AI in Industry

Academic Year 2022-2023

Job exit state prediction in HPC environment

Giuseppe Ruberto

Introduction

The project has been structured in this way:

- Preprocessing:
.   - Statistical Analysis
.   - Imputing missing values
- Classification

Preprocessing

Most of the object features have no data when we encounter the yearmonth $20 - 11$, from this date forward: no data. So I've decided to parse the dataset until the yearmonth $20 - 11$.
Many features have been ignored:
• Metadata
• Strange domain $-- >$ unique value
• Redundant feature $-- >$ infos already present
• Posterior $-- >$ infos that I have at runtime.
Features are encoded in a correct way and appropriate data type (initially they were object type).

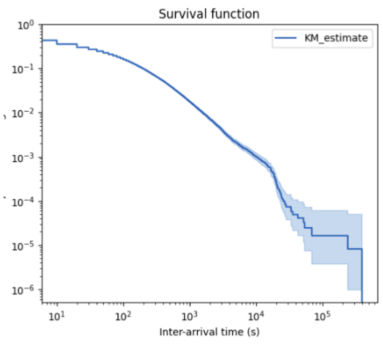
Statistical Analysis

Now in a sequential manner I am going to indicate what i have analyzed (with a small description):
- class proportion
- class proportion distribution w.r.t time granularity (month, weekday, day, hour)$- >$useful for time discretization.
- studio della variabilità per quanto riguarda la numerosità rispetto ore e giorni (per ogni non completed jobs). - cumulative distribution of the difference between submittime and starttime per each class (spot differences)
- users distribution with respect the jobs classes

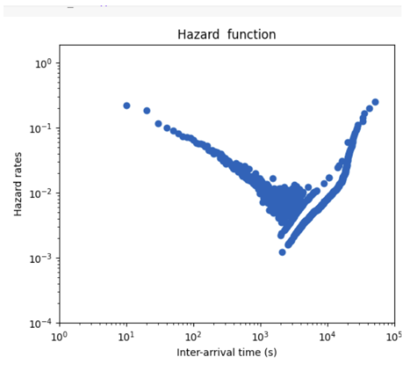
What i have figured out:
- strong user locality w.r.t non completed jobs
- we have big variability of the numerosity of failed jobs w.r.t some hours, stronger presence of outliers. - for time out jobs we can label some user as bad user (high number of timeout jobs and high proportion with respect the total jobs submitted by the same user), principal cause $- >$ lower time limit assigned - out of memory jobs have lower num memory allocation
- difference between cumulative distribution of non completed jobs (distribution of difference submit and start time) and completed... the first ones show tails, moreover we can say that the Delta values for completed jobs is in a more tight range.

At the end of my statistical analysis I've perfomed a survival analysis, the purpose is to catch peculiarities regarding failed jobs' submission inter-arrival time (for simplicity and established that not completed jobs have similar features in a certain sense, i consider timeout, out of memory and failed jobs all under the failed label).

It is structured in this way: sorting non completed jobs and using as feature (the classic feature for survival analysis:'duration') the inter-arrival time between non completed jobs.
Time unit is the passed seconds.
Estimator: Keplen Meier, a non parametric estimator.



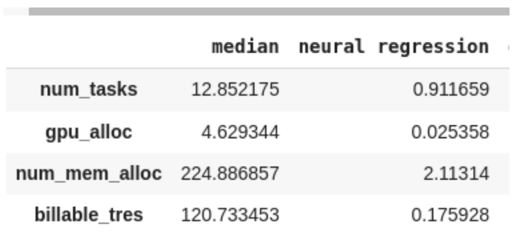
A survival question: **Which is the probability that after 100 seconds from the last failed jobs, a new failed jobs is not occured**?
We can view: approximately 25 %.
An additional insight can be provided by the hazard function (survival function is monotonically decreasing): the points is **Which is the probability of obtaining a failed jobs in an interval of time, given that is not occured before this interval**?



We can view: Hazard rate of failed jobs submission inter arrival time has a **V shape**: that means the probability of submitting a failed jobs, after the last failed jobs, decreases with the passing seconds until approximately 1000 seconds and then increase consinstently along with the time passing...thus failed jobs are either submitted continuously within short range of seconds or they are dispersi.
What i have figured out:
We know that having a not uniform temportal distribution of jobs, we need to be cautious for statement, but we can say one thing: **the vast majority of non completed jobs are submitted in a very short range of seconds by the same user**...This happens because HPC users often submit jobs in batches.

Imputing missing values

In 4 variables we have missing values -¿ num tasks, billabletres, num mem alloc, gpu allocation.
Method: using a test set to test performance (so there we have masked the values) then applying to realset (the one with missing values). Median vs Neural regression model...Neural regression model has performed so well, easy relations between independent variables and the rest of x.
This is the Mean Absolute Error in test sets:



	median	neural regression
num_tasks	12.852175	0.911659
gpu_alloc	4.629344	0.025358
num_mem_alloc	224.886857	2.11314
billable_tres	120.733453	0.175928

Classification

For the 'parameter optimization process' I will use Surrogate Bayesian Optimization process with Gaussian Process.
Metric used: F1 score macroaveraged.
Classification model used:

• **Svm**: sadly applying SVM (with kernel trick) on a 1 mil dataset is quite heavy (complexity O(n^3)). So we use Downsampling in this way: in my training set i will have a pretty balanced situation (so heavily downsampling the major class), obviously we lose infos and we can introduce some bias. Instead I m going to have a validation set and a test set that maintain the original distribution of classes $--$ > Balance consistency$-$complexity.

• **XGBoost**: class weights used and stratified splitting between sets.
• **Neural Network**: 4 settings for neural network:
- vanilla classifcation neural model
- class weights
- focal loss
- upsampling in train set

Until now the best model is XGBoost with 0.81 f1 macroaverage

I tried also a **binary classification** where i have considered completed and non completed jobs, optimized at the end using AUC Precision Recall, and always with the XGBoost I have obtained as **F1 score:0.84**.

Probabilistic Classification NN (4 classes):
Until now in my analysis, we have omitted a proper quantification of the uncertainty in the predictions. Since there can be very different reasons on why a job can fails (some jobs have equal values for the features X but different job state), we try to consider the process generating the data with a stochastic meaning. We are dealing with an aleatoric uncertainty.
Until now using categorical cross entropy we intended the conditional distribution $P(y|x, w)$ as a categorical distribution. Now in place of the softmax dense layer I will use OneHotCategorical probabilistic layer.
But considering the F1 score macroaverage metric, it works bad w.r.t the XGBoost