Python for Data Analytics - Anokha-19

Instructors:
M. Vamsee Krishna Kiran
Archanaa R

Target Audience:
B.Tech 1 & 2 Years

Contents:

Python for Data Analysis

Why Python for Data Analysis

Python has gathered a lot of interest recently as a choice of language for data and statistical analysis. Below are some reasons which go in favour of learning Python:

- Open Source
- Python Has a Healthy, Active and Supportive Community
- Easy to learn
- Python Has Big Data
- Python Has Amazing Libraries

Needless to say, it has some drawbacks too:

- It is an interpreted language rather than compiled language – hence might take up more CPU time.
- Weak in Mobile Computing: Python has made its presence on many desktop and server platforms, but it is seen as a weak language for mobile computing. This is the reason very few mobile applications are built in it like Carbonnelle.

Languages for Data analysis

1. Python
2. Java
3. Matlab
4. R
5. Julia
6. Scala etc.

How to Install Python

1. Download python from https://www.python.org/downloads/ and install.
2. Pip installer will come with python installation. It will be available in Python installation folder/scripts.
3. Navigate to that path in the command prompt and execute the command pip install numpy

4. To install matplotlib execute command pip install matplotlib.
5. To install using a single command: python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose
6. To install on ubuntu sudo apt-get install python-numpy python-scipy python-matplotlib ipython ipython-notebook python-pandas python-sympy python-nose

Installing using Anaconda (Best installation method - single shot)
https://www.anaconda.com/distribution/

Python notebooks

https://notebooks.azure.com/
https://cocalc.com/
https://colab.research.google.com/
https://paiza.cloud/en/jupyter-notebook-online
https://jupyter.org/try

Python Prerequisites before jumping into Data Analytics

1. List
2. Tuple
3. Dictionary
4. Strings
5. Importing libraries
6. Iterations and conditional statements
7. Working with matrices
8. Statistical functions
9. Importing and plotting data

Python Libraries

NUMPY

NumPy is the fundamental package for scientific computing with Python:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays. The implementation is even aiming at huge matrices and arrays, better know under the heading of "big data". Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

Numpy containers

- Lists
- Tuples
- Sets
- Dictionaries

Other Features:

- Arithmetic operations on matrices
- Matrix Library
- Linear Algebra
- Fourier transforms

PANDAS

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

SCIPY

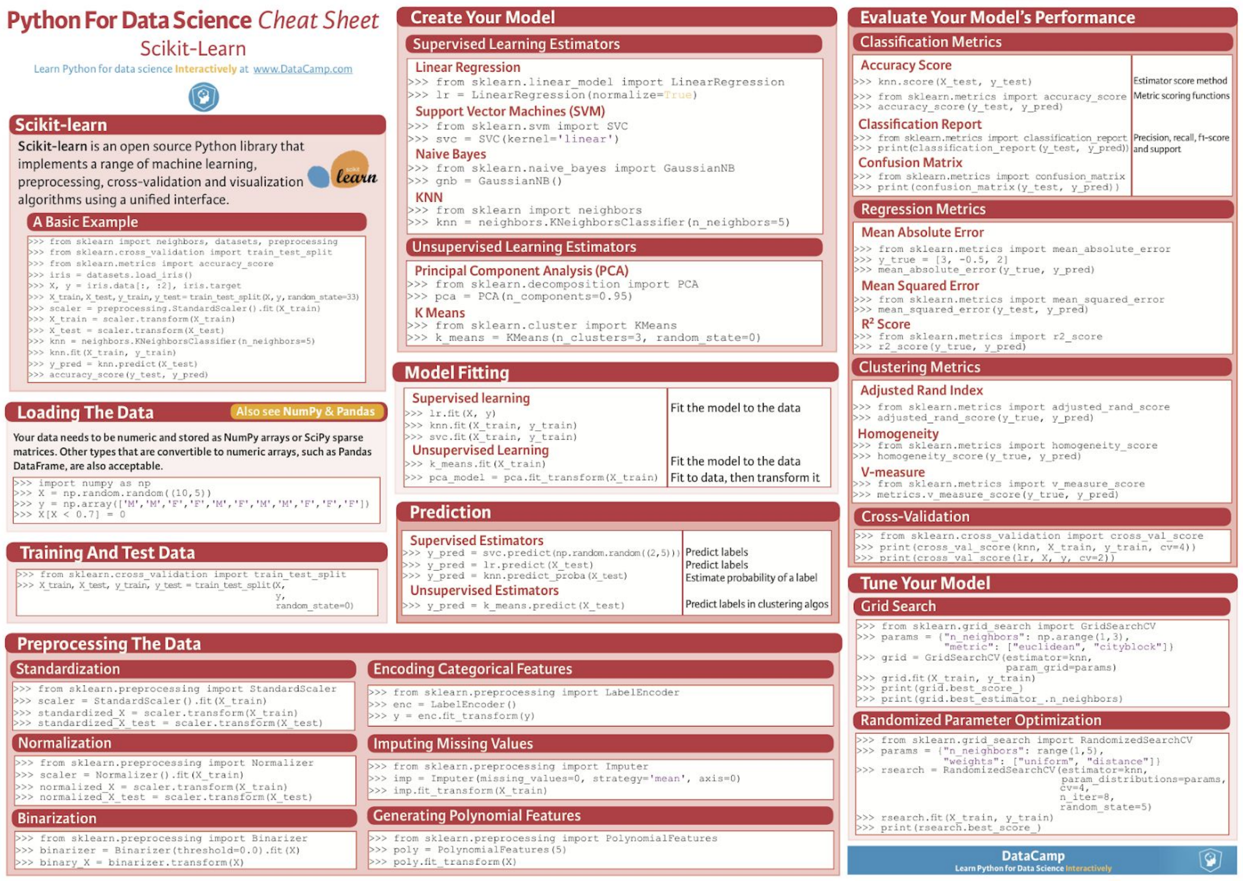
SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy needs Numpy, as it is based on the data structures of Numpy and furthermore its basic creation and manipulation functions. It extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others.
SciPy uses NumPy arrays as the basic data structure, and comes with modules for various commonly used tasks in scientific programming, including **linear** algebra, integration (calculus), ordinary differential equation solving, and signal processing.

MATPLOTLIB

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab.

SCIKIT-LEARN

Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression
```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```
Support Vector Machines (SVM)
```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```
Naive Bayes
```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```
KNN
```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)
```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```
K Means
```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning
```
>>> lr.fit(X, y)                                Fit the model to the data
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```
Unsupervised Learning
```
>>> k_means.fit(X_train)                        Fit the model to the data
>>> pca_model = pca.fit_transform(X_train)      Fit to data, then transform it
```

Prediction

Supervised Estimators
```
>>> y_pred = svc.predict(np.random.random((2,5)))   Predict labels
>>> y_pred = lr.predict(X_test)                     Predict labels
>>> y_pred = knn.predict_proba(X_test)              Estimate probability of a label
```
Unsupervised Estimators
```
>>> y_pred = k_means.predict(X_test)                Predict labels in clustering algos
```

Preprocessing The Data

Standardization
```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization
```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization
```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features
```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values
```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features
```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score
```
>>> knn.score(X_test, y_test)                            Estimator score method
>>> from sklearn.metrics import accuracy_score           Metric scoring functions
>>> accuracy_score(y_test, y_pred)
```
Classification Report
```
>>> from sklearn.metrics import classification_report    Precision, recall, f1-score
>>> print(classification_report(y_test, y_pred))         and support
```
Confusion Matrix
```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error
```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```
Mean Squared Error
```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```
R^2 Score
```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index
```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```
Homogeneity
```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```
V-measure
```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation
```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization
```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

PANDAS

Series

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called *index*. Pandas Series is nothing but a column in an excel sheet.
Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.
pandas.Series(data, index, dtype, copy)

Data Frames

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
This is something like a SQL table with rows and columns or you can imagine like a excel spreadsheet.

Features:

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

A pandas DataFrame can be created using various inputs like −

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

Building Machine learning Models

(Refer to the Jupyter Notebooks for sample implementations)

Conclusion

In this workshop we have discussed the Python Basics and also the importance of different python packages like Numpy, Scipy, Matplotlib and Pandas. We understood that Numpy and Scipy will allow us to do some complex mathematical functions on the data including matrices. Matplotlib is a plotting tool to visualize the data and results. Pandas package will allow us to play with data and most importantly used for preprocessing the data i.e. making the data ready to apply it on a machine learning algorithm. Scikit-learn provides whole set of machine learning models that we can use readily on the preprocessed datasets. All these packages comes along with the installation of Anaconda in a single shot. As python is open-source and supports vast varieties of packages like the above mentioned ones, it is gaining a lot of support in industry as well as academecia.

Hope you have enjoyed and learned something from this session. Thank you…!