Estimating time of a CAL module execution

Statistic under AI and its application to engineering sciences

Jan Bielecki

Abstract— **Abstract ...**

Keywords— **Machine learning, BalticLSC ...**

I. Introduction

Computation Application Language (CAL) is designed to write large scale application in due to perform big data processing. Each CAL program consists of modules that are separetely executed in sequence (with parallelization possibility) on virtual machines and sending the results futher to next module of an application. CAL language is a part of the system developed under the Baltic LSC[1] project.

During the data processing within the application run, each module is executed with some input data. It could be different kind of data e.g. data frame or set of images. The execution of a module is invoke on a virtual machine with limited resources (RAM, mCPUs, GPUs). The details of an input data and the execution environment resources will be used to estimate the overall application execution time and price on a concrete cluster.

Baltic Large Scale Computing (BalticLSC[2]) project is using CAL language to perform data processing tasks. Each task is an execution of a CAL application. Each module can be used in many CAL applications. We can say that a single module is a one block (commonly as a docker image) and an application is the schema of executing concrete set of blocks. Figure 1 shows the example application that consits of a few modules to perform some image processing. Some of the modules (*R*, *G* and *B processors*) can be executed in parallel.

The aim of this project is to estimate the module execution time based on input data and limited resources of the execution. The choosen approach is to create machine learning models using historical data of the module executions times.

The WCET[1] execution time is a crucial feature of a real-time systems[3] when response should be guaranteed within a specified timing constraint or system should meet the specified deadline. In this project we do not strictly focus on providing an estimation of maximum execution time (WCET). We will try to estimate the average-case execution time (ACET) which is absolutely enought for this use case requirements.

II. Data

To predict time of module execution we create a machine learning model for each module. With each execution of the module, within some application, we will get another data point to train our model.

[1]Worst-case execution time - the maximum amout of time that the execution can take.

We will use the following features (explanatory variables)[2] as an input data for the model:

1. mCPU limit - called *mili cores* - the fraction of a physical CPU used to carry out the module execution,
2. total size of an input data,
3. max element size of an input data (if the input data is a set of files type it is the biggest part of data to be processed),
4. avg size of an input element,
5. number of input data elements.

This last three features makes sense only if the input data is a set of files. Otherwise, if the data is just a single file (e.g. data frame), the features set should be reduced to just the first two elements from the list above.

Obviously, our dependent value (that we are going to estimate) is an execution time of a module. The example data frame that will be used to train and validate our models is presented in the table I.

III. Algorithms

A. Support Vector Regression

As a one of machine learning algorithms to estimate time of a CAL module execution we will use *Epsilon-Support Vector Regression*[4] (SVR) - it is based on Support Vector Machine (SVM, orginally named *support-vector networks*[5]).

We will use SVR model with a RBF kernel[6] which is the most known flexible kernel and it could project the features vectors into infinite dimensions. It uses Taylor expansion which is equivalent to use an infinite sum over the polynomial kernels. It allows to model any function that is a sum of unknown degree polynomials.

Using kernel, the resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The RBF kernel have the following form:

$$K_{RBF}(\vec{x}, \vec{x}') = \exp\left(-\gamma ||\vec{x} - \vec{x}'||\right) \text{ , where:}$$

- $||\vec{x} - \vec{x}'||$ is the squared Euclidean distance between the two vectors of features,
- γ - hyperparameter described in more details below.

[2]As a docker container that will be used to execute a module do not use SWAP memory, RAM is not colerated with execution time. In this program we will not use modules with GPU support. Summarazing, the GPUs and RAM resources limits are not taken into account for modeling.

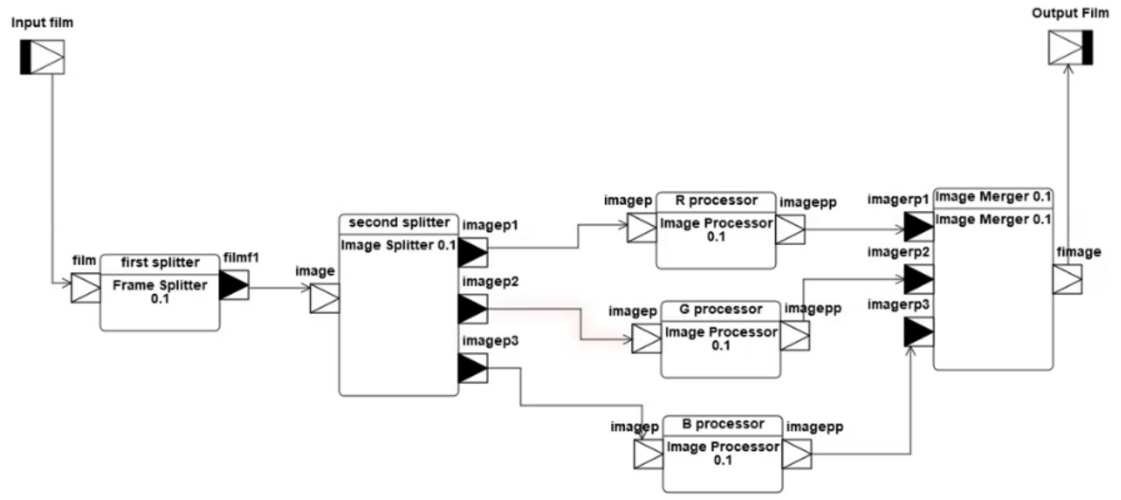


Fig. 1: The example of an image processing application written in the CAL language.

Tabla I: The example data frame for models training and validations.

Module ID	mCPU	total size [KB]	max size [KB]	average size [KB]	number of elements	time [ms]
1	1.1	12414	1341	871	21	7813
1	0.5	12414	1341	871	21	12406
1	3.6	54001	2190	891	82	9017
...	...	...	...	...	...	...

In the SVR algorithm we are looking for a hyperplane y in the following form:

$$y = \vec{w}\vec{x} + b, \text{ where:}$$

- $\vec{x}$ - vector of features,
- $\vec{w}$ - normal vector to the hyperplane y, using a kernel the $\vec{w}$ is also in the transformed space.

Training the original SVR means solving:

$$\frac{1}{2}||w||^2 + C\sum_{i}^{N}(\xi_i + \xi_i^*),$$

with the following constraints:

$$y_i - \vec{w}x_i - b \leq \epsilon + \xi_i$$

$$-y_i + \vec{w}x_i + b \leq \epsilon + \xi_i^*$$

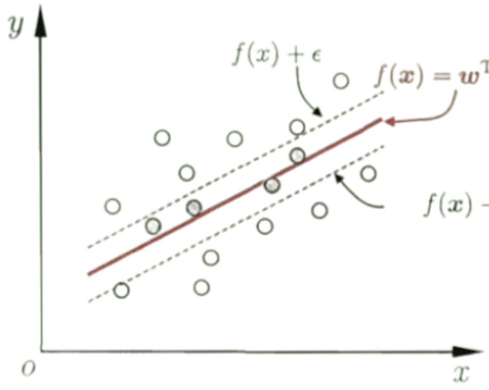
$$\xi_i\xi_i^* \geq 0$$

Figure 3 shows the wanted hyperplane with marked ξ and ϵ parameters. As we already choosed the RBF kernel for the SVR algorithm, our modelling is simplified to just find the best values of the following hyperparameters[7]:

1. C -the weight of an error cost. The regularization[3] hyperparameer, have to be strictly positive. The example from the figure use l1 penalty (the library that we use to modelling use the squared epsilon-insensitive loss with l2 penalty). The strength of the regularization is inversely proportional to C. The larger value of C the more variance is introduced into the model.
2. *epsilon* - It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. As it is shown of the figure 2 the grey data points do not provide any penalty to the loss function because they are within the allowed epsilon range around the approximation[4].
3. *gamma* - The *gamma* hyperparameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. Increasing the value of *gamma* hyperparameter causes the variance increase what is shown in the figure 4[4].

[3]Regularization is a way to give a penalty to certain models (usually overly complex ones)

Fig. 2: Visualization of the SVR's epsilon parameter [8].



You can find detailed information about C and gamma params in sklearn library documentation[10].

IV. Training and validation

Splitting data into training and test sets.

[4]The figure is based on some example data and it only shows the hyperparameter influence on model.

Fig. 3: SVR's C parameter [4] influence on model.

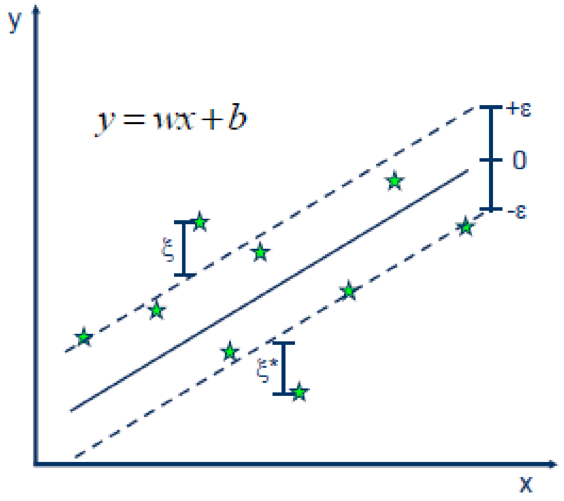
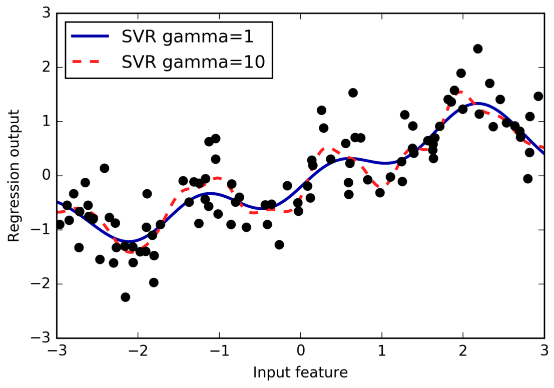


Fig. 4: SVR's gamma hyperparameter [9] influence on the model variance.



A. Training

...

B. Validation

...

V. Conclusiones

Conclusions ... `https://github.com/rasenjop/Plantilla-Jornadas-Sarteco`.

Thanks

Thanks for the help and support ...

Referencias

[1] FILL IT, *BalticLSC: main webpage of the project*, FILL IT, 14.03.2021.

[2] FILL IT, *BalticLSC: BalticLSC Admin Tool Technical Documentation, Design of the Computation Application Language, Design of the BalticLSC Computation Tool https://www.balticlsc.eu/wp-content/uploads/2020/03/05.2A_05.3A_05.4ABalticLSC_Software_Design.pdf*, FILL IT, 14.03.2021.

[3] FILL IT, *Estimation of the execution time in real-time systems*, FILL IT, 22.01.2016.

[4] Dr. Saed Sayad, *Support Vector Regression*, FILL IT, 17.01.2021.

[5] Vladimir Vapnik, *Support-Vector Networks by Corinna Cortes*, FILL IT, 1995.

[6] Wikipedia, *Radial basis function kernel*, Wikipedia, 03.03.2021.

[7] David Cournapeau et al., *Support Vector Regression*, sklearn, visited 27.12.2020.

[8] FILL IT, *Visualization of the epsilon parameter*, FILL IT, 14.03.2021.

[9] FILL IT, *Visualization of the gamma parameter*, FILL IT, 22.01.2016.

[10] David Cournapeau et al., *RBF parameters*, sklearn, since 2007.

[11] Leslie Lamport, *A Document Preparation System: LaTeX, User's Guide and Reference Manual*, Addison Wesley Publishing Company, 1986.

[12] Helmut Kopka, *LaTeX, eine Einführung*, Addison-Wesley, 1989.

[13] D.K. Knuth, *The TeXbook*, Addison-Wesley, 1989.

[14] D.E. Knuth, *The* METAFONT *book*, Addison Wesley Publishing Company, 1986.