APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 1)

03/10/19

LOADING ML DATA AND DESCRIPTIVE STATISTICS

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask your teaching assistant or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. CSV Files

You must be able to load your data before you can start your machine learning project. The most common format for machine learning data is CSV files. There are a number of ways to load a CSV file in Python. In the first section of this lab you will learn three ways that you can use to load your CSV data in Python:

1. Load CSV Files with the Python Standard Library.
2. Load CSV Files with NumPy.
3. Load CSV Files with Pandas.

There are a number of considerations when loading your machine learning data from CSV files. For reference, you can learn a lot about the expectations for CSV files by reviewing the CSV request for comment titled *Common Format and MIME Type for Comma-Separated Values (CSV) Files (URL:* https://tools.ietf.org/html/rfc4180)

File Header. Does your data have a file header? If so this can help in automatically assigning names to each column of data. If not, you may need to name your attributes manually. Either way, you should explicitly specify whether or not your CSV file had a file header when loading your data.

Comments. Does your data have comments? Comments in a CSV file are indicated by a hash (#) at the start of a line. If you have comments in your file, depending on the method used to load your data, you may need to indicate whether or not to expect comments and the character to expect to signify a comment line.

Delimiter. The standard delimiter that separates values in fields is the comma (,) character. Your file could use a different delimiter like tab or white space in which case you must specify it explicitly.

Quotes. Sometimes field values can have spaces. In these CSV files the values are often quoted. The default quote character is the double quotation marks character. Other characters can be used, and you must specify the quote character used in your file.

Pima Indians Dataset

The Pima Indians dataset is used to demonstrate data loading in this lab. It will also be used in many of the labs to come. This dataset describes the medical records for Pima Indians and whether or not each patient will have an onset of diabetes within five years. As such it is a classification problem. It is a good dataset for demonstration because all of the input attributes are numeric and the output variable to be predicted is binary (0 or 1).

Below lists the eight attributes for the dataset:

1. Number of times pregnant.
2. Plasma glucose concentration 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skin fold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (BMI).
7. Diabetes pedigree function.
8. Age (years).
9. Class, onset of diabetes within five years.

Given that all attributes are numerical makes it easy to use directly with machine learning algorithms that expect numerical inputs and output values. This dataset will also be used for the first few labs in the module, so keep it handy. Below is a sample of the dataset showing the first 5 rows of the 768 instances:

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
```

The baseline accuracy if all predictions are made as no onset of diabetes is 65.1%. Top results on the dataset are in the range of 77.7% accuracy using 10-fold cross-validation. You can learn more about the dataset below.

https://www.kaggle.com/uciml/pima-indians-diabetes-database

The Python API provides the module CSV and the function *reader*() that can be used to load CSV files. Once loaded, you can convert the CSV data to a NumPy array and use it for machine learning. For example, you can download the Pima Indians dataset into your local directory with the filename *pima-indians-diabetes.data.csv*. All fields in this dataset are numeric and there is no header line.

Before running the below code block, you will need to create a Jupyter notebook and upload the Pima Indians dataset (download it from Moodle). First, open Anaconda Navigator in your PC. Click the Launch button on Jupyter notebook.

Creating a new Jupyter Notebook is easy. Just use the New dropdown menu and select option Python 3 to open a new Jupyter Notebook for Python.

To upload the dataset, just use the Upload button.

The notebook itself consists of cells. A first empty cell is already available after having created the new notebook.

Copy the below code block and past on to the first cell.

```
# Load CSV Using Python Standard Library
import csv
import numpy
filename = 'pima-indians-diabetes.data.csv'
raw_data = open(filename, 'rt')
reader = csv.reader(raw_data, delimiter=',', quoting=csv.QUOTE_NONE)
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)
```

For open mode, please refer to below.

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating (reading and writing)
'U'	universal newlines mode (deprecated)

In *csv.reader* module, no automatic data type conversion is performed unless the QUOTE_NONNUMERIC format option is specified (in which case unquoted fields are transformed into floats).

.astype method will copy of the array, cast to a specified type.

The above code block loads an object that can iterate over each row of the data and can easily be converted into a NumPy array. Running the code block prints the shape of the array as below.

(768, 9)

You can load your CSV data using NumPy and the $numpy.loadtxt()$ function. This function assumes no header row and all data has the same format. The example below assumes that the file *pima-indians-diabetes.data.csv* is in your current working directory.

```
# Load CSV using NumPy
from numpy import loadtxt
filename = 'pima-indians-diabetes.data.csv'
raw_data = open(filename, 'rt')
data = loadtxt(raw_data, delimiter=",")
print(data.shape)
```

Running the above code block will load the file as a $numpy.ndarray$ and print the shape of the data.

This can be modified to load the same dataset directly from a URL as follows:

```
# Load CSV from URL using NumPy
from numpy import loadtxt
from urllib import urlopen
url = 'https://goo.gl/XXXXX'
raw_data = urlopen(url)
dataset = loadtxt(raw_data, delimiter=",")
print(dataset.shape)
```

For more information on the *numpy.loadtxt()* function see below the API documentation.

https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.loadtxt.html

You can load your CSV data using Pandas and the $pandas.read\_csv()$ function. This function is very flexible and is perhaps my recommended approach for loading your machine learning data. The function returns a $pandas.DataFrame$ that you can immediately start summarising and plotting. The example below assumes that the *pima-indians-diabetes.data.csv* file is in the current working directory.

For more information on the *pandas.DataFrame* see below the API documentation.

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

```
# Load CSV using Pandas
from pandas import read_csv
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
print(data.shape)
```

Note that in this code block we explicitly specify the names of each attribute to the $DataFrame$.

We can also modify this example to load CSV data directly from a URL.

```
# Load CSV using Pandas from URL
from pandas import read_csv
url = 'https://goo.gl/XXXXXXX'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(url, names=names)
print(data.shape)
```

To learn more about the $pandas.read\_csv()$ function you can refer to below the API documentation.

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Verify your understanding:

(a) How many instances and variables in the Pima Indian dataset?

(b) Are the variables all numeric? If not what would you do?

2. Descriptive Statistics

There is no substitute for looking at the raw data. Looking at the raw data can reveal insights that you cannot get any other way. It can also plant seeds that may later grow into ideas on how to better pre-process and handle the data for machine learning tasks. You can review the first 20 rows of your data using the $head()$ function on the Pandas DataFrame.

```
# View first 20 rows
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
peek = data.head(20)
print(peek)
```

You can see that the first column lists the row number, which is handy for referencing a specific observation.

```
    preg  plas  pres  skin  test  mass   pedi  age  class
0      6   148    72    35     0  33.6  0.627   50      1
1      1    85    66    29     0  26.6  0.351   31      0
2      8   183    64     0     0  23.3  0.672   32      1
3      1    89    66    23    94  28.1  0.167   21      0
4      0   137    40    35   168  43.1  2.288   33      1
5      5   116    74     0     0  25.6  0.201   30      0
6      3    78    50    32    88  31.0  0.248   26      1
7     10   115     0     0     0  35.3  0.134   29      0
8      2   197    70    45   543  30.5  0.158   53      1
9      8   125    96     0     0   0.0  0.232   54      1
10     4   110    92     0     0  37.6  0.191   30      0
11    10   168    74     0     0  38.0  0.537   34      1
12    10   139    80     0     0  27.1  1.441   57      0
13     1   189    60    23   846  30.1  0.398   59      1
14     5   166    72    19   175  25.8  0.587   51      1
15     7   100     0     0     0  30.0  0.484   32      1
16     0   118    84    47   230  45.8  0.551   31      1
17     7   107    74     0     0  29.6  0.254   31      1
18     1   103    30    38    83  43.3  0.183   33      0
19     1   115    70    30    96  34.6  0.529   32      1
```

Output of reviewing the first few rows of data.

Dimensions of your data

You must have a very good handle on how much data you have, both in terms of rows and columns.

- Too many rows and algorithms may take too long to train. Too few and perhaps you do not have enough data to train the algorithms.
- Too many features and some algorithms can be distracted or suffer poor performance due to the curse of dimensionality (will be explained in the next lecture).

You can review the shape and size of your dataset by printing the shape property on the Pandas DataFrame.

```
# Dimensions of your data
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
shape = data.shape
print(shape)
```

The results are listed in rows then columns. You can see that the dataset has 768 rows and 9 columns.

Data Type for Each Attribute

The type of each attribute is important. Strings may need to be converted to floating point values or integers to represent categorical or ordinal values. You can get an idea of the types of attributes by peeking at the raw data, as above. You can also list the data types used by the `DataFrame` to characterise each attribute using the `dtypes` property.

```
# Data Types for Each Attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
types = data.dtypes
print(types)
```

You can see that most of the attributes are integers and that $mass$ and $pedi$ are floating point types.

```
preg      int64
plas      int64
pres      int64
skin      int64
test      int64
mass    float64
pedi    float64
age       int64
class     int64
dtype: object
```

Output of reviewing the data types of the data

Descriptive Statistics

Descriptive statistics can give you great insight into the shape of each attribute. Often you can create more summaries than you have time to review. The `describe()` function on the Pandas DataFrame lists 8 statistical properties of each attribute. They are:

- Count.
- Mean.
- Standard Deviation.
- Minimum Value.
- 25th Percentile.
- 50th Percentile (Median).
- 75th Percentile.
- Maximum Value.

```
# Statistical Summary
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
description = data.describe()
print(description)
```

You can see that you do get a lot of data. You will note some calls to pandas.set_option() in the recipe to change the precision of the numbers and the preferred width of the output. This is to make it more readable for this example. When describing your data this way, it is worth taking some time

and reviewing observations from the results. This might include the presence of *NA* values for missing data or surprising distributions for attributes.

For more information on pandas.set_option() please see below.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.set_option.html

	preg	plas	pres	skin	test	mass	pedi	age	class
count	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000	768.000
mean	3.845	120.895	69.105	20.536	79.799	31.993	0.472	33.241	0.349
std	3.370	31.973	19.356	15.952	115.244	7.884	0.331	11.760	0.477
min	0.000	0.000	0.000	0.000	0.000	0.000	0.078	21.000	0.000
25%	1.000	99.000	62.000	0.000	0.000	27.300	0.244	24.000	0.000
50%	3.000	117.000	72.000	23.000	30.500	32.000	0.372	29.000	0.000
75%	6.000	140.250	80.000	32.000	127.250	36.600	0.626	41.000	1.000
max	17.000	199.000	122.000	99.000	846.000	67.100	2.420	81.000	1.000

Output of reviewing a statistical summary of the data.

Class Distribution (Classification only)

On classification problems you need to know how balanced the class values are. Highly imbalanced problems (a lot more observations for one class than another) are common and may need special handling in the data preparation stage of your project. You can quickly get an idea of the distribution of the class attribute in Pandas.

```
# Class Distribution
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
class_counts = data.groupby('class').size()
print(class_counts)
```

You can see that there are nearly double the number of observations with class 0 (no onset of diabetes) than there are with class 1 (onset of diabetes).

```
class
0    500
1    268
```

Output of reviewing a class breakdown of the data

Correlations between Attributes

Correlation refers to the relationship between two variables and how they may or may not change together. The most common method for calculating correlation is *Pearson's Correlation Coefficient*, that assumes a normal distribution of the attributes involved. A correlation of -1 or 1 shows a full negative or positive correlation respectively. Whereas a value of 0 shows no correlation at all. Some machine learning algorithms like linear and logistic regression can suffer poor performance if there are highly correlated attributes in your dataset. As such, it is a good idea to review all of the pairwise correlations of the attributes in your dataset. You can use the $corr()$ function on the Pandas $DataFrame$ (a multi-dimensional array where the rows and the columns can be labelled) to calculate a correlation matrix.

```
# Pairwise Pearson correlations
from pandas import read_csv
from pandas import set_option
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
```

```
data = read_csv(filename, names=names)
set_option('display.width', 100)
set_option('precision', 3)
correlations = data.corr(method='pearson')
print(correlations)
```

The matrix lists all attributes across the top and down the side, to give correlation between all pairs of attributes (twice, because the matrix is symmetrical). You can see the diagonal line through the matrix from the top left to bottom right corners of the matrix shows perfect correlation of each attribute with itself.

```
        preg   plas   pres   skin   test   mass   pedi    age  class
preg   1.000  0.129  0.141 -0.082 -0.074  0.018 -0.034  0.544  0.222
plas   0.129  1.000  0.153  0.057  0.331  0.221  0.137  0.264  0.467
pres   0.141  0.153  1.000  0.207  0.089  0.282  0.041  0.240  0.065
skin  -0.082  0.057  0.207  1.000  0.437  0.393  0.184 -0.114  0.075
test  -0.074  0.331  0.089  0.437  1.000  0.198  0.185 -0.042  0.131
mass   0.018  0.221  0.282  0.393  0.198  1.000  0.141  0.036  0.293
pedi  -0.034  0.137  0.041  0.184  0.185  0.141  1.000  0.034  0.174
age    0.544  0.264  0.240 -0.114 -0.042  0.036  0.034  1.000  0.238
class  0.222  0.467  0.065  0.075  0.131  0.293  0.174  0.238  1.000
```

Skew of Univariate Distributions

Skew refers to a distribution that is assumed Gaussian (normal or bell curve) that is shifted or squashed in one direction or another. Many machine learning algorithms assume a Gaussian distribution. Knowing that an attribute has a skew may allow you to perform data preparation to correct the skew and later improve the accuracy of your models. You can calculate the skew of each attribute using the $skew()$ function on the Pandas $DataFrame$.

```
# Skew for each attribute
from pandas import read_csv
filename = "pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = read_csv(filename, names=names)
skew = data.skew()
print(skew)
```

The skew result show a positive (right) or negative (left) skew. Values closer to zero show less skew.

```
preg     0.901674
plas     0.173754
pres    -1.843608
skin     0.109372
test     2.272251
mass    -0.428982
pedi     1.919911
age      1.129597
class    0.635017
```

Output of reviewing skew of attribute distribution in the data.

Verify your understanding:

(c) Is the dataset balanced?

(d) Does any variable have unusual distribution? If yes what could be the problem?

(e) What do you think should be done to rectify the situation?

(f) Are there any variables have similar information? If yes then what should be done?

3. References

[1]. Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.
[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.
[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
[5]. Brownlee, J., 2014. Machine learning mastery.
[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.