Python
Data Preprocessing

Integrated Master's in Informatics Engineering

Learning and Extraction of Knowledge

2018/2019

<u>**Synthetic Intelligence Lab**</u>

Filipe Gonçalves

César Analide

Data Preprocessing

- Data collection methods are frequently applied in a vague way, resulting in values outside the range (e.g. income: – 100), combinations of impossible data (e.g. gender: male, pregnant: yes), missing values, etc.
- With irrelevant and redundant information present (or noisy and unreliable data), it leads to the process of extracting knowledge during the training phase, becoming a more complex process
- Data pre-processing includes irrelevant/redundant data filtering, instance selection, normalization, transformation, extraction and selection of features, among other mecanisms

Data Preprocessing Techniques

- Standardization & Normalization
- Label Encoding & Binarization
- Missing Value Replacement
- Discretization
- Feature Selection

Standardization & Normalization

- **Standardization**: To transform data so that it has zero mean and unit variance. Also called scaling
  - The values of every feature in a data point can vary between random values. So, it is important to scale them so that this matches specified rules.
- **Normalization**: to transform data so that it is scaled to the [0,1] range.
  - Involves adjusting the values in the feature vector so as to measure them on a common scale.
  - Values of a feature vector are adjusted so that they sum up to 1
  - Normalization is used to ensure that data points do not get boosted due to the nature of their features

Example code (Standardization & Normalization)

```
# Import libraries
import numpy as np
from sklearn import preprocessing

# Create sample data
input_data = np.array([[3, -1.5, 3, -6.4],
                       [0, 3, -1.3, 4.1],
                       [1, 2.3, -2.9, -4.3]])
```

```
# Scale
data_standardized = preprocessing.scale(input_data)
print("\nStandardized data =", data_standardized)

''' Result
Standardized data = [[ 1.33630621 -1.39936232  1.36473933 -0.9258201 ]
                     [-1.06904497  0.87670892 -0.36125453  1.38873015]
                     [-0.26726124  0.5226534  -1.0034848  -0.46291005]]
'''
```

```
data_scaler = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled = data_scaler.fit_transform(input_data)
print("\nMin max scaled data =", data_scaled)

''' Result
Min max scaled data = [[1.   0.   1.   0.]
                       [0.  1.   0.27118644  1.]
                       [0.33333333 0.84444444  0.   0.2]]
'''
```

```
data_normalized = preprocessing.normalize(input_data, norm='l1')
print("\nL1 normalized data =", data_normalized)

''' Result
L1 normalized data = [[ 0.21582734 -0.10791367 0.21582734 -0.46043165]
                      [0.  0.35714286 -0.1547619 0.48809524]
                      [0.0952381 0.21904762 -0.27619048 -0.40952381]]
'''
```

Label Encoding & Binarization

- **Label Encoding:** used to change the word labels into numbers so that the algorithms can understand how to work on them
  - In supervised learning, we mostly come across a variety of labels which can be in the form of numbers or words
  - If they are numbers, then they can be used directly by the algorithm
  - However, many times, labels need to be in readable form. Hence, the training data is usually labelled with words.
- **Binarization:** used to convert a numerical feature vector into a Boolean vector
  - This technique is helpful when we have prior knowledge of the data

Example Code (Label Encoding)

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford', 'bmw']
label_encoder.fit(input_classes)

print("\nClass mapping:")

for i, item in enumerate(label_encoder.classes_):
    print(item, '-->', i)
```

```
''' Result
Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3
'''
```

```
labels = ['toyota', 'ford', 'suzuki']
encoded_labels = label_encoder.transform(labels)
print("\nLabels =", labels)
print("Encoded labels = ", list(encoded_labels))
```

```
''' Result
Labels = ['toyota', 'ford', 'suzuki']
Encoded labels = [3, 1, 2]
'''
```

Example Code (Binarization)

```python
import numpy as np
from sklearn import preprocessing

# Create sample data
input_data = np.array([[3, -1.5, 3, -6.4],
                       [0, 3, -1.3, 4.1],
                       [1, 2.3, -2.9, -4.3]])
```

```python
data_binarized = preprocessing.Binarizer(threshold=1.4).transform(input_data)
print("\nBinarized data = ", data_binarized)
```

```python
''' Result
Binarized data = [[ 1. 0. 1. 0.]
                  [ 0. 1. 0. 1.]
                  [ 0. 1. 0. 0.]]
'''
```

Missing Value Replacement

- Real-world data often has missing values.
- Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.
- Handling missing data is important as many machine learning algorithms do not support data with missing values
- Imputation transformer applied for completing missing values

Example Code (Missing Value Replacement)

```python
import numpy as np
from sklearn.preprocessing import Imputer

X = np.array([[23.56],
              [53.45],
              ['NaN'],
              [44.44],
              [77.78],
              ['NaN']])

print(X)

imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
print("\nImputer = ", imp.fit_transform(X))
```

```
''' Result
Imputer = [[ 22.56 ]
           [ 53.45 ]
           [ 49.8075 ]
           [ 44.44 ]
           [ 77.78 ]
           [ 49.8075 ]]
'''
```

Discretization

- **Data Discretization:** Discretization considers numeric features, and replaces them in the new data set with corresponding categorical features
  - Discretize variable into equal-sized buckets based on rank or based on sample quantiles
  - Can be performed with **cut** and **qcut** functions available in pandas

Example Code (Discretization)

```
factors = np.random.randn(30)

pd.qcut(factors, 5).value_counts()

''' Result - bins will be chosen so that you
have the same number of records in each bin
[-2.578, -0.829]    6
(-0.829, -0.36]     6
(-0.36, 0.366]      6
(0.366, 0.868]      6
(0.868, 2.617]      6
'''
```

```
pd.cut(factors, 5).value_counts()

''' Result - choose the bins to be evenly spaced
according to the values themselves and not the
frequency of those values.

(-2.583, -1.539]    5
(-1.539, -0.5]      5
(-0.5, 0.539]       9
(0.539, 1.578]      9
(1.578, 2.617]      2
'''
```

Feature Selection

- **Feature selection:** process which selects high relevant features in a dataset that contributes most to predict the target variable value / output
  - Irrelevant or partially relevant features can negatively impact model performance
  - Some feature selection techniques available are:
    - ***Univariate Selection***: Univariate feature selector based on statistical tests
      - ***SelectKBest***: Select features according to the k highest scores
    - **Recursive Feature Elimination** (***RFE)***: Feature ranking with Recursive Feature Elimination
    - **Principal Component Analysis (or PCA):** uses linear algebra to transform the dataset into a compressed form
    - ***VarianceThreshold***: Feature selector that removes all low-variance features

Example Code (*Univariate Selection*)

```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)
import pandas
import numpy
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]

# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)

# summarize scores
numpy.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)

# summarize selected features
print(features[0:5,:])

''' Result
[  111.52   1411.887    17.605    53.108  2175.565   127.669     5.393
   181.304]
[[ 148.     0.     33.6    50. ]
 [  85.     0.     26.6    31. ]
 [ 183.     0.     23.3    32. ]
 [  89.    94.     28.1    21. ]
 [ 137.   168.     43.1    33. ]]
'''
```

Example Code (Recursive Feature Elimination)

```
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]

# feature extraction
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d") % fit.n_features_
print("Selected Features: %s") % fit.support_
print("Feature Ranking: %s") % fit.ranking_

''' Result
Num Features: 3
Selected Features: [ True False False False False  True  True False]
Feature Ranking: [1 2 3 5 6 1 1 4]
'''
```

Example Code (Principal Component Analysis)

```
# Feature Extraction with PCA
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA

# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]

# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)

# summarize components
print("Explained Variance: %s") % fit.explained_variance_ratio_
print(fit.components_)

''' Result
Explained Variance: [ 0.88854663  0.06159078  0.02579012]
[[ -2.02176587e-03   9.78115765e-02   1.60930503e-02   6.07566861e-02
    9.93110844e-01   1.40108085e-02   5.37167919e-04  -3.56474430e-03]
 [  2.26488861e-02   9.72210040e-01   1.41909330e-01  -5.78614699e-02
   -9.46266913e-02   4.69729766e-02   8.16804621e-04   1.40168181e-01]
 [ -2.24649003e-02   1.43428710e-01  -9.22467192e-01  -3.07013055e-01
    2.09773019e-02  -1.32444542e-01  -6.39983017e-04  -1.25454310e-01]]
'''
```

Example Code (Variance Threshold)

```
X = [[0, 2, 0, 3],
     [0, 1, 4, 3],
     [0, 1, 1, 3]]

selector = VarianceThreshold()
selector.fit_transform(X)

''' Result
array([[2, 0],
       [1, 4],
       [1, 1]])
'''
```

Python
Data Preprocessing

Integrated Master's in Informatics Engineering

Learning and Extraction of Knowledge

2018/2019

<u>Synthetic Intelligence Lab</u>

Filipe Gonçalves

César Analide

ISLab