ROITRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Python for Data Scientists

CHAPTER 1: TOOLSET OVERVIEW

Chapter Objectives

In this chapter, we will introduce:

- What machine learning is
- The basics of Python
- The toolset to be introduced in this course

Chapter Concepts

Machine Learning

- Machine learning goes beyond simply summarizing data and instead uses complex math to:
  - Make predictions of future values
  - Find hidden patterns
  - Spot outliers or anomalies
- Is more calculation and CPU intensive than traditional reporting
  - Applies mathematical analysis to find how various attributes influence others
  - Requires an understanding of both the raw data and the business in order to explore the deeper meaning of the information
  - Each question asked leads to new questions and involves a lot of trial and error
- Can be used to find actionable business insights

Use Cases

- Predict a future value based on past experience
  - How much of a product should be produced to meet expected demand
  - Probability a disease will respond to treatment based on a mix of patient characteristics
  - What the future price of a commodity is likely to be based on
  - Weather forecast
- Classify a record into a different category
  - Determine if a person should qualify for a credit card
  - Decide if a new card purchase is likely to be legitimate or fraudulent
  - Predict whether a patient is at high or low risk of disease to determine correct treatment
  - Is it likely to rain tomorrow?
- The methods used in machine learning transcend any one field or discipline and can be applied equally in the financial, medical, scientific, entertainment fields, and more

The Science Part

- The Science part of Data Science comes from the fact that the way we approach solving these problems is the same way scientists in general approach their work, the Scientific Method:
  - Ask a question
  - Construct a hypothesis
  - Test the hypothesis by doing experiments and acquiring data
  - Analyze the data and draw a conclusion about whether it supports or rejects the hypothesis
  - Communicate the findings to others
- Models are the key to how scientists accomplish this
  - Representation of an idea, system, object, process, or phenomenon that cannot be directly experienced or observed
  - Simulation of reality that can be used to see how close the model is at describing the real thing we want to understand
  - Always have some level of uncertainty
    - Scientists rarely say always or never, they speak in terms of probabilities

Models

- Even before computers mathematicians were able to identify certain patterns in numbers that are useful in modeling a simulation of reality
- With powerful computers capable of doing huge amounts of calculations these statistical models have been coded into easy to use preprogrammed modules that are more approachable to a wider audience
- The basic ideas behind these models have been coded by different programmers in different ways and on a variety of platforms and languages, but the basic principles remain the same
- To use them, you do not need to understand all the math behind them, but a rudimentary understanding helps
- A big picture understanding of which model can be used to generate a certain result and basic skills in preparing the data is all that is needed to harness the power they offer

Basic Models

- There are a lot of different models and even within a particular type of model, there are subtle variations and implementation differences
- The most fundamental and most widely used models generally fall into one of the following types:
  - Linear Regression
  - Classification
  - Dimension reduction
  - Tree-based methods
  - Clustering
  - Many more

Supervised Models

- Models can be supervised or unsupervised
- Supervised
  - Need to be trained using a dataset with known values we are trying to predict
  - Require a training and testing set to validate how good a job the model does
  - Use a new set of data with unknown values to try to predict what they will be
  - Examples:
    - Regression – good for continuous values like predicting a price
    - Classification – good at identifying the data as a member of a group or category
- Requires you to have well labeled features and known values you are trying to predict

Unsupervised Models

- Unsupervised
  - Can do their analysis on a static set of data without the need to separate it into two sets
- Examples:
  - Cluster Analysis
  - Dimension Reduction
  - Principal Component Analysis
  - Anomaly Detection
  - Association and Recommendation Engines
  - Autoencoders
- You don’t give it a set of known values you are trying to predict, instead you give it raw data and let it identify natural groupings that help shed light on what the data may represent
- Difficult to measure how accurate it is, but it’s helpful in analyzing data to build towards a supervised classification model

Semi-Supervised and Reinforcement

- A hybrid between supervised and unsupervised
- Useful in cases where it’s not easy to label the datasets
  - For example, reading CT scans
- Looks for natural patterns to help label the data then trains itself how to predict based on the patterns it finds
- Generative Adversarial Network invented in 2014
  - Creates two neural networks that compete with each other to see which does a better job
- Low Density Separation is another technique that uses a variation on Support Vector Machines to teach itself about the categories in the dataset
- Reinforcement Learning uses rewards to help an agent accomplish its goal

Chapter Concepts

Why Python for Data Analysis?

- Many choices of tools for data analysis, exploratory computing, and data visualization
  – R
  – MATLAB
  – SAS
  – Etc.
- Python provides the following which makes it a strong choice:
  – Strong library support for analysis
  – General purpose programming language

Python

- Python is a general-purpose programming language that is well suited to processing data and machine learning
- Some of its basic features are:
  - Interpreted
  - Easy to master syntax
  - Wealth of libraries in a variety of fields
  - Interoperates well with C for performance enhancements
- Python comes in versions 2 and 3
  - Differences are largely unimportant for machine learning
  - Ultimately choose the version that has the libraries you need
  - Most common libraries exist for both versions
  - At this point it makes the most sense to start new projects with Python 3 since Python 2 will not be maintained after 2020

Data Types

- Python has all the basic data types one would expect in a language
  - Strings (`str`) can be enclosed in single, double, or triple quotes
  - Integers (`int`) and floating point (`float`) numbers
  - Booleans (`bool`) for True and False
  - Dates (`date`, `time`, `datetime`)

- There are also complex types used to store collections of data
  - List (`list`) uses square brackets `[]`
    - Flexibly stores an ordered collection of almost any type of objects
  - Set (`set`) uses curly braces `{}`
    - Stores a unique set of any type of objects
  - Dictionary (`dict`) uses curly braces `{}` and colon `:`
    - Key and Value pair to store any type of objects
  - Tuple (`tuple`) uses parentheses `()`
    - Like a list but immutable and slightly faster
    - Usually used to encode row-like object that is a collection of columns

Lists

- Can contain any type of data, are mutable and stored in order
- Created by enclosing values in `[]` or by supplying another collection object as a parameter to `list()`
  - `numbers = [1, 2, 3, 4]`
  - `students = [['Abe', 10], ['Betty', 11], ['Charles', 12]]`
- Can get individual elements or slices using `[]`
  - `numbers[0]` → `1`
  - `numbers[1:3]` → `[2, 3]`
  - `numbers[:3]` → `[1, 2, 3]`
  - `numbers[1:]` → `[2, 3, 4]`
  - `numbers[-2]` → `3`
  - `students[1]` → `['Betty',11]`
  - `students[1][0]` → `'Betty'`
- Has methods to append, sort, remove items, etc.

Sets

- Like a list, but only contains unique elements
  - `numbers = {1, 2, 3, 2}` → `{1, 2, 3}`
- Has methods to add and remove items and to find intersection, difference, and union with another set
  - `numbers.add(5)` → `{ 1, 2, 3, 5}`
  - `numbers.add(3)` → `{1, 2, 3, 5}`
  - `numbers.union({1, 3, 4, 6})` → `{1, 2, 3, 4, 5, 6}`
  - `numbers.remove(5)` → `{1, 2, 3}`
  - `numbers.intersection({1, 2, 4})` → `{1, 2}`
  - `numbers.difference({2, 4})` → `{1, 3}`
- Set is often used to make it unique
  - `names = ['Abe','Betty','Carl','Abe']`
  - `names = list(set(names))` → `['Abe','Betty','Carl']`

Tuples

- Tuples are like lists, but immutable and slightly more efficient
- Generally used to represent a row structure or multiple different values
- Has no methods to modify them, in fact the only real method is `count()`
- `numbers = (1, 2, 3, 2)`
- `numbers[0]` → `1`

Dictionaries

- Like in JSON, dictionaries are used to encode key value pairs
- Useful for storing any kind of data
- Instead of being indexed by position, the value is indexed by a key so it is efficient to look up
  - `person = {'firstname':'John', 'lastname':'Smith', 'age':40}`
  - `people = {101:{'firstname':'John', 'lastname':'Smith', 'Age':40}, 201:{'firstname':'Mary', 'lastname':'Jones', 'Age':35}}`
  - `person['firstname']` → `'John'`
  - `people[201]['lastname']` → `'Jones'`
  - `k = person.keys()` → `['firstname', 'lastname', 'age']`
  - `v = person.values()` → `['John', 'Smith', 40]`
  - `x = person.items()` → `[('firstname', 'John'), ('lastname', 'Smith'), ('Age', 40)]`
  - `zip(k,v)` → `{'firstname':'John', 'lastname':'Smith', 'age':40}`

List Comprehensions

- All collection objects can be queried in a SQL-like manner using list comprehensions
- Enclose an expression in `[]` to return list, `()` to return generator, or `{}` to return set or dictionary
  - `[x * 2 for x in [1, 2, 3, 4]]` → `[2, 4, 6, 8]`
  - `[x for x in [1, 2, 3, 4, 4] if x % 2 == 0]` → `[2, 4, 4]`
- Enclose an expression in `()` to return generator
  - `(x * 2 for x in [1, 2, 3, 4])` → `<generator object <genexpr> at 0x7fdd70291480>`
  - `tuple((x * 2 for x in [1, 2, 3, 4]))` → `(2, 4, 6, 8)`
- Enclose an expression in `{}` to return set
  - `{x for x in [1, 2, 3, 4, 4] if x % 2 == 0}` → `{2, 4}`
- Enclose an expression in `{}` to return dictionary
  - `{ k: v['firstname'] for k,v in people.items()}` → `{101: 'John', 201: 'Mary'}`

Functions and Lambdas

- To create a function in Python is simple:

  -
    ```
    def func(x):
        body
        return value
    ```

  -
    ```
    def double(x):
         return x * 2
    ```
- Sometimes a function is so short and just returns a calculation you could abbreviate as a lambda expression
  - `lambda x : x * 2`
  - `double = lambda x : x * 2`
- Some functions can take other functions as a parameters
  - You can either pass a named function or embed a lambda expression in such cases

Collection Functions

- Processing collections is fairly common, so to avoid writing loops there are some helpful functions
  - `data = [10, 2, 30, 4, 5]`
- `map` is a common function used to call a function on each element of a collection. Similar to return value of list comprehension.
  - `map(lambda x : x * 2, data)` → `[20, 4, 60, 8, 10]`
- `filter` can be used to return elements from a collection that meet a condition. Similar to `if` clause of list comprehension.
  - `filter(lambda x : x < 10, data)`→ `[2, 4, 5]`
- `sorted` returns a list in a sorted order
  - `sorted(data)` → `[2, 4, 5, 10, 30]`
  - `sorted(data, reverse=True)` → `[30, 10, 5, 4, 2]`
  - `sorted(data, key=lambda x : (x % 2, x))` → `[2, 4, 10, 30, 5]`
- `reduce` can do an aggregation function on a collection
  - `from functools import reduce`
    `reduce(lambda x, y: x + y, data)` → `51`

Modules

- Modules are Python code that has been bundled up for convenience
  - Could also be written in C
- Contain libraries full of function and classes programmed for a particular functionality
- You import a module to include the code in your scripts
  - `import module`
  - `import module as alias`
  - `from module import function`
  - `from module import function as alias`
- Once imported, you can use the code in the module using the module name and function name or alias
  - `module.function()`
  - `alias.function()`
  - `function()`
  - `alias()`

Chapter Concepts

Essential Python Libraries and Tools

- In this course, we will use the following scientific libraries:
  - NumPy
  - Pandas
  - Matplotlib
  - SciPy
  - Scikit-learn
  - Seaborn
- As well as various UIs:
  - IPython
  - Jupyter Notebook
  - Spyder
  - PyCharm
- And explore other resources like:
  - Google Cloud Platform (GCP)
  - Hadoop
  - Spark

NumPy

- Numerical Python is the foundational package for scientific computing in Python
- NumPy provides:
  - Fast and efficient multidimensional array object `ndarray`
  - Functions for performing computations on arrays
  - Reading/writing array-based data sets to disk
  - Linear algebra operations
  - Fourier transforms
- NumPy arrays are much more efficient for storing and manipulating data than other Python structures
- `import numpy as np`

SciPy

- Collection of packages for scientific computing:
  - `integrate`
    - Numerical integration
  - `linalg`
    - Linear algebra and matrix decomposition
  - `optimize`
    - Function optimizers and root finding algorithms
  - `signal`
    - Signal processing tools
  - `sparse`
    - Sparse matrices and sparse linear system solvers
  - `stats`
    - Standard continuous and discrete probability distributions
- `import scipy as sp`

Pandas

- Provides rich data structures and functions
  - Makes working with structured data fast and easy
- Combines high-performance array features of NumPy with flexible data manipulation features of spreadsheets and databases
- Indexing functionality enables reshaping, slice/dicing of data
- Provides time-series functionality useful for financial data
- `import pandas as pd`

Matplotlib and Seaborn

- Used for producing plots and 2D data visualizations
- Provides interactive environment for plotting and exploring data
- Plots are interactive, enabling zoom and pan of plots
- `import matplotlib as mp`
- `from matplotlib import pyplot as plt`
- Seaborn builds on Matplotlib and adds more plots and visualization
- `import seaborn as sns`

Scikit-learn

- Toolset for data mining and data analysis
- Built on NumPy, SciPy, and Matplotlib
- Support for a number of areas, such as:
  - Classification
  - Regression
  - Clustering
- `import sklearn as sk`

User Interfaces

- IPython is an enhanced Python shell
  - Provides interactive environment for scientific computing
- Useful when interactively working with data using matplotlib
- Also provides:
  - Jupyter Notebook for working with web browser to combine markdown and Python code mixed together
  - GUI console with inline plotting, multiline editing, and syntax highlighting
- PyCharm is a popular, general purpose Integrated Development Environment (IDE)
- Spyder is another popular IDE among data scientists

Other Resources

- Hadoop is a cluster of computers that allows for storage and processing of huge multi-terabyte datasets
- Spark is a processing engine that:
  - Works very fast on a cluster
  - Supports a lot of machine learning algorithms
  - Uses Python in addition to Scala and Java as a programming language
- GCP provides the ability to create cloud-based machines or use existing services to do a lot of the machine learning

Installation and Setup

- Simplest way of obtaining tools is to install Anaconda
  - Provided by Continuum Analytics
- You can also use regular Python and install the libraries you need with pip
  - `pip install numpy scipy pandas matplotlib sklearn seaborn`

Chapter Concepts

Chapter Summary

In this chapter, we have introduced:

- What machine learning is
- The basics of Python
- The toolset to be introduced in this course