A Comparative Study of Python and R for Data Science

Kruttika Jain

kjain4@ucsc.edu

1. Introduction

Data Science is an interdisciplinary field that uses Mathematics, Statistics, Computer Science and Algorithms to gain insights from Data. The data itself may be of any type; text, image, time-series, signals, etc. Data Science is related to Machine Learning, Data Mining and Deep Learning and sometimes these terms are used interchangeably . There are many Programming Languages used in the field of Data Science such as Python, R, Scala, Julia, SQL, SAS, MATLAB, etc. However, over the last several years the two most popular programming languages in this domain remain Python and R. Thus, this project aims to perform a comprehensive comparison of Python and R for Data Science. In this project, two machine learning algorithms i.e Logistic Regression(hand coded) and Naive Bayes and a Deep Neural Network are built and compared in both Python and R. Comparison over all the stages of Data analysis is performed, i.e from data loading and cleaning until prediction and visualization. Time and Space analysis is also performed for the code developed in the two languages. Other parameters such as code readability, writability, portability, flexibility, etc are are also checked.

2. Motivation

I have been programming in Python for a while and despite having worked on multiple Data Science projects, I never had the opportunity to learn R. While Python seems to be the most popular language of choice for assignments and courses at university, R is also high in demand at the industry and research level specially in areas with high scientific computing requirements. I wanted to take this opportunity to broaden my horizons and learn a new programming language in the field of my interest.

3. The Two Programming Languages

3.1. Python

Python is a general purpose, object oriented programming language created by Guido van Rossum, and first released in 1991. It is a highly readable and writable language. While it is primarily object oriented and structured, it also offers some support for Functional and Meta-programming. Although there are many Syntax and Semantic [7] specifications unique to Python, some major highlights are:

- Python uses indentation rather than curly braces/ brackets for code blocks.
- It is dynamically typed, i.e unlike other traditional languages at a given time a name is bound to a certain data type but after reassignment it can be changed. For example successive assignments such as a=10 followed by b=10 followed by c=10 will end up in a maximum of three variable allocations and one numeric allocation.
- Lambda expressions are used to implement Anonymous functions
- Python supports a term called list comprehension, array slicing and indexing, various kinds of string literals and distinguishes between lists and tuples
- Python methods have an explicit 'self' parameter to access data as compared to the implicit 'self' of most languages
- It uses duck typing, which essentially means that variables are untyped but objects are typed. Despite being dynamically typed, it is still strongly typed and thus disallows code that is not well-defined.
- It supports many standard libraries for various functionalities such as Web development, Networking , Image Processing, App development, etc.
- It also supports the popular Data Science and Machine Learning libraries such as SciPy, Scikit-learn, Matplotlib, etc.

3.2. R

R is a language and environment for statistical analysis, scientific computing and graphics[8]. It is an implementation of the S language created by Ross Ihaka and Robert Gentleman that was first released in 1995 with a stable version released in 2000. Some major features of this programming language are:

- It supports high level statistical features such as classical statistical testing, non linear modeling , clustering, time-series analysis, etc.
- Variables are dynamically typed and lexically scoped
- R supports Object Oriented Programming for some functions as well as Procedural programming
- Functions are very easily created and variables in the function remain local to the function. These functions can return any data types.
- Most of R's specialized capabilities are extended through the many packages that have been developed by users and maintained at CRAN
- R also has its own Latex-like documentation format

4. Machine Learning with Python and R

There are many supervised and unsupervised Machine Learning[5] algorithms prevalent that are used for tasks such as classification, clustering, structured prediction, anomaly detection, dimensionality reduction, etc. Based on the known data that is provided, i.e the 'training' data, these Machine Learning algorithms build a Mathematical model for classification, regression etc. that is done on unseen data called the 'test' set. There are many popular high performing algorithms such as Naive Bayes, Linear Regression, Logistic Regression, Decision Trees, etc. All of these algorithms can be implemented in Python and R through packages, although some data cleaning and hyper parameter tuning is often required. In this project I first implemented the Logistic Regression algorithm from scratch

4.1. Dataset

The Dataset used for the Machine Learning algorithms was the Pima Indians Diabetes Onset Prediction data[1] . This is available at UC Irvine's Machine Learning Repository. The dataset consists of 9 column predictor variables and one Outcome. The predictor variables are diagnostic indicators such as BMI, insulin level, age, etc. Thus this is a simple classification problem with a relatively small dataset. On this dataset a hand coded logistic Regression algorithm was implemented and also compared to the in built library. The Naive Bayes algorithm was also fit on the data and compared.

4.2. Logistic Regression from scratch

At the basic level Logistic regression[2] uses the log function and then models dependent variables that are binary in nature. It has further extensions to Multinomial logistic Regression, Ordinal Logistic Regression, etc. It is commonly used for classification problems. In Logistic regression, essentially the weighted sum of inputs are taken (similar to Linear Regression) and then passed through an activation function that would map any value to a value between 0 and 1. This activation function is the sigmoid function. The sigmoid function is exactly 0.5 at x=0 in the sigmoid graph, thus this threshold can be used to determine the class, i.e classification as 0 or 1.

4.2.1. Data Loading

To build the logistic regression algorithm from scratch in both Python and R, first the data (which is a csv file) was loaded. Although initially Python csv loading was hand coded as well, later it was read into a Pandas Dataframe because the csv in R was also read into R's Data.frame().

4.2.2. Hypothesis and Cost Function

The sigmoid function is given by:

$$\sigma(y) = \frac{1}{1 + e^{-\theta y}} \tag{1}$$

And the hypothesis of linear regression is given by:

$$h(x) = \theta^T x \tag{2}$$

Thus in simple terms, the hypothesis of logistic regression is given by:

$$h(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{3}$$

The Cost function is also programmed such that it penalizes the model whenever it makes an incorrect prediction. The cost function is minimized using Gradient Descent.

4.2.3. Other Helper functions

Other functions such as sigmoid function , total input and probability calculation are used to implement this Logistic Regression algorithm.

4.2.4. Data training

The Data is divided into X and Y based on features and labels. An initial theta value is also defined which is the initial set of weights with all 0. A model fit function is defined that takes X, Y and theta as inputs to fit the data. It used the $fmin_tnc$ function to optimize the weights.

4.2.5. Prediction and Accuracy

The prediction function uses the parameters obtained from the model fit function as theta and returns the probabilities on the test set. The accuracy function uses a threshold of 0.5 and the fraction of correctly predicted labels to output the accuracy. The accuracy obtained by hand coded Logistic Regression in both Python and R is comparable to the in built algorithm as seen in Table 1.

4.3. Naive Bayes

The Naive Bayes [3]is a probabilistic classifier that uses the Bayes Algorithm for classification assuming conditional independence. To further extend this comparative study, this popular baseline Machine Learning algorithm was implemented to compare results and analysis of the two Languages in the default setting, i.e without any hyper parameter tuning.

ML algorithm	**Python**	**R**
LR hand coded	0.7712	0.7712
LR function	0.7874	0.78
Naive Bayes	0.7244	0.752

Table 1: Accuracies

4.4. Time and Memory Analysis

To evaluate the time to fit the model in Python , the 'time' module was imported and used. The code returned the time of execution in seconds. To evaluate the time to fit the model in R, the 'proc.time()' function was used which is an in-built function in R. This function returns three values- user time, CPU time and total elapsed time.

ML algorithm	**Python**	**R**
LR hand coded	0.031	0.179
LR function	0.013	0.139
Naive Bayes	0.121	0.296

Table 2: Time to fit data

To perform memory analysis, the 'memory-profiler' [4]module from PyPI was used which monitors memory consumption for the entire script as well as line-by-line analysis. For memory analysis, the R tool 'Profvis'[7] was used which is an interactive interface for profiling R code. Figure 1 and 2 show the memory consumption in executing the Logistic Regression from scratch in Python and R respectively. Since R used about 6.4 MB while Python consumed about 120 MB, R is more space efficient.

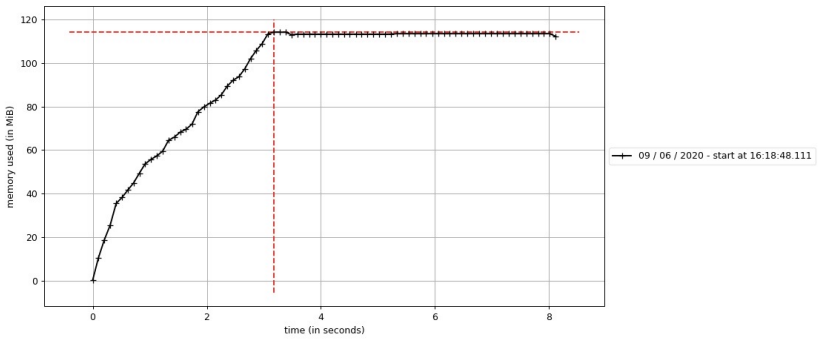
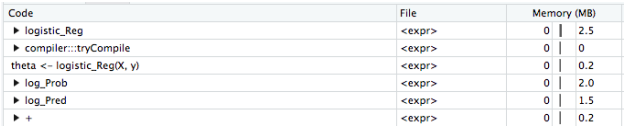


Figure 1: Python: Memory consumption for hand coded LR script

5. Deep Learning with Python and R

Deep Learning is a subset of Machine Learning and also part of Data Science. This field was initially inspired from the neurons in a human brain.



Code	File	Memory (MB)	
▶ logistic_Reg	<expr>	0	2.5
▶ compiler:::tryCompile	<expr>	0	0
theta <- logistic_Reg(X, y)	<expr>	0	0.2
▶ log_Prob	<expr>	0	2.0
▶ log_Pred	<expr>	0	1.5
▶ +	<expr>	0	0.2

Figure 2: R: Memory consumption for hand coded LR script

Deep Learning is commonly used when the dataset or predictors are large in number and the data itself is unstructured, correlated and complex. Deep Learning methods are commonly different types of Artificial Neural Networks with the learning being supervised, semi supervised or unsupervised. They can have various architectures such as fully connected Deep Neural Networks, Recurrent Neural Networks, Convolutional Neural Networks ,etc. and have applications in Natural Language Processing, Medical Image classification, Speech Recognition, etc. Thus it is an important metric for evaluation of the two languages not only because of it's power but also because of the resources needed to train these Networks.

5.1. Dataset

Given the need of a larger dataset, for the case of Deep Learning the 'Titanic' dataset[10] developed by kaggle and encyclopedia-titanica has been used. The data is in the form of a csv and consists of 25 columns. It consists of features such as Age, Sex, Ticket, etc. and the Outcome is Survival, where 1 means the passenger survived and 0 means that the passenger did not.

5.2. Data Cleaning

Before training the model many basic Data cleaning and pre-processing tasks had to be performed. Unrelated features such as Name, Cabin, PassengerId, etc were dropped. NaN values were handled. For example, the feature Age had many NaN values, in Python, this was handled using Pandas's 'fillna' function, whereas in R this was done using the R's 'preProcess' function and impute method.

5.3. Deep Neural Network

TensorFlow and PyTorch are two of the most popular Deep learning frameworks in Python. R has many packages that support Deep Learning such as 'deepnet', 'neuralnet', 'h2o', 'tensorflow', etc. To ensure similarity in development, the Deep Neural Network trained in both languages was done

with TensorFlow using the Keras API. The Network consists of input, output layer and two Dense hidden layers with Relu and Sigmoid activations. The model is compiled using' binary cross entropy', 'RMS Prop' and 'Accuracy' as loss, optimizer and evaluation metrics. The data is divided into batches of 10 and the number of epochs are increased from 10 to 100. The training and validation accuracy obtained in R 0.8635 and 0.8640. In the case of Python, the values obtained are 0.7942 and 0.8202. Since the difference is not significant and can be attributed to different test/ validation split, we consider these accuracies comparable.

5.4. Data Visualization

Based on the training for 100 epochs, data visualization was done with standard notations. While Python required importing 'matplotlib' (the visualization library), R only required the code 'plot(history)'. An interesting feature of R's visualization was that real time visualization could be seen in the 'Viewer' tab of RStudio while the training was occurring. The visualization outputs can been in Figure 3 and Figure 4.

5.5. Time Analysis

To perform time analysis, the time taken to train the Neural Network starting from 10, 20,30...100 epochs for both Python and R was evaluated. In all cases, Python beat R's train time and as the number of epochs increased, the difference between the time also increased thus showing that at least given data and Neural Network of this dimension, Python is far superior to R. This is further illustrated in Figure 5.

5.6. Memory Analysis

To perform memory analysis, Python's memory profiler and R's Profvis tools were used again and the results are shown in Figure 6 and Figure 7. As seen, Python's memory(about 325 MB) usage is far more than that of R(about 36MB)

6. Comparative Study

Figure 8 shows a comparison of Python and R over various parameters based on the experience of this project

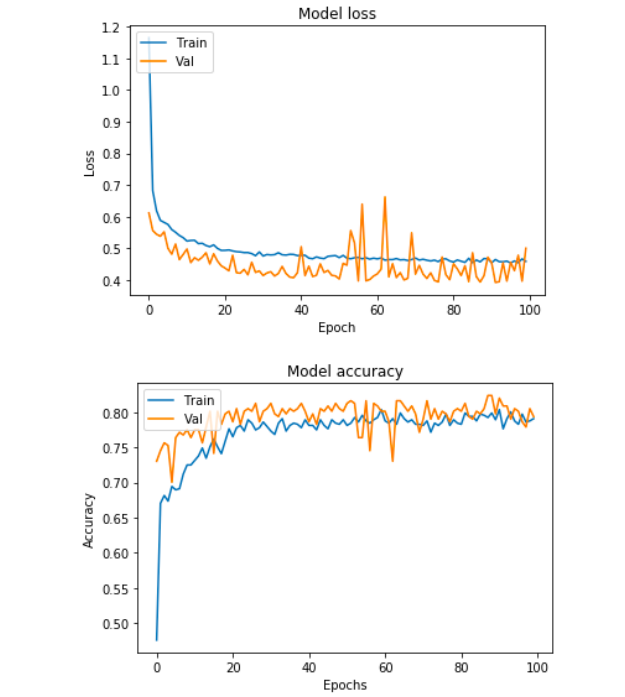


Figure 3: Python: Deep Neural Network training visualization

7. Conclusion

This project aimed to compare Python and R for Data Science. For Machine Learning, the Logistic Regression algorithm was hand coded and compared in both languages. Next, Logistic Regression and Naive Bayes were also fit and evaluated on the data. Accuracy, time analysis and memory analysis was performed. In all cases, Python fit the data faster than R. However, R proved to be more memory efficient. For Deep Learning, a larger dataset was loaded and pre-processing done. A Keras based Deep Neural Network was trained in both Python and R. Time analysis was done in which Python proved to be far superior to R, with training times half of those of R in some cases. In terms of memory consumption, R beat Python with about 10 times less memory consumed while running the Deep Learning script. This project has been a great learning experience. I programmed a Machine Learning algorithm from scratch, which strengthened my fundamental understanding. This code also gave comparable performance to that of both languages' in built libraries Logistic Regression libraries. I also learnt to implement a Deep

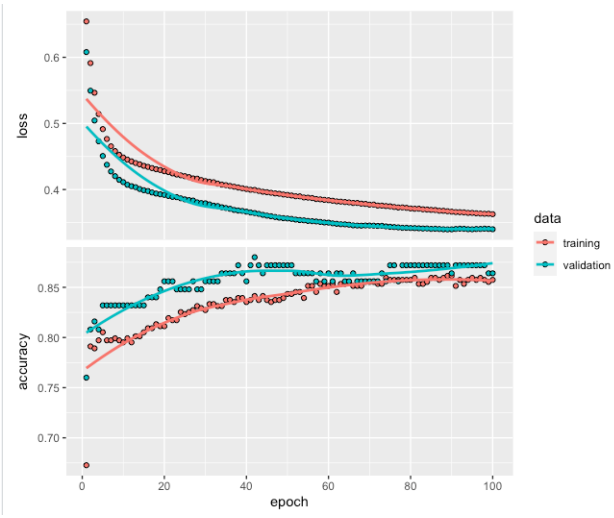


Figure 4: R: Deep Neural Network training visualization

Learning model in R. This project also gave me the opportunity to learn R which I had always wanted to, and while the learning curve was steep, I have now learnt some code snippets are better done in R than in Python.

Acknowledgments

Thanks to Prof. Cormac Flanagan for teaching this course on Programming languages and providing insightful instructions on this project and homeworks.
Thanks to the TA Sherry Lin for all her helpful feedback.

References

[1]]Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261–265). IEEE Computer Society Press.

[2]]David D. Lewis Genkin, Alexander and David Madigan. 2007. Large-scale Bayesian logistic regression for text categorization. technometrics 49, 3 (2007), 291–304.

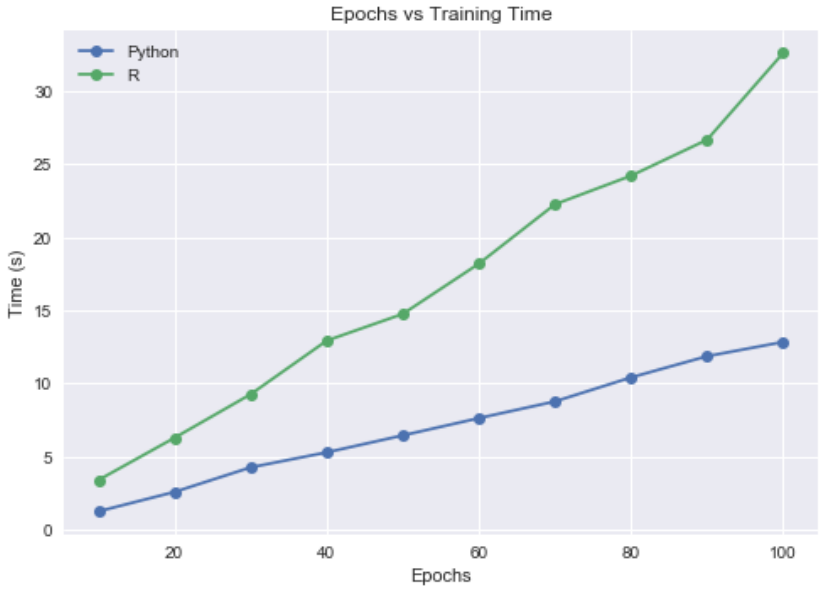


Figure 5: Epochs vs Training Time comparison of Python and R

[3]]Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for Naive Bayes text classification. In Proceedings of AAAI-98, Workshop on Learning for Text Categorization. AAAI Press, 41–48.

[4]] Fabian Pedregosa retrieved from https://pypi.org/project/memory-profiler/

[5]] Christopher M. Bishop, 2006, Pattern Recognition and Machine Learning

[6]] https://www.r-project.org/about.html

[7]] https://rstudio.github.io/profvis/

[8]] https://docs.python.org/3/reference/lexical$_a$*nalysis.html*

[9]]https://www.r-project.org/about.html

[10]]https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html

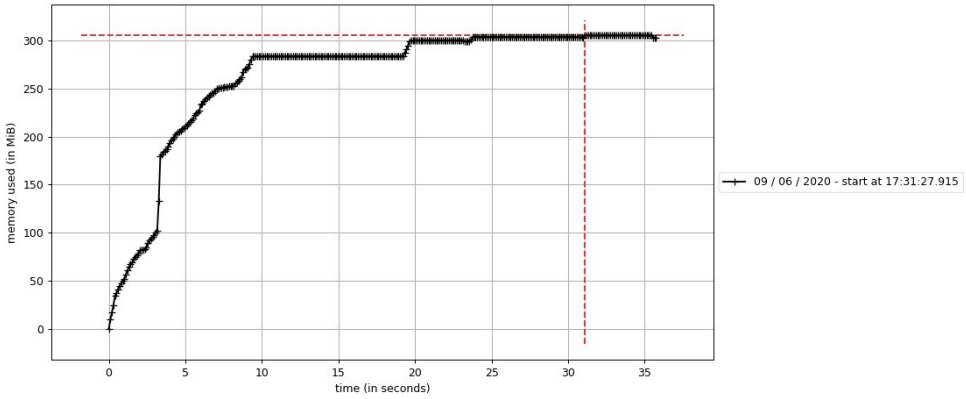


Figure 6: Python: Memory consumption while running Deep Learning script

Code	File	Memory (MB)
▶ %>%	<expr>	0 | 8.6
▶ predict	<expr>	0 | 23.1
▶ plot	<expr>	0 | 0.4
▶ summary	<expr>	0 | 0.0
▶ barplot	<expr>	0 | 0.5
print	<expr>	0 | 0.1
▶ dummyVars	<expr>	0 | 0.6
▶ createDataPartition	<expr>	0 | 1.7
▶ to_categorical	<expr>	0 | 0.8
keras_model_sequential	<expr>	0 | 0.0

Figure 7: R: Memory consumption while running Deep Learning script

Metric	**Python**	**R**
Environment	Jupyter Notebook- open source, free	RStudio- only basic version free
Requirement of third party packages	Less installations, more imports	Many more installations required, many dependencies
Maintenance of packages	PyPI maintains most libraries but not all	All packages maintained at CRAN
Code Readability	More readable to general programmers	Easy but learning syntax takes time
Code Writibility	Easy	Easy
Efficiency (Time to fit/train)	Faster	Comparatively slower
Memory	Higher consumption	Much lower consumption
Visualization	Good	More Superior
Portability & Flexibility	More	Less
Support for other CS tasks	More	Less

Figure 8: Comparative study based on this project