Machine Learning (2021 Fall semester)

Programming Assignment:
Classification of Titanic Data set

1. Benchmark Dataset

사용하지 않는 feature 들을 drop 하는 과정

```
train = pd.read_csv("train.csv")
train_drop = train.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis = 1)
```

2. Preprocessing

2.1 train data 의 결측 치(NULL) 처리

Train data 의 결측 값은 Age 와 Embarked feature 이다. 해당 결측 값 확인은 isnull().sum() 함수를 통해 파악했다. 우선 Age feature 결측 값은, 나이의 평균값을 채워 넣는 방식으로 처리했다. 또한, embarked feature 결측 값은, age feature 처럼 평균값으로 처리할 수 없으므로, 결측 된 column 은 제거하였다.

```
age_avg = round(train_drop['Age'].mean())
values  = {"Age" : age_avg}
train_drop = train_drop.fillna(value=values)
train_drop_ = train_drop.dropna()
```

2.2 One-Hot encoding 을 수행할 필요가 있는 feature

컴퓨터는 문자보다는 숫자 처리에 조금 더 용이하다. 따라서 자연어 처리에서는 문자를 숫자로 바꾸는 여러가지 기법들이 존재한다. 그 중에서, One-Hot encoding 은 단어를 표현하는 가장 기본적인 방법이고, 표현하고 싶은 단어의 인덱스에 특정한 정수 값을 부여하는 단어의 벡터 표현 방식이다.

Titanic train data set 에는, 성별을 나타내는 sex feature 와 embarked feature 가 문자열로 이루어진다. 따라서, 해당 feature 의 수식계산을 통한 학습을 위해서는 One-Hot encoding 의 처리가 필요하다. 우선적으로, sex feature 의 value 인 male 과 female 을 표현하는 정수 값으로 각각 0 과 1 을 부여했다. 그리고, embarked feature 의 value 인 S, C, Q 에는 각각 0, 1, 2 를 부여했다. 이후, mapping 함수를 이용해 train data 의 feature 값을 map 한 value 로 변환했다.

```
sex_map = {"male":0, "female":1}
embark_map = {"S":0, "C":1,"Q":2}

train_drop_map = train_drop_.copy()
train_drop_map['Sex'] = train_drop_map['Sex'].map(sex_map)
train_drop_map['Embarked'] = train_drop_map['Embarked'].map(embark_map)
```

2.3 Split train data

```
label = train_scale.iloc[:, 0].values
feature = train_scale.iloc[:, 1:7].values

train_label, test_label, train_feature, test_feature = train_test_split(label, feature, test_size = 0.3, random_state = 42)
```

2.4 그 외 진행한 전 처리 과정 (Remove Outlier)

Data outlier 는 이상이 있는 데이터를 뜻한다. 일반적인 데이터 패턴과 다르게 매우 이상한 패턴을 가지고 있는 데이터를 뜻한다. Machine learning 에 있어서, 이러한 outlier data 로 인해 모델의 성능이 크게 영향을 받는다. 따라서 titanic data set 에 **Sigma clipping** 연산을 수행하여, outlier 를 제거하였다. Sigma clipping 연산을 수행한 feature 는 age feature 이다.

```
quartiles = np.percentile(train_drop_map['Age'], [25, 50, 75])
mu = quartiles[1]
sig = 0.74 * (quartiles[2] - quartiles[0])

train_rm_outlier = train_drop_map.query(' (Age > @mu - 5*@sig) & (Age < @mu + 5*@sig)')
```

3. Machine Learning Models

3.1 K-Nearest Neighbors (KNN)

KNN 에서는, K(이웃)의 개수를 [1~5]까지 변화시키면서 test data 의 결과가 어떻게 변하는지 분석하였다.

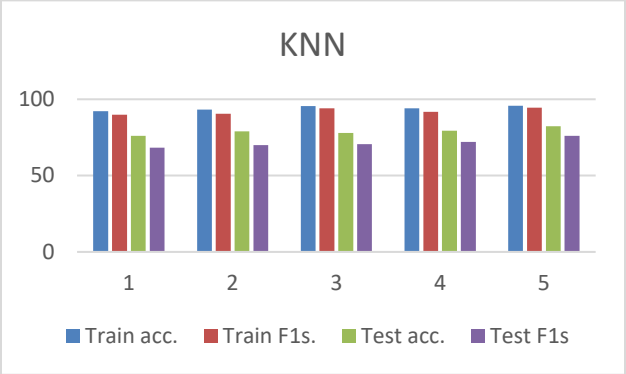

표 1: KNN 결과 [2]

	[Train acc.]	[Train F1s.]	[Test acc.]	[Test F1s.]
[k : 1]	98.068	97.479	68.539	54.839
[k : 2]	84.219	74.211	69.663	43.357
[k : 3]	83.253	77.966	69.288	55.435
[k : 4]	79.871	70.309	68.539	48.148
[k : 5]	80.193	73.774	71.536	58.242

그림 1: KNN 결과 [2]

[표 1]과 [그림 1]을 통해 알 수 있는 결과는 다음과 같다. 우선, K 의 값이 증가할수록 학습 데이터 상에서 정확도가 감소함을 알 수 있다. 하지만 반대로, 테스트 데이터의 정확도는 K 의 값이 증가할수록, 비례하게 증가함을 알 수 있다. 또한, KNN 모델의 테스트 데이터의 정확도는 68%에서 71% 전후임을 알 수 있다.

3.2 Logistic Regression

Logistic Regression 에서는, iteration 횟수를 [0~100] 범위에서 20 씩 변화시키면서 test data 에서 결과가 어떻게 변하는지 분석하였다. 또한, Iteration 횟수를 100 으로 고정한 후 regularization term 이 [0~5] 범위에서 1 씩 변화하면서 test data 에서 결과가 어떻게 변하는지 분석하였다.

표 2 : Logistic Regrssion 결과 [1] (iteration)

	[Train acc.]	[Train F1s.]	[Test acc.]	[Test F1s.]
[#iter : 0]	66.667	47.059	69.288	48.101
[#iter : 20]	79.871	71.910	80.524	74.000
[#iter : 40]	80.193	72.483	80.150	72.539
[#iter : 60]	79.388	71.681	80.899	74.627
[#iter : 80]	79.388	72.052	80.899	74.627
[#iter : 100]	78.744	71.429	80.899	74.877

그림 2: Logistic Regression 결과 [2] (iteration)

우선, iteration 의 횟수를 변경하면서 변화되는 test data 의 결과이다. [표 2]와 [그림 2]를 통해 알 수 있는 결과는, iteration 이 0 인 것을 제외하고는 학습 데이터와 테스트 데이터 모두 변동이 크지 않음을 알 수 있다.

표 3: Logistic Regression 결과 [1] (C)

	[Train acc.]	[Train F1s.]	[Test acc.]	[Test F1s.]
[c : 1]	78.744	71.429	80.899	74.877
[c : 2]	78.905	71.584	80.524	74.510
[c : 3]	78.905	71.584	80.524	74.510
[c : 4]	78.905	71.584	80.524	74.510
[c : 5]	78.905	71.584	80.524	74.510

그림 3: Logistic Regression 결과 [2] (C)

다음은. Iteration 횟수를 100 으로 고정한 후 regularization term 에 변화를 주어 변화되는 test data 의 결과이다. 위의 결과와 유사하게, C 값의 변동은 학습 데이터와 테스트 데이터의 변동이 거의 없었다.

3.3 Decision Tree

Decision Tree 에서는, Information Gain 을 통해 test data 에서 결과가 어떻게 나오는지 분석하고, tree 를 시각화 하는 과정을 거친다. 또한, Bagging 기법을 이용하여 bag 의 수에 따라 test data 의 결과가 어떻게 변하는지 decision tree 와 비교하며 분석하는 과정을 가진다.

```
Train data set accuracy : 98.068
Train data set F1_Score : 97.436
Test data set accuracy : 78.652
Test data set F1_Score : 70.157
```

그림 4: Decision Tree 결과

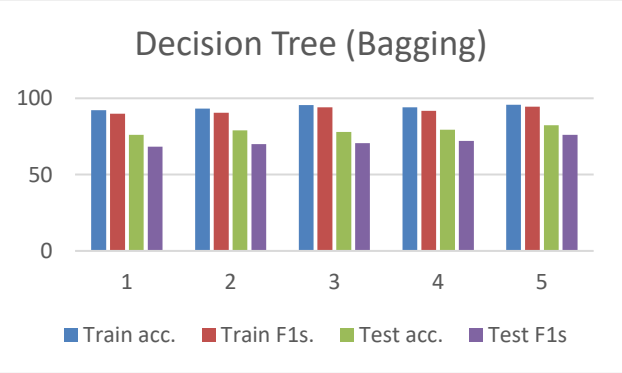

표 4: Decision Tree (Bagging) 결과 [1]

	[Train acc.]	[Train F1s.]	[Test acc.]	[Test F1s.]
[bag : 1]	92.271	89.916	76.030	68.317
[bag : 2]	93.237	90.583	79.026	69.892
[bag : 3]	95.491	94.093	77.903	70.647
[bag : 4]	94.042	91.868	79.401	72.081
[bag : 5]	95.813	94.538	82.397	76.142

그림 5: Decision Tree (Bagging) 결과 [2]

Information gain 을 통한 test data 의 결과는 [그림 4]를 통해 나타난다. 또한, Decision Tree 기반으로 Bagging 기법을 bag 의 수에 따라 test data 의 변화량은 [표 6] 과 [그림 5]를 통해 나타난다. Information gain 을 통한 test data 의 학습데이터 정확도는 Bagging 기법을 통한 정확도 보다 높은 것으로 나타난다. 하지만 테스트 데이터의 정확도는 bag 가 1 일때를 제외하면, bagging 기법을 통한 정확도가 더 높음을 알 수 있다. 또한, bag 의 수가 5 일때가 학습데이터 정확도와 테스트 데이터의 정확도 모두 가장 높은 것을 알 수 있다. [그림 6]은 tree 를 시각화 한 것이다.

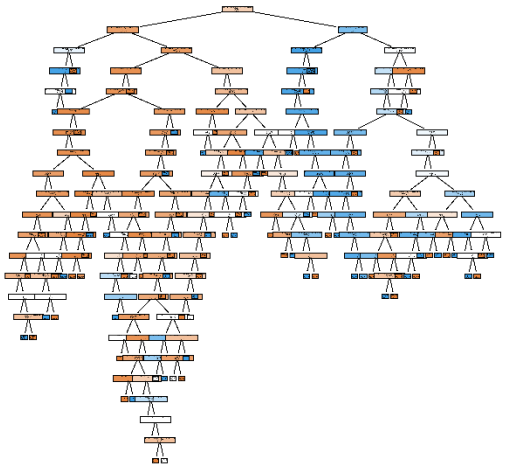

그림 6: Visualize Tree