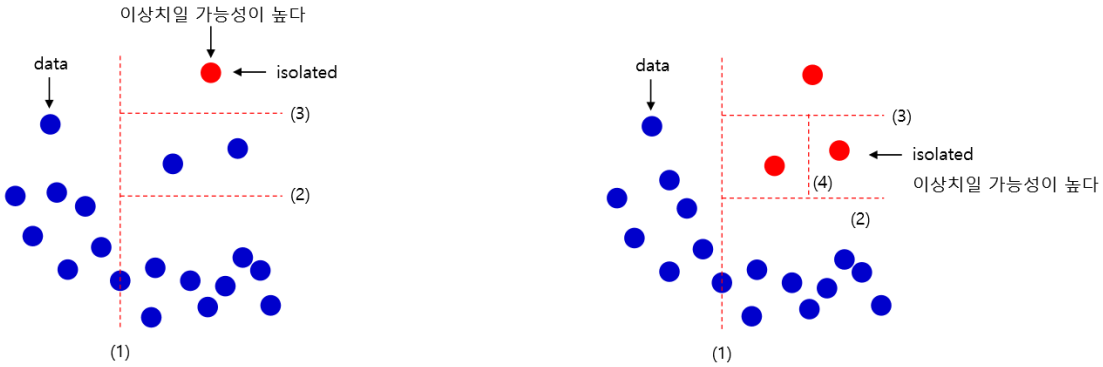
출처가 명시되지 않은 모든 자료(이미지 등)는 조성현 강사님 블로그 및 강의 자료 기반.

<< 머신러닝 - 앙상블 >>

# [Isolation Forest]

## 1. 개념




**이상 데이터일수록 *빠르게* 고립된다**는 아이디어에서 시작한다. 이상 데이터와 정상 데이터가 섞여 있는 데이터셋을 의사결정나무 알고리즘에 의해 분류했을 때, 이상치일 수록 depth가 낮은 리프 노드에 혼자 남게 될 가능성이 높아진다는 말이다.

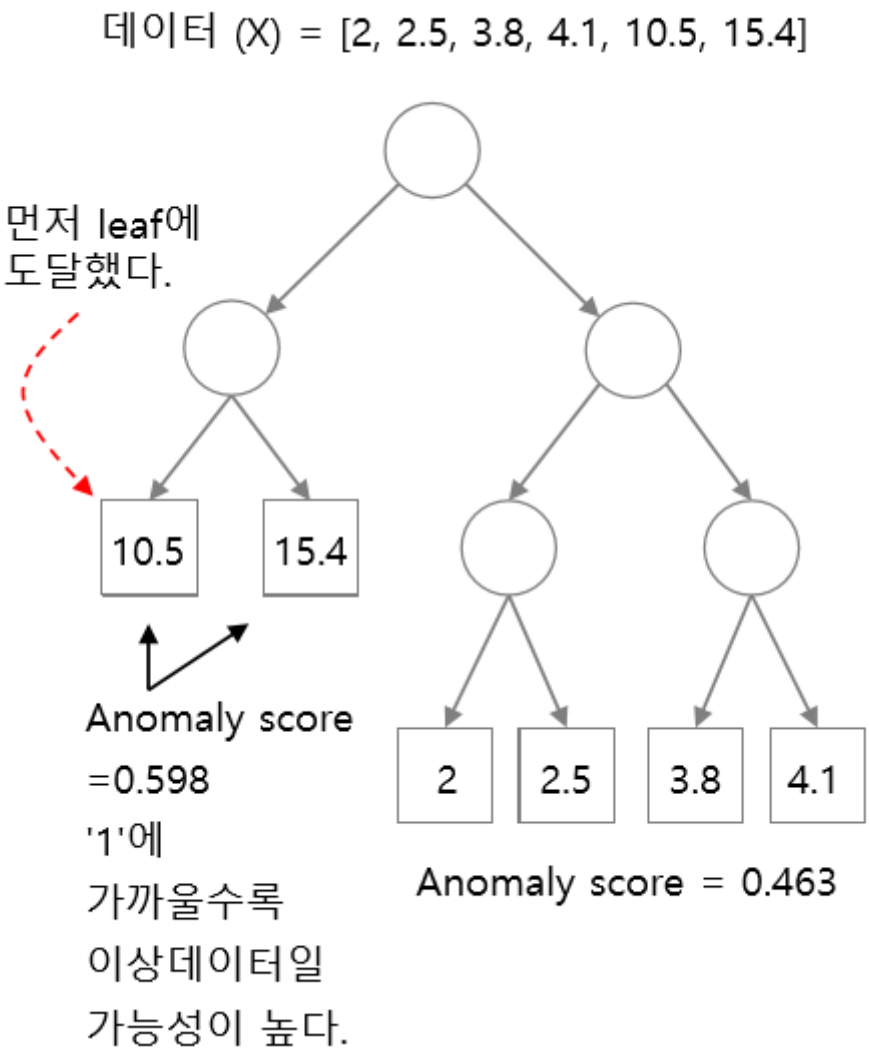
정상 데이터들일수록 일정 범위 안에 몰려 있기 때문에, 분리하기 위해서는 트리 깊이가 깊어져야 하고 분기가 많아져야 한다. 반면 이상치일수록 별로 분기하지 않더라도 쉽게 분리할 수 있다. 예컨대 위의 그림 중 왼쪽에서 빨간색 데이터를 이상치로 본다면, (3)까지만 분기하면 된다. 이 때 빨간색 이상 데이터의 depth는 2이다. 파란 데이터들의 경우, 분류하려면 더 깊이 들어가야 하기 때문에 depth가 빨간 데이터보다 커진다.

## 2. 분류 알고리즘

데이터 분류 시, 트리의 평균 depth를 계산하여 그보다 depth가 작은 데이터는 이상 데이터, 큰 데이터는 정상 데이터라고 간주한다. 위에서 보았듯, 이상한 원소가 껴 있으면 그 원소가 분류된 트리 노드의 depth는 작을 것이기 때문이다.

> 정상 데이터가 고립되려면, 트리의 말단 노드에 가까운 depth를 가진다. 이상 데이터가 고립되려면, 트리의 루트 노드에 가까운 depth를 가진다. _출처: 데이터과학 삼학년

아래의 교재 예시 트리를 통해 어떻게 트리의 평균 depth를 계산할 수 있는지 생각해 보자.




## Rough한 계산

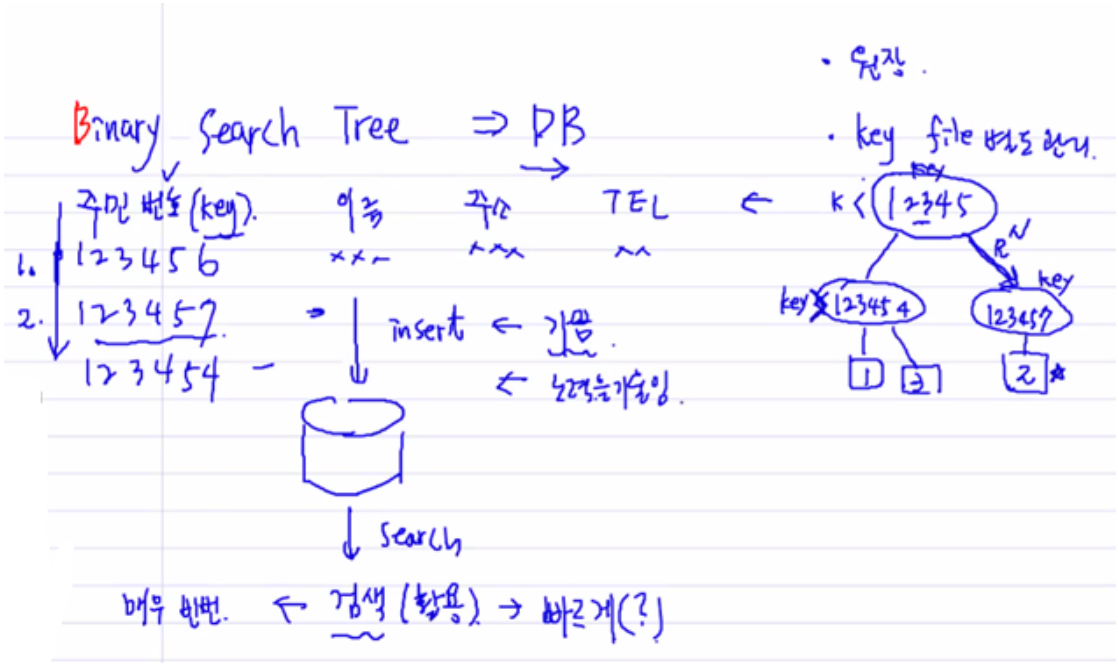
단순하게 생각한다면, 각 데이터별로 depth를 계산하여 평균을 낼 수 있다.

$$depth = [2,\ 2,\ 3,\ 3,\ 3,\ 3]$$
$$E(depth) = \frac{2 \times 2 + 3 \times 4}{6} \fallingdotseq 2.67$$

## Average Path Length

위와 같이 rough한 계산 말고, 트리 노드 개수로 평균 depth를 측정해 보자. 이진 탐색 트리의 노드 개수를 통해 Average Path Length를 계산하는 식을 차용한다.

`Isolation Forest` 알고리즘을 설계한 사람이 논문에서 평균 depth를 구하기 위해, BST의 평균 depth를 구하는 공식을 사용하겠다고 말한 것이다. 그 배경에 대해 너무 깊이 들어가지는 않아도 된다.

> *참고*
>
> 
> 
>
> 보통 DB에서 탐색을 위해 이진 트리를 사용한다.
>
> > - `indexed` : 이진 트리 DB.
> > - `hash` : 해시값 사용 DB.
>
> 이진 탐색 트리에서 평균적으로 얼마 동안 탐색해야 원하는 값을 찾을 수 있는지 찾아내는 공식이다. DB 이론에서 자주 사용하는 공식이다. max_depth의 경우 끝까지 내려갔는데도 찾지 못한 경우다.

`BST Average Path Depth` 공식을 차용해 `Isolation Forest`에서 생성한 트리의 평균 depth를 구한다.

$$c(n) = 2H(n-1) - \left(\frac{2(n-1)}{n}\right)$$

$$H(i) = \log(i) + 0.5772156649\ (Euler's\ constant)$$

- `n` : 리프 노드의 개수.
- `C(n)` : 트리의 평균 depth.
- `H(i) a.k.a h(i)` : 데이터 `i` 까지의 depth 계산.

위의 공식에 의해 교재 예시에서의 평균 depth를 다시 계산해 보면, 2.70 정도가 나온다. Rough하게 계산한 것과 비슷한 수치다.

## Anomaly Score

이제 위에서 계산한 Average Path Depth를 데이터의 평균 depth로 하여, 각 데이터의 이상치 점수를 계산하는 데 사용한다. 각각 데이터의 depth가 평균 depth보다 깊은지 얕은지 `Anomaly Score`를 계산하기 위한 기준 수치 역할을 한다.

이제 각 데이터의 Anomaly Score를 계산하는 공식은 다음과 같다.

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

- `x` : 각 데이터, `n` : 리프 노드의 개수.
- `S(x, n)` : 각 데이터의 이상치 점수.
- `E(h(x))` : 기본적으로 앙상블 방법이기 때문에 여러 트리를 형성하고, 그 트리 각각에 대해 `h(x)` 점수를 계산한다. 그 값의 기댓값을 취한 것을 이상치 점수 계산에 활용한다.

위의 Score는 `Isolation Forest` 알고리즘을 만든 저자가 정의한 점수이다. 예컨대, 교재 예시에서 2라는 데이터의 Anomaly Score를 계산하면 다음과 같다.

$$S(2,6) = 2^{-\frac{3}{2.7}} \fallingdotseq 0.46$$

이제 이 식이 어떤 의미를 가지는지 해석해 보자.

`h(x)`가 데이터 x의 path 길이가 된다. 그리고, 앙상블 모형이기 때문에 여러 트리를 만들고 그 많은 트리들 중에서 `h(x)`의 평균적인 값을 낸 데이터 `x`의 평균 depth가 `E(h(x))`라고 하자. `E(h(x))`가 어떤 값에 가까운지에 따라 `Anomaly Score`가 달라진다.

$$(1)\ E(h(x)) \to C(n) : s \to 0.5$$
$$(2)\ E(h(x)) \to 0 : s \to 1$$
$$(3)\ E(h(x)) \to n-1 : s \to 0$$

먼저 (1)의 경우, 데이터 x의 평균 depth가 트리 전체의 평균 depth `c(n)`과 비슷한 경우다. 평균적인 데이터 깊이를 가지고 있다고 보는데, 이 때 `s`는 0.5가 된다. 만약 (2)와 같이 데이터 x의 평균 depth가 0에 가깝다면, 즉, 위쪽에 위치한 경우라면, `s`는 1에 가까워 진다. 그리고 (3)의 경우와 같이 데이터 x의 평균 depth가 매우 크다면(*원래는 무한대로 표시해야 하지만, 일단 굉장히 깊다는 의미로 `n-1`을 사용했다. 그렇게 받아들이자.*) `s`는 0에 가까워 진다.

결과적으로 `s(Anomaly Score)`가 1에 가까울수록 비정상 데이터이고, 0에 가까울수록 정상 데이터로 간주한다. 특히, (1)과 같이 score가 0.5 이하이면 정상 데이터로 판단한다.

> - A score close to 1 indicates anomalies
> - Score much smaller than 0.5 indicates normal observations
> - If all scores are close to 0.5 then the entire sample does not seem to have clearly distinct anomalies
>
> _출처: https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

*참고*

교재 예시에서는 데이터 간 depth 차이가 크지 않아 anomaly score 차이가 크지 않았다.

## 실습 1. numpy 데이터

사이킷런 앙상블 `IsolationForest` 함수로 제공된다. score 계산을 알아보는 것이 목적이므로, 트리를 하나만 생성한다. 이상 데이터가 무엇인지 확인해 보자.

물론 트리를 만들기 때문에, 한 번 돌려서 강사님 코드와 같은 결과가 나온다고 보장할 수는 없다.

- `score_samples` : Anomaly Score 계산.
- `predict` : `1`이면 정상 데이터, `-1`이면 비정상 데이터로 간주.
- 강사님 코드 변형한 부분: 함수 만들고 `n_estimators` 여러 개 만들어 봄. 이상치로 판단되는 데이터는 똑같았다. 다만 Anomaly Score는 조금씩 달라진다.

```
# iForest model
def iForest(num_of_models, data):
    iForest = IsolationForest(n_estimators=num_of_models)
    iForest.fit(data)
    # prediction
    y_pred = iForest.predict(data)
    # anomaly score
    score = abs(iForest.score_samples(data))
    return y_pred, score

# 실행 결과
판정 결과: tree 1개일 때
```

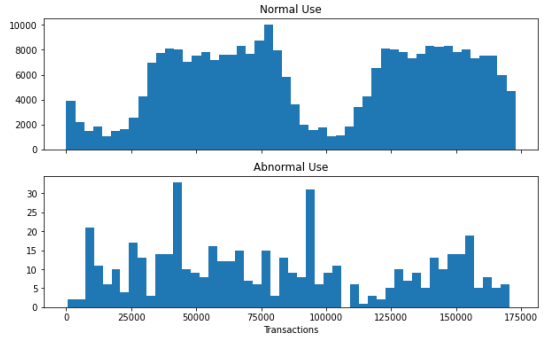
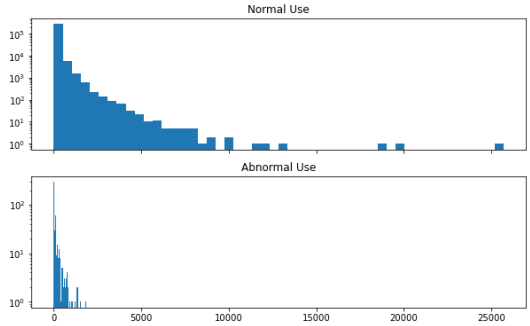
```
[ 1  1  1  1 -1 -1]
========== anomaly score ==========
[0.3404535  0.3404535  0.3404535  0.46381291 0.59918632 0.77407126]
판정 결과: tree 50개일 때
[ 1  1  1  1 -1 -1]
========== anomaly score ==========
[0.45344297 0.42072409 0.40340475 0.42117123 0.55713345 0.64703475]
```

## 실습 2. Credit Card Fraud

- Kaggle Data, 19개 feature, PCA에 의해 차원이 축소된 상태.
- class 0이 정상 데이터, 1이 비정상 데이터.
- 사용시간대, 사용금액 별 정상 데이터와 비정상 데이터.

  *[참고] 사용 금액은 yscale을 로그로 조정했다.*

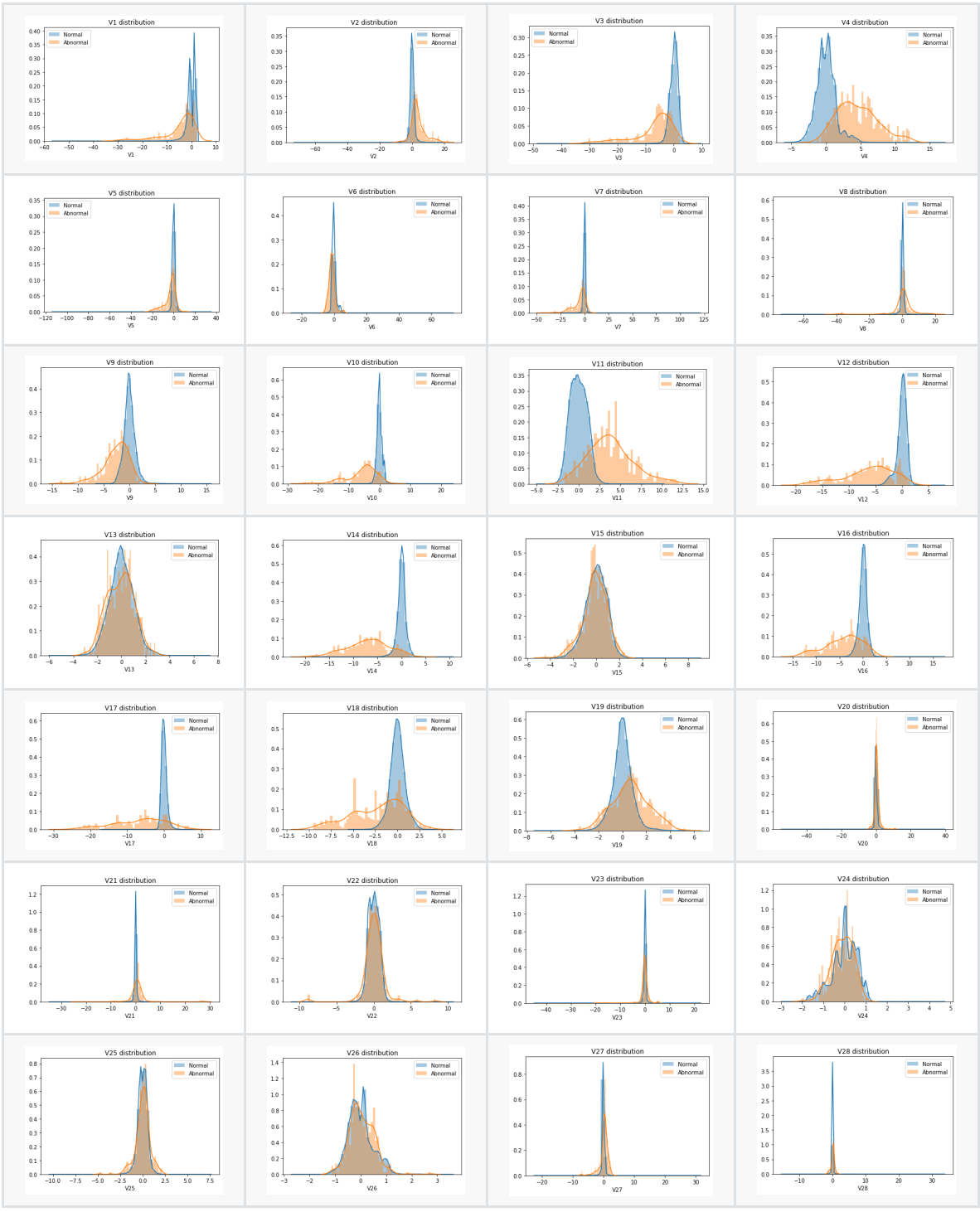
```
# 시간대별 사용금액 시각화
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10, 6))
ax1.hist(credit['Time'][credit['Class'] == 0], bins=50)
ax2.hist(credit['Time'][credit['Class'] == 1], bins=50)
plt.xlabel('Transactions')
ax1.set_title('Normal Use')
ax2.set_title('Abnormal Use')
plt.show()
```

| 사용 시간대 | 사용 금액 |
| --- | --- |
|  |  |



- 각 feature별 분포도

```
# feature 시각화
def visualize_Features(df, feature):
    sns.distplot(df[feature][df['Class'] == 0], bins=50)
    sns.distplot(df[feature][df['Class'] == 1], bins=50)
    plt.legend(['Normal', 'Abnormal'], loc='best')
    plt.title(feature + ' distribution')
    plt.show()
```




- 이상치로 판단하는 Anomaly Score 기준을 0.65로 설정했다. 다음의 코드로 이상치의 개수를 센다.

```
y_pred = (score > 0.65).astype(int) # 판정 기준
fraud_count = (y_pred == 1).sum() # 이상치 판단 개수
```

- 강사님 자료 코드 오류 수정

```
# 실제 정상 데이터를 비정상으로 잘못 판정한 비율
cm[0, 1] / cm[0, :].sum()

# 실제 비정상 데이터를 정상으로 잘못 판정한 비율
cm[1, 0] / cm[1, :].sum()
```

- `contamination` : outlier 비율을 알아서 사이킷런이 정해준다.

## 3. 알고리즘 평가

- 이상치 검출에 자주 활용된다.
- Recall과 Precision을 모두 잡을 수는 없다. 알고리즘 태생 상 뭐 하나가 높아지면 낮아질 수밖에 없다.
- 일반적으로 50~100개(사이킷런에서는 `n_estimators` 매개변수로 조절한다) 정도의 모델을 만들면 `Anomaly Score`가 안정화된다고 한다.