Yonsei University, Department of Applied Statistics

# Interpreting and Unifying Outlier Scores

12기 신보현

February 15, 2019

Reference Paper: Interpreting and Unifying Outlier Scores [by Hans-Peter Kriegel Peer Kr¨oger Erich Schubert Arthur Zimek]

0. Abstract

이상치 탐지 기법은 크게 이상치인지 아닌지 label을 만드는 기법과 이상치의 '정도'를 수치로 나타내는 기법으로 구성된다. 후자 기법의 경우, 다양한 이상치 모델이 있고 그 값의 범위, 의미가 각 모델마다 다름은 당연하다. 따라서 이러한 값들을 비교하며 해석하기가 쉽지 않다. 따라서 본 논문에서는 다양한 이상치 모델이 제시하는 값을 통일된 기준으로 비교하기 위해서, 이러한 값들을 0과 1사이의 값으로 변환하여 이상치의 정도를 나타낸다. 해당 논문에서 적용되는 기법은 ABOD, SOD, FBOD이다.

1. Introduction

이상치 모델마다 이상치 정도를 나타내는 값의 범위, 의미가 다르다. 심지어 이상치 모델 A에서의 값 x가 어떤 데이터 Z에 관해서는 이상치로 판단되는 값이지만 어떤 데이터 Y에 관해서는 이상치가 아닌 정상적인 값일수도 있다. 이러한 문제점을 방지하기 위해서 해당 논문에서는 두 가지 방법을 제시한다.

- 주어진 데이터 세트에서 특정한 데이터 분포로부터 독립적이게 만드는 정규화
- 이상치 값을 0과 1사이로 만들어 확률로 해석할 수 있게 하는 변환

2. Method

regular과 normal의 의미부터 확실하게 짚고 넘어가자. 논문의 표현을 그대로 쓰면 아래와 같다.

$$An\ outlier\ score\ \mathcal{S}\ is\ called\ regular\ if\ S(o) \geq 0\ for\ any\ object\ o$$

$$inlier\ if\ S(o) \approx 0,\ outlier\ S(o) \gg 0$$

$$An\ outlier\ score\ S\ is\ normal\ if\ S\ is\ regular\ and\ the\ values\ are\ restricted\ by\ S(o) \in [0,1]$$

regular(normal) 인 점수로 바꾸는 변환을 regular(normal) 하다고 부른다.
변환의 중요한 특징은 변환 이후에 원래 점수의 순서를 유지해야 한다는 것이다. 이를 아래와 같이 표현한다.

$$\forall o_1, o_2 :\ S(o_1) \leq S(o_2) \Longrightarrow T_S(o_1) \leq T_S(o_2)$$

또는 순서과 바뀌어도 그것을 그대로 유지하기만 하면 된다.

$$\forall o_1, o_2 :\ S(o_1) \leq S(o_2) \Longrightarrow T_S(o_1) \geq T_S(o_2)$$

이렇게 변환 이후에도 순서가 바뀌지 않으면(바뀌어도 모두 같은 방향으로 바뀐다면) 이러한 변환을 ranking-stable 이라고 부른다.

변환은 크게 두 단계로 이루어진다. 첫 번째는 regularization, 두 번째는 normalization이다. 첫 번째 단계인 regularization(앞으로 Reg이라고 부른다)에서는 이상치 점수를 $[0, \infty)$의 범위로 변환시켜, $Reg(o) \approx 0$일 때는 inliers, $Reg_S(o) \gg 0$일 때는 outliers라고 판단할 수 있는 환경을 만든다. 두 번째 단계인 normalization에서는 $[0, \infty)$의 범위에 있는 값을 $[0, 1]$ 범위 안에 들도록 변환을 한다.

### 2.1 Regularization

다른 이상치 점수는 regular인 상태가 되기 위해서 각자 다른 변환을 필요로 한다. 다양한 이상치 점수에 맞는 여러 regularization을 소개한다.

#### 2.1.1 Baseline Regularization

local outlier scores인 LOF와 LDOF 그리고 그것들의 변형은 아직 regular하지 않은데, 그 이유는 non-outliers(inliers)가 가질 것으로 예상되는 값이 0이 아니기 때문이다(앞서 이상치 점수가 0과 가까울 때, 정상치라고 판단하는 기준을 가진다면 regular하다고 했었다) LOF와 그 변형에 관해서는 inlier는 1의 값을 가질 것으로 기대된다 ($base_{LOF} = 1$) LDOF에 관해서는 $\frac{1}{2}$의 값을 가질 것으로 기대된다($base_{LDOF} = \frac{1}{2}$) 각 모델에 관해서는 $\gg base$라면 이상치라고 판단된다. 이를 regular하게 만들기 위해서, $o$의 이상치 점수인 $S(o)$에서 baseline 값인 $base_S$을 빼는 변환을 생각해보자. 원래 이상치 점수는 $[base, \infty)$의 범위에 있었지만 이러한 변환을 통해서 $[0, \infty)$의 범위에 있게 될 것이다. 변환되기 전의 이상치 점수가 $base_S$보다 작아도 inliers이라고 판명했으므로, 음수 값이 나오지 않기 위해서 다음과 같이 조정을 한다.

$$Reg_S^{base_S}(o) := max\{0, S(o) - base_S\}$$

해당 변환은 ranking-stable하다.

$$\begin{aligned} S(o_1) \leq S(o_2) \Longleftrightarrow & S(o_1) - base_S \leq S(o_2) - base_S \\ \Longrightarrow & max\{0, S(o_1) - base_S\} \leq max\{0, S(o_2) - base_S\} \\ \Longleftrightarrow & Reg_S^{base_S}(o_1) \leq Reg_S^{base_S}(o_2) \end{aligned}$$

#### 2.1.2 Linear Inversion

어떤 모델에서는 높은 점수가 inliers을 의미하기도 한다. 만약 가우시안 모델의 밀도함수를 사용한다면 높은 밀도는 inliers을 의미하고 0에 가까운 밀도는 outliers을 의미할 것이다. 이러한 모델을 regularize하기 위해 아래와 같은 변환을 생각한다.

$$Reg_S^{lininv}(o) := S_{max} - S(o)$$

$S_{max} \geq S(o)$이기 때문에 이 변환은 regular하다(범위가 0이상 이므로) ranking-stability또한 쉽게 보일 수 있다.

$$\begin{aligned} S(o_1) \leq S(o_2) &\Longleftrightarrow -S(o_1) \geq -S(o_2) \\ &\Longleftrightarrow Reg_S^{lininv}(o_1) \geq Reg_S^{lininv}(o_2) \end{aligned}$$

2.1.3 Logarithmic Inversion

위에서 언급된 변환은 단순 뺄셈이기 때문에, 매우 낮은 contrast(여기서 contrast는 outlier와 inlier의 차이, 즉 ABOD가 낮은 contrast을 가진다는 것은 outlier와 inlier가 큰 차이가 없다는 뜻??)을 가지는 ABOD와 같은 방법에는 적절하지 않다. 이를 위해 아래와 같은 변환을 생각한다.

$$Reg_S^{loginv}(o) \coloneqq -log(S(o)/S_{max})$$

ABOD는 0보다 큰 값을 가지기 때문에 위의 변환을 사용할 수 있다. 또한 로그 함수가 단조 증가 함수이기 때문에 로그 변환은 ranking-stable하다.

2.2 Normalization

이상치 점수가 normalized scale에 있게 하기 위한 가장 간단한 방법은 최소값이 0이 되도록하는 선형 변환(simple linear normalization)을 적용하는 것이다.

$$Norm_S^{linear}(o) \coloneqq \frac{S(o) - S_{min}}{S_{max} - S_{min}}$$

여기서 주목할 점은 $S$가 regular하다면 $S_{min} = 0$이고 위의 공식을 간단하게할 수 있다. 하지만 이는 점수의 분포에 contrast을 추가하지 않는다. 따라서 아래에는 outlier와 inlier 사이의 차이, 즉 contrast을 더 강화하는 방법에 대해서 살펴본다.

2.2.1 Statistical Scaling of Outlier Scores

데이터의 분포에 대한 가정을 하지 않고 이상치 점수의 분포를 분석하는 것이 핵심이다. 이를 위해서 uniform 분포를 가정한다면 과적합이 일어날 수 있으므로 특정 모수를 통해 여러 모양을 만들어내는 다른 분포를 생각하는 것이 좋다. 이러한 방법의 직관은 이상치 점수를 직접적으로 해석할 방법은 없지만 얼마나 '평범하지 않은지'(unusual) 그 정도를 평가하는 것이다. 논문에서는 가우시안 분포와 감마 분포를 예시로 든다. 하지만 다른 분포도 적용될 수 있다. 예를 들어서 비율에 기반한 LOF나 LDOF는 코쉬분포와 F 분포가 좋은 후보이다.
가장 최적의 분포 선택은 알고리즘, 데이터 세트에 달려있다. 따라서 여기서는 최적의 분포가 어떤 것이라고 제시하지는 않는다. 하지만 여러 실험을 통해서 분포를 임의로 선택하는 것도 꽤 성능을 향상시킴이 밝혀졌다.

2.2.2 Gaussian Scaling

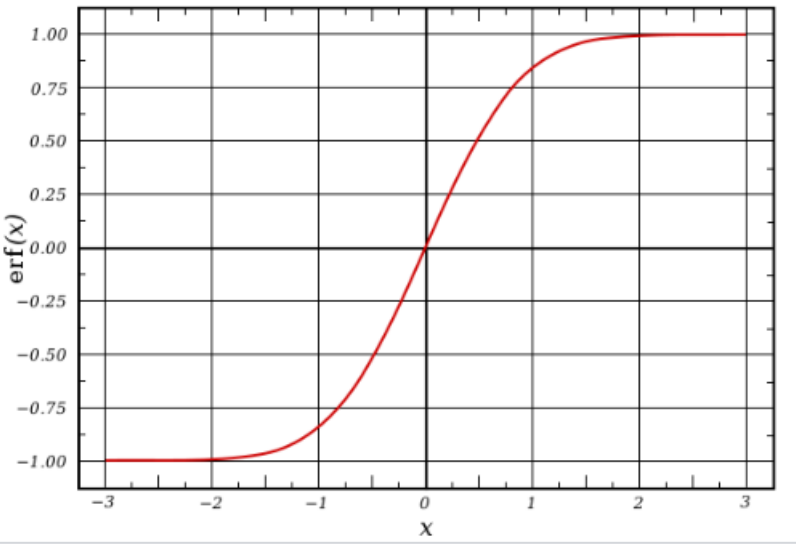
가우시안 스케일링은 주어진 이상치 점수들의 평균($\mu_S$)과 표준편차($\sigma_S$)와 누적분포함수(cumulative distribution function) 그리고 가우시안 에러 함수(Gaussian error function: $erf()$)을 이용하여 이상치 점수를 확률 값으로 변화시킨다.

$$Norm_S^{gauss}(o) \coloneqq max\left\{0, erf\left(\frac{S(o) - \mu_S}{\sigma_S \cdot \sqrt{2}}\right)\right\}$$

$$where\ \widehat{\mu} = \frac{1}{n}\sum_{i=1}^{n} S(o_i),\ \widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} S(o_i)^2 - \left(\frac{1}{n}\sum_{i=1}^{n} S(o_i)\right)^2$$

이 정의를 자세히 살펴보자. 우선 가우시안 에러 함수의 정의는 아래와 같다.

$$
\begin{aligned}
erf(x) &:= P(-x \leq Y \leq x) \; where \; Y \sim N\left[0, \frac{1}{2}\right] \\
&= \int_{-x}^{x} \frac{1}{\sqrt{2\pi \cdot \frac{1}{2}}} exp\left\{-\frac{1}{2 \cdot \frac{1}{2}} t^2\right\} dt \\
&= \int_{-x}^{x} \frac{1}{\sqrt{\pi}} exp\left\{-t^2\right\} dt \\
&= 2\int_{0}^{x} \frac{1}{\sqrt{\pi}} exp\left\{-t^2\right\} dt
\end{aligned}
$$



Plot of the error function

시그모이드 함수와 유사한 모양이고 함수 값은 -1에서 1까지 가지는 것을 확인할 수 있다. 위 그림에서 볼 수 있듯이 에러 함수는 x가 0이하이면 음수 값을 가진다. 따라서 $Norm_S^{gauss}(o)$을 정의할 때 0과 함수값의 max로 정함으로써 음수가 나오는 것을 방지한 것으로 보인다. 그렇다면 $Norm_S^{gauss}(o) := max\left\{0, erf\left(\frac{S(o)-\mu_S}{\sigma_S \cdot \sqrt{2}}\right)\right\}$에서 $erf\left(\frac{S(o)-\mu_S}{\sigma_S \cdot \sqrt{2}}\right)$가 음수가 나올 경우에 이 값을 0으로 대체해버린다는 뜻인데 이렇게 하는 것이 올바른 방법일까? $erf\left(\frac{S(o)-\mu_S}{\sigma_S \cdot \sqrt{2}}\right)$가 음수라는 얘기는 위 그래프에서 볼 수 있듯이 $\frac{S(o)-\mu_S}{\sigma_S \cdot \sqrt{2}}$가 음수여야 하고 분모는 무조건 양수이므로 분자인 $S(o)-\mu_S$가 음수여야한다. $S(o)-\mu_S$가 음수라는 얘기는 $S(o) < \mu_S$라는 뜻이다. 즉, 어떤 점 $o$에 대한 이상치 점수인 $S(o)$가 이상치 점수들의 평균보다 작다는 것이다. 이상치 점수가 작다는 것은 무엇을 의미할까? 이상치일 가능성이 작다는 것이다. 이상치일 가능성이 작다는 것은 $Norm_S^{gauss}(o)$값을 0으로 만드는 것과 어느정도 일맥상통한다. $Norm_S^{gauss}(o)$은 $[0,1]$의 범위에 있고 이는 이상치일 가능성을 뜻하기 때문에 $Norm_S^{gauss}(o)$이 0이라는 것은 이상치일 가능성이 작다는 뜻이기 때문이다.

그렇다면 왜 하필 erf의 함수값으로 $\frac{S(o)-\mu_S}{\sigma_S \cdot \sqrt{2}}$ 을 사용할까? 이것은 표준 정규분포의 cdf 함수와 erf 함수와의 관계와 관련이 있다. 표준 정규분포의 cdf 함수는 아래와 같이 정의된다.

$$\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} exp\left\{-\frac{1}{2}x^2\right\}$$

이를 이용해서 erf 함수를 변형하면 아래와 같다.

$$\begin{aligned}
erf(x) &= 2\int_0^x \frac{1}{\sqrt{\pi}} exp\left\{-t^2\right\} dt \\
&Let\ t = \frac{k}{\sqrt{2}}, then\ \frac{dt}{dk} = \frac{1}{\sqrt{2}} \\
&= \frac{2}{\sqrt{\pi}} \int_0^{x\sqrt{2}} exp\left\{-\frac{k^2}{2}\right\} dk \cdot \frac{1}{\sqrt{2}} \\
&= 2\int_0^{x\sqrt{2}} \frac{1}{\sqrt{2\pi}} exp\left\{-\frac{k^2}{2}\right\} dk \\
&= 2\left[\int_{-\infty}^{x\sqrt{2}} \frac{1}{\sqrt{2\pi}} exp\left\{-\frac{k^2}{2}\right\} dk - \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} exp\left\{-\frac{k^2}{2}\right\} dk\right] \\
&= 2\left[\Phi(x\sqrt{2}) - \Phi(0)\right] \\
&= 2\left[\Phi(x\sqrt{2}) - \frac{1}{2}\right]
\end{aligned}$$

$$\therefore \Phi(x\sqrt{2}) = \frac{1}{2}\left[erf(x) + 1\right]$$

$$\iff \Phi(x) = \frac{1}{2}\left[erf\left(\frac{x}{\sqrt{2}}\right) + 1\right]$$

그런데, 여기서 $x$는 표준화된 값을 의미한다. 따라서 만약, $x$가 표준화되지 않았다면 이를 표준화시켜줘야한다. 즉, 다시 말해서

$$\Phi(x, \mu, \sigma) = \frac{1}{2}\left[erf\left(\frac{x-\mu}{\sigma \cdot \sqrt{2}}\right) + 1\right]$$

표본을 통해서는 $\mu, \sigma$에 대한 추정치를 통해서 표준화를 진행하면 된다. 위 식을 이상치 값인 $S(o)$ 관점에서 보면 아래와 같다.

$$cdf_S^{gauss}(o) \coloneqq \frac{1}{2}\left[erf\left(\frac{S(o)-\mu_S}{\sigma_s \cdot \sqrt{2}}\right) + 1\right]$$

여기서 $cdf_S^{gauss}(o)$는 관측된 $S(o)$에 대한 표본 cdf을 의미한다. 해당 cdf 함수에 $\mu_{cdf} = cdf_S^{gauss}(\mu_S) = \frac{1}{2}, max_{cdf} = 1$을 적용하여 ***linear regularization***을 시행하면 아래와 같다.

$$\begin{aligned}
\frac{cdf_S^{gauss}(o) - \mu_{cdf}}{max_{cdf} - \mu_{cdf}} &= \frac{cdf_S^{gauss}(o) - 1/2}{1 - 1/2} \\
&= 2 \cdot \{cdf_S^{gauss} - 1\}
\end{aligned}$$

이를 이용하면 cdf의 linear regularization을 시행한 것이 결국 맨 앞에서 정의한 $Norm_S^{gauss}(o)$와 동일함을 알 수

있다.

$$max\left\{0, \frac{cdf_S^{gauss}(o) - \mu_{cdf}}{max_{cdf} - \mu_{cdf}}\right\} = max\left\{0, 2 \cdot \{cdf_S^{gauss} - 1\}\right\} = Norm_S^{gauss}(o)$$

<결론>

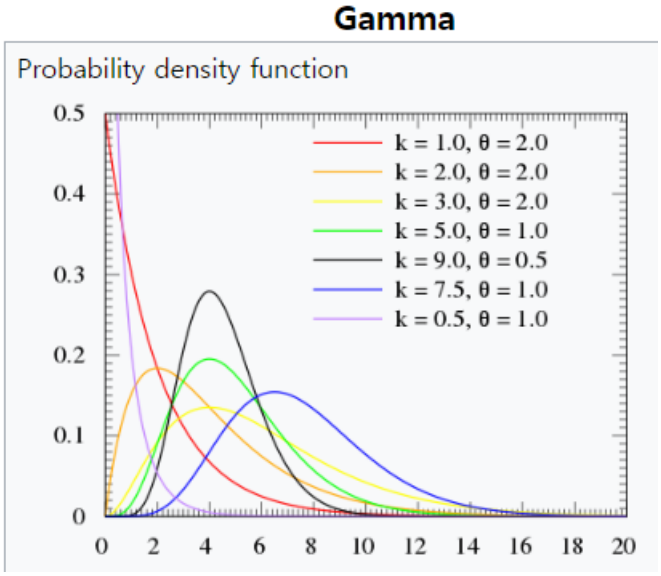
$Norm_S^{gauss}(o) := max\left\{0, erf\left(\frac{S(o) - \mu_S}{\sigma_S \cdot \sqrt{2}}\right)\right\}$라는 정의는 ***cdf의 linear regularization***으로부터 비롯되었다. 그런데 여기서 살펴보아야할 의미는 아래와 같다.

1. linear regularization? $\frac{cdf_S^{gauss}(o) - \mu_{cdf}}{max_{cdf} - \mu_{cdf}} \Longrightarrow$ 최소값을 -1, 최대값을 1로 만들어주는 regularization(cdf의 최대값이 1, 최소값이 0이기 때문에)

2. cdf를 왜 썼을까? 이상치의 분포가 정규분포를 따른다고 생각을 하고 모수 $\mu, \sigma$를 적률추정량을 통해서 추정한 후, 추정된 모수로 만든 $cdf$는 $cdf$의 특징상 0과 1사이의 값을 가진다. 그리고 이상치 점수가 크면 클수록 이상치일 가능성이 높다는 뜻인데, 이상치 점수가 큰 관측치는 해당 $cdf$ 값이 클 것이다. 이러한 특징 때문에 $cdf$를 사용한 것처럼 보인다.

### 2.2.3 Gamma Scaling

$X \sim Gamma(k, \theta)$일 때, 모수에 따른 감마 분포의 모양을 살펴보면 아래와 같다.




정규 분포보다는 좀 더 다양한 모양을 표현할 수 있기 때문에 감마 스케일링을 사용하기도 하는 것 같다. 감마 분포의 모수는 앞서 가우시안 분포에서와 마찬가지로 적률추정량을 이용하여 추정을 한다. $\widehat{k} := \frac{\widehat{\mu}^2}{\widehat{\sigma}^2}, \widehat{\theta} := \frac{\widehat{\sigma}}{\widehat{\mu}^2}$

가우시안 스케일링과 유사하게, 감마 스케일링도 cdf의 linear regularization을 이용하여 아래와 같이 정의한다.

$$Norm_S^{gamma}(o) := max\left\{0, \frac{cdf_S^{gamma}(o) - \mu_{cdf}}{max_{cdf} - \mu_{cdf}}\right\}$$

결국 분포를 이용한 normalization에서 어떤 분포든간에, 해당 분포 cdf의 linear regularization을 이용하는데, 정규분포인 경우에는 그 형태가 특별하게 $erf$을 사용하여 나타낼 수 있었던 것 뿐이다.

3. Practice

```
load("C:/Users/sbh0613/Desktop/와이빅타/이상치분석/Func.trans/workspace.RData")
library(HighDimOut)

## Warning: package 'HighDimOut' was built under R version 3.5.2

data(TestData)
mydata = TestData
```

```
# ordinary ABOD
abod.result = Func.ABOD(mydata[,-3],basic=T,perc=1)
```

```
mydata[order(abod.result,decreasing=F),] # abod는 값이 작을 수록 outlier라고 판단.

##             x         y       Lab
## 58 0.66079779 0.9919061   Outlier
## 60 0.06178627 0.7774452   Outlier
## 57 0.94467527 0.7176185   Outlier
## 54 0.90820779 0.3841037   Outlier
## 55 0.20168193 0.7698414   Outlier
## 56 0.89838968 0.4976992   Outlier
## 46 0.07760997 0.2495257 Pattern_2
## 40 0.54413646 0.2500764 Pattern_2
## 59 0.62911404 0.3800352   Outlier
## 53 0.57285336 0.6870228   Outlier
## 52 0.37212390 0.1765568   Outlier
## 28 0.47579031 0.2470755 Pattern_2
## 34 0.14362179 0.2514280 Pattern_2
## 51 0.26550866 0.2059746   Outlier
## 36 0.38320471 0.2509809 Pattern_2
## 26 0.42383041 0.2495390 Pattern_2
## 5  0.64701682 0.5108079 Pattern_1
## 23 0.65151674 0.7151912 Pattern_1
```

```
## 47 0.17363856 0.2467513 Pattern_2
## 11 0.64705975 0.6804190 Pattern_1
## 32 0.18334295 0.2457706 Pattern_2
## 14 0.64884104 0.7085769 Pattern_1
## 48 0.33587289 0.2524670 Pattern_2
## 24 0.64625555 0.6992160 Pattern_1
## 50 0.20593508 0.2536454 Pattern_2
## 42 0.29451225 0.2510154 Pattern_2
## 35 0.41565370 0.2542862 Pattern_2
## 33 0.19344094 0.2453554 Pattern_2
## 7  0.65444675 0.4762462 Pattern_1
## 30 0.25472160 0.2509571 Pattern_2
## 16 0.64997699 0.4715401 Pattern_1
## 10 0.64561786 0.6133336 Pattern_1
## 49 0.29889545 0.2460499 Pattern_2
## 4  0.65408208 0.6252223 Pattern_1
## 12 0.64676557 0.5942893 Pattern_1
## 19 0.64880035 0.5457112 Pattern_1
## 29 0.35607461 0.2472866 Pattern_2
## 17 0.65217619 0.6046726 Pattern_1
## 44 0.36182433 0.2475817 Pattern_2
## 22 0.64712143 0.6726751 Pattern_1
## 13 0.65187023 0.6503608 Pattern_1
## 15 0.65269841 0.5309046 Pattern_1
## 39 0.26232973 0.2548510 Pattern_2
## 31 0.21679567 0.2507487 Pattern_2
## 45 0.28273765 0.2522931 Pattern_2
## 6  0.65398390 0.6435683 Pattern_1
## 27 0.27206537 0.2501117 Pattern_2
## 21 0.65434705 0.5350528 Pattern_1
## 38 0.32661374 0.2502603 Pattern_2
## 9  0.65129114 0.6377396 Pattern_1
## 37 0.27726713 0.2506090 Pattern_2
## 43 0.32501413 0.2473887 Pattern_2
## 3  0.65072853 0.5588489 Pattern_1
## 41 0.22046609 0.2518279 Pattern_2
## 8  0.65160798 0.5775732 Pattern_1
```

```
## 18 0.65491906 0.5764293 Pattern_1
## 25 0.64767221 0.5570487 Pattern_1
## 20 0.65277445 0.5566690 Pattern_1
## 2  0.64872124 0.5700785 Pattern_1
## 1  0.64765509 0.5710538 Pattern_1
```

```
# Fast ABOD
# Feature Bagging
fbod.result = Func.FBOD(mydata[,-3],iter=10,k.nn=5)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
mydata[order(fbod.result,decreasing=T),] # fbod는 값이 클 수록 outlier라고 판단.
```

```
##             x         y       Lab
## 57 0.94467527 0.7176185   Outlier
## 54 0.90820779 0.3841037   Outlier
## 56 0.89838968 0.4976992   Outlier
## 52 0.37212390 0.1765568   Outlier
## 53 0.57285336 0.6870228   Outlier
## 51 0.26550866 0.2059746   Outlier
## 40 0.54413646 0.2500764 Pattern_2
## 59 0.62911404 0.3800352   Outlier
## 58 0.66079779 0.9919061   Outlier
## 60 0.06178627 0.7774452   Outlier
## 46 0.07760997 0.2495257 Pattern_2
## 55 0.20168193 0.7698414   Outlier
## 45 0.28273765 0.2522931 Pattern_2
## 39 0.26232973 0.2548510 Pattern_2
## 35 0.41565370 0.2542862 Pattern_2
## 16 0.64997699 0.4715401 Pattern_1
## 7  0.65444675 0.4762462 Pattern_1
## 48 0.33587289 0.2524670 Pattern_2
## 41 0.22046609 0.2518279 Pattern_2
## 34 0.14362179 0.2514280 Pattern_2
## 50 0.20593508 0.2536454 Pattern_2
## 26 0.42383041 0.2495390 Pattern_2
```

```
## 15 0.65269841 0.5309046 Pattern_1
## 21 0.65434705 0.5350528 Pattern_1
## 10 0.64561786 0.6133336 Pattern_1
## 5  0.64701682 0.5108079 Pattern_1
## 19 0.64880035 0.5457112 Pattern_1
## 17 0.65217619 0.6046726 Pattern_1
## 28 0.47579031 0.2470755 Pattern_2
## 14 0.64884104 0.7085769 Pattern_1
## 24 0.64625555 0.6992160 Pattern_1
## 33 0.19344094 0.2453554 Pattern_2
## 27 0.27206537 0.2501117 Pattern_2
## 47 0.17363856 0.2467513 Pattern_2
## 2  0.64872124 0.5700785 Pattern_1
## 4  0.65408208 0.6252223 Pattern_1
## 12 0.64676557 0.5942893 Pattern_1
## 18 0.65491906 0.5764293 Pattern_1
## 6  0.65398390 0.6435683 Pattern_1
## 49 0.29889545 0.2460499 Pattern_2
## 22 0.64712143 0.6726751 Pattern_1
## 42 0.29451225 0.2510154 Pattern_2
## 9  0.65129114 0.6377396 Pattern_1
## 43 0.32501413 0.2473887 Pattern_2
## 36 0.38320471 0.2509809 Pattern_2
## 8  0.65160798 0.5775732 Pattern_1
## 32 0.18334295 0.2457706 Pattern_2
## 38 0.32661374 0.2502603 Pattern_2
## 13 0.65187023 0.6503608 Pattern_1
## 37 0.27726713 0.2506090 Pattern_2
## 30 0.25472160 0.2509571 Pattern_2
## 23 0.65151674 0.7151912 Pattern_1
## 3  0.65072853 0.5588489 Pattern_1
## 29 0.35607461 0.2472866 Pattern_2
## 1  0.64765509 0.5710538 Pattern_1
## 44 0.36182433 0.2475817 Pattern_2
## 11 0.64705975 0.6804190 Pattern_1
## 31 0.21679567 0.2507487 Pattern_2
## 20 0.65277445 0.5566690 Pattern_1
```

```
## 25 0.64767221 0.5570487 Pattern_1
```

```
# Func.SOD
sod.result = Func.SOD(mydata[,-3],k.nn=10,k.sel=5,alpha=0.8)
getwd()
```

```
## [1] "C:/Users/sbh0613/Desktop/와이빅타/이상치분석/Func.trans"
```

```
mydata[order(sod.result,decreasing=T),] # sod는 값이 클 수록 outlier라고 판단.
```

```
##             x         y       Lab
## 60 0.06178627 0.7774452   Outlier
## 55 0.20168193 0.7698414   Outlier
## 57 0.94467527 0.7176185   Outlier
## 58 0.66079779 0.9919061   Outlier
## 59 0.62911404 0.3800352   Outlier
## 54 0.90820779 0.3841037   Outlier
## 53 0.57285336 0.6870228   Outlier
## 52 0.37212390 0.1765568   Outlier
## 56 0.89838968 0.4976992   Outlier
## 51 0.26550866 0.2059746   Outlier
## 49 0.29889545 0.2460499 Pattern_2
## 36 0.38320471 0.2509809 Pattern_2
## 23 0.65151674 0.7151912 Pattern_1
## 42 0.29451225 0.2510154 Pattern_2
## 16 0.64997699 0.4715401 Pattern_1
## 14 0.64884104 0.7085769 Pattern_1
## 27 0.27206537 0.2501117 Pattern_2
## 11 0.64705975 0.6804190 Pattern_1
## 24 0.64625555 0.6992160 Pattern_1
## 18 0.65491906 0.5764293 Pattern_1
## 35 0.41565370 0.2542862 Pattern_2
## 10 0.64561786 0.6133336 Pattern_1
## 50 0.20593508 0.2536454 Pattern_2
## 12 0.64676557 0.5942893 Pattern_1
## 19 0.64880035 0.5457112 Pattern_1
## 1  0.64765509 0.5710538 Pattern_1
```

```
## 33 0.19344094 0.2453554 Pattern_2
## 7  0.65444675 0.4762462 Pattern_1
## 32 0.18334295 0.2457706 Pattern_2
## 28 0.47579031 0.2470755 Pattern_2
## 34 0.14362179 0.2514280 Pattern_2
## 20 0.65277445 0.5566690 Pattern_1
## 4  0.65408208 0.6252223 Pattern_1
## 25 0.64767221 0.5570487 Pattern_1
## 2  0.64872124 0.5700785 Pattern_1
## 15 0.65269841 0.5309046 Pattern_1
## 17 0.65217619 0.6046726 Pattern_1
## 21 0.65434705 0.5350528 Pattern_1
## 41 0.22046609 0.2518279 Pattern_2
## 47 0.17363856 0.2467513 Pattern_2
## 8  0.65160798 0.5775732 Pattern_1
## 31 0.21679567 0.2507487 Pattern_2
## 46 0.07760997 0.2495257 Pattern_2
## 3  0.65072853 0.5588489 Pattern_1
## 26 0.42383041 0.2495390 Pattern_2
## 5  0.64701682 0.5108079 Pattern_1
## 9  0.65129114 0.6377396 Pattern_1
## 40 0.54413646 0.2500764 Pattern_2
## 6  0.65398390 0.6435683 Pattern_1
## 13 0.65187023 0.6503608 Pattern_1
## 22 0.64712143 0.6726751 Pattern_1
## 29 0.35607461 0.2472866 Pattern_2
## 30 0.25472160 0.2509571 Pattern_2
## 37 0.27726713 0.2506090 Pattern_2
## 38 0.32661374 0.2502603 Pattern_2
## 39 0.26232973 0.2548510 Pattern_2
## 43 0.32501413 0.2473887 Pattern_2
## 44 0.36182433 0.2475817 Pattern_2
## 45 0.28273765 0.2522931 Pattern_2
## 48 0.33587289 0.2524670 Pattern_2


# Transform outlier scores into range 0 and 1
trans.abod = Func.trans(abod.result,method="ABOD")
```

```r
trans.fbod = Func.trans(fbod.result,method="FBOD")
trans.sod = Func.trans(sod.result,method="SOD")
trans.merge = trans.abod + trans.fbod + trans.sod
mydata[order(trans.merge,decreasing=T),]
```

```
##             x         y       Lab
## 57 0.94467527 0.7176185   Outlier
## 54 0.90820779 0.3841037   Outlier
## 58 0.66079779 0.9919061   Outlier
## 60 0.06178627 0.7774452   Outlier
## 56 0.89838968 0.4976992   Outlier
## 55 0.20168193 0.7698414   Outlier
## 59 0.62911404 0.3800352   Outlier
## 53 0.57285336 0.6870228   Outlier
## 52 0.37212390 0.1765568   Outlier
## 40 0.54413646 0.2500764 Pattern_2
## 51 0.26550866 0.2059746   Outlier
## 46 0.07760997 0.2495257 Pattern_2
## 28 0.47579031 0.2470755 Pattern_2
## 34 0.14362179 0.2514280 Pattern_2
## 36 0.38320471 0.2509809 Pattern_2
## 26 0.42383041 0.2495390 Pattern_2
## 5  0.64701682 0.5108079 Pattern_1
## 23 0.65151674 0.7151912 Pattern_1
## 1  0.64765509 0.5710538 Pattern_1
## 2  0.64872124 0.5700785 Pattern_1
## 3  0.65072853 0.5588489 Pattern_1
## 4  0.65408208 0.6252223 Pattern_1
## 7  0.65444675 0.4762462 Pattern_1
## 8  0.65160798 0.5775732 Pattern_1
## 9  0.65129114 0.6377396 Pattern_1
## 10 0.64561786 0.6133336 Pattern_1
## 11 0.64705975 0.6804190 Pattern_1
## 12 0.64676557 0.5942893 Pattern_1
## 14 0.64884104 0.7085769 Pattern_1
## 15 0.65269841 0.5309046 Pattern_1
## 16 0.64997699 0.4715401 Pattern_1
```

```
## 17 0.65217619 0.6046726 Pattern_1
## 18 0.65491906 0.5764293 Pattern_1
## 19 0.64880035 0.5457112 Pattern_1
## 20 0.65277445 0.5566690 Pattern_1
## 21 0.65434705 0.5350528 Pattern_1
## 24 0.64625555 0.6992160 Pattern_1
## 25 0.64767221 0.5570487 Pattern_1
## 27 0.27206537 0.2501117 Pattern_2
## 31 0.21679567 0.2507487 Pattern_2
## 32 0.18334295 0.2457706 Pattern_2
## 33 0.19344094 0.2453554 Pattern_2
## 35 0.41565370 0.2542862 Pattern_2
## 41 0.22046609 0.2518279 Pattern_2
## 42 0.29451225 0.2510154 Pattern_2
## 47 0.17363856 0.2467513 Pattern_2
## 49 0.29889545 0.2460499 Pattern_2
## 50 0.20593508 0.2536454 Pattern_2
## 6  0.65398390 0.6435683 Pattern_1
## 13 0.65187023 0.6503608 Pattern_1
## 22 0.64712143 0.6726751 Pattern_1
## 29 0.35607461 0.2472866 Pattern_2
## 30 0.25472160 0.2509571 Pattern_2
## 37 0.27726713 0.2506090 Pattern_2
## 38 0.32661374 0.2502603 Pattern_2
## 39 0.26232973 0.2548510 Pattern_2
## 43 0.32501413 0.2473887 Pattern_2
## 44 0.36182433 0.2475817 Pattern_2
## 45 0.28273765 0.2522931 Pattern_2
## 48 0.33587289 0.2524670 Pattern_2
```