JULIAHUB INDIA PVT LTD

JULIA VS PYTHON

A COMPARATIVE ANALYSIS OF EXECUTION TIME FOR SIMILAR CODE

RAGAV RAJAN S
DATA SCIENCE STUDENT
MANIPAL INSTITUTE OF TECHNOLOGY

INTRODUCTION

In today's world, where data analysis and scientific computing have become more and more important, there are a wide variety of programming languages available to tackle these tasks. Two of the most popular ones are Python and Julia. Both languages have their strengths and weaknesses, and one important aspect to consider when choosing between them is their execution time.

The execution time of a program can have a significant impact on its usability, particularly in applications where large amounts of data need to be processed. In this project, we aim to compare the execution time of similar code in Python and Julia. By doing so, we hope to gain insights into which language is better suited for specific types of tasks.

To carry out this project, we will first select a set of tasks that are commonly performed in image processing and data analysis, and then write equivalent code in Python and Julia. The program we are going to write the code for is "**Detection of CVD from ECG images**". We will use appropriate libraries and functions in each language to ensure a fair comparison. Next, we will measure the execution time of each program on a variety of input sizes and compare the results.

Overall, this project will help us understand the performance characteristics of Python and Julia and guide us in selecting the most appropriate language for specific applications.

PROBLEM STATEMENT

Cardiovascular diseases (heart diseases) are the leading cause of death worldwide. The earlier they can be predicted and classified; the more lives can be saved. Electrocardiogram (ECG) is a common, inexpensive, and non-invasive tool for measuring the electrical activity of the heart and is used to detect cardiovascular disease.

In this project, the power of ML learning techniques will be used to predict the four major cardiac abnormalities:

abnormal heartbeat,

myocardial infarction,

history of myocardial infarction, and

normal person classes using the public ECG images dataset of cardiac patients

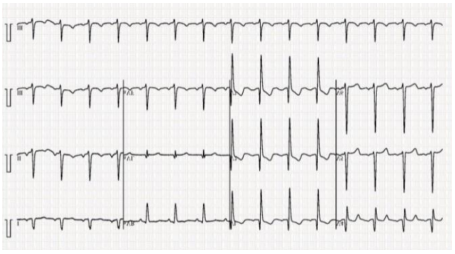
META DATA

ECG images dataset of Cardiac Patients created under the auspices of Ch. Pervaiz Elahi Institute of Cardiology Multan, Pakistan that aims to help the scientific community for conducting the research for cardiovascular diseases. Dataset has 4 divisions on the total 928

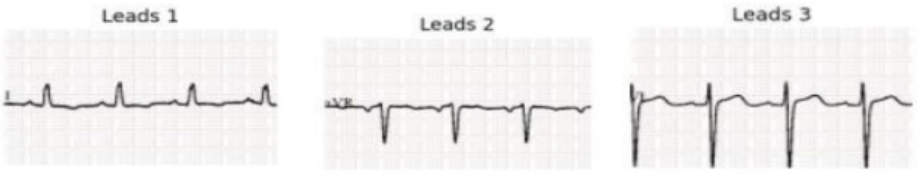
TABLE II
PUBLIC ECG IMAGES DATASET DESCRIPTION.

No.	Class	Number of images
1.	Normal person	284
2.	Abnormal Heartbeat	233
3.	Myocardial Infarction	239
4.	History of Myocardial Infarction	172
Total		928

PREDICTION WORKFLOW

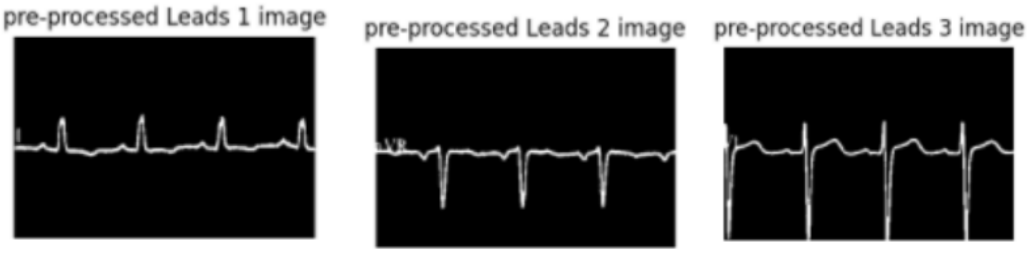


Divide to leads



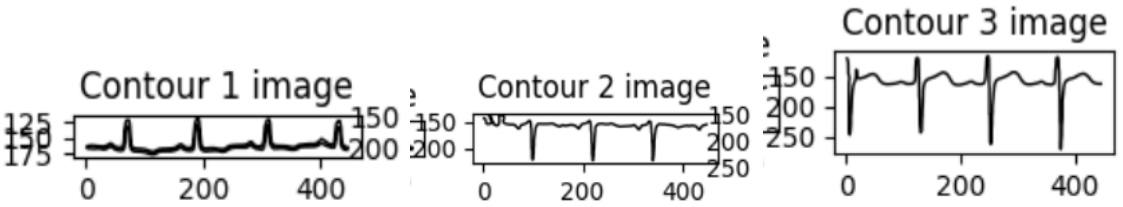
… 12 leads

Image Processing/Binarization



…12

Extract non-zero coordinates



Signal Extraction

	X	Y
0	0.965073	1.000000
1	0.968143	0.985969
2	0.933932	0.974678
3	0.872649	0.968364
4	0.801153	0.966147
...	...	...
250	0.691543	0.970260
251	0.787879	0.971464
252	0.870046	0.975395
253	0.922587	0.984726
254	0.927877	0.998471

	X	Y
0	0.358585	1.000000
1	0.376589	0.995035
2	0.389733	0.989042
3	0.367891	0.984787
4	0.340224	0.979534
...	...	...
250	0.215897	0.016788
251	0.179116	0.012930
252	0.138087	0.009515
253	0.105450	0.004618
254	0.080392	0.000000

	X	Y
0	0.100230	1.000000
1	0.100230	0.991466
2	0.100232	0.982325
3	0.101825	0.973332
4	0.110629	0.965900
...	...	...
250	0.097651	0.031017
251	0.100274	0.022747
252	0.105174	0.014258
253	0.113836	0.006914
254	0.120354	0.000000

…12

Extract only X

	X
0	0.965073
1	0.968143
2	0.933932
3	0.872649
4	0.801153
...	...
250	0.691543
251	0.787879
252	0.870046
253	0.922587
254	0.927877

(255)

	X
0	0.358585
1	0.376589
2	0.389733
3	0.367891
4	0.340224
...	...
250	0.215897
251	0.179116
252	0.138087
253	0.105450
254	0.080392

	X
0	0.100230
1	0.100230
2	0.100232
3	0.101825
4	0.110629
...	...
250	0.097651
251	0.100274
252	0.105174
253	0.113836
254	0.120354

…12

Concatenate the signals

	0	1	2	3	4	5	6	7	8	9	...	245
X	0.965073	0.968143	0.933932	0.872649	0.801153	0.709114	0.612502	0.511295	0.418616	0.377721	...	0.669516

1 rows × 3060 columns

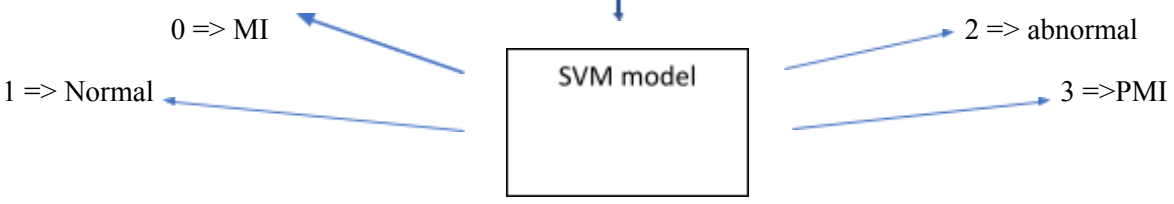
(255 X 12 = 3060)

Principal component Analysis

	0	1	2	3	4	5	6	7	8	9	...	390
0	2.706208	-5.060729	5.657208	2.806774	1.035182	-0.668244	-3.560201	1.744537	1.20501	2.319972	...	0.139657

1 rows × 400 columns

400



FUNCTIONAL COMPARISON

7 functions used to implement this project

Julia	Python
1) function getImage(ecg::ECG, image::AbstractString) load()	1) def getImage(self,image): imread()
2) function DividingLeads(::ECG, image::Matrix{RGB{N0f8}}) same	2) def DividingLeads(self,image): same

3) function PreprocessingLeads(ecg::ECG, DividingLeads::Vector{Matrix{RGB{N0f8}}})
 	
#HSV (Hue, Saturation, Value) conversion
hsv_img = HSV.(i)
 	
#Extract dim from image
channelview(float.(hsv_img))
 	
#Extract color channels
colorview(Gray, mask)
 	
#Resize
imresize(binary_img, (300, 450))
	3) def PreprocessingLeads(self,Leads):
	
	#converting to gray scale
	grayscale=color.rgb2gray(y)
	
	#smoothing image
	gaussian(grayscale, sigma=0.7)
	
	#getting threshold value
	threshold_otsu(blurred_image)
	
	#Resize
	resize(binary_global, (300, 450))

4) function SignalExtraction_Scaling(ecg::ECG, ecg_preprocessed_leads::Vector{Any}) findall(i .!= 0)	4) def SignalExtraction_Scaling(self,bin_glob): np.transpose(np.nonzero(i))

5) function CombineConvert1Dsignal(ecg::ECG, ecg_preprocessed_leads::Vector{Any})
 	
dt1 = fit(UnitRangeTransform, arr)
df1=StatsBase.transform(dt1, arr)
	5) def CombineConvert1Dsignal(self, resized_leads):
	
	#Normalizing
	scaler.fit_transform()

6) function DimensionalReduction(ecg::ECG, ecg_1dsignal::Vector{Any}) same	6) def DimensionalReduction(self,scaled_leads): same
7) function ModelLoad_predict(ecg::ECG, result::Matrix{Float64}) same	7) def ModelLoad_predict(self,result): same

EXECUTION TIME AND ACCURACY COMPARISON

Julia	Python
Accuracy(same for all runs) :- 92.26%	Accuracy(same for all runs) :- 91.90%
Run 1 :- 164.83 seconds	Run 1 :- 188.88 seconds
Run 2 :- 161.33 seconds	Run 2 :- 186.74 seconds
Run 3 :- 168.45 seconds	Run 3 :- 215.43 seconds
Run 4 :- 174.20 seconds	Run 4 :- 193.57 seconds

Run 5:- 167.02 seconds	Run 5 :- 190.37 seconds
Average accuracy:- 167.17 second	Average accuracy:- 194.99 seconds

Looking at the comparison, we can conclude that Julia reduces execution time by almost half a minute and with a 0.3% percent more accuracy.

FUTURE ENHANCEMENTS

1) The Julia code could be even more faster.

The code implemented in Julia is 95 % Julia and 5% python. In the sense, I have used 2 to 3 python package in the Julia version of the code for ease of implementation. In case, I have used all Julia, the code would have been even faster.

2) Parallelization

The code mainly runs on loops which doesn't have any dependencies on each other. This could be parallelized to further reduce the execution time.