Machine Learning 1st Term project

Classification - 학생들의 Dropout, Graduate 여부를 예측하는 분류 알고리즘

2021741062 김현준

데이터 선정

데이터는 Kaggle에서 수집을 하였음. (https://www.kaggle.com/datasets) 분류 알고리즘에 적합한 다양한 데이터들이 존재하였는데 다음과 같은 기준을 가지고 데이터를 선정하였음

1. 데이터의 양이 많이 많은 데이터셋
2. 사용하는 독립 변수의 수가 많은 데이터셋 (X의 파라미터 수)
3. 신뢰가 가능한 데이터셋
4. Kaggle 상에서 Code 예제가 적은 데이터셋
5. 데이터의 용량이 적당한 데이터 셋

다음과 같은 기준으로 다양한 데이터셋을 찾아본 결과, 태아의 건강을 예측하는 등 다양한 질병을 예측하거나 사막, 비 등의 날씨를 예측하거나, 사기를 예측하는 다양한 데이터 셋을 찾을 수 있었음. 이 중 책이나 학교에서 학습한 부분을 제외하고, 용량이 너무 큰 데이터 셋들을 제외한 결과 학생들의 중퇴 여부를 예측하는 데이터 셋이 제일 적합하다고 판단했음. 선택한 데이터 셋은 논문(https://www.mdpi.com/2306-5729/7/11/146)에 사용된 데이터 셋이며 중퇴 여부를 결정할 때 34개의 인자가 사용됐으며 89KB의 낮은 용량을 가지고 있어 사용할 때 불편함 없이 사용할 수 있었음. 또한 코드 예제들을 살펴봤을 때 KNN을 사용하는 몇 개의 예제 말고는 Classification 예제를 찾을 수 없었기에 다양한 시도를 해보면서 학습 모델들의 성능을 직접 향상시킬 수 있다는 장점이 있었음.

데이터 전처리

다운받은 데이터는 그대로 지도 학습에 사용할 수 없는데 그 이유는 scikit - learn은 문자열 데이터를 받아내지 못하기 때문임. 그렇기에 데이터의 범주형 데이터와 숫자형 데이터의 형태를 바꾸어 학습에 적합한 부분으로 바꿔주는 과정은 필수적임.

Dataset.csv를 열어본 결과 Target 부분 이외에는 이미 라벨 인코딩이 되어있음을 확인할 수 있었음.

	Marital status	Application mode	Application order	Course	Daytime/evening attendance	Previous qualification	Nacionality	Mothers qualification	Fathers qualification	Mothe occupati
0	1	8	5	2	1	1	1	13	10	
1	1	6	1	11	1	1	1	1	3	
2	1	1	5	5	1	1	1	22	27	
3	1	8	2	15	1	1	1	23	27	
4	2	12	1	3	0	1	1	22	28	
...	...	...	...	...	...	...	...	...	...	
4419	1	1	6	15	1	1	1	1	1	
4420	1	1	2	15	1	1	19	1	1	
4421	1	1	1	12	1	1	1	22	27	
4422	1	1	1	9	1	1	1	22	27	
4423	1	5	1	15	1	1	9	23	27	

4424 rows × 35 columns

또한 data.info()와 data.isna().sum()을 통해 누락값이 있는지 없는지 확인해봐야함. 확인 결과 결측값은 존재하지 않음을 확인할 수 있었음.

```
In [30]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4424 entries, 0 to 4423
Data columns (total 35 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Marital status                                  4424 non-null   int64
 1   Application mode                                4424 non-null   int64
 2   Application order                               4424 non-null   int64
 3   Course                                          4424 non-null   int64
 4   Daytime/evening attendance                      4424 non-null   int64
 5   Previous qualification                          4424 non-null   int64
 6   Nacionality                                     4424 non-null   int64
 7   Mother's qualification                          4424 non-null   int64
 8   Father's qualification                          4424 non-null   int64
 9   Mother's occupation                             4424 non-null   int64
 10  Father's occupation                             4424 non-null   int64
 11  Displaced                                       4424 non-null   int64
 12  Educational special needs                       4424 non-null   int64
 13  Debtor                                          4424 non-null   int64
 14  Tuition fees up to date                         4424 non-null   int64
 15  Gender                                          4424 non-null   int64
```

바로 학습에 사용할 수 있는 데이터이긴 했지만, 이 상태로 사용할 수는 없었음. 범주형 데이터의 경우에 데이터의 성질에 따라 순서를 가지거나 가지지 않을 수 있는데, 이에 따라 라벨 인코딩을 진행하거나 One-hot 인코딩을 진행하냐에 따라서 학습 모델의 성능이 달라질 수 있었기 때문임.

하지만 저렇게 정리된 데이터를 어떠한 기준으로 라벨 인코딩을 유지할지, One-hot으로 변경할지에 대한 고민을 하였는데, 이는 dataset에 적혀있는 설명과, 관련 논문을 찾아봄으로써 해결할 수 있었음. dataset에서 사용하고 있는 변수들의 목록은 다음과 같음.

Marital status, Application mode, Application order, Course, Daytime/evening attendance, Previous qualification, Nacionality, Mother's, Father's qualification, Mother's, Father's occupation, Displaced, Educational special needs, Debtor, Tuition fees up to date, Gender, Scholarship holder, Age at enrollment, International, Curricular units 1^{st},2^{nd} sem (credited), Curricular units 1^{st},2^{nd} sem (enrolled), Curricular units 1^{st},2^{nd} sem (evaluations), Curricular units 1^{st},2^{nd} sem (approved), Curricular units 1^{st},2^{nd} sem (grade), Curricular units 1^{st},2^{nd} sem (without evaluations), Unemployment rate, Inflation rate, GDP

이 중 범주형 데이터는 Marital status, Application mode, Course, Daytime/evening attendance, Previous qualification, Nacionality, Mother's, Father's qualification, Mother's, Father's occupation, Displaced, Educational special needs, Debtor, Tuition fees up to date, Gender, Scholarship holder, International 부분임.

Table A1. Marital status values.

Attribute	Values
Marital status	1—Single 2—Married 3—Widower 4—Divorced 5—Facto union 6—Legally separated

Table A2. Nationality values.

Attribute	Values
Nationality	1—Portuguese 2—German 3—Spanish 4—Italian 5—Dutch 6—English 7—Lithuanian 8—Angolan 9—Cape Verdean 10—Guinean 11—Mozambican 12—Santomean 13—Turkish 14—Brazilian 15—Romanian 16—Moldova (Republic of) 17—Mexican 18—Ukrainian 19—Russian 20—Cuban 21—Colombian

Table A3. Application mode values.

Attribute	Values
Application mode	1—1st phase—general contingent 2—Ordinance No. 612/93 3—1st phase—special contingent (Azores Island) 4—Holders of other higher courses 5—Ordinance No. 854-B/99 6—International student (bachelor) 7—1st phase—special contingent (Madeira Island) 8—2nd phase—general contingent 9—3rd phase—general contingent 10—Ordinance No. 533-A/99, item b2) (Different Plan) 11—Ordinance No. 533-A/99, item b3 (Other Institution) 12—Over 23 years old 13—Transfer 14—Change in course 15—Technological specialization diploma holders 16—Change in institution/course 17—Short cycle diploma holders 18—Change in institution/course (International)

Table A4. Course values.

Attribute	Values
Course	1—Biofuel Production Technologies 2—Animation and Multimedia Design 3—Social Service (evening attendance) 4—Agronomy 5—Communication Design 6—Veterinary Nursing 7—Informatics Engineering 8—Equiniculture 9—Management 10—Social Service 11—Tourism 12—Nursing 13—Oral Hygiene 14—Advertising and Marketing Management 15—Journalism and Communication 16—Basic Education 17—Management (evening attendance)

Table A5. Previous qualification values.

Attribute	Values
Previous qualification	1—Secondary education 2—Higher education—bachelor's degree 3—Higher education—degree 4—Higher education—master's degree 5—Higher education—doctorate 6—Frequency of higher education 7—12th year of schooling—not completed 8—11th year of schooling—not completed

Table A5. *Cont.*

Attribute	Values
	9—Other—11th year of schooling 10—10th year of schooling 11—10th year of schooling—not completed 12—Basic education 3rd cycle (9th/10th/11th year) or equivalent 13—Basic education 2nd cycle (6th/7th/8th year) or equivalent 14—Technological specialization course 15—Higher education—degree (1st cycle) 16—Professional higher technical course 17—Higher education—master's degree (2nd cycle)

Table A6. Mother's and Father's values.

Attribute	Values
Mother's qualification Father's qualification	1—Secondary Education—12th Year of Schooling or Equivalent 2—Higher Education—bachelor's degree 3—Higher Education—degree 4—Higher Education—master's degree 5—Higher Education—doctorate 6—Frequency of Higher Education 7—12th Year of Schooling—not completed 8—11th Year of Schooling—not completed 9—7th Year (Old) 10—Other—11th Year of Schooling 11—2nd year complementary high school course 12—10th Year of Schooling 13—General commerce course 14—Basic Education 3rd Cycle (9th/10th/11th Year) or Equivalent 15—Complementary High School Course 16—Technical-professional course 17—Complementary High School Course—not concluded 18—7th year of schooling 19—2nd cycle of the general high school course 20—9th Year of Schooling—not completed 21—8th year of schooling 22—General Course of Administration and Commerce 23—Supplementary Accounting and Administration 24—Unknown 25—Cannot read or write 26—Can read without having a 4th year of schooling 27—Basic education 1st cycle (4th/5th year) or equivalent 28—Basic Education 2nd Cycle (6th/7th/8th Year) or equivalent 29—Technological specialization course 30—Higher education—degree (1st cycle) 31—Specialized higher studies course 32—Professional higher technical course 33—Higher Education—master's degree (2nd cycle) 34—Higher Education—doctorate (3rd cycle)

One-hot 인코딩은 범주가 너무 넓은 경우, 너무 많은 열을 가지는 새로운 데이터가 생성되기 때문에 과적합의 위험이 있음. 그렇기에 순서가 없는 데이터들 중 적당한 크기를 가지는 데이터에만 One – hot 인코딩을 적용하고자 했음. Pandas의 pd.get_dummies를 활용하여 One – hot 인코딩을 진행하였음

```
encoded = pd.get_dummies(data, columns=['Gender', 'Displaced', 'Debtor',
                                        'Tuition fees up to date', 'Scholarship holder',
                                        'Application mode','Course'])
encoded.head()
```

	Marital status	Application order	Daytime/evening attendance	Previous qualification	Nacionality	Mothers qualification	Fathers qualification	Mothers occupation	Fathers occupation	Educational special needs	...	Course_8	Course_9
0	1	5	1	1	1	13	10	6	10	0	...	0	0
1	1	1	1	1	1	1	3	4	4	0	...	0	0
2	1	5	1	1	1	22	27	10	10	0	...	0	0
3	1	2	1	1	1	23	27	6	4	0	...	0	0
4	2	1	0	1	1	22	28	10	10	0	...	0	0

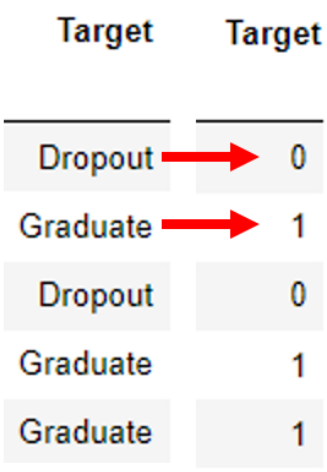
5 rows × 73 columns

One – hot 인코딩을 적용한 결과 35개의 Columns이 73개의 Columns로 전처리 되었음. One – hot 인코딩을 진행한 이후에 문자열로 구성되어 있는 Target 부분을 수정하고 필요없는 부분을 삭제하는 과정을 진행하였음.

또한 Map 함수를 이용해 Dropout은 0으로 Enrolled는 2로 Graduate는 1로 매핑하였으며, 예측하고자 하는 것은 학생들의 Dropout과 Graduate를 예측하는 것이기 때문에 사용하지 않는 Enrolled 데이터는 삭제하였음. 그 결과 3630개의 행만 남았음.

```
data['Target'] = data['Target'].map({'Dropout':0, 'Enrolled':2, 'Graduate':1 })
data
```

```
data = data[data.Target != 2]
data
```



	Marital status	Application mode	Appli
0	1	8	
1	1	6	
2	1	1	
3	1	8	
4	2	12	
...	...	...	
4419	1	1	
4420	1	1	
4421	1	1	
4422	1	1	
4423	1	5	

4424 rows × 35 columns

	Marital status	Application mode
0	1	8
1	1	6
2	1	1
3	1	8
4	2	12
...	...	...
4419	1	1
4420	1	1
4421	1	1
4422	1	1
4423	1	5

3630 rows × 35 columns

범주형 데이터를 전처리 했다면, 이제는 숫자형 데이터를 전처리 해야함. 숫자형 데이터들은 sklearn에서 제공하는 MinMaxScaler를 이용하여 정규화를 진행하였음. 숫자형 데이터의 경우 모두 전처리를 해야 학습 시 성능이 떨어지는 것을 막을 수 있음.

```
from sklearn.preprocessing import MinMaxScaler

columns_to_normalize = ['Application order', 'Age at enrollment', 'Unemployment rate', 'In
                        'Curricular units 1st sem (enrolled)', 'Curricular units 1st sem (
                        'Curricular units 1st sem (approved)', 'Curricular units 1st sem (
                        'Curricular units 2nd sem (enrolled)', 'Curricular units 2nd sem (
                        'Curricular units 2nd sem (approved)', 'Curricular units 2nd sem (

scaler = MinMaxScaler()

# 열들의 값을 0~1로 정규화
encoded[columns_to_normalize] = scaler.fit_transform(data[columns_to_normalize])

encoded.head()
```

	Marital status	Application order	Daytime/evening attendance	Previous qualification	Nacionality	Mothers qualification	Fathers qualification	Mothers occupation	Fathers occupation	Educa sp r
0	1	0.833333	1	1	1	13	10	6	10	
1	1	0.166667	1	1	1	1	3	4	4	
2	1	0.833333	1	1	1	22	27	10	10	
3	1	0.333333	1	1	1	23	27	6	4	
4	2	0.166667	0	1	1	22	28	10	10	

5 rows × 73 columns

다음과 같이 전처리를 한 이후 Random Forest 알고리즘 등 다양한 알고리즘으로 학습시킨 결과 과적합 문제를 발견할 수 있었음. 이에 PCA를 활용해서 차원을 축소하여 문제를 해결하고자 하였으나, 차원 축소 시 데이터 유실이 많이 발생하는 탓인지, 정답 예측률이 심각하게 감소하는 것을 확인하였음. 이에 필요 없는 열을 삭제하여 Columns의 수를 줄이고자 하였음. 이에 feature_importances를 활용하여 어떤 특성이 predict에 중요하고 중요하지 않음을 찾았음.

```
# 특성 중요도를 가져옴
feature_importances = clf.feature_importances_

# 중요도에 따라 특성들을 정렬
sorted_idx = feature_importances.argsort()
```

```
columns_to_drop = ['Marital status','International','Daytime/evening attendance',
                   'Nacionality', 'Educational special needs','Curricular units 1st sem (
                   'Curricular units 2nd sem (without evaluations)','Curricular units 1st
                   'Curricular units 1st sem (credited)', 'Previous qualification']
data = data.drop(columns=columns_to_drop)
```

다음과 같이 영향력이 없는 columns를 삭제하여 필요없이 생성되는 열의 수를 제한함으로써 과적합의 위험을 줄이고자 하였음.

	Application order	Mothers qualification	Fathers qualification	Mothers occupation	Fathers occupation	Age at enrollment	Curricular units 1st sem (enrolled)	Curricular units 1st sem (evaluations)	Curricular units 1st sem (approved)	Curricular units 1st sem (grade)	...	Course_8	Course_9	Cour
0	5	13	10	6	10	20	0	0	0	0.000000	...	0	0	
1	1	1	3	4	4	19	6	6	6	14.000000	...	0	0	
2	5	22	27	10	10	19	6	0	0	0.000000	...	0	0	
3	2	23	27	6	4	20	6	8	6	13.428571	...	0	0	
4	1	22	28	10	10	45	6	9	5	12.333333	...	0	0	

5 rows × 64 columns

위의 데이터는 최종적으로 사용하는 전처리 된 데이터로 기본 인코딩이 되어 있는 상태에서, 필요 없는 데이터는 삭제하고, 범주형 데이터 중 일부에는 One-hot 인코딩을 진행하고, 숫자 데이터에는 MinMaxScaler를 사용하여 정규화를 진행하였음.

분류 및 파라미터 최적화

수업시간에 다양한 분류 알고리즘에 대해 학습해보았는데 수업시간에 배웠던 다양한 알고리즘으로 Train, Test를 진행해보고 가장 좋은 알고리즘이 무엇인지 파악해보고자 함.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify = y, ran
```

모든 알고리즘은 다음과 같이 일정하게 분할된 X_train, X_test, y_train, y_test를 사용하며, randomstate는 42로 선택하였음.

알고리즘은 Decision Tree, Random Forest, KNN, XGboost, Logistic Regression, SVM ,GBM, AdaBoost를 사용하였고 각 알고리즘에 대해 Train accuracy, Test accuracy를 구한다음, 만약 둘의 차이가 크다면 과적합이 일어날 수도 있는 부분이기 때문에 교차 검증과 최적 파라미터를 구하는 과정을 통하여 학습이 잘 되고 있는지를 확인해보고자 함.

Decision Tree

```
dt_clf = DecisionTreeClassifier(max_depth = 15)

dt_clf.fit(X_train,y_train)
dt_pred = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test,dt_pred)

# DecisionTree 학습 데이터에 대한 예측
dt_train_pred = dt_clf.predict(X_train)
dt_train_accuracy = accuracy_score(y_train, dt_train_pred)
print(f"DecisionTree Training Accuracy: {dt_train_accuracy*100:.2f}%")
print(f"DecisionTree Test Accuracy: {dt_accuracy*100:.2f}%")

# 교차 검증
cv_scores = cross_val_score(dt_clf, X, y, cv=5, scoring='accuracy')
#print(f"Cross Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean()*100:.2f}%")
print(f"Standard Deviation of CV Accuracy: {cv_scores.std()*100:.2f}%")
```

Max_depth를 제한하지 않고 설정한 결과 Train accuary의 값이 100%가 나와 과적합의 의심됨. 과적합을 방지하기 위해 max depth 파라미터만 15로 제한을 하고 학습을 시킨 결과 일반 Train accuary는 98.7%, 교차 검증을 시킨 Train accuary는 86.07%로 나왔음.

하이퍼 파라미터 최적화는 사이킷런에서 제공하는 GridSearchCV로 찾을 수 있음. GridSearchCV로 최적 파라미터를 찾고 이 파라미터를 사용하는 학습 모델을 만든 후 test data를 넣어 학습률이 늘어났는지를 판단했음.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 4, 6, 8, 10],
    'max_features': [None, 'sqrt', 'log2']
}

grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           n_jobs=-1,
                           verbose=2)

grid_search.fit(X_train, y_train)

print(f"Best Hyperparameters: {grid_search.best_params_}")
print(f"Mean CV Accuracy: {grid_search.best_score_:.4f}")

# 최적의 하이퍼파라미터로 훈련된 모델을 사용
best_dt_clf = grid_search.best_estimator_

best_dt_pred = best_dt_clf.predict(X_test)
best_dt_accuracy = accuracy_score(y_test, best_dt_pred)

print(f"DecisionTree with Best Hyperparameters Test Accuracy: {best_dt_accuracy*100:.2f}%")
```

학습 결과 최적의 파라미터는 다음과 같이 나왔고 Train accuracy는 88.39%, Test accuracy는 87.51%가 나왔음을 확인하였음.

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
Best Hyperparameters: {'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 6, 'min_samples_split':
2}
Best Score: 0.8839
DecisionTree with Best Hyperparameters Test Accuracy: 87.51%
Confusion Matrix for Best Hyperparameters:
```

KNN

```
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
knn_pred = knn_clf.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_pred)

knn_train_pred = knn_clf.predict(X_train)
knn_train_accuracy = accuracy_score(y_train, knn_train_pred)

print(f"KNN train Accuracy: {knn_train_accuracy*100:.2f}%")
print(f"KNN test Accuracy: {knn_accuracy*100:.2f}%")

# 교차 검증
cv_scores = cross_val_score(knn_clf, X, y, cv=5, scoring='accuracy')
#print(f"Cross Validation Scores: {cv_scores}")
print(f"Mean CV Accuracy: {cv_scores.mean()*100:.2f}%")
print(f"Standard Deviation of CV Accuracy: {cv_scores.std()*100:.2f}%")
```

KNN은 기본적으로 n_neighbors 파라미터를 지정해야하기 때문에 5로 지정하여 Train, Test Accuracy를 계산하였음. Train Accuracy는 88.63%, Test Accuracy는 83.93%로 과적합이 의심되지는 않음.

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15],  # 예시로 3부터 15까지의 홀수를 사용
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

KNN은 위와 같은 파라미터를 사용하는데 weight는 예측에 사용되는 가중치 함수, metric은 거리 사이의 길이를 잴 때 사용하는 방법임.

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
Best parameters found:  {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'distance'}
Best cross-validation score: 0.85
Test set accuracy with best parameters: 84.39%
```

Logistic Regression

로지스틱 회귀는 이름만 보면 분류가 아니라 회귀라고 생각할 수 있지만 0과 1 로 값을 분류해내는 분류 알고리즘임. 분류하려는 Target의 source가 2개 일 때 원활하게 사용 가능하며, 3개 이상일 때에도 사용이 가능함.

```
# 로지스틱 회귀 모델 생성 및 훈련
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)

# 로지스틱 회귀 학습 데이터에 대한 예측
lr_train_pred = lr.predict(X_train)
lr_train_accuracy = accuracy_score(y_train, lr_train_pred)
print(f"Logistic Regression train Accuracy: {lr_train_accuracy*100:.2f}%")
print(f"Accuracy: {accuracy*100:.2f}%")
```

반복횟수 파라미터인 max_iter 부분만 값을 조정하여 대입하였고, 예측 결과 Train Accuracy는 92.25% Test Accuracy는 91.00%로 예측되었음.

```
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}
```

로지스틱 회귀에 사용되는 파라미터는 총 4개로, C는 역규제 파라미터, penalty는 규제의 종류, solver는 최적화에 사용하는 알고리즘, max_iter은 최대 반복 횟수임.

```
Best Parameters: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-validation Score: 0.9142095818572777
Test set Accuracy: 0.9090909090909091
```

SVM

SVM의 경우 분류와 회귀에 모두 사용되는데, 이번에는 분류 모델에 사용하기 때문에 SVC를 IMPORT 해서 사용함.

```
from sklearn.svm import SVC

# SVM 모델 생성 및 훈련
svm_clf = SVC(kernel='linear')
svm_clf.fit(X_train, y_train)

svm_predictions = svm_clf.predict(X_test)

# SVM 학습 데이터에 대한 예측
svm_train_predictions = svm_clf.predict(X_train)
print("SVM Train Accuracy:", accuracy_score(y_train, svm_train_predictions))
print("SVM Test Accuracy:", accuracy_score(y_test, svm_predictions))
```

Train Accuracy는 92.2%, Test Accuracy는 90.9%가 기록되었음.

밑에서 소개할 방식은 **앙상블**이라는 기법을 사용하는 부분으로써, 앙상블이란 집단 지성과 비슷한 의미를 가짐. 여러 개의 같은 알고리즘을 사용하여 답을 예측하고 이를 합산하여 최종 답안을 만드는 방식임.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(max_depth = 10, random_state=42)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)

# Random Forest 학습 데이터에 대한 예측
rf_train_pred = rf_clf.predict(X_train)
rf_train_accuracy = accuracy_score(y_train, rf_train_pred)

print(f"Random Forest Accuracy: {rf_train_accuracy*100:.2f}%")
print(f"Random Forest Test Accuracy: {rf_accuracy*100:.2f}%")
```

Train Accuracy는 95.83%, Test Accuracy는 90.73%로 측정되었음.

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

Decision Tree에서 심화된 것이 Random Forest임으로, 사용하는 파라미터가 흡사함을 확인할 수 있다. 최적 파라미터로 계산 결과 Train Accuracy는 91%, Test Accuracy는 90.17로 최적 파라미터를 계산했지만 오히려 처음보다 예측 실력이 낮아진 것을 확인할 수 있었음.

```
Best parameters found:  {'bootstrap': False, 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5,
'n_estimators': 100}
Best cross-validation score: 0.91
Accuracy with best parameters: 90.17%
```

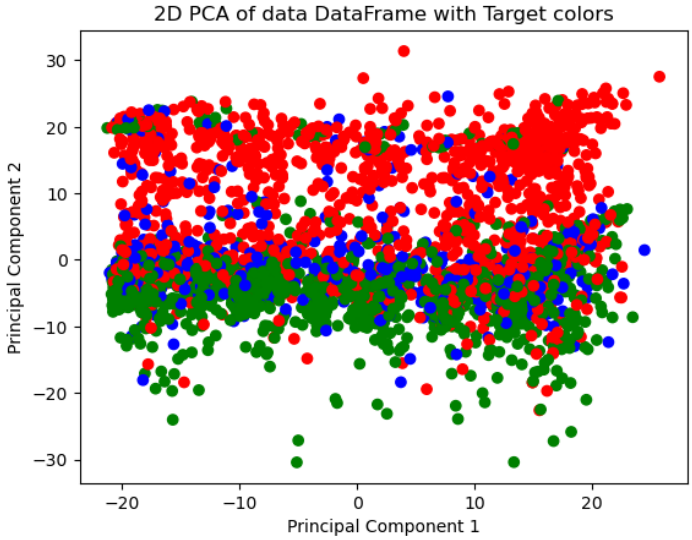
XGBOOST, GBM, AdaBoost

위 3개의 모델들은 실질적으로 분석하기보단 부스팅 알고리즘을 사용하는 방식이 더 높은 예측 성능을 보일지 궁금하여 Train, Test Accuracy를 계산해보았음. Adaboost와 GBM은 부스팅 방식을 사용하여 계산하는데 GBM은 가중치 업데이트를 할 때 경사하강법을 사용한다는 차이가 있음.

XGboost의 경우 Train 97.4% Test 91.09%. GBM의 경우 Train 91.26%, Test 90.36%. AdaBoost의 경우 Train 91.26%, Test 89.62%가 나왔음.

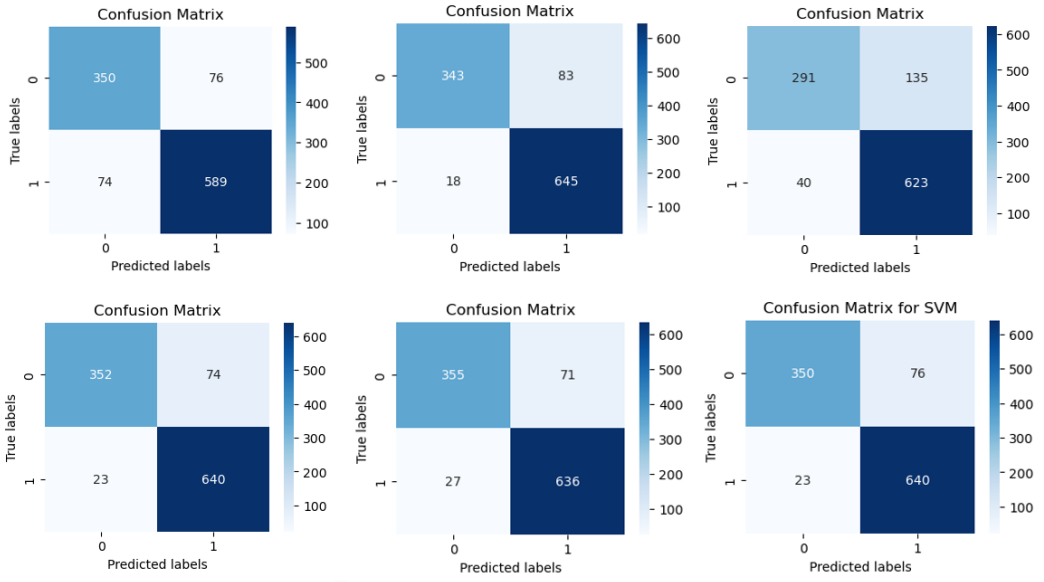
결과 및 분석, 고찰

처음에 데이터를 받은 후, 어떤 알고리즘을 사용해서 분석하는 것이 제일 좋은 효과를 낼지 궁금하여 PCA를 이용해 차원을 두개로 축소하였고 이를 2D 그래프로 그렸음.

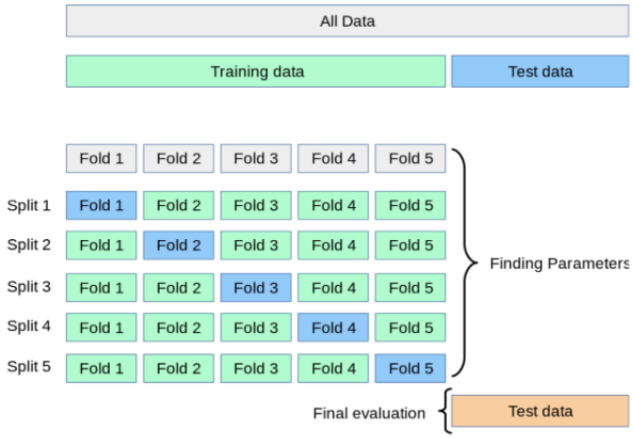


사용하지 않을 Enrolled 데이터는 파란색으로 표시가 되어 있으며 이를 확인한 후 초록색은 초록색끼리 근접하여 있고 빨간색은 빨간색끼리 잘 근접되어 있기 때문에 KNN 방식을 사용하는 것이 제일 적합할 것이라고 예상하였음. 하지만 전처리 이후 다양한 알고리즘을 사용해 테스트해본 결과 KNN 알고리즘의 Test Accuracy는 83.93%으로 상대적으로 낮게 나옴을 확인할 수 있었음.

왜 예상과 다르게 나왔는가를 생각해본 결과 Decision Tree와 Random Forest는 이진 분류에 적합한 알고리즘으로 Enrolled를 사용하지 않는 시점에서는 이진 분류라고 봐도 되기에 굉장히 강력한 영향을 끼칠 수 있었을 것이며, 전처리를 하는 과정 속에서 분포가 바뀌어 KNN 보다 다른 알고리즘이 더 강력하게 작용할 수 있었을 것이라고 생각할 수 있었음.



위의 그림은 각 알고리즘별로 학습을 진행한 이후 Confusion Matrix를 계산하고 seaborn library를 이용하여 시각화 한 부분임. 전체적으로 보았을 때 각 알고리즘의 Test Accuracy가 85~93%로 측정되었고, Confusion Matrix의 대각선 부분이 진하게 표시되었기에 대부분 좋은 분류를 할 수 있다고 판단할 수 있으나, 딥러닝 모델을 사용하는 것이 아니였기에 epoch 함수를 그릴 수 없어 본 모델이 과적합이 일어난건지 그렇지 않은건지를 확인할 수 있는 방법이 거의 없었음. 그랬기에 Train, Test accuracy가 차이가 크면 과적합이 일어났다고 판단, 교차 검증을 통하여 안정적인 Train accuracy를 구하고자 하였음. 하지만 교차 검증은 과적합을 예방할 수 있다고는 하지만 결국 학습 데이터에 대한 새로운 Train accuracy 점수만을 기록하기에, 실제로 test_data를 넣어 과적합이 일어나는지 아닌지는 확인할 수가 없다는 단점이 있었음.



각 알고리즘의 적정 하이퍼 파라미터를 계산하는 코드를 통해 대부분의 경우 파라미터를 더 제한거는 것이 과적합도 막고 Test Accuracy도 향상시키는 결과를 보였지만, 일부 알고리즘의 경우 하이퍼 파라미터를 제시하였을 때 오히려 성능이 떨어지는 것을 확인하였음. 하이퍼 파라미터를 계산하여 제한을 하는 것이 오히려 안 좋을 수도 있다는 것을 알았음.

처음에는 Enrolled 데이터 또한 유효한 데이터 값이라고 생각하여 0 1 2 로 인코딩 한 후 학습을 진행하였음. Target을 3개로 설정하고 코딩하였을 때에도 어느정도 감지는 성공하였음. 하지만 Test Accuracy가 너무 낮은 수준에서 올라오지 못하였음. 전처리를 다시 해보고 PCA도 사용해 차원을 줄여봤지만 더 높게 상승하지 못하였음. 현재의 데이터가 영역이 깔끔하게 나눠져 있지 않고 서로 겹쳐져 있었기에 더 분류가 힘들었다고 생각됨. 다음 기회에는 다른 전처리 방식과 다른 알고리즘으로 Target이 3개인 데이터도 높은 정확도로 분류해보고 싶음

```
DecisionTree Training Accuracy: 97.35%
DecisionTree Test Accuracy: 70.18%
Mean CV Accuracy: 78.23%
Standard Deviation of CV Accuracy: 0.59%
Confusion Matrix:
[[313  73  55]
 [ 72  91  82]
 [ 46  68 528]]
```