# 빅데이터 애널리틱스 리포트

신용 카드 사기 거래 식별

팀명: 1조

이름: 유동완 남국현 김민송 이시현

## 0. 요약

신용 카드 거래 내용 분석을 통해 사기 거래를 식별하는 모델을 만드는 것이 목표이다. 심각한 불균형 데이터 세트를 해결하기 위해 오버 샘플링, 언더 샘플링을 진행하였고, NN을 이용하여 모델링을 한 뒤, 각 모델 별 f1-score를 비교해보았다.

## 1. 서론

### 1.1. 문제 정의

신용카드의 거래 내용을 분석을 통해 사기 거래를 식별하는 모델을 만들어 사기 거래를 식별한다. 카드사가 사기성 카드 거래를 인식하여 고객이 구매하지 않은 품목에 대해 수수료를 받지 않게 하고, 정상 거래를 한 고객을 사기 거래로 식별하지 않도록 한다.

### 1.2. 모델 성능 결과

SMOTE 기법을 통해 오버 샘플링 하여 NN모델을 구축하여 test data에 대해 0.74의 f1-score를 구할 수 있었다. 14개의 정상 거래를 오 분류하였고 test data 98개의 사기 거래 중 72개의 사기 거래를 잡아낼 수 있었다.

## 2. 데이터 분석

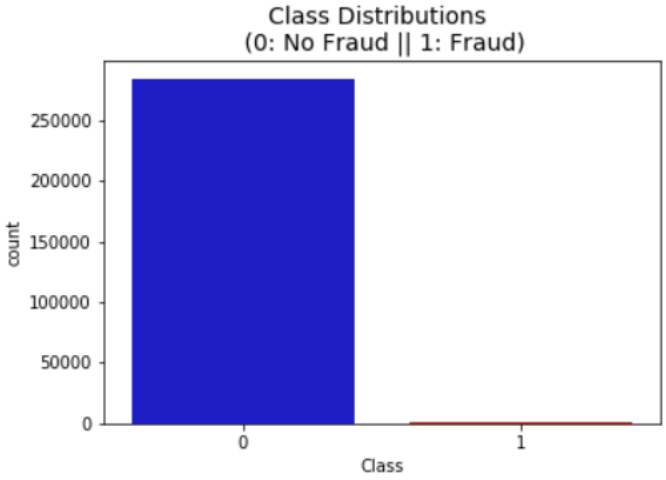
### 2.1. 데이터 설명

데이터 세트에는 신용카드를 보유한 유럽인들의 2013년 9월 거래 내용이 담겨있다. 이 데이터에는 2일간의 신용 거래 내용이 있으며 총 28만 4,807건의 거래가 있고 이 중 492건에서 사기 거래가 발생했다. 데이터는 개인정보 보호를 위하여 실제 거래 정보들은 '시간'과 '금액'을 제외한 다른 feature들은 PCA

변환된 결과인 숫자들만 기재되어 있다. PCA 변환된 V1 ~ V28과 '시간', '금액', '클래스'를 합쳐 총 31개의 feature가 있다. 시간은 각 거래 사이의 지난 시간을 금액은 거래된 금액을 의미하며 클래스는 사기 거래의 클래스 값을 1, 그렇지 않을 땐 0의 값을 가진다.

앞서 말했듯이 총 28만 4,807건의 거래 중 492건의 거래만이 사기 거래를 한 데이터이다. 즉, 데이터 세트가 매우 심각한 불균형 데이터 세트임을 알 수 있다.

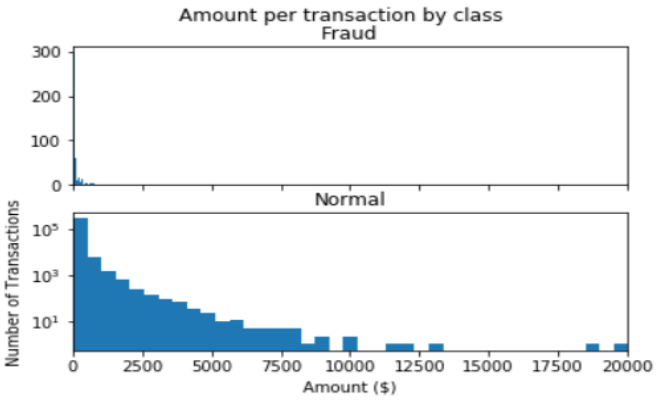
a. Class Distributions




<그림 1>

<그림 1>에서도 확인할 수 있듯이 매우 불균형한 데이터로 정상 거래의 비율은 99.83%, 사기 거래의 비율은 0.17%이다. 만약 극심한 불균형 데이터 세트를 그대로 모델에 적용할 경우 에러가 많이 나타날 것이다. 또한, 알고리즘이 대부분의 거래를 정상 거래로 추정할 것이기 때문에 train data에 대한 Overfitting이 발생할 확률이 높을 것이다. '신용카드 거래 내용' 데이터를 연구하는 목적이 사기 거래를 식별하는 것이기 때문에 정상 거래를 잘 예측하는 것 또한 중요하지만 사기 거래의 패턴을 파악하고 잘 예측하는 모델을 만들어야 할 것이다.

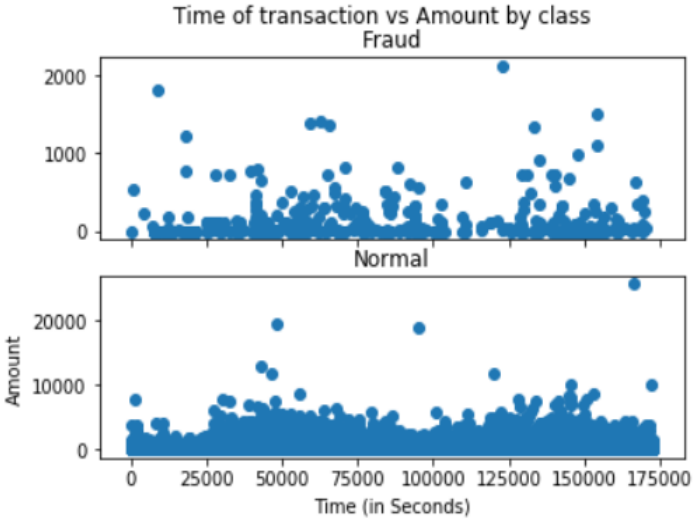
b. Transactions amount




<그림 2>

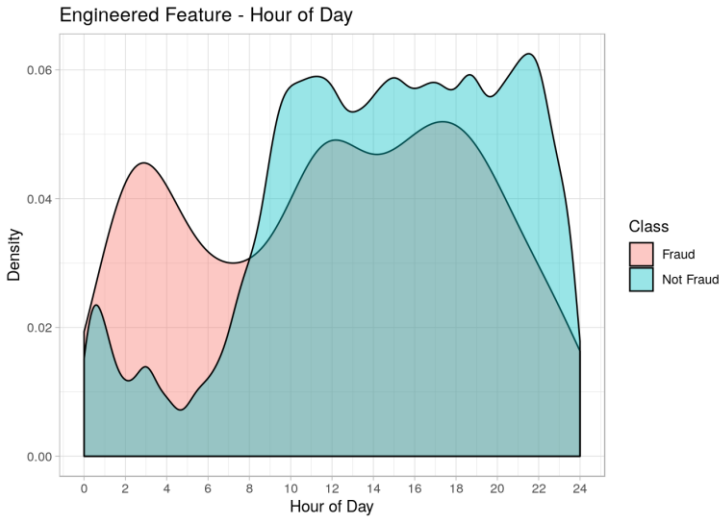
<그림 2>는 정상 거래와 사기 거래 수 별 거래 금액을 표현한 그래프이다. 사기 거래의 금액이 많지 않은 것을 확인할 수 있다.

c. Transactions in time




<그림 3>

<그림 3>은 정상 거래와 사기 거래의 시간에 따른 거래 금액을 보여주는 그래프이다. 어떤 특정 시간에 사기 거래가 더 자주 일어나는지는 <그림 3>의 그래프만 보고는 알기가 어렵다.




<그림 4>

<그림 4>와 같이 단위를 초에서 시간으로 바꾸었더니 새벽부터 이른 아침 시간대에 사기 거래가 정상 거래보다 급격히 많아지는 것을 확인할 수 있다.

## 2.2. 분석 및 시각화

### 2.2.1. 전처리

a. Scaling and Splitting the Data

이 단계에서는 Time과 Amount feature를 살펴보려고 한다. 이 두 개의 feature는 다른 feature들과 단위가 맞지 않기 때문에 스케일링을 먼저 진행해야 한다. 또한, 정상 거래, 사기 거래가 불균형이 심각한 데이터이기 때문에 두 사례의 비율을 비슷하게 맞추기 위해 데이터 세트의 서브 데이터 세트를 만들어야 한다. 여기에서 서브 데이터 세트란, 다수 사례와 소수 사례의 비율을 1대 1로 맞춘 데이터 세트를 뜻한다. 서브 데이터 세트를 만드는 이유는 이전에 언급한 바와 같이, 데이터가 굉장히 불균형 함을 확인했고 이는 아래와 같은 문제를 발생시키기 때문이다.

■ Overfitting: 불균형한 데이터이기 때문에 모든 새로운 데이터에 대해 거래가 정상이라고 할 것이다. Training Data에 사기 거래 사례가 많이 없을

것이기 때문에 분류기가 이에 대해 학습을 잘 못 할 것이기 때문이다. 하지만 우리는 극소수의 사기 거래를 감지해내야 하므로 이는 큰 문제이다.

■ Wrong Correlations: PCA 처리된 V라는 feature들이 무엇을 의미하는지는 잘 모르지만, 이 feature들이 각 feature와 어떤 상관관계인지, 판정에 어떤 영향을 미칠지 이해한다면 굉장히 유용할 것이다. 그러나 상관계수 분석을 불균형 데이터로 진행한다면 정상과 사기 거래에 대한 진실한 feature들의 상관관계를 파악하지 못할 것이다.

위와 같은 이유로, 우선 스케일링을 진행하였다. 아래 <그림 5>에서 보는 것과 같이 Time feature와 Amount feature를 스케일링하여 스케일링 된 feature들을 데이터 프레임에 넣어 기존의 Time feature와 Amount feature를 대체하였다. 스케일링 방법은 RobustScaler를 이용했다.

※ RobustScaler: 모든 특성들이 같은 크기를 갖는다는 점에서 StandardScale와 비슷하지만, 평균과 분산 대신 median과 quartile을 사용한다. Outlier에 강건하다.

| | scaled_amount | scaled_time | V1 | V2 | V3 | V4 | V5 |
|---|---|---|---|---|---|---|---|
| 0 | 1.783274 | -0.994983 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 |
| 1 | -0.269825 | -0.994983 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 |
| 2 | 4.983721 | -0.994972 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 |
| 3 | 1.418291 | -0.994972 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 |
| 4 | 0.670579 | -0.994960 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 |

<그림 5>

b. Splitting the Data

언더 샘플링과 오버 샘플링 모두 진행하기 전에 원 데이터를 분리해야 한다. 그렇지 않으면 test data까지 랜덤 샘플링의 영향에 들어갈 것이고, 모델을 검증해야 할 데이터 샘플의 신뢰도가 떨어지기 때문이다.

2.2.2. Random Undersampling

a. Random Undersampling

원 데이터 세트에는 총 28만 4,807건의 거래 데이터 중 492개의 사기 거래가 있다. 따라서 무작위로 정상 거래 사례 492개를 뽑아내어 두 개의 데이터 세트를 합하여 새로운 서브 데이터프레임을 만든다. 이러한 기법을 언더 샘플링이라고 한다. 언더 샘플링이란 기본적으로 보다 균형 잡힌 데이터 세트를 얻기 위해 다수 사례의 데이터들을 제거하고 모델이 Overfitting되지 않도록 하는 기법이다. 샘플링 한 뒤 서브 데이터 세트의 클래스 빈도수를 확인해 보면 <그림 6>과 같이 두 클래스 빈도수가 같음을 확인할 수 있다.


<그림 6>

■ 절차

① 클래스의 불균형 확인

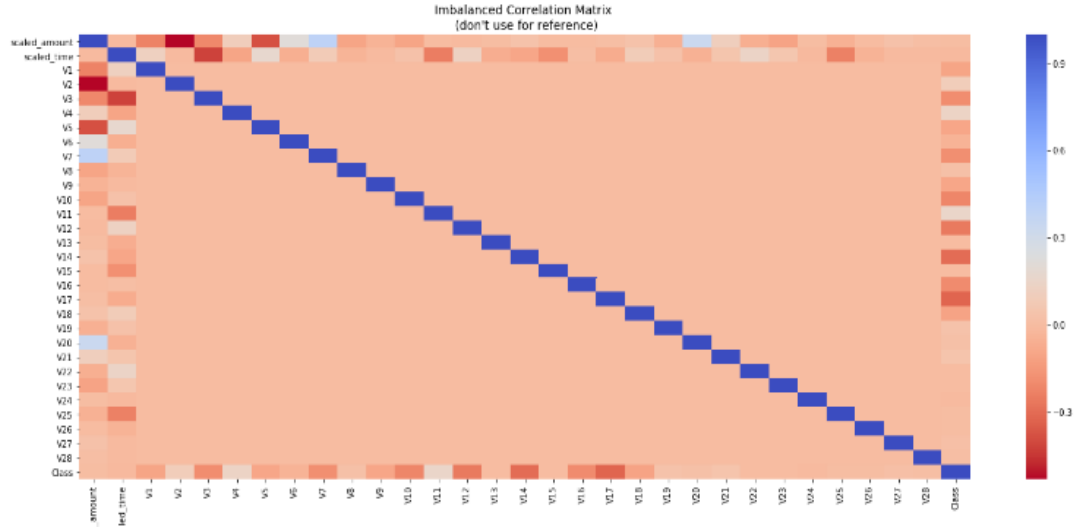
② 소수 사례의 수만큼 데이터 세트에서 다수 사례를 뽑아 균형 1:1로 맞추기

③ 마지막으로 데이터를 계속 무작위로 섞어주어 스크립트를 실행할 때마다 정확도를 유지하는지 확인해 주어야 한다.

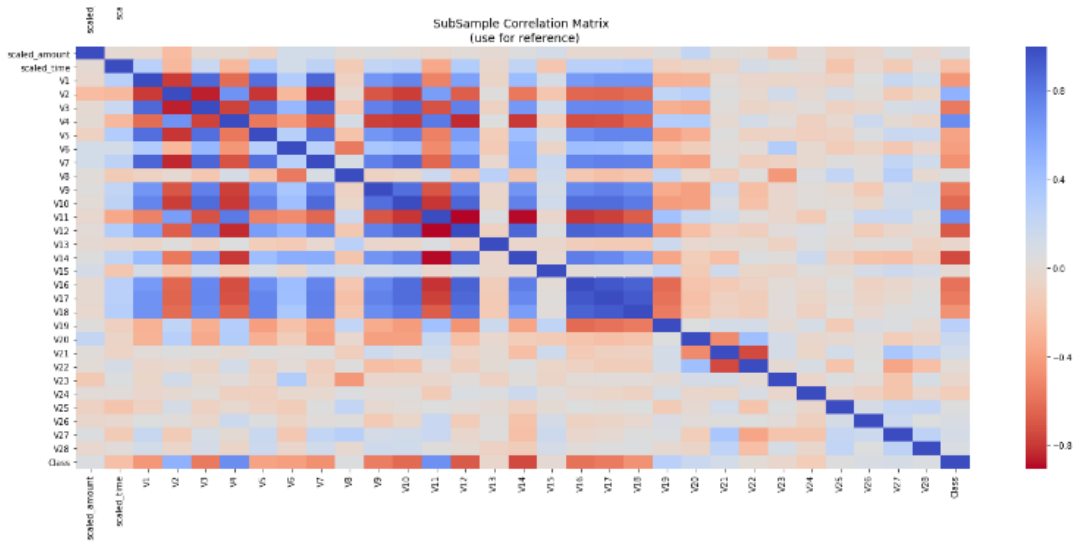
■ 주의점: 랜덤 언더 샘플링 기법은 기본적으로 기존의 데이터를 삭제하는 방법이기 때문에 분류 모델의 학습에서 기존 데이터의 특성을 정확하게, 충분하게 반영하지 못한다는 단점이 있다. 284,315개의 정상 거래 데이터 중, 492개의 정상 거래 데이터만 가져온다. 즉, 언더 샘플링을 실행할 시, 정상 거래 데이터 283,331개를 제거해야 함을 의미한다.

b. Distributing and Correlating

상관계수의 분석은 데이터를 이해하는 데 있어 핵심적인 역할을 한다. 특정한 거래가 정상인지, 사기인지 판단하는 데 큰 영향을 미치는 feature가 어떤 것인지 확인하려면 필수적인 과정이다. 이를 위해 언더 샘플링을 진행한 서브 데이터 세트로 상관계수를 분석하였다. 서브 데이터 세트로 상관관계를 분석한 이유는 <그림 7>과 <그림 8>을 비교해보면 알 수 있듯이 원 데이터 세트는 불균형 데이터 세트이기에 올바른 상관관계를 파악하기 어렵기 때문이다.



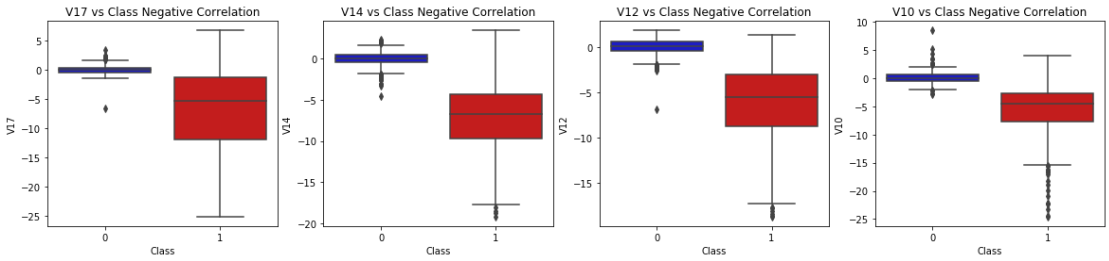

<그림 7>



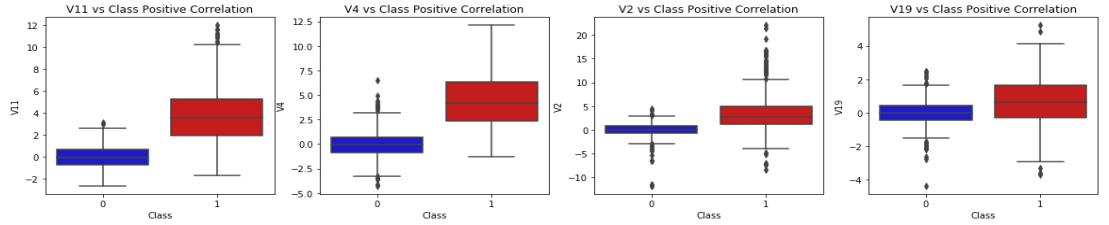

<그림 8>

상관계수 분석 결과를 참고하여 높은 관계를 가진 feature들을 클래스별로 박스 플롯을 만들어 보았다. <그림 9>, <그림 10>에서 보면 명확하게 알 수 있다.

V17, V14, V12, V10이 작을수록, V11, V4, V2, V10은 클수록 거래가 사기일 확률이 높다는 사실을 파악할 수 있었다.




<그림 9>




<그림 10>

2.2.3. Outlier 탐지

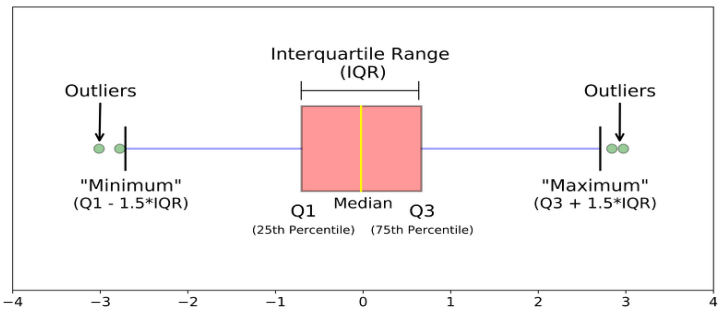
Outlier를 제거할 때, ESD, IQR 등의 방법으로 Outlier를 판별하더라도 전부 지워서는 안 된다. 이 데이터 세트는 PCA된 결과이므로 feature의 의미를 정확히 파악하기 어려워서 Outlier로 판별이 되었어도 정상 데이터일 가능성도 존재하기 때문이다. 그러나 아무것도 지우지 않는다면 모델의 정확도에 악영향을 미칠 것이므로 Outlier를 적당한 수준으로 지워야 할 필요가 있다. 너무 많은 Outlier를 지우게 되면 정보 손실이 커지므로 오히려 악영향을 미치게 된다. 이러한 점을 인지한 상태에서 Outlier 제거를 실시하였다.

Outlier제거는 종속feature인 'Class' feature와 상관관계가 높은 feature들을 선정하고(모든 feature에 대하여 할 경우 너무 많은 정보 손실이 발생), 사기 거래로 감지된 샘플에 한해 Outlier 제거를 진행하였다. 사기 거래가 정상 거래보다 분포가 지나치게 넓었기 때문이다. 양의 상관관계가 높은 feature 중 상위 4개의 feature를 선정하여 Outlier 제거를 한 경우, 음의 상관관계가 높은 상위 4개 feature를 선정하여 Outlier제거를 한 경우, 높은 양의 상관관계와

높은 음의 상관관계를 갖는 8개 feature의 Outlier를 제거한 경우 세 가지로 나누어 Outlier를 제거했을 때 제거되는 데이터의 수를 계산하였다.

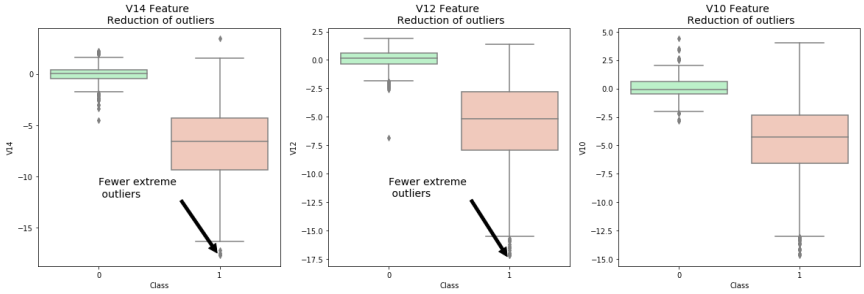
양의 상관관계가 높은 상위 네 개의 feature의 Outlier를 제거한 경우 61개의 샘플이 지워졌으며, 음의 상관관계가 높은 상위 네 개의 feature의 Outlier를 제거한 경우 35개의 샘플이 지워졌다. 높은 양의 상관관계와 높은 음의 상관관계를 갖는 모든 feature의 Outlier를 제거한 경우 80개 이상의 행이 제거되었다. 너무 많은 Outlier를 지우게 되면 정보 손실이 커지기 때문에 이 중 정보의 손실을 가장 줄일 수 있는 음의 상관관계가 높은 상위 네 개의 feature에 대한 Outlier를 제거하였다.

Outlier의 선정 기준은 IQR Rule로 하였으며 Box plot과 함께 설명해보려 한다. 1분위수와 3분위수 사이의 거리를 IQR로 지정한 뒤, 1분위수 - 1.5*IQR 보다 작은 값과 3분위수 + 1.5*IQR 보다 큰 값을 Outlier로 지정하는 것이다. <그림 11>을 보면 더욱 직관적으로 확인할 수 있다.




<그림 11>

Box plot을 이용해 Outlier를 탐지하여 상한선과 하한선 이상의 값을 극단 Outlier로 판단하였다. 극단 Outlier에 해당되는 값이 있는 행은 제거하였다. Outlier 값들 전부를 대상으로 하는 것이 아닌 극단 Outlier를 적당히 제거하여 정보가 손실되면서 생기는 위험을 낮추기 위해 노력했다. 아래의 <그림 12>에서 확인할 수 있는 것과 같이 Box plot을 사용해 극단 Outlier의 감소를 확인한다.




<그림 12>

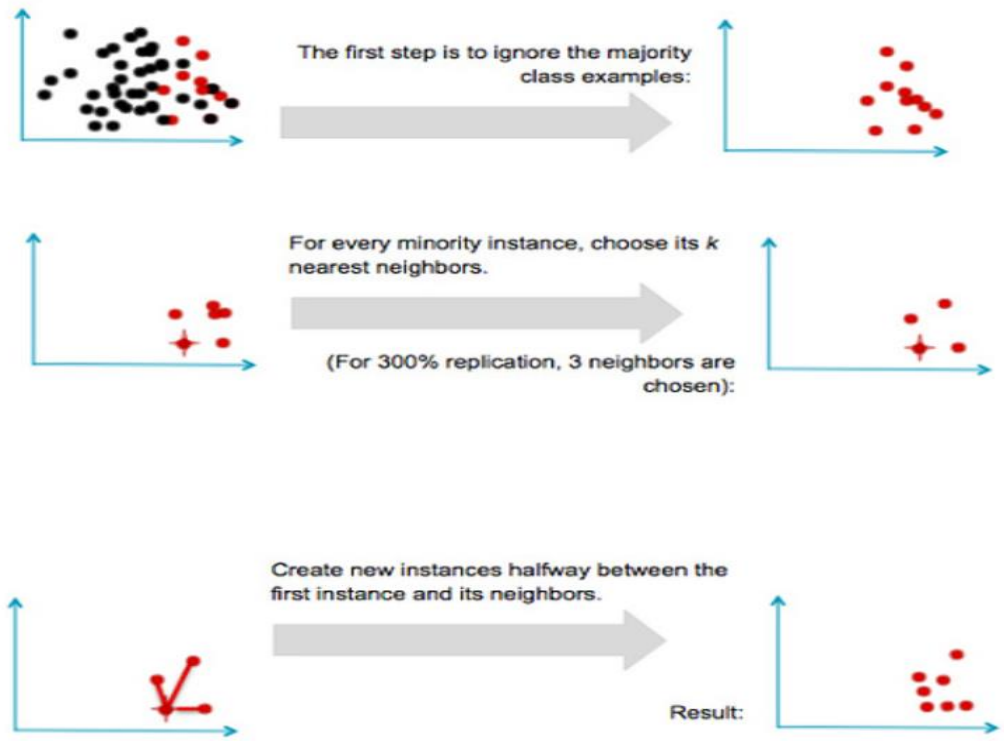
2.2.4. OverSampling

이 프로젝트에서는 클래스 불균형 문제를 해결하기 위해 언더 샘플링, 오버 샘플링 두 가지 경우를 모두 고려해주었다. 두 가지 방법을 통해 데이터 편향성을 해결하고, 각각의 모델 성능을 도출해 이 둘의 경우를 비교해보았다.

오버 샘플링은 소수사례의 샘플을 늘리는 것이다. 정보를 잃지 않지만, training data에게서는 높은 성능을 보이고 test data에서는 성능이 낮아질 수 있다. 이처럼 대부분의 오버 샘플링 방법은 Overfitting의 문제를 포함하고 있다. 이를 피하고자 SMOTE(Synthetic Minority Over-sampling Technique)를 사용한다. SMOTE를 통해 생성된 각 샘플은 주변 데이터를 고려하려 데이터를 생성하기 때문에 Overfitting의 가능성이 낮아지게 된다.

■ SMOTE Technique (Over-Sampling)

SMOTE란, 합성 소수 샘플링 기술로 기존 소수 데이터를 단순 복제하는 것이 아니라 새로운 복제본을 만들어 내는 것을 의미한다. 아래 <그림 13>과 같이 먼저 분류 개수가 적은 쪽의 데이터의 샘플을 취한 뒤 이 샘플의 k 최근접 이웃을 찾는다. 그리고 현재 샘플과 이들 k개 이웃 간의 차를 구하고, 이 차이에 0 ~ 1 사이의 임의의 값을 곱하여 원래 샘플에 더한다. 이렇게 만든 새로운 샘플을 training data에 추가한다. 결과적으로 SMOTE는 기존의 샘플을 주변의 이웃을 고려해 약간씩 이동시킨 점들을 추가하는 방식으로 동작한다.




<그림 13>

이를 통해, 소수 데이터의 개수를 다수 데이터만큼 늘려, 클래스 불균형을 해결하였다. 언더 샘플링 방식과는 달리 행을 삭제할 필요가 없어 많은 정보가 보존된다는 장점이 있다. 언더 샘플링보다 오버 샘플링(SMOTE)의 정확도가 높을 가능성이 크지만, 훈련 시간이 언더 샘플링보다는 더 걸린다는 단점이 있다.

2.3. 소결론

신용 카드 거래 데이터 세트는 총 28만 4,807건의 거래 중 492건의 거래만이 사기 거래 데이터이다. 즉, 데이터 세트가 매우 심각한 불균형 데이터 세트임을 알 수 있다. 데이터 세트의 30개의 feature 중 스케일링이 되지 않은 두 개의 feature, 'Time'과 'Amount'에 대해서 Robust 스케일링을 해주었다. 클래스 불균형을 없애기 위해 언더 샘플링과 오버 샘플링을 해주었다.

먼저 언더 샘플링을 진행하였고, 종속 feature와 독립 feature 간의 상관관계를 분석해 Outlier 제거를 진행하였다.

V17, V14, V12 그리고 V10 feature들은 종속feature와 음의 상관관계를 띄고 있다. 즉, 이 feature들의 값이 낮을수록, 거래가 사기일 확률이 높아짐을 의미한다. V2, V4, V11 그리고 V19 feature는 종속feature와 양의 상관관계를 띄고 있다. 즉, 이 feature들의 값이 높을수록, 거래가 사기일 확률이 높아짐을 의미한다.

음의 상관관계를 갖고 있는 feature들의 Outlier를 제거하는 것이 양의 상관관계를 띄고 있는 feature들을 통해 Outlier를 제거하는 것보다 정보 손실이 적기 때문에 음의 상관관계를 갖는 feature들을 대상으로 Outlier를 제거하였다. Outlier 제거 후 <그림12>를 통해 극단 Outlier의 수가 상당히 감소했음을 알 수 있었다.

다음으로는 오버 샘플링을 진행했다. 오버 샘플링 기법은 SMOTE 기법을 이용하였다.

## 3. 모델

### 3.1. Input과 Output

a. Input

카드 사기 거래 예측 데이터는 고객 거래 정보가 담긴 PCA 처리한 28개의 feature와 'time', 'amount'를 독립feature로 가지고 있다. 이들을 Input으로 사용했다.

b. Output

카드 사기 거래 예측 모델은 이 거래가 사기인지 아닌지를 예측하는 모델이다. 독립feature로는 사기일 경우는 1, 아닐 경우는 0의 값을 가지고 있는 'Class' feature를 독립feature이자 Output 값으로 사용했다.

### 3.2. 사용한 모델

training data와 test data의 분리는 0과 1이 같은 비율로 나뉠 수 있도록 계층적 분할을 이용했고 test data의 크기는 0.2로 했다. 카드 사기 거래 데이터로 사기인지 아닌지를 판별하는 문제이기 때문에 지도학습 중 분류

모델을 선택했다. 사용한 모델로는 Decision tree, Support Vector, KNN, Logistic Regression, 신경망이 있다.

그 중 오버 샘플링과 언더 샘플링의 성능을 비교하는 데 쓰였던 인공신경망에 대해서 자세히 설명해 보겠다. 인공신경망은 두뇌의 신경세포, 즉 뉴런이 연결된 형태를 수학적으로 모방한 모델이다. 신경망은 입력층과 은닉층 그리고 출력층으로 이루어진다. 먼저 입력층은 데이터를 입력받는 층이고, 은닉층은 입력받은 데이터를 가중치와 편향 등을 통해 변환하고, 마지막으로 출력층은 은닉층에서 전달받은 값을 출력한다.

신경망은 설정해야 할 파라미터가 많은데 먼저 은닉층의 개수부터 정했다. 은닉층은 1개로도 충분히 복잡한 데이터도 설명할 수 있다고 판단하여 1개로 설정하고 진행했다. 다음으로 노드의 개수를 설정해야 했는데 출력층은 class가 2개이므로 2개로 설정했다. 가장 중요한 은닉층의 노드의 수는 처음에 의도적으로 많이 만들었는데, 200개부터 시작해서 overfitting 시킨 후 줄여나간 결과 32개의 노드를 갖게 되었다. 각 층의 활성화 함수는 가장 준수한 성능을 보이는 함수로 설정했는데, 그 결과 출력층은 soft max 나머지는 relu로 설정하여 신경망의 다음과 같은 구조를 구성했다.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_1 (Dense) | (None, 30) | 930 |
| dense_2 (Dense) | (None, 32) | 992 |
| dense_3 (Dense) | (None, 2) | 66 |

Total params: 1,988
Trainable params: 1,988
Non-trainable params: 0

<그림 16>

## 4. 실험결과

### 4.1. 성능 지표와 Hyperparameter

f1-score 사용했다. f1-score는 Recall과 Precision의 조화평균 값이다. 불균형 데이터 세트에 대한 평가지표로 주로 사용된다. Recall과 Precision의 가중치를 같다고 가정하고 산출하는 값이다. 사기 거래를 최대한 많이 감지해 내는 것도 중요하지만, 정상 거래를 사기 거래라고 오분류 하는 사례도 줄여야 하기 때문이다.

### 4.2. 결과값

a. 언더 샘플링

먼저, 언더 샘플링된 데이터에 4가지 모델(Decision tree, Support Vector, KNN, Logistic Regression)을 적용했다. 별도의 하이퍼 파라미터 설정 없이 교차 검증의 정확도를 이용하여 성능을 확인했다. 그 결과 Logistic Regression과 KNN이 대략 93%로 가장 좋은 성능을 보였다(Decision tree - 89%, Support Vector - 92%).

다음으로 하이퍼 파라미터를 튜닝 후 성능을 확인하기 위해 다음과 같이 설정 후 그리드 서치를 진행하였다.

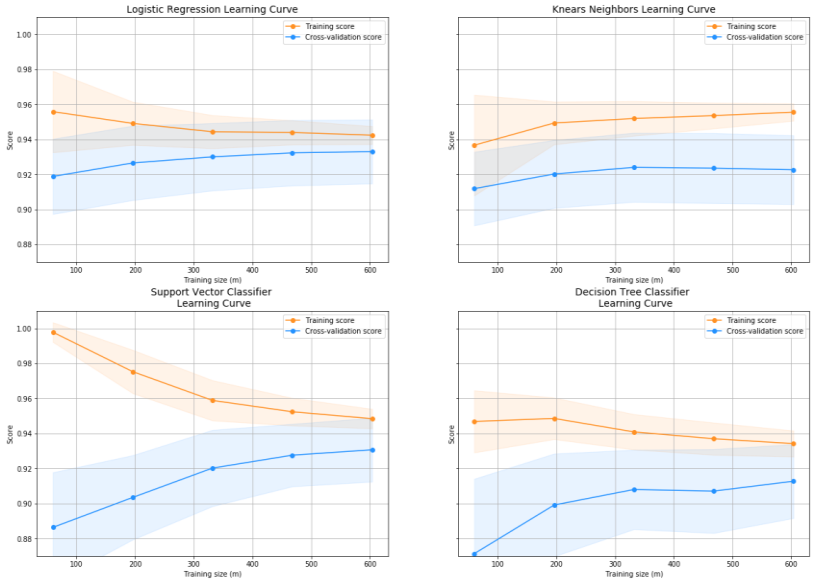
- Logistic Regression: penalty - ['l1', 'l2'], C - [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- KNN: n_neighbors - list(range(2,5,1)), 'algorithm' - ['auto', 'ball_tree', 'kd_tree', 'brute']
- Support Vector: C - [0.5, 0.7, 0.9, 1], kernel - ['rbf', 'poly', 'sigmoid', 'linear']
- Decision tree: criterion - ["gini", "entropy"], max_depth - list(range(2,4,1)), min_samples_leaf - list(range(5,7,1))

이에 대한 최적의 하이퍼 파라미터는 다음과 같다.

```
print("Logistic best hyperparameter ==> {}".format(grid_log_reg.best_params_))
print("Knears best hyperparameter ==> {}".format(grid_knears.best_params_))
print("Support Vector best hyperparameter ==> {}".format(grid_svc.best_params_))
print("DecisionTree best hyperparameter ==> {}".format(grid_tree.best_params_))
```

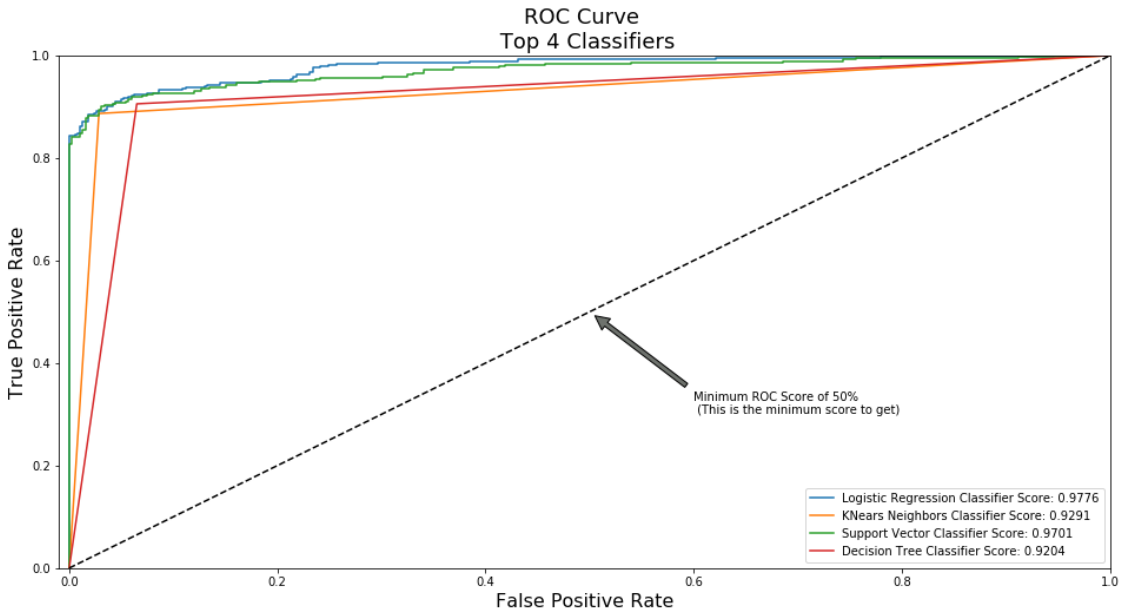
```
Logistic best hyperparameter ==> {'C': 0.1, 'penalty': 'l2'}
Knears best hyperparameter ==> {'algorithm': 'auto', 'n_neighbors': 2}
Support Vector best hyperparameter ==> {'C': 0.5, 'kernel': 'linear'}
DecisionTree best hyperparameter ==> {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 5}
```

최적의 하이퍼 파라미터를 적용한 모델을 만들고, 교차 검증을 하면 성능은 Logistic Regression와 Support Vector가 약 95%로 가장 좋은 성능을 보인다. 학습 곡선을 확인하면 다음과 같다. Training Score와 CV Score의 사이가 넓을수록 overfitting의 가능성이 크다. 또한, 두 성능지표의 수치가 낮을수록 underfitting의 가능성이 크다. <그림 14>을 보면 Logistic Regression가 가장 우수한 성능을 나타내고 있음을 알 수 있다.




<그림 14. 4개 모델에 대한 학습곡선>

<그림 15>에 4가지 모델에 대한 ROC 곡선도 확인해보면 다음과 같다.




<그림 15>

4가지 모델에 대한 ROC 커브를 확인해보면, 여기서도 Logistic Regression(AUC=0.977)가 가장 좋은 성능을 보인다.

다음으로 가장 좋은 성능을 보여주었던 Logistic Regression 모델을 이용하여 언더 샘플링된 데이터로 훈련된 모델로 원본 데이터의 test data를 예측해보았다. <표 1>에 다음과 같은 정확도(accuracy), 정밀도(precision), 재현율(recall), f1-score를 확인할 수 있다.

<표 1.>

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.97 | 0.98 | 56863 |
| 1 | 0.05 | 0.90 | 0.09 | 98 |
| accuracy | | | 0.97 | 56961 |
| macro avg | 0.52 | 0.93 | 0.54 | 56961 |
| weighted avg | 1.00 | 0.97 | 0.98 | 56961 |

<표 1>을 참고해보면 recall 성능과 0에 대한 precision은 매우 우수하다. 하지만 1에 대한 precision이 매우 낮은데, 이는 사기 거래가 아닌 정상 거래를 사기 거래로 많이 분류했기 때문이다. 실제로 이에 대한 혼동행렬을 참고해보면, 총 1,791개를 사기 거래로 예측하였고 이 중에서 88개만 실제 사기 거래였다. 이를 해결하기 위해 recall을 좀 낮추는 방법도 있지만 0.05라는 수치는 그렇게 조정하기에 심각하게 낮은 수치라고 판단하고 오버 샘플링을 하는 방법을 시도해보았다.

b. 오버 샘플링

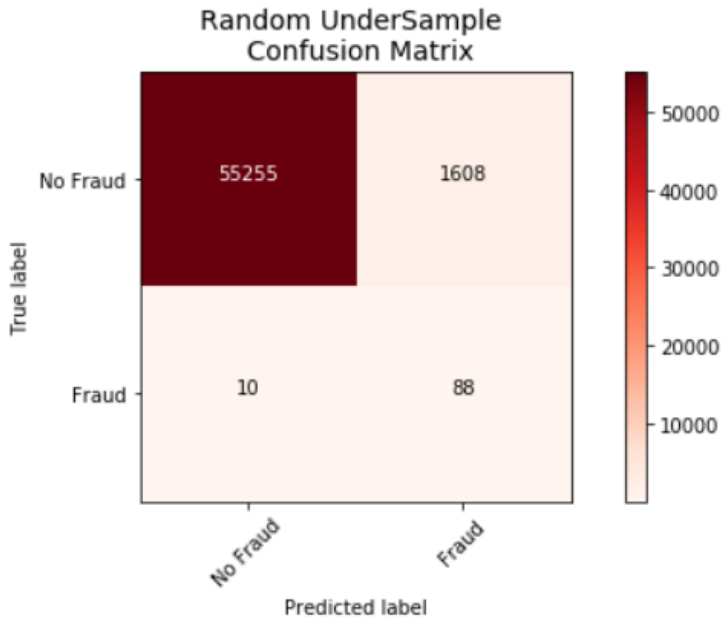
오버 샘플링도 마찬가지로 가장 좋은 성능을 보인 Logistic Regression 모델을 사용하였다. 원본 데이터의 X_train, y_train만 오버 샘플링하여 만든 데이터로 훈련된 모델로 원본 데이터의 test data를 예측했다. 아래의 <표 2>와 같이 정확도(accuracy), 정밀도(precision), 재현율(recall), f1-score를 확인할 수 있었다.

<표 2>

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 0.99 | 56863 |
| 1 | 0.10 | 0.86 | 0.17 | 98 |
| accuracy | | | 0.99 | 56961 |
| macro avg | 0.52 | 0.92 | 0.58 | 56961 |
| weighted avg | 1.00 | 0.99 | 0.99 | 56961 |

<표 2>를 참고해보면 기존의 언더 샘플링으로 나온 1에 대한 precision 결과(0.05)보다 2배가량 올라간 0.1이 나왔다. 하지만 여전히 1에 대한 precision이 낮다. 마찬가지로 recall의 성능을 낮춰 precision을 올릴 수 있으나 그렇게 하기에는 0.1이라는 수치도 낮다고 판단했다. 따라서 다른 방법이 필요했고, 기존 모델이 아닌 신경망 모델을 사용했다.

먼저 언더 샘플링된 데이터로 <그림 16>의 구조로 신경망 모델을 훈련하였다. 훈련과정의 평가 방식은 accuracy로 반복 횟수는 20회로 하여 진행하였다. 그 결과는 <그림 17>와 <표 3>이다.



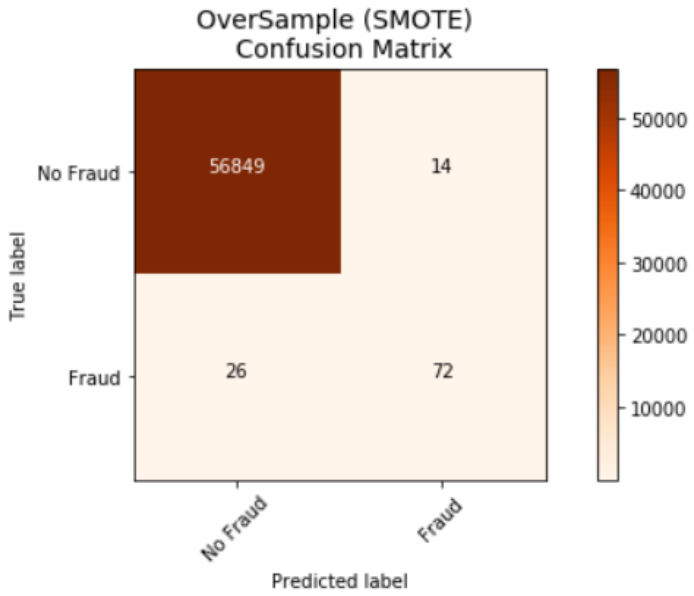

<그림 17>

먼저 혼동행렬을 확인해보면 여전히 정상 거래를 사기 거래로 예측하는 수치가 높다. 좀 더 직관적으로 와 닿는 성능을 확인하기 위해 정확도(accuracy), 정밀도(precision), 재현율(recall), f1-score를 확인하면 <표 3>과 같다.

<표 3>

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.97 | 0.99 | 56863 |
| 1 | 0.05 | 0.86 | 0.10 | 98 |
| accuracy | | | 0.97 | 56961 |
| macro avg | 0.53 | 0.93 | 0.54 | 56961 |
| weighted avg | 1.00 | 0.97 | 0.98 | 56961 |

언더 샘플링의 경우 기존의 Logistic Regression 모델의 성능과 크게 달라진 것이 없다. 다음으로 오버 샘플링 데이터로 신경망 모델을 만들어 확인해보았다. 오버 샘플링은 언더 샘플링의 구조와 비슷하게 만들어 진행했다. 그 결과는 <그림 18>와 <표 4>이다.




<그림 18>

정상 거래를 사기 거래로 많이 예측하는 문제가 해결되었다. 그만큼 사기 거래를 정상 거래라고 잘못 예측하는 비율도 늘어나 1에 대한 recall이 떨어졌지만, 기존의 모델들보다 훨씬 더 안정적인 성능을 보인다. 정확도(accuracy), 정밀도(precision), 재현율(recall), f1-score를 확인해보면 <표 4>와 같다.

<표 4>

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 0.99 | 56863 |
| 1 | 0.84 | 0.73 | 0.78 | 98 |
| accuracy | | | 1.00 | 56961 |
| macro avg | 0.53 | 0.93 | 0.89 | 56961 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56961 |

혼동행렬에서 나타난 것처럼 1에 대한 f1-score가 매우 많이 올라 가장 좋은 성능을 보여주었다.

5. 결과

모델의 예측력을 높이기 위해 언더 샘플링과 오버 샘플링을 진행한 뒤 NN을 적용한 뒤 비교해보았다. 언더 샘플링은 모델을 세우기 이전에 Outlier 처리를 해주어서 성능 향상을 할 수 있었다. 그에 반해 오버 샘플링의 SMOTE 기법은 기존 데이터를 기반으로 가상 데이터가 만들어지기 때문에 Outlier 제거가 어렵고, 한다면 원본 데이터에서 Outlier 제거가 이뤄져야 하는데 두 클래스의 비율의 차이가 극심해 Outlier의 제거가 효과적으로 이루어지기 힘들 것으로 판단하여 Outlier의 제거는 생략하였다. 결과적으로 언더 샘플링을 통해 구축한 모델은 0.1의 f1-score, 오버 샘플링을 통해 구축한 모델은 0.78의 f1-score를 구할 수 있었다. NN 모델의 특성상 어떤 feature가 중요한지, 가중치의 선정은 어떻게 진행되었는지 직관적으로 알기는 힘들다. 모든 test data에 대해 정상 거래라고 판정하였으면 총 98개의 사기 거래를 놓쳤을 것이다. 그러나 오버 샘플링을 통해 구축한 NN 모델을 적용하여 14개의 정상 거래를 오분류 한 대신, 72개의 사기 거래를 잡아낼 수 있었다.

출처

<그림 11>: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

<그림 13>: https://mkjjo.github.io/python/2019/01/04/smote_duplicate.html