

```

In [1]: import pandas as pd
import numpy as np
from scipy import stats
import math

# matplotlib and seaborn for visualizations
import seaborn as sns
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 12

# Suppress warnings from pandas
import warnings
warnings.filterwarnings('ignore')

# modeling
import lightgbm as lgb

# utilities
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score

sns.set_style("darkgrid")
# memory management
import gc
import os
# 운영체제별 한글 폰트 설정
if os.name == 'posix': # Mac 환경 폰트 설정
    plt.rc('font', family='AppleGothic')
elif os.name == 'nt': # Windows 환경 폰트 설정
    plt.rc('font', family='Malgun Gothic')

plt.rc('axes', unicode_minus=False) # 마이너스 폰트 설정

# 글씨 선명하게 출력하는 설정
%config InlineBackend.figure_format = 'retina'

df = pd.read_csv("dataset2.csv")
# import mglearn
# from sklearn.model_selection import KFold
# from sklearn.preprocessing import LabelEncoder

```

```

In [2]: len(df[df['당뇨여부']==1])

```

Out[2]: 46579

```

In [3]: nodang = df[df['당뇨여부']==0].copy()
dang = df[df['당뇨여부']==1].copy()
print(len(nodang))
print(len(dang))

```

1133541  
46579

```

In [4]: x = df[['허리둘레', '연령대코드 (5세단위)', '감마지티피', '수축기혈압', '(혈청지오티)AST']].copy()
y = df[['당뇨여부']].copy()

```

비당뇨 : 당뇨 = 10 : 8

```
In [5]: from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42, sampling_strategy=0.8)
X_rus, y_rus = rus.fit_resample(X, y)
```

```
In [6]: print(len(X_rus))
print(len(y_rus))
print(np.sum(y_rus))
```

```
104802
104802
당뇨여부      46579.0
dtype: float64
```

```
In [7]: y_rus
```

```
Out[7]:
```

	당뇨여부
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0
...	...
104797	1.0
104798	1.0
104799	1.0
104800	1.0
104801	1.0

104802 rows × 1 columns

```
In [8]: X_train, X_test, y_train, y_test = train_test_split(X_rus, y_rus.values, stratify=y_rus,
from lightgbm import LGBMClassifier, plot_importance
from sklearn.preprocessing import StandardScaler, RobustScaler
sc = RobustScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)
lgb = LGBMClassifier(n_estimators=400)
evals = [(x_test, y_test)]
```

```
In [9]: # lgb.fit(x_train, y_train, early_stopping_rounds=100, eval_metric="logloss",
lgb.fit(x_train, y_train, eval_metric="logloss", eval_set=evals, verbose=False)
```

```
Out[9]: LGBMClassifier(n_estimators=400)
```

```
In [10]: y_pred = lgb.predict(x_test)
```

## 정확도

```
In [11]: lgb.score(x_test, y_test)
```

```
Out[11]: 0.6887553074757884
```

## 예측 결과

```
In [12]: print("당뇨로 예측한 데이터 수 :", np.sum(y_pred))
print("실제 당뇨 데이터 수 :", np.sum(y_test))
print("전체 데이터 수 :", len(y_test))
```

당뇨로 예측한 데이터 수 : 9968.0  
실제 당뇨 데이터 수 : 9316.0  
전체 데이터 수 : 20961

## TN, FP, FN, TP

```
In [13]: from sklearn import metrics
metrics.confusion_matrix(y_test, y_pred)

# [[TN, FP],
#  [FN, TP]]
```

```
Out[13]: array([[8057, 3588],
               [2936, 6380]])
```

## TN, FP, FN, TP 검토

```
In [14]: y_test = y_test.reshape(-1)
```

```
In [15]: P = sum(y_test)
TP = sum((y_test==1) & (y_pred==1))
TPR = TP/P
FN = sum((y_test==1) & (y_pred==0))
FNR = FN/P
N = sum(y_test==0)
TN = sum((y_test==0) & (y_pred==0))
TNR = TN/N
FP = sum((y_test==0) & (y_pred==1))
FPR = FP/N
print(TN, FP, FN, TP)
print(TPR, FPR)
```

8057 3588 2936 6380  
0.6848432803778446 0.3081150708458566

## 데이터 별 당뇨 분류 확률

```
In [16]: y_pred_proba = lgb.predict_proba(x_test)
y_pred_proba
```

```
Out[16]: array([[0.53488506, 0.46511494],
               [0.33347366, 0.66652634],
               [0.79135248, 0.20864752],
               ...,
               [0.67556595, 0.32443405],
```

```
[0.6209959 , 0.3790041 ],  
[0.50636287, 0.49363713]])
```

[당뇨X 확률, 당뇨 확률]

```
In [17]: y_pred_proba.shape
```

```
Out[17]: (20961, 2)
```

## 데이터 별 당뇨로 예측할 확률

```
In [18]: pos_proba = y_pred_proba[:,1]  
# np.unique(pos_proba)  
print(len(pos_proba[pos_proba > 0.8]))
```

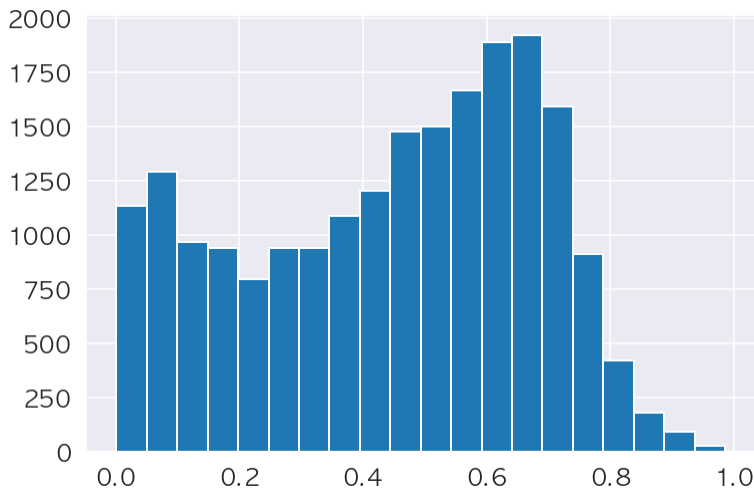
```
588
```

```
In [19]: print("pos_proba 최솟값 :", min(pos_proba))  
print("pos_proba 최댓값 :", max(pos_proba))
```

```
pos_proba 최솟값 : 0.0010101002835964416  
pos_proba 최댓값 : 0.9855479055182629
```

```
In [20]: plt.hist(pos_proba, range=(min(pos_proba),max(pos_proba)), bins=20)
```

```
Out[20]: (array([1135., 1289., 968., 940., 795., 941., 937., 1089., 1203.,  
1478., 1500., 1664., 1888., 1919., 1591., 913., 420., 177.,  
89., 25.]),  
array([0.0010101 , 0.05023699, 0.09946388, 0.14869077, 0.19791766,  
0.24714455, 0.29637144, 0.34559833, 0.39482522, 0.44405211,  
0.493279 , 0.54250589, 0.59173278, 0.64095967, 0.69018656,  
0.73941345, 0.78864034, 0.83786723, 0.88709412, 0.93632102,  
0.98554791]),  
<BarContainer object of 20 artists>)
```



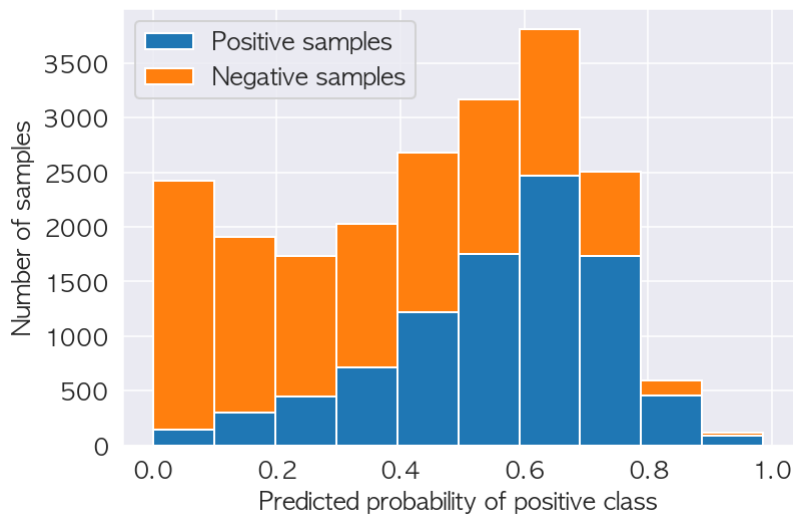
## 당뇨 예측 확률과 실제 데이터

```
In [21]: pos_sample_pos_proba = pos_proba[y_test==1]  
neg_sample_pos_proba = pos_proba[y_test==0]
```

```
In [22]: plt.hist([pos_sample_pos_proba, neg_sample_pos_proba], histtype="barstacked")  
plt.legend(["Positive samples", "Negative samples"])
```

```
plt.xlabel("Predicted probability of positive class")
plt.ylabel("Number of samples")
```

Out[22]: Text(0, 0.5, 'Number of samples')

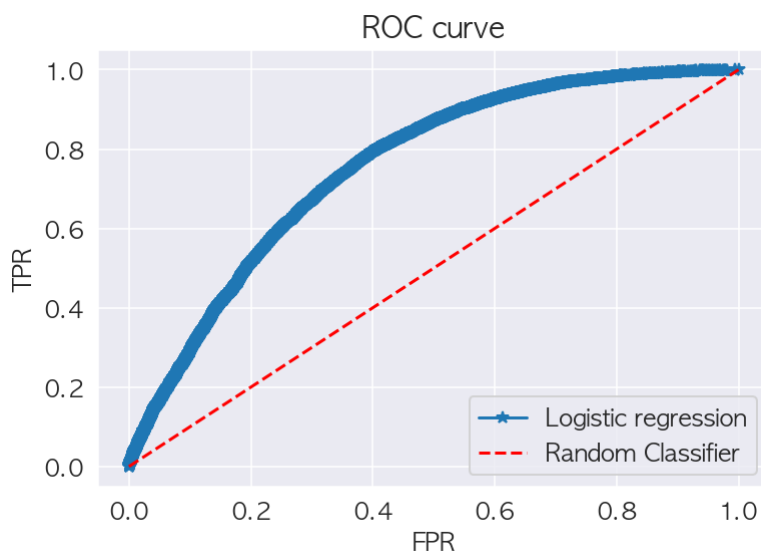


## ROC Curve

```
In [23]: fpr, tpr, thresholds = metrics.roc_curve(y_test, pos_proba)
```

```
In [24]: plt.plot(fpr, tpr, '*-')
plt.plot([0,1], [0,1], 'r--')
plt.legend(['Logistic regression', 'Random Classifier'])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
```

Out[24]: Text(0.5, 1.0, 'ROC curve')



## AUC Score

```
In [25]: metrics.roc_auc_score(y_test, pos_proba)
```

Out[25]: 0.755140350511712

# Precision, Recall, Accuracy

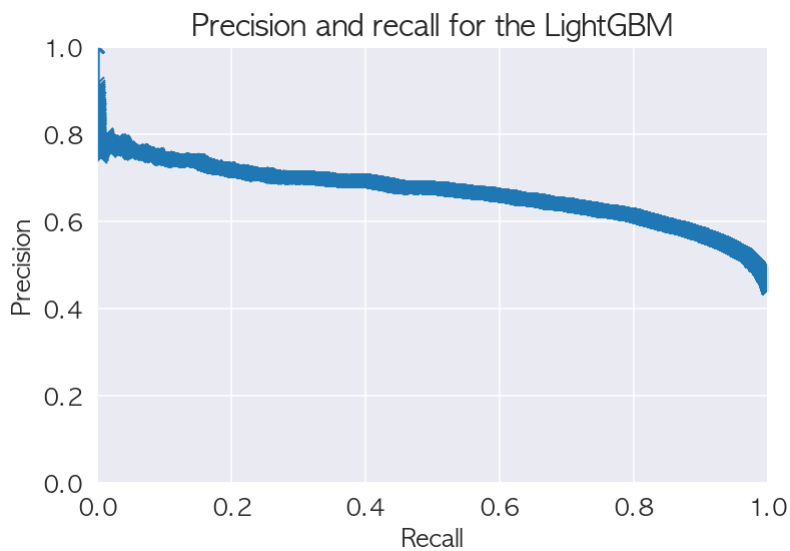
```
In [26]: precision = TP/(TP+FP)
recall = TP/(TP+FN)
accuracy = (TP+TN)/(TP+FP+TN+FN)
print("precision :",precision)
print("recall :", recall)
print("accuracy :", accuracy)
```

```
precision : 0.6400481540930979
recall : 0.6848432803778446
accuracy : 0.6887553074757884
```

```
In [27]: precision2, recall2, thresholds2 = metrics.precision_recall_curve(y_test, pos_
```

```
In [28]: plt.plot(recall2, precision2, '-x')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision and recall for the LightGBM')
plt.xlim([0,1])
plt.ylim([0,1])
```

```
Out[28]: (0.0, 1.0)
```



```
In [29]: metrics.auc(recall2, precision2)
```

```
Out[29]: 0.6646002493385829
```

```
In [ ]:
```