In [1]:
```
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv("./data/train_완.csv",encoding='CP949')

X = data.iloc[:, 4:13] # 두개의 클래스와 두개의 특성만 선택
y = data.iloc[:, 13:]

# 타깃 벡터에서 00이 아닌 클래스는 모두 1로 만들기
y = np.where((y==0), 0, 1)

# 특성 표준화
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

In [3]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_std, y, test_size=0.3, random_s
```

Logistic Regression

In [35]:
```
from sklearn.linear_model import LogisticRegression

logistic = LogisticRegression()

# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : [0.001, 0.01, 0.1, 1, 10, 25, 50, 100],
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}

# k-폴드 교차검증
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# 계층적 k-fold
skf = StratifiedKFold(n_splits=5, shuffle=True)

# RepeatedKFole 분할기
rfk = RepeatedStratifiedKFold(n_splits=5, n_repeats=100, random_state=42)

# gridsearch 객체
```

```
gridsearch_logistic = GridSearchCV(logistic,                    # model
                                   param_grid = parameters,     # hyperparameters
                                   scoring='accuracy',          # metric for scoring
                                   cv=skf)                       # number of folds

# 그리드 서치 수행
best_model = gridsearch_logistic.fit(X_train, y_train)
```

In [36]:

```
result = pd.DataFrame(gridsearch_logistic.cv_results_['params'])
result['mean_test_score'] = gridsearch_logistic.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

Out[36]:

	C	penalty	solver	mean_test_score
9	0.010	l2	newton-cg	0.631522
10	0.010	l2	lbfgs	0.631522
29	10.000	l2	liblinear	0.623913
28	10.000	l2	lbfgs	0.623913
27	10.000	l2	newton-cg	0.623913
11	0.010	l2	liblinear	0.623188
17	0.100	l2	liblinear	0.623188
47	100.000	l2	liblinear	0.615580
40	50.000	l2	lbfgs	0.615580
35	25.000	l2	liblinear	0.615580
34	25.000	l2	lbfgs	0.615580
33	25.000	l2	newton-cg	0.615580
32	25.000	l1	liblinear	0.615580
41	50.000	l2	liblinear	0.615580
39	50.000	l2	newton-cg	0.615580
44	100.000	l1	liblinear	0.615580
45	100.000	l2	newton-cg	0.615580
46	100.000	l2	lbfgs	0.615580
38	50.000	l1	liblinear	0.615580
22	1.000	l2	lbfgs	0.614855
23	1.000	l2	liblinear	0.614855
21	1.000	l2	newton-cg	0.614855
5	0.001	l2	liblinear	0.614855
16	0.100	l2	lbfgs	0.614130
15	0.100	l2	newton-cg	0.614130
26	10.000	l1	liblinear	0.606884
20	1.000	l1	liblinear	0.606522
14	0.100	l1	liblinear	0.597464

	C	penalty	solver	mean_test_score
3	0.001	l2	newton-cg	0.530072
4	0.001	l2	lbfgs	0.530072
8	0.010	l1	liblinear	0.521377
2	0.001	l1	liblinear	0.521377
0	0.001	l1	newton-cg	NaN
1	0.001	l1	lbfgs	NaN
6	0.010	l1	newton-cg	NaN
7	0.010	l1	lbfgs	NaN
12	0.100	l1	newton-cg	NaN
13	0.100	l1	lbfgs	NaN
18	1.000	l1	newton-cg	NaN
19	1.000	l1	lbfgs	NaN
24	10.000	l1	newton-cg	NaN
25	10.000	l1	lbfgs	NaN
30	25.000	l1	newton-cg	NaN
31	25.000	l1	lbfgs	NaN
36	50.000	l1	newton-cg	NaN
37	50.000	l1	lbfgs	NaN
42	100.000	l1	newton-cg	NaN
43	100.000	l1	lbfgs	NaN

In [37]:
```
# 테스트 세트 점수
test_score = gridsearch_logistic.score(X_test, y_test)

print("테스트 세트 점수: {:.2f}".format( test_score ))
print("최적 매개변수: {}".format(gridsearch_logistic.best_params_))
print("최고 교차 검증 점수: {:.2f}".format(gridsearch_logistic.best_score_))
```

```
테스트 세트 점수: 0.63
최적 매개변수: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-cg'}
최고 교차 검증 점수: 0.63
```

In [38]:
```
logistic = LogisticRegression(C= 0.01, penalty= 'l2', solver= 'newton-cg')
logistic.fit(X_train,y_train)
pred = logistic.predict(X_test)
accuracy = accuracy_score(y_test,pred)
accuracy
```

Out[38]: 0.6274509803921569

In [39]:
```
pred
```

Out[39]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

Support Vector Machine (SVM)

In [9]:
```
from sklearn.svm import SVC

svc = SVC()

# parameter grid
parameters =  {'C' : [0.001, 0.01, 0.1, 1, 10, 25, 50, 100],
               'gamma': [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0],
               'kernel': ['rbf', 'poly', 'sigmoid']
}

# k-폴드 교차검증
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# 계층적 k-fold
skf = StratifiedKFold(n_splits=5, shuffle=True)

# RepeatedKFole 분할기
rfk = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state=42)

# gridsearch 객체
gridsearch_svc = GridSearchCV(svc,                  # model
                              param_grid = parameters,   # hyperparameters
                              scoring='accuracy',        # metric for scoring
                              cv=skf)                     # number of folds

# 그리드 서치 수행
best_model = gridsearch_svc.fit(X_train, y_train)
```

In [10]:
```
result = pd.DataFrame(gridsearch_svc.cv_results_['params'])
result['mean_test_score'] = gridsearch_svc.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

Out[10]:

	C	gamma	kernel	mean_test_score
137	50.000	0.10	sigmoid	0.692391
95	10.000	0.10	sigmoid	0.691667
158	100.000	0.10	sigmoid	0.683333
116	25.000	0.10	sigmoid	0.665942
90	10.000	0.01	rbf	0.630435
...	...	...	...	...
58	0.100	10.00	poly	0.470290
40	0.010	100.00	poly	0.470290
156	100.000	0.10	rbf	0.460145
76	1.000	1.00	poly	0.453261
16	0.001	10.00	poly	0.453261

168 rows × 4 columns

In [11]:
```
# 테스트 세트 점수
test_score = gridsearch_svc.score(X_test, y_test)

print("테스트 세트 점수: {:.2f}".format( test_score ))
print("최적 매개변수: {}".format(gridsearch_svc.best_params_))
print("최고 교차 검증 점수: {:.2f}".format(gridsearch_svc.best_score_))
```

```
테스트 세트 점수: 0.69
최적 매개변수: {'C': 50, 'gamma': 0.1, 'kernel': 'sigmoid'}
최고 교차 검증 점수: 0.69
```

In [40]:
```
#위의 결과로 나온 최적 하이퍼 파라미터로 다시 모델을 학습하여 테스트 세트 데이터에서 (

svc = SVC(C=50, gamma=0.1, kernel='sigmoid')
svc.fit(X_train,y_train)
pred = svc.predict(X_test)
accuracy = accuracy_score(y_test,pred)
accuracy
```

Out[40]:
```
0.6862745098039216
```

In [41]:
```
pred
```

Out[41]:
```
array([0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0])
```

Random Forest

In [122…
```
params ={
    'n_estimators':[100],
    'max_depth':[6,8,10,12],
    'min_samples_leaf':[8,12,18],
    'min_samples_split':[8,16,20]
}

rf = RandomForestClassifier(random_state=0, n_jobs=-1)
grid_rf = GridSearchCV(rf, param_grid=params, cv=skf,scoring='accuracy', n_jobs=-1)
best_model = grid_rf.fit(X_train,y_train)
```

In [123…
```
result = pd.DataFrame(grid_rf.cv_results_['params'])
result['mean_test_score'] = grid_rf.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

Out[123…

	max_depth	min_samples_leaf	min_samples_split	n_estimators	mean_test_score
0	6	8	8	100	0.642029
9	8	8	8	100	0.642029
28	12	8	16	100	0.642029
27	12	8	8	100	0.642029
19	10	8	16	100	0.642029

	max_depth	min_samples_leaf	min_samples_split	n_estimators	mean_test_score
1	6	8	16	100	0.642029
10	8	8	16	100	0.642029
18	10	8	8	100	0.642029
20	10	8	20	100	0.633333
11	8	8	20	100	0.633333
29	12	8	20	100	0.633333
2	6	8	20	100	0.633333
8	6	18	20	100	0.608333
34	12	18	16	100	0.608333
33	12	18	8	100	0.608333
26	10	18	20	100	0.608333
25	10	18	16	100	0.608333
24	10	18	8	100	0.608333
7	6	18	16	100	0.608333
35	12	18	20	100	0.608333
17	8	18	20	100	0.608333
16	8	18	16	100	0.608333
15	8	18	8	100	0.608333
6	6	18	8	100	0.608333
4	6	12	16	100	0.590580
21	10	12	8	100	0.590580
5	6	12	20	100	0.590580
23	10	12	20	100	0.590580
3	6	12	8	100	0.590580
14	8	12	20	100	0.590580
13	8	12	16	100	0.590580
30	12	12	8	100	0.590580
31	12	12	16	100	0.590580
32	12	12	20	100	0.590580
12	8	12	8	100	0.590580
22	10	12	16	100	0.590580

In [124…

```
# 테스트 세트 점수
test_score = gridsearch_svc.score(X_test, y_test)

print("테스트 세트 점수: {:.2f}".format( test_score ))
print("최적 매개변수: {}".format(grid_rf.best_params_))
print("최고 교차 검증 점수: {:.2f}".format(grid_rf.best_score_))
```

```
테스트 세트 점수: 0.69
최적 매개변수: {'max_depth': 6, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_esti
mators': 100}
최고 교차 검증 점수: 0.64
```

In [128…]
```
#위의 결과로 나온 최적 하이퍼 파라미터로 다시 모델을 학습하여 테스트 세트 데이터에서 (

rf = RandomForestClassifier(max_depth= 6,
                            min_samples_leaf= 8,
                            min_samples_split=8,
                            n_estimators= 100,
                            class_weight="balanced",
                            random_state=0)
rf.fit(X_train,y_train)
pred = rf.predict(X_test)
accuracy = accuracy_score(y_test,pred)
accuracy
```

Out[128…]
```
0.7058823529411765
```

In [129…]
```
pred
```

Out[129…]
```
array([0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0])
```

In [130…]
```
from sklearn.metrics import classification_report

print("RandomForest")
print(classification_report(y_test,pred))
```

```
RandomForest
              precision    recall  f1-score   support

           0       0.79      0.71      0.75        31
           1       0.61      0.70      0.65        20

    accuracy                           0.71        51
   macro avg       0.70      0.70      0.70        51
weighted avg       0.72      0.71      0.71        51
```

AdaBoostClassifier

In [47]:
```
params ={
    'n_estimators' : [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 1.0]
}

adaboost = AdaBoostClassifier(random_state=10)
grid_adaboost = GridSearchCV(adaboost, param_grid=params, cv=skf, scoring='accuracy')
best_model = grid_adaboost.fit(X_train,y_train)
```

In [48]:
```
result = pd.DataFrame(grid_adaboost.cv_results_['params'])
result['mean_test_score'] = grid_adaboost.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

Out[48]:

	learning_rate	n_estimators	mean_test_score
34	0.1000	50	0.608333
29	0.0100	100	0.607971
44	1.0000	50	0.607246
35	0.1000	60	0.600000
49	1.0000	100	0.599638
27	0.0100	80	0.599638
30	0.1000	10	0.599638
41	1.0000	20	0.599275
25	0.0100	60	0.591304
28	0.0100	90	0.590942
9	0.0001	100	0.590580
1	0.0001	20	0.590580
2	0.0001	30	0.590580
42	1.0000	30	0.590580
3	0.0001	40	0.590580
38	0.1000	90	0.590580
4	0.0001	50	0.590580
5	0.0001	60	0.590580
6	0.0001	70	0.590580
10	0.0010	10	0.590580
8	0.0001	90	0.590580
7	0.0001	80	0.590580
0	0.0001	10	0.590580
16	0.0010	70	0.590580
11	0.0010	20	0.590580
12	0.0010	30	0.590580
13	0.0010	40	0.590580
14	0.0010	50	0.590580
15	0.0010	60	0.590580
18	0.0010	90	0.590580
17	0.0010	80	0.590580
36	0.1000	70	0.582971
39	0.1000	100	0.582609
26	0.0100	70	0.582246
48	1.0000	90	0.581884
33	0.1000	40	0.581884

	learning_rate	n_estimators	mean_test_score
37	0.1000	80	0.574275
21	0.0100	20	0.573913
19	0.0010	100	0.573913
20	0.0100	10	0.573913
43	1.0000	40	0.573551
45	1.0000	60	0.573551
32	0.1000	30	0.573551
31	0.1000	20	0.573551
24	0.0100	50	0.565217
23	0.0100	40	0.565217
22	0.0100	30	0.565217
40	1.0000	10	0.564855
46	1.0000	70	0.556159
47	1.0000	80	0.530797

In [55]:
```
# 테스트 세트 점수
test_score = grid_adaboost.score(X_test, y_test)

print("테스트 세트 점수: {:.2f}".format( test_score ))
print("최적 매개변수: {}".format(grid_adaboost.best_params_))
print("최고 교차 검증 점수: {:.2f}".format(grid_adaboost.best_score_))
```

```
테스트 세트 점수: 0.73
최적 매개변수: {'learning_rate': 0.1, 'n_estimators': 50}
최고 교차 검증 점수: 0.61
```

In [109…
```
#위의 결과로 나온 최적 하이퍼 파라미터로 다시 모델을 학습하여 테스트 세트 데이터에서 (
from sklearn.tree import DecisionTreeClassifier

adaboost = AdaBoostClassifier(learning_rate = 0.1, n_estimators = 50, random_state=10
adaboost.fit(X_train,y_train)
pred = adaboost.predict(X_test)
accuracy = accuracy_score(y_test,pred)
accuracy
```

Out[109…
```
0.7254901960784313
```

In [110…
```
pred
```

Out[110…
```
array([0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0])
```

In [111…
```
from sklearn.metrics import classification_report

print("AdaBoostClassifier")
print(classification_report(y_test,pred))
```

```
AdaBoostClassifier
              precision    recall  f1-score   support

           0       0.77      0.77      0.77        31
           1       0.65      0.65      0.65        20

    accuracy                           0.73        51
   macro avg       0.71      0.71      0.71        51
weighted avg       0.73      0.73      0.73        51
```

성남시 스쿨존 사고 예측

In [112…

```
seongnam  = pd.read_csv("./data/성남시(완2).csv",encoding='CP949')
seongnam = pd.DataFrame(seongnam)
X = seongnam.iloc[:, 4:13]

# 특성 표준화
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

In [119…

```
#위의 결과로 나온 최적 하이퍼 파라미터로 다시 모델을 학습하여 테스트 세트 데이터에서 (
from sklearn.tree import DecisionTreeClassifier

pred = adaboost.predict(X_std)

pred= pd.DataFrame(pred)
pred
```

Out[119…

	0
0	0
1	0
2	0
3	0
4	0
…	…
68	0
69	0
70	0
71	0
72	1

73 rows × 1 columns