인공지능보안
기말 프로젝트

10팀
인공지능학부 20223175 배세은, 20223201 임혜진

Index

01 데이터 셋 설명

1. **Pima Indians diabetes Database : Pima Indian 혈통의 21세 이상인 여성의 데이터**
   - 당뇨병과 관련이 높은 feature들로 이루어진 데이터 셋
   - feature 대부분이 연속적 실수 값을 가지는 변수이기 때문에 이상 탐지에 알맞은 데이터셋이라고 판단

2. **당뇨병은 신체적 지표와 직접적으로 관련이 있는 질병**
   - 다양한 feature들을 통해 당뇨병의 종합적 원인을 탐지하고 분석해보기 위함

01 Goal

당뇨병 데이터셋 => **의료데이터**
의료데이터에서 중요한 지표 : **Recall**
우리 데이터셋에서 recall이 나타내는 것
당뇨병인데 정상으로 판정하는 것 -> **위험**한 상황
거짓양성으로 판단하는 비율이 높더라도 당뇨병을 놓치는 것이 없도록 하는 것이 중요!

01 Upsampling

1. 기존 데이터셋 비율로는 이상탐지 task가 불가능하다고 판단
-> **정상 데이터셋을 업샘플링 시켜 당뇨병 데이터가 이상치로 판단될 만큼의 비율로 억지로 만듦**

2. 업샘플링 시키면 정상 데이터가 **복사**됨
-> 2개의 feature에 평균 0, 표준편차 0.01인 **노이즈 추가**

500:268 -> 5591:268

```
원래 클래스 분포:
 Outcome
0    500
1    268
Name: count, dtype: int64
업샘플링 후 데이터 크기: (5859, 8)
업샘플링 후 클래스 분포:
 Outcome
0    5591
1     268
Name: count, dtype: int64
```

```python
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

X = diabetes_data.drop('Outcome', axis=1)
y = diabetes_data['Outcome']
print('원래 클래스 분포:\n', y.value_counts())

# 정상데이터 비율을 95%로 업샘플링해서 이상치 탐지 태스크를 수행할 수 있게 함
ratio = 0.95
curr_ratio = y.value_counts(normalize=True)[0]

# 업샘플링 해야 하는 비율 계산
sampling_strategy_value = (ratio * (1 - curr_ratio)) / (curr_ratio * (1 - ratio))
sampling_strategy = {0: int(y.value_counts()[0] * (1 + sampling_strategy_value))}

# RandomOverSampler 객체 생성
ros = RandomOverSampler(sampling_strategy=sampling_strategy, random_state=42)

# 오버샘플링 적용
X_resampled, y_resampled = ros.fit_resample(X, y)
y_resampled_series = pd.Series(y_resampled)

# X_resampled을 pandas DataFrame으로 변환
X_resampled_df = pd.DataFrame(X_resampled, columns=X.columns)

# 노이즈 추가
# noise = np.random.normal(loc=0, scale=0.01, size=X_resampled.shape)
# X_resampled = X_resampled + noise

# BMI와 DiabetesPedigreeFunction 열에 노이즈 추가
noise_bmi = np.random.normal(loc=0, scale=0.01, size=X_resampled.shape[0])
noise_dpf = np.random.normal(loc=0, scale=0.01, size=X_resampled.shape[0])

X_resampled_df['BMI'] += noise_bmi
X_resampled_df['DiabetesPedigreeFunction'] += noise_dpf

# 결과 출력
print("업샘플링 후 데이터 크기:", X_resampled_df.shape)
print("업샘플링 후 클래스 분포:\n", y_resampled_series.value_counts())
```

01 Scaling

```
#데이터 표준화
scaler = StandardScaler()
scaler.fit(X_resampled_df)
df_scaled = scaler.transform(X_resampled_df)
df_scaled = pd.DataFrame(df_scaled)
x_df_scaled = df_scaled.copy()
df_scaled['target'] = y_resampled_series
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.594516	0.635887	50
1	1	85	66	29	0	26.589290	0.353829	31
2	8	183	64	0	0	23.299810	0.656766	32
3	1	89	66	23	94	28.092085	0.165441	21
4	0	137	40	35	168	43.102573	2.291617	33
...	...	...	...	...	...	...	...	...
5854	0	67	76	0	0	45.296447	0.206386	46
5855	2	84	50	23	76	30.402283	0.956763	21
5856	1	91	54	25	100	25.193243	0.249070	23
5857	1	143	86	30	330	30.102953	0.889361	23
5858	3	84	72	32	0	37.196561	0.283054	28

→

	0	1	2	3	4	5	6	7
0	0.878776	1.347385	0.191504	1.008811	-0.690661	0.391390	0.665672	1.593833
1	-0.772522	-0.980475	-0.133061	0.607274	-0.690661	-0.501722	-0.272016	-0.029753
2	1.539295	2.640640	-0.241249	-1.333488	-0.690661	-0.921105	0.735084	0.055699
3	-0.772522	-0.832674	-0.133061	0.205737	0.232080	-0.310127	-0.898302	-0.884272
4	-1.102782	0.940933	-1.539507	1.008811	0.958493	1.603593	6.170071	0.141151
...	...	...	...	...	...	...	...	...
5854	-1.102782	-1.645578	0.407880	-1.333488	-0.690661	1.883294	-0.762185	1.252026
5855	-0.442263	-1.017425	-0.998566	0.205737	0.055385	-0.015595	1.732409	-0.884272
5856	-0.772522	-0.758774	-0.782190	0.339583	0.290978	-0.679707	-0.620282	-0.713368
5857	-0.772522	1.162634	0.948820	0.674197	2.548748	-0.053757	1.508335	-0.713368
5858	-0.112003	-1.017425	0.191504	0.808042	-0.690661	0.850622	-0.507302	-0.286109

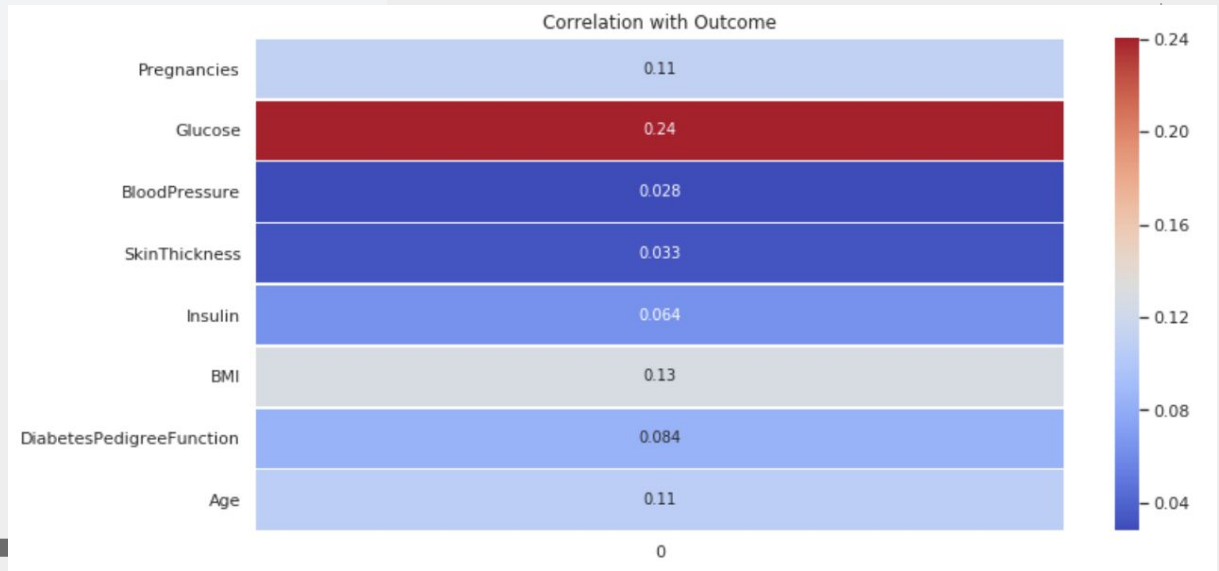
02 Feature Importance 분석

- 상관계수
- Logistic Regression
- Random Forest

02 Feature Importance 분석- 상관계수

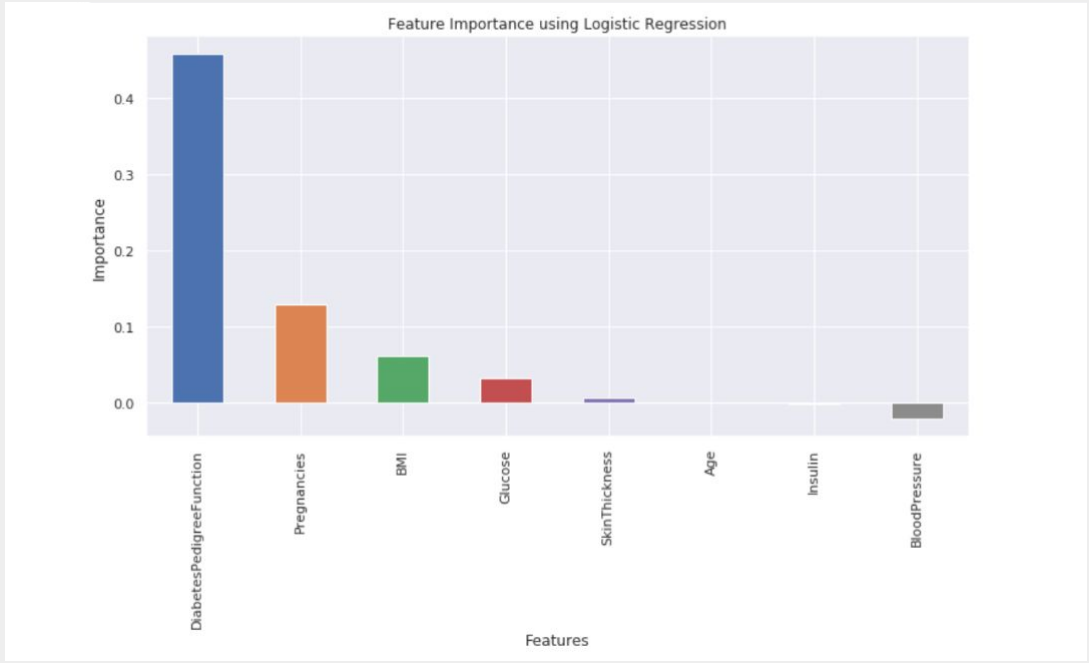
```python
# 상관계수 분석

#각 특성과 라벨 간의 상관계수 계산
correlation_matrix = X_resampled_df.corrwith(y_resampled_series)
print(correlation_matrix)

#시각화
plt.figure(figsize=(12, 6))
sns.heatmap(correlation_matrix.to_frame(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation with Outcome')
plt.show()
```

```
Pregnancies                 0.110379
Glucose                     0.240449
BloodPressure               0.028007
SkinThickness               0.032797
Insulin                     0.064428
BMI                         0.128907
DiabetesPedigreeFunction    0.084196
Age                         0.106995
dtype: float64
```



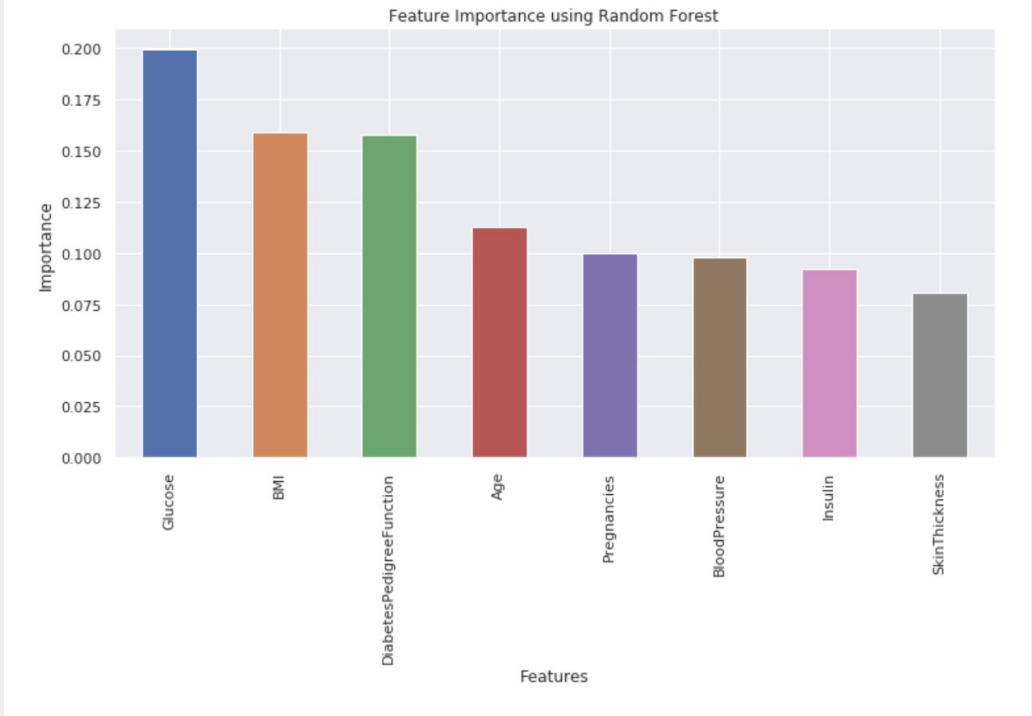
02 Feature Importance 분석- Logistic Regression

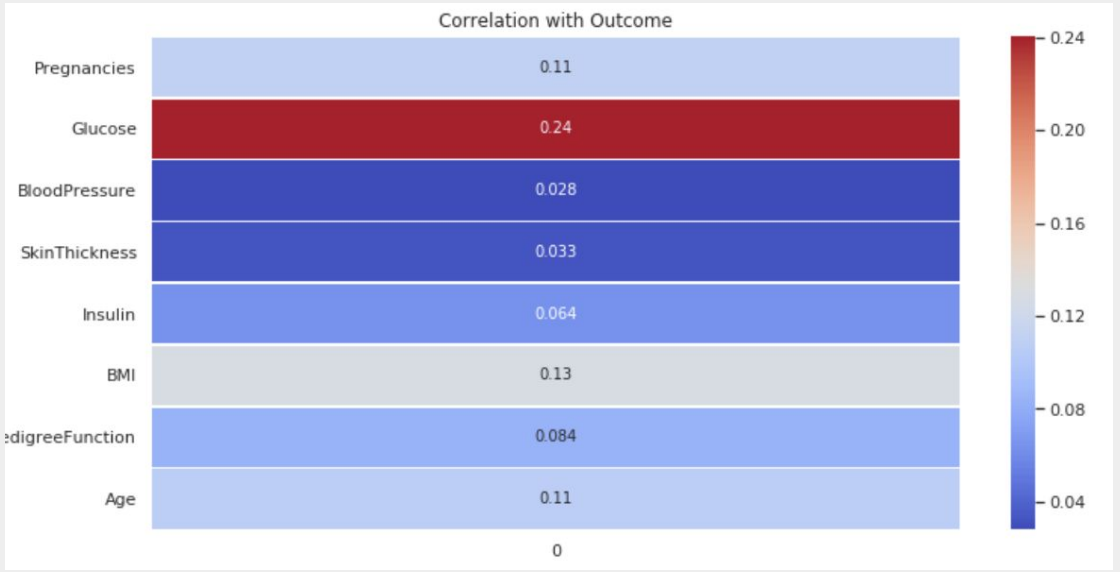
```python
# 로지스틱 회귀를 이용한 피처 중요도 분석

#max_iter으로 모델의 최대 반복 횟수를 설정
log_reg = LogisticRegression(max_iter=10000)

#로지스틱 회귀 모델 훈련
log_reg.fit(X_resampled, y_resampled)

#회귀 계수를 통해 특성 중요도 계산
importance = log_reg.coef_[0] #훈련된 모델의 계수 가져옴
feature_importance = pd.Series(importance, index=X.columns).sort_values(ascending=False)
#각 계수를 특성 이름과 매칭 시켜 Series로 변환

#특성 중요도 시각화
plt.figure(figsize=(12, 6)) #그림의 크기 설정
feature_importance.plot(kind='bar') #bar차트로 특성 중요도 시각화
plt.title('Feature Importance using Logistic Regression')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```

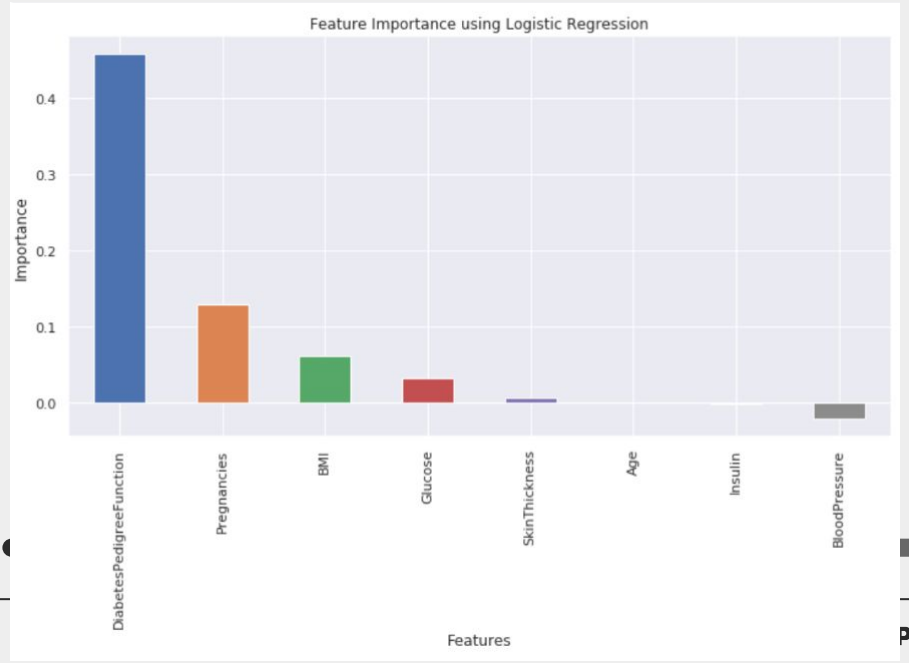


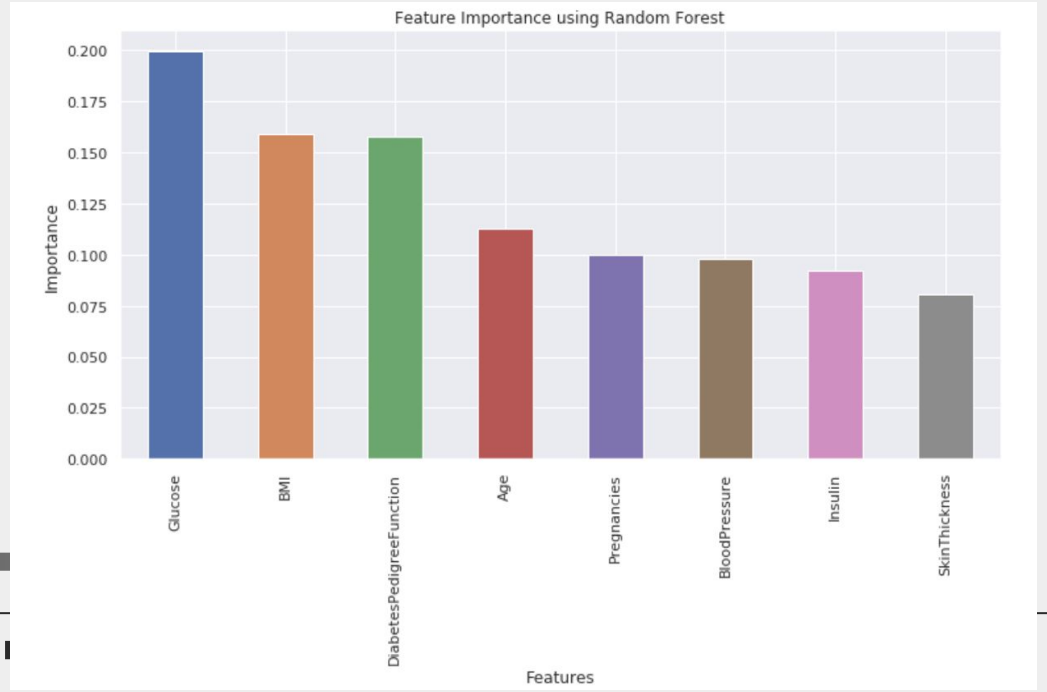
02 Feature Importance 분석- Random Forest

```python
# 랜덤 포레스트를 이용한 피처 중요도

#랜덤 포레스트 모델 초기화
rf_model = RandomForestClassifier(random_state=42) #재현성을 위해 random_state = 42로 설정

#랜덤 포레스트 모델 훈련
rf_model.fit(X_resampled, y_resampled)

#특성 중요도 계산
importance = rf_model.feature_importances_

#특성 중요도를 Series로 변환하고 정렬
feature_importance = pd.Series(importance, index=X.columns).sort_values(ascending=False)

#특성 중요도 시각화
plt.figure(figsize=(12, 6))
feature_importance.plot(kind='bar')
plt.title('Feature Importance using Random Forest')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.show()
```




Correlation with Outcome
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
edigreeFunction
Age
0.11
0.24
0.028
0.033
0.064
0.13
0.084
0.11
0
0.24
0.20
0.16
0.12
0.08
0.04


Feature Importance using Logistic Regression
Importance
0.4
0.3
0.2
0.1
0.0
DiabetesPedigreeFunction
Pregnancies
BMI
Glucose
SkinThickness
Age
Insulin
BloodPressure
Features


Feature Importance using Random Forest
Importance
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Glucose
BMI
DiabetesPedigreeFunction
Age
Pregnancies
BloodPressure
Insulin
SkinThickness
Features

03 이상탐지

- Isolation Forest
- OneClass SVM
- Random Forest
- EllipticEnvelop (가우시안)
- AutoEncoder

03-1 Isolation Forest

```python
#01
from sklearn.ensemble import IsolationForest
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Glucose 및 DiabetesPedigreeFunction 특성만 선택
X_train_subset = X_train[['Glucose', 'DiabetesPedigreeFunction']]
X_test_subset = X_test[['Glucose', 'DiabetesPedigreeFunction']]

# Isolation Forest 모델 훈련
iso_forest = IsolationForest(contamination=0.095, random_state=42)
iso_forest.fit(X_train_subset)

# 이상치 예측
y_pred_test = iso_forest.predict(X_test_subset)

# 이진 분류에서는 이상치를 양성 클래스(1)로 설정
y_true_binary = (y_test == 1)
y_pred_binary = (y_pred_test == -1)

# 혼동 행렬 생성
cm = confusion_matrix(y_true_binary, y_pred_binary)

print("Confusion Matrix:")
print(cm)

# 성능 지표 계산
accuracy = accuracy_score(y_true_binary, y_pred_binary)
precision = precision_score(y_true_binary, y_pred_binary)
recall = recall_score(y_true_binary, y_pred_binary)
f1 = f1_score(y_true_binary, y_pred_binary)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Confusion Matrix:
[[1008  105]
 [  37   22]]
Accuracy: 0.878839590443686
Precision: 0.1732283464566929
Recall: 0.3728813559322034
F1 Score: 0.2365591397849462
```

03-2 One Class SVM

```python
# 특성 선택
X_train_subset = X_train[['Glucose', 'DiabetesPedigreeFunction']]
X_test_subset = X_test[['Glucose', 'DiabetesPedigreeFunction']]

# One-Class SVM 모델 정의
oc_svm = OneClassSVM()

# 랜덤 서치를 위한 하이퍼파라미터 그리드 설정
param_dist = {
    'nu': [0.01, 0.05, 0.1, 0.2, 0.5],  # nu의 후보값들
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # 커널 종류
    'gamma': ['scale', 'auto', 0.01, 0.1, 1, 10]  # gamma의 후보값들
}

# 성능 평가를 위한 사용자 정의 스코어 함수 정의
def custom_scorer(y_true, y_pred):
    # 이진 분류에서는 이상치를 양성 클래스(1)로 설정
    y_true_binary = (y_true == 1)
    y_pred_binary = (y_pred == -1)

    return f1_score(y_true_binary, y_pred_binary)

scorer = make_scorer(custom_scorer, greater_is_better=True)

# 랜덤 서치 객체 생성
random_search = RandomizedSearchCV(oc_svm,
                                   param_distributions=param_dist,
                                   n_iter=50, scoring=scorer,
                                   refit=True, cv=5, random_state=42)

# 랜덤 서치 모델 훈련
random_search.fit(X_train_subset, y_train)
```

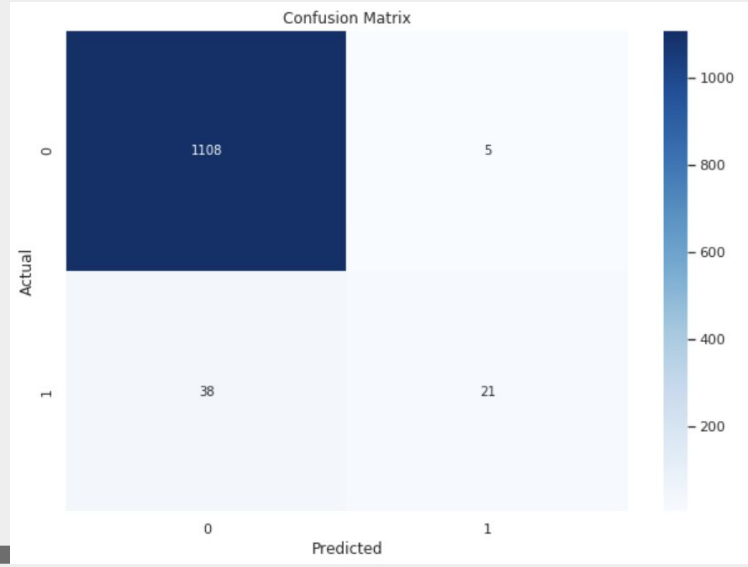
```
Best Parameters:  {'nu': 0.05, 'kernel': 'rbf', 'gamma': 1}
Confusion Matrix:
[[1042   71]
 [  31   28]]
Accuracy: 0.9129692832764505
Precision: 0.2828282828282828
Recall: 0.4745762711864407
F1 Score: 0.35443037974683544
```

03-3 Random Forest

```python
# Glucose,DiabetesPedigreeFunction 특성 선택
X_train_RFC = X_train[['Glucose', 'DiabetesPedigreeFunction']]
X_test_RFC = X_test[['Glucose', 'DiabetesPedigreeFunction']]

# 랜덤 포레스트 모델 정의
rf_model = RandomForestClassifier(random_state=42)

# 랜덤 서치를 위한 하이퍼파라미터 범위 설정
param_dist = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False]
}

# 랜덤 서치 객체 생성
random_search = RandomizedSearchCV(estimator=rf_model,
                                   param_distributions=param_dist,
                                   n_iter=100, scoring='accuracy',
                                   cv=5, random_state=42,
                                   n_jobs=-1, verbose=2)

# 랜덤 서치 모델 훈련
random_search.fit(X_train_RFC, y_train)

# 최적의 파라미터 출력
print("Best Parameters: ", random_search.best_params_)

# 최적의 모델로 테스트 데이터에 대한 예측 수행
best_rf_model = random_search.best_estimator_
y_pred_test_RFC = best_rf_model.predict(X_test_RFC)
```

Best Parameters: {
'n_estimators': 50,
'min_samples_split': 2,
'min_samples_leaf': 2,
'max_features': 'sqrt',
'max_depth': 40, 'bootstrap': False}

```
Accuracy: 0.9633105802047781
Precision: 0.8872667472143911
Recall: 0.6757199202034507
F1 Score: 0.7375413379163086
```



03-4 EllipticEnvelop (가우시안)

```python
from sklearn.covariance import EllipticEnvelope

# Glucose feature만 학습에 사용
X_train = X_train[[1]]
X_test = X_test[[1]]

# Elliptic Envelope 모델 초기화
elliptic_env = EllipticEnvelope(contamination=0.04)  # contamination은 이상치 비율

# Elliptic Envelope 모델을 학습
elliptic_env.fit(X_train)

# 예측 (1: 정상치, -1: 이상치)
predictions = elliptic_env.predict(X_test)

# 예측 결과를 0 (정상치)와 1 (이상치)로 변환
predictions = np.where(predictions == 1, 0, 1)

# 예측 결과를 새로운 열로 추가
elliptic_X_test = X_test.copy()
elliptic_X_test['anomaly'] = predictions
```

```
Confusion Matrix:
[[1070   43]
 [  43   16]]
Accuracy: 0.9266211604095563
Precision: 0.2711864406779661
Recall: 0.2711864406779661
F1 Score: 0.2711864406779661
```

03 - 5 Autoencoder

모델구조

```
import tensorflow as tf
from tensorflow.keras import layers, models

# 모델 설계
input_dim = X_train.shape[1]  # 입력 차원 (feature 수)
encoding_dim = 15  # 압축할 차원 (예: 4)

# 모델 구조
input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="relu",
                activity_regularizer=regularizers.l1(1e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
decoder = Dense(int(encoding_dim / 2), activation='relu')(encoder)
decoder = Dense(input_dim, activation='linear')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, 8)	0
dense_28 (Dense)	(None, 15)	135
dense_29 (Dense)	(None, 7)	112
dense_30 (Dense)	(None, 7)	56
dense_31 (Dense)	(None, 8)	64

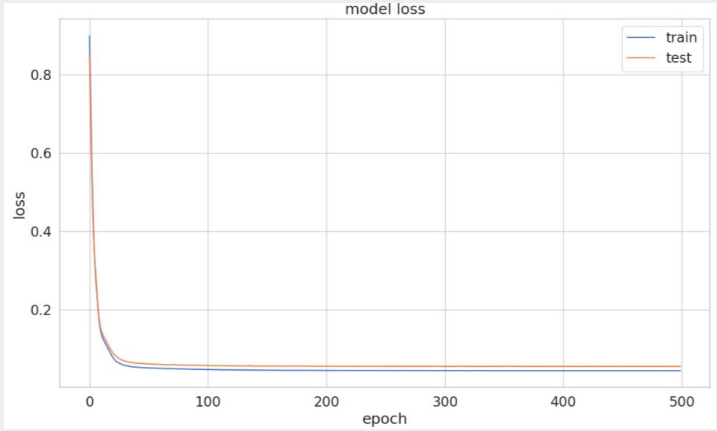
Total params: 367 (1.43 KB)
Trainable params: 367 (1.43 KB)
Non-trainable params: 0 (0.00 B)

모델학습

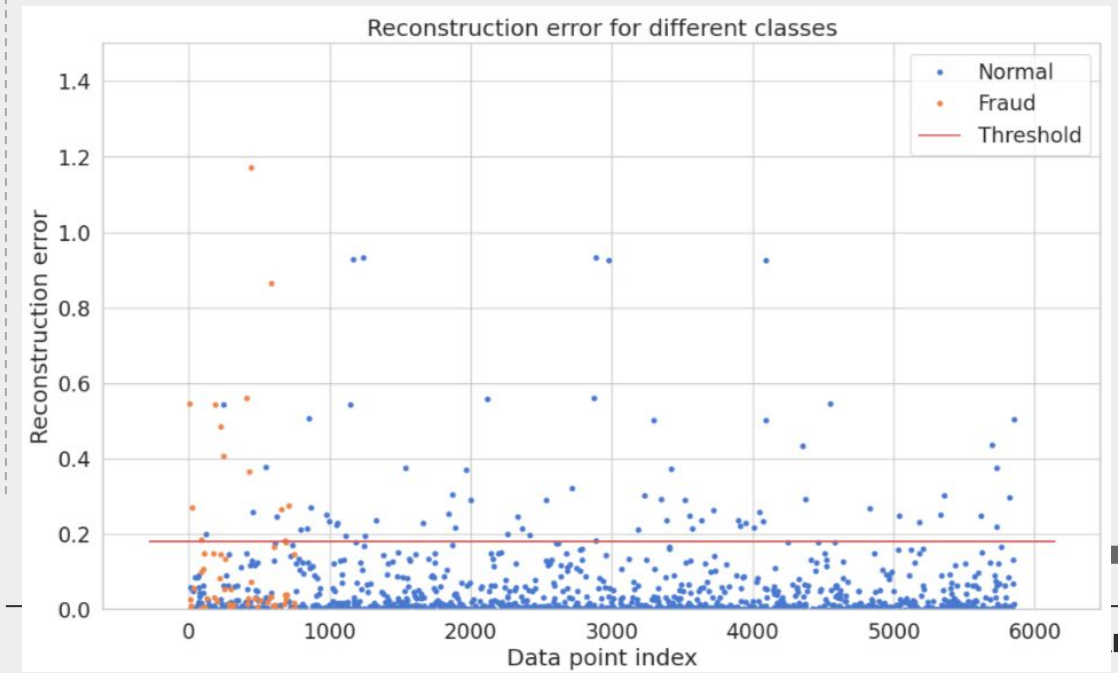
-> train시에 normal 데이터로만 학습

```
# 모델 학습
nb_epoch = 500
batch_size = 64


autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])

history = autoencoder.fit(X_train_normal, X_train_normal, # train시에 정상 데이터만으로 학습
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(X_test, X_test),     # validation시에는 정상 + 비정상 데이터로 검증
                    verbose=1).history
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')

cp = ModelCheckpoint(filepath="/kaggle/working/autoencoder_classifier.keras",
                                save_best_only=True,
                                verbose=0)
```

03 - 5 Autoencoder




```python
# test data로 reconstruction error 계산
reconstructions = autoencoder.predict(X_test)
reconstruction_errors = np.mean(np.square(X_test - reconstructions), axis=1)

# 재구성 오류 기반 이상치 여부 예측
threshold = 0.18              # 이상치로 판단할 기준
predicted_anomalies = (reconstruction_errors > threshold).astype(int)
```




```
Confusion Matrix:
[[1043   70]
 [  46   13]]
Accuracy: 0.9010238907849829
Precision: 0.1566265060240964
Recall: 0.22033898305084745
F1 Score: 0.18309859154929578
```

04 이상탐지 결과

- Isolation Forest - 0.37
- OneClass SVM - 0.47
- Random Forest - 0.68
- EllipticEnvelop (가우시안) - 0.27
- AutoEncoder - 0.22

05 역할 분담

- 배세은
  - 데이터 업샘플링
  - 이상탐지
  - 발표 자료 제작 및 발표
- 임혜진:
  - 피처 중요도 분석
  - 이상탐지
  - 발표 자료 제작 및 발표

06

프로젝트를 통해 배운 점 - 임혜진

06

프로젝트를 통해 배운 점 - 배세은

감사합니다