과목명	딥러닝/클라우드
담당교수	오세종 교수님
학과	소프트웨어학과
학번	32153180
이름	이상민
제출일자	2020.10.25

1. Prepare dataset

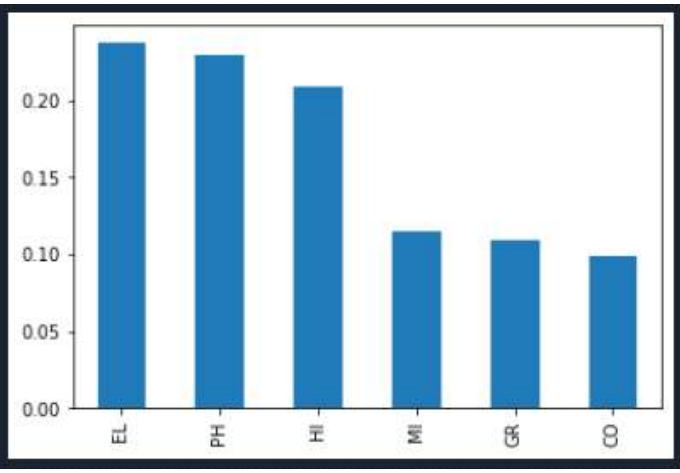
필요한 라이브러리

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import lightgbm as lgb
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from imblearn.over_sampling import SMOTE
```

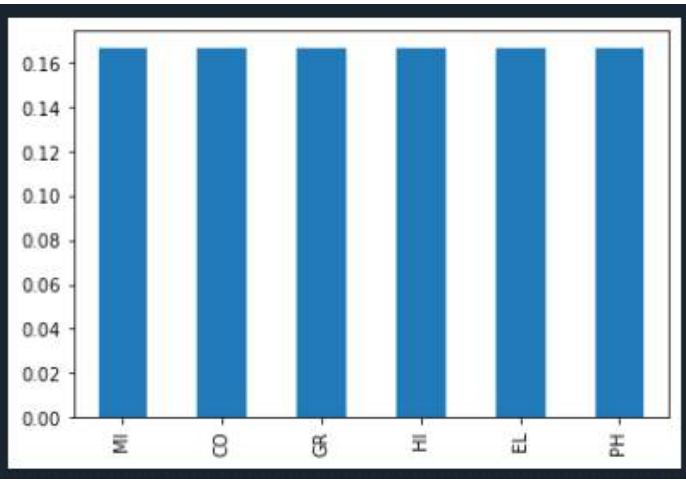
```
tr_data = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드
/competition/trainset.csv',
                      header = None)
ts_data = pd.read_csv('C:/Users/sangmin/Desktop/학교생활/4-2/딥러닝클라우드
/competition/testset.csv',
                      header = None)

tr_X = tr_data.loc[:, tr_data.columns != 0]
tr_y = tr_data[0]
ts_X = ts_data
tr_X.columns = np.arange(31)
ts_X.columns = np.arange(31)

print('train data :', tr_data.shape)
print('test data :', ts_data.shape)
print(tr_y.value_counts())
tr_y.value_counts(normalize=True).plot(kind='bar')

smote = SMOTE(random_state=10, k_neighbors=5)
tr_X_over, tr_y_over = smote.fit_sample(tr_X, tr_y)

print('before SMOTE :', tr_X.shape, tr_y.shape)
print('after SMOTE :', tr_X_over.shape, tr_y_over.shape)
print(tr_y_over.value_counts())
tr_y_over.value_counts(normalize=True).plot(kind='bar')
```

각 클래스의 수가 현저히 차이나는 것은 아니지만 약간의 불균형을 해소하기 위해 SMOTE를 사용했다. 데이터 불균형을 해결하고자 할 때 주로 오버샘플링이나 언더샘플링 기법을 이용한다고 한다. 많은 데이터를 확보할수록 좋다고 판단되어 오버샘플링, 그 중 SMOTE 알고리즘을 이용했다. 오버샘플링 특성 상 과적합될 수 있다는 점은 염두해야 한다.

```
train data : (4280, 32)
test data : (1833, 31)
EL    1017
PH     981
HI     897
MI     492
GR     469
CO     424
```




```
before SMOTE : (4280, 31) (4280,)
after SMOTE : (6102, 31) (6102,)
MI    1017
CO    1017
GR    1017
HI    1017
EL    1017
PH    1017
```



2. Feature selection

- whole features
- filter method
- RFE (backward elimination)

세 가지 방법에 대해 6개의 모델 사용 (전부 default)

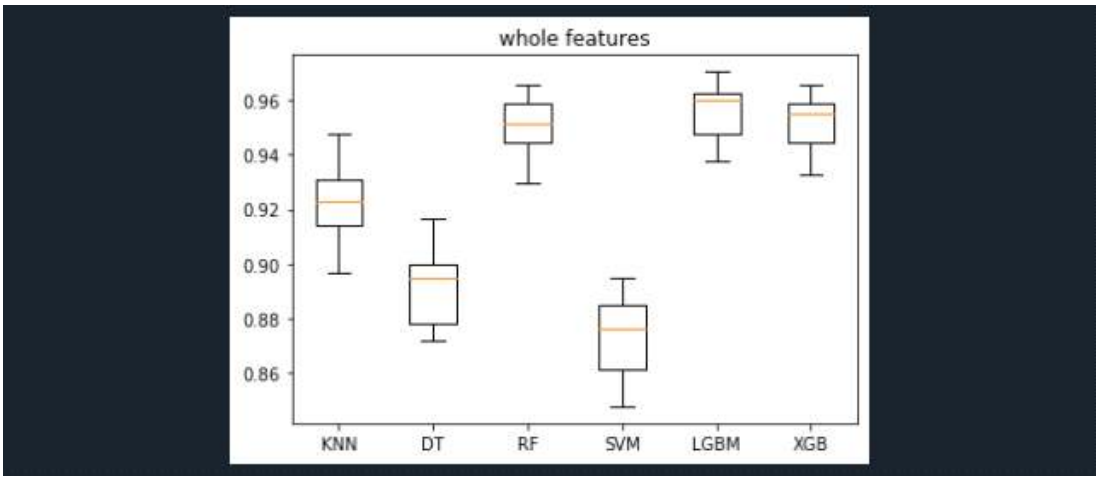
1. kNN
2. Decision Tree
3. Random Forest
4. Support Vector Machine
5. LightGBM
6. Xgboost

whole features

```
models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier(random_state=10)))
models.append(('RF', RandomForestClassifier(random_state=10)))
models.append(('SVM', SVC(random_state=10)))
models.append(('LGBM', LGBMClassifier(random_state=10)))
models.append(('XGB', XGBClassifier(random_state=10)))

whole_results = []
whole_names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=10,
shuffle=True)
    cv_results = model_selection.cross_val_score(model, tr_X_over, tr_y_over,
\
                                                 cv=kfold, scoring='accuracy')
    whole_results.append(cv_results)
    whole_names.append(name)
    print('{} : {} ({})'.format(name, np.round(cv_results.mean(), 4),
np.round(cv_results.std(), 4)))
```

```
KNN : 0.9235 (0.014)
DT : 0.8913 (0.0136)
RF : 0.9499 (0.0115)
SVM : 0.8735 (0.0155)
LGBM : 0.9554 (0.0103)
XGB : 0.9515 (0.0106)
```

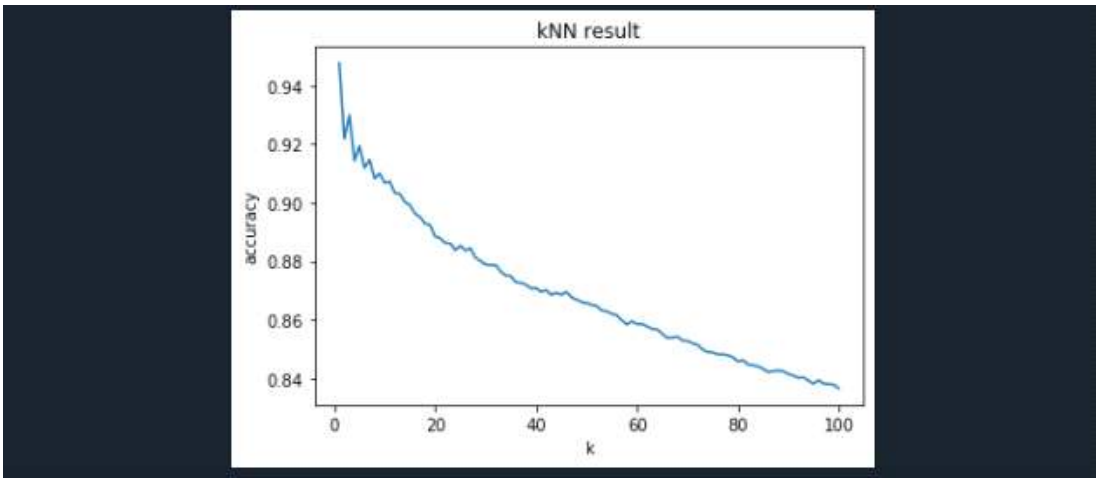
```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('whole features')
plt.boxplot(whole_results)
ax.set_xticklabels(whole_names)
plt.show()
```



정확도가 비슷하게 높게 나오는 Random Forest / LightGBM / Xgboost를 사용해 filter method와 REF 방식으로 feature selection을 진행했다.

kNN도 높은 편이지만 k 값이 증가할수록 정확도가 급격히 낮아져 배제했다.

```
k_list = np.arange(1, 101)
accuracies = []

for k in k_list:
    model = KNeighborsClassifier(n_neighbors=k)
    kfold = model_selection.KFold(n_splits=5, random_state=10, shuffle=True)
    cv_results = model_selection.cross_val_score(model, tr_X_over, tr_y_over, scoring='accuracy')
    accuracies.append(cv_results.mean())

plt.plot(k_list, accuracies)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.title('kNN result')
plt.show()
```



filter method

```
test = SelectKBest(score_func=chi2, k=tr_X_over.shape[1])
fit = test.fit(tr_X_over, tr_y_over)
print(np.round(fit.scores_, 3))
f_order = np.argsort(-fit.scores_)
sorted_columns = tr_X_over.columns[f_order]

models = []
models.append(('RF', RandomForestClassifier(random_state=10)))
models.append(('LGBM', LGBMClassifier(random_state=10)))
models.append(('XGB', XGBClassifier(random_state=10)))

filter_results = []
filter_names = []
for name, model in models:
    print('<< ' + name + ' >>')
    for i in range(1, tr_X_over.shape[1] + 1):
        fs = sorted_columns[0:i]
        tr_X_selected = tr_X_over[fs]
        scores = cross_val_score(model, tr_X_selected, tr_y_over, cv=5)

        print(fs.tolist())
        print(np.round(scores.mean(), 4))
    filter_results.append(scores)
    filter_names.append(name)
```

Random Forest max accuracy

```
[14, 17, 24, 28, 20, 7, 15, 4, 12, 26, 22, 30, 9, 2, 13, 16, 10, 23, 27, 19, 6, 18,
11, 5, 25, 8, 21]
0.9487
```

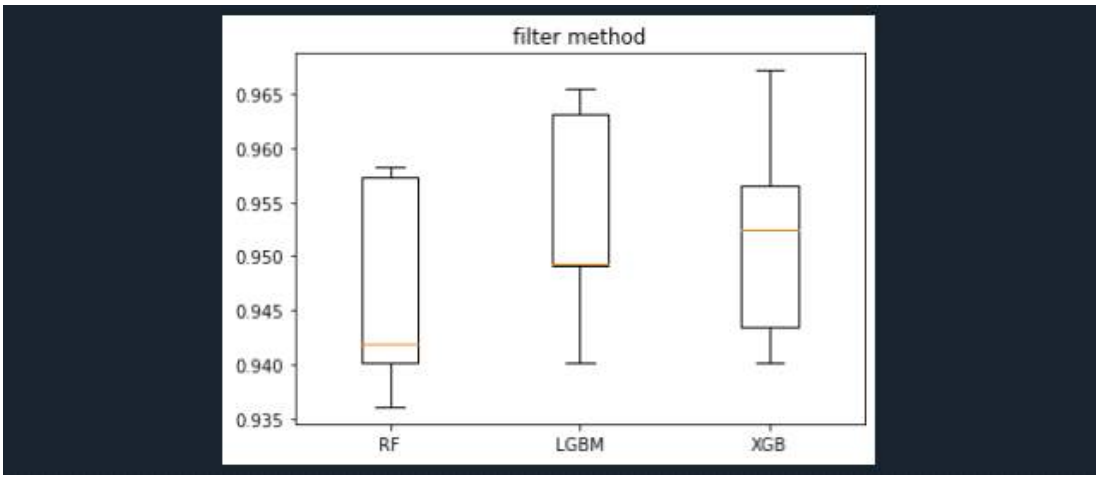
lightGBM max accuracy

```
[14, 17, 24, 28, 20, 7, 15, 4, 12, 26, 22, 30, 9, 2, 13, 16, 10, 23, 27, 19, 6, 18,
11, 5, 25, 8, 21, 29, 3, 1]
0.9541
```

Xgboost forest max accuracy

```
[14, 17, 24, 28, 20, 7, 15, 4, 12, 26, 22, 30, 9, 2, 13, 16, 10, 23, 27, 19, 6, 18,
11, 5, 25, 8, 21, 29, 3, 1, 0]
0.952
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('filter method')
plt.boxplot(filter_results)
ax.set_xticklabels(filter_names)
plt.show()
```



RFE

```
models = []
models.append(('RF', RandomForestClassifier(random_state=10)))
models.append(('LGBM', LGBMClassifier(random_state=10)))
models.append(('XGB', XGBClassifier(random_state=10)))

rfe_results = []
rfe_names = []
number = [25, 26, 27, 28, 29, 30, 31]
for name, model in models:
    print('〈〈 ' + name + ' 〉〉')
    for n in number:
        rfe = RFE(model, n_features_to_select=n)
        fit = rfe.fit(tr_X_over, tr_y_over)
        fs = tr_X_over.columns[fit.support_].tolist()
        scores = cross_val_score(model, tr_X_over[fs], tr_y_over, cv=5)
        print(fs)
        print(np.round(scores.mean(), 4))
    rfe_results.append(scores)
    rfe_names.append(name)
```

Random Forest max accuracy

```
[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24,
25, 26, 27, 28, 29, 30]
0.9479
```

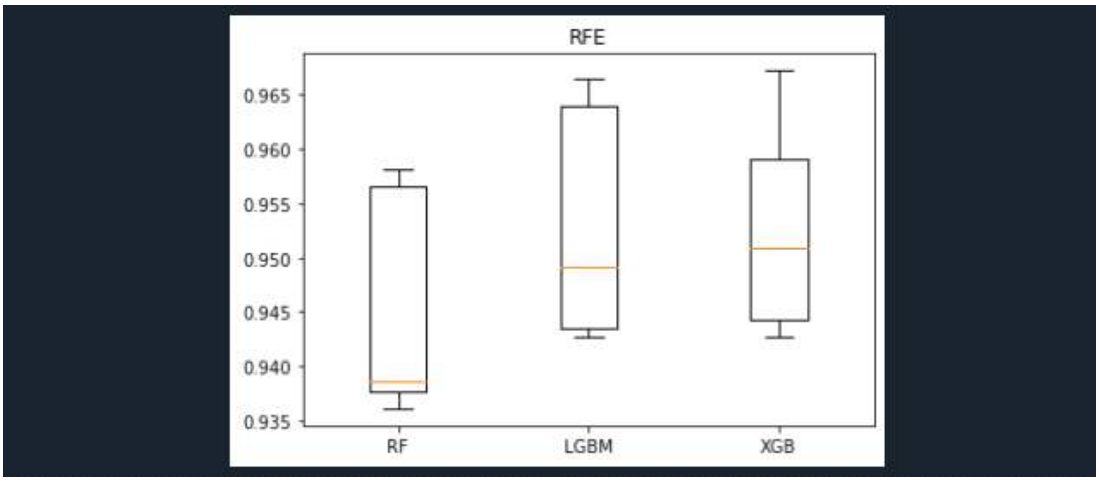
lightGBM max accuracy

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23,
24, 25, 26, 27, 28, 30]
0.9558
```

Xgboost forest max accuracy

```
[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25, 26, 27, 28, 29, 30]
0.9538
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_title('RFE')
plt.boxplot(rfe_results)
ax.set_xticklabels(rfe_names)
plt.show()
```



	whole features	filter method	RFE
Random Forest	0.9499	0.9487	0.9479
LightGBM	0.9554	0.9541	0.9558
Xgboost	0.9515	0.952	0.9538

확연한 차이를 보여주진 않지만 Random Forest는 전체 변수를 모두 사용하고 LightGBM과 Xgboost에서는 각각 RFE 결과로 나온 변수를 사용하는 것이 더 나은 모델을 만들 것이라 예상한다.

3. Algorithm selection

- Random Forest : whole features (31개)
- LightGBM : RFE로 택한 변수 (29개)
- Xgboost : RFE로 택한 변수 (30개)

default로 측정한 세 알고리즘의 정확도 차이가 매우 근소해 세 알고리즘 모두 hyper parameter tuning 과정을 진행해야겠다고 판단했다.

```
lgbm_idx = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 22, 23, 24, 25, 26, 27, 28, 30]
xgb_idx = [0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

tr_X_rf = tr_X_over
tr_X_lgbm = tr_X_over[lgbm_idx]
tr_X_xgb = tr_X_over[xgb_idx]

train_X_rf, test_X_rf, train_y_rf, test_y_rf = \
    train_test_split(tr_X_rf, tr_y_over, test_size=0.3, random_state=100)
train_X_lgbm, train_y_lgbm, test_X_lgbm, test_y_lgbm = \
    train_test_split(tr_X_lgbm, tr_y_over, test_size=0.3, random_state=100)
train_X_xgb, train_y_xgb, test_X_xgb, test_y_xgb = \
    train_test_split(tr_X_xgb, tr_y_over, test_size=0.3, random_state=100)
```

다음 장의 n_estimators=100 모델의 정확도와 비교하겠다.

```
rf_model = RandomForestClassifier(random_state=100, n_estimators=100)
rf_model.fit(train_X_rf, train_y_rf)
pred = rf_model.predict(test_X_rf)

lgbm_model = LGBMClassifier(random_state=100, n_estimators=100)
lgbm_model.fit(train_X_lgbm, train_y_lgbm)
pred = lgbm_model.predict(test_X_lgbm)

xgb_model = XGBClassifier(random_state=100, n_estimators=100)
xgb_model.fit(train_X_xgb, train_y_xgb)
pred = xgb_model.predict(test_X_xgb)

print('100 estimators RF   :', np.round(accuracy_score(pred, test_y_rf), 4))
print('100 estimators LGBM :', np.round(accuracy_score(pred, test_y_lgbm),
4))
print('100 estimators XGB  :', np.round(accuracy_score(pred, test_y_xgb),
4))
```

```
100 estimators RF   : 0.9345
100 estimators LGBM : 0.9416
100 estimators XGB  : 0.9421
```

4. Hyper parameter tuning

Random Forest

```
rf_param_grid = {
    'n_estimators' : [200, 300],
    'max_depth' : [16, 18, 20, 22],
    'min_samples_leaf' : [1, 3, 5],
    'min_samples_split' : [2, 4, 6]
    }

rf_grid = GridSearchCV(rf_model, param_grid=rf_param_grid,
scoring='accuracy', n_jobs=-1, verbose=1, cv=5)
rf_grid.fit(train_X_rf, train_y_rf)
print(np.round(rf_grid.best_score_, 4))
rf_grid.best_params_

arr_X_rf = np.array(tr_X_rf)
arr_y_rf = np.array(tr_y_over)
rf_new_model = RandomForestClassifier(random_state=100,
n_estimators=200,
                                      max_depth=18, min_samples_leaf=1,
min_samples_split=2)
kf = KFold(n_splits=5, random_state=100, shuffle=True)

acc = np.zeros(5)
i = 0
for train_index, test_index in kf.split(arr_X_rf):
    train_X, test_X = arr_X_rf[train_index], arr_X_rf[test_index]
    train_y, test_y = arr_y_rf[train_index], arr_y_rf[test_index]

    rf_new_model.fit(train_X, train_y)

    pred = rf_new_model.predict(test_X)

    acc[i] = accuracy_score(test_y, pred)
    print('{}-fold accuracy : {:.4f}'.format(i, acc[i]))
    i += 1

print('mean of accuracy : {:.4f}'.format(np.mean(acc)))
```

```
In [221]:
     ...: print(np.round(rf_grid.best_score_, 4))
     ...: rf_grid.best_params_
0.9344
Out[221]:
{'max_depth': 18,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
0-fold accuracy : 0.9345
1-fold accuracy : 0.9353
2-fold accuracy : 0.9369
3-fold accuracy : 0.9459
4-fold accuracy : 0.9475
mean of accuracy : 0.9400
```

LightGBM

```
lgbm_param_grid = {
    'learning_rate' : [0.01, 0.05, 0.1],
    'num_iterations' : [100, 200],
    'max_depth' : [12, 14, 16],
    'num_leaves' : [40, 60, 80],
    }

lgbm_grid = GridSearchCV(lgbm_model, param_grid=lgbm_param_grid,
scoring='accuracy', n_jobs=-1, verbose=1, cv=5)
lgbm_grid.fit(train_X_lgbm, train_y_lgbm)
print(np.round(lgbm_grid.best_score_, 4))
lgbm_grid.best_params_

arr_X_lgbm = np.array(tr_X_lgbm)
arr_y_lgbm = np.array(tr_y_over)
lgbm_new_model = LGBMClassifier(random_state=100, n_estimators=200,
max_depth=16, num_leaves=40,
                                objective='multiclass', metric='multi_logloss')
kf = KFold(n_splits=5, random_state=100, shuffle=True)

acc = np.zeros(5)
i = 0
for train_index, test_index in kf.split(arr_X_lgbm):
    train_X, test_X = arr_X_lgbm[train_index], arr_X_lgbm[test_index]
    train_y, test_y = arr_y_lgbm[train_index], arr_y_lgbm[test_index]

    lgbm_new_model.fit(train_X, train_y)

    pred = lgbm_new_model.predict(test_X)

    acc[i] = accuracy_score(test_y, pred)
    print('{}-fold accuracy : {:.4f}'.format(i, acc[i]))
    i += 1

print('mean of accuracy : {:.4f}'.format(np.mean(acc)))
```

```
In [206]: print(np.round(lgbm_grid.best_score_, 4))
0.9457

In [207]: lgbm_grid.best_params_
Out[207]:
{'learning_rate': 0.1,
 'max_depth': 16,
 'num_iterations': 200,
 'num_leaves': 40}
```

```
0-fold accuracy : 0.9517
1-fold accuracy : 0.9451
2-fold accuracy : 0.9451
3-fold accuracy : 0.9525
4-fold accuracy : 0.9516
mean of accuracy : 0.9492
```

Xgboost

```
xgb_param_grid = {
    'learning_rate' : [0.01, 0.05, 0.1],
    'num_iterations' : [100, 200, 300],
    'min_child_weight': [1, 3, 5, 7],
    'max_depth': [2, 4, 6, 8],
    'gamma': [0, 1, 2]
    }

xgb_grid = GridSearchCV(xgb_model, param_grid=xgb_param_grid,
scoring='accuracy', n_jobs=-1, verbose=1, cv=5)
xgb_grid.fit(train_X_xgb, train_y_xgb)
print(np.round(xgb_grid.best_score_, 4))
xgb_grid.best_params_

arr_X_xgb = np.array(tr_X_xgb)
arr_y_xgb = np.array(tr_y_over)
xgb_new_model = XGBClassifier(random_state=100, n_estimators=100,
max_depth=10,
                              learning_rate=0.1, min_child_weight=1)
kf = KFold(n_splits=5, random_state=100, shuffle=True)

acc = np.zeros(5)
i = 0
for train_index, test_index in kf.split(arr_X_xgb):
    train_X, test_X = arr_X_xgb[train_index], arr_X_xgb[test_index]
    train_y, test_y = arr_y_xgb[train_index], arr_y_xgb[test_index]

    xgb_new_model.fit(train_X, train_y)

    pred = xgb_new_model.predict(test_X)

    acc[i] = accuracy_score(test_y, pred)
    print('{}-fold accuracy : {:.4f}'.format(i, acc[i]))
    i += 1

print('mean of accuracy : {:.4f}'.format(np.mean(acc)))
```

```
In [216]: print(np.round(xgb_grid.best_score_, 4))
0.9363

In [217]: xgb_grid.best_params_
Out[217]: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 1}
```

```
0-fold accuracy : 0.9459
1-fold accuracy : 0.9476
2-fold accuracy : 0.9525
3-fold accuracy : 0.9549
4-fold accuracy : 0.9541
mean of accuracy : 0.9510
```

	Random Forest	LightGBM	Xgboost
accuracy	0.9400	0.9492	0.9510

예측 결과를 30번까지 제출할 수 있어 세 모델 모두 제출해보니 LightGBM 정확도가 가장 높게 나왔다. 그래서 이 모델을 중점적으로 다시 튜닝했다.

5. Model evaluation

acc: 0.925764192139738

결국 정확도를 더 끌어올리는 데 실패했다. GridSearchCV나 RandomizedSearch CV를 사용하여 최적의 파라미터를 찾고자 노력했지만 번번히 낮아졌다. 5-fold 교차검증을 하면 정확도가 높아질 때도 있었으나 커널에 올리면 감소하는 걸 보니 과적합이 발생한 것 같다.

6. 소감

우선 아쉬움이 굉장히 많이 남은 경진대회였다. 나름대로 여러 모델을 사용해 거르고 걸러 결과를 제출했다고 생각했지만 마음에 드는 성적을 내지 못했다. 그래도 더 나은 모델을 도출하기 위해 자료를 찾아보고, 공부한 점이 뜻깊었다.

차라리 전처리 부분에서 힘을 덜 쓰고 파라미터 튜닝에 매진했으면 어땠을까 싶기도 하다. 튜닝하고 싶은 파라미터는 많은데 GridSearchCV에 모두 넣어 돌리려니 시간이 정말 오래 걸렸다. 왜 머신러닝 전문가들이 좋은 데스크탑 환경을 갖추고 테스트에 임하는지 알 수 있었다.

교수님께서 수업 시간에 말씀하신 것처럼 항상 최고의 정확도를 도출하는 알고리즘은 존재하지 않았다. 많은 사람들이 내가 사용한 Xgboost나 LightGBM이 굉장히 강력한 부스팅 알고리즘이라고 말하지만 결과적으로 높은 정확도를 얻지 못했으니 말이다. 앞으로 이러한 경진대회를 다시 경험할 수 있을지는 잘 모르겠지만, 만약 하게된다면 지금보다 더 쉽게 해결해낼 수 있을 것 같다. 그런 의미에서 이번 경진대회는 직접 머신러닝을 사용할 수 있게끔 좋은 틀을 만들어줬다.