Sklearn과 R 비교분석

김지범

Index

01

데이터

- 문제 정의
- 데이터 분석
- 데이터 전처리
- 데이터 시각화
- 데이터셋 생성

02

모델

- 모델 성능 평가
- 모델 예측

03

마무리

- 한계점
- 비교
- 느낀점

문제 정의

model	R	sklearn
LR	86%	?
Tree	87%	?
Boosting	88%	?
KNN(n=7)	82%	?
SVM	88%	?
LDA	88%	?
QDA	91%	?
RF	87%	?
BG	89%	?
NN	93%	?

같은 데이터셋과 같은 모델을 사용하여 R에서 했던 분류의 정확도와
Sklearn에서의 정확도를 비교해보면 어떤 결과가 나올까?

문제 정의

환자의 성별,나이,질병여부 등의 데이터로 조치 기간 중 사망했는지 예측

age : 환자의 연령
anaemia(빈혈) : 적혈구 또는 헤모글로빈 감소(혈소판)
high blood pressure : 고혈압인 경우
Creatinine phosphokinase : 혈액 내 CPK 효소 수치(mcg/L)
Diabetes : 당뇨병이 있는 경우
Ejection fraction : 매번 수축할 때마다 심장을 떠나는 혈액 비율(백분율)
Platelets : 혈중 혈소판(kiloplate/mL)
Sex : 여자 또는 남자
Serum creatinine : 혈청 크레아티닌 수치(mg/dL)
Serum sodium : 혈청나트륨 수치(mEq/L)
Smoking : 흡연 여부
Time : 조치 기간 (일)
[target] death event : 환자가 조치 기간 중 사망한 경우(사망)

데이터 분석

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	anaemia	creatinine_	diabetes	ejection_fr	high_bloo	platelets	serum_cre	serum_soc	sex	smoking	time	DEATH_EVENT	
2	75	0	582	0	20	1	265000	1.9	130	1	0	4	1	
3	55	0	7861	0	38	0	263358	1.1	136	1	0	6	1	
4	65	0	146	0	20	0	162000	1.3	129	1	1	7	1	
5	50	1	111	0	20	0	210000	1.9	137	1	0	7	1	
6	65	1	160	1	20	0	327000	2.7	116	0	0	8	1	
7	90	1	47	0	40	1	204000	2.1	132	1	1	8	1	
8	75	1	246	0	15	0	127000	1.2	137	1	0	10	1	
9	60	1	315	1	60	0	454000	1.1	131	1	1	10	1	
10	65	0	157	0	65	0	263358	1.5	138	0	0	10	1	
11	80	1	123	0	35	1	388000	9.4	133	1	1	10	1	
12	75	1	81	0	38	1	368000	4	131	1	1	10	1	
13	62	0	231	0	25	1	253000	0.9	140	1	1	10	1	
14	45	1	981	0	30	0	136000	1.1	137	1	0	11	1	
15	50	1	168	0	38	1	276000	1.1	137	1	0	11	1	

```
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       300 non-null    object
 1   1       300 non-null    object
 2   2       300 non-null    object
 3   3       300 non-null    object
 4   4       300 non-null    object
 5   5       300 non-null    object
 6   6       300 non-null    object
 7   7       300 non-null    object
 8   8       300 non-null    object
 9   9       300 non-null    object
 10  10      300 non-null    object
 11  11      300 non-null    object
 12  12      300 non-null    object
dtypes: object(13)
```

기본적으로 300개의 행과 13개의 columns, object 데이터 타입 형태

컬럼명이 1행에 담겨있으므로 299명의 환자 데이터로 구성

데이터 전처리

	age	anamia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
0	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
1	75	0	582	0	20	1
2	55	0	7861	0	38	0
3	65	0	146	0	20	0
4	50	1	111	0	20	0

```
1 # column 추가로 중복된 첫 번째행 삭제
2 df= df.drop(0,0)
3 df.head()
```

	age	anamia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
1	75	0	582	0	20	1
2	55	0	7861	0	38	0
3	65	0	146	0	20	0
4	50	1	111	0	20	0
5	65	1	160	1	20	0

컬럼 추가로 중복된 첫 번째행 삭제

데이터 전처리

```
2 df['age'].unique()
```

```
array(['75', '55', '65', '50', '90', '60', '80', '62', '45', '49', '82',
       '87', '70', '48', '68', '53', '95', '58', '94', '85', '69', '72',
       '51', '57', '42', '41', '67', '79', '59', '44', '63', '86', '66',
       '43', '46', '61', '81', '52', '64', '40', '60.667', '73', '77',
       '78', '54', '47', '56'], dtype=object)
```

```
1 # 60.667를 np.nan으로 변경
2 df['age'].replace('60.667',np.nan,inplace=True)
3 df.info
```

```
1 # 데이터안 nan값 60으로 변경
2 df=df.fillna(60)
```

```
1 # 변경 사항 확인
2 df['age'].unique()
```

```
array(['75', '55', '65', '50', '90', '60', '80', '62', '45', '49', '82',
       '87', '70', '48', '68', '53', '95', '58', '94', '85', '69', '72',
       '51', '57', '42', '41', '67', '79', '59', '44', '63', '86', '66',
       '43', '46', '61', '81', '52', '64', '40', 60, '73', '77', '78',
       '54', '47', '56'], dtype=object)
```

데이터 확인 중 발견한 잘못된 데이터 수정

Float -> int

60.667 -> nan -> 60

데이터 전처리

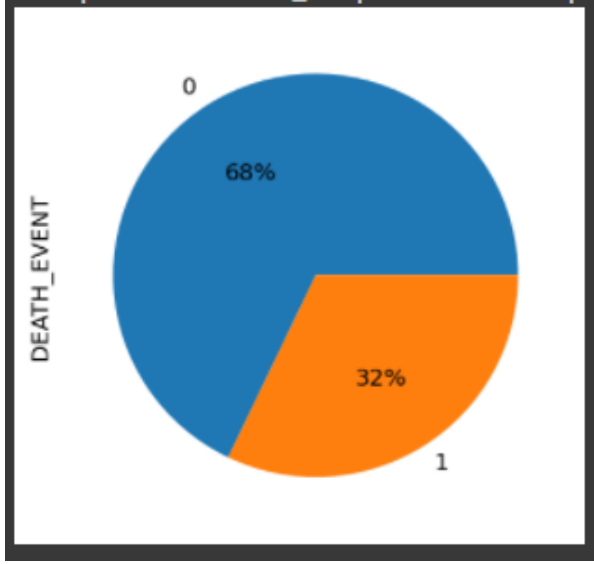
```
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    int64
 1   anamia                    299 non-null    category
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    category
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    category
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    category
 10  smoking                   299 non-null    category
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: category(5), float64(2), int64(6)
```

범주형과 수치형에 따라 데이터 타입 변경

데이터 시각화

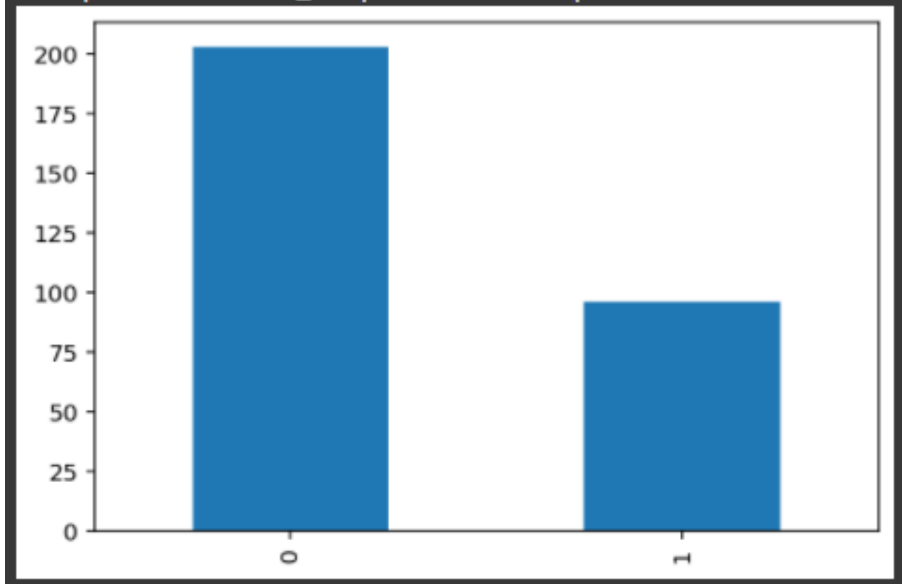
```
1 # 사망자-생존자 환자 비율 (0: 사망, 1: 생존)
2 df.DEATH_EVENT.value_counts().plot.pie(autopct = '%1.f%%')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8420d8b50>
```



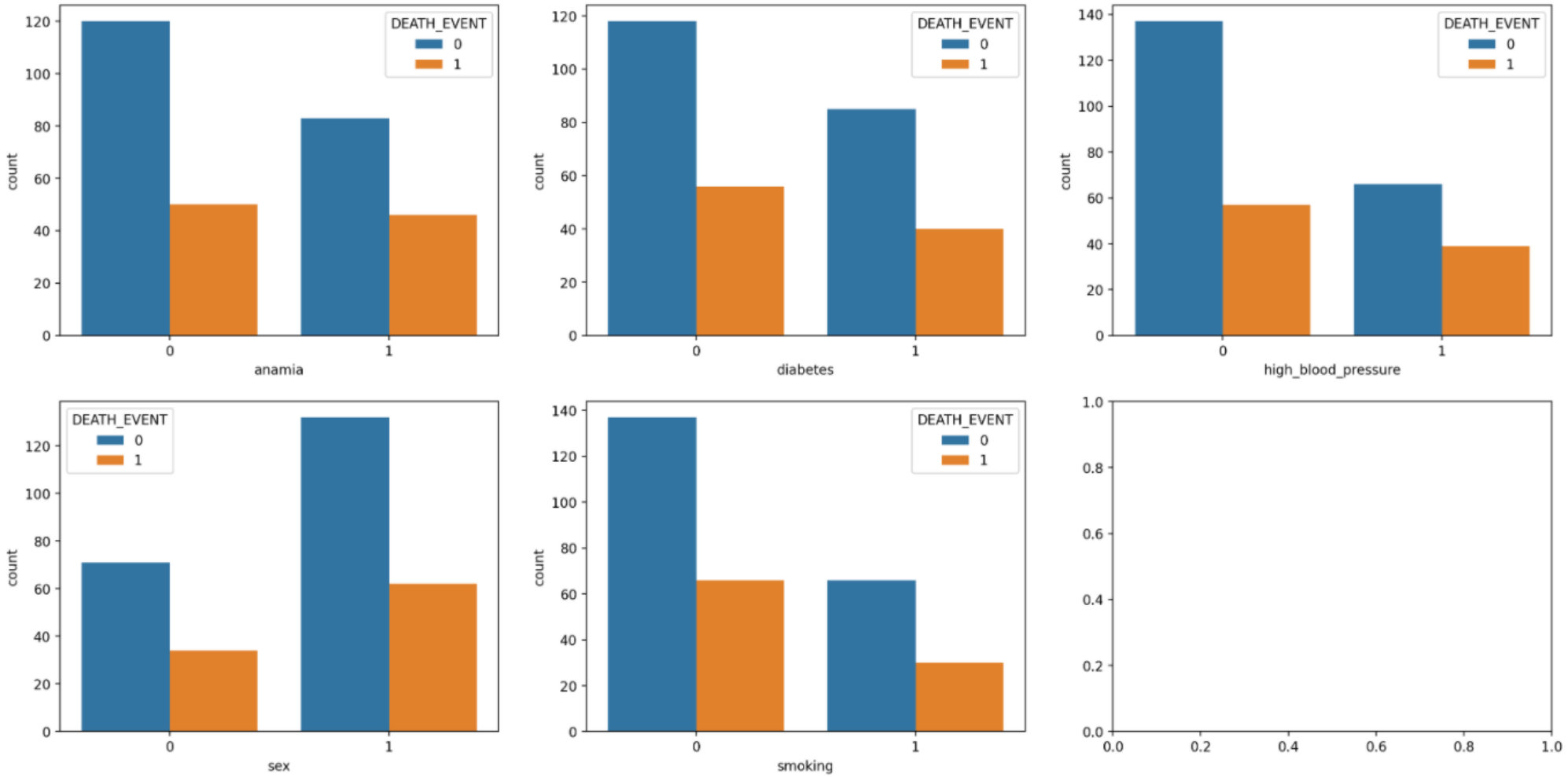

```
2 df.DEATH_EVENT.value_counts().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe8420c3150>
```



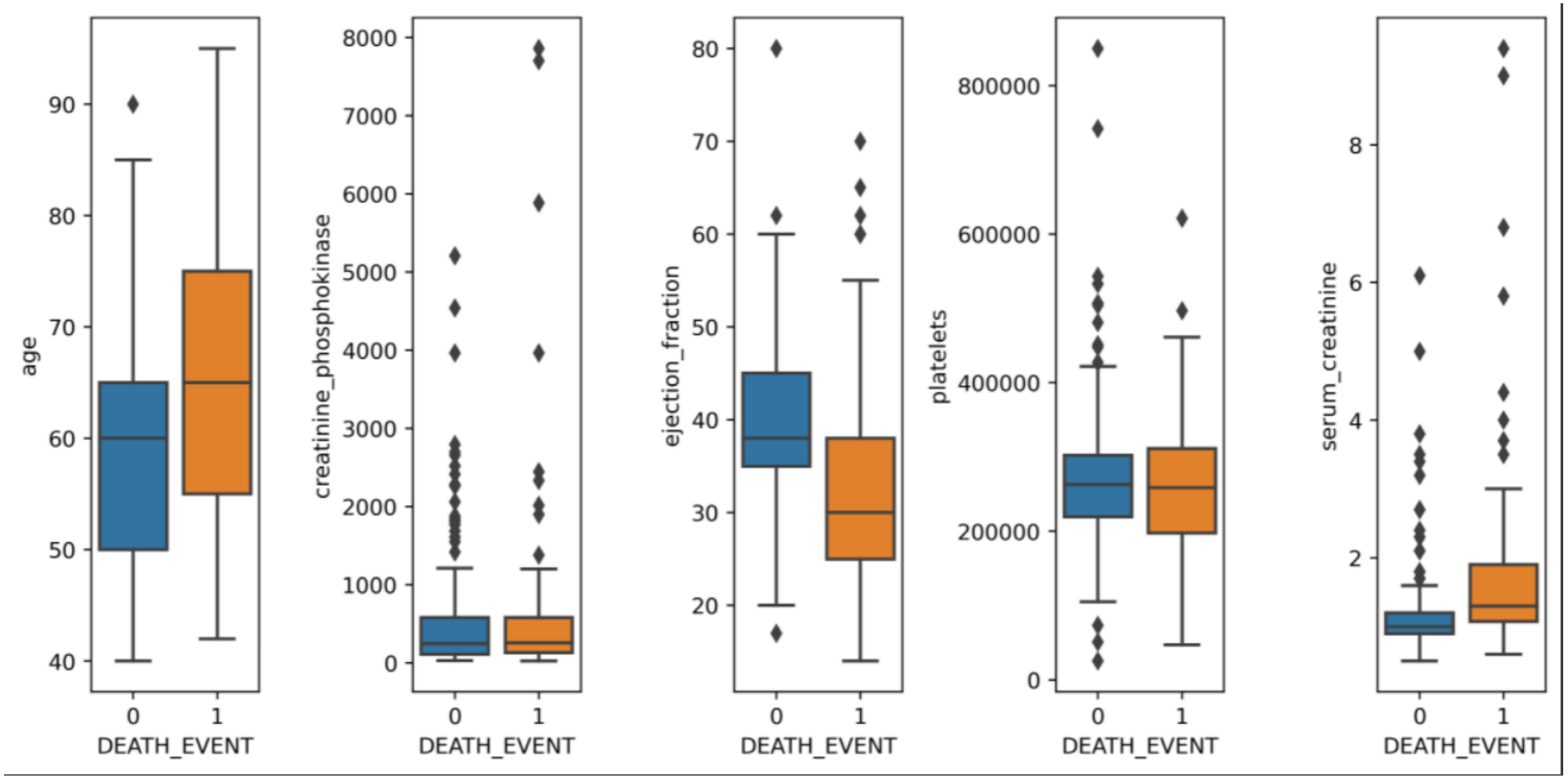
사망자 비율이 두배 이상 높음

데이터 시각화



범주형 데이터들의 그래프로 고혈압 그래프는 다른 그래프와 다르게 사망여부와 비례하지 않고 고혈압이 있는 환자의 사망률이 생각보다 높지 않다. (질병여부 1: True 0: False)

데이터 시각화



이상치가 많지만 데이터 수 부족으로 삭제 하지는 않음
(수치형 데이터는 환자의 혈청 수치등이므로 이상치라고 하기 불분명)

데이터셋 생성

```
# 원-핫 인코딩 처리
# 컬럼 -> 원-핫 인코딩 처리
temp = pd.get_dummies(X[categorical_var], drop_first=True)
```

```
# 기존 데이터랑 합치기
X_modified = pd.concat([X,temp], axis=1)
```

```
# 기존 컬럼 삭제
X_modified.drop(categorical_var, axis=1, inplace=True)
```

```
X_modified.head()
```

범주형 데이터를 원-핫 인코딩

차원 축소 -> 스케일링

```
# 수치형 데이터 정규화
scaler = StandardScaler()
X_train[numeric_var] = scaler.fit_transform(X_train[numeric_var])
X_test[numeric_var] = scaler.fit_transform(X_test[numeric_var])
```

age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium	time
1.548096	-0.045062	-0.302229	0.806502	0.521163	-0.568626	0.574896
1.304090	-0.517003	0.915622	1.353224	-0.318062	0.096539	1.021468
-0.810629	0.751152	-1.114130	0.035234	-0.485907	-1.455513	-1.160354
-1.542647	-0.561902	0.103721	-0.765323	-0.653752	0.761704	1.480799
0.978748	-0.045062	-1.520081	-0.039374	0.294572	-0.568626	0.881117

수치형 데이터 정규화

데이터셋 생성

훈련용, 테스트 데이터 분리하기

```
[ ]   1 X_train, X_test, y_train, y_test = train_test_split(X_modified,y,
      2                                                     train_size=0.7, # test_size
      3                                                     random_state=7)
```

훈련용 데이터 : 70% (210개)
테스트 데이터 : 30% (89개)

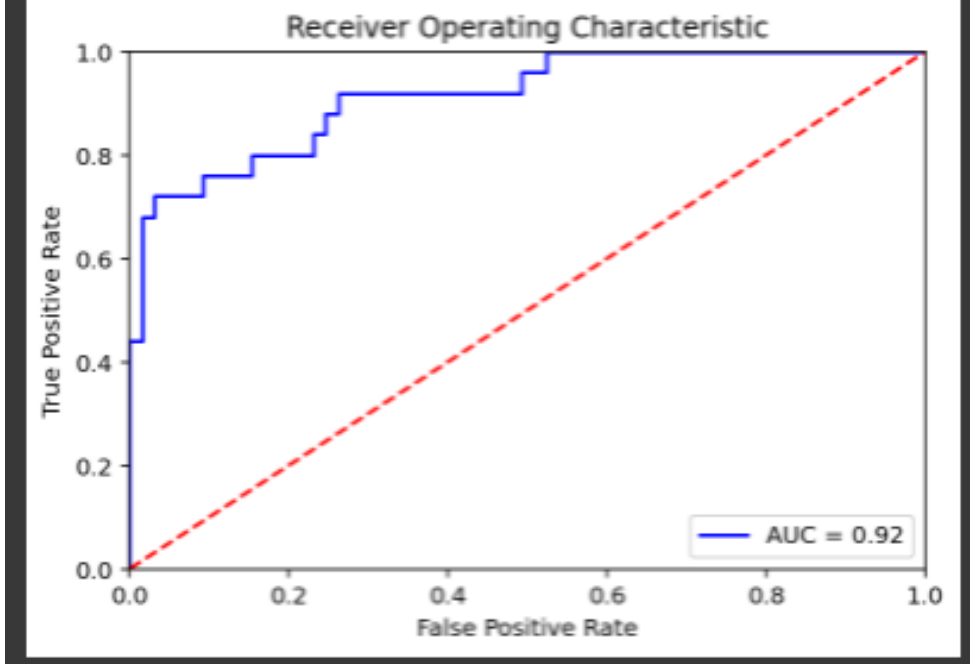
모델 성능 평가 -1) Logistic Regression

```python
# Logistic Regression 평가 지표
lg_y_hat = log_reg.predict(X_test)
lg_report = metrics.classification_report(y_test, lg_y_hat)
print('Logistic Regression 평가 지표')
print(lg_report)
print('accuracy', metrics.accuracy_score(y_test,lg_y_hat) )
print('precision', metrics.precision_score(y_test,lg_y_hat) )
print('recall', metrics.recall_score(y_test,lg_y_hat) )
print('f1', metrics.f1_score(y_test,lg_y_hat) )
```

```
Logistic Regression 평가 지표
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        65
           1       0.78      0.72      0.75        25

    accuracy                           0.87        90
   macro avg       0.84      0.82      0.83        90
weighted avg       0.86      0.87      0.86        90

accuracy 0.8666666666666667
precision 0.782608695652174
recall 0.72
f1 0.7499999999999999
```

```python
# Logistic Regression ROC
plot_auc_roc(log_reg)
```

```
findfont: Font family ['NanumBarunGothic'] not found. Fallin
```

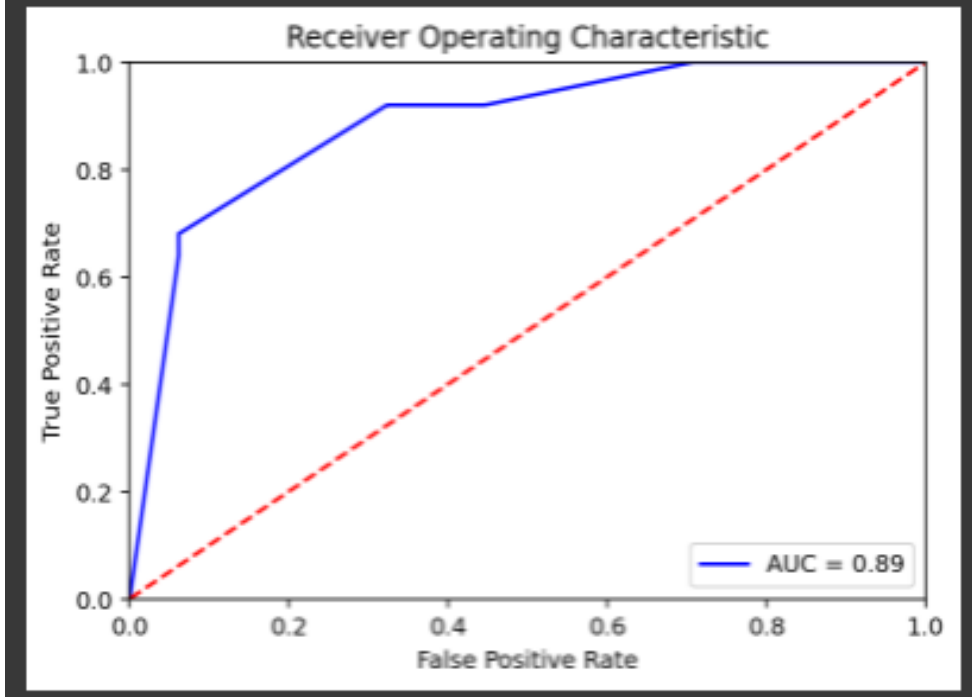


모델 성능 평가 -2) Decision Tree

```python
# tree 평가 지표
tree_y_hat = tree.predict(X_test)
tree_report = metrics.classification_report(y_test, tree_y_hat)
print('tree 평가 지표')
print(tree_report)
print('accuracy', metrics.accuracy_score(y_test,tree_y_hat) )
print('precision', metrics.precision_score(y_test,tree_y_hat) )
print('recall', metrics.recall_score(y_test,tree_y_hat) )
print('f1', metrics.f1_score(y_test,tree_y_hat) )
```

```
tree 평가 지표
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        65
           1       0.81      0.68      0.74        25

    accuracy                           0.87        90
   macro avg       0.85      0.81      0.82        90
weighted avg       0.86      0.87      0.86        90

accuracy 0.8666666666666667
precision 0.8095238095238095
recall 0.68
f1 0.7391304347826089
```

```python
# Tree
plot_auc_roc(tree)
```

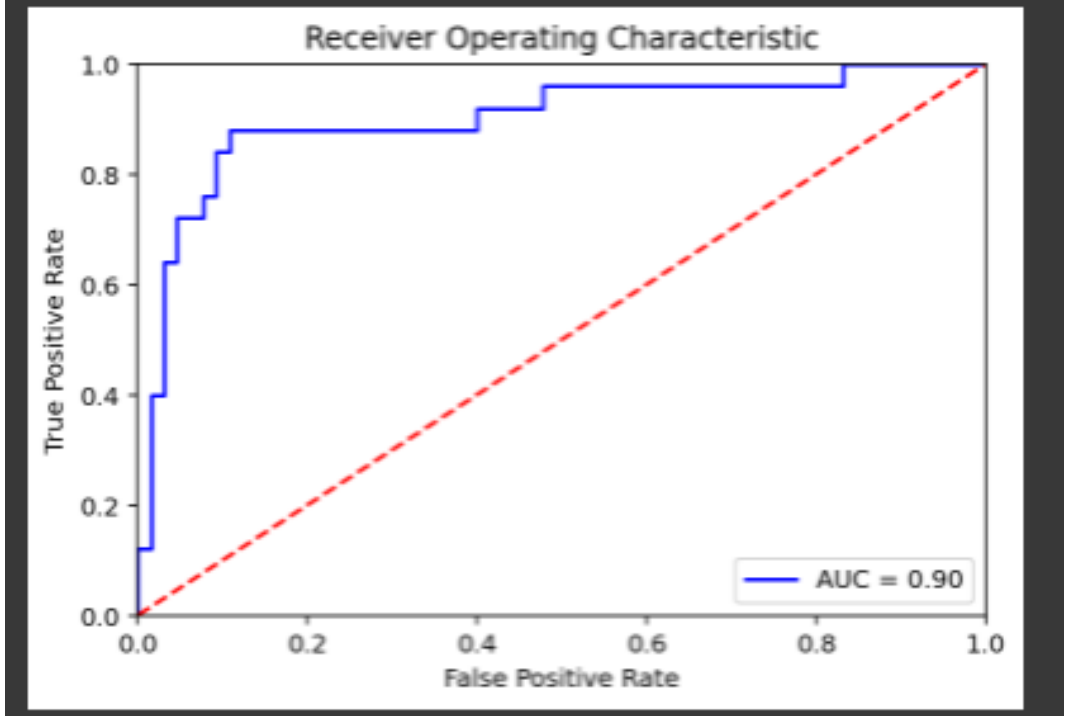


모델 성능 평가 -3) Gradient Boosting

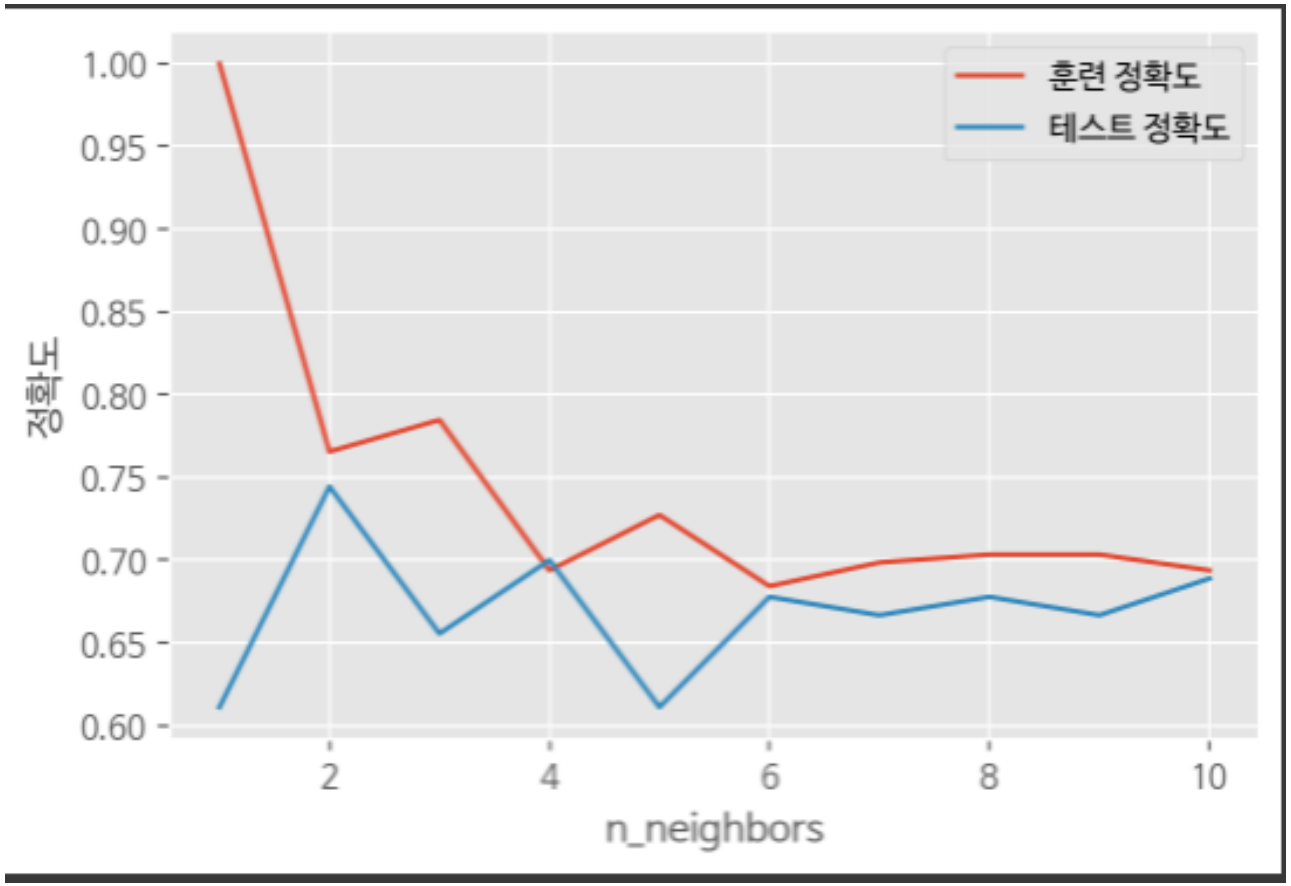
```python
# boost 평가 지표
boost_y_hat = boost.predict(X_test)
boost_report = metrics.classification_report(y_test, boost_y_hat)
print('boost 평가 지표')
print(boost_report)
print('accuracy', metrics.accuracy_score(y_test,boost_y_hat) )
print('precision', metrics.precision_score(y_test,boost_y_hat) )
print('recall', metrics.recall_score(y_test,boost_y_hat) )
print('f1', metrics.f1_score(y_test,boost_y_hat) )
```

```
boost 평가 지표
              precision    recall  f1-score   support

           0       0.94      0.89      0.91        65
           1       0.75      0.84      0.79        25

    accuracy                           0.88        90
   macro avg       0.84      0.87      0.85        90
weighted avg       0.88      0.88      0.88        90

accuracy 0.8777777777777778
precision 0.75
recall 0.84
f1 0.7924528301886793
```

```python
# GradientBoostingClassifier
plot_auc_roc(boost)
```



모델 성능 평가 -4) KNN (n=7)



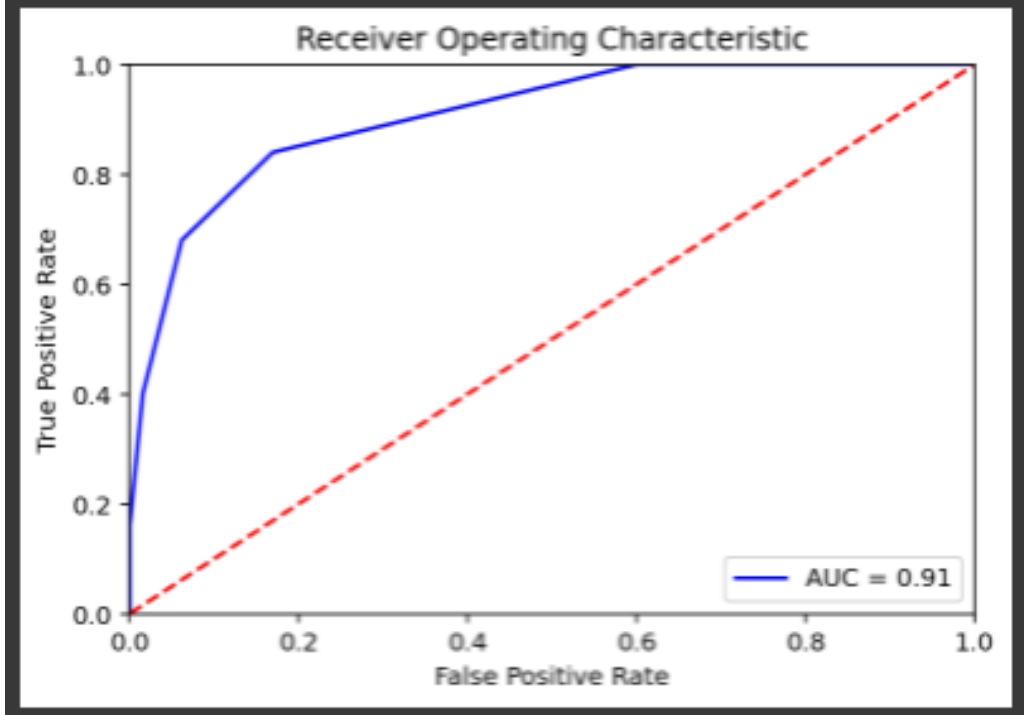
테스트 정확도(홀수)중 가장 높은 정확도를 가진 7을 선택

모델 성능 평가 -4) KNN (n=7)

```python
# KNN 평가 지표
KNN_y_hat = KNN.predict(X_test)
KNN_report = metrics.classification_report(y_test, KNN_y_hat)
print('KNN 평가 지표')
print(KNN_report)
print('accuracy', metrics.accuracy_score(y_test,KNN_y_hat) )
print('precision', metrics.precision_score(y_test,KNN_y_hat) )
print('recall', metrics.recall_score(y_test,KNN_y_hat) )
print('f1', metrics.f1_score(y_test,KNN_y_hat) )
```

```
KNN 평가 지표
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        65
           1       0.81      0.68      0.74        25

    accuracy                           0.87        90
   macro avg       0.85      0.81      0.82        90
weighted avg       0.86      0.87      0.86        90

accuracy 0.8666666666666667
precision 0.8095238095238095
recall 0.68
f1 0.7391304347826089
```

```python
# KNeighborsClassifier
plot_auc_roc(KNN)
```

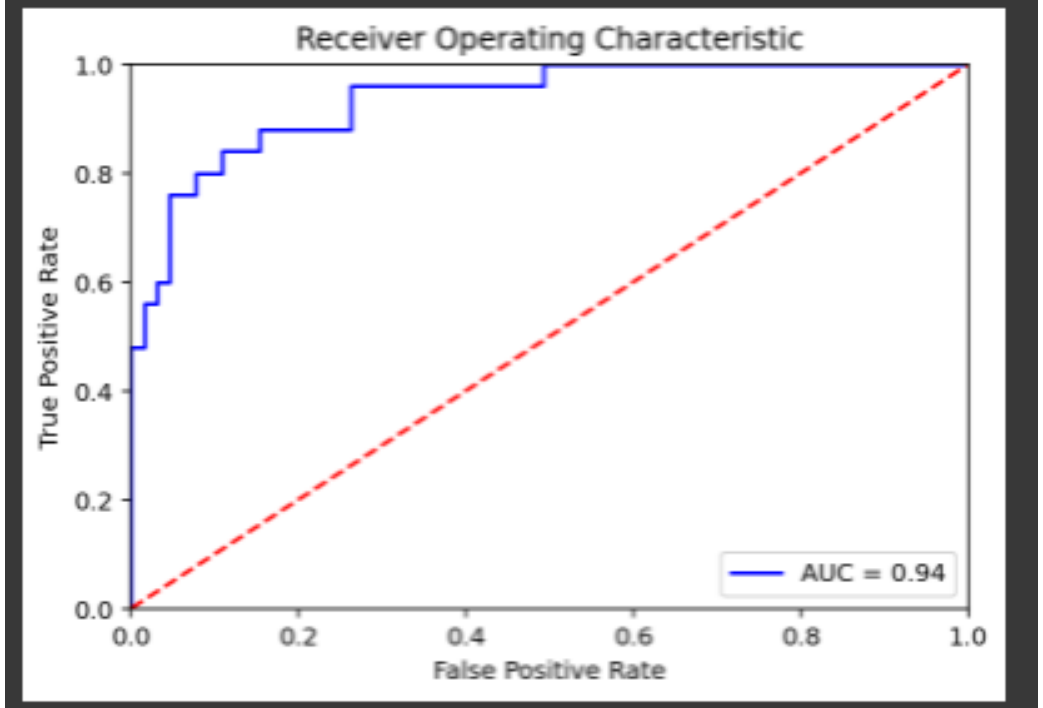


모델 성능 평가 -5) SVM

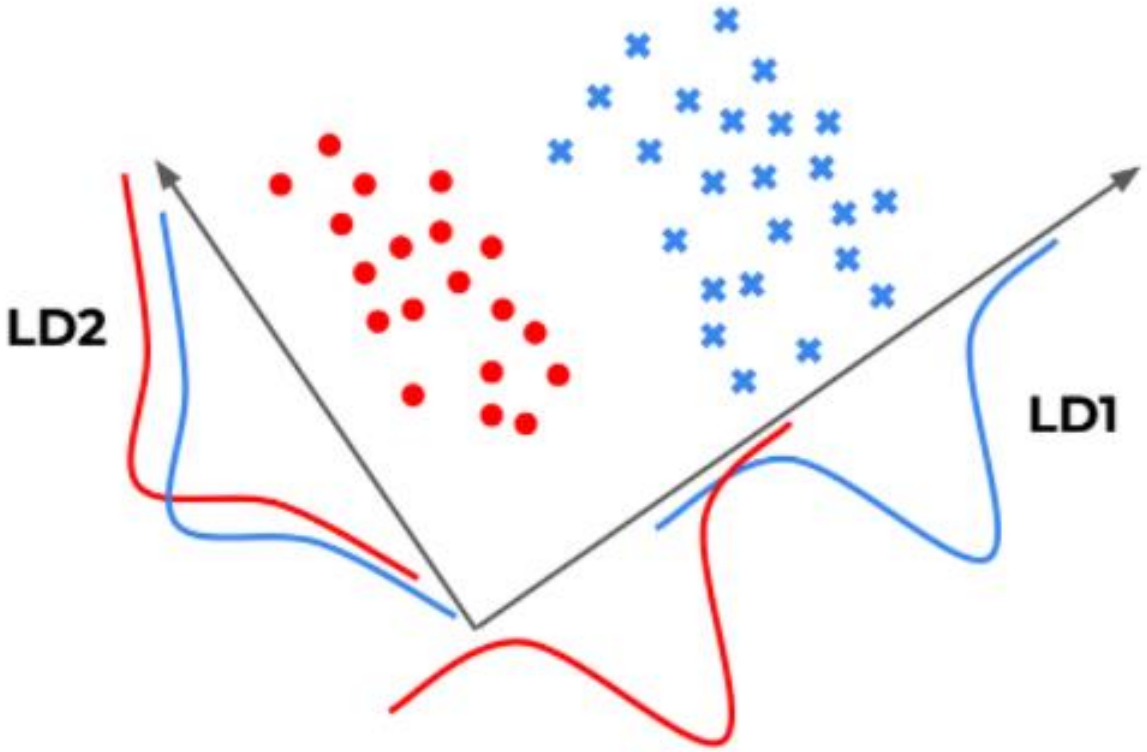
```python
# SVM 평가 지표
svm_y_hat = svm.predict(X_test)
svm_report = metrics.classification_report(y_test, svm_y_hat)
print('svm 평가 지표')
print(svm_report)
print('accuracy', metrics.accuracy_score(y_test,svm_y_hat) )
print('precision', metrics.precision_score(y_test,svm_y_hat) )
print('recall', metrics.recall_score(y_test,svm_y_hat) )
print('f1', metrics.f1_score(y_test,svm_y_hat) )
```

```
svm 평가 지표
              precision    recall  f1-score   support

           0       0.91      0.92      0.92        65
           1       0.79      0.76      0.78        25

    accuracy                           0.88        90
   macro avg       0.85      0.84      0.85        90
weighted avg       0.88      0.88      0.88        90

accuracy 0.8777777777777778
precision 0.7916666666666666
recall 0.76
f1 0.7755102040816326
```

```python
# Support Vector Machine
plot_auc_roc(svm)
```



모델 -6) LDA(선형판별분석)



데이터를 특정 한 축에 사영(projection)한 후에 두 범주(빨간색, 파란색)을 잘 구분할 수 있는 직선을 찾는 게 목표

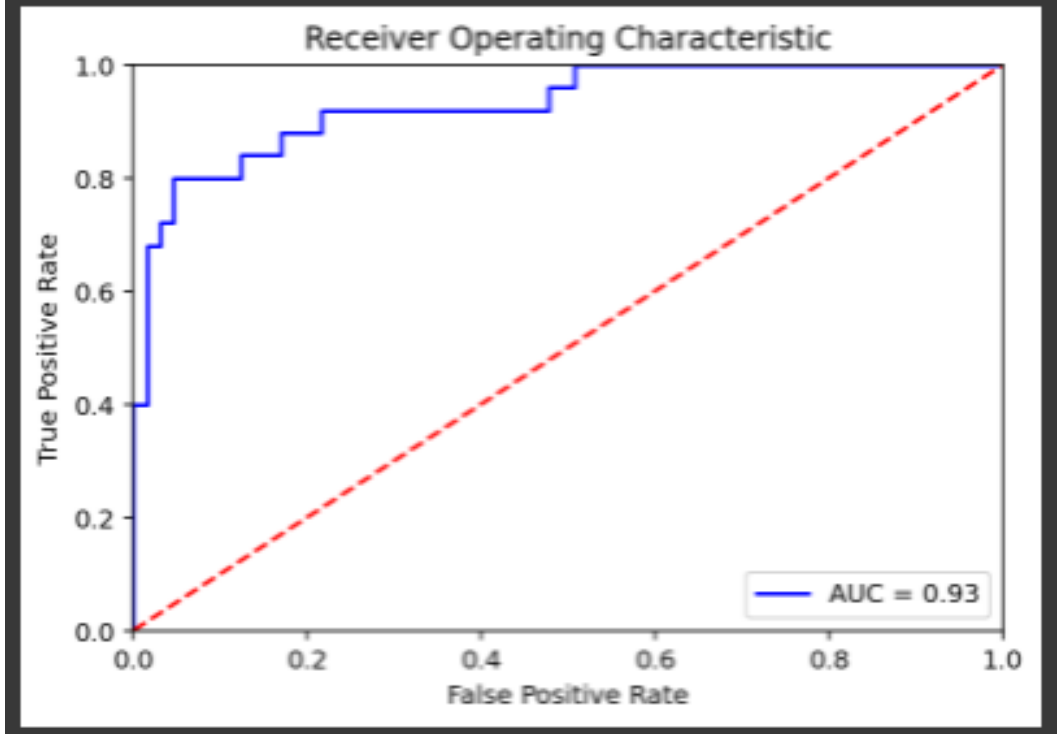
그림의 LD2보다는 LD1을 찾아 독립변수 x를 이용해 분류를 예측하는 모델

모델 성능 평가 -6) LDA(선형판별분석)

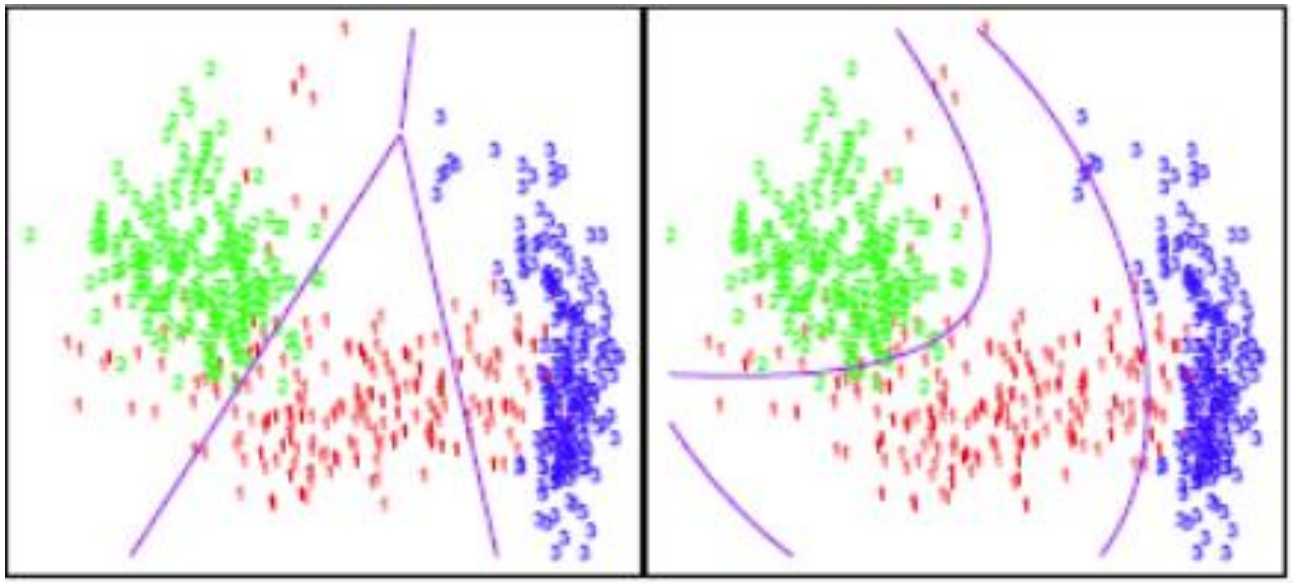
```python
# LDA 평가 지표
lda_y_hat = lda.predict(X_test)
lda_report = metrics.classification_report(y_test, lda_y_hat)
print('lda 평가 지표')
print(lda_report)
print('accuracy', metrics.accuracy_score(y_test,lda_y_hat) )
print('precision', metrics.precision_score(y_test,lda_y_hat) )
print('recall', metrics.recall_score(y_test,lda_y_hat) )
print('f1', metrics.f1_score(y_test,lda_y_hat) )
```

```
lda 평가 지표
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        65
           1       0.80      0.80      0.80        25

    accuracy                           0.89        90
   macro avg       0.86      0.86      0.86        90
weighted avg       0.89      0.89      0.89        90

accuracy 0.8888888888888888
precision 0.8
recall 0.8
f1 0.8000000000000002
```

```python
# Linear Discriminant Analysis
plot_auc_roc(lda)
```



모델 -7) QDA (이차판별분석)



LDA QDA

2차식으로 구성

장점
- 비선형 분류가 가능

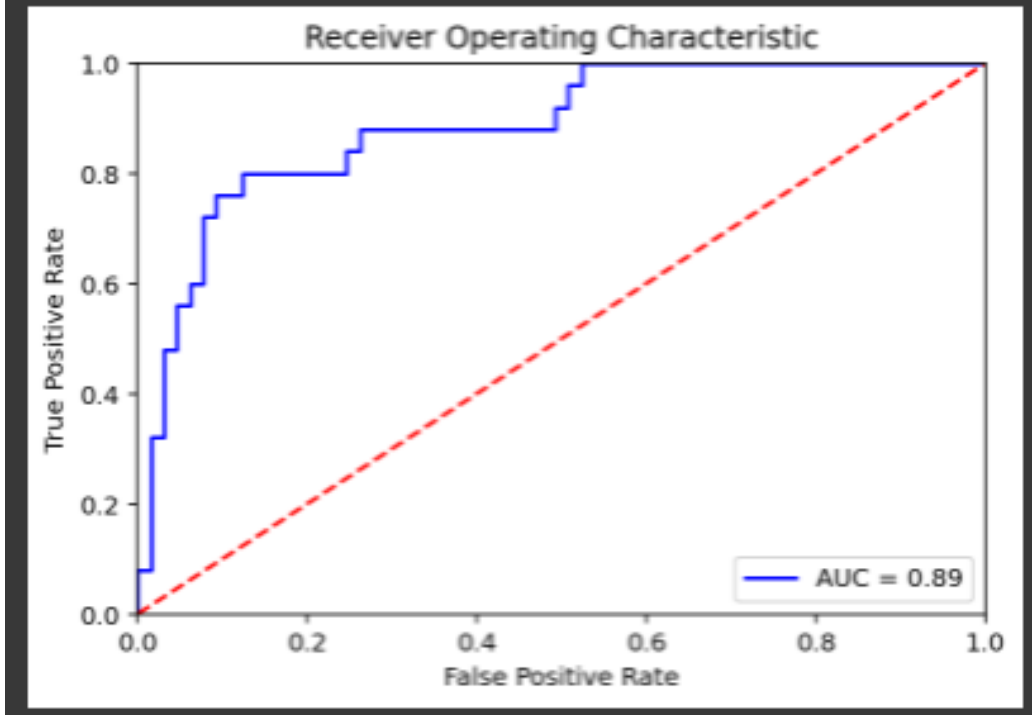
단점
- 변수의 개수가 많을 경우, 추정해야하는 모수가 많아짐 → 연산량이 큼

모델 성능 평가 -7) QDA (이차판별분석)

```python
# QDA 평가 지표
qda_y_hat = qda.predict(X_test)
qda_report = metrics.classification_report(y_test, qda_y_hat)
print('qda 평가 지표')
print(qda_report)
print('accuracy', metrics.accuracy_score(y_test,qda_y_hat) )
print('precision', metrics.precision_score(y_test,qda_y_hat) )
print('recall', metrics.recall_score(y_test,qda_y_hat) )
print('f1', metrics.f1_score(y_test,qda_y_hat) )
```

```
qda 평가 지표
              precision    recall  f1-score   support

           0       0.88      0.92      0.90        65
           1       0.77      0.68      0.72        25

    accuracy                           0.86        90
   macro avg       0.83      0.80      0.81        90
weighted avg       0.85      0.86      0.85        90

accuracy 0.8555555555555555
precision 0.7727272727272727
recall 0.68
f1 0.7234042553191491
```

```python
# Quadratic Discriminant Analysis
plot_auc_roc(qda)
```

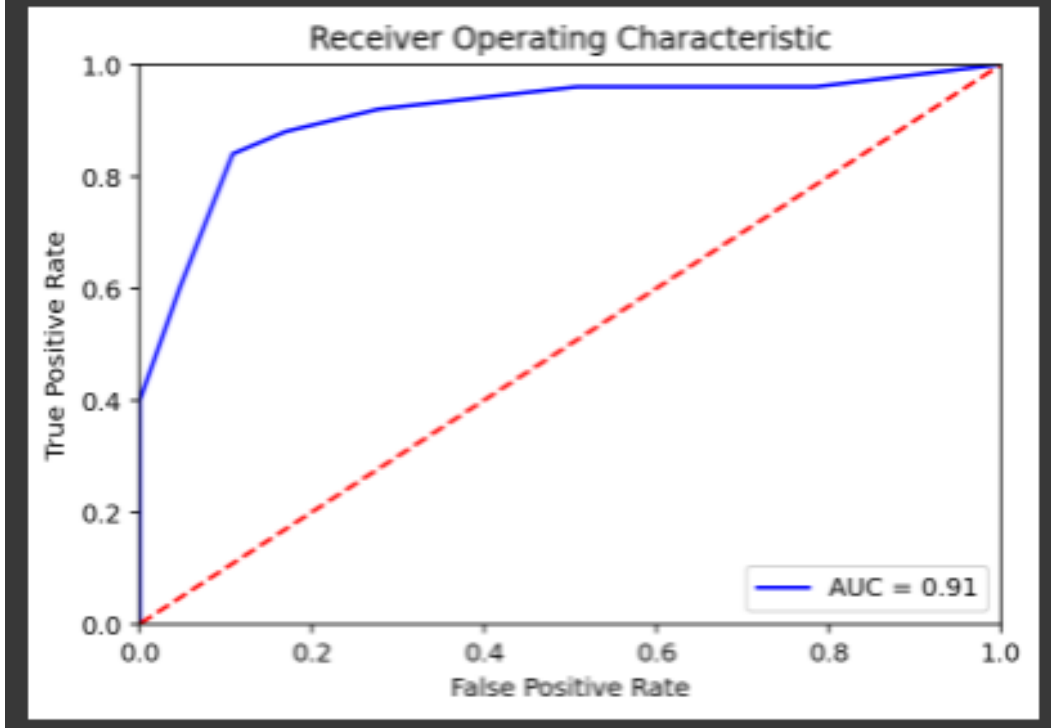


모델 성능 평가 -8) Random Forest

```python
# RF 평가 지표
rf_y_hat = rf.predict(X_test)
rf_report = metrics.classification_report(y_test, rf_y_hat)
print('rf 평가 지표')
print(rf_report)
print('accuracy', metrics.accuracy_score(y_test,rf_y_hat) )
print('precision', metrics.precision_score(y_test,rf_y_hat) )
print('recall', metrics.recall_score(y_test,rf_y_hat) )
print('f1', metrics.f1_score(y_test,rf_y_hat) )
```

```
rf 평가 지표
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        65
           1       0.78      0.72      0.75        25

    accuracy                           0.87        90
   macro avg       0.84      0.82      0.83        90
weighted avg       0.86      0.87      0.86        90

accuracy 0.8666666666666667
precision 0.782608695652174
recall 0.72
f1 0.7499999999999999
```

```python
# RandomForest
plot_auc_roc(rf)
```



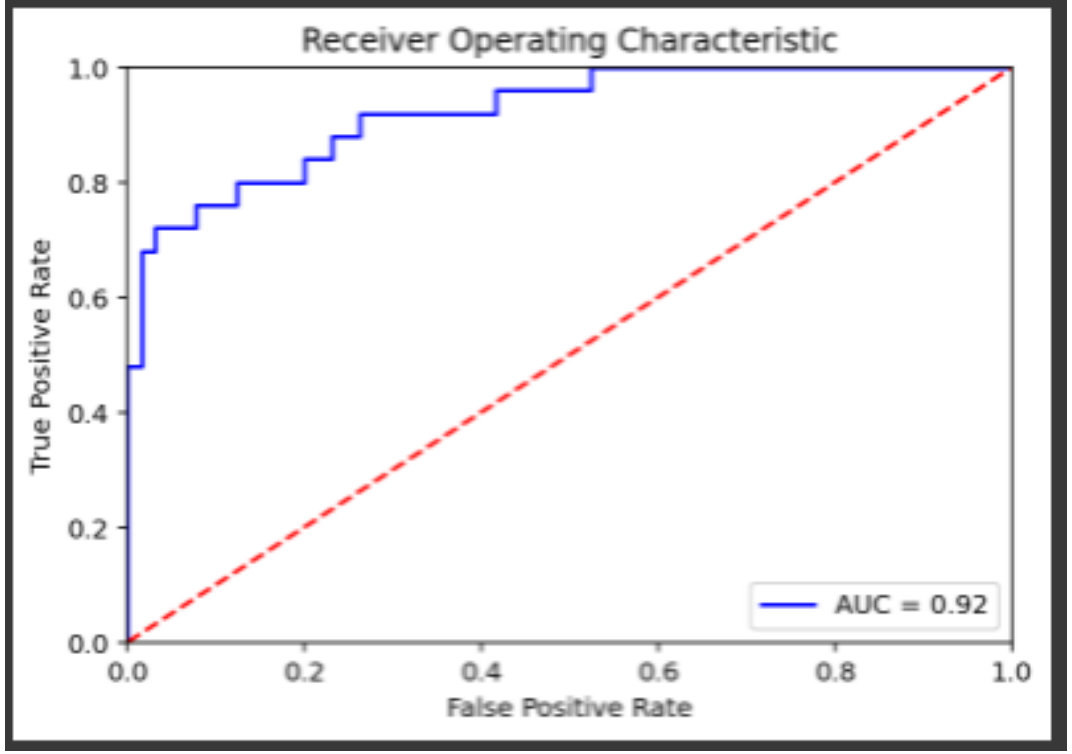
트리 100개로 구성된 랜덤 포레스트

모델 성능 평가 -9) Bagging

```python
# BAG 평가 지표
bag_y_hat = bag.predict(X_test)
bag_report = metrics.classification_report(y_test, bag_y_hat)
print('bag 평가 지표')
print(bag_report)
print('accuracy', metrics.accuracy_score(y_test,bag_y_hat) )
print('precision', metrics.precision_score(y_test,bag_y_hat) )
print('recall', metrics.recall_score(y_test,bag_y_hat) )
print('f1', metrics.f1_score(y_test,bag_y_hat) )
```

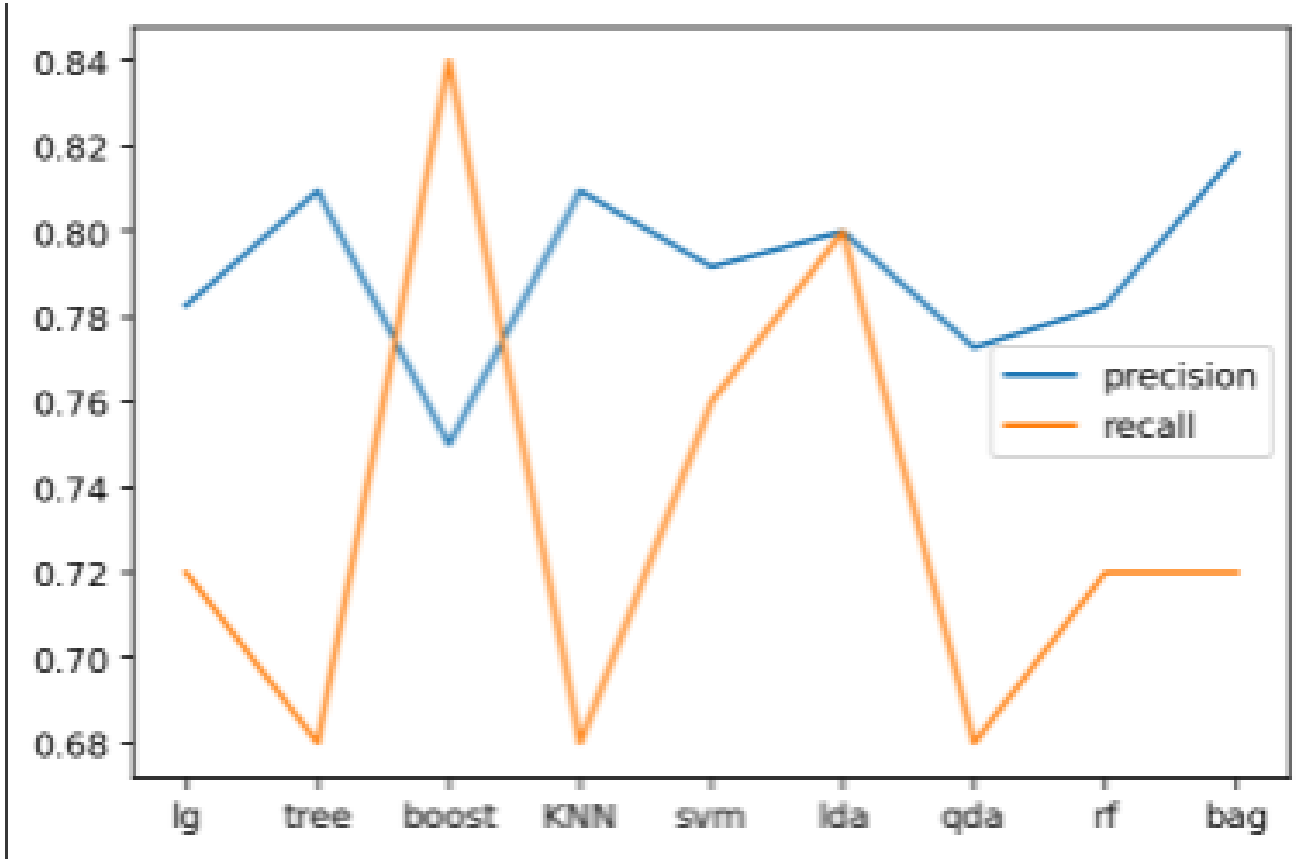
```
bag 평가 지표
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        65
           1       0.82      0.72      0.77        25

    accuracy                           0.88        90
   macro avg       0.86      0.83      0.84        90
weighted avg       0.88      0.88      0.88        90

accuracy 0.8777777777777778
precision 0.8181818181818182
recall 0.72
f1 0.7659574468085107
```

```python
# Bagging
plot_auc_roc(bag)
```



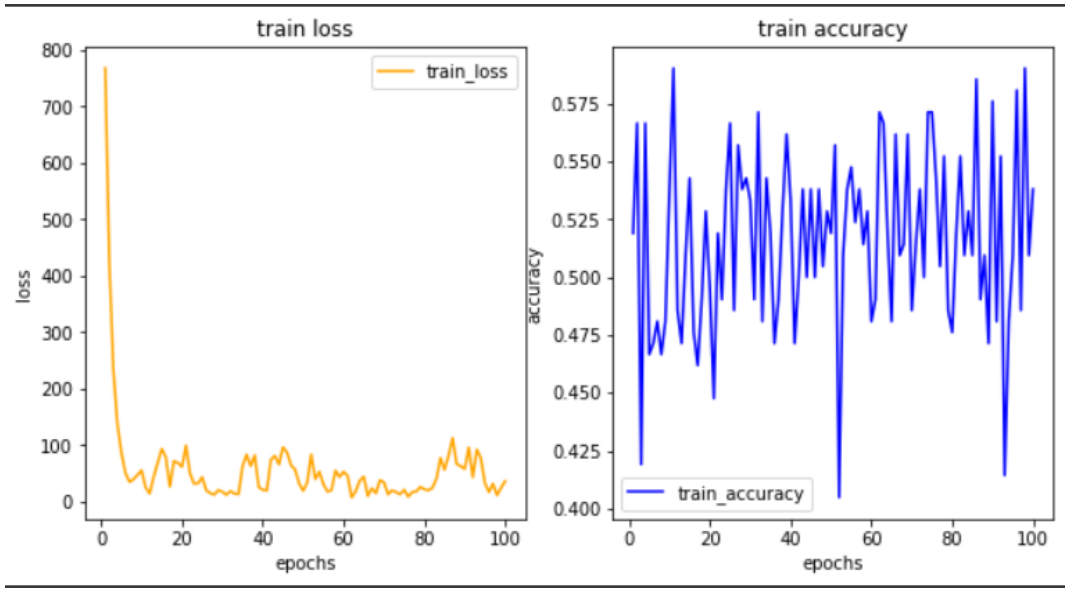
로지스틱 회귀 모델을 100개 훈련하여 앙상블한 배깅

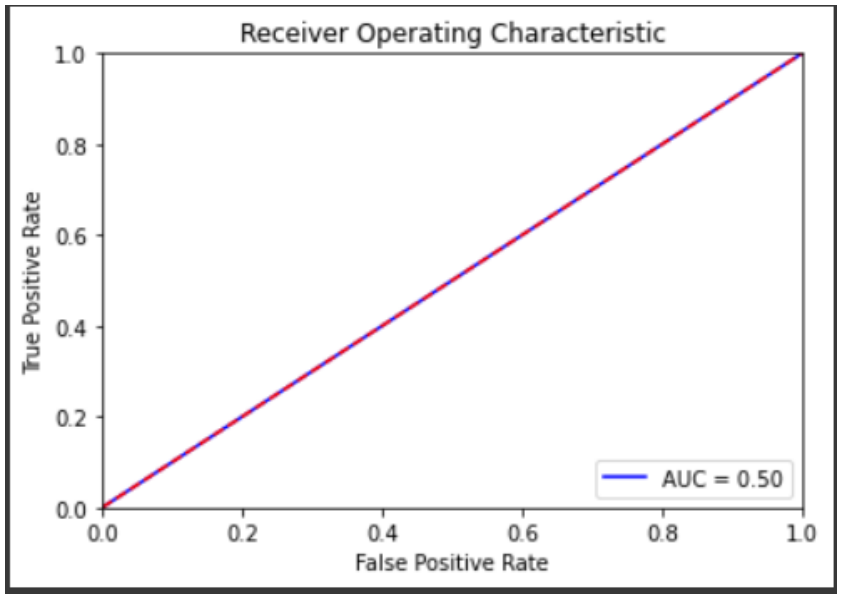
모델 예측



타깃이 사망 여부이므로 recall를 기준으로 보면 boosting모델이 효과적

한계점





	precision	recall	f1-score	support
0.0	0.92	1.00	0.96	82
1.0	0.00	0.00	0.00	7
accuracy			0.92	89
macro avg	0.46	0.50	0.48	89
weighted avg	0.85	0.92	0.88	89

MLP 모델이 다른 데이터에서는 결과가 잘 나오지만
개인 데이터에서는 결과가 일정하지 않음
Model을 수정하거나 tersorflow의 feature_column을
사용해도 결과가 일정하지 않음
타깃의 비율 맞춰줘도 결과가 일정하지 않음
-> 데이터 수, 데이터 타입의 영향으로 예상

비교

model	R	sklearn
LR	86%	86%
Tree	87%	86%
Boosting	88%	87%
KNN(n=7)	82%	86%
SVM	88%	87%
LDA	88%	88%
QDA	91%	85%
RF	87%	86%
BG	89%	87%
NN	93%	nan

비슷한 정확도를 보이지만 R에서의 정확도가 미세하게 높은 것을 확인

느낀점

- R로 분류를 했던 당시에 비해 데이터 사이언스에 대한 지식은 많이 늘어 데이터 전처리 과정을 더 세세하게 조정하여 분류하여 더 좋은 결과를 낼 거라고 예상하고 진행하였지만 R에 비해 아직 익숙하지 않은 sklearn은 모델을 조정하는 부분이 익숙치 못해 결과가 더 좋지 못했을 거라고 추측

- MLP 모델에서 제대로 결과를 도출하지 못한 아쉬움이 매우 커서 수정 중에 찾아봤던 방법들을 딥러닝 공부를 통해 해결해 나갈 예정

Thank You