University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Estimating Computational Resource Consumption of Dataset Processing

Prepared by

Prabal Gupta
p25gupta@edu.uwaterloo.ca
12 May 2019

Summary

This report revolves around development and verification of methods used to estimate computational resource consumption (primarily computation time) of processing large datasets. Since processing a dataset may take anywhere from hours to days' worth of computing time, it is important to be able to estimate how much resources would be consumed to process a dataset. Essentially, there is a need for a tool that can use a small portion of the dataset to estimate total resource consumption, relative resource consumption of different components, and probability distribution of resource consumption required for processing the entire dataset.

The major points documented in this report can be divided into three categories: test setup, regression techniques, evaluation metrics. Test setup involves generating random programs and random datasets for benchmarking various regression techniques. This report also outlines methods used to make the test data as realistic as possible (e.g. by emulating non-determinism exhibited by certain programs and using non-linear transformations of program inputs). Three different regression techniques were summarized and compared – 'Scaling', 'Linear Transformation of Input Samples', and 'Random Forest Regressor'. Additionally, many evaluation metrics were defined to measure regressors' ability to estimate total, relative, and probability distribution of resource consumption.

The major conclusion in this report is that all regressors have similar predictive power, but the 'Random Forest Regressor' is the best one. The 'Random Forest Regressor' technique is capable of properly 'learning' the non-linear relationship between input values in the dataset and resource requirements (unlike the other two regressors). Moreover, all the regressors seem to hold a large amount of predictive power considering that the ratio of size of training to test dataset was 1:99.

The major recommendation in this report is that the test setup can be significantly improved. The test setup can be modified to generate a wider variety of random programs with more realistic program structure and multiple types of input formats (besides just floating point – which is the case in this report). The test setup uses parameter V to decide number of inputs that the test program will receive, and parameter F to decide the number of functions that the test program will contain. It is recommended that more tests with multiple values of V and F parameters be run.

Table of Contents

List of Figures

List of Tables

1 Introduction

This report revolves around development and verification of methods used to estimate computational resource consumption of processing large datasets. This involves generating random datasets, producing random programs, and comparing multiple potential techniques for estimating resource consumption. The potential solutions have been compared by running tests against the randomly generated programs and datasets. The randomly produced programs and datasets have been generated using techniques that emulate various 'real-life' phenomenon, as detailed in Section 2.

This particular section aims to define the problem scope in its entirety. All necessary details about notation, evaluation metrics, solution requirements, and assumptions accompany the problem definition.

1.1 General Notation

This report refers to some important objects repeatedly, and thus, a specific notation is being used to simplify their representation

1.1.1 Randomly Generated Programs

Randomly generated programs are frequently mentioned in this report. For the sake of simplicity, every program is represented as a set of functions. Hereinafter, every program P with F functions $f_1, f_2, f_3, \dots, f_F$ and V inputs is represented as a set $P_F^V = \{f_1, f_2, f_3, \dots, f_F\}$. Moreover, the function first executed at beginning of program P is referred to as the **Root** of the program. Note that the input to a program can also be represented as a V-element vector.

1.1.2 Randomly Generated Datasets

This report uses randomly generated input vectors to test randomly generated programs and measure their resource consumption. A dataset contains both the randomly generated input vectors and information about program's resource consumption when processing each input vector.

Each dataset contains a matrix of input values whose each row is an input vector to be fed into some randomly generated program. Every dataset also contains a matrix of output values, whose

each row is a vector describing resource consumption of each function within the program when executing some input vector. Every input vector in a dataset has a corresponding output vector.

Input vectors and output vectors have specific constrains placed on them. Each vector element in an input vector is a real-valued random variable in range $[0, 1]$. Each element of an output vector is a real-valued number in range $[0, \infty]$. Note that an n-element dataset is essentially a set of tuples $\langle \text{Input Vector}_i,\ \text{Output Vector}_i \rangle$ for $i \in \{1, 2, \ldots, n\}$. Hereinafter, every dataset is represented as D_F^V i.e. a dataset with V-element input vectors, and F-element output vectors. The relationship between some dataset D_F^V and some program P_F^V is represented graphically in Figure 1.

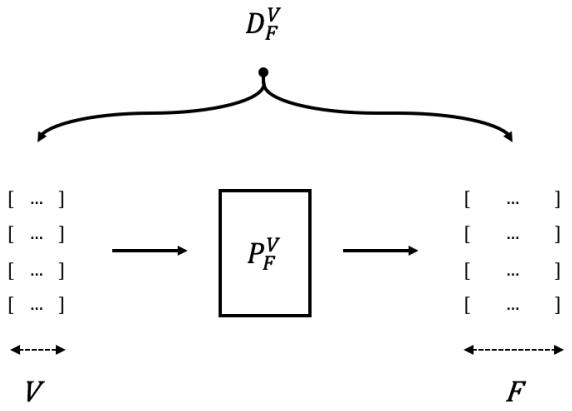


Figure 1: Relationship between a dataset D_F^V and program P_F^V

1.2 Problem Definition

This report aims to outline development of a technique that can:

1. Estimate total resource consumption of a program when provided with an input dataset for processing.
2. Estimate relative resource consumption of each of the program's functions to find performance bottlenecks.
3. Emulate the underlying probability distribution of resource usage (rather than just the average resource consumption).

Thus, finding a solution entails development and use of a **Regressor** that can 'learn' the relationship between input values and resource consumption [1]. Additional details about solution requirements are mentioned in Section 1.4 and Appendix A.

Since this report's aim covers a wide range of possibilities, the problem scope is limited by adding the following constraints to the problem definition:

1. Hereinafter, every program P can be assumed to receive only real-valued vectors as inputs (unless otherwise mentioned). Moreover, each valid input vector must have all elements in range $[0, 1]$.
2. It can be assumed that 'resource', 'cost' and other similar terms refer to processing time i.e. wall-clock time spent on processing. Note that in some program P, whenever a function f_1 calls another function f_2, then the processing time taken by f_2 is not counted by f_1.
3. The programs under consideration are stateless i.e. past inputs do not affect how the program executes when provided with the new inputs.
4. All simulations utilized by this report are executed on the same machine under similar conditions.

The procedure for testing a particular solution/regressor is outlined in

Figure 2 and can be summarized as follows:

1. **Test Setup:** Feed randomly generated input vectors to a randomly generated program P_F^V. Store the output vectors (containing processing time of each function) with corresponding input vectors to form a dataset D_F^V.
2. **Regression:** Split the dataset into training set, test set. Train the regressor to learn the mapping between input vectors and output vectors using the training set. After training, pass input vectors in test set through the regressor to generate predicted output vectors.
3. **Evaluation:** Evaluate regressor performance by comparing predicted output vectors with the ideal output vectors in the test set.

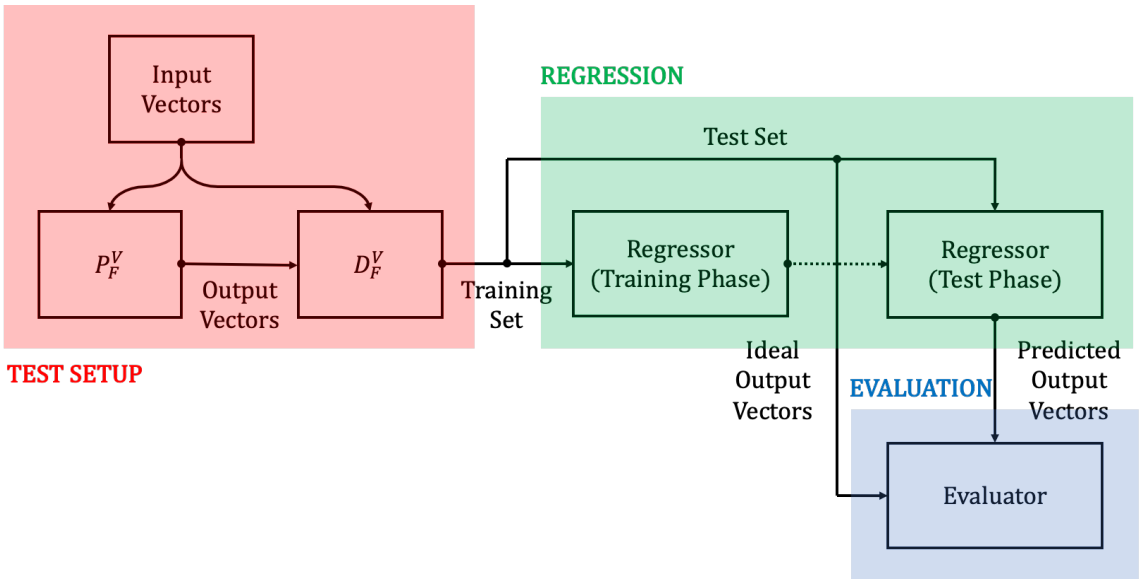


Figure 2: Summary of procedure for testing, evaluating a regressor

1.3 Evaluation Metrics

Custom evaluation metrics were defined to benchmark performance of different regressors. All the metrics were designed to be values between 0 and 1 for ease of comparison. The main evaluation metric is a measure of **Correctness,** which is calculated as a function of custom defined metrics α, β, and γ that measure similarity between the values of predicted output vectors and the ideal output vectors. More details on these metrics are available in Section 4.1.

1.4 Solution Requirements

Refer to Appendix A for complete details regarding solution requirements. The following are some key solution requirements:

1. For some dataset D_{12}^{10} generated using a program P_{12}^{10}, no more than 1% of D_{12}^{10} may be used as training data for the regressor.
2. Regressor should be trainable, testable in less than 5% of the time taken by P_{12}^{10} to process all input vectors in dataset D_{12}^{10}.
3. Correctness metric must exceed 90%. Moreover, additional constraints on values of α, β, and γ similarity metrics mentioned in Appendix A must be satisfied.

2 Test Setup

This section aims to dive into the test setup phase of the procedure outlined in Section 1.2. This involves analyses of technique used to generate random input vectors, random programs, and random datasets.

2.1 Random Transformer

Random Transformer a crucial component required for dataset generation. This component generates a random non-linear transformation and uses it to generate random input vectors. This aims to emulate 'real-life' datasets where the inputs often have non-trivial relationships among each other. Note that the random transformer guarantees that the transformed vectors have all their elements in range $[0, 1]$.

2.2 Input Vector Generation

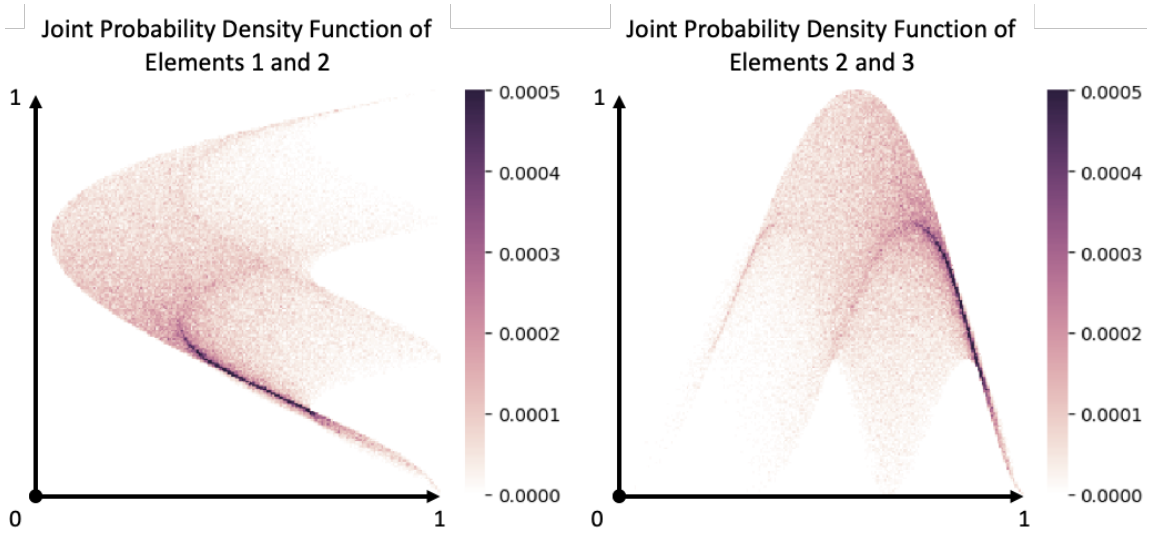
Test setup phase requires that random input vectors be generated. This is done by first simply generating vectors with every element a continuous uniform random variable in range $[0, 1]$. The probability density function of a continuous uniform random variable X in range $[0, 1]$ can be defined as (as per [2] and Table 4 in Appendix B.2):

$$f_{\mathrm{X}}(x) = \begin{cases} 0, & x \notin [0,1] \\ 1, & x \in [0,1] \end{cases} \tag{1}$$

Using (1), the probability density function of a vector of independent and identically distributed continuous uniform random variables in range $[0, 1]$ can be determined as:

$$f_{\vec{\mathrm{X}}}(\vec{x}) = \prod_{i=1}^{n} f_{\mathrm{X_i}}(x_i) = \begin{cases} 0, & \text{if some } x_k \notin [0,1] \\ 1, & \text{if all } x_k \in [0,1] \end{cases} \quad \text{for } k \in \{1, 2, \dots, n\} \tag{2}$$

It is quite rare to have all inputs be identically distributed and independent random variables as above in real-life situations. Thus, all input vectors are generated by applying a randomly generated non-linear transformation to uniformly distributed vectors (whose distribution is defined in Equation (2)). Figure 3 below displays the pairwise joint probability density function of elements of input vectors in some dataset D_F^3. The plots in Figure 3 have been generated with 40,000 bins for each pairwise distribution. Each pixel in the plots below represents a single bin. As expected, applying a random transformation to the uniformly distributed vectors changes their joint probability density function.



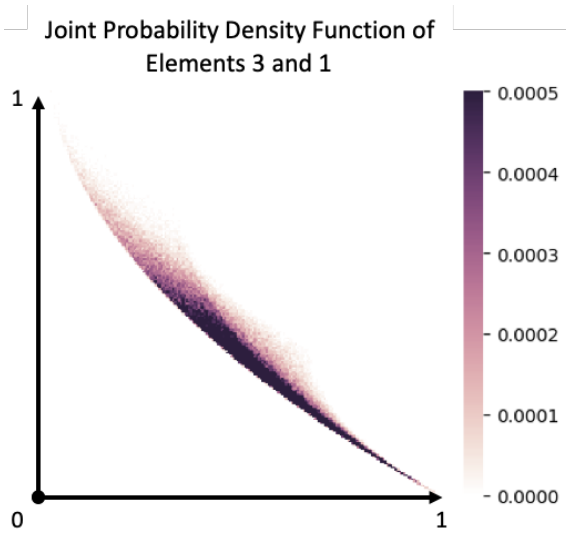


Figure 3: Pairwise joint probability density of elements of input vectors in some dataset D_F^3

Note that the random transformer ensures that each element of transformed vectors is in range $[0, 1]$.

2.3 Program Generation

To avoid bias, a random program generator is used to produce new programs for each test iteration. In each test iteration, all the regressors are tested against the same program $P_F^V = \{f_1, f_2, f_3, \dots, f_F\}$ to ensure that all alternatives are being tested under identical conditions. Moreover, the random program generator is designed to emulate various 'real-life' phenomenon. This section aims to provide details on how new programs are generated randomly.

2.3.1 Program Structure

A randomly generated program P_F^V is generated by creating a set of F functions. Every function is assigned the following:

1. **Work:** Each function is assigned tasks to perform. This is done by assigning a random number of **NOP** ('No Operation') iterations to each function.
2. **Children Function:** The 'Functions' section of Figure 4 summarizes the tree-like relationship among functions of a random program P_6^V. Each non-leaf function is assigned children functions. After a function completes its assigned 'work', it necessarily makes a decision and calls one of its children functions. The decision is made using the scalar obtained by 'flattening' the randomly transformed input vector (as mentioned in Section 2.3.2). Additionally, non-

deterministic behaviour of certain randomized algorithms is emulated by randomizing the decision ~10% of the times [3]. This allows some functions to execute when they would otherwise not be executed due to deterministic decision making.

2.3.2 Input Processing

Every function in a randomly generated program P_F^V makes decisions based on the value of the input vector. Before the program is actually executed, the input is randomly transformed, and then 'flattened'. Random transformations emulate the behaviour of realistic programs which make decisions based on processed input values. Note that the same transformation function is applied to all the input vectors fed to a given program. After transformation, the vector is 'flattened' to a scalar between 0 and 1 using the following expression:

$$\text{FLATTEN}(\vec{L}) = \text{mean}(\vec{L}) \tag{3}$$

Since the output of the random transformer is guaranteed to be in range $[0, 1]$ (as per Section 2.1), so is the scalar obtained after 'flattening'. Figure 4 summarizes the process by which input vectors are processed before being used by the functions composing the program.

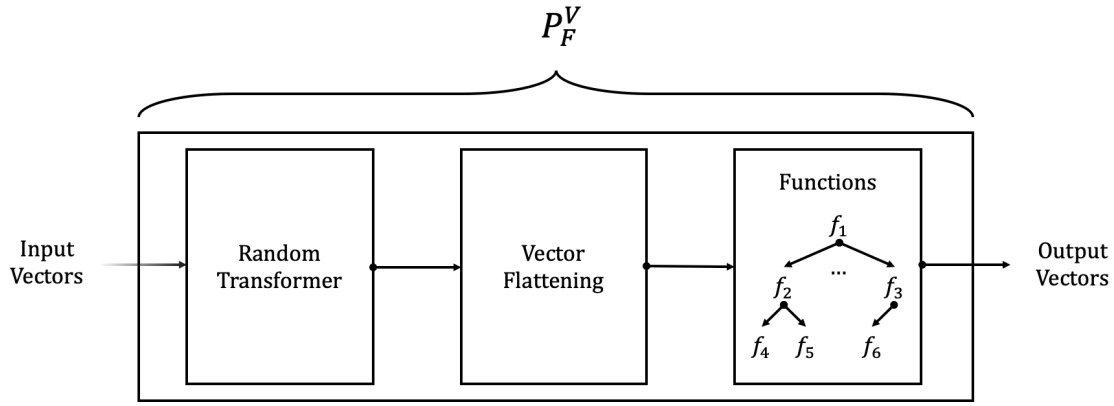


Figure 4: Detailed structure of a random program P_F^V

2.4 Dataset Generation

A dataset D_F^V is generated by feeding randomly generated input vectors to a program P_F^V and then combining the input vectors with the obtained output vectors. Each input vector has V elements and has a corresponding output vector that records how long each function f_i (where $i \in \{1, 2, \dots, F\}$) was executed when processing the input vector. The following figures show potential input and output vectors (generated using some program P_{12}^{10}) of some dataset D_{12}^{10} as heatmaps.

Note that the values in input vectors are unitless, while those in output vectors are measured in seconds. Every row is a unique sample, represented as a horizontal line in plots in Figure 5.

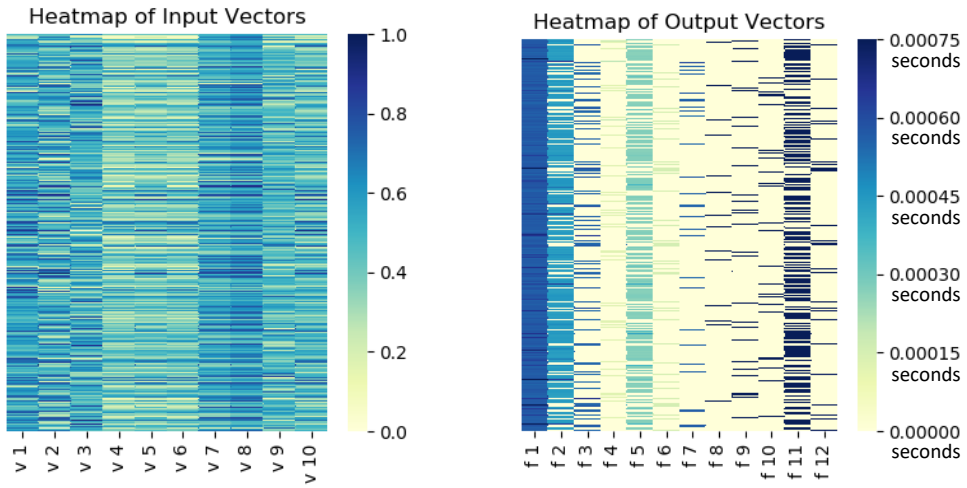


Figure 5: Heatmaps of input, output vectors of a sample dataset D_{12}^{10} with 10,000 samples

3 Regression Techniques

This section analyzes various potential solutions for use as predictive regressors in the regression phase of procedure outlined in Section 1.2. Let D_F^V be a dataset generated using program P_F^V. Note that a dataset D_F^V is essentially a set of tuples containing corresponding input, output vectors (see Section 1.1.2). Let datasets ${D_{(a)}}_F^V$ and ${D_{(b)}}_F^V$ be two randomly sampled, mutually exclusive subsets of D_F^V such that $D_F^V = {D_{(a)}}_F^V \cup {D_{(b)}}_F^V$. Moreover, the datasets ${D_{(a)}}_F^V$ and ${D_{(b)}}_F^V$ may be referred to as training, testing datasets, where $|{D_{(b)}}_F^V| \gg |{D_{(a)}}_F^V|$ i.e. ${D_{(b)}}_F^V$ is significantly larger than ${D_{(a)}}_F^V$. All the subsections in the current section may refer to aforementioned datasets D_F^V, ${D_{(a)}}_F^V$, ${D_{(b)}}_F^V$ and program P_F^V.

3.1 Scaling

This is one of the simplest ways to predict output vectors for new input vectors. This simply involves calculating the 'mean' output vector in the training dataset ${D_{(a)}}_F^V$ and then using this as the predicted output vector for all input vectors in the test dataset ${D_{(b)}}_F^V$. Note that every predicted output vector will be identical.

The major advantage of this technique is the minimal processing time, while still offering decent results (see Section 4.2). One major disadvantage of this technique is its inability to emulate the underlying distribution of the output itself since all the predicted output vectors are identical regardless of the input vector.

3.2 Linear Transformation of Input Samples

This approach requires interpreting the mapping between input vectors and output vectors as a linear transformation function. As shown in Figure 6, the group of input vectors (with n unique input vectors) in training dataset $D_{(a)_F^V}$ can be treated as a matrix $\mathbf{X}$ and group of corresponding output vectors (with n unique output vectors), as matrix $\mathbf{B}$. The required solution is a transformation matrix $\mathbf{A}$ that satisfies the equation $\mathbf{A}\mathbf{X}^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}$ (where $\mathbf{X}$ and $\mathbf{B}$ are known) and transforms each input vector to its corresponding output vector.

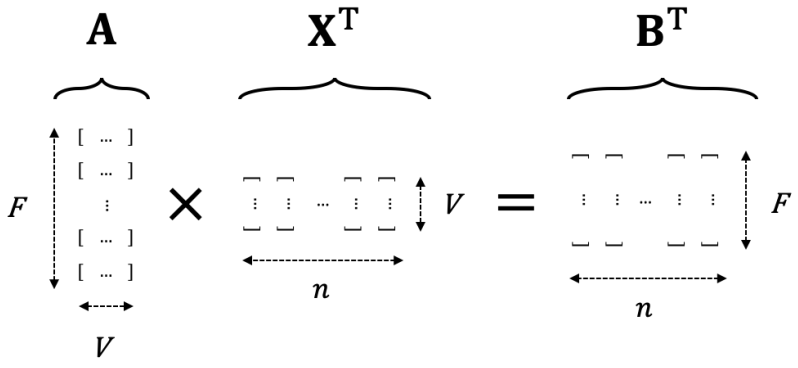


Figure 6: Reformulating regression problem as search for a linear transformation

Two major problems with the aforementioned approach are:

1. The transformation matrix $\mathbf{A}$ might not necessarily exist because matrix $\mathbf{X}^{\mathrm{T}}$ might not necessarily be invertible. This would mean that a single (or any) transformation matrix $\mathbf{A}$ might not exist.
2. Even if a linear transformation did exist, it might not be able to entirely capture the (likely) non-linear relationship between the input and the output vectors.

Of the two major problems, the first can be overcome by use of the **Pseudoinverse** of $\mathbf{X}^{\mathrm{T}}$ (represented as $\mathbf{X}^{\mathrm{T}^{+}}$), which will solve the equation $\mathbf{A}\mathbf{X}^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}$ by minimizing the norm of the error (i.e. 'in least square sense') and provide some approximate value $\overline{\mathbf{A}}$ for $\mathbf{A}$ [4]**.** The relevant matrix manipulations can be simply represented as follows:

$$\mathbf{A}\mathbf{X}^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}$$

$$\overline{\mathbf{A}} = \mathbf{B}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}^{+}}$$

The approximate solution $\overline{\mathbf{A}}$ can be used as the transformation matrix that maps the input vectors to corresponding output vectors. This transformation matrix $\overline{\mathbf{A}}$ can then be used to 'predict' output vectors for input vectors in test dataset ${D_{(b)}}_F^V$ during regression. Figure 7 compares the ideal output vectors in some test set ${D_{(b)}}_{12}^V$ against the predicted output vectors. It is evident that the current technique produces results much similar to the ideal results in comparison to the 'Scaling' technique mentioned in Section 3.1 which returns the same predicted output vector for every test input vector.

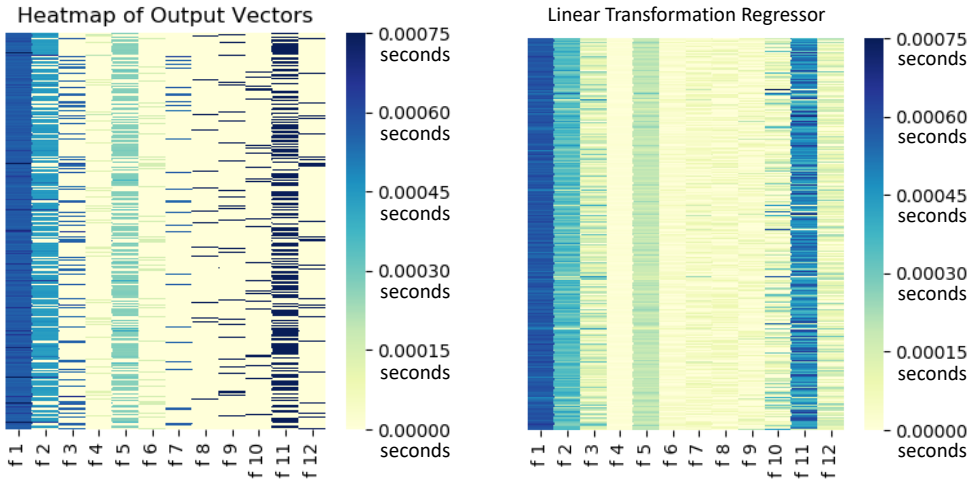


Figure 7: Heatmaps of ideal, predicted output vectors (predicted using 'Linear Transformation of Input Samples' regression technique) for a test dataset ${D_{(b)}}_{12}^V$ with 9,900 samples

3.3 Random Forest Regression

Random Forest Regression uses an ensemble ('group') of simple decision-tree regressors; the technique can be summarized as follows - various sub-samples ${D_{(a_1)}}_F^V$, ${D_{(a_2)}}_F^V$, ${D_{(a_3)}}_F^V$,… from training dataset ${D_{(a)}}_F^V$ are drawn, and a decision tree is trained on each one of them [5]. To make a prediction, the input is fed to every decision-tree, and then the averaged result from all decision trees is used as the output [5]. A pre-implemented random forest regressor provided in **Scikit-Learn** has been used to evaluate this technique.

The major advantage of this model is that it performs well on numerical features [5]. Moreover, unlike the 'Linear Transformation of Input Samples' (see Section 3.2) and 'Scaling' (see Section 3.1) approaches, it is able to capture non-linear relationships between the input vectors and output vectors [5].

One major disadvantage of this model is its relatively slow speed in comparison to other techniques being used in this report (see Table 1 in Section 4.2).

Figure 8 compares the test output vectors in $D_{(b)}{}_{12}^{V}$ against the predicted output vectors. Note that this is the same dataset that was mentioned in Figure 7 in Section 3.2. It is evident that the current technique produces results much similar to the ideal results in comparison to the 'Scaling' and 'Linear Transformation of Input Samples' techniques mentioned in Section 3.1 and Section 3.2 respectively.

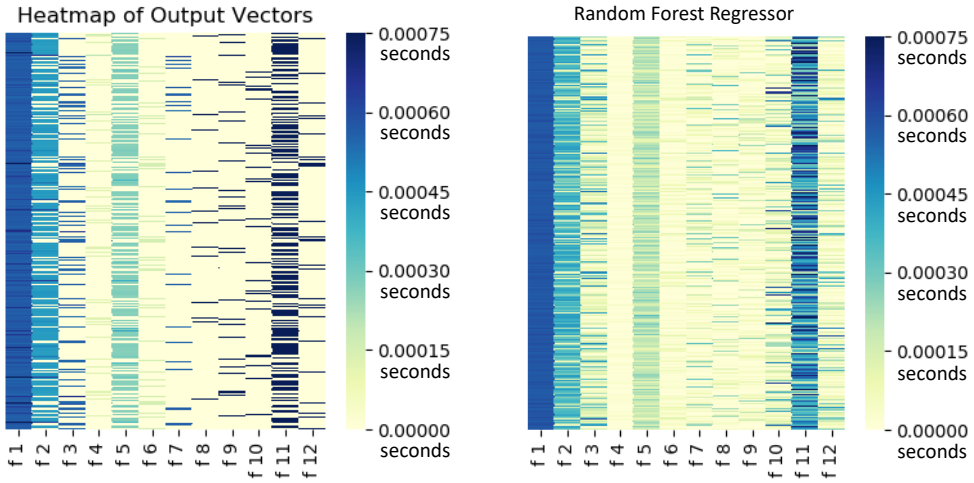


Figure 8: Heatmaps of ideal, predicted output vectors (predicted using 'Random Forest Regressor' technique) for a test dataset $D_{(b)}{}_{12}^{V}$ with 9,900 samples

4 Evaluation

This section aims to provide details on evaluation metrics and use them to benchmark regressors used in this report.

4.1 Evaluation Metrics

The main evaluation metric is a measure of 'Correctness', which is the final arbiter of regressor quality. Moreover, definition of Alpha, Gamma, and Beta metrics has also been provided in the following subsections.

4.1.1 Alpha Similarity Index

Alpha similarity index is used to compare 'similarity' of scalar values. It is defined as follows:

$$\Lambda(t, \bar{t}) = 1 - \min\left(1, \left|\frac{t - \bar{t}}{t}\right|\right) \tag{4}$$

Alpha similarity has been defined such that identical values have a similarity of 1, while values that are not the same have a similarity of less than 1. Very dissimilar values have a similarity of 0.

4.1.2 Beta Similarity Index

Beta similarity index (or angular cosine similarity) is used to compare 'similarity' of vectors [6] (see Table 5 in Appendix C). It is defined as follows:

$$\mathrm{B}\left(\vec{A}, \vec{B}\right) = \mathrm{S}\left(\vec{A}, \vec{B}\right) = 1 - \frac{\cos^{-1}\left(\frac{\vec{A} \cdot \vec{B}}{\overrightarrow{|A|} \cdot \overrightarrow{|B|}}\right)}{\pi} \tag{5}$$

Beta similarity of identical vectors (or any vectors pointing in same direction) is 1. Beta similarity discards the magnitude of the vectors and only considers their orientation when comparing them. Moreover, Beta similarity of highly dissimilar vectors (i.e. orthogonal vectors) is 0.

4.1.3 Gamma Similarity Index

Gamma similarity is used to compare 'similarity' of two matrices. It calculates the discrete probability density function of the values contained in each matrix and compares them using the angular cosine similarity index [6] (see Table 5 in Appendix C). The expression for Gamma similarity is defined as follows:

$$\Gamma(\mathbf{A}, \mathbf{B}) = \mathrm{S}\big(\mathrm{H}(\mathbf{A}), \mathrm{H}(\mathbf{B})\big) \tag{6}$$

In Equation (6), $\mathrm{H}(\mathbf{A})$ refers to the discrete probability density function of values contained in matrix $\mathbf{A}$. Since all values in input, output matrices are non-negative, the gamma index is guaranteed to be non-negative too. Gamma similarity for identical matrices is 1, and less for dissimilar matrices.

4.1.4 Correctness Index

Let there be a dataset D_F^V, whose group of input vectors can be represented as a matrix **X**, group of output vectors as a matrix **B**, and group of predicted output vectors generated by some regressor as $\mathbf{B}_{\mathrm{PRED}}$.

The correctness index is a custom-defined measure for evaluating regressor performance. A higher correctness index implies better regressor performance. It uses the matrix of ideal outputs **B** and predicted outputs $\mathbf{B}_{\mathrm{PRED}}$ to quantify a regressor's prediction power. The correctness index is calculated as the harmonic mean (see Table 5 in Appendix C) of parameters α, β, and γ parameters, and is defined as follows:

$$\text{Correctness Index } = \mathrm{HM}_3(\langle \alpha, \beta, \gamma \rangle) \tag{7}$$

The harmonic mean weighs α, β and γ equally. Moreover, if any of α, β or γ is extremely close to 0, so will be the harmonic mean. Thus, a high value of harmonic mean requires that none of α, β and γ be too small.

In equation (7), the parameter α measures how similar is the ideal total processing time of the dataset to the predicted total processing time. It is calculated as follows:

$$\alpha = \Lambda\big(\mathrm{sum}(\mathbf{B}), \mathrm{sum}(\mathbf{B}_{\mathrm{PRED}})\big) \tag{8}$$

Parameter β measures how similar is the ideal relative processing time of each function to the predicted relative processing times. It is calculated as follows ($\mathrm{mean}(\mathbf{X}, \updownarrow)$ calculates the average of each column vector in matrix **X**):

$$\beta = \mathrm{B}\big(\mathrm{mean}(\mathbf{B}, \updownarrow), \mathrm{mean}(\mathbf{B}_{\mathrm{PRED}}, \updownarrow)\big) \tag{9}$$

Parameter γ measures how similar is the ideal distribution of processing times to the predicted distribution of processing times. It is calculated as follows:

$$\gamma = \Gamma(\mathbf{B}, \mathbf{B}_{\mathrm{PRED}}) \tag{10}$$

4.2 Evaluating Regressors

Table 1 contains results of tests mentioned in Appendix A. As per solution requirements, multiple tests have been performed and the results have been averaged over 100 unique, randomly generated testcases.

Using the results in Table 1, the most important conclusion to be made is that 'Random Forest Regressor' is the best regressor as per the 'Correctness Index' – it beats the 'Scaling' and 'Linear Transformation of Input Samples' techniques by a significant margin. Moreover, it meets the timing constraint by taking less than 5% of the total time taken to process the input vectors in the test dataset (and thus, the entire dataset).

All regressors seem to be similarly capable of predicting total processing time of the entire program (as indicated by high α metric) and relative processing times of individual functions (as indicated by high β metric). Moreover, note that given the fact that only 1% of the randomly generated datasets were used for training, the regressors seem to hold significant amount of predictive power. This could be due to the fact processing time is also largely affected by caching, the predictable behavior of which ends up being 'learnt' by the regressors.

The γ parameter (used to determine similarity of distribution of values in ideal output matrix, predicted output matrix) seems to be the major differentiating factor in the comparison. This likely indicates that the 'Random Forest Regressor' is able to 'learn' potential non-linearities in the mapping between the input and output vectors which the other regressors cannot.

Table 1: Mean of various regressors' performance metrics over 100 iterations of benchmarking

Regressor	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{\gamma}$	$\boldsymbol{Correctness\ Index}$	**Processing Time (sec.)**
Scaling	0.98239	0.97642	0.59360	0.79975	0.00286
Linear Transformation of Input Samples	0.98231	0.98004	0.74181	0.87889	0.00368
Random Forest Regressor	0.98318	0.97657	0.94106	0.96638	0.80537
Ideal Output	**1.00000**	**1.00000**	**1.00000**	**1.00000**	**22.18641**

5 Conclusions

From the analysis of various regression techniques in the report body, it can be concluded that 'Random Forest Regressor' is the most effective regressor. It satisfies all performance constraints for required tests mentioned in Appendix A. The 'Random Forest Regressor' outperforms 'Scaling' and 'Linear Transformation of Input Samples' technique by a significant margin of ~19% and ~9% respectively when compared using the correctness metric. Moreover, it also satisfies the timing constraint by successfully completing training and testing in less than 5% of the time taken to process all input vectors in the test dataset.

All the regressors, especially the 'Random Forest Regressor', hold surprising amount of predictive power considering that the ratio of size of training dataset to size of test dataset is 1:99 i.e. only 1% of the total dataset was used for training. All the regressors have similar predictive power (as indicated by similar α, β values). The similarly large values of α, β parameters for all regressors might be reflecting the inherent predictability of caching which may be affecting execution times of functions in randomly generated programs.

The γ metric seems to be the differentiating factor in regressors' correctness metric. Large differences in γ metric indicate that all regressors are not equally capable of emulating the distribution of output values. Only 'Random Forest Regressor' seems to be capable of 'learning' non-linearities in the mapping between input and output vectors (as indicated by its high γ value). The 'Linear Transformation of Input Samples' regressor seems to only be partially capable of emulating the distribution of values in ideal output vectors, likely because it assumes that the mapping between input and output vectors is linear, and thus, cannot 'learn' non-linearities (indicated by its smaller γ value). Since the 'Scaling' regressor predicts the same output vector for any input vector, it cannot properly emulate distribution of values in ideal output vectors either (and has an even smaller γ value).

6 Recommendations

Based on the analysis and conclusions in this report, the following recommendations are made:

- To better gauge the quality of regressors, they should be tested with programs that take a larger variety of inputs like strings, integers, etc.
- For the sake of limiting the problem scope, processing time was used as the resource under consideration when predicting resource consumption. In future, this could be extended to other computational resources such as memory, disk usage, etc.
- Random program generation could be improved by replacing the tree-like relationship among existing functions (as mentioned in Section 2.3.1) with a graph-like relationship (with certain constraints) to reflect potentially recursive function calls.
- The tests mentioned in solution requirements only require using $V = 10$, and $F = 12$ to limit problem scope and complexity (see Appendix A). Ideally, it would be helpful to measure regressor quality over a range of values of V, F. Multithreaded tests could be used to reduce time taken for the tests to complete.
- Currently, the Gamma similarity index compares the distribution of values in two matrices. This could be improved by potentially comparing the distribution of values in each column or row vector in the two matrices.

Glossary

Alpha Similarity: A custom-defined measure of similarity used to compare two scalars.

Beta Similarity: A measure of similarity used to compare orientation of two vectors.

Cache: Hardware component that provides fast access to frequently used portions of memory. Code with minimal cache-misses operates multiple orders of magnitude faster than something with a large cache-miss rate.

Correctness Measure: A custom-defined measure of similarity used to compare one regressor to another.

Gamma Similarity: Custom measure of similarity for comparing contents of two different matrices.

NOP: Acronym for 'No Operation'; refers to an instruction that does nothing in the CPU.

Pseudoinverse: An approximate equivalent to inverse for non-invertible matrices.[1]

Random Transformer: A component that generates and applies a randomly generated non-linear transformation to vectors.

Random Variable: a variable that attains random values in accordance with certain properties and assumes a value based on the result of an experiment.[2]

- **Discrete Random Variable:** a random variable that can only a attain value from a set of discrete values.
- **Continuous Random Variable:** a random variable that can attain any value from a set of continuous values.

Random Forest Regression: A regression technique that uses weighed outputs of multiple, separately trained decision trees to generate a single output. [3]

Regressor: A tool that can 'learn' the relationship between input and output variables. This tool can be used to estimate outputs for new input values. [4]

Root: The function first to be executed when a program starts. This function always appears first in all possible sequences of function calls that some program executes when invoked with a specific input.

Scikit-Learn: A popular python library for generating machine learning models. [5]

---

[1] B. K. Horn, "Solving over- and under-determined sets of equations," [Online]. Available: http://people.csail.mit.edu/bkph/articles/Pseudo_Inverse.pdf. [Accessed May 2019].
[2] S. Ross, A First Course in Probability, Ninth Edition, Boston, Massachusetts: Pearson, 2014
[3] Turi, "Random Forest Regression," [Online]. Available: https://turi.com/learn/userguide/supervised-learning/random_forest_regression.html. [Accessed May 2019].
[4] B. Beers, "Regression Definition," Investopedia, 1 May 2019. [Online]. Available: https://www.investopedia.com/terms/r/regression.asp. [Accessed 10 May 2019].
[5] Scikit-Learn, "Scikit-Learn Repository," May 2019. [Online]. Available: https://github.com/scikit-learn/scikit-learn. [Accessed May 2019].

WKRPT: Work-Term Report; an acronym used by the University of Waterloo Undergraduate Calendar.[6]

[6] D. W. Harder, "Engineering report writing using Word 2010", University of Waterloo, Waterloo, 2015.

References

[1] B. Beers, "Regression Definition," Investopedia, 1 May 2019. [Online]. Available: https://www.investopedia.com/terms/r/regression.asp. [Accessed May 2019].

[2] S. Ross, A first course in probability, Boston: Pearson, 2014.

[3] P. Raghavan, "Randomized Algorithms," IBM Almaden Research Center, [Online]. Available: http://theory.stanford.edu/people/pragh/amstalk.pdf. [Accessed May 2019].

[4] B. K. Horn, "Solving over- and under-determined sets of equations," [Online]. Available: http://people.csail.mit.edu/bkph/articles/Pseudo_Inverse.pdf. [Accessed May 2019].

[5] Turi, "Random Forest Regression," [Online]. Available: https://turi.com/learn/userguide/supervised-learning/random_forest_regression.html. [Accessed May 2019].

[6] National Institute of Standards and Technology, "COSINE DISTANCE, COSINE SIMILARITY, ANGULAR COSINE DISTANCE, ANGULAR COSINE SIMILARITY," 3 July 2017. [Online]. Available: https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm. [Accessed May 2019].

Appendix A: Solution Requirements

This report aims to outline development of a technique that can:

1. Estimate total resource consumption of a program when provided with an input dataset for processing.
2. Estimate relative resource consumption of each of the program's functions to find performance bottlenecks.
3. Emulate the underlying probability distribution of resource usage (rather than just the average resource consumption).

The following are some key, quantitative solution requirements:

1. For some dataset D_{12}^{10} generated using a program P_{12}^{10}, no more than 1% of D_{12}^{10} may be used as training data for the regressor.
2. Regressor should be trainable, testable in less than 5% of the time taken by P_{12}^{10} to process all input vectors in dataset D_{12}^{10}.
3. Any dataset D_{12}^{10} used for evaluating (for training, testing the regressor) must have at least 10,000 data samples.
4. Tests must be executed at least 100 times i.e. regressor performance metrics will be averaged over 100 unique and randomly generated programs and datasets.
5. Comparison metrics for the 'chosen' regressor must be above minimum acceptable values mentioned in Table 2.

Table 2: Minimum acceptable value of comparison metrics for chosen regressor

Comparison Metric	**Minimum Acceptable Value**
α	90%
β	90%
γ	75%
Correctness Index	90%

Appendix B: Frequently Used Notation

B.1 Probability Definitions, Notations

The following table outlines the various terms and notations commonly used in this work-report:

Table 3: Definitions of frequently used terms and notational conventions [7] [8]

Term	Notation	Definition	
Probability Mass Function	$f_{\mathrm{X}}(x)$	$f_{\mathrm{X}}(x) = \mathrm{P}(\mathrm{X} = x)$	
Notation	$\mathrm{X} \sim f_{\mathrm{X}}(x)$	"Random Variable X has a Probability Mass Function $\mathrm{f}_{\mathrm{X}}(x)$"	
Expected Value	$\mathrm{E}[\mathrm{X}]$	*Continuous Random Variable*	$\mathrm{E}[\mathrm{X}] = \int_{-\infty}^{\infty} x f_{\mathrm{X}}(x) dx$
		Discrete Random Variable	$\mathrm{E}[\mathrm{X}] = \sum_{-\infty}^{\infty} k f_{\mathrm{X}}(k)$
Variance	$\mathrm{Var}(\mathrm{X})$	$\mathrm{Var}(\mathrm{X}) = \mathrm{E}[\mathrm{X}^2] - (\mathrm{E}[\mathrm{X}])^2$	
Program	$P_F^V = \{f_1, f_2, f_3, \dots, f_F\}$.	Some program P with F functions that accepts V-element vectors in range $[0, 1]$	
Dataset	D_F^V	Dataset where the inputs are vectors in $\mathbb{R}^V$ with elements in range $[0, 1]$, and the outputs are vectors in $\mathbb{R}^F$ with elements in range $[0, \infty]$	
Execution Sequence	$\langle f_{k_1}, f_{k_2}, f_{k_3}, \dots, f_{k_n} \rangle$	Vector indicating a sequence of function executions starting at f_{k_1}, ending at f_{k_n}	

[7] S. Ross, *A first course in probability*. Boston: Pearson., 2014.
[8] P. Gupta, *WKRPT 201: Simulation and Analysis of Stochastic Processes*. Waterloo: University of Waterloo, 2018, pp. 21-22.

B.2 Types of Random Variables

The following table outlines the different types of random variables frequently used in probability theory and their important properties.

Table 4: Properties of common random variables [9] [10]

Type	Probability Density Function	Properties
Discrete Uniform Random Variable	$f_{\mathrm{X}}(k) = \begin{cases} 0, & k \notin [a,b] \cap \mathbb{Z} \\ \dfrac{1}{b-a+1}, & k \in [a,b] \cap \mathbb{Z} \end{cases}$	$\mathrm{X} \in [a,b] \cap \mathbb{Z}$ $\mathrm{E}[\mathrm{X}] = \dfrac{a+b}{2}$
Binomial Random Variable	$f_{\mathrm{X}}(k) = \binom{n}{k} p^k (1-p)^{n-k}$	$\mathrm{X} \in [0,n] \cap \mathbb{Z}$ $\mathrm{E}[\mathrm{X}] = np$
Poisson Random Variable	$f_{\mathrm{X}}(k) = \begin{cases} 0, & k < 0 \\ e^{-\lambda}\dfrac{\lambda^k}{k!}, & k \geq 0 \end{cases}$	$\mathrm{X} \in [0,\infty) \cap \mathbb{Z}$ $\mathrm{E}[\mathrm{X}] = \lambda$
Continuous Uniform Random Variable	$f_{\mathrm{X}}(x) = \begin{cases} 0, & x \notin [a,b] \\ \dfrac{1}{b-a}, & x \in [a,b] \end{cases}$	$\mathrm{X} \in [a,b]$ $\mathrm{E}[\mathrm{X}] = \dfrac{a+b}{2}$
Continuous Degenerate Random Variable	$f_{\mathrm{X}}(x) = \delta(x-k) = \begin{cases} 0, & x \neq k \\ \infty, & x = k \end{cases}$	$\mathrm{X} = k$ $\mathrm{E}[\mathrm{X}] = k$
Normal Random Variable	$f_{\mathrm{X}}(x) = \dfrac{\exp\left(\dfrac{-(x-\mu)^2}{2\sigma^2}\right)}{\sqrt{2\pi}\sigma}$	$\mathrm{X} \in (-\infty,\infty)$ $\mathrm{E}[\mathrm{X}] = \mu$
Exponential Random Variable	$f_{\mathrm{X}}(x) = \begin{cases} 0, & x < 0 \\ \lambda e^{-\lambda x}, & x \geq 0 \end{cases}$	$\mathrm{X} \in [0,\infty)$ $\mathrm{E}[\mathrm{X}] = \dfrac{1}{\lambda}$

[9] S. Ross, *A first course in probability*. Boston: Pearson., 2014.
[10] P. Gupta, *WKRPT 201: Simulation and Analysis of Stochastic Processes*. Waterloo: University of Waterloo, 2018, pp. 21-22.

Appendix C: Miscellaneous Mathematical Tools and Metrics

This table outlines the definitions of important mathematical tools and metrics utilized in the work-report.

Table 5: Summary of important mathematical tools and metrics

Tool	Definition	Properties
Angular Cosine Similarity	$\mathrm{S}(\vec{A},\vec{B}) = 1 - \frac{\cos^{-1}\left(\frac{\vec{A}\cdot\vec{B}}{\overrightarrow{\lvert A\rvert}\cdot\overrightarrow{\lvert B\rvert}}\right)}{\pi}$	Metric in range $[0,1]$ used to measure 'similarity' of two vectors. Larger the value, more similar the vectors. [11]
Matrix Distribution	$\mathrm{H}(\mathbf{A})$	Treats matrix as a list of numbers and generates a discrete probability density.
Alpha Similarity	$\Lambda(t,\bar{t}) = 1 - \min\left(1, \left\lvert\frac{t-\bar{t}}{t}\right\rvert\right)$	Metric in range $[0,1]$ used to measure 'similarity' of two scalars. Larger the value, more similar the inputs.
Beta Similarity	$\mathrm{B}(\vec{A},\vec{B}) = \mathrm{S}(\vec{A},\vec{B})$	Identical to Angular Cosine Similarity
Gamma Similarity	$\Gamma(\mathbf{A},\mathbf{B}) = \mathrm{S}(\mathrm{H}(\mathbf{A}),\mathrm{H}(\mathbf{B}))$	Compares two matrices by measuring Angular Cosine Similarity of the distribution of their contents. Guaranteed to be in range $[0,1]$. Larger the value, more similar the inputs.
Harmonic Mean	$\mathrm{HM}_n(\vec{V}) = \frac{n}{\sum_{i=1}^{n}\left\lvert\frac{1}{V_i}\right\rvert}$	Harmonic mean of elements of n-vector $\vec{V}$ (each element must be non-negative).

[11] National Institute of Standards and Technology, "COSINE DISTANCE, COSINE SIMILARITY, ANGULAR COSINE DISTANCE, ANGULAR COSINE SIMILARITY" 3 July 2017. [Online].
Available: https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm. [Accessed May 2019].