April 1, 2025

Python programming for AI lab

Lab practice 1

Installing python

Unix and Linux Installation

Follow these steps to install Python on Unix/Linux machine.

- Open a Web browser and go to https://www.python.org/downloads
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the Modules/Setup file if you want to customize some options.
- run ./configure script
- make
- make install

This installs Python at the standard location /usr/local/bin and its libraries at /usr/local/lib/pythonXX where XX is the version of Python

Installing on Windows

If you use Windows, it is recommended that you use a SciPy-stack compatible distribution of Python 3. Anaconda is pretty popular and easy to use. You can find the installation instructions at: https://www.continuum.io/downloads.

Anaconda navigator

Launch spyder

If you are on Windows, you should have installed a SciPy-stack compatible version of Python 3.

Loading data

In order to build a learning model, we need data that's representative of the world. Now that we have installed the necessary Python packages, let's see how to use the packages to interact with data. Enter the Python command prompt by typing the following command:

$ python3

Let's import the package containing all the datasets:

```
from sklearn import datasets
```

Let's load the house prices dataset:

```
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
```

Print the data:

```
print(housing.data)
```

```
[[   8.3252       41.           6.98412698 ...    2.55555556
    37.88       -122.23       ]
 [   8.3014       21.           6.23813708 ...    2.10984183
    37.86       -122.22       ]
 [   7.2574       52.           8.28813559 ...    2.80225989
    37.85       -122.24       ]
 ...
 [   1.7          17.           5.20554273 ...    2.3256351
    39.43       -121.22       ]
 [   1.8672       18.           5.32951289 ...    2.12320917
    39.43       -121.32       ]
 [   2.3886       16.           5.25471698 ...    2.61698113
    39.37       -121.24       ]]
```

Preprocessing data

We deal with a lot of raw data in the real world. Machine learning algorithms expect data to be formatted in a certain way before they start the training process. In order to prepare the data for ingestion by machine learning algorithms, we have to preprocess it and convert it into the right format. Let's see how to do it.

Create a new Python file and import the following packages:

```
import numpy as np

from sklearn import preprocessing
```

Let's define some sample data:

```
input_data = np.array([[5.1, -2.9, 3.3],
```

```
[-1.2, 7.8, -6.1],
[3.9, 0.4, 2.1],
[7.3, -9.9, -4.5]])
```

We will be talking about several different preprocessing techniques. Let's start with binarization:

- Binarization
- Mean removal
- Scaling
- Normalization

Let's take a look at each technique, starting with the first.

Binarization

This process is used when we want to convert our numerical values into boolean values.

Let's use an inbuilt method to binarize input data using 2.1 as the threshold value.

Add the following lines to the same Python file:

```
# Binarize data
data_binarized =
preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\nBinarized data:\n", data_binarized)
```

If you run the code, you will see the following output:

```
Binarized data:
 [[ 1. 0. 1.]
 [ 0. 1. 0.]
 [ 1. 0. 0.]
 [ 1. 0. 0.]]
```

As we can see here, all the values above 2.1 become 1. The remaining values become 0

Mean removal

Removing the mean is a common preprocessing technique used in machine learning. It's usually useful to remove the mean from our feature vector, so that each feature is centered on zero. We do this in order to remove bias from the features in our feature vector.

Add the following lines to the same Python file as in the previous section:

```
# Print mean and standard deviation
print("\nBEFORE:")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
```

The preceding line displays the mean and standard deviation of the input data. Let's remove the mean:

```
# Remove mean
data_scaled = preprocessing.scale(input_data)
print("\nAFTER:")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
```

If you run the code, you will see the following printed on your Terminal:

```
BEFORE:
Mean = [ 3.775 -1.15 -1.3 ]
Std deviation = [ 3.12039661 6.36651396 4.0620192 ]
AFTER:
Mean = [ 1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [ 1. 1. 1.]
```

As seen from the values obtained, the mean value is very close to 0 and standard deviation is 1.

Scaling

In our feature vector, the value of each feature can vary between many random values. So it

becomes important to scale those features so that it is a level playing field for the machine learning algorithm to train on. We don't want any feature to be artificially large or small just because of the nature of the measurements.

Add the following line to the same Python file:

```
# Min max scaling
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)
```

If you run the code, you will see the following printed on your Terminal:

```
Min max scaled data:
[[ 0.74117647 0.39548023 1. ]
[ 0. 1. 0. ]
[ 0.6 0.5819209 0.87234043]
[ 1. 0. 0.17021277]]
```

Each row is scaled so that the maximum value is 1 and all the other values are relative to this value.

Normalization

We use the process of normalization to modify the values in the feature vector so that we can measure them on a common scale. In machine learning, we use many different forms of normalization. Some of the most common forms of normalization aim to modify the values so that they sum up to 1. L1 normalization, which refers to Least Absolute Deviations, works by making sure that the sum of absolute values is 1 in each row. L2 normalization, which refers to least squares, works by making sure that the sum of squares is 1.

In general, L1 normalization technique is considered more robust than L2 normalization technique. L1 normalization technique is robust because it is resistant to outliers in the data. A lot of times, data tends to contain outliers and we cannot do anything about it. We want

to use techniques that can safely and effectively ignore them during the calculations. If we are solving a problem where outliers are important, then maybe L2 normalization becomes a better choice.

Add the following lines to the same Python file:

```
# Normalize data
        data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
        data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
        print("\nL1 normalized data:\n", data_normalized_l1)
        print("\nL2 normalized data:\n", data_normalized_l2)
```

If you run the code, you will see the following printed on your Terminal:

```
        L1 normalized data:
        [[ 0.45132743 -0.25663717 0.2920354 ]
        [-0.0794702 0.51655629 -0.40397351]
        [ 0.609375 0.0625 0.328125 ]
        [ 0.33640553 -0.4562212 -0.20737327]]
        L2 normalized data:
        [[ 0.75765788 -0.43082507 0.49024922]
        [-0.12030718 0.78199664 -0.61156148]
        [ 0.87690281 0.08993875 0.47217844]
        [ 0.55734935 -0.75585734 -0.34357152]]
```

The code for this entire section is given in the preprocessing.py file.

Exercise (group lab exercise)

Write error free python programs for the following exercise and submit source code by 4:30 pm today.

Please write the team members names and ID in the source codes under the comment.

1. Suppose you are given the following data:
   data = {'Numbers': [10, 20, 30, 40, 50]}
   write a python code to scale the data in standard form between -1 and 1 and print result.
2. Normalize numerical columns (find mean) in the above question (#1) and print result.

3. Suppose you are given the following data:
   data = {
   'Name': ['John', 'Anna', 'Peter', 'Linda', 'Phil', 'Lucy'],
   'Age': [28, 24, 35, 32, 36, 25],
   'City': ['New York', 'Paris', 'Berlin', 'London', 'Barcelona', 'Rome'],
   'Salary': [5000, 6000, np.nan, 6500, 7000, 5500]
   }
   Write a python code to scale the numerical Data (age and salary) in standard form between -1 and 1 and print result.

4. Normalize numerical columns (find mean) in the above question (#3) and print result.