이성재 (19941117)
odobeuseKR@gmail.com

팀 정보
김준엽 (19920227)
이성재 (19941117)
고주원 (19950601)

Risk management in credit evaluation model by German credit data analysis

차례

서론

모든 금융권은 신용 대출에 대해 큰 리스크를 가지고 있다. 예를 들어 연 5%의 이자대출이 파산할 경우, 원금을 복구하기 위해서는 최소 같은 금액과 같은 이자율을 가지고 있는 대출자 20명 이상이 필요하다. 그래서 당연하게도 빌려주는 금융권에서는 빌려줄 사람이 빌려준 돈을 잘 갚을 것인지 정확하게 검증해야 한다.

대출이라는 특성상, 대출을 하는 것이 안 하는 것 보다 리스크가 크기 때문에, 신용평가 모델은 재현율이 특히 높아야 한다. 긍정라벨을 올바르게 판별하지 못해 거짓부정이 늘어난다면, 그 만큼 은행이 감당해야 하는 비용이 거짓 긍정으로 손해를 보는 경우보다 손해 금액이 더 크기 때문이다. 따라서 최대한 올바르게 대출자를 선별해야 한다는 점, 그리고 재현율이 높아야 한다는 점 두가지를 동시에 고려한 모델링이 되도록 노력하였다.

실제 은행에서도 이와 비슷한 Cost Matrix를 기반으로 대출심사를 진행하고, 대출 리스크를 산정할 것이기 때문에, 이 분석은 은행 대출 심사시에 참고자료로 사용될 수 있을 것으로 생각된다. 그리고 대출자 입장에서도 자신의 신용정보의 어떠한 부분이 중요한지 알고 이를 보완하려 노력할 수 있다는 점에서 의미가 있다. 이번 데이터 분석에서 우리 팀이 수집한 실제 데이터를 모델에 적용시켜 의미가 있는지 다시 한 번 확인하였다.

유명한 도박 만화, 카이지의 한 장면.

은행 입장에서는 이러한 악성 채무 가능성이 있는 고객을 사전에 판별하여 대출을 진행하지 않는 판별 모델이 필요하다.

본론
1. 원시 데이터 분석

신용도 분석을 위해 1994년에 발표된 German Credit Data 를 사용하였다.

데이터 출처

: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

데이터 개요

: 신용 대출을 신청한 독일인의 대출 승인 여부와 20가지의 개인정보
: 총 1000개의 대출 신청 및 승인 데이터
: Categorial 데이터와 이를 수치형으로 변환한 Numeric 데이터가 존재하며, 이는 StatLog 의 변환 기준을 따름

변수 이름 및 설명

1. Status of existing checking account : 예금 잔고 현황
   (유로화 통일 전, 독일에서 사용하던 DM 화폐 단위로 분류되었다)
2. Duration in month : 대출 기간
3. Credit history : 당행이나 타행에서의 대출 및 상환 기록
4. Purpose : 대출 목적
   (Numeric 데이터에서는 자동차에 대해서, 신차 혹은 중고차 구매 목적만 남김)
5. Credit amount : 대출 금액
6. Saving account / bond : 저축 예금 / 채권 잔고 현황
7. Present employment since : 고용 여부 및 근속 기간
8. Installment rate in percentage of disposable income : 가처분 소득 대비 할부 비율
9. Personal status and sex : 성별 및 결혼 여부
10. Other debtor / guarantors : 공동 채무자 / 보증인
11. Present residence since : 지속 거주 기간
12. Property : 부동산, 건물 등의 자산
13. Age in years : 연령
14. Other installment plans : 기타 할부 계획
15. Housing : 현재 주택의 소유 여부
16. Number of existing credits at this bank : 당행에서의 현재 대출상품 수
17. Job : 직업
18. Number of People being liable to provide maintenance for : 부양가족 수
19. Telephone : 휴대전화 보유 여부
20. Foreign worker : 외국인 노동자 여부

본론
2. 데이터 전처리

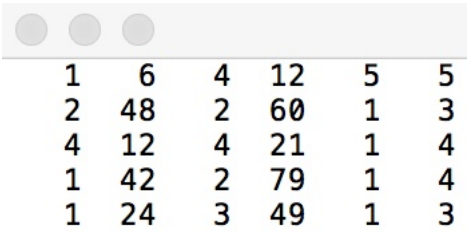
데이터가 보유한 Attribute의 기본적인 이해가 끝난 다음, 해당 데이터를 분석 환경으로 불러와 분석을 위한 전처리가 이루어져야 한다. 데이터의 확인 및 검증, 결측치 확인 및 처리, 연관성 분석 등이 이 과정에서 이루어진다.

A. Categorial / Numeric 데이터의 비교 및 수정.

출처를 통해 얻은 german.data.txt 와 german.data-numeric.txt 를 확인해 보면, 범주형 데이터는 21개, 수치형 데이터는 25개의 속성을 보유했다는 점을 알 수 있다. 이를 통해 유추할 수 있는 것은 특정 범주형 속성을 여러 개의 수치형 속성으로 나누어 표현했다는 점이었다. 데이터 설명에서는 StatLog 방식을 이용하여 범주형 데이터를 수치형 데이터로 변환했다고 하지만, 해당 방식을 정확하게 알지 못하여 두 데이터의 패턴을 분석하여 수치형 데이터의 속성명을 파악하였다.

	Check DM	Duration	Credit History	Credit Amount	Saving Account	Employment	Personal Status	Residence	Property	Age	...	New Car	Used Car	No Debtor	Co Debtor	Rent House	Own House	Job1
0	1	6	4	12	5	5	3	4	1	67	...	0	0	1	0	0	1	0
1	2	48	2	60	1	3	2	2	1	22	...	0	0	1	0	0	1	0
2	4	12	4	21	1	4	3	3	1	49	...	0	0	1	0	0	1	0
3	1	42	2	79	1	4	3	4	2	45	...	0	0	0	0	0	0	0
4	1	24	3	49	1	3	3	4	4	53	...	1	0	1	0	0	0	0

5 rows × 25 columns



전처리 과정에서 고려해야 했던 부분 중 하나는, 원본 Numeric 데이터의 두 번째 컬럼의 수치가 1자리수 또는 2자리수로 구성된다는 점이다. 여기서 1자리수 숫자는 띄어쓰기와 숫자가 결합된 형태이며, 2자리수 숫자는 공백이 없는 상태였다. 이 데이터를 읽어오는 과정에서 sep 변수를 '\s' 로 설정할 경우 띄어쓰기가 두 번 나타나는 과정에서 데이터가 하나씩 밀려나는 문제가 발생하였다. 이를 해결하기 위해 공백이 두 번 발생할 경우에 대해 예외를 처리하는 함수를 생성하였다. 함수의 코드는 아래와 같다.

```
dataset = []
for i in range(1000):
    a = list(" ".join(line[i].split()))
    memory = ""
    main = []
    for i in a:
        if i != " ":
            memory+=i
        else:
            main.append(int(memory))
            memory = " "
    if memory != " ":
        main.append(int(memory))
    dataset.append(main)
```

python 의 describe, info 기능을 활용하여 대략적인 속성의 분포를 알아보았다. 대부분이 범주형 속성으로, 연속적인 값을 가지는 속성의 중요도를 높게 생각하였다. 연속적인 값을 갖는 속성은 Check DM, Duration, Age, Credit Amount 등이 있다.

추가적으로 info 결과를 확인한 결과 NaN 값은 존재하지 않으며, 이에 데이터 전처리 과정 중 결측치 처리는 필요가 없을 것으로 결론내렸다.

	Check DM	Duration	Credit History	Credit Amount	Saving Account	Employment	Personal Status	Residence	Property	Age	...	New Car
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	...	1000.000000
mean	2.577000	20.903000	2.54500	32.711000	2.105000	3.384000	2.68200	2.845000	2.358000	35.546000	...	0.234000
std	1.257638	12.058814	1.08312	28.252605	1.580023	1.208306	0.70808	1.103718	1.050209	11.375469	...	0.423584
min	1.000000	4.000000	0.00000	2.000000	1.000000	1.000000	1.00000	1.000000	1.000000	19.000000	...	0.000000
25%	1.000000	12.000000	2.00000	14.000000	1.000000	3.000000	2.00000	2.000000	1.000000	27.000000	...	0.000000
50%	2.000000	18.000000	2.00000	23.000000	1.000000	3.000000	3.00000	3.000000	2.000000	33.000000	...	0.000000
75%	4.000000	24.000000	4.00000	40.000000	3.000000	5.000000	3.00000	4.000000	3.000000	42.000000	...	0.000000
max	4.000000	72.000000	4.00000	184.000000	5.000000	5.000000	4.00000	4.000000	4.000000	75.000000	...	1.000000

8 rows × 25 columns

B. 연관성 분석1

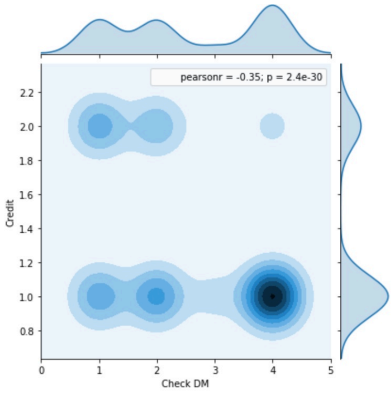
결과값인 ‘Credit’ 속성을 제외하면 24개의 속성이 존재하며, 이 중에서 ‘Credit’ 에 큰 영향을 미치는 속성이 무엇인지 알아보기 위해 예시 스케치에서 제시된 방법을 활용하여 보았다. Good / Bad Credt 별로 각각의 속성 평균값을 알아내어, 각각의 차이를 구하는 것이다. 더 나아가 해당 결과값을 속성 전체 평균값으로 나누어 최대한 일반화가 되도록 정렬해 보았다.

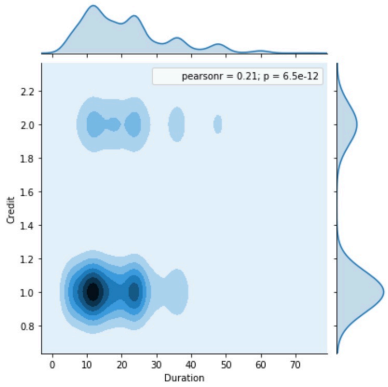
	Good	Bad	Result
Used Car	0.122857	0.056667	0.642626
Check DM	2.865714	1.903333	0.373450
Saving Account	2.290000	1.673333	0.292953
Credit History	2.707143	2.166667	0.212368
Own House	0.752857	0.620000	0.186335

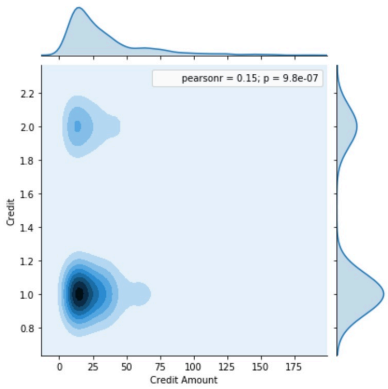
	Good	Bad	Result
Credit	1.000000	2.000000	-0.769231
Co Debtor	0.032857	0.060000	-0.662021
Rent House	0.155714	0.233333	-0.433626
New Car	0.207143	0.296667	-0.382580
Credit Amount	29.862857	39.356667	-0.290233
Duration	19.207143	24.860000	-0.270433

Credit 에 긍정적인 영향을 미치는 3대 요소는 중고차 구매 목적, 예금 잔고 현황, 저축 현황 이었으며, 부정적인 영향을 미치는 3대 요소는 공동 채무자의 여부, 전월세 주택 보유, 신차 구매 목적 등 이었다. 물론 스케치에서도 말했듯이 단순한 평균값의 차이로 이를 단정할 수는 없지만, 대략적인 지표로 활용할 수 있겠다.

이 중에서 몇몇 연속적 값을 갖는 속성을 추려내어 Credit 과의 상관관계를 심층적으로 분석해 보았다. 여기에 사용된 속성은 Check DM, Duration, Credit Amount 이며, python seaborn 패키지의 jointplot 기능을 사용하였다.







연속적인 값을 갖는 속성이기에 세 값을 활용하여 jointplot 을 그렸지만, Result 값이 가장 높게 나왔던 Check DM 을 제외하고는 큰 상관관계를 찾아보기 힘들었다. 위에서 분석한 연관성이 어느 정도 신뢰도가 있을 것으로 예측하고, 1차적으로 Used Car, Check DM, Co Debtor, Rent House 를 핵심 성분으로 사용하는 것이 좋겠다는 결론을 내렸다.

C. 연관성 분석2

조금 더 깊이 있는 방법으로 각 속성과 Credit 의 연관성을 찾기 위해 scikit - learn 패키지의 feature-selection 패키지 중에서 recursive feature elimination 함수를 사용하여 보았다.

```
from sklearn.feature_selection import RFE

rfe_model = RFE(logistic_model, 5)
rfe_model.fit(X_train_all, y_train)
print(rfe_model.support_)
print(rfe_model.ranking_)
```

```
[ True False False False False False False False False False False False
 False False  True False  True  True  True False False False False False]
[ 1 14  4 20 10 16 11 15  5 18 12  8  7 13  1  6  1  1  1  2  9  3 19 17]
```

```
rfe_list = list(rfe_model.support_)
indices = [i for i, x in enumerate(rfe_list) if x]
result_list = []
for i in indices:
    result_list.append(X_train_all.columns[i])
result_list
```

```
['Check DM', 'Foreign', 'Used Car', 'No Debtor', 'Co Debtor']
```

RFE 함수를 통해 Credit 과 가장 연관성이 높은 상위 5개 속성을 뽑아낸 결과, Check DM, Foriegn, Used Car, No Debtor, Co Debtor 가 나타났다. 이 중에서 Check DM, Used Car, Co Debtor 는 연관성 분석1 과 2 모두에서 나타났기 때문에 모델에 사용하기 적합한 속성으로 분류할 수 있겠다고 판단하였다.

본론
3. 가설 설정

필자는 앞의 주 성분 분석을 통해 아래와 같은 가설을 세웠다.

가설의 종속 변수는 credit의 good, bad 여부를 가리는 binary value이다.
독립 변수는 아래와 같다.

우선, Check DM 의 value가 높을 수록 credit value가 낮을 것이라고 가정했다. Check DM column이 1이면, 지금 현재 계좌가 마이너스 계좌라는 것이고, 해당 대출심사자가 이미 대출을 받았다는 의미이다. 보통 대출을 받으면 그만큼, 실제로 또 대출을 받을 수 있는 양이 줄어들기 때문에, 대출 심사에 부적격 판정(credit=2)을 받을 확률이 높다고 생각 했다. 반대로 Check DM column이 4이면 이 은행에 account가 없다는 것을 의미한다. 은행은 아마도, 해당 대출자가 이 대출 거래를 통해 해당 은행과 신용 기록을 쌓아나가, 장기적인 고객이 되길 원할 것이다. 그렇기 때문에 대출 심사에 적격 판정(credit=1)을 받을 것이다. Check DM이 3이면 계좌에 일정 금액 이상이 있다는 의미이기 때문에, Check DM이 4인 것과 관계가 없지만, 3인 경우에는 63개의 observation이 있고, 4인 경우가 394개가 있기 때문에, 거의 6배 차이가 나므로 전체적인 분포상 Check DM과 credit value가 부의 상관관계가 있다고 보기에 적절할 수 있다고 보았다.

또한 , Used car의 value가 높을 수록 대출 심사에 적격 판정(credit=1)을 받을 것이라고 가정했다. used car의 value가 1이라는 것은 중고차를 구매하기 위해 대출을 신청했다는 것을 의미한다. 중고차를 구매한다는 것은 대출 심사에서 중요한 두가지 의미가 있다. 첫 번째로, 자동차는 교통수단의 하나로써, 사회에서 생산활동에 간접적이지만 중요하게 기여하는 교통수단 중에 하나이다. 자동차가 있다면, 그 만큼 자동차의 소유자가 일할 수 있는 범위도 넓어지고, 그렇기 때문에 직장이 없다면 직장을 얻을 확률이 높아지고, 직장이 있다면, 직장을 편하게 다닐 수 있기 때문에 오랬동안 근속할 가능성이 높아진다. 사회에서 생산활동을 하고 있다는 것은 대출을 갚을 가능성이 높아진 다는 의미이기 때문에 대출시에 높은 점수를 받을 수 있다고 생각했다.두 번째로는, 중고차를 구매한다는 것은, 새 차를 구매할 때 보다 낮은 가격으로 자동차를 구매한다는 것이다. 일반적으로 대출 금액이 낮을 수록, 대출을 받을 수 있는 확률이 높아진다.

마지막으로, co-applicant 가 높을 수록, 대출 심사에 적격 판정(credit =1)을 받을 확률이 높다고 생각 했다. Co-applicant란, 대출에 공동 신청자의 여부이다. 대출을 신청할 때 같이 신청한 사람이 있다면, 그 만큼 대출자가 부담해야 하는 대출 금액이 줄어든다. 따라서 갚을 확률이 높아질 것이기 때문에 대출 심사에 적격 판정을 받을 확률도 높아질 것이라고 생각했다.

본론

4. 신용도 판별 모형 without CM

위에서의 결과를 종합하여 보았을 때, 한 사람의 신용도를 판별하는 가장 핵심적인 속성은 예금 잔고 현황이지만, 몇 가지 속성을 추가적으로 함께 고려하여 다양한 모형에 적용시켜 보고자 한다. 사용될 모형은 Logistic Regression, KNN, Decision Tree, Random Forest, Support Vector Machine, DNN, K Coss Validation 이며 이 중에서 가장 좋은 판별력을 가진 모형을 선택하고자 한다. 더 나아가 제시된 Cost Matrix 를 감안하여 모형별로 거짓긍정과 참부정 결과에 가중치를 주어 표준화한 점수를 새롭게 생성하고, 이를 모형 선택의 기준으로 삼고자 한다.

A. Train - Test data split : 학습 - 검정 데이터 분할

먼저 모형에 학습시킬 데이터와 학습된 모형을 검정할 데이터를 분할하는 과정이 필요하다. 하나의 데이터로 모형을 만들고, 이를 다시 검정한다면 답을 아는 상태에서 시험을 보는 것과 다름없기 때문이다. 분할을 위해서 Python 의 sklearn.model_selection 패키지에서 train_test_split 기능을 가져와 사용하였다. test_size 를 0.3 으로 주어 1000개의 데이터 중에서 700개를 학습 데이터로, 300개를 테스트 데이터로 사용하였다. 테스트 데이터는 X_sel 에 앞에서 Credit 에 중요한 영향을 미치는 것으로 판단되었던 Credit DM, Co Debtor, Rent House, Used Car, New Car 속성만을 뽑아서 넣었다.

Train - Test data split

```
from sklearn.model_selection import train_test_split
```

```
X_all = german_num.drop('Credit', axis = 1)
X_sel = german_num[['Check DM', 'Co Debtor', 'Rent House', 'Used Car', 'New Car']]
y = german_num['Credit']
X_train_all, X_test_all, y_train, y_test = train_test_split(X_all, y, test_size=0.30, random_state=101)
X_train_sel, X_test_sel, y_train_sel, y_test_sel = train_test_split(X_sel, y, test_size = 0.30, random_state = 101)
```

```

```

B. 모형 판별 기준의 정립

	Good Pred	Bad Pred
Good	185	19
Bad	56	40

학습 데이터를 통해 모형을 만들고, 이를 바탕으로 테스트를 진행하였을 때, 그 결과는 오른쪽과 같은 매트릭스 형태로 나타나게 된다. 왼쪽의 Good / Bad 가 실제 Credit 데이터이며, 상단의 Pred 는 해당 값을 Good / Bad 로 추정한 것이다. 이에 실제 데이터를 어느 정도 정확성으로 추정하였는지 모델을 평가할 수 있지만, 더 나아가 Cost Matrix 와 비교하여 간단한 점수를 만들 수 있다.

Cost Score = ((Good - Bad Pred) * 1 + (Bad - Good Pred) * 5) / len (test data)

위의 식을 사용하여 테스트 데이터의 개수와 상관없는 Cost Score를 만들어 모형을 판단하였다. 오른쪽과 같은 Confusion Matrix 결과가 나타날 경우 Cost Score 는 0.99 의 점수가 나타난다. 점수는 높을 수록 많은 비용을 야기하는 것으로, Cost Score 를 낮추는 것이 관건이다.

C. Logistic Regression Model

참과 거짓 등의 두 가지 결과값을 갖는 데이터를 학습시키는 것에 있어서, 가장 대표적인 모델은 로지스틱 회귀 모형일 것이다. 강의를 들으며 배웠던 내용을 토대로 Python 의 sklearn.linear_model 패키지의 LogisticRegression 기능을 불러와 사용하도록 한다.

Logistic Regression Basic

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
logistic_model = LogisticRegression()
logistic_model.fit(X_train_sel, y_train_sel)
logistic_prediction = logistic_model.predict(X_test_sel)
```

```
print(classification_report(y_test_sel, logistic_prediction))
print(confusion_matrix(y_test_sel, logistic_prediction))
print((confusion_matrix(y_test_sel, logistic_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, logistic_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.71      0.92      0.80       204
          2       0.56      0.21      0.30        96

avg / total       0.66      0.69      0.64       300

[[188  16]
 [ 76  20]]
1.32
```

Logistic Regression Model을 이용하여 Confusion Matrix 를 구성한 결과, 76명의 실제 Bad Credit 을 Good 으로 판별하였으며, 16명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 1.32 점으로 나타났으며, 총 오류의 개수는 92개 이다.

D-1. K - Nearest Neighborhood Model

Credit 데이터는 결국 Good / Bad 군집을 분류하는 것이므로, KNN 모델을 사용하여 분류가 가능하다고 생각하였다.

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(X_train_sel, y_train)
knn_prediction = knn_model.predict(X_test_sel)
```

```
print(classification_report(y_test_sel, knn_prediction))
print(confusion_matrix(y_test_sel, knn_prediction))
print((confusion_matrix(y_test_sel, knn_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, knn_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.73      0.73      0.73       204
          2       0.42      0.42      0.42        96

avg / total       0.63      0.63      0.63       300

[[149  55]
 [ 56  40]]
1.11666666667
```

KNN 을 이용하여 Confusion Matrix 를 구성한 결과, 56명의 실제 Bad Credit 을 Good 으로 판별하였으며, 55명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 1.12 점이며 총 오류의 개수는 111개 이다. Logistic Regression 모델과의 비교점은, KNN 이 잘못된 판단을 19회

더 많이했음에도 Bad Credit 을 Good Credit 으로 잘못 판단한 숫자가 56으로 더 적었기에 Cost Score 가 더 높았다는 점이다.

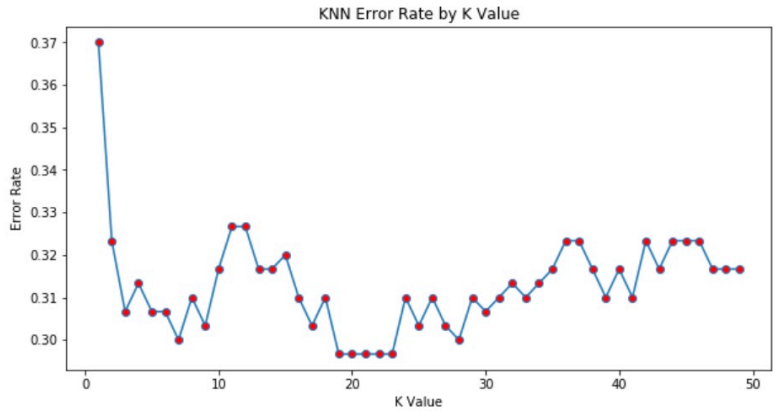
D-2. Upgraded K - Nearest Neighborhood Model

위에서 작성한 KNN 모델은 K 값을 1로 주었을 때의 결과이다. 이는 가장 가까운 점을 1개만 고려하기 때문에 K 값의 변화에 따라 더 좋은 모델이 나타날 수 있다. 이에 K 값을 1부터 50까지 변화시키며 최적의 K값을 찾아보았다.

Upgraded KNN

```
knn_error_rate = []

for i in range(1, 50):
    knn_upgraded_model = KNeighborsClassifier(n_neighbors=i)
    knn_upgraded_model.fit(X_train_sel, y_train_sel)
    i_knn_model = knn_upgraded_model.predict(X_test_sel)
    knn_error_rate.append(np.mean(i_knn_model != y_test_sel))
```

```
plt.figure(figsize = (10, 5))
plt.plot(range(1, 50), knn_error_rate, marker = 'o', markerfacecolor = 'red')
plt.title('KNN Error Rate by K Value')
plt.xlabel('K Value')
plt.ylabel('Error Rate');
```




```
knn_error_rate.index(min(knn_error_rate)) + 1
# find minimum error rate K Value
```

```
19
```

K 값의 변화에 따라서 예측한 모델이 결과를 얼마나 정확하게 예측하는지 Error Rate 를 계산하여, Error Rate 가 가장 적은 K 값으로 KNN 모델을 만들도록 한다. 이 때의 K값은 19로 나타났다.

```
knn_final_model = KNeighborsClassifier(n_neighbors=19)
knn_final_model.fit(X_train_sel, y_train_sel)
knn_final_prediction = knn_final_model.predict(X_test_sel)

print('With K Value = 2')
print(classification_report(y_test_sel, knn_final_prediction))
print(confusion_matrix(y_test_sel, knn_final_prediction))
print((confusion_matrix(y_test_sel, knn_final_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, knn_final_prediction)[0][1]) / len(y_test_sel))
```

```
With K Value = 2
             precision    recall  f1-score   support

          1       0.77      0.80      0.79       204
          2       0.54      0.49      0.51        96

avg / total       0.70      0.70      0.70       300

[[164  40]
 [ 49  47]]
0.95
```

K 값이 19인 KNN 을 이용하여 Confusion Matrix 를 구성한 결과, 49명의 실제 Bad Credit 을 Good 으로 판별하였으며, 40명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 0.95 점이며 총 오류의 개수는 89개 이다. 전체 오류의 개수도 KNN 보다 적을 뿐 아니라, Cost Score 도 지금까지의 판별 모델 중에서 가장 좋게 나왔다.

E. Decision Tree Classifier

의사결정 나무 모델 또한 이처럼 두 가지 경우에 대해 판별할 때 자주 사용되는 모델이다. 의사결정 나무 모델을 사용하여 판별한 결과는 다음과 같다.

4. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
dtree_model = DecisionTreeClassifier()
dtree_model.fit(X_train_sel, y_train_sel)
dtree_prediction = dtree_model.predict(X_test_sel)
```

```
print(classification_report(y_test_sel, dtree_prediction))
print(confusion_matrix(y_test_sel, dtree_prediction))
print((confusion_matrix(y_test_sel, dtree_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, dtree_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.71      0.90      0.80       204
          2       0.52      0.23      0.32        96

avg / total       0.65      0.69      0.64       300

[[184  20]
 [ 74  22]]
1.3
```

Decision Tree를 이용하여 Confusion Matrix 를 구성한 결과, 74명의 실제 Bad Credit 을 Good 으로 판별하였으며, 20명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 1.3 점이며 총 오류의 개수는 94개 이다. KNN 이나 Logistic 모델보다 나을 점이 없어 보인다.

F. Support Vector Machine

마지막으로 사용된 모델은 Support Vector Machine 이며, 결과값은 다음과 같다.

Support Vector Machine

```
from sklearn.svm import SVC
```

```
svc_model = SVC()
svc_model.fit(X_train_sel, y_train_sel)
svc_prediction = svc_model.predict(X_test_sel)
```

```
svc_model = SVC()
svc_model.fit(X_train_sel, y_train_sel)
svc_prediction = svc_model.predict(X_test_sel)
print(classification_report(y_test_sel, svc_prediction))
print(confusion_matrix(y_test_sel, svc_prediction))
print((confusion_matrix(y_test_sel, svc_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, svc_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.69      0.93      0.79       204
          2       0.46      0.14      0.21        96

avg / total       0.62      0.67      0.61       300

[[189  15]
 [ 83  13]]
1.43333333333
```

본론
5. 신용도 판별 모형 with CM

앞에서 사용된 모델들은 모두 주어진 CM (Cost Matrix)를 고려하지 않고 만들어진 것이다. 이번에는 Logistic Regression 과 Support Vector Machine 의 Class Weight 값을 활용하여 앞에서 임의로 만든 Cost Score 를 최저로 만드는 모델을 구현할 것이다.

A. Logistic Regression Model with Class Weight

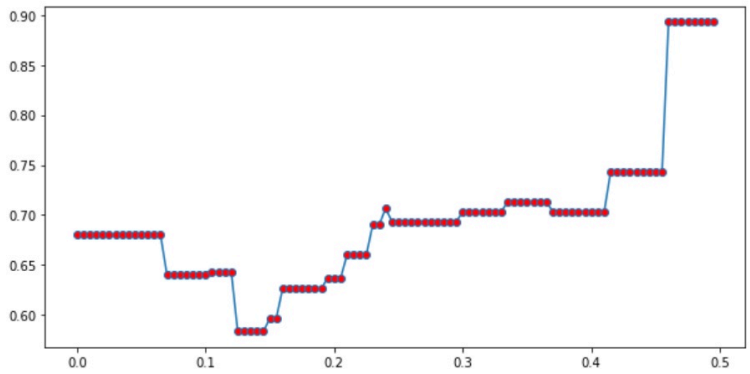
먼저 Logistic Regression 에서 Class Weight 를 사용하여 모델을 구성해 보았다. 이 때, for 문을 활용하여 0부터 0.5 사이의 weight 를 구성해 보며, 최종적인 Cost Score 가 가장 낮은 weight 를 선택하도록 한다.

Logistic Regression with Class Weight

```
logistic_weight = []
for i in np.arange(0, 0.5, 0.005):
    logistic_model = LogisticRegression(class_weight={1:i})
    logistic_model.fit(X_train_sel, y_train_sel)
    logistic_prediction = logistic_model.predict(X_test_sel)
    logistic_weight.append((confusion_matrix(y_test_sel, logistic_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, logistic_prediction)[0][1]) / len(y_test_sel))
logistic_weight.index(min(logistic_weight)) * 0.005
```

```
0.125
```

```
plt.figure(figsize = (10, 5))
plt.plot(np.arange(0, 0.5, 0.005), logistic_weight, marker = 'o', markerfacecolor = 'red');
```



Logistic Weight 에 따른 Cost Score 의 변화를 살펴보면 위의 그래프와 같다. 위에 나타난 0.125 값 까지는 점차 Cost Score 가 감소하지만, 그 이후부터는 다시 Cost Score 가 증가하는 추세를 보인다. 여기서 구한 0.125 Weight 값을 모델에 넣으면 다음의 결과가 나타난다.

```
logistic_model = LogisticRegression(class_weight={1:0.125})
logistic_model.fit(X_train_sel, y_train)
logistic_prediction = logistic_model.predict(X_test_sel)
print(classification_report(y_test_sel, logistic_prediction))
print(confusion_matrix(y_test_sel, logistic_prediction))
print((confusion_matrix(y_test_sel, logistic_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, logistic_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.89      0.39      0.54       204
          2       0.41      0.90      0.56        96

avg / total       0.73      0.55      0.55       300

[[ 79 125]
 [ 10  86]]
0.583333333333
```

Class Weight 를 고려한 Logistic Model을 이용하여 Confusion Matrix 를 구성한 결과, 10명의 실제 Bad Credit 을 Good 으로 판별하였으며, 125명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 0.58 점이며 총 오류의 개수는 135개 이다. 전체 오류의 개수는 이전보다 훨씬 높은 값을 가지지만, Cost Score 는 경이로울 정도로 높은 값이 나타난다.

B. SVM with Class Weight

이번에는 Class Weight 를 조정할 수 있는 다른 모델인 SVM 을 이용하여 위와 동일한 방식으로 테스트를 진행해 보았다. 이 때, Weight 값은 아까보다 조금 더 넓은 범위인 0부터 1까지로 진행하였다.

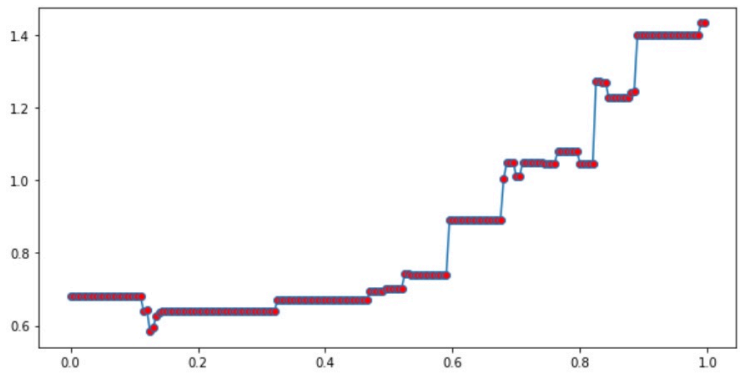
SVM with Class Weight

```
from sklearn.svm import SVC
```

```
svm_weight = []
for i in np.arange(0, 1, 0.005):
    svc_model = SVC(class_weight = {1:i})
    svc_model.fit(X_train_sel, y_train_sel)
    svc_prediction = svc_model.predict(X_test_sel)
    result = (confusion_matrix(y_test_sel, svc_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, svc_prediction)[0][1]) / len(y_test_sel)
    svm_weight.append(result)
( svm_weight.index(min(svm_weight)) + 1 ) * 0.005
```

```
0.13
```

```
plt.figure(figsize = (10, 5))
plt.plot(np.arange(0, 1, 0.005), svm_weight, marker = 'o', markerfacecolor = 'red');
```



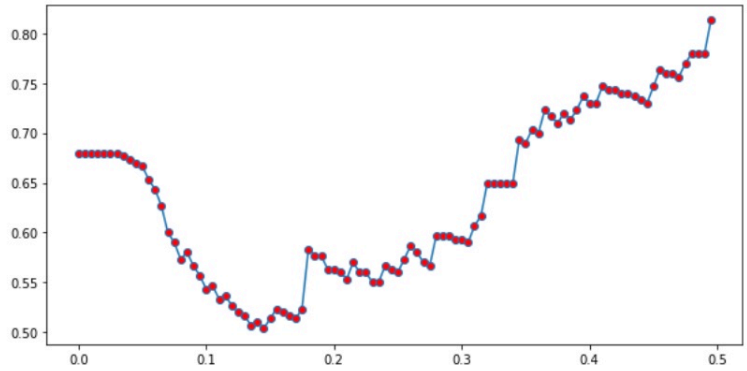
```
svc_model = SVC(class_weight={1:0.13})
svc_model.fit(X_train_sel, y_train_sel)
svc_prediction = svc_model.predict(X_test_sel)
print(classification_report(y_test_sel, svc_prediction))
print(confusion_matrix(y_test_sel, svc_prediction))
print((confusion_matrix(y_test_sel, svc_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, svc_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.88      0.39      0.54       204
          2       0.41      0.89      0.56        96

avg / total       0.73      0.55      0.55       300

[[ 80 124]
 [ 11  85]]
0.596666666667
```

Class Weight 를 고려한 SVM을 이용하여 Confusion Matrix 를 구성한 결과, 11명의 실제 Bad Credit 을 Good 으로 판별하였으며, 124명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 0.596 점이며 총 오류의 개수는 135개 이다.

여기서 욕심을 내서 Feature 를 조금 더 많이 고려하면 더 좋은 결과값을 얻을 수 있지 않을까 하는 생각이 들게 되었다. 이를 Logistic Regression with Class Weight 모델에 적용해 보았다.

C. Logistic Regression Model with Class Weight and more Feature




```
logistic_model = LogisticRegression(class_weight={1:0.145})
logistic_model.fit(X_train_sel, y_train)
logistic_prediction = logistic_model.predict(X_test_sel)
print(classification_report(y_test_sel, logistic_prediction))
print(confusion_matrix(y_test_sel, logistic_prediction))
print((confusion_matrix(y_test_sel, logistic_prediction)[1][0] * 5
 + confusion_matrix(y_test_sel, logistic_prediction)[0][1]) / len(y_test_sel))
```

```
             precision    recall  f1-score   support

          1       0.93      0.41      0.57       204
          2       0.43      0.94      0.59        96

avg / total       0.77      0.58      0.57       300

[[ 83 121]
 [  6  90]]
0.503333333333
```

약간 더 많은 Feature의 Class Weight 를 고려한 Logistic Model을 이용하여 Confusion Matrix 를 구성한 결과, 6명의 실제 Bad Credit 을 Good 으로 판별하였으며, 121명의 Good Credit 을 Bad 로 판별하였다. 이에 Cost Score 는 약 0.50 점이며 총 오류의 개수는 127개 이다. 이전보다 훨씬 좋은 결과를 얻을 수 있었으며 적절한 Feature 를 추가할 경우 더 좋은 예측이 가능하다는 점을 알 수 있었다.

본론
6. 실제 데이터에 접목

주어진 데이터를 넘어, 실제 데이터를 모델에 넣어보면 새로운 insight 가 나올 것으로 생각하여, Google Form 을 이용해 주변 사람들로부터 7개 정도의 설문 결과를 데이터로 만들어 테스트 하였다. 조사 대상은 모두 팀원들의 지인으로, 악성 채무경험이 없고 앞으로도 적절하게 채무를 변제할 것으로 보인다.

Google Form 의 설문 주소

https://docs.google.com/forms/d/14TBAE81UVga9BCQVhC2UVPmsjW2ZC9J65HMoWcGm3o

	Check DM	Co Debtor	Rent House	Used Car	New Car	Credit Amount	Duration	Own House	Credit History	Saving Account
0	0	0	0	0	0	30	0	1	0	2
1	1	0	1	1	0	25	0	0	0	3
2	1	0	1	0	0	5	0	0	0	1
3	3	0	1	0	0	0	1	0	0	1
4	3	1	0	0	0	20	5	1	0	1
5	1	0	0	0	0	0	0	0	0	1
6	0	0	1	0	0	50	0	0	0	5

```
logistic_prediction = logistic_model.predict(F_test)
```

```
logistic_prediction = logistic_model.predict(F_test)
print(classification_report(y_Ftest, logistic_prediction))
print(confusion_matrix(y_Ftest, logistic_prediction))
```

```
             precision    recall  f1-score   support

          1       0.00      0.00      0.00         7
          2       0.00      0.00      0.00         0

avg / total       0.00      0.00      0.00         7

[[0 7]
 [0 0]]
```

하지만 긍정적인 예측과 다르게, 최종 모델은 조사 대상인 7명 모두 악성 채무자로 판별하였다. 그 원인에 대해 알아보기 위해서는 각자의 데이터를 세부적으로 살펴보아야 겠지만, 아무래도 1990년대 독일인의 데이터 분석 결과로 2018년 현재 대한민국 시민의 신용도를 판별한다는 점에 문제가 있어서라고 생각된다. 뿐만 아니라, Cost Matrix 를 고려하며, 악성 채무자를 피하기 위해 선량한 채무자를 과다하게 악성 채무자로 판별하는 경향이 있기 때문에 이러한 결과가 나타난 것으로 보인다.

결론
1. 분석 결과의 정리

최종적으로 분석 결과를 정리하면 다음과 같다. 먼저 원시 데이터를 분석하고, Credit 과 각각의 변수가 어느 정도의 연관성을 갖는지에 대해 알아보았다. 이 때 5개 정도의 속성을 우선적으로 뽑아 다양한 모델들에 학습시키고, 테스트를 진행하였다. 하지만 각각의 모델은 판별에 있어서 전체 오류의 개수만 줄이는 방향으로 발전할 뿐이었고, 이에 비용 행렬을 고려한 새로운 모델을 만들 필요가 있었다. Cost Sensitive 와 관련된 내용을 학습하던 중, Logistic Regression 모델과 Support Vector Machine 모델에서 Class Weight 인자를 조절하여 비용 행렬을 참고할 수 있음을 알게 되었고, 이를 바탕으로 Bad Credit 을 가진 사람에 대해 민감하게 판별하는 모델의 구현이 가능하였다.

	cols	coefs
0	Check DM	-0.523568
1	Co Debtor	0.051849
2	Rent House	0.576300
3	Used Car	-0.970096
4	New Car	0.474604
5	Credit Amount	0.004388
6	Duration	0.044466
7	Own House	0.076745
8	Credit History	-0.262483
9	Saving Account	-0.148006

Logistic Regression Model의 coef는 왼쪽과 같다. 우리의 가설과 맞는 부분은 Check DM과 Used car 부분이다. Check DM이 높을 수록 Credit 이 1이 될 가능성이 높아졌다. Co-Debtor 부분은 Co-applicant 와 같은 부분인데, 이 부분은 오히려 Co-Debtor가 높을 수록 Credit을 받지 못할 확률이 높아지는 것으로 보인다.

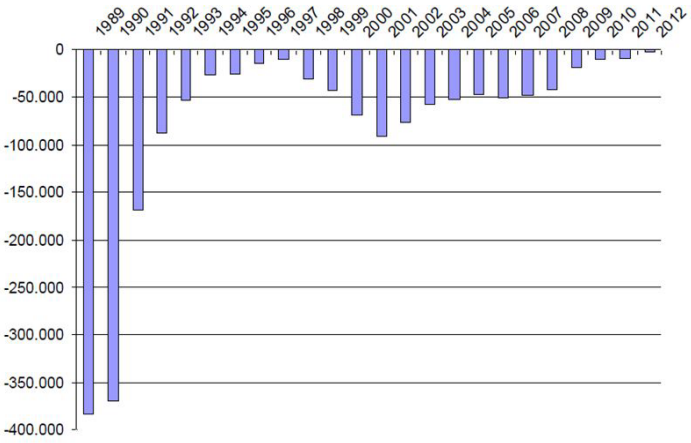
다시 생각 해 보면, 건전한 채무자일 수록 혼자서 해당 대출을 감당할 수 있을 것이다. 건전하지 못한 채무자일 수록 Co-applicant와 같이 대출을 받으려고 하여 자신의 대출받지 못할 리스크를 줄이려고 했을 것이다. 역시 대출자는 무서운 사람들이다. 나머지 feature 중에서도 의미있는 부분을 언급해보자. 먼저 집을 빌렸다면(Rent House), 대출을 받지 못할 확률이 큰 것으로 나타난다. 사람이 가질 수 있는 가장 중요하고 안전 자산인 집이 없다는 것은 다른 자산도 없을 가능성이 높을 것이다. 자산이 없다는 것은 그만큼 대출을 받을 때 좋은 점수를 받기 어려울 것이다. Own House가 체크되어 있어도 대출에 부정적인 영향을 끼친다는 점은, 집이라는 자산을 가지고 있더라도, 아마도 대출하려고 하는 정황상, 온전히 자기 것이 아닌 대출을 받아 산 집일 가능성이 높을 것이고 대출이 원래 있다면, 새로운 대출을 받는 데 어려움이 있을 수 밖에 없다.

새로운 차를 사려고 돈을 빌릴 수록(New Car) 대출을 받을 가능성이 줄어든다. 자동차는 새로 샀을 때, 감가상각이 굉장히 빨리 이루어지는 자산이다. 돈이 없어서 자동차를 사는 주제에,새로 산 자동차의 감가상각보다 더 빠르게 돈을 벌 수 있는 가능성은 없을 것이다. 그렇기 때문에 은행에서는 새 차를 사는 사람을 부정적으로 본다고 볼 수 있다. Saving Account는 현금자산을 의미하기 때문에,아무래도 Account가 많을 수록 거래기록이 많아 대출에 긍정적인 영향을 미쳤을 것이고, Credit history도 비슷한 맥락에서 의미가 있을것이다.

결론
2. 분석 활용 방안

분석에 사용된 데이터가 1994년에 제공된 독일인의 신용도 분석 결과로, 당시 독일은 통일이 된 지 5년이 채 되지 않은 상황이었다는 점에 착안하여 2018년 대한민국에 도움이 될 만한 방안을 생각해 보았다.

그래프를 확인해 보면 통일 초반에 상대적으로 경제상황이 좋지 않았던 동독에서 서독으로 많은 이주가 있는 것을 볼 수 있다. 우리나라 역시도 통일 직후 북한에서 많은 이주가 있을 것으로 예상된다. 그렇다면 북한 금융 데이터가 전산화가 제대로 되어있지 않은 상태에서 금융기관이 북한사람에게 돈을 빌려준다고 할 때 남한사람의 모델을 적용하는 것은 적합하지 않을 것으로 본다.



동독 주민의 서독으로 순 이주에 대한 그래프 데이터
- 독일통일 25주년의 경제적 성과 및 연구 동향 (김창권)

German Credit Data의 정확한 생성년도는 나타나지 않았다. 물론 추측이긴 하지만 데이터를 살펴보면 마지막 항목에 외국인인지 묻는 부분에서 대부분의 사람들이 외국인이라고 답하였다. 그 이유가 무엇인지 곰곰이 생각 해봤을 때, 우리는 1990년대 독일 통일 초반에 많은 동독 사람들이 서독에서 돈을 빌리려고 했고, German Credit Data 역시도 그 영향 때문에 외국인(동독 사람)의 비율이 높은 것이라고 추측했다.

German Credit Data 는 현대의 다양한 변수들(공과금 납부, 통신요금 연체 등)이 고려되지 않은 데이터다. 통일 직후에는 북한 사람들도 이런 변수들이 측정되어 있지 않은 상태일 것이다. 측정 되어 있다고 하더라도 과연 신뢰할 수 있는 데이터 인지도 의문이 든다. 독일 통일이 해당 신용도 판별 데이터에 영향을 주었다면 그 데이터를 토대로 작성한 우리의 모델은 한반도 통일 직후 유용하게 사용될 수 있을 것이라고 생각한다.

위기는 곧 기회라고 한다. 상대적으로 갚을지 안 갚을지 모르는 리스크를 가지면서 굳이 그들에게 돈을 빌려줄 필요는 없다. 하지만 그렇게 우물쭈물 하는 사이 다른 경쟁자들은 새로운 시장을 개척할 것이다.

결론
3. 분석 개선사항

우리가 가지고 있는 전체 분석의 개선점은 다음과 같다.

1. Data engineering

먼저 해당 데이터셋이 1000개 라는 점이 아쉬웠다. 데이터의 observation은 많으면 많을 수록 좋다. 1000개는 데이터 분석을 하기에는 그렇게 큰 값은 아니다. 특히 DNN 과 같은 딥러닝 알고리즘을 쓰기에는 그렇게 좋은 데이터 셋은 아니다. 또한, 데이터가 독일인 데이터라고 하기에는 매우 치우친 데이터이다. 외국인 노동자가 전체의 데이터의 95% 이상을 차지하고 있다. 그런데 feature 설명에서는 단순히 foreign worker라고 되어 있을 뿐이다. 매우 모호한 단어이다. 1990년대에 독일이 통일이 되었기 때문에, 서독 데이터인지, 동독 데이터인지도 판별하기 어렵고, 서독 입장에서 동독 노동자를 외국인 노동자로 판별한 것인지도 알 수 없다. 그리고 외국에서 일하고 있는 독일인 노동자인지,
혹은 독일 내에 근무하고 있는 외국인 노동자인지도 알 수 없었다. 마지막으로 numeric data를 보면서 굉장히 아쉬운 점은, feature의 one hot encoding이 제대로 되어 있지 않았다는 점이였다. 예를들어 Attribute 4의 purpose 같은 경우에는 numeric data에 car(new)와 car(used)부분만 구현되어 있었다. PCA를 하면서 데이터의 category를 2진수로 표현한 것은 굉장히 신선한 접근이였다. 보통 One hot encoding을 할 때에는 N개의 feature는 N개나 N-1개의 feature로 표현하기 때문이다. 그렇지만 이렇게 처리했다는 것을 description에 언급을 했어야 했다. 이 부분을 찾느라고 굉장히 고생했다.

2. Modeling

먼저, Cost matrix가 이자율에 대해 조금 더 업그레이드가 되었다면, 실제 경제 상황을 반영할 수 있을 것이라고 생각했다. 예를 들어 X%만큼의 이자를 내야하는 악성 채무자가 돈을 내지 못했다면, (100/X)*(Net Present Value)만큼의 같은 채무자가 존재해야 악성 채무자 만큼의 손해를 만회할 수 있다. 또한, 돈을 받을 수 있는데도, 악성채무자로 분류한 거짓 긍정의 사례도 미래의 기대값에 대한 포기이기 때문에, 이러한 부분이 Cost Matrix에 반영이 되었다면, 좀 더 정교한 모델을 만들 수 있었을 것이다. 그리고 우리 팀이 머신러닝을 배우면서 항상 들었던 의문이기도 하지만, 어떠한 상황에서 어떠한 통계적 모델이 효과적인지 어떻게 알 수 있을지 판단하는 방법을 알지 못해 알고 있는 모든 방법을 동원해야 했다. 다시 말해 우리가 알고 있는 모델을 바탕으로 brute force 방법을 쓴 것인데, 매우 좋지 않다고 생각한다. 그렇지만, 어떤 상황에서 의미있는 통계적인 모델이 있는지 알려주는 곳을 찾기가 어려웠다. 딥러닝 모델을 쓸 때(예를들어 DNN Classifier를 쓸 경우) 어떻게 node architecture를 세워야 의미있는 모델을 만들 수 있는지 알기가 어려웠다. 마치 데이터 셋에 통계적 모델을 끼워 맞추는 듯한 느낌이 들어 좀 더 모델링에 대한 공부를 많이 해야한다는 것을
느꼈다.

3. PCA

PCA와 Feature Engineering을 많이 하지 못한점이 아쉬웠다. 기존의 데이터들을 바탕으로 새로운 feature를 만들거나, 상관관계 분석 이외에도 해볼 수 있는 것들이 많았을 것이다. Numeric으로 변환된 데이터셋이 있다는 것에 안도해 해당 데이터셋을 쓸 생각만 했지, 실제로 그 데이터셋이 원본 데이터를 제대로 반영하고 있는지, One hot encoding이 되어 있다면 잘 되어 있는지, missing feature는 없는지 하나 하나 점검하지 못한점이 아쉬웠다. 데이터셋이 Categorical value이기 때문에 Outlier는 존재하지 않지만, 그렇지 않은 feature (예를 들어 age)의 분포를 좀 자세히 봤다면 더 의미있는 것이 있을 수 있지 않을까 하는 아쉬움이 있다. (예를 들어 생산가능인구 이외의 인구가 대출신청한 경우가 전체의 10%이내인데, 이 분포에서 의미있는 것인지 확인하고 싶었다)