3장 평가

신림프로그래머 최범균

주요 내용

- 성능 지표
  - 정확도
  - 오차행렬
  - 정밀도
  - 재현율
  - F1 스코어
  - ROC AUC

정확도

- 정확도(Accuracy) = 예측 결과가 동일한 데이터 건수 / 전체 예측 데이터 건수
- 정확도는 데이터 구성에 따라 모델 성능에 왜곡 발생
  - 예: 타이타닉 생존 데이터에서 여성이면 무조건 생존으로 예측할 경우 정확도는 0.7877
- 레이블 값 분포가 불균형할 경우 평가 지표로 적합하지 않음

오차 행렬, 재현율, 정밀도

예측

실제		Negative	Positive
	Negative	TN	FP
	Positive	FN	TP

지표	공식	비고
재현율	TP / (FN+TP)	실제 P인 데이터를 N으로 잘못 판단하면 큰 영향이 발생하는 경우에 사용 (FN을 감소) 예: 암 판단 모델, 보험 사기
정밀도	TP / (FP+TP)	실제 N인 데이터를 P로 잘못 판단하면 큰 영향이 발생하는 경우에 사용 (FP를 감소) 예: 스팸 메일

사이킷런 코드

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

def get_clf_eval(y_test, pred):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)

    print('오차 행렬')
    print(confusion)
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율:{2:.4f}'
          .format(accuracy, precision, recall))
```

예측 확률, 분류 결정 임곗값

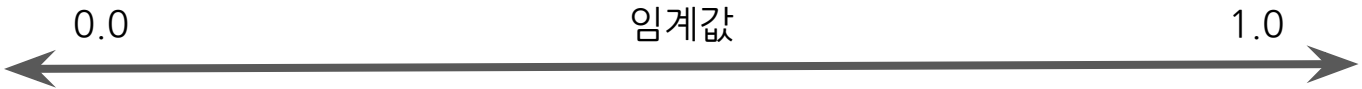
예측 확률:
- 레이블 값 확률
- predict_proba() 함수로 구함

분류 결정 임곗값
- 예측 확률이 이 값을 넘는지 여부로 레이블 값 결정
- 기본값 : 0.5

```
pred_proba = lr_clf.predict_proba(X_test)

print('pred_proba() 결과 Shape: {0}'
        .format(pred_proba.shape))
print('pred_proba 샘플 : \n', pred_proba[:7])
```

```
pred_proba() 결과 Shape: (179, 2)
pred_proba 샘플 :
 [[0.44935228 0.55064772]
  [0.86335513 0.13664487]
  [0.86429645 0.13570355]
  [0.84968519 0.15031481]
  [0.82343411 0.17656589]
  [0.84231224 0.15768776]
  [0.87095491 0.12904509]]
```

분류 결정 임곗값과 정밀도/재현율



P 예측이 많아짐
→ FP가 많아짐, FN이 줄어듬
→ 재현율 올라감

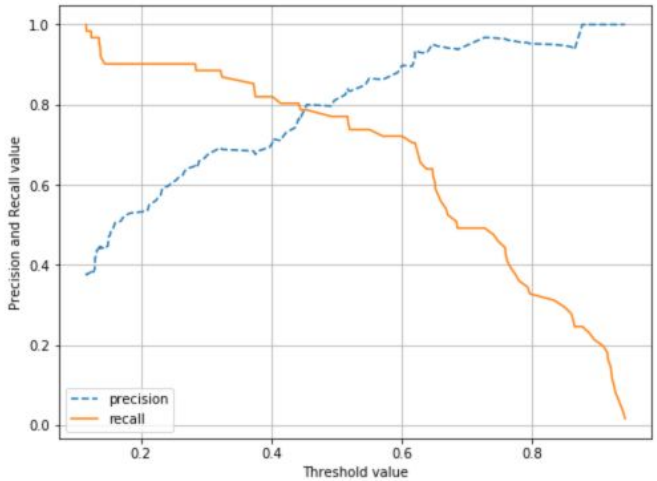
N 예측이 많아짐
→ FN이 많아짐, FP가 줄어듬
→ 정밀도 올라감

		예측	
		Negative	Positive
실제	Negative	TN	FP
	Positive	FN	TP

- precision_recall_curve(): 이진 분류에 대해 임곗값에 따른 정밀도/재현율 계산

예제 코드

```
def precision_recall_curve_plot(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_class1)

    plt.figure(figsize=(8, 6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle="--", label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary], label='recall')

    plt.xlabel('Threshold value')
    plt.ylabel('Precision and Recall value')
    plt.legend()
    plt.grid()
    plt.show()

precision_recall_curve_plot(y_test, lr_clf.predict_proba(X_test)[:, 1])
```



F1 스코어

$$F1 = 2 * \frac{정밀도 * 재현율}{정밀도 + 재현율}$$

예측

실제		Negative	Positive
	Negative	TN	FP
	Positive	FN	TP

- 정밀도와 재현율의 한 쪽으로 치우치지 않는 수치
- f1_score(실제, 예측) 함수 사용

ROC(Receiver Operation Characteristic) 곡선

ROC 곡선 : FPR이 변할 때 TPR의 변화
→ FPR이 0부터 1로 변할 때 TPR 값

- FPR = FP / (FP + TN)
  - 1 - TNR
- TPR = TP / (FN + TP)

FPR = 0 :

- 모두 negative 예측, 즉 임곗값 1

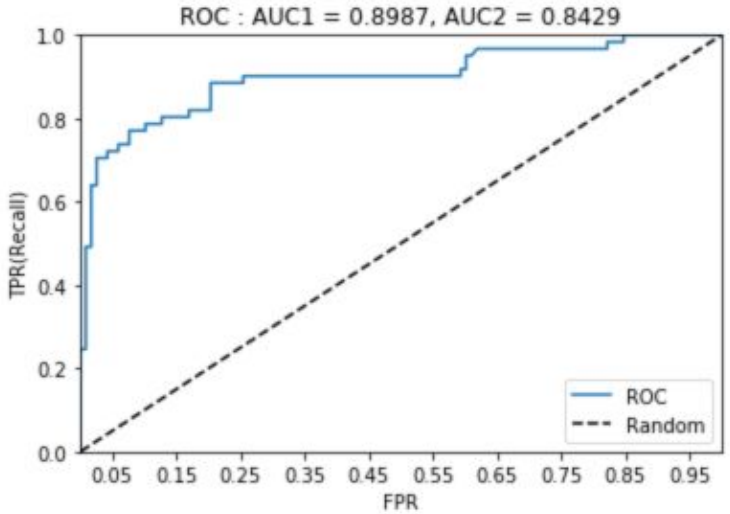
FPR = 1 :

- 모두 positive 예측, 즉 임곗값 0

AUC(Area Under Curve) : 곡선 아래 면적

- 이 값이 클수록 성능이 좋다고 볼 수 있음

		예측	
		Negative	Positive
실제	Negative	TN	FP
	Positive	FN	TP



ROC 샘플

```
def roc_curve_plot(y_test, pred_proba_c1, pred):
    fprs, tprs, thresholds = roc_curve(y_test, pred_proba_c1)
    roc_score1 = roc_auc_score(y_test, pred_proba_c1)
    roc_score2 = roc_auc_score(y_test, pred)

    plt.title('ROC : AUC1 = {0:.4f}, AUC2 = {1:.4f}'.format(roc_score1, roc_score2))
    plt.plot(fprs, tprs, label='ROC')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')

    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1), 2))
    plt.xlim(0, 1); plt.ylim(0, 1)
    plt.xlabel('FPR')
    plt.ylabel('TPR(Recall)')
    plt.legend()

pred_proba_class1 = lr_clf.predict_proba(X_test)[:, 1]
roc_curve_plot(y_test, pred_proba_class1, lr_clf.predict(X_test))
```

예제: 피마 인디언 당뇨병 예측

- 순서 : 데이터 탐색 → 전처리 → 학습 → 평가
- 몇 가지 탐색
  - Outcome 분포
  - 값 확인
  - 정보(info) : null 여부 등
  - describe() : 값의 분포도

6
```
diabetes_data = pd.read_csv('diabetes.csv')
print(diabetes_data['Outcome'].value_counts())
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

7
```
diabetes_data.head(3)
```

7

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

8
```
diabetes_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
diabetes_data.describe()
```

임신횟수 포도당 부하 검사 수치 혈압 피하지방 인슐린 체질량지수 당뇨 내력 가중치

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

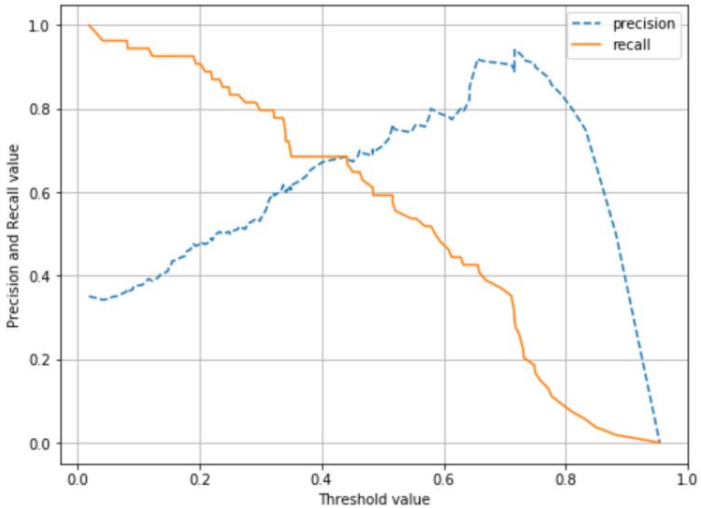
전처리 없이 학습, 평가

```
X = diabetes_data.iloc[:, :-1]
y = diabetes_data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=156, stratify=y)

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)
pred_proba_c1 = lr_clf.predict_proba(X_test)[:, 1]
precisions, recalls, thresholds = precision_recall_curve(
    y_test, pred_proba_c1)
```

오차 행렬
[[87 13]
 [22 32]]

y_test 값 분포:
0 100
1 54

정확도: 0.7727,
정밀도: 0.7111, **재현율:0.5926**,
F1:0.6465, AUC:0.7313



값 전처리

- 0 값 데이터 분포
  - Glucose 0 건수는 5, 퍼센트는 0.65 %
  - BloodPressure 0 건수는 35, 퍼센트는 4.56 %
  - SkinThickness 0 건수는 227, 퍼센트는 29.56 %
  - Insulin 0 건수는 374, 퍼센트는 48.70 %
  - BMI 0 건수는 11, 퍼센트는 1.43 %
- 데이터가 768개로 많지 않아 374개 데이터를 삭제할 수 없음
  - 0 값을 평균으로 대체

```
mean_zero_features = diabetes_data[zero_features].mean()
# mean_zero_features : Series: (5, )
diabetes_data[zero_features] = diabetes_data[zero_features].replace(0, mean_zero_features)
```

값 전처리

- 로지스틱 회귀 → 피처 스케일링
  - StandardScaler

```
X = diabetes_data.iloc[:, :-1]
y = diabetes_data.iloc[:, -1]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02
mean	2.544261e-17	-1.481743e-16	-8.655547e-17	2.385968e-16	-1.176721e-16	6.253678e-16	2.398978e-16	1.857600e-16
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00
min	-1.141852e+00	-2.553955e+00	-3.985352e+00	-2.037044e+00	-1.125139e+00	-2.074083e+00	-1.189553e+00	-1.041549e+00

예측 평가

- 간단 평가지표 생성 함수

```
def metric_table(y_test, pred_proba_c1):
    precisions, recalls, thresholds = precision_recall_curve(y_test, pred_proba_c1)
    accuracies = np.zeros(thresholds.shape[0])
    f1s = np.zeros(thresholds.shape[0])
    aucs = np.zeros(thresholds.shape[0])

    for i in range(thresholds.shape[0]):
        binarizer = Binarizer(threshold = thresholds[i])
        pred = binarizer.fit_transform(pred_proba_c1.reshape(-1, 1))
        accuracy = accuracy_score(y_test, pred)
        f1 = f1_score(y_test, pred)
        auc = roc_auc_score(y_test, pred)
        accuracies[i] = accuracy
        f1s[i] = f1
        aucs[i] = auc

    mt_df = pd.DataFrame()
    mt_df['Thresholds'] = thresholds
    mt_df['Accuracy'] = accuracies
    mt_df['Precision'] = precisions[0:-1]
    mt_df['Recalls'] = recalls[0:-1]
    mt_df['F1'] = f1s
    mt_df['AUC'] = aucs
    return mt_df
```

학습/평가

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=156, stratify=y)

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
pred_proba = lr_clf.predict_proba(X_test)

mt_df = metric_table(y_test, pred_proba[:, 1])

auc_idxmax = mt_df['AUC'].idxmax()
mt_df.iloc[auc_idxmax - 2:auc_idxmax + 3, :]
```

	Thresholds	Accuracy	Precision	Recalls	F1	AUC
74	0.458946	0.785714	0.700000	0.648148	0.679612	0.754074
75	0.464070	0.792208	0.714286	0.648148	0.686275	0.759074
76	0.466450	0.798701	0.729167	0.648148	0.693069	0.764074
77	0.484825	0.792208	0.744681	0.648148	0.680000	0.754815
78	0.488567	0.785714	0.739130	0.629630	0.666667	0.745556