How To Load Machine Learning Data in Python

For any machine learning algorithm to be implemented, you require apt datasets to work on. The following are the repositories that consist of various datasets for different types of machine learning applications.

UCI repository : https://archive.ics.uci.edu/ml/datasets.html

Kaggle : https://www.kaggle.com/datasets

Kdnuggets : https://www.kdnuggets.com/datasets/index.html

There are various file formats available for datasets.

CSV (Comma Separated Values)

xls (Excel file)

.txt (Plain text)

.json, docx, etc.

Most common is .CSV file(Comma Separated File)

a. CSV File Header

In a .CSV file for dataset, sometimes a file header may be present. For instance, in the following snapshot, the first line is the file header.

```
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,Diabetes
PedigreeFunction,Age,Outcome
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
```

```
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
```

This can help in automatically assigning names to each column of data. If file header is not present, you may need to name the attributes manually.

Ex Download the following three datasets and understand the .csv format.

https://archive.ics.uci.edu/ml/datasets/glass+identification

https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/

b. Delimiter

The standard delimiter that separates values in fields is the comma (“,”) character.

Your file could use a different delimiter like tab (“\t”) in which case you must specify it explicitly.

c. 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 “\””. Other characters can be used, and you must specify the quote character used in your file.

You must be able to load your data before you can start your machine learning project.

There are 3 specific techniques that you can use to load .csv file

- Load CSV with Python Standard Library.
- Load CSV File With NumPy.
- Load CSV File With Pandas.

a. Load CSV with Python Standard Library

The Python API provides the module *CSV* and the function *reader()* that can be used to load CSV files.

Once loaded, you convert the CSV data to a NumPy array and use it for machine learning.

For example, you can download the dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/ into your local directory. All fields are numeric and there is no header line. Following code will load the CSV file and convert it to a NumPy array.

```
# Load CSV (using python)
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)
```

The example loads an object that can iterate over each row of the data and can easily be converted into a NumPy array. Execute the program and see the result.

b. Load CSV File With NumPy

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* which you downloaded previously is in your current working directory.

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

c. Load CSV File With Pandas

You can load your CSV data using Pandas and the *pandas.read_csv()* function.

This function is very flexible and a recommended approach for loading your machine learning data. The function returns a pandas.DataFrame that you can immediately start summarizing and plotting.

The example below assumes that the ‘*pima-indians-diabetes.data.csv*‘ file is in the current working directory.

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

Data can also be directly loaded from URL as follows.

```
import pandas
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pandas.read_csv(url, names=names)
print(data.shape)
```

1. Load CSV File

The first step is to load the CSV file.

We will use the csv module that is a part of the standard library.

The **reader()** function in the csv module takes a file as an argument.

We will create a function called **load_csv()** to wrap this behavior that will take a filename and return our dataset. We will represent the loaded dataset as a list of lists. The first list is a list of observations or rows, and the second list is the list of column values for a given row.

Below is the complete function for loading a CSV file.

```
from csv import reader

# Load a CSV file
def load_csv(filename):
```

We can test this function by loading the Pima Indians dataset. Download the dataset and place it in the current working directory with the name **pima-indians-diabetes.csv**. Open the file and delete any empty lines at the bottom.

Taking a peek at the first 5 rows of the raw data file we can see the following:

```
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
```

The data is numeric and separated by commas and we can expect that the whole file meets this expectation.

Let’s use the new function and load the dataset. Once loaded we can report some simple details such as the number of rows and columns loaded.

Putting all of this together, we get the following:

```
from csv import reader

# Load a CSV file
def load_csv(filename):
```

Running this example we see:

```
Loaded data file pima-indians-di
```

A limitation of this function is that it will load empty lines from data files and add them to our list of rows. We can overcome this by adding rows of data one at a time to our dataset and skipping empty rows.

Below is the updated example with this new improved version of the **load_csv()** function.

```
# Example of loading Pima Indiai
from csv import reader

# Load a CSV file
```

Running this example we see:

```
Loaded data file pima-indians-di
```

2. Convert String to Floats

Most, if not all machine learning algorithms prefer to work with numbers.

Specifically, floating point numbers are preferred.

Our code for loading a CSV file returns a dataset as a list of lists, but each value is a string. We can see this if we print out one record from the dataset:

```
print(dataset[0])
```

```
print(dataset[0])
```

This produces output like:

```
['6', '148', '72', '35', '0', '33.6', '0.(
```

We can write a small function to convert specific columns of our loaded dataset to floating point values.

Below is this function called **str_column_to_float()**. It will convert a given column in the dataset to floating point values, careful to strip any whitespace from the value before making the conversion.

```
def str_column_to_float(datase
        for row in dataset:
                row[columr
```

We can test this function by combining it with our load CSV function above, and convert all of the numeric data in the Pima Indians dataset to floating point values.

The complete example is below.

```
from csv import reader

# Load a CSV file
def load_csv(filename):
```

Running this example we see the first row of the dataset printed both before and after the conversion. We can see that the values in each column have been converted from strings to numbers.

```
Loaded data file pima-indians-di
['6', '148', '72', '35', '0', '33.6', '0.(
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6
```

3. Convert String to Integers

The iris flowers dataset is like the Pima Indians dataset, in that the columns contain numeric data.

The difference is the final column, traditionally used to hold the outcome or value to be predicted for a given row. The final column in the iris flowers data is the iris flower species as a string.

Download the dataset and place it in the current working directory with the file name **iris.csv**. Open the file and delete any empty lines at the bottom.

For example, below are the first 5 rows of the raw dataset.

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
```

Some machine learning algorithms prefer all values to be numeric, including the outcome or predicted value.

We can convert the class value in the iris flowers dataset to an integer by creating a map.

1. First, we locate all of the unique class values, which happen to be: Iris-setosa, Iris-versicolor and Iris-virginica.
2. Next, we assign an integer value to each, such as: 0, 1 and 2.
3. Finally, we replace all occurrences of class string values with their corresponding integer values.

Below is a function to do just that called **str_column_to_int()**. Like the previously introduced **str_column_to_float()** it operates on a single column in the dataset.

```
# Convert string column to integ
def str_column_to_int(dataset, 
	class_values = [row[
	unique = set(class_va
```

We can test this new function in addition to the previous two functions for loading a CSV file and converting columns to floating point values. It also returns the dictionary mapping of class values to integer values, in case any users downstream want to convert predictions back to string values again.

The example below loads the iris dataset then converts the first 3 columns to floats and the final column to integer values.

```
from csv import reader

# Load a CSV file
def load_csv(filename):
```

Running this example produces the output below.

We can see the first row of the dataset before and after the data type conversions. We can also see the dictionary mapping of class values to integers.

```
Loaded data file iris.csv with 15
['5.1', '3.5', '1.4', '0.2', 'Iris-setos
[5.1, 3.5, 1.4, 0.2, 1]
{'Iris-virginica': 0, 'Iris-setosa': 1
```

Reading from .xlsx file

import pandas as pd

df = pd.read_excel(“/home/Loan_Prediction/train.xlsx”, sheetname = “Invoice”)

Reading from .txt file

```
text_file = open("text.txt", "r")
lines = text_file.read()
```

Exercises

1. **Detect and remove empty lines at the top or bottom of the file.**
2. **Detect and handle missing values in a column by replacing the missing value with the mean value of that column**
3. **Given a .txt file, find delimiters such as “|” (pipe) or white space and convert to .csv file.**
4. **For Iris dataset, take any two features as input and show a 2d plot. Repeat this for pima dataset**
5. **For iris dataset, draw various 2d plots X,Y, where X is the feature and Y is the label.**
6. **For glass dataset, input the % of records to be taken for training dataset. Split the data into training and test dataset and store in different files.**
7. **Normalize the glass dataset to consist all the feature values between 0 and 1.**
8. **For pima dataset, input the number of features ‘l’ to be selected. Select ‘l’ features randomly from the dataset alongwith the labelled feature.**