[머신러닝 및 딥러닝] 프로젝트 최종발표

유방암 데이터 분석

2018111374 경영정보학과 박세연

유전자별 수치적 특징을 이용해 유방암 여부를 예측하기

cancerdata.csv

590개 샘플

17814 features (genes)

Label
0: normal
1: tumor

	TCGA_Barcode	ELMO2	CREB3L1	RPS11	PNMA1	MMP2	C10orf90	ZHX3	ERCC5	GPR98	...	GRIP2	GPLD1	RAB8A	RXFP2	PIK3IP1	SLC39A6	SNRPD2	AQP7	CTSC	Label
0	TCGA-BH-A0DI-01A-21R-A12P-07	0.180917	1.37125	0.541625	-0.04075	2.685000	-1.61125	0.120333	0.18525	-2.289714	...	0.02700	-1.419833	0.1279	0.5085	2.87375	-0.7602	-0.4909	0.0500	0.4694	1
1	TCGA-A8-A097-01A-11R-A034-07	0.138417	0.69650	0.408250	-0.10525	1.383167	-2.00400	-0.396333	-0.01775	1.393125	...	-0.37175	-0.295000	0.2442	0.0405	2.29300	1.2055	0.0757	-0.9890	-0.0048	1
2	TCGA-AO-A0J6-01A-11R-A034-07	0.189667	0.68500	0.113750	-0.59325	-1.189167	-1.75600	0.662333	-0.79200	-0.546143	...	-0.31825	-0.970600	0.1242	0.8915	0.89825	-1.4979	0.2061	-1.5645	1.6926	1
3	TCGA-B6-A0I5-01A-11R-A034-07	-0.629583	1.59200	0.928500	0.21200	-0.642167	-1.85150	0.388167	0.67325	-0.658875	...	-0.44825	-0.622000	-0.2747	-0.1230	0.89475	0.7408	0.7055	0.0160	-0.9467	1
4	TCGA-E2-A15I-01A-21R-A137-07	-0.549333	1.70175	0.755500	-0.51250	0.668333	-2.32150	-0.383333	1.04500	0.275500	...	-0.16775	-1.581167	-0.6578	0.3440	2.07350	1.2089	-0.1986	-1.3735	-0.4623	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
585	TCGA-BH-A0AU-11A-11R-A12P-07	-0.496500	-0.22400	1.350500	-0.20175	-0.327333	0.93175	-0.464000	0.47375	0.264750	...	0.18400	-0.067167	-0.6067	-0.0165	2.72750	0.7778	-0.1336	1.0865	-0.8355	0
586	TCGA-AO-A03R-01A-21R-A034-07	-0.155000	0.85375	0.014125	-0.34125	-0.615500	-1.57675	-0.389667	-0.10775	-1.570375	...	-0.14925	0.105000	0.6177	-0.0025	2.14300	2.6502	-0.3464	0.6630	-0.5336	1
587	TCGA-A8-A09R-01A-11R-A00Z-07	1.349417	0.75325	0.064250	0.17950	1.051833	-2.29550	0.228000	-0.17750	-1.373000	...	-0.13650	-1.443000	0.9306	1.1690	2.60250	2.4011	-0.1116	-0.5255	0.0319	1
588	TCGA-E2-A10B-01A-11R-A10J-07	1.494583	0.92225	0.531875	0.62425	0.653167	-1.43075	0.491500	-0.16475	-1.677125	...	-0.16800	-0.356167	0.4708	0.1350	1.77275	0.9740	0.4100	-0.4880	-0.3867	1
589	TCGA-BH-A0AY-11A-23R-A089-07	0.204333	-0.24200	0.591875	0.53850	0.707667	0.27550	-0.117167	0.51850	-0.018375	...	0.27325	0.886167	-0.0019	0.8730	2.87875	1.3782	-0.2629	3.4080	0.0620	0

590 rows × 17816 columns

- 필요 없는 열 존재
- 많은 결측치 존재
- 컬럼 간 min, max, 평균값 차이가 존재
- 데이터 수에 비해 너무 많은 피처 수(차원)

TCGA Barcode 열 삭제

이미 라벨로 존재하는 정보이거나 암 여부를 예측하는 데에는 필요 없는 정보라 판단해 해당 열 삭제

	TCGA_Barcode	ELMO2	CREB3L1	RPS11	PNMA1	MMP2	C10orf90
0	TCGA-AR-A0TP-01A-11R-A084-07	0.628083	-0.22975	0.372625	-0.86550	-0.994333	2.27700
1	TCGA-C8-A12P-01A-11R-A115-07	0.563750	1.28450	-0.222500	0.10800	1.158833	-2.39825
2	TCGA-A8-A07C-01A-11R-A034-07	-0.290000	-0.35450	0.668625	-0.37025	-0.692167	0.48175
3	TCGA-BH-A18H-01A-11R-A12D-07	0.369250	0.82050	2.497500	-0.73025	-0.103667	-2.45875
4	TCGA-A1-A0SD-01A-11R-A115-07	0.507083	1.43450	0.765000	0.52600	1.506000	-2.01825
...	...	...	...	...	...	...	...

삭제

	ELMO2	CREB3L1	RPS11	PNMA1	MMP2	C10orf90
0	0.507955	0.255342	0.394180	0.244063	0.211172	0.961115
1	0.494029	0.582182	0.226438	0.520096	0.563554	0.345889
2	0.309221	0.228416	0.477610	0.384490	0.260624	0.724874
3	0.451927	0.482031	0.993094	0.282413	0.356936	0.337928
4	0.481763	0.614559	0.504774	0.638619	0.620370	0.395894
...	...	...	...	...	...	...

결측치 확인

590개의 행 중 534개의 행에서 결측값(NaN)이 확인됨

```
# 결측치 존재 여부 확인 (True - 결측치 O)
data.isnull().values.any()
```

True

```
# 결측값이 있는 열 확인(열이름, 결측치 개수)
data_checknull = pd.DataFrame(data.isnull().sum())
data_havenull = data_checknull[data_checknull[0]!=0]
```

```
# null값이 있는 컬럼 확인 (결측치존재열, 결측치 개수)
data_havenull
```

	0
OR2K2	5
ADAM5P	14
KIAA1486	1
C9orf11	2
C8G	1
...	...
OR1J4	14
MLNR	1
SALL3	2
CCDC73	1
RXFP2	1

534 rows × 1 columns

열마다 1~17개 사이의 결측치 존재

결측치 대체

방법1.

라벨별로 나눠 결측치가 있는 컬럼의 평균값으로 대체

예시)

```
# 라벨 1인 데이터의 OR2K2 평균값 확인
data[data['Label']==1]['OR2K2'].mean()
```

```
0.08751526717557252
```

```
# 라벨 0인 데이터의 OR2K2 평균값 확인
data[data['Label']==0]['OR2K2'].mean()
```

```
0.14919672131147543
```

결측치 대체

```
# 결측값 열,라벨별 평균값으로 대체
for col in havenull_columns:
  # (해당 열에 결측치 없음 && 라벨 1) 인 데이터의 평균값으로 열의 결측값 대체
  data.loc[(data[col].isnull()) & (data['Label']==1),col] = data[data['Label']==1][col].mean()
  data.loc[(data[col].isnull()) & (data['Label']==0),col] = data[data['Label']==0][col].mean()
```

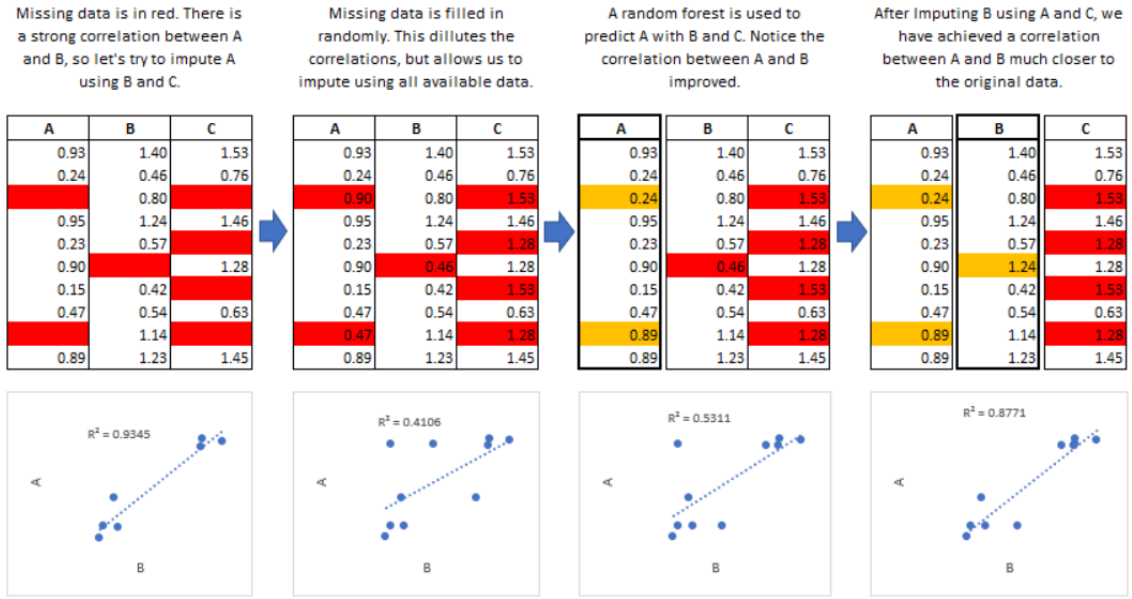
```
# 결측치 여부 확인 (False - 결측치 없음)
data.isnull().values.any()
```

```
False
```

방법2. ☑

MICE 알고리즘을 적용해 결측치 대체

MICE(Multiple Imputation by Chained Equations)
: 다른 열의 데이터를 보고 각 누락값에 대해 반복 예측



https://ichi.pro/ko/deiteo-seteueseo-gyeol-cheuggabs-eul-daechihaneun-mice-algolijeum-217004654686142

결측치 대체

MICE 알고리즘을 적용해 결측치 대체

MICE(Multiple Imputation by Chained Equations): 다른 열의 데이터를 보고 각 누락값에 대해 반복 예측
반복 수는 3으로, 특정 열의 결측치를 추정할 때 사용할 다른 피처 수는 5로 설정

```
df = data.copy()
df = IterativeImputer(random_state=100, max_iter=3, n_nearest_features=5).fit_transform(df)
df = pd.DataFrame(df)
df.columns = data.columns
df
```

	ELMO2	CREB3L1	RPS11	PNMA1	MMP2	C10orf90	ZHX3	ERCC5	GPR98	RXFP3	...
0	0.587000	0.11425	0.268500	-0.69200	-0.849333	-1.67225	-0.067333	0.29050	-1.950250	0.0650	...
1	-1.718500	0.48700	1.859250	-0.06575	0.484333	0.13300	-0.395167	0.38150	-0.450500	-0.1465	...
2	0.928833	0.65550	-0.340750	0.30225	0.050667	-2.47250	0.491333	-0.64550	0.881000	0.6195	...
3	0.430250	0.90625	0.258000	0.79850	1.577000	-2.43850	-0.281167	0.56700	-0.085875	0.2205	...
4	1.032917	0.84575	-0.277250	0.33800	0.643333	-2.62200	-0.078167	0.45625	1.326000	0.7475	...
...	...	...	...	...	...	...	...	...	...	...	...

```
# 결측치 존재 여부 확인 (False - 결측치 없어짐)
df.isnull().values.any()
```

```
False
```

https://ichi.pro/ko/deiteo-seteueseo-gyeol-cheuggabs-eul-daechihaneun-mice-algolijeum-217004654686142

데이터 정규화

MinMaxScaler를 통해 모든 값을 0과 1사이로 조정

정규화 전)

```
# 피처 분산 중 max, min 확인
print(data.var().max())
print(data.var().min())
```

```
11.907856670206852
0.03954672716073011
```

정규화 후)

	ELMO2	CREB3L1	RPS11	PNMA1	MMP2	C10orf90	ZHX3	ERCC5	GPR98	RXFP3
0	0.462966	0.460069	0.113695	0.206139	0.352572	0.637037	0.466738	0.318564	0.178522	0.314568
1	0.456400	0.379290	0.334249	0.492380	0.453003	0.420897	0.238774	0.607298	0.520437	0.327785
2	0.420593	0.717678	0.314414	0.654214	0.672167	0.440208	0.339231	0.477339	0.675951	0.246426
3	0.461974	0.803367	0.547863	0.361523	0.442884	0.370694	0.495994	0.504054	0.316556	0.351478
4	0.434249	0.543546	0.339358	0.390586	0.390732	0.423825	0.355042	0.413988	0.487864	0.344723
...	...	...	...	...	...	...	...	...	...	...
585	0.404791	0.612076	0.509072	0.539094	0.752959	0.319604	0.547728	0.593614	0.454280	0.376052
586	0.367252	0.804015	0.380192	0.292054	0.432655	0.416785	0.360688	0.795251	0.242872	0.550225
587	0.476910	0.575923	0.301166	0.676969	0.287109	0.379544	0.531111	0.431219	0.395826	0.260916
588	0.405855	0.389812	0.948737	0.626923	0.439365	0.410896	0.724818	0.367941	0.236343	0.282651
589	0.244679	0.436758	0.554945	0.500673	0.537859	0.480409	0.251627	0.389082	0.299272	0.350010

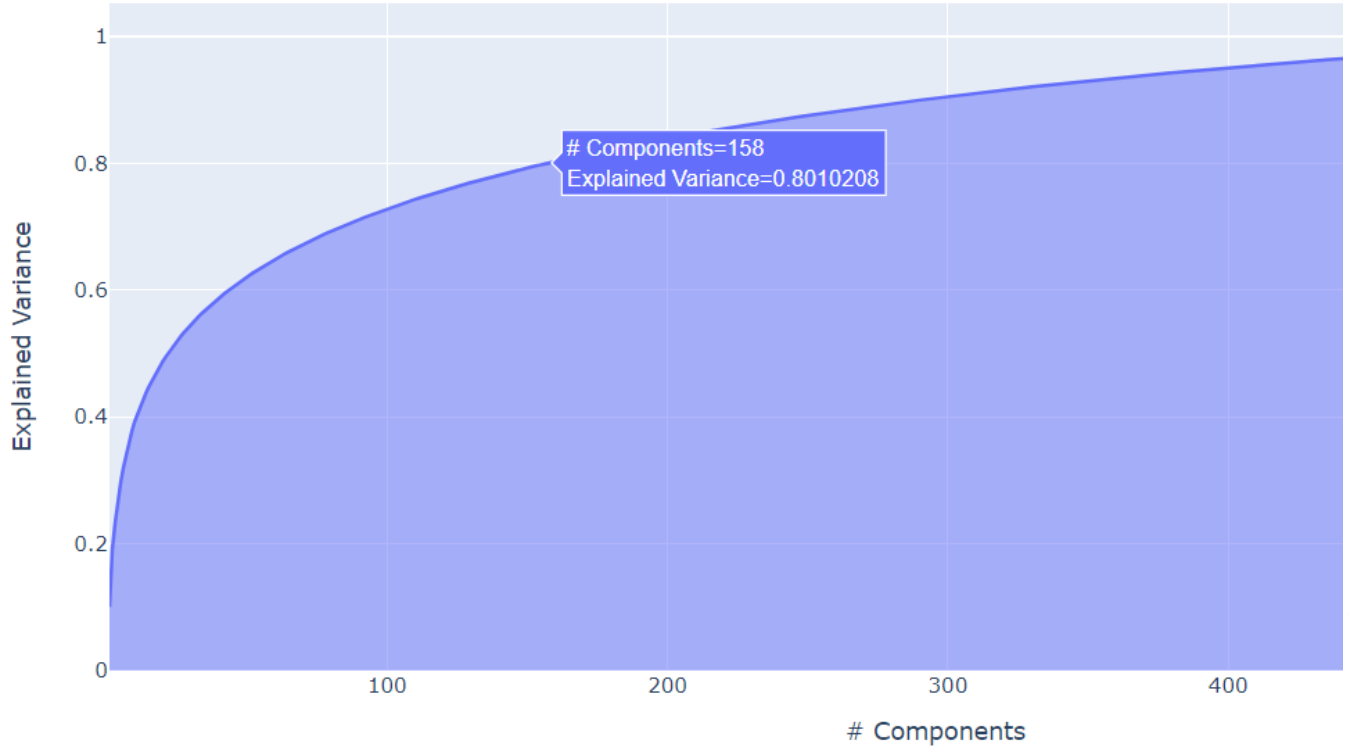
590 rows × 17815 columns

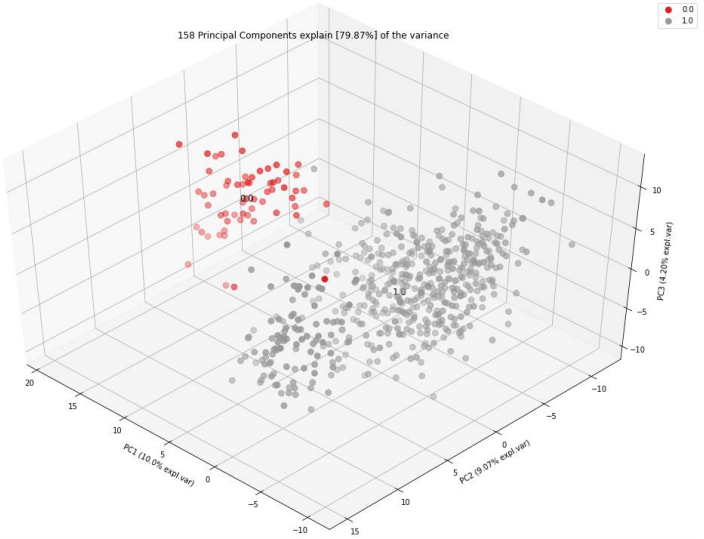
```
# 정규화 후 min, max 확인
print(data.max())
print(data.min())
```

```
ELMO2       1.0
CREB3L1     1.0
RPS11       1.0
PNMA1       1.0
MMP2        1.0
           ...
SLC39A6     1.0
SNRPD2      1.0
AQP7        1.0
CTSC        1.0
Label       1.0
Length: 17815, dtype: float64
ELMO2       0.0
CREB3L1     0.0
RPS11       0.0
PNMA1       0.0
MMP2        0.0
           ...
SLC39A6     0.0
SNRPD2      0.0
AQP7        0.0
CTSC        0.0
Label       0.0
Length: 17815, dtype: float64
```

차원 축소 - PCA(주성분분석)

너무 많은 피처를 줄이기 위해 차원 축소
주성분이 158개일 때 전체 분산의 약 80%를 설명





PCA로 생성된 데이터셋

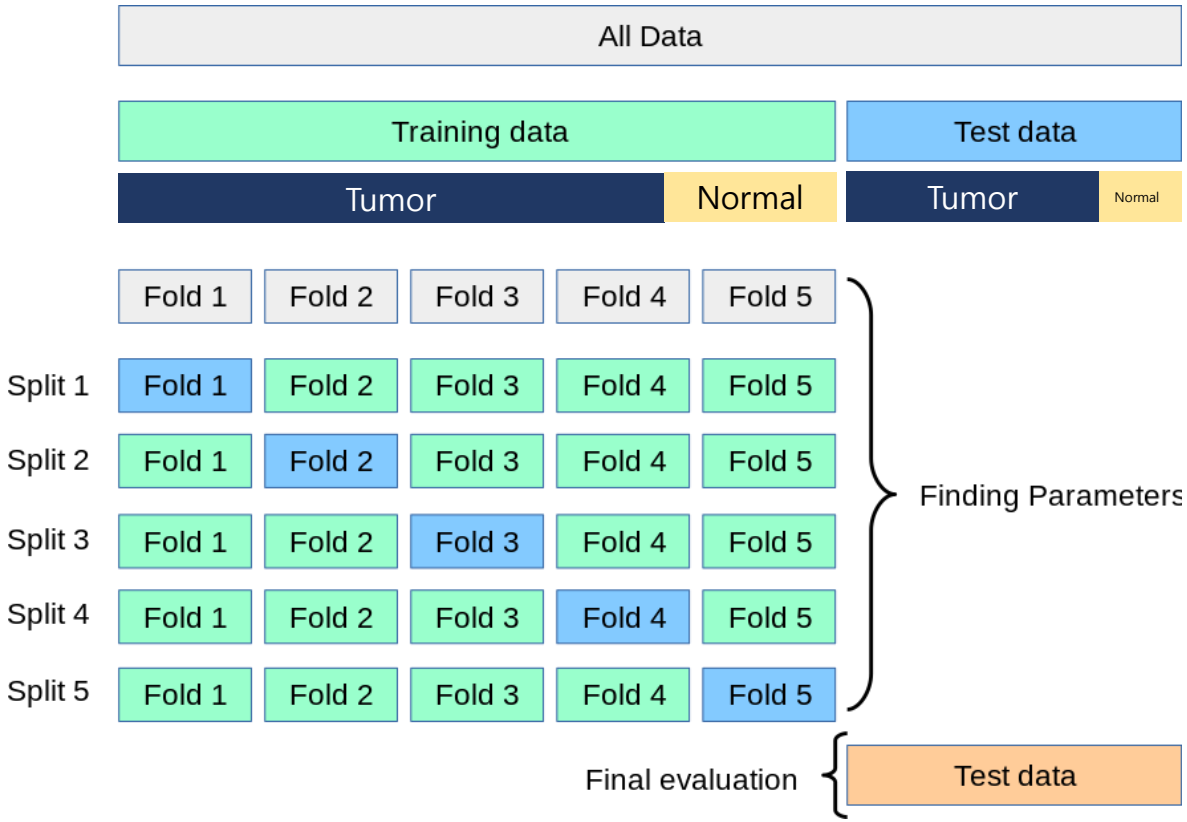
590개의 데이터, 158개 피처, 1개의 라벨

	pca1	pca2	pca3	pca4	pca5	pca6	pca7	pca8	pca9	pca10	...
0	-6.494447	12.521870	3.726510	-5.202321	-2.062288	-2.142860	-0.057223	-1.146435	-2.754533	-1.119958	...
1	-1.171160	2.623938	-5.291850	-2.237373	1.913409	2.509557	3.516055	3.190992	1.450114	0.095019	...
2	1.092819	-2.464298	-4.181340	-0.482211	0.612548	-2.339397	0.917369	1.515654	1.686762	-2.903395	...
3	5.768807	4.668721	-5.127323	1.429151	-1.667571	0.352744	1.002424	-3.363762	-1.032406	-0.450643	...
4	-5.914409	-0.677205	-2.743890	1.800382	-0.782090	3.292636	-3.298177	-2.257637	0.042433	1.900610	...

5 rows × 158 columns

Stratified k-Fold

train과 test셋에 라벨별 비율 유지, train:test = 8:2 로 분할
k=5로 설정해 교차검증을 통해 적은 데이터셋에서 성능 향상




```python
str_kf = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 100)

X = np.array(pca_df.iloc[:, :-1]) # features
y = pca_df['Label'] # label

for train_index, test_index in str_kf.split(X, y):
    #print(len(train_index)) # 472
    #print(len(test_index)) # 118
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

모델 학습

1. 다양한 모델 개별 학습

대부분 모델이 F1-score 99%에 가까운 성능을 보여줌

1) Logistic Regression
2) Decision Tree
3) SVM
4) Random Forest
5) Adaboost
6) Gradient Boosting(GBM)
7) XGBoosting
8) LGBM(Light GBM)
9) CatBoost

	Algorithm	Accuracy	Specificity	Precision	Recall	F1-score
0	Decision Tree	0.986	0.934	0.992	0.993	0.992
1	Random Forest	0.963	1.000	1.000	0.960	0.980
2	Logistic Regression	0.995	0.968	0.996	0.998	0.997
3	Gradient Boosting	0.986	0.934	0.992	0.993	0.992
4	LGBM	0.985	0.963	0.996	0.987	0.992
5	XGB	0.990	0.937	0.992	0.996	0.994
6	AdaBoost	0.992	0.951	0.994	0.996	0.995
7	SVM	0.993	0.966	0.996	0.996	0.996
8	CatBoost	0.990	0.937	0.992	0.996	0.994

Specificity: 실제 Normal에 대해 Normal로 예측한 확률
Precision: Tumor로 예측한 것 중 실제 Tumor인 확률
Recall: 실제 Tumor 중 Tumor로 예측한 확률

모델 학습

2. Soft Voting 앙상블

앞의 개별 모델 중 몇 개를 합쳐 soft voting으로 앙상블
개별 모델 성능이 괜찮은 선형 회귀, AdaBoost, SVM를 합친 결과,
잘못 예측한 경우 하나도 없이 100%의 성능을 보여줌

```
# voting 모델
voting_model = VotingClassifier(estimators=[('Logistic Regression',lr),('AdaBoost', adaboost),('SVM', svm)], voting='soft')

# 모델 학습
voting_model.fit(X_train, y_train)
y_pred = voting_model.predict(X_test)

# 성능 평가
accuracy= accuracy_score(y_pred, y_test)
precision = precision_score(y_pred, y_test)
recall = recall_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test)
specificity = specificity_score(y_pred, y_test)

print('Voting Classifier의 성능')
print("accuracy: {:.2f}".format(np.mean(accuracy)))
print("specificity: {:.2f}".format(np.mean(specificity)))
print("precision: {:.2f}".format(np.mean(precision)))
print("recall: {:.2f}".format(np.mean(recall)))
print("f1 score: {:.2f}".format(np.mean(f1)))
```

```
Voting Classifier의 성능
accuracy: 1.00
specificity: 1.00
precision: 1.00
recall: 1.00
f1 score: 1.00
```

모델 학습

3. 딥러닝 모델 TabNet

정형 데이터에 특화된 DT 기반 알고리즘 TabNet을 이용해 학습 진행

최대 F1-Score 99%의 성능을 가짐

```
clf = TabNetClassifier(optimizer_fn=torch.optim.Adam,
                       optimizer_params=dict(lr=1e-2),
                       scheduler_params={"step_size":50,
                                         "gamma":0.9},
                       scheduler_fn=torch.optim.lr_scheduler.StepLR,
                       mask_type='sparsemax' # "sparsemax", entmax
                      )
max_epochs = 150

clf.fit(
    X_train=X_train, y_train=y_train,
    eval_set=[(X_train, y_train)],
    eval_metric=['accuracy','balanced_accuracy','logloss'],
    max_epochs=max_epochs , patience=20,
    batch_size=64, virtual_batch_size=128,
    num_workers=0,
    weights=1,
    drop_last=False,
)
```

```
TabNet의 성능
accuracy: 0.98
specificity: 1.00
precision: 1.00
recall: 0.98
f1 score: 0.99
```

성능 평가

	Algorithm	Accuracy	Specificity	Precision	Recall	F1-score
0	Decision Tree	0.983	0.921	0.991	0.991	0.991
1	Random Forest	0.966	0.978	0.998	0.965	0.981
2	Logistic Regression	0.997	0.973	0.996	1.000	0.998
3	Gradient Boosting	0.983	0.922	0.991	0.991	0.991
4	LGBM	0.988	0.954	0.994	0.992	0.993
5	XGB	0.992	0.943	0.992	0.998	0.995
6	AdaBoost	0.992	0.971	0.996	0.994	0.995
7	SVM	0.993	0.971	0.996	0.996	0.996
8	CatBoost	0.988	0.927	0.991	0.996	0.993
9	Voting Classifier	0.975	0.857	0.981	0.990	0.986
10	TabNet	0.980	1.000	1.000	0.980	0.990

Specificity: 실제 Normal에 대해 Normal로 예측한 확률
Precision: Tumor로 예측한 것 중 실제 Tumor인 확률
Recall: 실제 Tumor 중 Tumor로 예측한 확률

감사합니다