Modelling VM Latent Characteristics and Predicting Application Performance using Semi-supervised Non-negative Matrix Factorization

Yuhui Lin
University of St Andrews
St Andrews, UK
Email: yl205@st-andrews.ac.uk

Adam Barker
University of St Andrews
St Andrews, UK
Email: adam.barker@st-andrews.ac.uk

John Thomson
University of St Andrews
St Andrews, UK
Email: j.thomson@st-andrews.ac.uk

***Abstract*—Selecting a suitable VM instance type for an application can be a difficult task because of the number of options and the variety of application requirements. Recent research takes a data-driven approach to model VM performance, but this requires carefully choosing a small set of relevant benchmarks as input. We propose a semi-supervised matrix-factorization-based latent variable approach to predict the performance of an unknown new application. This method allows to take a large set of benchmarks as input for VM performance modelling, and it uses the model and the performance measure of the new application on some of the target VMs to predict the performance on the rest of all VMs. We ran experiments with 373 micro-benchmarks from *stress-ng* and 37 AWS EC2 VMs to predict the scores of *Geekbench* accurately. Our initial results showed that the RMSE and STD of the predicted scores are 6.7 and 4.5 when sampling Geekbench on 5 VMs, and 10.0 and 2.8 when sampling 10.**

***Keywords*-Cloud Computing, Matrix Factorization, Latent Variable Model, Micro-Benchmarks**

I. Introduction

Cloud providers offer various predefined VM configurations as VM instance types for Infrastructure as a service (IaaS). For example, *c4.large* is a computation optimised VM with 2 vCPUs and 3.75 GB memory. However, selecting a suitable VM is a difficult task. Different applications perform differently depending on the characteristics of the underlying VM, and there is often little accurate information about how each VM is likely to perform. Typically, cloud providers give performance information from the traditional view of physical machines, e.g. vCPU, memory and network performance. However, this high-level description is not sufficient to estimate the performance of general applications.

Recent research [1, 2, 3] take a data-driven approach to train VM performance models from benchmark data using Machine Learning (ML) techniques. The word 'performance' here is a general term to reflect the measurements of users' service-level-objectives. The benchmarks used in their setup were carefully selected for the target applications, and the number of benchmarks used at the decision-making stage is small. This becomes a constraint because picking relevant benchmarks for a new application is a difficult task, especially when the application is a black-box or grey-box and the prior knowledge of the requirement of the application is limited.

In this paper, we propose an alternative latent variable approach to model VM performance so that users to can data from a large number of benchmarks to model VM performance. By structuring benchmark data as a VM-Benchmark matrix, we formulate the challenges of VM performance modelling as a matrix decomposition problems. By applying a low-rank approximation technique, called *Non-negative Matrix Factorization* (NMF) [4], VM latent variables and weights can be extracted for unobservable latent characteristics. The problem of performance prediction for a new application can then be formulated as a matrix completion problem, assuming that sample performance measures can be obtained beforehand, e.g. by running the application.

To improve the quality of VM latent characteristics, as inspired by *task-driven dictionary learning* [5, 6], we take a semi-supervised approach during the process of learning latent variables to utilise prior knowledge of VMs as labels, e.g. number of vCPU. The key contributions of this paper are:

- a novel approach to formulate VM performance modelling and prediction as matrix decomposition and completion problems;
- the application of the combination of unsupervised dimension reduction technique, i.e. NMF, and supervised ML in VM performance modelling;
- an extension of NMF to predict the performance of new applications using the latent model;
- experiments using 373 micro-benchmarks to predict *Geekbench* scores for 37 AWS EC2 VMs. RMSE and STD of the prediction are (6.7, 4.5) when 5 random samples are provided; and (10.0, 2.8) for 10 samples.

§II gives the background of NMF. §III formulates the modelling and prediction problem, §IV discusses the experiment results. Related work and future work are discussed in §V, followed by conclusion in §VI.

II. Non-negative Matrix Factorization

We adopt the following convention for notations. The mathematics presented in this paper are necessary to re-

produce our work. Bold capital letters for matrices, e.g. $\mathbf{X}$; bold lower-case letters for vector, e.g. $\vec{\mathbf{v}}$, vector symbol with transpose for column vector, e.g. $\vec{\mathbf{v}}_j^T$, non-bold letters for scalar, e.g. m, and subscripts for element position, e.g. x_{ij}, is the scalar in row i column j of the matrix $\mathbf{X}$.

Matrix Factorization (MF) is a low-rank approximation technique which decomposes a data matrix to two matrices with lower dimensions, i.e.,

$$\mathbf{V} \approx \mathbf{WH} \text{ s.t. } \mathbf{V} \in \mathbb{R}^{m \times n}, \mathbf{W} \in \mathbb{R}^{m \times r}, \mathbf{H} \in \mathbb{R}^{r \times n}$$

One typical use of MF is to learn features in an unsupervised manner from a set of vectorised data, e.g, $\vec{\mathbf{v}}_i \in \mathbb{R}^m$. The resultant matrices are considered as a collection of features (each column is a feature encoding), e.g. $\mathbf{W}$ consists of r features $\vec{\mathbf{w}}_j^T$ where $\vec{\mathbf{w}}_j^T \in \mathbf{R}^m$, and the weights of features for each data, e.g. $\mathbf{H}$ is latent variable weights for n VM, e.g. $\vec{\mathbf{h}}_j^T$ where $\vec{\mathbf{h}}_j^T \in \mathbf{R}^r$.

NMF [4] poses an additional non-negative constraint on the decomposition matrices, e.g. $w_{ij}, h_{jk} \geq 0$. Such constraint leads to a part-based linear representation of data, i.e.

$$v_{ij} \approx \vec{\mathbf{w}}_i \cdot \vec{\mathbf{h}}_j^T$$

Obtaining $\mathbf{W}$ and $\mathbf{H}$ is an optimisation problem, i.e.,

$$\min_{\mathbf{W},\mathbf{H}} D_\beta(\mathbf{V} \| \mathbf{WH}) \text{ s.t. } \mathbf{W}, \mathbf{H} \geq 0 \quad (1)$$

where D_β is the β-divergence [7]. Examples of β-divergence are the *Euclidean* distance (β = 2) [4], *Kullback-Leibler* (β = 1) divergence [4] and *Itakura-Saito* divergence (β = 0) [8]. In the case of Euclidean distance, (1) becomes:

$$\min_{\mathbf{W},\mathbf{H}} \|\mathbf{V} - \mathbf{WH}\|_F^2 \text{ s.t. } \mathbf{W}, \mathbf{H} \geq 0 \quad (2)$$

where the *Frobenius* norm ($\| \cdot \|_F$) is sum of the absolute squares of its element, i.e. $\|\mathbf{X}\|_F^2 \equiv \sum_{i=1}^m \sum_{j=1}^n |x_{ij}|^2$. In [9], *multiplicative update rules* were developed to find a local minimum while maintaining the non-negative property:

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{VH}^T)}{(\mathbf{WHH}^T)} \quad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{(\mathbf{W}^T\mathbf{V})}{(\mathbf{W}^T\mathbf{WH})}$$

where $\odot$ is *Hadamard product* (a.k.a. element-wise product) which takes two matrices of the same dimensions and generates a matrix with the same dimension where each element i, j is the product of elements i, j of the original two matrices; and $\frac{(\cdot)}{(\cdot)}$ is the element-wise division.

The multiplicative update rules can provide guarantees to find a local minimum but not a global one due to non-convex. Moreover, the solutions $\mathbf{W}$ and $\mathbf{H}$ can be non-unique which is known as *ill-posedness*.

III. Latent variable modelling and prediction

Predicting performance measures of an application from a given set of benchmark scores is essentially, to extract meaningful representation from the scores, and then to make predictions based on the correlation between the benchmark scores and sample performance measurements. In this section, we present our latent approach for modelling and prediction.

A. Problem formulation

Consider that there are m benchmarks and n VMs, a $m \times n$ benchmark data matrix can be structured, i.e. $\mathbf{V} \in \mathbb{R}^{m \times n}$. Each column vector $\vec{\mathbf{v}}_j^T$ becomes vectorised performance data for each VM. $\mathbf{V}_{m \times n}$ can be represented by a latent variable matrix $\mathbf{W}_{m \times r}$ and a weight matrix $\mathbf{H}_{r \times n}$. From the VM point of view, $\vec{\mathbf{v}}_j^T$ is vectorised VM performance data encoded by benchmark scores; each $\vec{\mathbf{w}}_j^T$ is a common VM latent characteristic extracted from $\mathbf{V}$; and $\vec{\mathbf{h}}_j^T$ are the weight for each VM. From the benchmark point of view, each element in $\mathbf{V}$ is the result of linear combinations of its weighted relation with the latent characteristics.

The prediction then becomes problems of tuning the relations between the new application and each latent characteristics to fit the results with the performance measure samples. This is depicted in Figure 1.

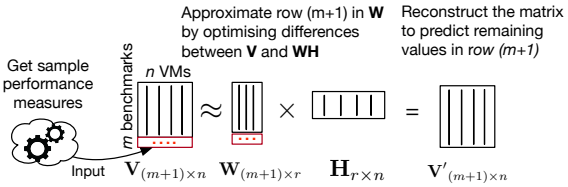

Figure 1: Predicting application performance.

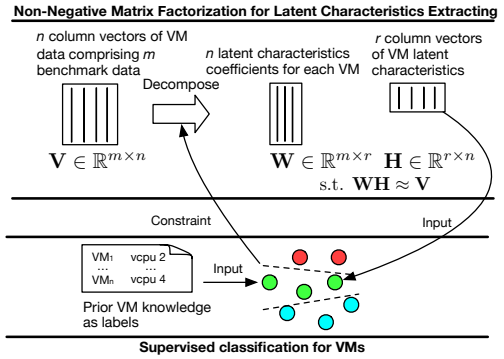

Figure 2: Extracting VM latent characteristics using NMF with constraints from supervised classification.

Obtaining $\mathbf{W}$ and $\mathbf{H}$ is, in general, a non-convex optimisation problem with multiple possible decomposition solutions. To tackle this problem, we combine NMF with supervised ML tasks, e.g. logistic regression, in the modelling process. As shown in Figure 2, we utilise prior knowledge of VM as labels for each VM when running the supervised ML tasks, e.g. multiclass classification for the label of vCPU numbers or machine type. During the process, the latent characteristics from NMF are used as input data for the supervised ML task. In return, the results of the tasks are fed back to NMF to drive characteristics modelling in a 'discriminative' way. In the rest of the section, we will present the details of the design choices.

B. Loss functions and constraints with supervised ML

The key to combine supervised tasks with NMF is to integrate their loss functions. Suppose there is a vector of labels for each VM, i,e, $\vec{\mathbf{y}} \in \mathbb{R}^n$, where type of $\mathcal{Y}$ depends on the supervised ML task, e.g $\mathcal{Y} := \mathbb{R}$ for linear regression; $\mathcal{Y} := \{-1, 1\}$ for binary classification; and $\mathcal{Y} := \{p_1, ..., p_k\}$ for multiclass classification. In the rest of section, we will illustrate our approach using multiclass classification.

We first apply Hadamard production to the loss function (2) so that null value can be allowed in the data matrix as proposed in [10], i.e. $\min_{\mathbf{W},\mathbf{H}} \|\mathbf{Q} \odot (\mathbf{V} - \mathbf{WH})\|_F^2$, Those entries with null values will be ignored when calculating the distance for the loss function. The motivation is to give users the choice to not provide partial benchmark data, because some benchmarks can fail to run due to technical problems or users just want to save costs for benchmarking. This also enables us to construct the data matrix with missing values during prediction. We will cover the prediction part in §III-D. The loss function ℓ_n for NMF is then defined as

$$\ell_n(\mathbf{V}) = \min_{\mathbf{W},\mathbf{H}} \|\mathbf{Q} \odot (\mathbf{V} - \mathbf{WH})\|_F^2 + \lambda_1\|\mathbf{H}\|_1 + \tfrac{\lambda_2}{2}\|\mathbf{H}\|_F^2 + \lambda_3\|\mathbf{W}\|_1$$

where the last three items are $L1$ and $L2$ norm regularization to prevent the elements from getting too large., and $L1$ norm can also lead to sparsity.

For the supervised ML task, we will use the weights of latent variables, i.e. $\mathbf{H}$, as input data to learn the correlation, i.e. $\mathcal{F}(\vec{\mathbf{h}}_i) = y_i$. Take multiclass classification with k options for example, one can use "one-versus-all" approach to generate k sets of labels for each options. That is, for $\vec{\mathbf{y}} \in \mathcal{Y}$ and $\mathcal{Y} := \{p_1, ..., p_k\}$ where $k \geq 2$, a set of vectors for labels are $\{\vec{\mathbf{y}'_1}, ..., \vec{\mathbf{y}'_k}\}$ where

$$\vec{\mathbf{y}'_i} = [\mathcal{F}_{\mathcal{Y}}(y_1, p_i), ..., \mathcal{F}_{\mathcal{Y}}(y_n, p_i)] \text{ and } \mathcal{F}_{\mathcal{Y}}(y, p) = \begin{cases} 1 & \text{if } y = p \\ -1 & \text{if } y \neq p \end{cases}$$

ℓ_s can then be defined as k logistic regression loss functions:

$$\ell_s(\vec{\mathbf{y}}, \mathbf{H}) = \ell_{s_1}(\vec{\mathbf{y}'_1}, \mathbf{H}) + ... + \ell_{s_k}(\vec{\mathbf{y}'_k}, \mathbf{H})$$

where each ℓ_{s_i} is a binary classification loss function:

$$\ell_s(\vec{\mathbf{y}}, \mathbf{H}) = \min_{\vec{\mathbf{a}} \in \mathbb{R}^r, b \in \mathbb{R}} \vec{\mathbf{1}}^T \log(\exp(-\vec{\mathbf{y}} \odot (\mathbf{H}^T\vec{\mathbf{a}} + b\vec{\mathbf{1}})) + \vec{\mathbf{1}}) + \frac{\nu_1}{2}\|\vec{\mathbf{a}}\|_2^2 + \frac{\nu_2}{2}b^2$$

Now we can integrate the NMF with supervised ML by combining their loss functions, i.e.

$$\ell_n(\mathbf{V}) + \alpha\ell_s(\vec{\mathbf{y}}, \mathbf{H}) \tag{3}$$

where α is used to balance the cost functions for NMF and supervised tasks.

C. Optimisation

$\mathbf{H}$ is the key to linking NMF with supervised ML tasks as it appears in both ℓ_n and ℓ_s. Algorithm 1 shows the process of optimising (3) It takes the following input: data matrix $\mathbf{V}$ for NMF, $\vec{\mathbf{a}}$ for supervised ML, λ, ν, α for regularization, η for learning rate, t for maximum iteration and d for the periodical adjustment. $\mathbf{W}$ and $\mathbf{H}$ are initialised using *NNDSVD* (Non-negative Double Singular Value Decomposition) [11], which is a SVD-based non-random initialisation for sparse decomposition.

Algorithm 1 Optimisation

Input: $\mathbf{V}, \vec{\mathbf{y}}, \alpha, \eta, \lambda, \nu, t, d$
Output: $\mathbf{W}, \mathbf{H}$

procedure
  $\vec{\mathbf{a}}, b, j = 0$
  $\mathbf{W}, \mathbf{H} \leftarrow$ *NNDSVD*.
  for $i = 1$ to t **do**
    Update $\mathbf{W}, \mathbf{H}$ using (4)
    if $j == d$ **then**
      Update $\tilde{\mathbf{a}}, b$ for ℓ_s
      $\mathbf{H} \leftarrow \mathcal{P}_+[\mathbf{H} - \eta_{\mathbf{H}}\nabla_{\mathbf{H}}\ell_s]$
    else
      $j \leftarrow j + 1$
    end if
    if Convergence **then**
      return $\mathbf{W}$, $\mathbf{H}$
    end if
  end for
end procedure

Algorithm 2 Prediction

Input: $\vec{\mathbf{v}}_{m+1}, \mathbf{H}, \lambda, t$
Output: $\vec{\mathbf{w}}_{m+1}$

procedure
  $w_{(m+1)j} \leftarrow Avr_+(\vec{\mathbf{w}}_j^T)$
  for $i = 1$ to t **do**
    Update $\vec{\mathbf{w}}_{m+1}$ with (6)
    if Convergence **then**
      return $\vec{\mathbf{w}}_{m+1}$
    end if
  end for
  return $\vec{\mathbf{w}}_{m+1}$
end procedure

The optimisation process iteratively applies the multiplicative updating approach to update non-negative variables $\mathbf{W}$ and $\mathbf{H}$ to maintain the non-negative property, i.e.

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{((\mathbf{Q}\odot\mathbf{V})\mathbf{H}^T)}{((\mathbf{Q}\odot(\mathbf{WH}))\mathbf{H}^T + \lambda_3)} \quad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{(\mathbf{W}^T(\mathbf{Q}\odot\mathbf{V}))}{(\mathbf{W}^T(\mathbf{Q}\odot\mathbf{WH}) + \lambda_1 + \lambda_2\mathbf{H})} \tag{4}$$

For every d iterations, gradient descent is applied to obtain the optimal values of $\vec{\mathbf{a}}$ and b with current $\mathbf{H}$ in the supervised ML loss function. Then, gradient descent is applied to $\mathbf{H}$ to bias the latent characteristics using the feedback from the supervised ML task. Take logistic regression for example, it is:

$$\nabla_{\mathbf{H}}\ell_s = -\vec{\mathbf{a}}\frac{(\phi\odot\vec{\mathbf{y}}^T)}{(\vec{\mathbf{1}}^T + \phi)} \text{ where } \phi = \exp\left(-\vec{\mathbf{y}}^T \odot \left(\vec{\mathbf{a}}^T \cdot \mathbf{H} + (b \cdot \vec{\mathbf{1}})^T\right)\right)$$

Note that because the gradient descent update cannot guarantee $\mathbf{H}$ to be non-negative, the algorithm will set w_{ij} to 0 if it becomes negative. This is denoted by $\mathcal{P}_+[\bullet]$.

D. Prediction

The intuition is to consider the new performance measurement as part of the benchmark data matrix, and then to run matrix factorization to approximate the unknown elements using the latent variables models and the weights, i.e. $\mathbf{W}$ and $\mathbf{H}$. Note that the current value in $\mathbf{W}$ and $\mathbf{H}$ remain the same during the process of prediction. When samples of performance measurements are obtained, a sparse vector can be structured and then appended to $\mathbf{V}$, i.e. $\vec{\mathbf{v}}_(m + 1)$. This new vector will be approximated by a new row in the $\mathbf{W}$ and the weights $\mathbf{H}$, i.e.

$$\vec{\mathbf{v}}_{m+1} \approx \vec{\mathbf{w}}_{m+1}\mathbf{H}$$

Prediction now becomes the following optimisation problem:

$$\ell_p = \min_{\vec{\mathbf{w}}_{m+1}} \|\vec{\mathbf{q}} \odot \vec{\mathbf{v}}_{m+1} - \vec{\mathbf{w}}_{m+1}\mathbf{H}\|_2^2 + \tfrac{\lambda_{wp}}{2}\|\vec{\mathbf{w}}_{m+1}\|_2^2 \tag{5}$$

A masking vector $\vec{\mathbf{q}}$ and the Hadamard production is also introduced to allow missing elements.

Now we utilise the $\mathbf{W}$ to initialise $\vec{\mathbf{w}}_{m+1}$ with more meaningful values rather than using an arbitrary numbers. Recall that each element w_{ij} can also be considered as the relation between the applications and the latent characteristic. We assign the initial value of $w_{(m+1)j}$ using the average of non-negative values in the same column. The application starts the optimisation at the position where it has an average relation with each latent characteristics. We now apply to the following update rule to optimise $\vec{\mathbf{w}}_{m+1}$:

$$\vec{\mathbf{w}}_{m+1} \leftarrow \vec{\mathbf{w}}_{m+1} \odot \frac{((\vec{\mathbf{q}} \odot \vec{\mathbf{v}}_{m+1})\mathbf{H}^T)}{((\vec{\mathbf{q}} \odot (\vec{\mathbf{w}}_{m+1}\mathbf{H}))\mathbf{H}^T + \lambda_{wp}\vec{\mathbf{w}}_{m+1})} \quad (6)$$

The details of the optimisation is provided in Algorithm 2.

IV. Experiments

We implement a prototype in *python* using *Matrix Calculus* [12] and ML algorithms from the *sklearn* library. As an initial experiments to demonstrate the latent variable approach, we set up a naive scenario that a user would like to get a comprehensive performance profile of VMs from AWS EC2 measured by a general-purpose high-level benchmark called *Geekbench* [13]. The set of VMs of interests comprise of 37 *AWS EC2* VMs from 4 categories, e.g. *General Purpose* and *Computation Optimized* and 16 instance type families, e.g. *M5* and *C4*, as shown in Table I. For the

Table I: List of AWS EC2 VMs used for evaluation

General Purpose	**Memory Optimized**
t2.micro, t3.micro, m5.{large, xlarge, 2xlarge}, m5a.{large, xlarge, 2xlarge}, m5n.{large, xlarge, 2xlarge}, m4.{large, xlarge, 2xlarge}	r5.{large, xlarge, 2xlarge}, r5a.{large, xlarge, 2xlarge}, r5n.{large, xlarge, 2xlarge}, r4.{large, xlarge, 2xlarge}
Compute Optimized	**Accelerated Computing**
c5n.large, c5.{large, xlarge}, c5d.{large, xlarge, 2xlarge}, c4.{large, xlarge, 2xlarge},	g4dn.xlarge, g3s.xlarge
Summary: 37 VMs with 4 categories, 16 families, vCPU range of [1,8]	

micro-benchmark data, we choose 353 micro-benchmarks (also called stressor) from *stress-ng* [14], covering different performance aspects, e.g. cpu, cpu-cache, I/O and memory

For the latent variable approach, we utilise the knowledge of vCPU numbers as the input for the Semi-supervised NMF (SNMF), i.e. classifying of the vCPU numbers as the supervised ML task. To compare the prediction results, we also apply *Random Forest* (RF), *Collaborative Filtering Matrix Factorization* (CF-MF) and *Collaborative Filtering Normal Matrix Factorization* (CF-NMF) to the same setting. We are interested in CF-MF and CF-NMF, because they are also in the family of low-rank matrix decomposition techniques that are applied in *collaborative filtering* (CF) for recommender systems The main difference, compared to ours, is that, the CF approach puts the prediction target directly in the data matrix and then reconstruct the whole matrix for prediction.

We ran each micro-benchmark for 3s on every VMs to get the scores. We then set the running instances to match vCPU numbers to get the full potential of multi-cores.

After getting the Geekbench scores for all VMs, we set up two groups of experiments where one takes 5 random scores from the normalised Geekbench scores as samples, while the other takes 10. For each group, we run 30 iterations to evaluate prediction accuracy for each prediction method, with randomly picked samples in each iteration.

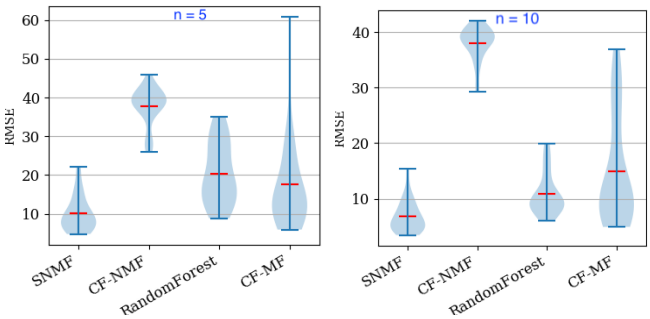

Figure 3: Prediction *RMSE* distribution for the experiments of predicting the *Geekbench* scores for all VMs using n randomly selected VMs as sample scores. The red line indicates the mean of the distribution; and lower value of RMSE (y-axis) suggests more accurate prediction; and a narrow range suggests more stable prediction.

Figure 3 (left) and 3 (right) show violin plots, for each group of experiments respectively, to summarise prediction result assessed by RMSE. SNMF makes a good prediction in term of accuracy and stability for both groups, with (`RMSE mean 10.0, std 4.5`) for the 5 samples group, and (`RMSE mean 6.7, std 2.8`) for the 10 samples one. Generally speaking, providing more samples could improve the prediction for all methods. This is because, when the number of the sample is low, it is more likely that a bad set of samples are drawn, e.g. VM with similar performance. Figure 4 shows a relatively bad prediction result for a SNMF when no sample is drawn for high performance VM. In the case when representative samples are drawn, all methods except for CF-NMF, can provide good prediction in this experiment. By comparing the worst case for all perdition methods, even with bad sample sets, SNMF can still manage to make a reasonable guess because of the more sensible latent characteristics biased by the supervised ML tasks.

It is interesting to compare the accuracy of CF-MF and CF-NMF to SNMF because all use matrix factorization. In SNMF, the modelling and predicting are two separated process, while in CF-NMF and CF-MF, they are combined into one same matrix decomposition. That is, fitting the sample results would have an impact on the latent characteristics to be extracted. This is a sensible setting in the recommender system because the missing data points for prediction are meant to be drawn from the same 'universal data distribution', which may not be true for VM performance prediction.

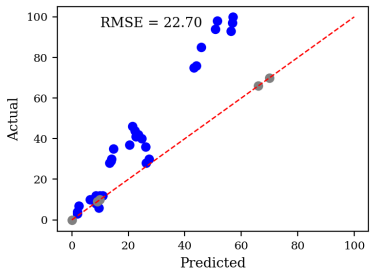

Figure 4: An example of a relatively bad prediction result due to an unrepresentative sample set. The blue dots are prediction, the grey ones are samples and the dotted red line is a reference line meaning perfect accuracy.

The result suggests that drawing sensible sample is crucial for prediction. In future, we consider to combine Bayesian optimization with SNMF to improve prediction stability.

V. Related work & Future work

Semi-supervised NMF was originally proposed in [10, 5]. More recently, they have been applied to the medical domain, e.g. analyse tumour images [15] and ICU mortality data analysis [16]. From the learning prospect, the key idea of all these applications are similar, but the exact formulation of modelling and prediction are different due to the nature of different application domains. In our case, we concern about both resultant matrices And both matrices are used as direct input or heuristics in prediction.

In [17], micro-benchmarks were used as machine performance indicators for software regression testing. But they did not further extract latent variables from the benchmarks. Comparing to other driven-driven approaches, e.g. *Ernest* [3], *Cloudbench* [1] and *Paris* [2], our approach makes the choices of benchmark easier because users can provide a large number of benchmarks and not every single benchmark has to be relevant. There are a number of tools to facilitate benchmarking VMs for decision making, e.g. *GooglePerfkit* [18], *DocLite* [19] and *Cloudbench* [1]. But they do not provide guidance on how to use benchmark scores.

Our experiments are an early attempt to demonstrate the feasibility of the latent approach. The setup is limited, e.g. Geekbench is not a representative cloud application and running each micro-benchmark for 3 seconds is not possible to cover some performance aspects. For the next steps, we plan to run experiments with application-level benchmarks to evaluate our approach in more realistic scenarios. We would like to discover more general latent VM characteristics by utilising the existing benchmark scores from the platform such as *OpenBenchmarking* [20]. It is also interesting to apply our method to industry prediction problems as suggested in [21]. Apart from the possible improvement with Bayesian optimization as discussed in §IV, we also interested in employing statistical approaches of likelihood theory and the bootstrap, as in [22], to handle the case when application performance required weeks to be stabilised due to periodic performance fluctuation.

VI. Conclusion

We have presented a novel latent variable approach to formulate VM performance modelling and prediction as matrix decomposition and completion problems. This approach can be applicable to the scenario of predicting the performance of an black-box or grey-box application with limited the prior knowledge. Choosing benchmarks is in general a non-trivial problem in such scenario. This approach allows users to provide a large set of potentially relevant benchmarks as input to model VM performance for performance prediction. As an initial evaluation, We ran a prototype with 373 micro-benchmarks to predict the scores of a general-purpose benchmark, i.e. *Geekbench*, for 37 AWS EC2 VMs. The results show that our method is effective in the setting, with RMSE and STD being (6.7, 4.5) when sampling Geekbench on 5 VMs, and (10.0, 2.8) when sampling 10.

Acknowledgment

This work is a part of the ABC (Adaptive Brokerage for the Cloud project) funded by EPSRC EP/R010528/1.

References

[1] J. Scheuner and P. Leitner, "Estimating cloud application performance based on micro-benchmark profiling," in *11th IEEE CLOUD*. IEEE, 2018, pp. 90–97.
[2] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best vm across multiple public clouds: A data-driven performance modeling approach," in *SoCC*. ACM, 2017, pp. 452–465.
[3] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: efficient performance prediction for large-scale advanced analytics," in *NSDI*, 2016.
[4] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.
[5] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE TPAMI*, vol. 34, no. 4, pp. 791–804, 2011.
[6] V. Bisot, R. Serizel, S. Essid, and G. Richard, "Feature learning with matrix factorization applied to acoustic scene classification," *IEEE/ACM TASLP*, vol. 25, no. 6, pp. 1216–1229, 2017.
[7] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the β-divergence," *Neural computation*, vol. 23, no. 9, pp. 2421–2456, 2011.
[8] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis," *Neural computation*, vol. 21, no. 3, pp. 793–830, 2009.
[9] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.
[10] H. Lee, J. Yoo, and S. Choi, "Semi-supervised nonnegative matrix factorization," *IEEE Signal Processing Letters*, vol. 17, no. 1, pp. 4–7, 2009.
[11] C. Boutsidis and E. Gallopoulos, "SVD based initialization: A head start for nonnegative matrix factorization," *Pattern recognition*, vol. 41, no. 4, 2008.
[12] S. Laue, M. Mitterreiter, and J. Giesen, "Computing higher order derivatives of matrix and tensor expressions," in *NIPS*, 2018.
[13] Geekbench Webpage, "Geekbench: cross-platform benchmark." [Online]. Available: https://www.geekbench.com/
[14] Ubuntu Manpage, "stress-ng reference webpage." [Online]. Available: https://kernel.ubuntu.com/~cking/stress-ng/
[15] G. Chao, C. Mao, F. Wang, Y. Zhao, and Y. Luo, "Supervised nonnegative matrix factorization to predict ICU mortality risk," in *IEEE BIBM*, 2018.
[16] J. Leuschner, M. Schmidt, P. Fernsel, D. Lachmund, T. Boskamp, and P. Maass, "Supervised non-negative matrix factorization methods for maldi imaging applications," *Bioinformatics*, vol. 35, no. 11, pp. 1940–1947, 2019.
[17] I. Jimenez, N. Watkins, M. Sevilla, J. Lofstead, and C. Maltzahn, "Quiho: Automated performance regression testing using inferred resource utilization profiles," in *2018 ACM/SPEC ICPE*. ACM, 2018, pp. 273–284.
[18] P. B. Webpage, "PerfKit benchmarker." [Online]. Available: http://googlecloudplatform.github.io/PerfKitBenchmarker/
[19] B. Varghese, L. T. Subba, L. Thai, and A. Barker, "Container-based cloud virtual machine benchmarking," in *IC2E*. IEEE, 2016, pp. 192–201.
[20] O. Webpage, "OpenBenchmarking." [Online]. Available: https://openbenchmarking.org/
[21] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville, "Academic cloud computing research: Five pitfalls and five opportunities," in *HotCloud 14*, 2014.
[22] S. He, G. Manns, J. Saunders, W. Wang, L. Pollock, and M. L. Soffa, "A statistics-based performance testing methodology for cloud applications," in *Proceedings of the 2019 27th ACM Joint MeESEC/FSE 2019*, 2019.