hw3 : Drug Classification

1. EDA

칼럼에 있는 레이블들은 아래와 같습니다.

Age	Sex	BP	Cholesterol	Na_to_K	Drug

1. Age : 나이
2. Sex : 성별로 F는 Female(여성), M은 Male(남성)입니다.
3. BP : Blood Pressure Level로 혈압입니다.
4. Cholesterol : Cholesterol Levels로 콜레스테롤 수치를 의미합니다.
5. Na_to_K : Na to Potassium Ration으로 나트륨 - 칼륨 비율을 의미한다.

Target feature는 Drug type으로 약물의 종류이다.

CODE

```
train = pd.read_csv("drug200.csv")
print(train.head(),"\n")
print("shape : ", train.shape, "\n")
print(train.describe())
```

우선 load한 dataset을 train이라고 저장하였습니다. train dataset의 상위 5개 행을 출력하고 shape(), describe() 메서드를 통해서 데이터의 분포를 아래와 같이 확인하였습니다.

```
   Age Sex      BP Cholesterol  Na_to_K   Drug
0   23   F    HIGH        HIGH   25.355  DrugY
1   47   M     LOW        HIGH   13.093  drugC
2   47   M     LOW        HIGH   10.114  drugC
3   28   F  NORMAL        HIGH    7.798  drugX
4   61   F     LOW        HIGH   18.043  DrugY

shape :  (200, 6)

              Age     Na_to_K
count  200.000000  200.000000
mean    44.315000   16.084485
std     16.544315    7.223956
min     15.000000    6.269000
25%     31.000000   10.445500
50%     45.000000   13.936500
75%     58.000000   19.380000
max     74.000000   38.247000
```

이후 unique() 메서드를 활용해서 열의 레이블 값들의 항목을 확인하였습니다.

```
#Dataset 분석 2
str_label = train.columns
str_label = str_label.drop(labels = ['Age','Na_to_K'])

for label in str_label:
    print(train[label].unique())

print("\nTarget Data Check\n",train['Drug'].value_counts())
```

str_label에 데이터셋의 칼럼 값들을 저장한 후에 수치 데이터에 해당하는 레이블은 제거했습니다. 위의 과정을 통해 칼럼값들의 항목을 확인한 결과 아래와 같았습니다.

```
['F' 'M']
['HIGH' 'LOW' 'NORMAL']
['HIGH' 'NORMAL']
['DrugY' 'drugC' 'drugX' 'drugA' 'drugB']

Target Data Check
 DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
```

이제 위의 데이터들을 encoding 하면 됩니다.

2. Preprocessing

Label Encoding

데이터를 전처리 하기 위해서 먼저 Label Encoding을 진행해주었습니다. Label Encoding이란 모든 데이터를 수치형 데이터로 변환하는 전처리 작업을 뜻합니다. 예를 들어, 원숭이, 호랑이, 토끼라는 레이블이 있다면 각각 0 , 1 ,2로 매칭해주는 과정입니다. 쉽게 말해 문자열 값을 수치형 카테고리로 변환해주는 것입니다. 위의 경우 Age는 원래 수치형 데이터이므로 따로 처리가 필요하지 않고 Sex, BP, Cholesterol, Na_to_K, Drug에 대해서 encoding 과정이 필요합니다.

```
encoder = LabelEncoder()
def label_encoder(label):
    train[label] = encoder.fit_transform(train[label])

for label in str_label:
    label_encoder(label)
```

label_encoder라는 함수를 정의하겠습니다. label_encoder 함수는 문자열을 받아 문자열과 일치하는 데이터셋의 레이블의 범주형 데이터를 숫자로 인코딩 시켜줍니다. 앞서 정의한 str_label을 활용해서 label_encoder 함수에 칼럼 값을 문자열 형태로 전달해주었습니다. 여기서 LabelEncoder()의 fit_transform() 메서드를 통해 인코더에 Series 객체를 전달해서 인코딩된 레이블을 반환받습니다.

Missing Values

이후 Nan 값 처리를 위해 isna() 메서드로 결측치를 확인해보았습니다.

```
train.isna().sum()
```

확인 결과 결측지가 없었다. 따라서 결측치를 따로 처리할 필요가 없었습니다.

Train test split

본격적인 모델 학습에 들어가기에 앞서서 training dataset과 test dataset을 분리했습니다. X에는 Drug를 제외한 feature의 값들을 저장하였고 y에는 정답 데이터인 Drug의 값들을 저장하였습니다. train_test_split 메서드를 활용해서 분리를 진행했고 test_size 옵션을 0.2로 주었습니다.

```
X = train[train.columns.drop('Drug')]
y = train['Drug']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

3. Training and Evaluation of classification model

1) KNN(K Nearest Neighbor)

첫 분류 모델은 knn입니다. knn 알고리즘은 새로운 데이터가 입력되면 그 데이터 주변의 k개의 이웃을 확인합니다. k개의 이웃과의 거리를 구한후 카테고리별로 가까운 이웃을 구하고 새로운 데이터는 가까운 데이터가 가장 많이 분포하는 카테고리에 편입됩니다.

```
for k in range(3,11):
    knn = KNeighborsClassifier(n_neighbors = k)#k를 정해서 knn model의 객체를 만든다.
    knn.fit(X_train, y_train)#knn으로 학습을 시킨다.
    #훈련 결과를 확인하는 과정.
    print("k가 ",k,"일때 훈련 정확도 : ",accuracy_score(y_train,knn.predict(X_train)))
    print("k가 ",k,"일때 검증 정확도 : ",accuracy_score(y_test,knn.predict(X_test)))
    print("---")
```

저는 knn 모델의 k 값은 3~10까지의 값으로 하였습니다. for문을 이용해서 k값을 1씩 증가시키면서 학습하였습니다. 매 반복마다 fit 메서드를 이용해서 학습을 진행시켰습니다. 이후 사이킷런 라이브러리의 metrics의 accuracy_score() 메서드를 활용해서 훈련 정확도와 검증 정확도를 출력하였습니다. 결과는 아래와 같습니다.

```
k가  3 일때 (훈련 정확도, 검증 정확도) :  ( 0.8125 , 0.8 )
k가  4 일때 (훈련 정확도, 검증 정확도) :  ( 0.85 , 0.775 )
k가  5 일때 (훈련 정확도, 검증 정확도) :  ( 0.825 , 0.75 )
k가  6 일때 (훈련 정확도, 검증 정확도) :  ( 0.775 , 0.75 )
k가  7 일때 (훈련 정확도, 검증 정확도) :  ( 0.78125 , 0.725 )
k가  8 일때 (훈련 정확도, 검증 정확도) :  ( 0.7875 , 0.75 )
k가  9 일때 (훈련 정확도, 검증 정확도) :  ( 0.75 , 0.775 )
k가  10 일때 (훈련 정확도, 검증 정확도) :  ( 0.7625 , 0.75 )
```

정확도를 반복적으로 확인한 결과 k가 3이나 4일때 가장 높은 훈련/검증 정확도를 보였습니다.

SVM(Support Vector Machine)

svm은 클래스가 다른 데이터 포인트가 있을때 hyperplane(2차원에서는 직선)으로 클래스를 구합니다. 두 클래스의 데이터 포인트를 분리하는 hyperplane은 매우 많지만 svm의 목표는 그중에서 margin이 가장 큰 것을 고르는 것입니다. 학습 과정에서 각 그룹의 중심점을 찾고 이를 통해 최적의 decision boundary를 찾습니다. 만약 이들이 직선으로 구분되면 선형 분류 모델을 사용하고, 그렇지 않을 경우 비선형 분류 모델을 사용합니다. 학습 과정에서 parameter가 수정이 되고, 이를 위해서 error를 정의해야합니다. svm에는 두가지 error가 있는데 classification error와 margin error입니다. classification error는 주어진 점이 hyperplane의 잘못된 편에 있을때이고, 만약 점이 margin 내부에 있다면 이것은 margin error 입니다. svm에서 hinge loss function을 이용하면 hinge 함수의 hyperparameter C 값으로 두가지 error의 비율을 조절할 수 있습니다. C값이 작으면 SVM이 error에 관대해서 더 나은 일반화를 진행할 수 있지만 C값이 증가하면 SVM은 이러한 오차에 대해 민감해집니다.

> 📌 SVM Error = C * (classification error) + (margin error)

데이터 Scaling

데이터셋을 SVM의 input으로 사용하려면 우선 데이터의 scaling을 진행해야합니다. 데이터에 대해 scaling 하는 이유는 다음과 같습니다. 첫째, 데이터의 scale이 너무 다르면 분류를 할때 bias가 생길수 있기 때문입니다. 둘째, 이상치의 영향력이 감소합니다. 셋째, 연산 효율이 증가합니다. Scale을 하기 위해 scaler를 사용합니다. Scaler의 종류는 아래와 같습니다.

Scaler의 종류

1. Standard Scaler

   기존의 변수의 범위를 정규 분포로 변환시킵니다. 데이터의 최대 최소를 모를때 사용합니다.
   모든 Feature의 평균을 0, 분산을 1로 만듭니다. 이상치가 있다면 평균과 표준편차에 영향을 미치기 때문에
   데이터의 확산에 변화가 생기기 때문에 이상치가 있다면 사용하지 않는 것을 권장합니다.
   공식 : **(x - x의 평균값) / (x의 표준편차)**

2. Normalizer

   Standard Scaler, Robust Scaler, MinMax Scaler는 각 컬럼의 통계치를 이용한다면, Normalizer는 각 로우마다 정규화됩니다. 각 변수의 값을 원점으로부터 1만큼 떨어져있는 범위로 변환하고 정규화를 하게 되면 Spherical contour(구형 윤곽)을 갖게 되는데, 빠르게 변환할 수 있고, overfitting한 확률을 줄일수 있습니다.

3. MinMaxScaler

   데이터의 값들을 0~1 사이의 값으로 변환시키는 것으로 최대값이 1이 되고 최솟값이 0이 됩니다. 각 변수가 정규 분포가 아니거나 표준 편차가 작을 때 효과적입니다. 데이터가 2차원 셋일 경우, x축과 y축 값 모두 0과 1사이의 값을 가지고 이상치가 있는 경우 변환된 값이 매우 좁은 범위로 압축될 수 있어서 Standard Scaler 처럼 이상치에 민감합니다.
   공식 : **(x - x의 최소값) / (x의 최대값 - x의 최소값)**

4. Robust Scaler

모든 feature가 같은 값을 갖는다는게 Standard Scaler와 비슷하지만 평균과 분산 대신에 중앙값(median) 과 사분위수(IQR)를 사용하고 StandardScaler에 의한 표준화보다 동일한 값을 더 넓게 분포합니다. 따라서 Standard Scaler에 비해 이상치에 영향이 적어 이상치를 포함하는 데이터를 표준화하는 경우 효과적입니다.

```
scaler = StandardScaler()#standard scaler 객체를 만든다.
scaler.fit(X)#fit을 통해서 X의 데이터에 대해서 mean, var를 계산을 한다.(평균 표준편차 구한다)
X_scaled = scaler.transform(X)#fit과정이 끝나면 모든 X값에 대해서 앞서 구한 평균, 표준편차로 정규화를 진행한다.
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)
```

앞서 진행한 knn처럼 svm 모델의 데이터셋 역시 train_test_split() 메서드를 활용해서 분리했습니다. 분리할때 사용한 X 데이터는 앞서 언급한 StandardScaler를 활용해서 scaling을 진행한 데이터입니다.

```
for cv in range(1,10):
    svm = LinearSVC(C=cv,loss='hinge')#모델 설정
    svm.fit(X_train,y_train)#fit으로 training input과 output을 넣어줘서 훈련을 수행한다. 즉, parameter를 구한다.
    pred = svm.predict(X_test)
    print("C가 ",cv,"일때 정확도는  :", accuracy_score(pred, y_test))
```

LinearSVC로 SVM 모델을 설정하였습니다. 이후 C값을 1부터 9까지 변경하면서 fit() 메서드로 학습을 진행하였습니다. 결과는 아래와 같습니다.

```
C가  1 일때 정확도는  : 0.9
C가  2 일때 정확도는  : 0.9
C가  3 일때 정확도는  : 0.925
C가  4 일때 정확도는  : 0.925
C가  5 일때 정확도는  : 0.95
C가  6 일때 정확도는  : 0.95
C가  7 일때 정확도는  : 0.95
C가  8 일때 정확도는  : 0.95
C가  9 일때 정확도는  : 0.95
```

정확도를 반복적으로 확인한 결과 C = 5일때 일반적으로 가장 정확도가 높았습니다.

Decision Tree

DT(Decision Tree)는 어떤 질문을 던지고, 그 질문에 대한 답이 참인지 거짓인지에 따라 브랜치를 뻗어나가면서 대상을 좁혀갑니다. DT는 노드와 브랜치로 구성되는데 각 노드에는 기준점이 있어서 기준에 따라 데이터를 분류합니다. DT의 중요한 개념중에 하나는 불순도(impurity) 입니다. 불순도란 해당 범주 안에 서로 다른 데이터가 얼마나 섞여있는지를 의미합니다. 한 카테고리에 하나의 데이터만 있다면 데이터는 불순도는 최소입니다. 불순도를 수치적으로 나타낸 척도가 두가지 있는데 entropy와 gini index입니다. 이 두 척도로 information gain을 얻을 수 있습니다. Information gain은 기준점을 잡기 전과 후에 차이를 통해서 얻는 정보를 의미합니다. DT의 목적은 여러 개의 기준점 중에서 information gain이 극대화 되는 기준점을 구하는 것입니다. 불순도가 높다면 기준점으로 나눈 클래스에서 다시 기준점을 잡고 이 과정을 반복합니다. DT는 데이터를 가장 잘 구분할 수 있는 질문을 기준으로 기준점을 잡지만 질문이 너무 많다면 (다른 말로 트리의 깊이가 너무 깊다면) overfitting될 가능성이 있습니다. 따라서 트리의 깊이를 적당히 조절하는게 중요합니다. 이때 minimum number of leaf를 정해서 그 숫자 이하로 남으면 기준점을 잡는 과정을 중단할 수 있습니다.

```
tree_e = DecisionTreeClassifier(criterion="entropy", max_depth = 4, min_samples_leaf = 3)
tree_g = DecisionTreeClassifier(criterion="gini", max_depth = 4, min_samples_leaf = 3)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)

tree_e.fit(X_train, y_train)
tree_g.fit(X_train, y_train)
```

tree_e에 entropy를 불순도의 척도로 활용하는 의사결정트리 모델 객체를 생성하였고 tree_g에 gini index를 불순도의 척도로 활용하는 모델 객체를 생성하였습니다. 두 객체 모두 깊이를 최대 4로, minimum number of leaf는 3으로 하였습니다.

```
print ("Accuracy of entropy: ", accuracy_score(y_test,tree_e.predict(X_test))*100)
print ("Report of entropy : ",  classification_report(y_test, tree_e.predict(X_test)))

print ("Accuracy of gini : ", accuracy_score(y_test,tree_e.predict(X_test))*100)
print ("Report of gini: ",  classification_report(y_test, tree_e.predict(X_test)))
```

Entropy를 사용한 경우와 gini index를 사용한 경우의 정확도를 아래와 같이 구해보았습니다.

```
Accuracy of entropy:  97.5
Report of entropy :                precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.86      1.00      0.92         6
           2       1.00      0.75      0.86         4
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00        14

    accuracy                           0.97        40
   macro avg       0.97      0.95      0.96        40
weighted avg       0.98      0.97      0.97        40

Accuracy of gini :  97.5
Report of gini:                precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.86      1.00      0.92         6
           2       1.00      0.75      0.86         4
           3       1.00      1.00      1.00         1
           4       1.00      1.00      1.00        14

    accuracy                           0.97        40
   macro avg       0.97      0.95      0.96        40
weighted avg       0.98      0.97      0.97        40
```

반복적으로 두 경우의 정확도를 비교한 결과 모두 정확도가 97.5로 상당히 높았습니다.

4. Conclusion

앞에서는 KNN, SVM, DT 총 3가지의 머신러닝 모델로 하나의 데이터셋에 대해 학습을 진행했습니다. 가장 높은 정확도를 기준으로 KNN부터 차례대로 각각 96.9%, 92.5%, 97.5%였습니다. 모든 머신러닝 모델들이 우수한 결과를 보였지만 경험적으로 보았을때 Decision Tree 모델을 이용한 예측이 가장 정확하

였습니다. 따라서 Drug Classification의 경우에는 Decision Tree 모델을 사용하는 것이 가장 적합합니다.