College of Liberal Arts and Sciences
Department of Computer Science and Physics
CSC 410 Data Engineering

Tutorial: Data Preprocessing, visualization, train and test ML model using Python

Data Preprocessing refers to the steps applied to make data more suitable for data science project. The steps used for Data Preprocessing usually fall into two categories: 1. selecting data objects and attributes for the analysis. 2. creating/changing the attributes.

Importing the libraries:

```
# libraries
import numpy as np # used for handling numbers
import pandas as pd # used for handling the dataset
from sklearn.model_selection import train_test_split # used for splitting training
and testing data
from sklearn.preprocessing import LabelEncoder #used for encoding
import matplotlib.pyplot as plt # used for visualization
```

If you are using Jupyter notebook use commands:

```
pip install numpy
pip install pandas
pip install sklearn
```

If you are using PyCharm (windows): File→Settings→Project:.. →Project Interpreter→click on + sign→search for each package.
If you are using PyCharm in Mac, little bit different.

Importing the Dataset:

Download and save the dataset mpg.csv file in the same folder (otherwise you have to use the complete path)

`DataFrame.iloc`: Purely integer-location based indexing for selection by position. `.iloc[]` is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.
`[:,:-1]` means all rows and all column except the last column.

```
#Importing the Dataset:
# to import the dataset into a variable with
# Removing the first unnamed column

dataset = pd.read_csv('mpg.csv',index_col=0)

#print(dataset) #print all
#print(dataset.head()) #print the first 5 rows
#print(dataset.head(10))  # print the first n rows
```

Handling Missing data

```
# Total missing values for each feature
print(dataset.isnull().sum())

# Replace using median
median = dataset['cyl'].median()
dataset['cyl'].fillna(median, inplace=True) #fillna fills the NaN values with a given
number with which you want to substitute
print(dataset.isnull().sum())# Check again any missing or not
```

Encode categorical data with numeric data

```
# Now we are going to encode the categorical data into numerical data type.
print(dataset.dtypes)# See the datatypes

# encode categorical data
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset["manufacturer"]=le.fit_transform(dataset["manufacturer"])
dataset["model"]=le.fit_transform(dataset["model"])
dataset["trans"]=le.fit_transform(dataset["trans"])
dataset["drv"]=le.fit_transform(dataset["drv"])
dataset["fl"]=le.fit_transform(dataset["fl"])
dataset["class"]=le.fit_transform(dataset["class"])
print(dataset.dtypes)
```

Visualize the data

```
#Visualize the data with boxplot
import matplotlib.pyplot as plt

for column in dataset:
    plt.figure()
    dataset.boxplot([column])
    plt.show()

# Visualize all column in one histogram
fig, axes = plt.subplots(ncols=len(dataset.columns), figsize=(10,5))
for col, ax in zip(dataset, axes):
    dataset[col].value_counts().sort_index().plot.bar(ax=ax, title=col)
plt.tight_layout()
plt.show()
# Visualize all column in separate histogram
fig = plt.figure(figsize = (8,8))
ax = fig.gca()
```

```
dataset.hist(ax=ax)
plt.show()
```

Splitting the attributes into independent and dependent attributes:

```
# Splitting the attributes into independent and dependent attributes
X = dataset.iloc[:, :-1].values # attributes to determine independent variable / Class
Y = dataset.iloc[:, -1].values # dependent variable / Class
print(X)
print(Y)
```

Splitting the dataset into training and testing datasets

Any machine learning algorithm needs to be tested for accuracy. In order to do that, we divide our data set into two parts: **training set** and **testing set.** As the name itself suggests, we use the training set to make the algorithm learn the behaviours present in the data and check the correctness of the algorithm by testing on testing set. In Python, we do that as follows:

```
# splitting the dataset into training set and test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

Import your model from Scikit-learn

```
#import your model from scikit-learn
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

Train you model

```
#Train Random Forest Model
rf_classifier = RandomForestClassifier(n_estimators=20, max_depth=4)
rf_model=rf_classifier.fit(X_train,Y_train)
```

Predict from your model using test data

```
#Predict Label
pred=rf_model.predict(X_test)
```

Test your model

```
#Test Model
print("Accuracy:",metrics.accuracy_score(Y_test, pred))
```

Using KFold validation:

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5,random_state=42,shuffle=True)
# Splitting the attributes into independent and dependent attributes
X = dataset.iloc[:, :-1].values # attributes to determine independent variable /
Class
Y = dataset.iloc[:, -1].values # dependent variable / Class
accuracies = []
for train_index, test_index in kf.split(X):

    data_train   = X[train_index]
    target_train = Y[train_index]

    data_test    = X[test_index]
    target_test  = Y[test_index]

    # if needed, do preprocessing here

    clf = RandomForestClassifier(n_estimators=20, max_depth=4)
    clf.fit(data_train,target_train)

    preds = clf.predict(data_test)

    # accuracy for the current fold only
    accuracy = metrics.accuracy_score(target_test,preds)

    accuracies.append(accuracy)

# this is the average accuracy over all folds
average_accuracy = np.mean(accuracies)
print("Using Kfold validation, average accuracy:", average_accuracy)
```