Drug Type Prediction

Student ID: 62011163

Full Name : Nattapat Charoenruangdet

URL of your colab :
https://colab.research.google.com/drive/1wlWQx9fHEcFDhtyl8xldnIqXF1h6Ecbe?usp=sharing

---

Dataset is at https://rathachai.github.io/DA101/data/drug-dataset.csv

The features are:

- **Age**
- **Sex**
- **BP** : Blood Pressure Levels
- **Cholesterol** : Cholesterol Levels
- **Na_to_K** : Na to Potassium Ration

The label is

- **Drug** : Drug type

Instruction

1. Save a Copy of this notebook using your KMITL account.
2. Change the filename with your student id e.g. "drug (xxxxxxxx)".
3. Fill your student id, fullname, and URL of your colab (after save a copy)
4. Use all features to predict the label following the instruction.
5. Export PDF and submit.
6. Share your notebook with "Anyone on the internet with this link can view", and submit the link.

"The university students who cheat on their exam will not be graded in that semester and the next academic semester suspension will then be awarded as a punishment."

Q-01 : Import all libraries

Every library must be here

Note: Your work is mainly based on the libraries **Pandas** and **Scikit-Learn**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn import neural_network
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
```

Q-02 : Load the dataset into a dataframe **df**

```
df = pd.read_csv('https://rathachai.github.io/DA101/data/drug-dataset.csv')
df
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	15	M	NORMAL	HIGH	9.084	drugX
1	15	F	HIGH	NORMAL	16.725	DrugY
2	15	M	HIGH	NORMAL	17.206	DrugY
3	16	M	LOW	HIGH	12.006	drugC
4	16	F	HIGH	NORMAL	15.516	DrugY
...	...	...	...	...	...	...
195	73	F	NORMAL	HIGH	19.221	DrugY
196	74	M	HIGH	HIGH	9.567	drugB
197	74	M	LOW	NORMAL	11.939	drugX

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      195 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

```
df['Na_to_K'] = df['Na_to_K'].fillna(df.groupby('Drug')['Na_to_K'].transform('mean'))
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

```
lb_make = LabelEncoder()
df["Gender·Code"]·=·lb_make.fit_transform(df["Sex"])
df
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug	Gender Code
0	15	M	NORMAL	HIGH	9.084	drugX	1

```
df·=·pd.get_dummies(df,·columns=["Cholesterol"])
df = pd.get_dummies(df, columns=["BP"])
df
```

	Age	Sex	Na_to_K	Drug	Gender Code	Cholesterol_HIGH	Choleste
0	15	M	9.084	drugX	1	1	
1	15	F	16.725	DrugY	0	0	
2	15	M	17.206	DrugY	1	0	
3	16	M	12.006	drugC	1	1	
4	16	F	15.516	DrugY	0	0	
...	...	...	...	...	...	...	
195	73	F	19.221	DrugY	0	1	
196	74	M	9.567	drugB	1	1	
197	74	M	11.939	drugX	1	0	

Q-03 : Preparing **X** and **y**

- **X** (capital X) is a set of features
- **y** (lower y) is a label

```
X = df.drop(['Drug','Sex'],axis=1)
y = df['Drug']
X
```

	Age	Na_to_K	Gender Code	Cholesterol_HIGH	Cholesterol_NORMAL
0	15	9.084	1	1	0
1	15	16.725	0	0	1
2	15	17.206	1	0	1
3	16	12.006	1	1	0
4	16	15.516	0	0	1

```
y
```

```
0      drugX
1      DrugY
2      DrugY
3      drugC
4      DrugY
       ...
195    DrugY
196    drugB
197    drugX
198    DrugY
199    DrugY
Name: Drug, Length: 200, dtype: object
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=100)
print("Size of X_train", X_train.shape)
print("Size of X_test", X_test.shape)
print("Size of y_train", y_train.shape)
print("Size of y_test", y_test.shape)
```

```
Size of X_train (160, 8)
Size of X_test (40, 8)
Size of y_train (160,)
Size of y_test (40,)
```

Q-04 : To work with the technique: **Decision Tree**

1. Use 5-fold cross-validation
2. Configure any parameters by your own decision
3. Show the model performance of this technique

```
from sklearn import tree
from sklearn.metrics import accuracy_score
ml = tree.DecisionTreeClassifier()
ml.fit(X, y)
y_pred = ml.predict(X_test)
```

```
cm_labels = df["Drug"].unique()
print("***** Confusion Matrix *****")
cm_labels = df["Drug"].unique()
print(cm_labels)
print(confusion_matrix(y_test, y_pred, labels=cm_labels))
# Print Report
print()
print("***** Report *****")
print(classification_report(y_test,y_pred))
print()
print("***** F1 *****")
f1 = f1_score(y_test, y_pred, average='weighted')
print ("F1 = ", f1)
acc = accuracy_score(y_test, y_pred)
print("Accuracy :", acc)
```

```
***** Confusion Matrix *****
['drugX' 'DrugY' 'drugC' 'drugA' 'drugB']
[[10  0  0  0  0]
 [ 0 22  0  0  0]
 [ 0  0  2  0  0]
 [ 0  0  0  3  0]
 [ 0  0  0  0  3]]

***** Report *****
              precision    recall  f1-score   support

       DrugY       1.00      1.00      1.00        22
       drugA       1.00      1.00      1.00         3
       drugB       1.00      1.00      1.00         3
       drugC       1.00      1.00      1.00         2
       drugX       1.00      1.00      1.00        10

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40


***** F1 *****
F1 =  1.0
Accuracy : 1.0
```

Use 5-fold cross-validation

```
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


k = 5
kf = KFold(n_splits=k)
```

```
round_num = 1
F1 = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  ml = tree.DecisionTreeClassifier()
  ml.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = ml.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  f1 = f1_score(y_test, y_pred, average='weighted')
  F1.append(f1)
  print("***** F1 *****")
  print ("F1 = ", f1)
  round_num+=1
  acc = accuracy_score(y_test, y_pred)
  print("Accuracy :", acc)
  print("------------------------------------")
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
***** F1 *****
F1 =  1.0
Accuracy : 1.0
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
***** F1 *****
F1 =  1.0
Accuracy : 1.0
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
***** F1 *****
F1 =  1.0
Accuracy : 1.0
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
```

```
***** F1 *****
F1 =  0.919047619047619
Accuracy : 0.9
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.9747096399535424
Accuracy : 0.975
------------------------------------
```

Q-05 : To work with the technique: **Naive Bayes**

1. Use 5-fold cross-validation
2. Configure any parameters by your own decision
3. Show the model performance of this technique

```
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
clf = GaussianNB()
y_pred = clf.fit(X_train, y_train).predict(X_test)
print("***** F1 *****")
f1 = f1_score(y_test, y_pred, average='weighted')
print ("F1 = ", f1)
acc = accuracy_score(y_test, y_pred)
print("Accuracy :", acc)
```

```
***** F1 *****
F1 =  0.5845324675324676
Accuracy : 0.55
```

Use 5-fold cross-validation

```
round_num = 1
F1 = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  clf = GaussianNB()
```

```
    clf.fit(X_train,y_train)

    # (6.1) to predict from the test set
    y_pred = clf.predict(X_test)

    # (6.2) to evaluate with some evaluation methods
    f1 = f1_score(y_test, y_pred, average='weighted')
    F1.append(f1)
    print("***** F1 *****")
    print ("F1 = ", f1)
    acc = accuracy_score(y_test, y_pred)
    print("Accuracy :", acc)
    print("------------------------------------")
    round_num+=1
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
***** F1 *****
F1 =  0.7850741296018657
Accuracy : 0.775
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
***** F1 *****
F1 =  0.6988076923076924
Accuracy : 0.725
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
***** F1 *****
F1 =  0.7416958041958042
Accuracy : 0.725
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
***** F1 *****
F1 =  0.5127518315018315
Accuracy : 0.525
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.5845324675324676
Accuracy : 0.55
------------------------------------
```

Q-06 : To work with the technique: **Logistic Regression**

1. Use 5-fold cross-validation
2. Configure any parameters by your own decision
3. Show the model performance of this technique

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(max_iter·=·3000)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='weighted')
print ("F1 =", f1)
acc = accuracy_score(y_test, y_pred)
print("Accuracy :", acc)
```

```
F1 = 0.6857721226142279
Accuracy : 0.7
```

```
round_num = 1
F1 = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  clf = LogisticRegression(max_iter = 3000)
  clf.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = clf.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  f1 = f1_score(y_test, y_pred, average='weighted')
  F1.append(f1)
  print("***** F1 *****")
  print ("F1 = ", f1)
  acc = accuracy_score(y_test, y_pred)
  print("Accuracy :", acc)
  print("-----------------------------------")
  round_num+=1
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
***** F1 *****
F1 =  0.8386990927238607
```

```
Accuracy : 0.825
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
***** F1 *****
F1 =  1.0
Accuracy : 1.0
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
***** F1 *****
F1 =  0.9586904761904762
Accuracy : 0.95
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
***** F1 *****
F1 =  0.7217905405405405
Accuracy : 0.775
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.6857721226142279
Accuracy : 0.7
------------------------------------
```

Q-07 : To work with the technique: **Support Vector Machine**

1. Use 5-fold cross-validation
2. Configure any parameters by your own decision
3. Show the model performance of this technique

```
from sklearn import svm
clf = svm.SVC()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='weighted')
print ("F1 =", f1)
acc = accuracy_score(y_test, y_pred)
print("Accuracy :", acc)
```

```
F1 = 0.6762419871794871
Accuracy : 0.75
```

```
round_num = 1
F1 = []
```

```
for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  clf = clf = svm.SVC()
  clf.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = clf.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  f1 = f1_score(y_test, y_pred, average='weighted')
  F1.append(f1)
  print("***** F1 *****")
  print ("F1 = ", f1)
  acc = accuracy_score(y_test, y_pred)
  print("Accuracy :", acc)
  print("------------------------------------")
  round_num+=1
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
***** F1 *****
F1 =  0.6261904761904762
Accuracy : 0.725
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
***** F1 *****
F1 =  0.6333333333333333
Accuracy : 0.725
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
***** F1 *****
F1 =  0.5765151515151515
Accuracy : 0.675
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
***** F1 *****
F1 =  0.5412024756852343
Accuracy : 0.65
```

```
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.6762419871794871
Accuracy : 0.75
------------------------------------
```

Q-08 : To work with the technique: **Neural Network**

1. Use 5-fold cross-validation
2. Use default configuration
3. Show the model performance of this technique

```
from sklearn import neural_network
clf = neural_network.MLPClassifier()
clf.fit(X_train, y_train)
y_pred·=·clf.predict(X_test)
f1·=·f1_score(y_test,·y_pred,·average='weighted')
print·("F1·=",·f1)
acc·=·accuracy_score(y_test,·y_pred)
print("Accuracy·:",·acc)
```

```
F1 = 0.6874897818021758
Accuracy : 0.7
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
```

```
round_num = 1
F1 = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  clf = neural_network.MLPClassifier()
  clf.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = clf.predict(X_test)
```

```
    # (6.2) to evaluate with some evaluation methods
    f1 = f1_score(y_test, y_pred, average='weighted')
    F1.append(f1)
    print("***** F1 *****")
    print ("F1 = ", f1)
    acc = accuracy_score(y_test, y_pred)
    print("Accuracy :", acc)
    print("------------------------------------")
    round_num+=1
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
***** F1 *****
F1 =  0.3059322033898305
Accuracy : 0.475
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
***** F1 *****
F1 =  0.2793103448275862
Accuracy : 0.45
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
***** F1 *****
F1 =  0.5480911983032873
Accuracy : 0.65
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
***** F1 *****
F1 =  0.619404761904762
Accuracy : 0.675
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.32352941176470584
Accuracy : 0.275
------------------------------------
```

Q-09 : To work with the technique: **Neural Network** (tuning)

1. Use 5-fold cross-validation

2. Configure any parameters of the Neural Network until you are satisfied with the model performance.
3. Show the model performance of this technique with tuning

```
from sklearn import neural_network
from tensorflow.keras import layers
from tensorflow.keras import regularizers

clf = neural_network.MLPClassifier(activation = 'relu',hidden_layer_sizes = 4000,random_state
clf.fit(X_train,y_train)
y_pred·=·clf.predict(X_test)
f1·=·f1_score(y_test,·y_pred,·average='weighted')
print·("F1·=",·f1)
acc·=·accuracy_score(y_test,·y_pred)
print("Accuracy·:",·acc)
```

```
F1 = 0.9285771065182831
Accuracy : 0.925
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
```

```
round_num = 1
F1 = []

for train_index, test_index in kf.split(X):
  print("Round", round_num)
  print("  TRAIN:", train_index[0:10],"...")
  print("  TEST:", test_index[0:5],"...")

  # (5.1) to split train and test datasets
  X_train, X_test = X.loc[train_index], X.loc[test_index]
  y_train, y_test = y.loc[train_index], y.loc[test_index]

  # (5.2) to train and create a linear regression model
  clf = neural_network.MLPClassifier(activation = 'relu',hidden_layer_sizes = 4000,random_sta
  clf.fit(X_train,y_train)

  # (6.1) to predict from the test set
  y_pred = clf.predict(X_test)

  # (6.2) to evaluate with some evaluation methods
  f1 = f1_score(y_test, y_pred, average='weighted')
  F1.append(f1)
```

```
    print("***** F1 *****")
    print ("F1 = ", f1)
    acc = accuracy_score(y_test, y_pred)
    print("Accuracy :", acc)
    print("------------------------------------")
    round_num+=1
```

```
Round 1
  TRAIN: [40 41 42 43 44 45 46 47 48 49] ...
  TEST: [0 1 2 3 4] ...
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
***** F1 *****
F1 =  0.9208986643437862
Accuracy : 0.925
------------------------------------
Round 2
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [40 41 42 43 44] ...
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
***** F1 *****
F1 =  1.0
Accuracy : 1.0
------------------------------------
Round 3
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [80 81 82 83 84] ...
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
***** F1 *****
F1 =  0.9278554778554777
Accuracy : 0.9
------------------------------------
Round 4
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [120 121 122 123 124] ...
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
***** F1 *****
F1 =  0.7843624393624393
Accuracy : 0.775
------------------------------------
Round 5
  TRAIN: [0 1 2 3 4 5 6 7 8 9] ...
  TEST: [160 161 162 163 164] ...
***** F1 *****
F1 =  0.9285771065182831
Accuracy : 0.925
------------------------------------
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  % self.max_iter, ConvergenceWarning)
```

Q-10 : Save your Best Model

Save your prediction model that provides that provides the best performance among your experiments (Q04 - Q09).

Save it in to a filename "**clf.model**"

```
import pickle


filename = 'ml.model'


pickle.dump(clf, open(filename, 'wb'))


loaded_clf = pickle.load(open(filename, 'rb'))


y_pred = loaded_clf.predict(X_test)
f1 = f1_score(y_test, y_pred, average='weighted')
print ("F1 =", f1)
acc = accuracy_score(y_test, y_pred)
print("Accuracy :", acc)
```

```
F1 = 0.9285771065182831
Accuracy : 0.925
```

Good Luck

● ✕