Credit Card Fraud Detection

Member	㉧ 김보성
Notebook Link	https://www.kaggle.com/shivamb/semi-supervised-classification-using-autoencoders
Notebook Name	Semi Supervised Classification using AutoEncoders
Topic	Other datasets
Week	Week 4️⃣
추천	⭐⭐⭐⭐

대회 개요

'익명화' 된 카드 송금 내역 만으로, 사기/ 일반 송금을 구분해내는 대회.

이전 보험사 문제와 같이, 사기였다/ 아니다를 제외한 대부분의 정보가 식별이 불가능하다.

다만, 시간과 금액만이 보존되어있다.

태블러 데이터셋을 다루는 것 또한 배울만해보이는데, 대다수의 노트북이 태블러 데이터셋에 뉴럴넷을 안쓰는 와중에 오토인코더를 쓰는 노트북이라는 말에 뒤도 안보고 클릭했다.

문제점

역시 대다수의 현실 데이터셋이 그렇듯, 클래스 불균형 문제가 심각하다.

문제 없는 송금이 99.83퍼센트,

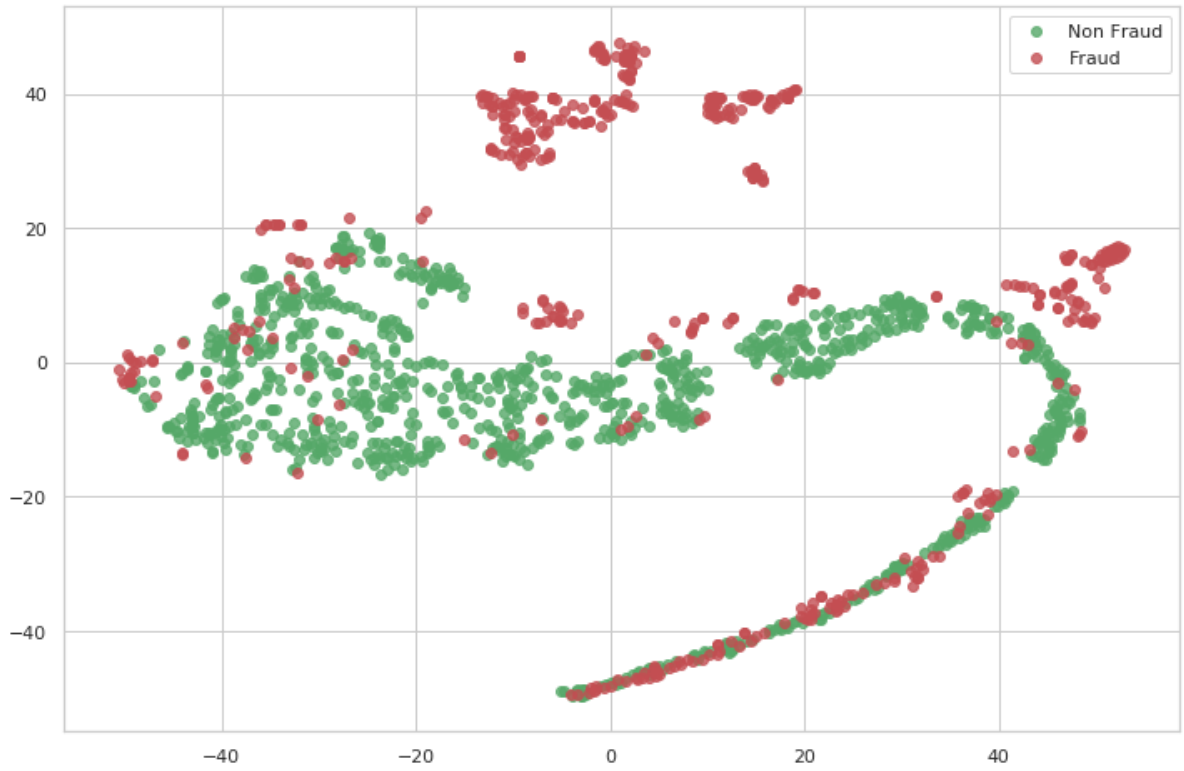
사기 송금이 0.17퍼센트이다.

이걸 어떻게 풀었던 걸까?

EDA

극심한 불균형으로 인해, 그냥 시각화 했다가는 아무런 인사이트를 얻을 수 없다.

일반 송금 사례 중 1000개만 사용하기로 하고, 시각화하였다.



이 플롯에서 알 수 있는 것은, 상당수의 사기 사례가 일반 송금 사례와 그 피쳐가 비슷하며, 이로서 모델링 하기가 그리 쉽지 않을 것이란 것을 예상할 수 있다. 아마 이런 이유로 해결책으로서 뉴럴넷을 찾아보신게 아닐까 하는 생각이다. 이를 해결하기 위해 필자는 오토인코더를 도입한다.

솔직히 다른 오토인코더를 가져오려나 했는데 정말 바닐라 오토인코더를 가져옴

정말 오토인코더의 정의 그대로, 항등 네트워크를 만들어 반복학습 시켜 우리가 인지할 수 없는 어떠한 피쳐를 습득시킨 듯 하다.

```
input_layer = Input(shape=(X.shape[1],))

## encoding part
encoded = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoded = Dense(50, activation='relu')(encoded)

## decoding part
decoded = Dense(50, activation='tanh')(encoded)
decoded = Dense(100, activation='tanh')(decoded)

## output layer
output_layer = Dense(X.shape[1], activation='relu')(decoded)
```

이 간편한 모델을 봐라. 완전 연결층 4개를 대칭으로 쌓은 것 뿐이다.

물론 이러나 저러나 예의 클래스 불균형 문제로부터 자유롭기 힘들어서, 데이터가 많은 쪽을 언더샘플링 하는 기법을 사용해 그를 보완하였다.

일반 송금 내역중 2000개만을 뽑아서 사용하였다.

단 한개의 클래스를 바이너리하게 가려내는 것을 학습시켜, 자동으로 그렇지 않은 것 또한 학습시키게 하려고 한 듯 하다.

```
Train on 1600 samples, validate on 400 samples
Epoch 1/10
1600/1600 [==============================] - 1s 358us/step - loss: 0.7218 - val_loss: 0.4842
Epoch 2/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.4609 - val_loss: 0.3261
Epoch 3/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.3052 - val_loss: 0.2208
Epoch 4/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.1991 - val_loss: 0.1238
Epoch 5/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.1303 - val_loss: 0.1265
Epoch 6/10
1600/1600 [==============================] - 0s 18us/step - loss: 0.1313 - val_loss: 0.0889
Epoch 7/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.0986 - val_loss: 0.0975
Epoch 8/10
1600/1600 [==============================] - 0s 17us/step - loss: 0.1131 - val_loss: 0.0793
Epoch 9/10
1600/1600 [==============================] - 0s 18us/step - loss: 0.0857 - val_loss: 0.0797
Epoch 10/10
1600/1600 [==============================] - 0s 19us/step - loss: 0.0966 - val_loss: 0.0721
```

`이게 에러가 줄긴 하네요...?`

다만, 이분은 이 오토인코더로 정답을 얻으려고 한 것이 아닌 것 같다.

그저, 우리가 EDA로 찾아내지 못할 representation을 학습시켜 그것을 이용하려고 했을 뿐.

이후 이 학습된 모델을 레이어 별로 잘라내어, 원래 차원으로 돌아오기 전 레이어까지 가져와 그 '숨은 표현' 을 이용하려고 한다. 위의 모델에서 3번 레이어까지만 떼어와 시퀀셜하게 쌓은 필자는, 그 모델로 단순한 추론을 진행한다.

그에 따른 결과값을 다시 시각화 했을때,

위에서 크게 구분되지 않던 non-fraud와 fraud의 feature가 확실히 갈렸고, 이런 모양은 간단한 선형 분류 모델만 사용해도 이제 구분이 가능한 수준이 된다.

```
train_x, val_x, train_y, val_y = train_test_split(rep_x, rep_y, test_size=0.25)
clf = LogisticRegression(solver="lbfgs").fit(train_x, train_y)
pred_y = clf.predict(val_x)

print ("")
print ("Classification Report: ")
print (classification_report(val_y, pred_y))

print ("")
print ("Accuracy Score: ", accuracy_score(val_y, pred_y))
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       754
         1.0       1.00      0.87      0.93       119

   micro avg       0.98      0.98      0.98       873
   macro avg       0.99      0.94      0.96       873
weighted avg       0.98      0.98      0.98       873


Accuracy Score:  0.9828178694158075
```

(아까의 rep x, rep y를 학습시킨다. 그리고 98퍼센트의 정확도를 얻게 되었다.

후기

태뷸러 데이터셋에 뉴럴네트워크를 적용할때, 단순히 뉴럴 네트워크를 통해 클래스를 분류하는 것이 아니라 오토인코더의 항등 특성 학습 성질을 이용해 일종의 전처리를 해준 것이 상당히 신선했다.

태뷸러 데이터셋을 다룰때 눈여겨볼만한 일인 것 같다. 정말 훌륭한 아이디어가 아닐 수 없다.