

# 对股票数据时间序列的异常点识别-最终报告

小组成员：叶超 3220180894 王宇 3220180871

岳鑫 3220180898 刘张敏 3220180832

## 1. 简介

股票市场作为风险与收益都较高投资市场，一直受到投资者的密切关注。各个国家的股票交易所每天都要产生巨量的交易数据，投资者与投资机构等在交易证券和投资股票时也越来越将数据作为首要参考依据，实时交易数据，比如 K线、分时线等等常常作为市场的客观反映。投资者和投资机构通常选取这些历史数据来进行分析和预测，以期获得更高利润。

## 2. 问题描述

异常点的来源包括航天设备故障信息、金融领域中金融诈骗、银行资产安全中欺诈信息... 在数据异常点识别算法中，对于时间序列的异常点检测更为常用和重要，该项目根据股票实时交易数据构建异常点识别模型，用于实时检测股票交易中出现的异常数据。本课题对于采集到的实时数据，利用 Sliding Window 对数据进行分割，采用 LOF 和 Isolation Forest 算法对于分割后的数据进行异常点检测，对于超过或者低于阈值的数据点进行标记。

## 3. 技术路线

### 3.1 软件平台

语言：Python3

深度学习框架：LOF 和 Isolation Forest 算法

本实验所需程序包：numpy, matplotlib, sklearn, scipy

## 3.2 数据处理

为了能实现股票的预测，最基本的也是不可或缺的条件是有每一支股票每日交易的日线 数据，即包括日期、开盘价、最高价、最低价、收盘价、成交量这六个基本变量。我们利用 choice 金融终端的 excel 插件将股票交易历史数据导入 excel 中，最终选取的是深市编号在 0 到 1000 的股票的交易数据，数据存放在 sz1000\_data 文件夹中，为随后的进一步挖掘提供了数据支持。

上证指数：

网易财经 [http://quotes.money.163.com/trade/lsjysj\\_zhishu\\_000001.html](http://quotes.money.163.com/trade/lsjysj_zhishu_000001.html)

抓取网址：[http://quotes.money.163.com/trade/lsjysj\\_股票代码.html](http://quotes.money.163.com/trade/lsjysj_股票代码.html)

经过对股票常用技术指标的筛选，最后筛选并确定的指标如下表所示

| 指标标记  | 指标名称                 | 计算方式  |
|-------|----------------------|---|
| s_x1  | 当日涨幅                 | (当日收盘价-前第 n 日收盘价)/前第 n 日收盘价 x 100%                |
| s_x2  | 2 日涨幅                |   |
| s_x3  | 5 日涨幅                |   |
| s_x4  | 10 日涨幅               |   |
| s_x5  | 30 日涨幅               |   |
| s_x6  | 10 日涨跌比率<br>ADR      | 10 日内股票上涨天数之和/N 日内股票下跌天数之和                        |
| s_x7  | 10 日相对强弱指标 RSI       | RSI= 100xRS/(l+RS).<br>RS = n 日的平均上涨点数/n 日的平均下跌点数 |
| s_x8  | 当日 K 线值              | (收盘价-开盘价) /(最高价-最低价)                              |
| s_x9  | 3 日 K 线值             | (收盘价-3 日前开盘价) / (3 日内最高价-3 日内最低价)                 |
| s_x10 | 6 日 K 线值             | (收盘价-6 日前开盘价) / (6 日内最高价-6 日内最低价)                 |
| s_x11 | 6 日乖离率(BIAS)         | 乖离率=[(当日收盘价-6 日平均价)/6 日平均价]x 100%                 |
| s_x12 | 10 日 乖 离 率<br>(BIAS) | 乖离率=[(当日收盘价-10 日平均价)/10 日平均价]x 100%               |
| s_x13 | 9 日 RSV              | (n 日收盘价-n 日最低价)/(n 日最高价-n 日最低价)x 100%             |
| s_x14 | 30 日 RSV             |   |
| s_x15 | 90 日 RSV             |   |
| s_x16 | 当日 OBV 量比            |   |

### 3.3 数据标准化

数据标准化的目的是消除变量间的量纲（单位）影响和变异大小因子的影响，使变量具有 可比性。这里将用均值方差归一化法来对数据进行标准化， 所得数据在[0, 1]之间，代码为 step3，在 matlab 执行此段代码后，训练样本 和 预测样本都被进行了标准化，且分别被保存在 train\_sample.xlsx 和 forecast\_sample.xlsx 两个文件中。

标准化公式：

$$n_i = \frac{p_i}{p_0} - 1$$

$n_i$  为处理后数据  $p_i$  为处理前数据  $p_0$  为处理窗口的第一个数据

### 3.4 异常点检测算法

常用的异常点检测算法可以分为三大类。第一类是基于统计学的方法来处理异常数据，这种方法一般会构建一个概率分布模型，并计算对象符合该模型的概率，把具有低概率的对象视为异常点。比如特征工程中的 RobustScaler 方法。第二类是基于聚类的方法来做异常点检测。由于大部分聚类算法是基于数据特征的分布来做的，通常如果我们聚类后发现某些聚类簇的数据样本量比其他簇少很多，而且这个簇里数据的特征均值分布之类的值和其他簇也差异很大，这些簇里的样本点大部分时候都是异常点。比如 BIRCH 聚类算法和 DBSCAN 密度聚类算法。第三类是基于专门的异常点检测算法来做。这些算法不像聚类算法，检测异常点只是一个赠品，它们的目的就是专门检测异常点的，这类算法的代表是 One Class SVM 和 Isolation Forest 等。我们在时间序列异常点检测过程中主要使用了 LOF 算法与 Isolation Forest 算法。

### 3.4.1 LOF 算法

LOF 算法 (Local Outlier Factor, 局部离群因子检测方法), 是一种基于密度的离群点检测方法。该算法会给数据集中的每个点计算一个离群因子 LOF, 通过判断 LOF 是否接近于 1 来判定是否是离群因子。若 LOF 远大于 1, 则认为是离群因子, 接近于 1, 则是正常点。

#### (1) 对象 p 的 k 距离

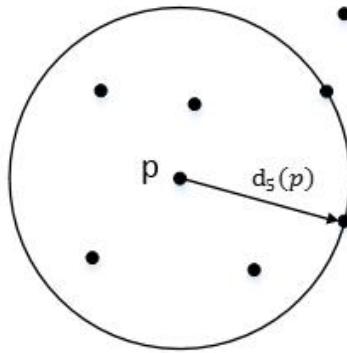
对于正整数  $k$ , 对象  $p$  的第  $k$  距离可记作  $k\text{-distance}(p)$ 。在样本空间中, 存在对象  $o$ , 它与对象  $p$  之间的距离基座  $d(p,o)$ 。如果满足以下两个条件, 我们则认为  $k\text{-distance}(p)=d(p,o)$ : 在样本空间中, 至少存在  $k$  个对象  $q$ , 使得  $d(p,q)\leq d(p,o)$ ; 在样本空间中, 至多存在  $k-1$  个对象  $q$ , 使得  $d(p,q)<d(p,o)$ 。

$$k\text{-distance}(p)=\max\{\|p-o\|\}$$

使用  $k\text{-distance}(p)$  来量化对象  $p$  的局部空间区域范围, 那么对于对象密度较大的区域,  $k\text{-distance}(p)$  值较小, 而对象密度较小的区域,  $k\text{-distance}(p)$  值较大。

#### (2) 对象 p 的第 k 距离领域

已经对象  $p$  的第  $k$  距离, 那么, 与对象  $p$  之间距离小于等于  $k\text{-distance}(p)$  的对象集合称为对象  $p$  的第  $k$  距离领域, 记作:  $N_k(p)$ 。该领域其实是以  $p$  为中心,  $k\text{-distance}(p)$  为半径的区域内所有对象的集合 (不包括  $p$  本身)。由于可能同时存在多个第  $k$  距离的数据, 因此该集合至少包括  $k$  个对象。离群度越大的对象的范围往往比较大, 而离群度比较小的对象范围小。



#### (3) 对象 p 相对于对象 o 的可达距离

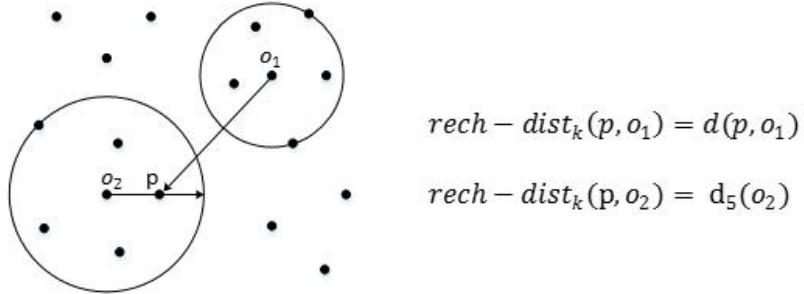
$$\text{reachdist}(p,o)=\max\{k\text{-distance}(o),\|p-o\|\}$$

也就是说, 如果对象  $p$  远离对象  $o$ , 则两者之间的可达距离就是它们之间的实际距离, 但是如果它们足够近, 则实际距离用  $o$  的  $k$  距离代替。

#### (4) 局部可达密度

对象 p 的局部可达密度定义为 p 的 k 最近邻点的平均可达密度的倒数

$$lrdk(p) = |N_k(p)| \sum_{o \in N_k(p)} \text{reachdist}_k(p, o)$$



## (5) 局部离群点因子

表征了 p 是离群点的程度，定义如下：

$$LOF_k(p) = \sum_{o \in N_k(p)} lrdk(o) / lrdk(p) |N_k(p)|$$

如果对象 p 不是局部离群点，则 LOF(p) 接近于 1。即 p 是局部离群点的程度较小，对象 o 的局部可达密度和对象 p 的局部可达密度相似，最后所得的 LOF(p) 值应该接近 1。相反，p 是局部离群点的程度越大，最后所得的 LOF(p) 值越高。通过这种方式就能在样本空间数据分布不均匀的情况下也可以准确发现离群点。

### 3.4.2 Isolation Forest 算法

南大周志华老师的团队在 2010 年提出一个异常检测算法 Isolation Forest，在工业界很实用，算法效果好，时间效率高，能有效处理高维数据和海量数据。

#### iTree 的构造

森林都是由树构成的，在看 Isolation Forest（简称 iForest）前，先看 Isolation-Tree（简称 iTree）是怎么构成的，iTree 是一种随机二叉树，每个节点要么有两个女儿，要么就是叶子节点，一个孩子都没有。给定一堆数据集 D，这里 D 的所有属性都是连续型的变量，iTree 的构成过程如下：

- a. 随机选择一个属性 Attr；
- b. 随机选择该属性的一个值 Value；
- c. 根据 Attr 对每条记录进行分类，把 Attr 小于 Value 的记录放在左女儿，把大于等于 Value 的记录放在右孩子；
- d. 然后递归的构造左女儿和右女儿，直到满足以下条件：  
传入的数据集只有一条记录或者多条一样的记录；

树的高度达到了限定高度：

---

**Algorithm 2 :  $iTree(X, e, l)$** 

---

**Inputs:**  $X$  - input data,  $e$  - current tree height,  $l$  - height limit

**Output:** an iTree

```
1: if  $e \geq l$  or  $|X| \leq 1$  then
2:   return  $exNode\{Size \leftarrow |X|\}$ 
3: else
4:   let  $Q$  be a list of attributes in  $X$ 
5:   randomly select an attribute  $q \in Q$ 
6:   randomly select a split point  $p$  from  $\max$  and  $\min$  values of attribute  $q$  in  $X$ 
7:    $X_l \leftarrow filter(X, q < p)$ 
8:    $X_r \leftarrow filter(X, q \geq p)$ 
9:   return  $inNode\{Left \leftarrow iTree(X_l, e + 1, l),$ 
10:    Right \leftarrow iTree(X_r, e + 1, l),
11:    SplitAtt \leftarrow q,
12:    SplitValue \leftarrow p\}
```

---

13: end if

## iForest 的构造

iTree 搞明白了，我们现在来看看 iForest 是怎么构造的，给定一个包含  $n$  条记录的数据集  $D$ ，如何构造一个 iForest。iForest 和 Random Forest 的方法有些类似，都是随机采样一部分数据集去构造每一棵树，保证不同树之间的差异性，不过 iForest 与 RF 不同，采样的数据量不需要等于  $n$ ，可以远远小于  $n$ 。除了限制采样大小以外，还要给每棵 iTree 设置最大高度，这是因为异常数据记录都比较少，其路径长度也比较低，而我们也只需要把正常记录和异常记录区分开来，因此只需要关心低于平均高度的部分就好，这样算法效率更高。

---

**Algorithm 1 :  $iForest(X, t, \psi)$** 

---

**Inputs:**  $X$  - input data,  $t$  - number of trees,  $\psi$  - sub-sampling size

**Output:** a set of  $t$  iTrees

```
1: Initialize Forest
2: set height limit  $l = ceiling(\log_2 \psi)$ 
3: for  $i = 1$  to  $t$  do
4:    $X' \leftarrow sample(X, \psi)$ 
5:   Forest  $\leftarrow Forest \cup iTree(X', 0, l)$ 
6: end for
7: return Forest
```

---

IForest 构造好后，对测试进行预测时，需要进行综合每棵树的结果，于是表示记录  $x$  在每棵树的高度均值，另外计算需要改进，在生成叶节点时，算法记录了叶节点包含的记录数量，这时候要用这个数量估计一下平均高度，的计算方法如下：

---

**Algorithm 3 : PathLength( $x, T, e$ )**

---

**Inputs :**  $x$  - an instance,  $T$  - an iTree,  $e$  - current path length;  
to be initialized to zero when first called

**Output:** path length of  $x$

```
1: if  $T$  is an external node then
2:   return  $e + c(T.size)$  { $c(\cdot)$  is defined in Equation 1}
3: end if
4:  $a \leftarrow T.splitAtt$ 
5: if  $x_a < T.splitValue$  then
6:   return  $PathLength(x, T.left, e + 1)$ 
7: else { $x_a \geq T.splitValue$ }
8:   return  $PathLength(x, T.right, e + 1)$ 
9: end if
```

---

## 3.5 算法运行和模型建立

### 3.5.1 时间序列构造模块算法:

- 1) Olympic Model (Seasonal Naive) : 简单的窗口模型, 对点  $P_x$  的预测为点  $P_x$  前  $n$  个值的平均值.

代码如下:

```
import java.util.Properties;
import java.util.ArrayList;
import java.util.Collections;
public class OlympicModel extends TimeSeriesAbstractModel {
    // methods /////////////////////////////////
    private static final long serialVersionUID = 1L;

    public int getNumWeeks() {
        return numWeeks;
    }

    public int getNumToDrop() {
        return numToDrop;
    }

    public int[] getTimeShifts() {
        return timeShifts;
    }
```

```

public int[] getBaseWindows() {
    return baseWindows;
}

public ArrayList<Float> getModel() {
    return model;
}

// Number of weeks to look back when computing the
// estimate.
protected int numWeeks;
// Number of lowest and highest points to drop.
protected int numToDrop;
// Stores the historical values.
protected TimeSeries.DataSequence data;
// Stores the possible time-shifts.
// time-shifts are used to fix the time-series
// that has been shifted due to day-light savings.
protected int[] timeShifts;
// Stores the possible base windows.
// The default base window is 1 week, however
// trying multiple possible windows seems to improve
// performance.
protected int[] baseWindows;

// The actual model that stores the expectations.
protected ArrayList<Float> model;

public OlympicModel(Properties config) {
    super(config);

    if (config.getProperty("NUM_WEEKS") == null) {
        throw new IllegalArgumentException("NUM_WEEKS is NULL");
    }
    if (config.getProperty("NUM_TO_DROP") == null) {
        throw new IllegalArgumentException("NUM_TO_DROP is
NULL");
    }
    if (config.getProperty("TIME_SHIFTS") == null) {
        throw new IllegalArgumentException("TIME_SHIFTS is NULL");
    }
    if (config.getProperty("BASE_WINDOWS") == null) {

```

```

        throw new IllegalArgumentException("BASE_WINDOWS is
NULL");
    }

    this.numWeeks = new Integer(config.getProperty("NUM_WEEKS"));
    this.numToDrop = new
Integer(config.getProperty("NUM_TO_DROP"));
    this.timeShifts =
FileUtils.splitInts(config.getProperty("TIME_SHIFTS"));
    this.baseWindows =
FileUtils.splitInts(config.getProperty("BASE_WINDOWS"));
    model = new ArrayList<Float>();
}

public void reset() {
    // At this point, reset does nothing.
}

public void train(TimeSeries.DataSequence data) {
    this.data = data;

    int n = data.size();

    java.util.Arrays.sort(baseWindows);
    java.util.Arrays.sort(timeShifts);
    float precision = (float) 0.000001;

    for (int i = 0; i < n; i++) {
        float baseVal = Float.POSITIVE_INFINITY;
        float tmpbase = (float) 0.0;

        // Cannot compute the expected value if the time-series
        // is too short preventing us from getting the reference
        // window.
        if ((i - baseWindows[0]) < 0) {
            model.add(data.get(i).value);
            continue;
        }

        // Attempt to shift the time-series.
        for (int w = 0; w < baseWindows.length; w++) {
            for (int j = 0; j < timeShifts.length; j++) {
                if (timeShifts[j] == 0) {
                    tmpbase = computeExpected(i, baseWindows[w]);
                }
            }
        }
    }
}

```

```

        if ((Math.abs(tmpbase - data.get(i).value) -
Math.abs(baseVal - data.get(i).value)) < precision) {
            baseVal = tmpbase;
        }
    } else {
        if (i + timeShifts[j] < n) {
            tmpbase = computeExpected(i + timeShifts[j],
baseWindows[w]);
            if ((Math.abs(tmpbase - data.get(i).value) -
Math.abs(baseVal - data.get(i).value)) < precision) {
                baseVal = tmpbase;
            }
        }
        if (i - timeShifts[j] >= 0) {
            tmpbase = computeExpected(i - timeShifts[j],
baseWindows[w]);
            if ((Math.abs(tmpbase - data.get(i).value) -
Math.abs(baseVal - data.get(i).value)) < precision) {
                baseVal = tmpbase;
            }
        }
    }
    model.add(baseVal);
}

initForecastErrors(model, data);

logger.debug(getBias() + "\t" + getMAD() + "\t" + getMAPE() + "\t" +
getMSE() + "\t" + getSAE() + "\t" + 0 + "\t" + 0);
}

public void update(TimeSeries.DataSequence data) {

}

public String getModelName() {
    return "OlympicModel";
}

private Float sum(ArrayList<Float> list) {
    float sum = 0;
    for (float i : list) {

```

```

        sum = sum + i;
    }
    return sum;
}

private float computeExpected(int i, int pl) {
    ArrayList<Float> vals = new ArrayList<Float>();
    float precision = (float) 0.000001;

    int j = 1;

    if ((i - pl * j) < 0) {
        return Float.POSITIVE_INFINITY;
    }
    while (j <= this.numWeeks && (i - pl * j) >= 0) {
        float lastWeeksVal = data.get(i - pl * j).value;
        // If dynamic parameters are turned on,
        // then we check if our error improved from last time,
        // if not, then we stop and use the old result.
        if (dynamicParameters == 1 && vals.size() > 0) {
            float withNewVal = (sum(vals) + lastWeeksVal) / (vals.size()
+ 1);
            float withoutNewVal = (sum(vals)) / (vals.size());
            if ((Math.abs(withNewVal - data.get(i).value) -
Math.abs(withoutNewVal - data.get(i).value)) > precision) {
                break;
            }
        }
        vals.add(lastWeeksVal);
        j++;
    }

    Collections.sort(vals);
    j = 0;

    if (vals.size() > (2 * this.numToDrop)) {
        while (j < this.numToDrop) {
            vals.remove(vals.size() - 1);
            vals.remove(0);
            j++;
        }
    }

    float baseVal = sum(vals) / vals.size();
}

```

```

        return baseVal;
    }

    public void predict(TimeSeries.DataSequence sequence) throws
Exception {
    int n = data.size();
    for (int i = 0; i < n; i++) {
        sequence.set(i, (new Entry(data.get(i).time, model.get(i))));
        logger.info(data.get(i).time + "," + data.get(i).value + "," +
model.get(i));
    }
}
}

```

### 3.5.2 异常点检测模块:

代码如下:

```

import java.util.ArrayList;
public class AnomalyDetector {

    protected TimeSeries metric = null;
    protected ArrayList<AnomalyDetectionModel> models = new
ArrayList<AnomalyDetectionModel>();
    protected ArrayList<Boolean> isTuned = new ArrayList<Boolean>();
    protected long firstTimeStamp = 0;
    protected long period;

    // Construction /////////////////////////////////
    public AnomalyDetector(TimeSeries theMetric, long period,
                           long firstTimeStamp) throws Exception {
        if (theMetric == null) {
            throw new Exception("The input metric is null.");
        }

        metric = theMetric;
        this.period = period;
        this.firstTimeStamp = firstTimeStamp;
    }

    public AnomalyDetector(TimeSeries theMetric, long period) throws
Exception {

```

```

        if (theMetric == null) {
            throw new Exception("The input metric is null.");
        }

        metric = theMetric;
        this.period = period;

        if (metric.data.size() > 0) {
            this.firstTimeStamp = metric.time(0);
        }
    }

public AnomalyDetector(String theMetric, long period) throws Exception {
    this.period = period;
    // TODO:
    // 1 - load the models related to theMetric from ModelDB
    // 2 - push the loaded models into 'models'
    // 3 - create a new TimeSeries for theMetric and set 'metric'
    // 4 - set 'firstTimeStamp'

    int modelNum = models.size();
    for (int i = 0; i < modelNum; ++i) {
        isTuned.set(i, true);
    }
}

// Configuration Methods /////////////////////////////////
public void setMetric(TimeSeries theMetric, long period) {
    metric = theMetric;
    this.period = period;

    if (metric.data.size() > 0) {
        this.firstTimeStamp = metric.time(0);
    }

    reset();
}

public void setMetric(TimeSeries theMetric, long period, long
firstTimeStamp) {
    metric = theMetric;
    this.period = period;
    this.firstTimeStamp = firstTimeStamp;
}

```

```

        reset();
    }

public void setMetric(String theMetric, long period) {
    this.period = period;
    firstTimeStamp = 0;
    models.clear();
    isTuned.clear();

    // TODO:
    // 1 - load the models related to theMetric from ModelDB
    // 2 - push the loaded models into 'models'
    // 3 - create a new TimeSeries for theMetric and set 'metric'
    // 4 - set 'firstTimeStamp'

    int modelNum = models.size();
    for (int i = 0; i < modelNum; ++i) {
        isTuned.set(i, true);
    }
}

public void addModel(AnomalyDetectionModel model) {
    model.reset();
    models.add(model);
    isTuned.add(false);
}

// Algorithmic Methods /////////////////////////////////
public void reset() {
    int i = 0;
    for (AnomalyDetectionModel model : models) {
        model.reset();
        isTuned.set(i, false);
        i++;
    }
}

public void tune(TimeSeries.DataSequence expectedValues,
                IntervalSequence anomalySequence) throws Exception {
    int i = 0;

    metric.data.setLogicalIndices(firstTimeStamp, period);
}

```

```

        for (AnomalyDetectionModel model : models) {
            if (!isTuned.get(i)) {
                model.tune(metric.data, expectedValues,
anomalySequence);
                isTuned.set(i, true);
            }
            i++;
        }
    }

    public ArrayList<Anomaly> detect(TimeSeries observedSeries,
                                         TimeSeries.DataSequence
expectedSeries) throws Exception {
        for (Boolean b : isTuned) {
            if (!b) {
                throw new Exception(
                    "All the models need to be tuned before
detection.");
            }
        }
}

ArrayList<Anomaly> result = new ArrayList<Anomaly>();
observedSeries.data.setLogicalIndices(firstTimeStamp, period);
expectedSeries.setLogicalIndices(firstTimeStamp, period);

for (AnomalyDetectionModel model : models) {
    Anomaly anomaly = new Anomaly(observedSeries.meta.name,
                                    observedSeries.meta);
    anomaly.modelName = model.getModelName();
    anomaly.type = model.getType();
    anomaly.intervals = model.detect(observedSeries.data,
                                      expectedSeries);
    anomaly.intervals.setLogicalIndices(firstTimeStamp, period);
    anomaly.intervals.setTimeStamps(firstTimeStamp, period);
    result.add(anomaly);
}

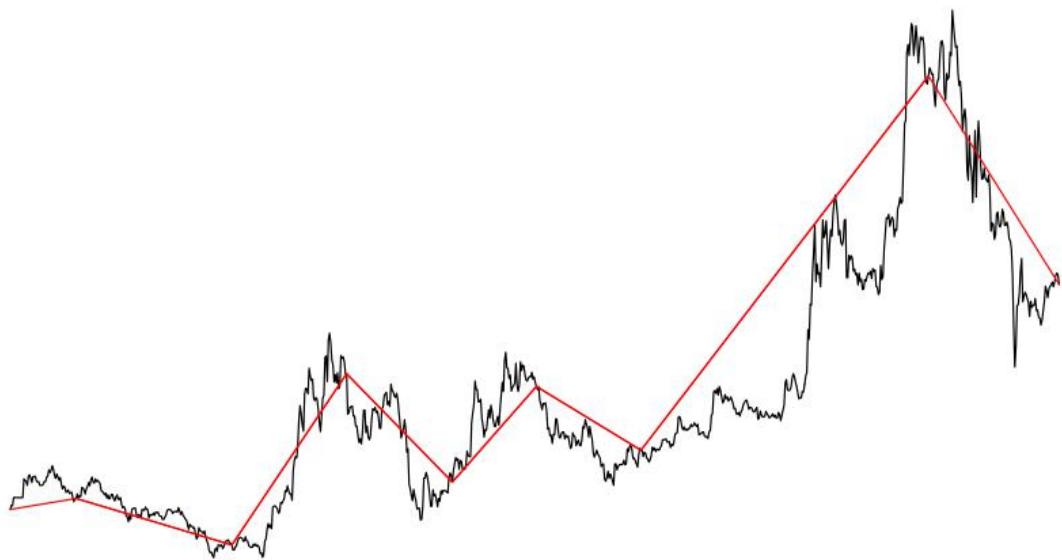
return result;
}
}

```

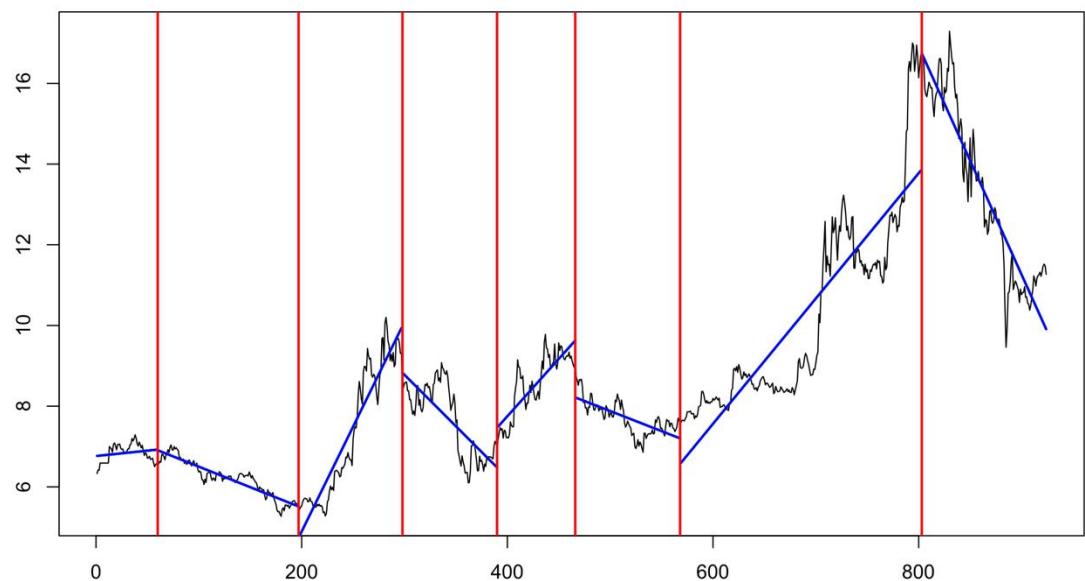
## 3.6 算法结果和模型评估

### 3.6.1 算法结果：

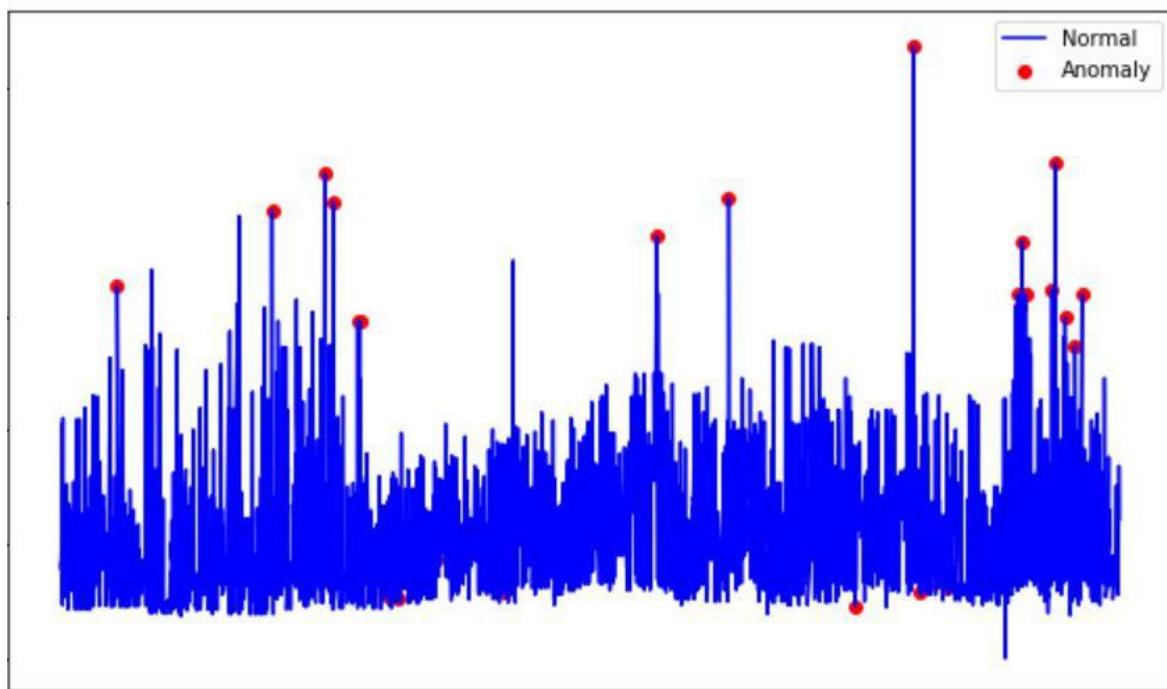
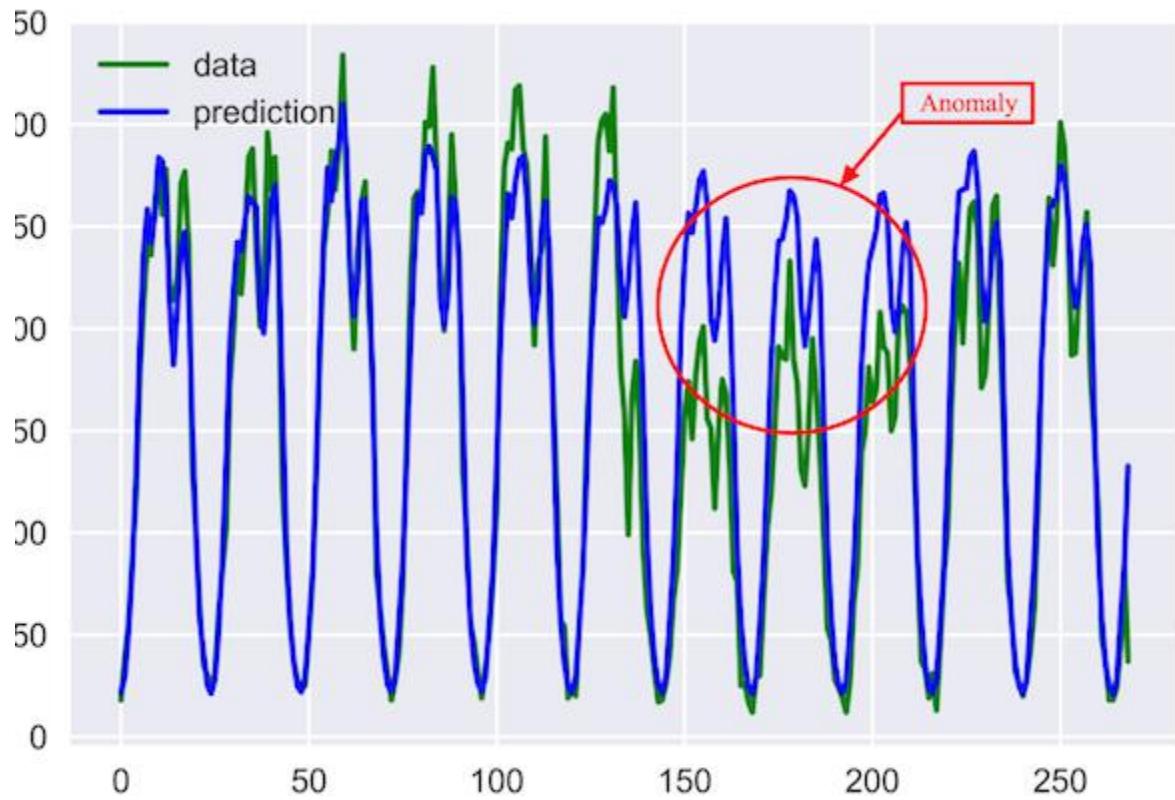
时间序列分段如图所示：



图中每个红色线条的转折点，就是找到的断点，将两个断点之间的数据分割出来进行拟合，得到如图所示：



针对每段数据采用异常点检测算法，将发现的异常点标记出来。



3.6.2 模型评估

模型的效果可以通过准确率来进行衡量，在时间序列异常点检测场景下，某个模型的准确率可以定义为：

$$P = \frac{\text{模型检测出来的异常量}}{\text{数据真实存在的异常量}}$$

可以得到模型效果如图所示：

|        | <b>IF</b> | <b>LOF</b> |
|--------|-----------|------------|
| 正确率    | 0.891     | 0.91       |
| 召回率    | 0.9       | 0.93       |
| 特效度    | 0.875     | 0.875      |
| 消耗时间/s | 1         | 108        |

## 4. 结论

由于个股股价受多方因素影响较大，单纯利用历史交易记录来识别其中异常点是极其困难的，而且需要一定的专家知识才能真正确认，但是本模型任有一定意义。它可以通过大量股票数据来筛选稳定的个股（这些个股的预测误差往往比较低），同时还可以利用模型对这些稳定股票中的异常数据进行判断来制定初步的交易策略。

源码和数据 github 地址：

[https://github.com/Lzm1996/datamining\\_groupwork](https://github.com/Lzm1996/datamining_groupwork)