day_20_03Assignment

January 22, 2023

Subject of notebook : Apply machine learning algorithm on any dataset other than titanic or iris
Name of the auther : Qadir Shahbaz
Where to contact : qadir_shahbaz@yahoo.co.uk
date : 22/01/2023

0.1 import Libraries

```
[ ]: import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.model_selection import train_test_split
```

0.2 Import dataset

```
[ ]: df = sns.load_dataset("penguins")
```

0.3 Understand the data

```
[ ]: df.head()
```

```
[ ]:   species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
     0  Adelie  Torgersen            39.1           18.7              181.0
     1  Adelie  Torgersen            39.5           17.4              186.0
     2  Adelie  Torgersen            40.3           18.0              195.0
     3  Adelie  Torgersen             NaN            NaN                NaN
     4  Adelie  Torgersen            36.7           19.3              193.0

        body_mass_g     sex
     0       3750.0    Male
     1       3800.0  Female
     2       3250.0  Female
     3          NaN     NaN
     4       3450.0  Female
```

```
[ ]: df.columns
[ ]: Index(['species', 'island', 'bill_length_mm', 'bill_depth_mm',
            'flipper_length_mm', 'body_mass_g', 'sex'],
           dtype='object')
```

```
[ ]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

0.4 Machine learning algorithms

```
[ ]: X = df[["bill_length_mm", "bill_depth_mm","flipper_length_mm" ]]
     X.fillna(value= X.mean(), inplace = True)
     y = df["species"]
```

```
C:\Users\qadir\AppData\Local\Temp\ipykernel_13016\467528285.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.fillna(value= X.mean(), inplace = True)
C:\Users\qadir\AppData\Local\Temp\ipykernel_13016\467528285.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.fillna(value= X.mean(), inplace = True)
C:\Users\qadir\AppData\Local\Temp\ipykernel_13016\467528285.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X.fillna(value= X.mean(), inplace = True)
```

[ ]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   bill_length_mm     344 non-null    float64
 1   bill_depth_mm      344 non-null    float64
 2   flipper_length_mm  344 non-null    float64
dtypes: float64(3)
memory usage: 8.2 KB
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
  ↪recall_score
from sklearn.model_selection import train_test_split
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)

models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
  ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
  ↪'KNN']
```

[ ]:
```
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    models_scores.append([model_name,accuracy])
```

```
c:\Users\qadir\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:

```
sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
for model in sorted_models:
    print("Accuracy Score: ",f'{model[0]} : {model[1]:.2f}')
```

```
Accuracy Score:  Logistic Regression : 0.97
Accuracy Score:  Random Forest : 0.97
Accuracy Score:  Decision Tree : 0.93
Accuracy Score:  KNN : 0.93
Accuracy Score:  SVM : 0.77
```

[ ]:

```
models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
 ↪'KNN']
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    Precision = precision_score(y_test,y_pred, average='micro')
    models_scores.append([model_name,Precision])
```

[ ]:

```
models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
 ↪'KNN']
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    Recall = recall_score(y_test, y_pred)
    models_scores.append([model_name,Recall])
```

[ ]:

```
models = [LogisticRegression(), SVC(), DecisionTreeClassifier(),␣
 ↪RandomForestClassifier(), KNeighborsClassifier()]
model_names = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random Forest',␣
 ↪'KNN']
models_scores = []
for model, model_name in zip(models, model_names):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    F1 = f1_score(y_test, y_pred)
    models_scores.append([model_name,F1])

sorted_models = sorted(models_scores, key=lambda x: x[1], reverse=True)
```

```python
for model in sorted_models:
    print("F1 Score: ",f'{model[0]} : {model[1]:.2f}')
```