SPAM: Simplifying Python for Approaching Machine Learning

Joel A. Rosiene
Department of Computer Science
Eastern Connecticut State University
Windham, CT 06226
rosienej@easternct.edu

Carolyn Pe Rosiene
Department of Computing Sciences
University of Hartford
W. Hartford, CT 06117
rosiene@hartford.edu

***Abstract*—This WIP paper presents an approach to teaching Python in a first course in machine learning for non-majors. The flexibility of Python enables a skilled programmer to be very expressive, but it can be troublesome to the new student. Students who come to Python from an imperative language often approach the language "non-Pythonically", leading to reliance on traditional programming constructs. We present an alternative approach whereby the concept of functors and monads are introduced early to aid in teaching and learning machine learning.**

Keywords—Python, machine learning, functors, monads

I. Introduction

Traditional imperative programing constructs focus on the "how" to implement a function or the structure of data. The building blocks of sequencing, selection and iteration being used to implement the function algorithmically and how to structure the computation. Rather than the traditional approach to programming where we specify the steps to take to implement an algorithm, in the functional approach presented here, the problem solver asks, "What function do I need to solve the problem?" We use set-builder notation to specify a collection of data and functions that are applied to the set. In this approach, while iteration is present, the application of the map does not imply an order, separating the processing of orders required from coincidental orderings. The students are introduced to the concepts of functors and monads early (without introducing the notation and terminology), hopefully, influencing their problem-solving approach and taking advantage of functional features of Python[1].

Short Python brain teasers posed in set-builder notation are used to develop the student skills in set comprehension and function composition in preparation for the machine learning component. It is at this point where Python packages are introduced as a mechanism to construct complex sets of objects (data) and map existent functions across these sets.

The above approach focuses on the formal definition of sets, functions and mapping, and leads naturally to the problem of "learning" functions from examples. One aspect of machine learning is the construction of functions given examples from the domain and specifying the element in the range (the training set) and the composition of the resulting functions. In this approach, we discuss the differences between the estimation of an unknown function with incomplete training data (bias), the fitting of a known function and inaccurate data (modeling and regression) and estimation of a random variable (randomness).

The amount of information on machine learning can be daunting for a student who is new to programming. Approaches which focus on control structures or the data structure requires a degree of mathematical and problem solving sophistication which is not a prerequisite for this course. The course is in our liberal arts curriculum and is designed to be accessible to the general student population. The students will have completed a freshman mathematics requirement, but otherwise there is no assumption that the students will have significant mathematics skills. It is believed that the requirement of mastery of procedural constructs prior to exploring applications of machine learning adds additional complexity to the more function-oriented (functional) approach and is appropriate for only the most motivated students. While Python is not a functional language, it is popular and can be used to explore sets, functional programming and machine learning. The ability to use an online python environment for free (in this case Azure notebooks) removes another barrier from the student.

In the remainder of this WIP paper, we outline the approach, how set builder notation is utilized and relates to Python, and the flow of the course from the purchase of functions to the investigation of automated function construction.

II. Approach

A typical characterization of programming paradigms usually highlights the following problem solving approaches:

- Imperative: "If only I tell the computer to do this and then this."
- Object-Oriented: "If only I had a thing that did that when I told it this."
- Functional: "If only I had a function which gave me that when I gave it this."

It can be confusing for the student to mix the paradigms. We try to work "Pythonically" and since we are doing machine learning, we adhere to the functional approach.

The idea, is to think in terms of functions being applied to elements of sets. As a class we engaged in the "if we had a

function which did this to this data, the problem would be easy." There is a bit of preliminary material we need to introduce, since some of the students have limited exposure to the concepts of sets and functions.

III. Preliminaries

The first step is to use list comprehension to create lists of objects. The student needs to understand how we can apply filters to sets to make new sets with specific properties. The Pythonic form:

[f(x) for x in S if g(x)]

is similar to set builder notation

$$\{f(x): x \in S, g(x)\}$$

which we read as f(x) such that x is in the set S, and g(x) holds. The student is introduced to the concept of mapping functions across a set of so the next step is to introduce the Python map function, f(x),

V=list(map(f:S))

which is simply $f: U \to V$, or f takes the elements of U to V. Here we discuss the laziness, and how map returns something that can be used to produce elements, and how the list actually gets the elements. With this we can apply a purchased function from any vendor (such as Microsoft Azure[2]) to make interesting sets. In Python 3, map alone when applied to an iterable returns an iterable, but for a functor[1], we would like the datatype to be preserved, so the list is applied. We also introduce the concept of a Monoid[1] to flatten lists, sum([[1,2,3],[4,5],[6]],[]) where sum is the function used to combine (accumulate) the lists. The students tend to be confused with the terminology, but the concept of "laziness" is well understood and once explained, Python behaves as expected.

IV. Early Usage, Buying Functions

With a little work, we can purchase functions to translate sets of words, and filter sets of words which meet requirements in another language.

```
# Create a set of Strings.
mySetofStrings = ["Hello","Cat","Dog","Goodbye"]

X = [el for el in mySetofStrings if (len(ToFrench(el))<5)]
print(X)
```

Figure 1. Filter a set of words based on length of translation

The function in Figure 1 is simple but does not account for the masculine verses the feminine forms (in this case only the masculine is used) but illustrates the concept of functions as a service.

```
# -*- coding: utf-8 -*-
import os, requests, uuid, json

def ToFrench(word):
    subscription_key='********************************'
    endpoint='https://api.cognitive.microsofttranslator.com'

    path = '/translate?api-version=3.0'
    params = '&to=fr'
    constructed_url = endpoint + path + params

    headers = {
        'Ocp-Apim-Subscription-Key': subscription_key,
        'Content-type': 'application/json',
        'X-ClientTraceId': str(uuid.uuid4())
    }

    myDictionary = {'text':word}
    body = [myDictionary]

    request = requests.post(constructed_url, headers=headers, json=body)
    response = request.json()


    return(response[0]["translations"][0]["text"])
```

Figure 2. Purchased translator function

Similarly, it is also simple to analyze the sentiment of a sentence or a tweet by purchasing a function and using Tweepy[3] (a Python library for Twitter API) to get some public tweets from a feed. A sample purchased function is shown in Figure 2. The Tweepy code and the Azure code are the standard reference code from the vendors. The generation of a plot using these purchased functions is given in Figure 3. The challenging part of purchasing the functions is getting the account on Azure, and introducing the concept of authorization keys and service endpoints.

```
myListOfScores=[]

AzureClient = authenticate_client()
TweepyApi = tweepy.API(auth)

public_tweets = api.home_timeline()

for tweet in public_tweets:
    document = [tweet]
    score = sentiment_analysis_example(AzureClient,document)
    myListOfScores.append(score[0])

plt.plot(myListOfScores)
plt.ylabel('Tweet Sentiment')
plt.xlabel('Tweet Number')
plt.show()
```

Figure 3. An application of purchased functions to generate a plot

V. Doing Stupid Things Quickly Appears Intelligent

To transition from purchasing functions to creating functions which seem intelligent, the student is introduced to trial and error and random approaches. Backtracking and Las Vegas algorithms[4] are two problem solving approaches which seem "intelligent", if the computer is fast enough.

A. BackTracking

The backtracking algorithm is a simple problem-solving method. For Sudoku puzzles, the main solver routine is:

```python
def solveIt(Puzzle):
    (freeSpotRow,freeSpotCol) = findEmpty(Puzzle)

    if freeSpotRow==None:
        return Puzzle
    else:
        for el in findPossibles(Puzzle,freeSpotRow,freeSpotCol):
            Puzzle[freeSpotRow][freeSpotCol]=el
            if solveIt(Puzzle):
                return(Puzzle)
        Puzzle[freeSpotRow][freeSpotCol]=0
        return None
```

Figure 4. Simple backtracking function

The functions used are *findEmpty* given a puzzle, find the possible values given the puzzle and the location to check called *findPossibles*, and the routine *solveIt*. Once more, all of the students seem to grasp the concept that the function we are writing is the one that will use after making a guess.

B. Las Vegas/Randomized Algorithms

Las Vegas algorithms are ones which try random actions until the problem is solved. If solvable, these will almost surely provide the solution but may take an arbitrarily long time. To demonstrate this, we consider the sliding 8-puzzle in Figure 5.

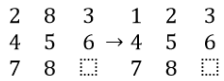

Figure 5. Sliding 8-puzzle

The sliding 8-puzzle was used to introduce the concept of a Las Vegas algorithm to randomly apply actions to an empty space until it is solved.

The goal of the puzzle is to put the tiles in in increasing order. Randomly moving the space will result in very long sequence of actions (sequences in excess of 100,000 are common) when it is known that only 31 actions are need. The student is then tasked to improve the performance by eliminating short cycles. The improvement changes the selection of the next random action based on the prior action introducing the concept of conditional probabilities. Since the performance of the student's solution change each time it runs, this introduces the concept of a trial, average performance and worse case performance, for both a single puzzle and the class of problem. This is an important concept to keep in mind as we move to "learning" functions.

VI. Learning Functions from Examples

After exploring the construction of functions in many forms, the student then explores SciKit Learn[5] to learn functions. The first approach to "learning" functions is the problem of class assignment,

$$y = f(x), x \in X.$$

X is the set of all possible vectors of features. The goal is to "learn the function" given examples, or the training set,

$$\{\{y_1, x_1\}, \{y_2, x_2\}, \cdots \{y_n, x_n\}\}$$

The two approaches to learning a simple classification function we have used this time are K-Nearest-Neighbors (KNN) and Support-Vector-Machines (SVM)[6]. KNN is used since it makes few assumptions, and simply keeps around the examples of the desired mapping. SVM is used to show how parameterization can speed up the process, but requires assumptions about the shapes of the classes in feature space. This is somewhat "meta" as we are using functions to produce the function we want.

A. KNN

KNN simply assigns a given new sample x to the class which has neighbors which are in some sense closest to the given point. This is a robust approach, with the disadvantage that the training set is the model and it is costly to algorithmically evaluate the function for each given feature vector.

B. SVM

To reduce the time to evaluate the function, and reduce the number of parameters in the model, SVM calculates and stores the decision boundary curves which are used to classify new feature vectors.

SciKit Learn provides a simple interface to both of these functions and the student can easily compare the performance of both approaches to simulated data and the standard datasets by changing a single line of code

VII. Conclusion

An overview of the approach to introduce machine intelligence concepts to a non-CS major is provided. We play the game, "If only I had a function" and explore the concept of purchasing the function (Function as a service) and then graduate to developing functions which solve problems in a naive manner, but faster than the human. Finally, the student is introduced to the concept of having the machine learn the function given a number of samples of the desired response.

Our current experience is producing mixed results, with a class of only 9 students, two of which have significant Python experience having no issues, and the non-majors struggling with some of the concepts, but having success with the mechanics of using the functions from Microsoft Azure and the Jupyter Notebooks. Since the paper was submitted, the course had to be transitioned online, which was eased with Azure Notebooks. We intend to further refine the approach with more puzzles to illustrate monads and functors.

References

[1] Python Software Foundation, Python, www.python.org.

[2] Microsoft Corporation, Azure Notebooks, notebooks.azure.com.

[3] Tweepy, An easy-to-use Python library for accessing the Twitter API, www.tweepy.org.

[4] Rosiene, Joel A. and Rosiene, Carolyn Pe, "Design Patterns for Sorting Algorithms," ASEE/IEEE Frontiers in Education Conference, October 16-19, 2019 Cincinnati, OH.

[5] Pedregosa, F., Varoquaux G., Gramfort A., Michel V., Thirion B., "Scikit-learn: Machine Learning in Python," JMLR 12, pp. 2825-2830, 2011.

[6] "Nearest Neighbors Classification", scikit-learn.org/stable/modules/neighbors.html#classification.