UNIVERSITY OF WATERLOO
Faculty of Mathematics

USING PYTHON FOR MACHINE LEARNING PIPELINES AND WHEN WE SHOULD STOP

Under the supervision of Dr. Rumi Chunara at New York University
Waterloo, Ontario

Prepared by
Jiaxin Liu
3A Scientific Computation
ID 20468322
August 24, 2015

45 Rothbury Rd
Richmond Hill, Ontario
L4S0B3

August 24, 2015

Dr. Rumi Chunara, Assistant Professor
NYU Global Institute of Public Health
41 E 11th St
New York, NY 10003

Dear Dr. Chunara:

As we agreed, I have prepared the enclosed report "Using Python for Machine Learning Pipelines and when to stop." for my 3A work report and for ChunaraLab. This report, second of four reports that the school requires I complete as a part of my coop degree requirements, has not received academic credit.

This summer I was a research intern at ChunaraLab. My research project was to build a machine learning pipeline to identify social medias posts that where alcohol related and to try to find instances of binge drinking and other problem drinking symptoms at a population level. This report is a analysis of why python should be used to build the tools I used and when and why we might have to switch to a different toolset in the future.

The Faculty of Mathematics requests that you evaluate this report for command of topic and technical content/analysis. Following your assessment, the report, together with you evaluation will be submitted to the Math Undergrad Office for evaluation on campus by qualified work report markers. The combined marks determine whether the report will receive credit and whether it will be considered for an award.

Thank you for your assistance in preparing this report.

Sincerely,

Jiaxin Liu

Table of Contents

List of Tables and Figures iv

Summary v

1.0 Introduction 1

2.0 Exploration 3

- 2.1 Interactive Python Notebooks 3
- 2.2 Data Visualization 4

3.0 Prototyping 6

- 3.1 Preprocessing 6
- 3.2 Pipelining 6
  - 3.2.1 Classifiers 7

4.0 Model Evaluation 8

- 4.1 Model Inspection 10

5.0 Beyond Python 10

- 5.1 Vowpal Wabbit 11
- 5.2 Spark MLLib 12

6.0 Conclusions 12

References 14

List of Tables and Figures

Figure 1 seaborn.py example gallery 4

Figure 2 screen shot of a research notebook 5

Figure 3 example pipeline code 7

Figure 4 scikit-learn cheatsheet 8

Figure 5 various model diagnostic plots 9

Summary

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

1.0 Introduction

Python is as multi-paradigm language that allowed for object oriented, structural and functional programming. This dynamic language, along with it's large community and timing made it a great choice for both academics who wanted something easy to use and for engineers who wanted to rapidly develop products. Another factor is NumPy (Numeric Python) and its extension, SciPy (Scientific Python), which are a collection of open source Python projects that support large-scale scientific data processing. Once these tools went mainstream, data analysis with python skyrocketed to become what it is today.

This means that there are a lot of packages that exist in the Python ecosystem that solve a lot of existing problems. Take `https://github.com/vinta/awesome-python` as an example, it is a collection of packages for all occasions. Whatever you want to do, it can be done in python. Its incredibly important in research to not build everything from scratch, and to also be able to pass on work between parties. Python, as easy as it is to get started, is often a common language between developers.

Machine learning, a particularly popular form of data analysis became extremely accessible with the advent of a few tools tools in the Python ecosystem:

- python notebooks: a way to explore data and code interactively
- pandas: a fast and efficient data manipulation tool
- scikit-learn: an extensive machine learning package that contains pipeline abstractions and ready to use algorithms

These three tools allows us to rapidly develop end to end machine learning platforms with extremely low technical debt while maintain high maintainability and readability.

From how easy it is to explore the data and generate reproducible research to the fast and efficient data manipulation and consistent and easy to understand machine learning pipeline apis provided my scikit learn, python is a great tool to get starting in building a machine learning model. However it is without its draw backs. Once that data gets larger and the computing must be out of core, we must either move on to different tools in the python ecosystem or move on to different languages that provide more power at the cost of simplicity.

2.0 Exploration

Since python is very light weight and dynamic, it is a highly productive language and as a result, very effective in getting quick results.

2.1 Interactive Python Notebooks

Notebooks provide a "workspace" for developing code and is great for supporting reproducible research by having one to one input output mappings. This way, as long as someone can open the notebook on their machine and run the cells, results can be generated on the fly.

It's writing cells also support Markdown and LaTeX for when we want to convey our ideas and equations right from the notebook. This allows us to document our process better than comments in a script ever could. In fact, notebooks have now become a base tolo for teaching tools in the python community, an example of this would be "Mining the Social Web" from O'Reilly, where notebooks provide an interactive way to follow along with the examples.

By developing with notebooks, we can provide step by step instructions on how execute certain programs or better explain design principals to other members. Lastly, the notebooks set up a homogenous environment for collaborators to share ideas.

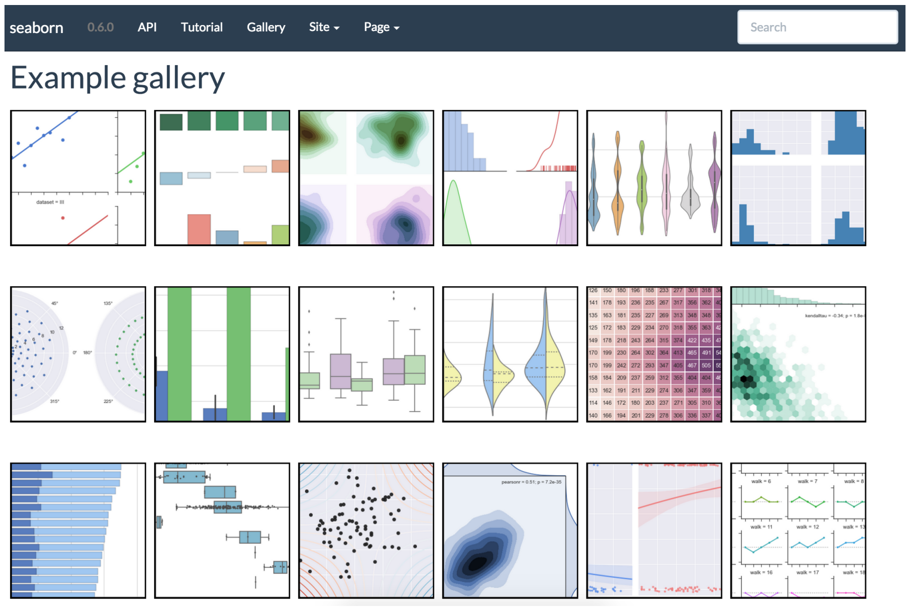
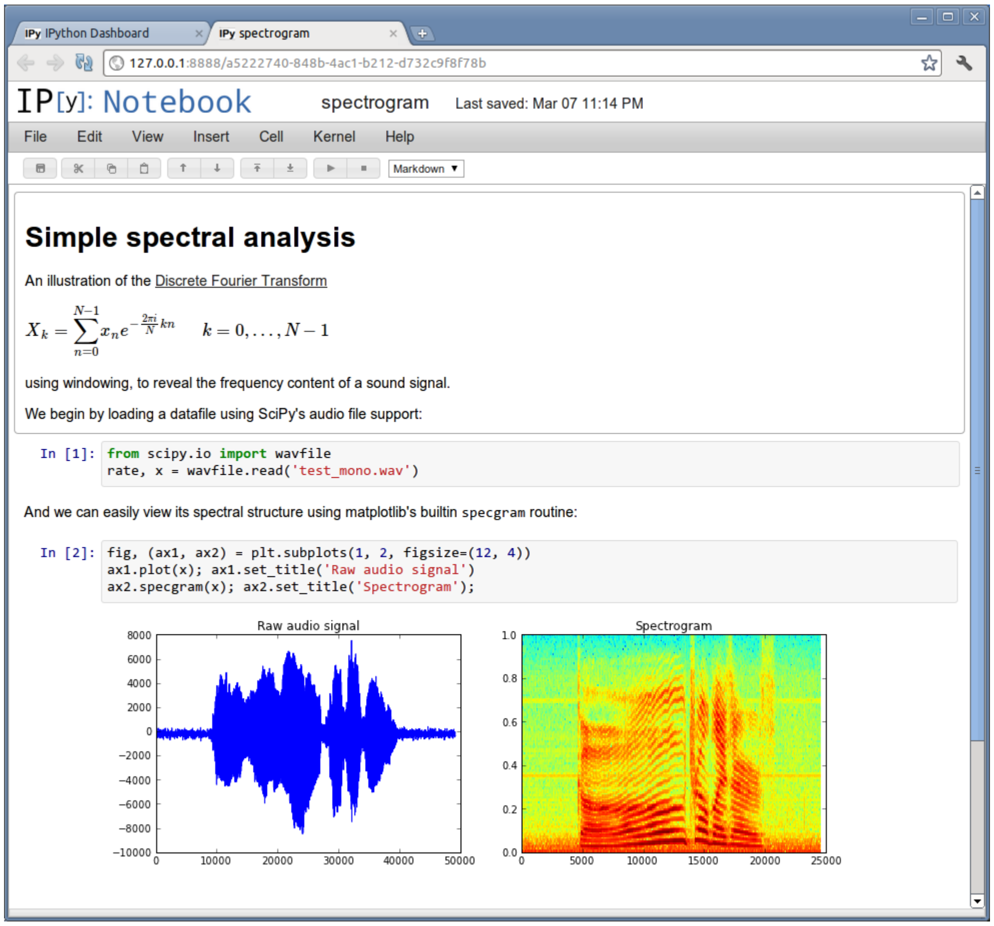


Figure 1: seaborn.py example gallery

2.2 Data Visualization

Before jumping into building models its important to explore the data thats in front of you to get a better understanding of the relationships. Exploratory data analysis was promoted by John Tukey to encourage analysts to format new hypothesis that could lead to new data collection or experiments.

Luckily, python has some really great data visualization packages, one of them being seaborn. Seaborn is a library that provides a high level interface for drawing statistical graphics. Together with python notebook,s we have plenty of transformations and visualizations to play with right out of the box.



Simple spectral analysis

An illustration of the Discrete Fourier Transform

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \dots, N-1$$

using windowing, to reveal the frequency content of a sound signal.

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

```
In [2]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

Figure 2: screen shot of a research notebook

3.0 Prototyping

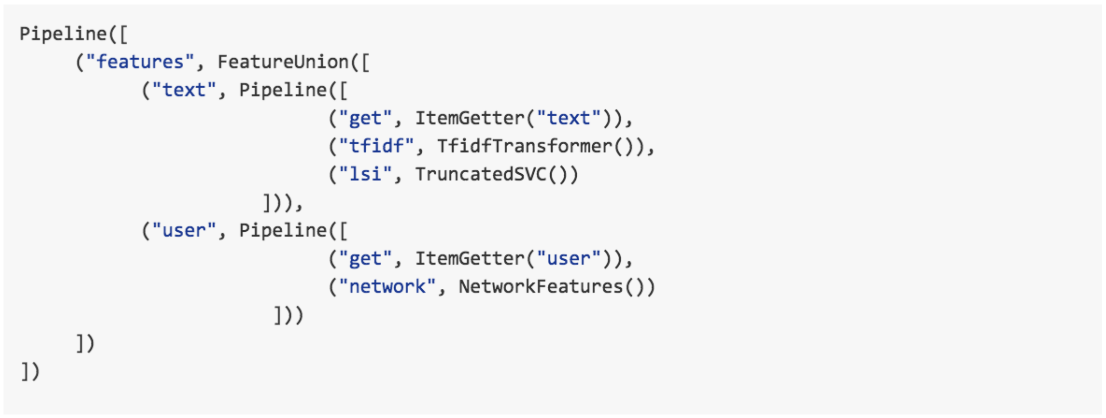
3.1 Preprocessing

Similar to R, Python's Pandas package provides a DataFrame for doing data manipulation. As mention before due to how dynamic python is, its much more productive to do processing in python than other languages like Java or C. Moreover, since its so much faster for the developer, time saved could be spent finding bottlenecks and optimized in C, however if you write your code correctly with pandas, all of it is run in C to begin with.

3.2 Pipelining

We must also remember that real data does not come in a perfect computer readable format and that most machine learning algorithms requires data to come in the form of a matrix in R^n.

Consider the task of text classification. We first need to normalize the text, then count words and normalize using inverse document frequency, then factorize the matrix to a lower dimension to find the topics and then turned into a sparse matrix for the algorithm to ingest. By using the pipelines api, it becomes extremely easy to code modular pieces of work and compose/recompose them like the figure 3.



```
Pipeline([
    ("features", FeatureUnion([
         ("text", Pipeline([
                         ("get", ItemGetter("text")),
                         ("tfidf", TfidfTransformer()),
                         ("lsi", TruncatedSVC())
                    ])),
         ("user", Pipeline([
                         ("get", ItemGetter("user")),
                         ("network", NetworkFeatures())
                     ]))
    ])
])
```

Figure 3: example pipeline code

3.2.1 Classifiers

Unlike other packages, scikit learn has a huge collection of reimplemented state of the art algorithms that rely on well implemented open source tools like LibSVM and LibLinear. Along with plenty of algorithms, injecting them into the pipeline is as simple as adding a classifier at the end of the pipeline, this makes trying different algorithms and pipelines very effective.

By taking advantage of the pipeline api and python meta-programming, the secondary problem of hyper parameter optimization is solved with a builtin implementation of grid search using cross validation. While frameworks require custom grid search code to be developed along with complex parameter injections, scikit alleviates a lot of technical debt by letting us write less code.

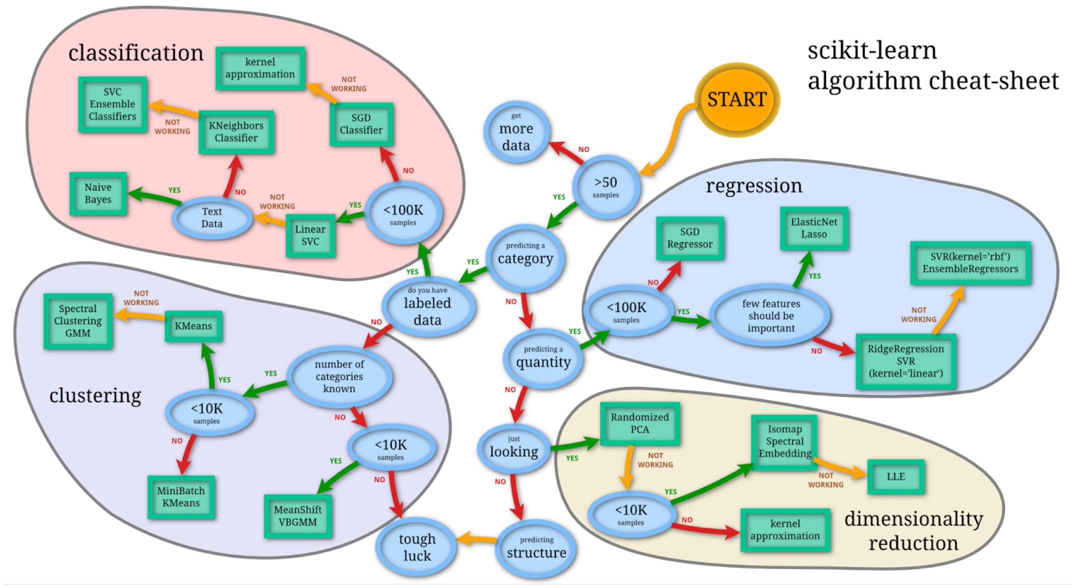
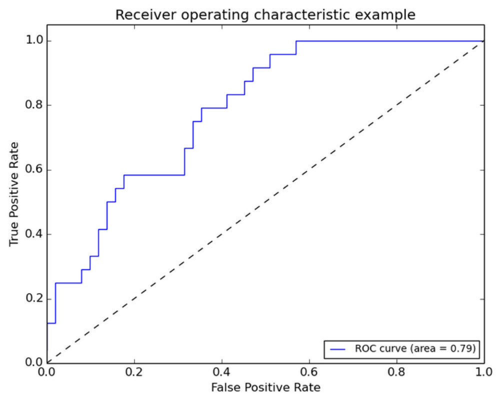


Figure 4: scikit-learn cheatsheet

4.0 Model Evaluation

Once we've build various models, we'll want to find the best ones and scikit also allows us to do this simply as well. scikit.metrics is a helpful module that contains many ways to measure performance. For example, consider the problem of detecting a rare cancer. If the cancer shows up on average once in every hundred patients, then saying that no one has cancer would yield a 99% accuracy. Although the algorithm was simply "return False", it essentially fails on the accuracy metric. Instead we may use f1-score to consider the false positive and false negatives as a way to measure success.



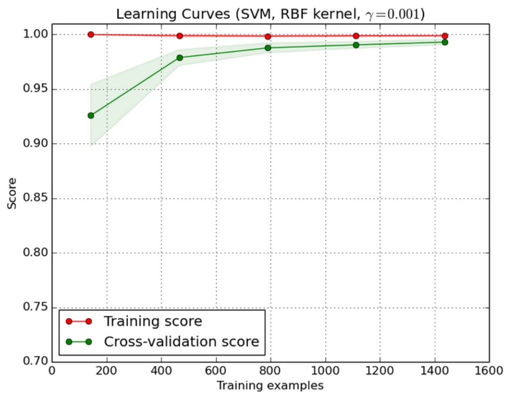


Figure 5: various model diagnostic plots

4.1 Model Inspection

One downfall, present in many libraries is the lack of model inspection. After the function that maps the inputs to the results is learned, it is very hard to understand the various feature importances or do hypothesis tests on the features significances. This question in itself sets up a divide between the rigour of statistical methods rather than black box machine learning methods where we are simply minimizing a cost function. However more and more packages are coming out for inspecting the models, particularly in tree based methods. If statistical significance matters more performance, then statsmodels is a package that could provide these features.

5.0 Beyond Python

Python allows use to do three things very well with minimal boilerplate code.

- exploration
- pipelines
- evaluation

In this section we will discuss some downfalls and how to avoid them along with some solutions that might exist outside of python.

Python notebooks have recently inspired many other languages to follow in it's foot steps and recently announced project Jupyter which serves as a language agnostic notebook that currently supports many languages including, R, Spark, and Lua, all fantastic tools for data analysis or machine learning. Scikit learn's pipeline api design has been so successful that larger projects like Spark are designing it's apis to parallel scikit's own. This is to say that starting out with python's ecosystem allows us to learn transferable design patterns.

The main drawback however is that due to pythons primary limitations (these can be avoided with a lot of work) scaling to many cores or many nodes is difficult and training on large datasets is almost impossible after 1million rows. Once the data grows, what we can do with a end to end python pipeline becomes a bit limited. In this section I will introduce two tools that we may use to alleviate some of the pains of long compute hours.

5.1 Vowpal Wabbit

"Vowpal Wabbit is a machine learning system which pushes the frontier of machine learning with techniques such as online, hashing, allreduce, reductions, learning2search, active, and interactive learning."

Due to these efficient tricks, Vowpal Wabbit (VW) is the king of large data single machine problems. However its api is far from simple and it lacks any builtin pipeline or grid search as it is primarily a set of CLI tools. At the cost of developer time we gain computing performance. What is recommended it to explore a subset of the data and prototype models in the python be more moving to VW, depending on how you implement the pipelines, the preprocessing pipelines can be recycled as long as you write a pipe element that converts the data to the VW format.

5.2 Spark MLLib

"Apache Spark? is a fast and general engine for large-scale data processing."

If the data is so large that it must be in HDFS then it is recommended we move to Spark if we have access to clusters of nodes that we can parallelize over. Spark has been a big contender for bid data processing and development of MLLib, it's machine learning library has been very strong. Recently it has included DataFrames and Pipelines into the framework and are working on implementing gridsearch. Spark is probably the closest to a end to end data processing pipeline like Python is, however it may often be overkill.

6.0 Conclusions

The Python ecosystem is great for fast data analysis and machine learning. It has a great community with tonnes of support and packages. Minimizing the amount of code on needs to write before getting results.

Notebooks allow us to produce informative reports of our analysis and thought process in a reproducible way. Scikit also provides a lot of efficiencies. By using their pipeline, feature unions, and massive library of ready to go algorithms, we alleviate a lot of technical debt on the people who have to use our code in the future.

The biggest bottleneck is that scikit is bound my memory and a single for, which makes massive parallelization difficult. If we want to avoid writing too much complex code, we might want to consider different tools. Two examples of such tools are vowpal wabbit which is very complex but powerful on a single machine and spark which requires a hadoop setup and many nodes to be effective. The problems with these however is that they are great for large production systems that take in millions of rows of data. This makes python idea for building fast prototypes. We can explore a space and build models extremely productively for proof of concept analysis. Once the data is too bed then we can recycle some of our analysis code for vowpal wabbit or migrate to spark.

References

1 Champion, R., Paci, T. & Vardon, J. (2012). PD 2: Critical Reflection and Report Writing. Retrieved 1 March, 2012 from https://learn.uwaterloo.ca/d2l/le/content/80224/viewContent/605550/View