OPEN SOURCE SOFTWARE LAB (15B17CI575)

Lab Assignment 7 (Practice Lab)

Topic Coverage: Scikit-learn: Machine Learning in Python

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

Step a) Importing the Dataset

You can import the dataset with Pandas.

```
import pandas as pd
```

# Define path data

```
COLUMNS = ['age','workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship',
'race', 'sex', 'capital_gain', 'capital_loss',
'hours_week', 'native_country', 'label']
```

## Prepare the data

```
features = ['age','workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship',
'race', 'sex', 'capital_gain', 'capital_loss',
'hours_week', 'native_country']

PATH = "https://archive.ics.uci.edu/ml/machine-
learning-databases/adult/adult.data"

df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
```

We first need to identify the continuous and categorical features in the dataset.

# List Categorical features

```
CATE_FEATURES=df_train.iloc[:,:1].select_dtypes('object').colums
print(CATE_FEATURES)
```

#List continuous features

```
CONTI_FEATURES =  df_train._get_numeric_data()
print(CONTI_FEATURES)
```

Note that you need to convert the type of the continuous variables in float format.

```
df_train[CONTI_FEATURES.columns]=df_train[CONTI_FEATURES.columns ].astype('float64')
```

## This gives description of only continuous features

```
df_train.describe()
```

## For description of all features

```
df_train.describe(include='all')
```

## Get the column index of the categorical features

```
conti_features = []
for i in CONTI_FEATURES:
    position = df_train.columns.get_loc(i)
    conti_features.append(position)
print(conti_features)
```

## Get the column index of the categorical features

```
categorical_features = []
for i in CATE_FEATURES:
    position = df_train.columns.get_loc(i)
    categorical_features.append(position)
print(categorical_features)
```

Step b) Data Pre-processing

Each categorical feature is a string. You cannot feed a model with a string value. You need to preprocess the data:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
print(CATE_FEATURES)
df_train['workclass']=le.fit_transform( df_train['workclass'])
df_train['education']=le.fit_transform( df_train['education'])
df_train['marital']=le.fit_transform( df_train['marital'])
df_train['occupation']=le.fit_transform( df_train['occupation'])
df_train['relationship']=le.fit_transform(df_train['relationship
'])
df_train['race']=le.fit_transform( df_train['race'])

df_train['sex']=le.fit_transform( df_train['sex'])

df_train['native_country']=le.fit_transform( df_train['native_country'])

df_train
```

Now we can replace missing value with mean. You can explore other option too.

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values=0, strategy='mean', axis=0)
imp.fit_transform(df_train[features])
```

Now we need to divide data in test and train sets

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df_train[features],df_train['label'],random_sta
te=0)
```

Step c: Model Training and Testing

Classification

i) Naïve Bayes

```
from      sklearn.naive_bayes      import
GaussianNB  from  sklearn.metrics  import
accuracy_score gnb = GaussianNB()
```

```
#train the algorithm on training data and predict using the
testing data
pred = gnb.fit(X_train, y_train).predict(X_test)
print("Naive-Bayes accuracy : ",accuracy_score(y_test,
pred, normalize = True))
```

ii) Linear Support Vector Machine

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
svc_model = LinearSVC(random_state=0)
pred = svc_model.fit(X_train, y_train).predict(X_test)
print("LinearSVC accuracy : ",accuracy_score(y_test, pred,
normalize = True))
```

iii) KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score neigh =
KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
pred = neigh.predict(X_test)
print ("KNeighbors accuracy score :",accuracy_score(y_test,
pred))
```

Clustering:

```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=2,random_state=0).fit(df_train[features
])
kmeans.labels_
kmeans.cluster_centers_
```

Linear Regression:

Here y will not give you label instead it will have some real values.

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X, y)
```

Questions:

1. Write a Python program to create a 2-D array with ones on the diagonal and zeros elsewhere. Now convert the NumPy array to a SciPy sparse matrix in CSR format
2. Write a Python program using SciPy library to view basic statistical details like percentile, mean, std etc. of iris data.
3. Write a Python program using Scikit-learn to split the iris dataset into 70% train data and 30% test data. Out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records. Print both datasets.
4. Write a Python program using Scikit-learn to split the iris dataset into 80% train data and 20% test data. Out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records. Train or fit the data into the model and using the K Nearest Neighbor Algorithm and create a plot of k values vs accuracy. Similarly show the results for other classification techniques as well.
5. In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors'). Write a Python program to get the accuracy of the Logistic Regression and Linear Regression on Boston Housing Dataset. You can load the dataset as:

```
>>>from sklearn.datasets import load_boston
>>> boston_dataset = load_boston()
```