2020-2 STA617 고급통계적머신러닝

FINAL REPORT

ANALYSIS REGARDING AN IMBALANCED DATA

2020021218 통계학과 이소연

erinlee42@naver.com

1. Introduction

1,1. 레포트 목적 및 개요

데이터 불균형(Class Imbalance)은 특정 데이터에서 각 범주가 갖고 있는 데이터의 양에 큰 차이가 있는 경우를 말한다. 실생활 속 이러한 불균형 데이터는 다양하게 존재한다. 본 프로젝트는 가상의 시뮬레이션을 통해 데이터 불균형 해소의 중요성을 알아보고, 과소표집과 과대표집을 적용한 각 새로운 데이터에 다양한 학습 모형을 적용하여, 재표집(Resampling) 방법의 차이를 비교하고, 최고성능을 도출하는 모형을 찾을 것이다.

1.2. 데이터 소개

우선, 다음과 같이 불균형 데이터를 생성하였다. 설명변수 X는 10개, 목적변수 y의 범주는 4개로, 각 5%, 10%, 15%, 그리고 70%의 비중을 갖도록 하였다. 표본크기는 1000으로 설정했다.

```
X,y = make_classification(n_samples=1000, n_redundant=0, n_repeated=0, n_classes=4, n_clusters_per_class=1,
                          weights=[0.05, 0.10, 0.15, 0.7], n_features=10, random_state=100)
print('Original Data %s' % Counter(y))
```

```
Original Data Counter({3: 692, 2: 154, 1: 101, 0: 53})
```

1.3. 데이터 불균형 해소 방법

데이터 불균형을 해소하기 위한 재표집 방법은 크게 두 가지가 있다. 바로, 데이터를 과소표집(Under sampling) 하는 방법과 과대표집(Over sampling) 하는 방법이다. 우선, 다음과 같은 과정을 거쳐 표본의 크기가 가장 작은 네 번째 클래스의 크기에 맞게, 다수 클래스(major class)의 표본을 임의로(randomly) 줄였다. 그 결과, 표본의 크기가 기존 1000개에서 212개로 줄었다.

```
rus = RandomUnderSampler(random_state=100)
rus.fit(X, y)
X_usampled, y_usampled = rus.fit_resample(X, y)
print('Resampled Data from UnderSampling %s' % Counter(y_usampled))
print('X shape:',X_usampled.shape,", y shape:",y_usampled.shape)
```

```
Resampled Data from UnderSampling Counter({0: 53, 1: 53, 2: 53, 3: 53})
X shape: (212, 10) , y shape: (212,)
```

다음으로, 대표적인 과대표집 방법인 합성소수표집법(SMOTE)와 조절합성표집법(ADASYN)을 통해, 표본의 크기가 가장 큰 첫 번째 클래스의 크기에 맞춰, 소수클래스(minor class)의 표본을 증식하였다. KNN의 초모수인 k는 5로 설정하였으며, 그 결과는 다음장에 첨부하였다. SMOTE 경우, 각 범주가 동일한 크기가 되어, 총 표본의 크기가 2768로 늘어났다. 반면에 ADASYN의 경우, 각 소수클래스의 KNN 셋에 포함된 다수 클래스의 표본 수에 비례하게 추출되었기 때문에, 전체 표본의 크기가 SMOTE보단 조금 작은 2758로 늘어났다.

```
sm=SMOTE(k_neighbors=5, random_state=100)
X_sm, y_sm = sm.fit_resample(X,y)
ada=ADASYN(n_neighbors=5, random_state=100)
X_ada, y_ada = ada.fit_resample(X,y)
print('Resampled Data from SMOTE OverSampling %s' % Counter(y_sm))
print('X shape:',X_sm.shape,", y shape:",y_sm.shape)
print('Resampled Data from ADASYN OverSampling %s' % Counter(y_ada))
print('X shape:',X_ada.shape,", y shape:",y_ada.shape)
```

```
Resampled Data from SMOTE OverSampling Counter({2: 692, 3: 692, 1: 692, 0: 692})
X shape: (2768, 10) , y shape: (2768,)
Resampled Data from ADASYN OverSampling Counter({1: 694, 3: 692, 2: 689, 0: 683})
X shape: (2758, 10) , y shape: (2758,)
```

1.4. 진행방향

시뮬레이션 데이터이기 때문에 표준화 및 차원 축소 과정은 생략하였다. 다음으로, 앞선 세 가지 재표집 기법을 적용한 데이터와 원 데이터를 각각 7:3의 Train 데이터와 Test 데이터로 나누었다. 이는 Test 데이터의 분류 성능을 통해 데이터 불균형 해소 전후와, 각 재표집 방법 및 학습 모형을 비교하기 위해서이다. 모형 비교에 사용할 성능 평가 측도는 정확도(Accuracy)와, 데이터 구조가 불균형일 때 주로 사용되는 재현율(Recall), 그리고 F1-값의 세가지이다. 이 때 정확도의 경우, nested cross-validation에 의해 구한 Test 데이터의 평균 Accuracy로 도출할 것이다.

2. Model Fitting

본 프로젝트에서 비교할 학습 모형은 로지스틱 회귀(Logistic Regression), 선형판별분석(LDA), 서포트 벡터 머신(SVM), 랜덤 포레스트(Random Forest), 그리고 XGB 부스팅(Extreme Gradient Boosting)이다. 각 모형의 초모수는 5-fold CV 의 GridSearch 를 거쳐 선택할 것이다. 또한, 시뮬레이션 데이터를 사용했기 때문에, 자료 타입에 민감여부, 해석의 용의성 등을 배제한 오직 모형의 분류 성능을 기준으로 최종 모형을 선택할 것이다.

2.1. Logistic Regression

Y 의 범주가 두 개 이상이므로, One-versus-Rest 의 접근법을 통해 로지스틱 회귀모형을 적합하였다. 또한, 모형의 과대적합(Over Fitting)을 방지하기 위해 L2 규제화(Regularization)인 릿지(Ridge) 페널티를 사용하였다. 아래와 같은 함수를 이용하여 각 네 경우별 정확도, 재현율, F1-값, 그리고 test 데이터의 오차행렬을 도출하였다.

```
def logistic_cv(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
    param_grid_logistic={'C':[1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3]}
    logistic_cv=GridSearchCV(logisticRegression(penalty = 'l2'), param_grid_logistic, cv=5)
    logistic_cv.fit(X_train,y_train)
    print(logistic_cv.best_params_)
    logistic_train=logistic_cv.predict(X_train)
    logistic_test=logistic_cv.predict(X_test)
    print("Train:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_train,logistic_train),2),'   /' ,
          round(recall_score(y_train, logistic_train, average='macro'),2),'   /' ,
          round(f1_score(y_train,logistic_train, average='macro'),2))
    print("Test:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_test,logistic_test),2),'   /' ,
          round(recall_score(y_test, logistic_test, average='macro'),2),'   /' ,
          round(f1_score(y_test,logistic_test, average='macro'),2))
    scores = cross_val_score(logistic_cv, X, y, cv=5)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

먼저, 기존 데이터(Raw data)를 적합하였다. 그 결과, 아래와 같이 최상의 초모수값 $C(=1/\lambda)$는 GridSearchCV 에 의해 10 으로 결정되었다. 이는 불균형 데이터에 대한 결과이므로 Accuracy 로 성능을 판단할 수 없으며, F1 score 기준 약 76%의 성능을 보이는 것으로 판단하였다.

```
logistic_cv(X,y)
```

```
{'C': 10.0}
Train:
 Accuracy / Recall / F1 score
 0.88    / 0.75    / 0.8
Test:
 Accuracy / Recall / F1 score
 0.85    / 0.73    / 0.76
CV accuracy: 0.860 +/- 0.030
```

동일한 함수를 이번엔 과소표집 샘플에 적용하였다. 그 결과, 아래와 같이 초모수 $C(=1/\lambda)$가 위와 다른 0.1 로 결정되었다. 또한, 각 범주를 해당 범주로 알맞게 예측할 확률인 Recall Score 가 앞선 경우보다 눈에 띄게 높아졌다. 이는 소수클래스의 사례가 상대적으로 너무 작았던 기존 데이터의 한계를 본 재표집 방법이 보완했기 때문이라고 볼 수 있다.

```
logistic_cv(X_usampled,y_usampled)
```

```
{'C': 0.1}
Train:
 Accuracy / Recall / F1 score
 0.8    / 0.79    / 0.79
Test:
 Accuracy / Recall / F1 score
 0.78    / 0.8    / 0.79
CV accuracy: 0.751 +/- 0.072
```

SMOTE 과 ADASYN 으로 과대표집을 한 표본의 경우, 초모수는 각각 0.1 과 1 로 결정되었다. Recall 점수가 위 경우보다도 더 높게 나왔으며, 그 중 특히 SMOTE 표본이 높은 성능을 보였다. 이는 표본의 수를 줄여, 모형의 정밀도를 떨어뜨리는 과소표집의 한계를 보완했기 때문이다.

```
logistic_cv(X_sm, y_sm)
```

```
{'C': 0.1}
Train:
 Accuracy / Recall / F1 score
 0.85    / 0.85    / 0.85
Test:
 Accuracy / Recall / F1 score
 0.82    / 0.83    / 0.83
CV accuracy: 0.839 +/- 0.022
```

SMOTE(위) / ADASYN(아래)

```
logistic_cv(X_ada, y_ada)
```

```
{'C': 1}
Train:
 Accuracy / Recall / F1 score
 0.8    / 0.8    / 0.8
Test:
 Accuracy / Recall / F1 score
 0.79    / 0.8    / 0.8
CV accuracy: 0.787 +/- 0.019
```

2.2. Linear Discriminant Analysis

이제 분산-공분산 행렬이 범주에 관계없이 일정하고, X 가 다변량 정규분포를 따른다고 가정하여 선형판별분석(LDA)을 적용하겠다. 이 모형은 앞선 가정 이외의 초모수를 사용하지 않으므로, GridSearch 를 수행하지 않았으며, 다음과 같은 함수를 작성하였다.

```
def lda(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
    lda = LinearDiscriminantAnalysis(store_covariance=False)
    lda.fit(X_train,y_train)
    lda_train=lda.predict(X_train)
    lda_test=lda.predict(X_test)
    print("Train:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_train,lda_train),2),'   /' ,
          round(recall_score(y_train, lda_train, average='macro'),2),'   /' ,
          round(f1_score(y_train,lda_train, average='macro'),2))
    print("Test:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_test,lda_test),2),'   /' ,
          round(recall_score(y_test, lda_test, average='macro'),2),'   /' ,
          round(f1_score(y_test,lda_test, average='macro'),2))
    scores = cross_val_score(lda, X, y, cv=5)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

먼저, 기존 데이터(Raw data)를 적합하였다. 그 결과, 아래와 같이 앞선 로지스틱 회귀모형보다 전체적으로 낮은 성능이 기록되었다. 특히, 굉장히 낮은 Recall 과 F1 Score 를 보였다.

```
lda(X,y)
```

```
Train:
 Accuracy / Recall / F1 score
 0.84    / 0.66    / 0.68
Test:
 Accuracy / Recall / F1 score
 0.82    / 0.63    / 0.65
CV accuracy: 0.832 +/- 0.025
```

다음으로 과소표집 샘플의 경우, 앞선 결과보다 전체적인 성능이 향상되긴 하였으나, 아직도 로지스틱 회귀모형보단 낮은 결과를 보였다. 하지만 마찬가지로, 데이터 불균형 문제가 해소되어 앞선 결과보다 Recall 과 F1 Score 가 매우 상향되었다.

```
lda(X_usampled, y_usampled)
```

```
Train:
 Accuracy / Recall / F1 score
 0.78    / 0.76    / 0.76
Test:
 Accuracy / Recall / F1 score
 0.75    / 0.77    / 0.76
CV accuracy: 0.727 +/- 0.050
```

과대표집 샘플의 경우, 마찬가지로 과소표집의 한계를 보완했기 때문에, 재현율이 보다 상승하였다. 앞선 로지스틱 회귀모형의 경우와 같이, LDA 또한 SMOTE 가 ADASYN 보다 전체적으로 더 좋은 성능을 보였다. 다음은 차례대로 SMOTE 와 ADASYN 표본의 결과이다.

```
lda(X_sm, y_sm)
```

```
Train:
 Accuracy / Recall / F1 score
 0.8    / 0.8    / 0.79
Test:
 Accuracy / Recall / F1 score
 0.8    / 0.8    / 0.8
CV accuracy: 0.798 +/- 0.019
```

```
lda(X_ada, y_ada)
```

```
Train:
 Accuracy / Recall / F1 score
 0.77    / 0.77    / 0.77
Test:
 Accuracy / Recall / F1 score
 0.76    / 0.77    / 0.76
CV accuracy: 0.757 +/- 0.030
```

2.3. Support Vector Machine

다음으로, 비선형 분류선(kernel)과 support vector 를 통해 데이터를 분류하는 서포트 벡터 머신 모형을 적합하였다. 마찬가지로, 과대적합을 방지하기 위해 각 표본별로 최선의 'c'와 'gamma'값을 찾기 위한 GridSearchCV 를 수행하였다.

```
def svm_cv(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
    param_grid_svm={'C':[10**(i+1) for i in range(-3,2)], 'gamma':[10**(i+1) for i in range(-3,2)],'kernel':['rbf','sigmoid']}
    svm_cv=GridSearchCV(SVC(random_state=1206), param_grid_svm, cv=5)
    svm_cv.fit(X_train,y_train)
    print(svm_cv.best_params_)
    svm_train=svm_cv.predict(X_train)
    svm_test=svm_cv.predict(X_test)
    print("Train:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_train,svm_train),2),'   /' ,
          round(recall_score(y_train, svm_train, average='macro'),2),'   /' ,
          round(f1_score(y_train,svm_train, average='macro'),2))
    print("Test:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_test,svm_test),2),'   /' ,
          round(recall_score(y_test, svm_test, average='macro'),2),'   /' ,
          round(f1_score(y_test,svm_test, average='macro'),2))
    scores = cross_val_score(svm_cv, X, y, cv=5)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

먼저, 기존 데이터(Raw data)를 적합하였다. 최상의 초모수값으로 'c'는 10, gamma 는 0.01, kernel 은 'rbf'로 설정되었다. 적합 결과, 아래와 같이 앞선 두 모형보다 전체적으로 성능이 좋았다. 특히, 불균형이 있음에도 상대적으로 높은 Recall 과 F1 Score 를 보였다.

```
svm_cv(X,y)
```

```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Train:
 Accuracy / Recall / F1 score
 0.9    / 0.79    / 0.85
Test:
 Accuracy / Recall / F1 score
 0.9    / 0.78    / 0.84
CV accuracy: 0.887 +/- 0.028
```

다음으로 과소표집 샘플의 경우, 앞선 결과보다 Test 데이터에 대한 전체적인 성능이 향상되었지만, CV accuracy 로 본 모형의 성능은 로지스틱 회귀분석의 성능보다 낮았다. 하지만 이 경우 또한 데이터 불균형 문제가 해소되니, 앞선 원 데이터의 결과보다 재현율이 향상되었다.

```
svm_cv(X_usampled, y_usampled)
```

```
{'C': 10, 'gamma': 0.01, 'kernel': 'sigmoid'}
Train:
 Accuracy / Recall / F1 score
 0.8    / 0.8    / 0.8
Test:
 Accuracy / Recall / F1 score
 0.84    / 0.85    / 0.85
CV accuracy: 0.745 +/- 0.065
```

과대표집 샘플의 경우, ADASYN 을 사용한 표본의 초모수값 c 가 100 으로 커졌으며, 두 경우 모두 Train 데이터에 한해선 완벽한 성능을 보였고, Test 데이터에 대한 성능 또한 상당히 높은 걸로 보아 과대적합 문제 또한 일어나지 않았다. 앞선 과소표집 방법의 한계를 보완했기 때문에, Recall과 F1 Score가 직전에 비해 매우 상승하였다. 단일 결과로만 보았을 땐 ADASYN의 성능이 SMOTE 보다 조금 더 좋아 보였지만, CV accuracy를 확인하면 다른 모형들과 동일하게, SMOTE가 ADASYN 보다 높은 정확도를 보임을 알 수 있었다. 아래는 차례대로 SMOTE 와 ADASYN 표본의 결과이다.

```
svm_cv(X_sm, y_sm)
```

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.95    / 0.95    / 0.95
CV accuracy: 0.978 +/- 0.008
```

```
svm_cv(X_ada, y_ada)
```

```
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.96    / 0.96    / 0.96
CV accuracy: 0.960 +/- 0.010
```

2.4. Random Forest

다음으로, 앙상블(Ensemble)기법의 일종으로, 의사결정나무의 배깅(Bagging)버전인 Random forest 모형을 적합하였다. 각 표본마다 GridSearchCV 를 통해 불순도 측도인 'criterion'과 특성변수의 개수인(k) 'max_features'를 설정하였다. 반복 수인 M 은 가능한 크게 선택되도록, 제한을 두지 않았다. 사용한 함수는 아래와 같다.

```
def rf_cv(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
    param_grid_rf={'criterion':["gini","entropy"] ,'max_features':['sqrt','log2',X_train.shape[1]]}
    rf_cv=GridSearchCV(RandomForestClassifier(random_state=100), param_grid_rf, cv=5)
    rf_cv.fit(X_train,y_train)
    print(rf_cv.best_params_)
    rf_train=rf_cv.predict(X_train)
    rf_test=rf_cv.predict(X_test)
    print("Train:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_train,rf_train),2),'   /' ,
          round(recall_score(y_train, rf_train, average='macro'),2),'   /' ,
          round(f1_score(y_train,rf_train, average='macro'),2))
    print("Test:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_test,rf_test),2),'   /' ,
          round(recall_score(y_test, rf_test, average='macro'),2),'   /' ,
          round(f1_score(y_test,rf_test, average='macro'),2))
    scores = cross_val_score(rf_cv, X, y, cv=5)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

먼저, 기존 데이터(Raw data)를 적합하였다. 불순도 측도는 지니지수, $k = \sqrt{d}$ 로 설정되었다. 적합 결과는 다음 장에 첨부하였다. 처음으로 원 데이터의 Train 데이터에 대해 완벽한 성능이 관측되었으며, Test 데이터에 대한 성능 또한 약 87%로(F1 Score), 앞선 모델 중 가장 높은 결과를 보였다. Recall 지수 또한 앞선 SVM 보다도 높은 결과가 도출되었음을 확인할 수 있다.

```
rf_cv(X,y)
```

```
{'criterion': 'gini', 'max_features': 'sqrt'}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.92    / 0.82    / 0.87
CV accuracy: 0.919 +/- 0.024
```

다음으로 과소표집 샘플의 경우, 앞선 결과보다 Test 데이터에 대한 전체적인 성능이 오히려 떨어졌다. 특히 F1 score 의 경우, 확연히 저하된 결과(약 83%)를 보였으며, 표본 크기의 감소가 본 모형에 매우 부정적인 영향을 미친 것으로 판단되었다. Test 데이터의 결과에 비해 Train 데이터에 대해서만 완벽한 성능을 갖는 것으로 보아, 과대적합(Overfitting)이 의심되었다.

```
rf_cv(X_usampled,y_usampled)
```

```
{'criterion': 'entropy', 'max_features': 10}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.81    / 0.82    / 0.83
CV accuracy: 0.783 +/- 0.039
```

과대표집 샘플의 경우, 데이터 불균형 문제와 과소표집의 한계를 보완했기 때문에, Test 데이터에 대한 전체적인 성능이 상승하였다. 앞선 SVM 과 같이, Random Forest 또한 단일 결과로만 보았을 땐 ADASYN 의 성능이 SMOTE 보다 조금 더 좋아 보이지만, CV accuracy 를 확인하면 역시 SMOTE 가 ADASYN 보다 높은 정확도를 보임을 알 수 있었다. SVM 모형과 유사하게, 전체적인 분류 성능이 매우 우수했다. 아래는 각 SMOTE와 ADASYN 표본의 결과이다.

```
rf_cv(X_sm,y_sm)
```

```
{'criterion': 'entropy', 'max_features': 10}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.95    / 0.95    / 0.95
CV accuracy: 0.962 +/- 0.011
```

```
rf_cv(X_ada,y_ada)
```

```
{'criterion': 'gini', 'max_features': 'sqrt'}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.97    / 0.97    / 0.97
CV accuracy: 0.938 +/- 0.032
```

2.5. Extreme Gradient Boosting

마지막으로, 앞선 Random Forest 모형처럼 의사결정나무를 기반으로 하는 앙상블(Ensemble) 기법이지만, 이번엔 부스팅(Boosting)을 사용하는 XGB 모형을 사용해 보았다. GridSearchCV 를 통해 각 경우별 'max depth,' 'n_estimators,' 그리고 'learning_rate'를 설정하였다.

```
def xgb_cv(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=100)
    param_grid_xgb={'max_depth': [3, 5, 7, 9], 'n_estimators': [5, 10, 15, 20, 25, 50, 100],'learning_rate': [0.01, 0.05, 0.1]}
    xgb_cv=GridSearchCV(xgb.XGBClassifier(random_state=100), param_grid_xgb, cv=5)
    xgb_cv.fit(X_train,y_train)
    print(xgb_cv.best_params_)
    xgb_train=xgb_cv.predict(X_train)
    xgb_test=xgb_cv.predict(X_test)
    print("Train:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_train,xgb_train),2),'   /' ,
          round(recall_score(y_train, xgb_train, average='macro'),2),'   /' ,
          round(f1_score(y_train,xgb_train, average='macro'),2))
    print("Test:\n","Accuracy / Recall / F1 score  \n",round(accuracy_score(y_test,xgb_test),2),'   /' ,
          round(recall_score(y_test, xgb_test, average='macro'),2),'   /' ,
          round(f1_score(y_test,xgb_test, average='macro'),2))
    scores = cross_val_score(xgb_cv, X, y, cv=5)
    print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

먼저, 기존 데이터(Raw data)를 적합하였다. XGB 모형 역시 앞선 Random Forest 모형처럼 불균형이 존재하는 데이터에 대해서도 상대적으로 높은 성능을 보였다. 특히, Test 데이터의 F1 Score 에 한해선 Random Forest(약 87%)보다도 우수한 결과(약 89%)를 보였다.

```
xgb_cv(X,y)
```

```
{'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}
Train:
 Accuracy / Recall / F1 score
 0.97    / 0.93    / 0.95
Test:
 Accuracy / Recall / F1 score
 0.92    / 0.85    / 0.89
CV accuracy: 0.918 +/- 0.026
```

다음으로 과소표집 샘플의 경우, 이 또한 Test 데이터에 대한 전체적인 성능이 원 데이터보다 오히려 떨어졌다. 특히 F1 Score 의 경우, 확연히 저하된 결과(약 79%)를 보였으며, Random Forest 모형처럼 표본 크기의 감소가 모형에 매우 부정적인 영향을 미친 것으로 판단되었다. Train 데이터에 대해선 완벽한 성능을 보이는 것으로 보아, 과대적합(Overfitting)이 의심되었다.

```
xgb_cv(X_usampled,y_usampled)
```

```
{'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 25}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.78    / 0.8    / 0.79
CV accuracy: 0.774 +/- 0.042
```

과대표집 샘플의 경우, 데이터 불균형 문제와 과소표집의 한계를 보완했기 때문에, Test 데이터에 대한 전체적인 성능이 상승하였다. 전체적인 성능이 Random Forest 모형의 결과와 유사했다. XGB 모형 역시 SMOTE 가 ADASYN 보다 높은 정확도를 보였다. 각 SMOTE 와 ADASYN 표본의 결과는 다음페이지에 첨부하였다.

```
xgb_cv(X_sm,y_sm)
```

```
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 100}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.96    / 0.96    / 0.96
CV accuracy: 0.953 +/- 0.016
```

```
xgb_cv(X_ada,y_ada)
```

```
{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100}
Train:
 Accuracy / Recall / F1 score
 1.0    / 1.0    / 1.0
Test:
 Accuracy / Recall / F1 score
 0.96    / 0.96    / 0.96
CV accuracy: 0.930 +/- 0.023
```

3. Model & Method Selection

앞서 각 경우에서 Train결과와 Test결과의 차이를 살펴본 결과, 크게 과대적합 혹은 과소적합의 문제가 생긴 모형이 없다고 판단하였다. 따라서, 이제 앞서 다룬 5가지 모형의 Test 데이터에 대한 결과를 각 성능 평가 측도로 요약한 표를 통해 본 프로젝트의 시뮬레이션 데이터에 가장 적합한 모형을 찾겠다.

[CV Accuracy]

	LOGISTIC	LDA	SVM	RF	XGB
ORIGINAL	0.86	0.83	0.89	0.92	0.92
UNDERSAMPLE	0.75	0.73	0.75	0.78	0.77
SMOTE	0.84	0.80	0.98	0.96	0.95
ADASYN	0.79	0.76	0.96	0.94	0.93

먼저, 정확도에 관한 표이다. 정확도는 주로 균형데이터에 대한 모형 평가 측도로 쓰이기 때문에, 원 데이터에 대한 결과를 제외하면, SVM, Random Forest, 그리고 XGB모형이 비교적 높은 정확도를 보였다. 재표집 방법 중에선, 과대표집의 SMOTE 기법을 적용한 표본이 모든 모형에 대해 가장 높은 성능을 보였다. 하지만, 이는 시뮬레이션 데이터의 소수클래스에 큰 이상치가 없기 때문이며, 이상치가 다소 존재하는 실제 데이터에선 ADASYN으로 과대표집하는 것이 선호될 수 있겠다고 생각했다. SMOTE는 소수 클래스에 속한 모든 관측치를 선으로 이어 합성관측치를 생성하는 작업을 반복적으로 수행하기 때문이다. 과소표집의 경우, 표본의 수를 줄이는 작업 때문에 모형의 정확도가 모든 경우에서 확연하게 떨어졌다.

[Recall Score]

	LOGISTIC	LDA	SVM	RF	XGB
ORIGINAL	0.73	0.63	0.78	0.82	0.85
UNDERSAMPLE	0.80	0.77	0.85	0.82	0.80
SMOTE	0.83	0.80	0.95	0.95	0.96
ADASYN	0.80	0.77	0.96	0.97	0.96

다음은, 재현율에 관한 표이다. 원 데이터와 재표집된 표본의 결과 비교를 통해, 데이터 불균형 해소의 중요성을 알아볼 수 있다. 모든 모형에 있어, 원 데이터보다 재표집된 데이터를 사용한 모형의 재현율이 모든 경우에서 높게 나왔다. 특히, 과소표집보단 SMOTE나 ADASYN과 같이 과대표집 방법을 적용하는 것이 더 우수한 점수를 기록했다. 이는 앞서 설명했듯, 표본의 수를 줄이는 것보단 늘리는 것이 모형의 정밀도를 보존하기 때문이다. 두 앙상블 모형(Random Forest과 XGB 모형)은 데이터의 불균형 정도와 관계없이 꾸준히 높은 재현율을 도출하였다.

[F1 SCORE]

	LOGISTIC	LDA	SVM	RF	XGB
ORIGINAL	0.76	0.65	0.84	0.87	0.89
UNDERSAMPLE	0.79	0.76	0.85	0.83	0.79
SMOTE	0.83	0.80	0.95	0.95	0.96
ADASYN	0.80	0.76	0.96	0.97	0.96

이제 정밀도(Precision)와 재현율의 조화평균인 F1 Score를 살펴보았다. 불균형 데이터의 경우, 무조건 다수클래스로 판정하면 높은 정밀도를 가지게 되기 때문에, 원 데이터의 결과는 크게 상관하지 않았다. XGB모형의 경우, 언더샘플링을 했을 때 F1-값이 앞선 Recall값보다 낮아져, 정밀도가 상대적으로 떨어졌음을 확인할 수 있었다. 반면 SVM, 그리고 Random Forest모형은 재표집 방법과 관계없이 꾸준히 높은 F1 Score을 도출하였다.

위 결과를 종합하면, Accuracy, Recall, F1 Score에서 모두 가장 우수한 성능을 보인 Random Forest 모형을 사용하는 것이 합당하다고 판단되었다. SVM, XGB 모델 또한 우수하지만, 모형 평가 척도의 상대적인 값으로 비교하였을 땐, 본 시뮬레이션 데이터에 한해선 Random Forest 모형이 최적이라고 결론지을 수 있었다.

본 프로젝트를 통해 데이터의 불균형을 처리하지 않은 채 분류 알고리즘을 만든다면, 목표한 이상적인 성능(Recall Score)이 나올 수 없음을 확인하였다. 이는 성능을 높이기엔 소수클래스에 대한 사례가 너무 적기 때문이다. 또한, 이를 해결하기 위해 재표집 방법을 사용할 땐, 과소표집보단 과대표집이 더 우수한 결과를 보임을 명백히 확인할 수 있었다. 특히 과소표집의 경우, 표본의 크기가 매우 줄어듦에 따라 원 데이터보다도 정밀도가 저하될 수 있었다.