JobTracker 是 Map-reduce 的集中处理点，存在单点故障。 JobTracker 完成了太多的任务，造成...

raincoffee (/u/a62849f621bf?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

一个人是否成熟，看他被欺负时的样子 (/p/fef1203e22... (/p/fef1203e2248?

文 | 剑圣喵大师 01 有一款女生热爱的手游叫《恋与制作人》，里面有一位26岁就博导加教授的许墨博士，让无数少女为之尖叫，疯狂氪金。 可我的好友石...

剑圣喵大师 (/u/77a2e0d51afc?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

渡人不如度己 (/p/22e456b5b7e7?utm_campaign=ma... (/p/22e456b5b7e7?

在没有写作之前，我一直在工作，生活很忙碌，几乎没有思考的时间。 那时候似乎也不知道要停下来去反思。 开始写作之后，我接触了一个新的词语叫终...

无戒 (/u/46abcf684093?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

《前任3》：再爱也不回头，只是因为“作”么？ (/p/78c... (/p/78cb574a2750?

文 | 焱公子 看完《前任3》，第一感觉是男主孟云和女主林佳实在太“作”。 明明心里有对方，偏偏始终绷着不说。一个电话就能解决的事，却宁可秀给全世界...

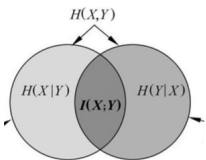
焱公子 (/u/eee6e7dea98c?
utm_campaign=maleskine&utm_content=user&utm_medium=pc_all_hots&utm_source=recommendation)

(PMBOK®指南) 第5版术语表 (/p/395efa04f2c4?utm_campaign=males...

术语表 (英文排序) 1 . 术语取舍## 本术语表包括以下术语：## 项目管理专用或几乎专用的术语 (如项目范围说明书、工作包、工作分解结构、关键 路径法) ； 虽非项目管理专用，但与一般日常用法相比，具有不...

飞行佳 (/u/f32e64038d29?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/c123c7534500?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

浅谈自然语言处理基础 (上) (/p/c123c7534500?utm_campaign=maleski...

本系列第三篇，承接前面的《浅谈机器学习基础》和《浅谈深度学习基础》。自然语言处理绪论 什么是自然语言处理？自然语言处理是研究人与交际中以及人与计算机交际中的语言问题的一门学科。自然语言处...

我偏笑_NSNirvana (/u/2293f85dc197?
utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/88cd5f3e0f75?

utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

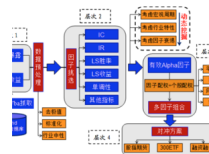
券商研报学习笔记 (/p/88cd5f3e0f75?utm_campaign=...

20101022-长江证券-多因子选股系列报告(二):多因子模型Beta估计方法 比较了窗口式的滚动最小二乘、全部历史数据滚动最小二乘、均值回复的状态空...



上官soyo (/u/7191311f1384?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)



灰度预测模型 (/p/b2b280883ed1?utm_campaign=maleskine&utm_cont...

常用的预测方法(如回归分析),需要较大的样本,如果样本较小,常造成较大的误差,使预测目标失效。灰度预测模型(Gray Forecast Model)是通过利用少量的、不完全的信息,建立数学模型并做出预测的一...



风逝流沙 (/u/173108d3e759?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

种地大学生的数据分析笔记之常见的时间序列模型 (/p/76eb6c7772a9?utm...

1. 移动平均法(MA) 1.1. 简单移动平均法 设有一时间序列 y_1, y_2, \dots , 则按数据点的顺序逐点推移求出N个数的平均数,即可得到一次移动平均数。 1.2. 加权移动平均 为比较临近的点的的数据赋予更高的权重。 1.3. 趋势移...



屈屈_ea7d (/u/359e8d79f0c6?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/89ef92680325?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

【筌论】浅谈一点点“著作权”的规定 (/p/89ef92680325?utm_campaign=...

今天想要来写一篇和平日里的文章风格差距较大的文章,可能会让看到这篇文章的老朋友感到有些不适应,只是,我认为还是因为在自己一段时间的学习后,及时的将自己了解到的一些东西做一个总结,而...



一方筌白 (/u/1eff5216dfed?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

android list file为null (/p/853f633eaaa9?utm_campaign=maleskine&ut...

问题: android 扫描SDcard目录, File.listFiles()总是返回null 解决方案: 修改 targetSdkVersion@build.gradle(Module:app)低于22 原文链接http://blog.csdn.net/xiaopang_lov...



树绿29 (/u/e6465b46dea9?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

群解散之倒计时 (/p/50b763090367?utm_campaign=maleskine&utm_co...

第五期好报30天写作群还有两天就要解散了,群里很多的朋友在聊自己通过每天500字写作有所改变,或是坚持写作得到的收获,包括关注度、上首页等。还有的朋友已经坚持了好几期,超过百天写作、码字好几...



薇薇薛 (/u/38db6c372ab1?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

你不知道那些爱情有多美 (/p/cbcba856aa70?utm_campaign=maleskine&...

忙碌的花店,策划情人节套餐的餐厅,各种app里关于购买礼物的推送,无处不在的信息都在提醒你情人节到了,不管你是高举火把一路高喊烧烧烧的单身汪,还是已经找到另一半正享受着恋爱甜蜜的情侣们,还是...



简书 (/u/yZq3ZV?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

