检测模型算法

方法类别	方法名称	异常标准简述	备注
基于分布	3σ	值> $\mu+3\sigma$ 或 值< $\mu-3\sigma$	
	Boxplot	值>q3+1.5IQR 或 值<q1-1.5IQR	
	Grubbs	z_score > Grubbs临界值	
	weighted_avg	值> $\mu+3\sigma$ 或 值< $\mu-3\sigma$	计算残差的均值和标准差
基于时间序列	ewma	预测值在区间范围之内	
	arima	预测值在区间范围之内	

代码块

```python
# ThreeSigma
# 传入的数据不包括需要检测的数据
class OutlierDetectorThreeSigma:
    def __init__(self, data, threshold=3.0):
        self.data = np.array(data)
        self.mean = np.mean(data)
        self.std_dev = np.std(data)
        self.threshold = threshold

    def is_outlier(self, value):
        """
        判断一个值是否为异常值。
        如果值落在均值±3*标准差之外，则认为是异常值。
        """
        lower_bound = self.mean - self.threshold * self.std_dev
        upper_bound = self.mean + self.threshold * self.std_dev
        return value < lower_bound or value > upper_bound

    def get_up_down_level(self):
        lower_bound = self.mean - self.threshold * self.std_dev
```

```
        upper_bound = self.mean + self.threshold * self.std_dev
        return round(lower_bound, 2), round(upper_bound, 2)

# BoxPlot (箱线图)
# 传入的数据不包括需要检测的数据
class OutlierDetectorBoxPlot:
    def __init__(self, data, threshold=1.5):
        self.data = np.array(data)
        self.threshold = threshold

    def detect_anomaly_data(self, new_data):
        """
        判断一个值是否为异常值。
        如果值落在下四分位数和上四分位数之外，则认为是异常值。
        """
        lower_bound, upper_bound = self.get_up_down_level()
        if new_data < lower_bound or new_data > upper_bound:
            return True
        return False

    def get_up_down_level(self):
        # 获取箱线图的特征
        '''
        使用 numpy.percentile 函数计算数据的第25百分位数（下四分位数）、第50百分位数（中位数）和第75百分位数（上四分位数）。
        结果存储在 quartiles 数组中
        '''
        quartiles = np.percentile(self.data, [25, 50, 75])
        '''
        计算四分位距，四分位距（Interquartile Range, IQR）是上四分位数和下四分位数之间的差值。
        它是一个衡量数据分散程度的统计量。
        '''
        iqr = quartiles[2] - quartiles[0]
        # 计算上下界
        lower_bound = quartiles[0] - self.threshold * iqr
        upper_bound = quartiles[2] + self.threshold * iqr
        return round(lower_bound, 2), round(upper_bound, 2)

# Grubbs (格鲁布斯检验)
# 基于假设检验的思想，通过计算一个统计量来判断数据集中是否存在显著偏离其他数据点的异常值
# 传入的数据包括需要检测的数据
class OutlierDetectorGrubbs:
    def __init__(self, data, alpha=0.05):
        # 设置显著性水平
        self.alpha = alpha
        self.data = data
```

```
        # 计算均值
        self.mean = np.mean(self.data)
        # 计算标准差
        self.std_dev = np.std(self.data, ddof=1)
        # 计算最大偏差
        self.max_deviation = np.max(np.abs(self.data - self.mean))
        # 计算 Grubbs 统计量
        self.grubbs_statistic = self.max_deviation / self.std_dev
        # 计算Grubbs的临界值
        self.critical_value = self.calculate_critical_value(len(self.data))

    def calculate_critical_value(self, n):
        # 计算Grubbs' Test的临界值，基于样本大小n和显著性水平alpha
        """
        计算 t 值
        t.ppf 是 SciPy 库中的累积分布函数（CDF）的逆函数，用于计算 t 分布的临界值。
        1 - self.alpha / (2 * n) 是用于计算双尾检验的显著性水平。
        n - 2 是自由度，因为 Grubbs' Test 使用的是 t 分布，自由度为 n - 2
        """
        t_value = t.ppf(1 - self.alpha / (2 * n), n - 2)
        # 计算 Grubbs 临界值
        critical_value = ((n - 1) * np.sqrt(t_value**2)) / (np.sqrt(n) *
np.sqrt(n - 2 + t_value**2))
        return critical_value

    def test(self):
        """
        如果 Grubbs 统计量大于临界值，则认为数据集中存在异常值。
        使用 np.argmax(np.abs(self.data - mean)) 找到最大偏差对应的数据点的索引
        根据索引返回异常值
        """
        if self.grubbs_statistic > self.critical_value:
            outlier_index = np.argmax(np.abs(self.data - self.mean))
            return True, data[outlier_index]
        else:
            return False, None

    def get_up_down_level(self, data):
        """
        找出数据中的所有上下区间。
        """
        lower_bound = self.mean - self.critical_value * self.std_dev
        if lower_bound < 0.0:
            lower_bound = 0.0
        upper_bound = self.mean + self.critical_value * self.std_dev
        return round(lower_bound, 2), round(upper_bound, 2)

```

```python
    # WeightedAverage (加权平均)
    # 传入的数据包括需要检测的数据
    class OutlierDetectorWeightedAverage:
      def __init__(self, weights=None):
          self.weights = weights if weights is not None else None
          self.data = None
          self.weighted_averages = None

      def fit(self, data, weights=None):
          self.data = np.array(data)
          # 设置每个输入数据的权重，默认都为1
          self.weights = np.array(weights) if weights is not None else None
          if self.weights is None:
              self.weights = np.ones_like(self.data)
          # 使用选定的权重计算每个输入数据的加权平均值
          self.weighted_averages = np.average(self.data, weights=self.weights)

      def detect_anomalies(self, threshold=3):
          # 计算每个数据的实际值与加权平均值之间的残差
          deviations = np.abs(self.data - self.weighted_averages)
          # 计算所有残差的标准差
          standard_deviation = np.sqrt(np.average((deviations -
np.average(deviations))**2, weights=self.weights))
          # 如果某个数据的残差超出定义的阈值，则认为该数据是异常的
          anomalies = np.where(deviations > threshold * standard_deviation)[0]
          return anomalies.tolist()

       def get_up_down_level(self, threshold):
          """
          找出数据中的所有上下区间。
          """
          lower_bound = self.weighted_averages - threshold *
self.standard_deviation
          if lower_bound < 0.0:
              lower_bound = 0.0
          upper_bound = self.weighted_averages + threshold *
self.standard_deviation
          return round(lower_bound, 2), round(upper_bound, 2)

    # EWMA (Exponentially Weighted Moving Average，指数加权移动平均)
    # 传入的数据不包括需要检测的数据
    class OutlierDetectorEWMA:
      def __init__(self, alpha=0.2, threshold=3):
          self.alpha = alpha
          self.threshold = threshold
          self.ewma = None
          self.anomalies = None
```

```
156             self.deviation_var = 0.0
157
158         def fit(self, data):
159             self.data = np.array(data)
160             self.ewma = np.zeros_like(self.data)
161             # 每个时间点的EWMA值
162             self.ewma[0] = self.data[0]
163             for i in range(1, len(self.data)):
164                 # EWMA公式 St=αxt+(1-α)St-1
165                 self.ewma[i] = self.alpha * self.data[i] + (1 - self.alpha) *
    self.ewma[i-1]
166             deviations = np.abs(self.data - self.ewma)
167             self.deviation_var = deviations[1:].std()
168
169         def is_outlier(self, new_data):
170             """
171             判断一个值是否为异常值。
172             """
173             # 计算新传入的时间点的EWMA值
174             new_data_alpha = self.alpha * new_data + (1 - self.alpha) *
    self.ewma[-1]
175             # 计算新传入的时间点的实际值与计算得出的EWMA值之间的残差
176             deviation = np.abs(data - data_new)
177             # 如果新传入的时间点的残差超出定义的阈值，则认为新传入的时间点的数据是异常的
178             if deviation <= self.deviation_var * self.threshold:
179                 return False
180             return True
181
182         def get_up_down_level(self, new_data):
183             """
184             找出数据中的所有上下区间。
185             """
186             # 计算新传入的时间点的EWMA值
187             new_data_alpha = self.alpha * new_data + (1 - self.alpha) *
    self.ewma[-1]
188             lower_bound = data_new - self.threshold * self.deviation_var
189             if lower_bound < 0.0:
190                 lower_bound = 0.0
191             upper_bound = self.ewma[-1] + self.threshold * self.deviation_var
192             return round(lower_bound, 2), round(upper_bound, 2)
193
194     # ARIMA （AutoRegressive Integrated Moving Average，自回归积分滑动平均模型）
195     # 传入的数据不包括需要检测的数据
196     class OutlierDetectorARIMA:
197         def __init__(self, data, order=(1, 1, 1), threshold=3.0):
198             """
199             初始化ARIMA异常检测器
```

```
        使用历史数据拟合ARIMA模型。
        Parameters:
        order (tuple): ARIMA模型的(p, d, q)参数
        确定模型的参数（p，d，q），其中：
        p 是自回归部分的阶数。
        d 是差分的次数。
        q 是移动平均部分的阶数。
        threshold (float): 用于判断异常的阈值
        """
        self.order = order
        self.threshold = threshold
        self.model = None
        self.fitted_values = None
        self.data = data
        self.std_residuals = 0.0

    # 根据AIC和BIC可以用于比较不同 ARIMA 模型（即不同的 (p, d, q) 参数组合）
    def get_best_pdq(self):
        # 定义 p, d, q 的范围
        p_values = range(0, 3)
        d_values = range(0, 2)
        q_values = range(0, 3)

        best_aic = float("inf")
        best_bic = float("inf")
        best_total_aic_bic = float("inf")
        best_order = None
        best_order_bic = None
        best_order_aic_bic = None

        for p in p_values:
            for d in d_values:
                for q in q_values:
                    try:
                        # 拟合 ARIMA 模型
                        model = ARIMA(self.data, order=(p, d, q))
                        model_fit = model.fit()

                        # 记录 AIC 和 BIC 值
                        aic = model_fit.aic
                        bic = model_fit.bic

                        total_aic_bic = model_fit.aic + model_fit.bic

                        if aic < best_aic:
                            best_aic = aic
                            best_order = (p, d, q)
```

```
247
248                         if bic < best_bic:
249                             best_bic = bic
250                             best_order_bic = (p, d, q)
251
252                         if total_aic_bic < best_total_aic_bic:
253                             best_total_aic_bic = total_aic_bic
254                             best_order_aic_bic = (p, d, q)
255
256                         print(f"Order: ({p},{d},{q}) - AIC: {aic}, BIC: {bic},
    AIC+BIC: {total_aic_bic}")
257                     except:
258                         continue
259
260         print(f"Best AIC Order: {best_order}, Best AIC: {best_aic}")
261         print(f"Best BIC Order: {best_order_bic}, Best BIC: {best_bic}")
262         print(f"Best AIC+BIC Order: {best_order_aic_bic}, Best AIC+BIC:
    {best_order_aic_bic}")
263         # 我们默选Best AIC+BIC
264         return best_order_aic_bic
265
266     def fit(self):
267         """
268         拟合ARIMA模型
269
270         Parameters:
271         data (pd.Series): 输入的时间序列数据
272         """
273         self.model = ARIMA(self.data, order=self.order)
274         self.model = self.model.fit()
275         # 使用ARIMA模型预测时间序列的未来值
276         self.fitted_values = self.model.fittedvalues
277         residuals = self.data - self.fitted_values
278         # 因为对于处理GMV等数据较大的值,所以需要去掉residuals中的第一个值
279         self.std_residuals = np.std(residuals[1:])
280
281     # 判断一个值是否是异常值
282     def is_outlier(self, test_data):
283         # 使用训练好的ARIMA模型来预测后下一个值
284         new_data = self.model.forecast(steps=1)
285         if abs(test_data - new_data[0]) <= self.threshold * self.std_residuals:
286             return False
287         return True
288
289     # 计算上下限
290     def get_up_down_level(self):
291         """
```

```
292             找出数据中的所有上下区间。
293             """
294             # 使用训练好的ARIMA模型来预测后下一个值
295             new_data = self.model.forecast(steps=1)
296             lower_bound = new_data[0] - self.threshold * self.std_residuals
297             if lower_bound < 0.0:
298                 lower_bound = 0.0
299             upper_bound = new_data[0] + self.threshold * self.std_residuals
300             return round(lower_bound, 2), round(upper_bound, 2)
301
302
```

ARIMA异常检测介绍

ARIMA（AutoRegressive Integrated Moving Average，自回归积分滑动平均模型）是一种广泛用于时间序列分析和预测的方法。虽然ARIMA主要用于预测未来值，但它也可以用于检测时间序列中的异常值。通过构建ARIMA模型，我们可以预测时间序列的未来值，并将实际值与预测值进行比较，从而识别出异常值。

基本概念

ARIMA模型的基本概念

1. **自回归（AR）部分**：
   - 使用过去的值来预测未来的值。
   - 模型形式：$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t$
2. **差分（I）部分**：
   - 对时间序列进行差分操作，使其变得平稳。
   - 模型形式：$y'_t = y_t - y_{t-1}$
3. **移动平均（MA）部分**：
   - 使用过去的误差项来预测未来的值。
   - 模型形式：$y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$

使用ARIMA进行异常检测的步骤

1. 数据准备：

- 收集并清洗时间序列数据，确保数据的完整性和一致性。

2. 构建ARIMA模型：
   - 使用历史数据拟合ARIMA模型。
   - 确定模型的参数（p, d, q），其中：
     - p 是自回归部分的阶数。
     - d 是差分的次数。
     - q 是移动平均部分的阶数。
3. 预测未来值：
   - 使用ARIMA模型预测时间序列的未来值。
4. 计算残差：
   - 计算实际值与预测值之间的残差。
5. 计算残差的统计特性：
   - 计算残差的均值和标准差。
6. 定义异常检测阈值：
   - 通常使用3倍标准差作为阈值。
7. 检测异常值：
   - 如果某个时间点的残差超出定义的阈值，则认为该时间点的数据是异常的。

检测效果展示

异常数据检测源数据-核心指标-在营门店

EWMA异常检测介绍

EWMA（Exponentially Weighted Moving Average，指数加权移动平均）是一种常用的时间序列分析方法，特别适用于检测数据中的异常值。EWMA通过赋予较新的数据更高的权重，而较旧的数据权重逐渐减小，使得模型对数据的最新变化更加敏感。以下是使用EWMA进行异常检测的具体步骤和方法。

基本概念

1. **EWMA公式**：

$$S_t = \alpha x_t + (1 - \alpha) S_{t-1}$$

其中：

- S_t 是时间 t 的EWMA值。
- x_t 是时间 t 的观测值。
- S_{t-1} 是时间 $t-1$ 的EWMA值。
- α 是平滑因子（smoothing factor），取值范围为 $0 < \alpha \leq 1$。

2. **初始值**：

- 通常，初始值 S_0 可以设置为第一个观测值 x_1，或者设置为一个合理的初始估计值。

异常检测步骤

1. 数据准备：
   - 收集时间序列数据，确保数据的时间顺序性和完整性。
2. 计算EWMA值：
   - 使用上述公式计算每个时间点的EWMA值。
3. 计算残差：
   - 计算每个时间点的实际值与EWMA值之间的残差。
4. 计算残差的统计特性：
   - 计算残差的均值和标准差。
5. 定义异常检测阈值：
   - 通常使用3倍标准差作为阈值。
6. 检测异常值：
   - 如果某个时间点的残差超出定义的阈值，则认为该时间点的数据是异常的。

检测效果展示

异常数据检测源数据-核心指标-在营门店