2020

Lab 5: Dealing with Missing Values & PCA

Dr. Mohammed Al-Sarem

Taibah University, Information System Department

2/29/2020

Lab 6: Dealing with Missing Values & PCA

Lab Objectives:

Data mining tasks aim at extracting hidden information from the data. According to CRISP-DM, this process goes through well-defined steps. Data preparation is the most time-consuming and labor-intensive task. Before moving ahead, data scientist might explore the data to get a general overview. In the previous lab, the basic exploratory data analysis (EDA) was performed and several plots are depicted. In this lab, you are going to learn how to deal when your data have missing values. In addition, a feature reduction technique is also presented namely: principle component analysis (PCA). After completion of this module, you will be able to:

- Deal with missing values
- Explore PCA technique
- Reduce data dimensionality

Methodology

Similar to what we did in the previous lab, in this lab you have to use the Iris Flower Species Dataset. This dataset involves the prediction of iris flower species. Since the Iris data set comes without any missing data, we first fabricate some data and fill some columns with missing values. Then, the simplest data imputation will be applied. Later, we introduce how to apply PCA to perform data reduction.

In class task:

At the end of this lab, the student will be able to:

- Deal with missing values in python.
- Write a complete Python code that employ PCA.

home task:

Complete your **Course Project** (See Home task in lab 5).

References:

- Missing Data: https://en.wikipedia.org/wiki/Missing_data
- https://github.com/matthewbrems/ODSC-missing-data-may-18/blob/master/Analysis%20with%20Missing%20Data.pdf
- https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

This tutorial is divided into the following parts:

- Exploring briefly how to deal with missing values;
- Working with missing values on the iris dataset.
- Exploring how to execute PCA technique to reduce data dimensionality.

1. Exploratory Data Analysis (EDA): Missing Value

1.1. What is the source of missing values?

There are several reasons why the data is missing. Some of these sources are just simple random mistakes. Others are made intentionally:

- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

1.2. Why dealing with missing values is important?

Exploratory Data Analysis or (EDA) is understanding the data sets by summarizing their main characteristics often plotting them visually. However, data in real world tend to be incomplete, noisy, and inconsistent and it is an important task of a Data scientist to prepossess the data by **filling missing values**. This is very important step that you have to handle before move forward since ignoring missing values might lead to wrong prediction or classification for any given model being used.

1.3. How to deal with missing data?

There are three main approaches that the data scientist can follow to handle the missing values: (1) imputation- the idea behind this approach is to fill new values in place of the missing data; (2) omission- in this case, samples with invalid data are discarded from further analysis; and (3) analysis- by directly applying methods unaffected by the missing values.

2. Handling Missing Value in Python

3. Working on IRIS data set

Let's first import the required libraries to *jupyter* notebook. Similar to what you did in the previous lab, you have to import the following libraries: numpy, pandas, matplotlib and seaborn. The code below presents how to import these libraries:

Import the required libraries

```
In [*]: # Importing required libraries.
        import pandas as pd
        import numpy as np
        import seaborn as sns #visualisation
        import matplotlib.pyplot as plt #visualisation
        %matplotlib inline
        sns.set(color_codes=True)
```

2.1 Loading the data into the data frame

Now we need to create a pandas data frame from the iris dataset. load_iris is a function in sklearn.datasets which is a bunch of data and target variables and the description of data set.

```
In [6]: dataset=datasets.load_iris()

        data=pd.DataFrame(dataset['data'],columns=['Petal length','Petal Width','Sepal Length','Sepal Width'])
        data['Species']=dataset['target']
        data['Species']=data['Species'].apply(lambda x: dataset['target_names'][x])
```

- we use DataFrame function in the pandas library to convert the array of data to Pandas Dataframe with the columns "Petal length","Petal Width","Sepal Length","Sepal Width" and create a new column "Species" with target values from the dataset.
- Also, lambda function is applied to convert the target values which are 0,1,2 to the corresponding target values("setosa","versicolor","virginica") for better understanding.

`Display the first 10 rows!`

```
In [3]: dataset=datasets.load_iris()

        data=pd.DataFrame(dataset['data'],columns=['Petal length','Petal Width','Sepal Length','Sepal Width'])
        data['Species']=dataset['target']
        data['Species']=data['Species'].apply(lambda x: dataset['target_names'][x])
        data.head(10)
```

Out[3]:

	Petal length	Petal Width	Sepal Length	Sepal Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

- Use `describe()` function to see the statistics of the dataset such as mean, median, mode, standard deviation etc.

2.2 Looking for missing data

- Let's see if there are any **null** values present in the dataset. If there are any null values present, then we need to follow one of the below steps:
  - Drop records which have NA values
  - Substitute mean value (mean if the feature is numerical or mode if the feature is categorical) of the column/feature for the NA values
  - Fill the NA values with "?" or -9999

Exercise 1.1: Use `data.isnull().sum()`! And write your observation below.

```
In [5]: print(modData.isnull().sum())

Petal length    0
Petal Width     0
Sepal Length    0
Sepal Width     0
Species         0
dtype: int64
```

there is no missing value and the function it show where is the missing value if there was .

You might observe that iris data set has not missing values. To move forward in this lab, we add some rows with missing values. For this purpose, add new row with missing values as follows:

```
In [66]: modData = data.append({'Petal length' : np.nan , 'Petal Width' : 3.6, 'Sepal Length': 0,
                                'Sepal Width': 0.2, 'Species': 'setosa' } , ignore_index=True)
         modData.describe()
```

Out[66]:

	Petal length	Petal Width	Sepal Length	Sepal Width
count	150.000000	151.000000	151.000000	151.000000
mean	5.843333	3.060927	3.733113	1.192715
std	0.828066	0.436850	1.785785	0.764033
min	4.300000	2.000000	0.000000	0.100000
25%	5.100000	2.800000	1.550000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.350000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Now, our new dataset "modData" has new row with a missing value. Can you determine in which column is it? petal length What does `np.nan` mean? NaN is for not a number

Exercise 1.2: Repeat the task in Exercise 1.1 and write your observation below.

```
In [5]: print(modData.isnull().sum())

Petal length    1
Petal Width     0
Sepal Length    0
Sepal Width     0
Species         0
dtype: int64
```

it shows that there is missing value in petal length column

We can also define which column has a zero value. The "Sepal length" column has a value =0.

```
In [72]: print('Columns with missing values')
         print(modData.isnull().sum())
         print('\n Columns with zero values')
         print((modData[['Petal length','Petal Width','Sepal Length','Sepal Width','Species']]==0).sum())

Columns with missing values
Petal length    1
Petal Width     0
Sepal Length    0
Sepal Width     0
Species         0
dtype: int64

 Columns with zero values
Petal length    0
Petal Width     0
Sepal Length    1
Sepal Width     0
Species         0
dtype: int64
```

In Python, specifically `Pandas, NumPy and Scikit-Learn`, we mark missing values as `NaN`. Values with a `NaN` value are ignored from operations like sum, count, etc. We can mark values as `NaN` easily with the Pandas `DataFrame` by using the replace() function on a subset of the columns we are interested in. After we have marked the missing values, we can use the `isnull()` function to mark all of the `NaN` values in the dataset as `True` and get a count of the missing values for each column.

```
In [75]: modData[['Petal length',
                  'Petal Width',
                  'Sepal Length',
                  'Sepal Width','Species']] = modData[['Petal length',
                                                       'Petal Width',
                                                       'Sepal Length',
                                                       'Sepal Width',
                                                       'Species']].replace(0, np.NaN)
         print('Columns with missing values')
         print(modData.isnull().sum())

Columns with missing values
Petal length    1
Petal Width     0
Sepal Length    1
Sepal Width     0
Species         0
dtype: int64
```

As stated earlier, Data scientist has three options to handle missing data. In this lab we only present imputation approach.

2.3 Impute Missing Values

Imputing refers to using a model to replace missing values. There are many options we could consider when replacing a missing value, for example:

- A constant value that has meaning within the domain, such as 0, distinct from all other values.
- A value from another randomly selected record.
- A mean, median or mode value for the column.
- A value estimated by another predictive model.

Pandas provides the `fillna()` function for replacing missing values with a specific value. For example, we can use `fillna()` to replace missing values with the mean value for each column, as follows:

```
In [77]: # fill missing values with mean column values
         modData.fillna(modData.mean(), inplace=True)
         # count the number of NaN values in each column
         print(modData.isnull().sum())

         Petal length    0
         Petal Width     0
         Sepal Length    0
         Sepal Width     0
         Species         0
         dtype: int64
```

You can instead of use `mean()`, you might to fill the missing data with specific value. In this case, just pass through `fillna()` method, the value you want to pass as follows: `yourdataframe.fillna(value, inplace=True).`

`Exercise 1.3:` `Instead of using mean(). Try to use Median.`

```
In [13]: modData.fillna(modData.median(), inplace=True)
         print(modData.isnull().sum())

         Petal length    0
         Petal Width     0
         Sepal Length    0
         Sepal Width     0
         Species         0
         dtype: int64
```

4. Dimensionality Reduction: PCA

In this section, we learn how to apply principle component analysis (PCA) technique to: visualize your data as well as to speed up machine learning algorithms. To understand the value of using PCA for data visualization, the first part of this lab goes over a basic visualization of the IRIS dataset after applying PCA.

4.1. PCA for Data Visualization

In several machine learning applications, sometimes it is helpful to visualize data. Visualizing 2 or 3 dimensional data is not that challenging. However, as you noted that IRIS dataset is 4 dimensional. Thus, we can use PCA to reduce that 4 dimensional data into 2 or 3 dimensions so that we can plot and hopefully understand the data better.

4.1.1. Another way to load Data

Let we do this task from scratch so that we reload the data again as follows:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# Load dataset into Pandas DataFrame
PCA_df = pd.read_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])
```

In the previous labs, we demonstrated how to import Iris data set using **`numpy`** library and also we used **`skleaen.datasets`** to import the data. The code above presents another way to import the data however in this case by downloading the data directly from the repository (https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data).

4.1.2. Standardize the Data

PCA is effected by scale so you need to scale the features in your data before applying PCA. We can use `MinMaxScaler` to standardize the dataset's features onto a given range and `StandardScaler` to standardize the dataset's features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of many machine learning algorithms.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
# Separating out the features
x = PCA_df.loc[:, features].values
# Separating out the target
y = PCA_df.loc[:,['target']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
print (x)
```

4.1.3. PCA Projection to 2D

The original data has 4 columns (sepal length, sepal width, petal length, and petal width). Our target in this section is to project the original data which is 4 dimensional into 2 dimensions. I should note that after dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

-

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

	sepal length	sepal width	petal length	petal width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

PCA
(2 components)

	principal component 1	princial component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

PCA and Keeping the Top 2 Principal Components

Now we have to concatenate DataFrame along axis = 1. finalDf is the final DataFrame before plotting the data.

```
finalDf = pd.concat([principalDf, PCA_df[['target']]], axis = 1)
```

principalDf

	principal component 1	principal component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

df[['target']]

	target
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

pd.concat(axis = 1)

finalDf

	principal component 1	principal component 2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa

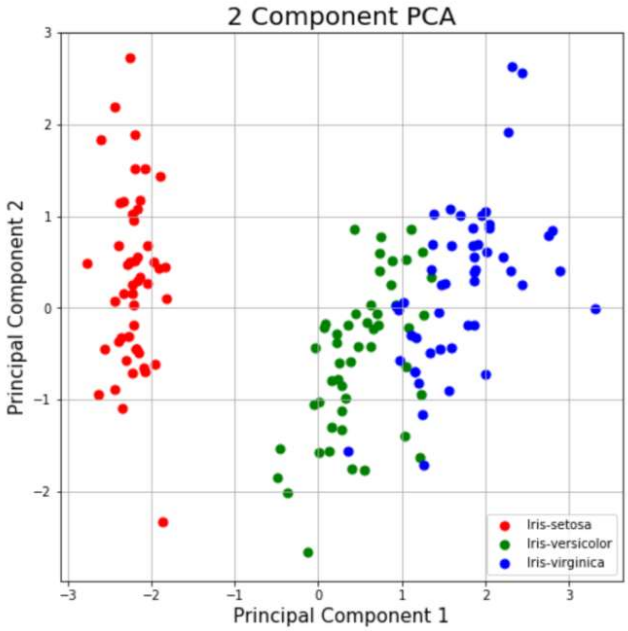
Concatenating dataframes along columns to make finalDf before graphing

4.1.4. Visualize 2D Projection

This section is just plotting 2 dimensional data. Notice on the graph below that the classes seem well separated from each other.

```
import matplotlib.pyplot as plt
%matplotlib inline
#sns.set(color_codes=True)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



4.1.5. Explained Variance

The explained variance tells you how much information (variance) can be attributed to each of the principal components. This is important as while you can convert 4 dimensional space to 2 dimensional space, you lose some of the variance (information) when you do this. By using the attribute `explained_variance_ratio_`, you can see that the first principal component contains 72.77% of the variance and the second principal component contains 23.03% of the variance. Together, the two components contain 95.80% of the information.

```
In [15]: print(pca.explained_variance_ratio_)
         [0.72770452 0.23030523]
```

4.2. PCA to Speed-up Machine Learning Algorithms

We skip this until now. However, in the for coming labs we will demonstrate how can PCA be useful for speeding up machine learning algorithms. Until that time, your task is to work on your data that is chosen to finalize the course's project.

Good luck