DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Predicting File Lifetimes With Machine Learning

Florent Monjalet Thomas Leibovici
CEA/DAM

September 24, 2019

Motivations

Hierarchical data storage:

- ***Top* tiers** (*e.g.* NVMe): high performance (latency, throughput), small capacity
- ***Bottom* tiers** (*e.g.* tape library): lower performance, big capacity
- Expensive data movements between tiers (*e.g.* tape library access latency)

Goal:

- Use the *top* tier for files used regularly
- Get unused files out of the *top* tier as soon as possible
- The sooner a file can be evicted, the more space can be used on the *top* tier
- Don't evict a file too soon

Common solutions include:

- LRU[1] policy, *e.g.* evict files that have not been accessed for 1 month
- Pattern based: some files are known not to be accessed after a given amount of time, *e.g.*:
  - `/home/user1/job1/**/*.log` files will be written for 1 day and read for maximum 1 week.
  - `/scratch/**/checkpoint-*` files will be written once and never read again
  - Tedious and ad-hoc (tied to user habits)

[1] Least Recently Used
[2] Time from creation to last read

Common solutions include:

- LRU[1] policy, *e.g.* evict files that have not been accessed for 1 month
- Pattern based: some files are known not to be accessed after a given amount of time, *e.g.*:
  - `/home/user1/job1/**/*.log` files will be written for 1 day and read for maximum 1 week.
  - `/scratch/**/checkpoint-*` files will be written once and never read again
  - Tedious and ad-hoc (tied to user habits)

Alternative approach: infer file lifetimes[2] from previously seen files

- **Random Forest Regressor**: decision tree based regressor
- **Convolutional Neural Networks** (CNN): powerful model, known to be good at automatically learning patterns

[1] Least Recently Used
[2] Time from creation to last read

Common solutions include:

- LRU[1] policy, *e.g.* evict files that have not been accessed for 1 month
- Pattern based: some files are known not to be accessed after a given amount of time, *e.g.*:
  - `/home/user1/job1/**/*.log` files will be written for 1 day and read for maximum 1 week.
  - `/scratch/**/checkpoint-*` files will be written once and never read again
  - Tedious and ad-hoc (tied to user habits)

Alternative approach: infer file lifetimes[2] from previously seen files

- **Random Forest Regressor**: decision tree based regressor
- **Convolutional Neural Networks** (CNN): powerful model, known to be good at automatically learning patterns

Advantages of this approach:

- Estimating file lifetime allows to build policies that make finer decisions on which files should be evicted and when
- *Data and user behaviours drive the policy without manual analysis*

[1] Least Recently Used

[2] Time from creation to last read

Input: a path (*e.g.* `/home/coyote/aerodyn/profiles/road_runner.npy`)

Output: the lifetime of this path (*e.g.* 20 seconds or 30 days)

Problem Specification

Input: a path (*e.g.* `/home/coyote/aerodyn/profiles/road_runner.npy`)

Output: the lifetime of this path (*e.g.* 20 seconds or 30 days)

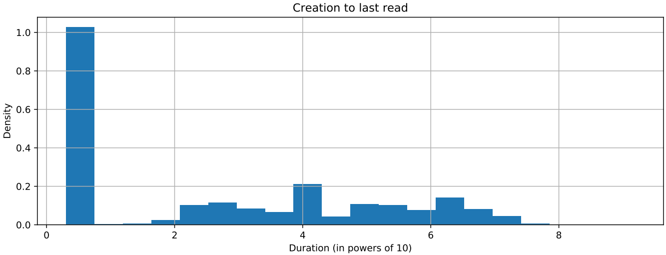
- **Lifetime** (in this presentation): duration from creation to last read
- The method is independant from the lifetime definition
- Only the training data gives meaning to the output
- For this problem, we want to avoid lifetime underestimations
  - Underestimation $\Rightarrow$ early eviction $\Rightarrow$ performance loss

$\approx$ **6,000,000 files** with:

- absolute path
- creation time
- last access time
- last modification time

Extracted from the Robinhood[3] database storing metadata of a production Lustre filesystem

[3] https://github.com/cea-hpc/robinhood



45.80% < 10 seconds

Issue: if 50% of the lifetimes are 0, always predicting low lifetimes can give the illusion of a good average accuracy.

- Need to ensure good accuracy over the whole range of values
- Error measures will have to take this into account by detailing error profile for different lifetime orders of magnitude

Dataset preprocessing: paths

Machine learning algorithm only work on constant size arrays of numbers:

- Paths do not respect this constraint
- Algorithm performance can depend on the vectorization method

Dataset preprocessing: paths

Machine learning algorithm only work on constant size arrays of numbers:

- Paths do not respect this constraint
- Algorithm performance can depend on the vectorization method

Chosen vectorization method:

- Left pad or truncate to 256 characters
- Characters are one-hot encoded so that all characters are seen as equidistant

`/bac/bc`

Padding

\0	\0	/	b	a	c	/	b	c	eof

\0	1	1	0	0	0	0	0	0	0	0
eof	0	0	0	0	0	0	0	0	0	1
/	0	0	1	0	0	0	1	0	0	0
a	0	0	0	0	1	0	0	0	0	0
b	0	0	0	1	0	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

Machine learning algorithm only work on constant size arrays of numbers:

- Paths do not respect this constraint
- Algorithm performance can depend on the vectorization method

Chosen vectorization method:

- Left pad or truncate to 256 characters
- Characters are one-hot encoded so that all characters are seen as equidistant

/bac/bc

Padding

\0	\0	/	b	a	c	/	b	c	eof

\0	1	1	0	0	0	0	0	0	0	0
eof	0	0	0	0	0	0	0	0	0	1
/	0	0	1	0	0	0	1	0	0	0
a	0	0	0	0	1	0	0	0	0	0
b	0	0	0	1	0	0	0	1	0	0
c	0	0	0	0	0	1	0	0	1	0

- Each input path is now a matrix of *path_len* $\times$ *alphabet_size* (here 256 $\times$ 106)

Durations are scaled logarithmically ($log_{10}(duration)$)

- Reflects the nature of the error that interests us
- We are interested in the *order of magnitude* of durations

Intuitively:

- $+10^3$ seconds is negligible if the duration is 10^9
- $+10^3$ seconds is huge if the duration is 10^1
- $\times 10$ is perceived as the same error for 10^9 and 10^1

Random Forest:

- Several `sklearn` regressors tested, Random Forest gave the best results
- Good computational cost / precision ratio
- 16 estimators (decision trees) gives good results
- Increasing this number does not improve performance significantly

Convolutional Neural Networks:

- Good at learning patterns
- Paths can be seen as sequences of patterns of characters
- Based on known architectures and fine tuned with experimentation:
  - Classic image recognition networks (namely the VGG family)
  - Other character level convolution networks (eXpose)

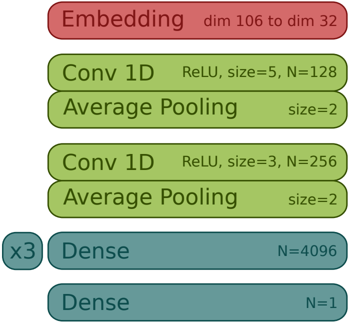


Figure 1: CNN Architecture

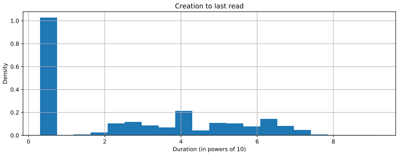
Loss: function optimized by the network, defines how the network weights are updated

Selected losses[4]:

- **logcosh:** $loss(err) = log(cosh(err))$
  - Behaves like *mean squared error* for small errors
  - But more resilient to outliers
- **quantile 99:** $loss(err) = max(0.99 \times err, -0.01 \times err)$
  - Underestimations are way more penalized than overestimations
  - Results in a lower accuracy
  - But very low underestimation rate (theoretically around 1%)

[4] With $err = truth - prediction$ since the network works on $log_{10}(duration)$

Results: Disclaimer



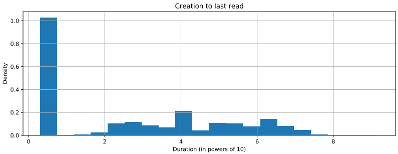
Two training situations will be demonstrated:

- 70% training set, 30% validation set
- 95% training set, 5% validation set

Training times:

- Random Forest: `sklearn`, ≈ 5 minutes training on 24 CPU
- CNN: `tensorflow`, ≈ 2h to 3h training on 4 NVidia Tesla V100-SXM2-16GB (100 epochs)

Results: Disclaimer



Two training situations will be demonstrated:

- 70% training set, 30% validation set
- 95% training set, 5% validation set

Training times:

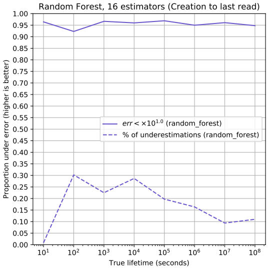
- Random Forest: `sklearn`, $\approx$ 5 minutes training on 24 CPU
- CNN: `tensorflow`, $\approx$ 2h to 3h training on 4 NVidia Tesla V100-SXM2-16GB (100 epochs)

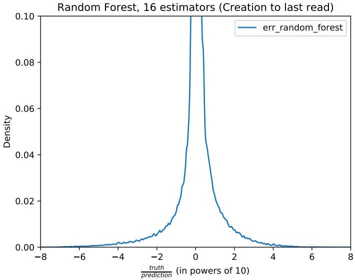
$$err = \frac{truth}{prediction}$$

Random Forest Regressor Results (70%-30%)

Result summary on the “creation to last read” dataset, 70%-30% split:

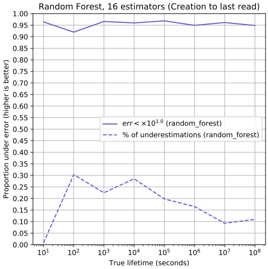
- 96.47% of estimation are less than a $\times 10$ factor away from the truth
- 86.81% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
- 10.55% underestimations (1.78% underestimations $> \times 10$)

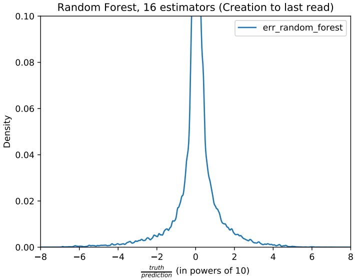




Result summary on the “creation to last read” dataset, 95%-5% split:

- ~~96.47%~~ 96.75% of estimation are less than a $\times 10$ factor away from the truth
- ~~86.81%~~ 87.21% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
- ~~10.55%~~ 9.54% underestimations (~~1.78%~~ 1.50% underestimations $> \times 10$)

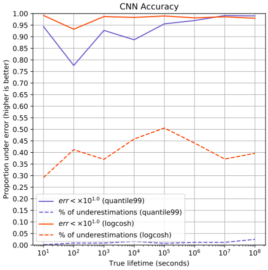


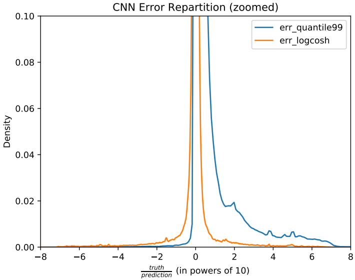


CNN Results (70%-30%)

Result summary on the “creation to last read” dataset, 70%-30% split:

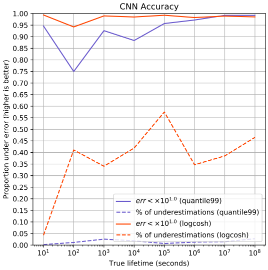
- **logcosh:**
  - 98.79% of estimation are less than a $\times 10$ factor away from the truth
  - 91.57% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
  - 36.62% underestimations (0.63% underestimations $> \times 10$)
- **quantile 99:**
  - 94.59% of estimation are less than a $\times 10$ factor away from the truth
  - 66.47% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
  - 0.68% underestimations (0.12% underestimations $> \times 10$)

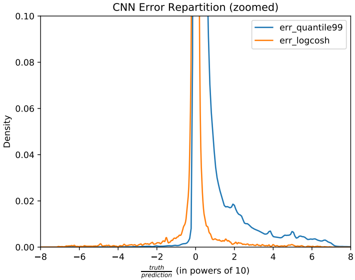




Result summary on the “creation to last read” dataset, 95%-5% split:

- **logcosh:**
  - ~~98.79%~~ 99.03% of estimation are less than a $\times 10$ factor away from the truth
  - ~~91.57%~~ 93.26% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
  - ~~36.62%~~ 24.83% underestimations (~~0.63%~~ 0.50% underestimations $> \times 10$)
- **quantile 99:**
  - ~~94.59%~~ 94.69% of estimation are less than a $\times 10$ factor away from the truth
  - ~~66.47%~~ 70.42% of estimation are less than a $\times 10^{0.1}$ factor away from the truth
  - ~~0.68%~~ 0.94% underestimations (~~0.12%~~ 0.09% underestimations $> \times 10$)





70% - 30% split	Accuracy[5]	Underest.	Underest. $< \times 10$
Random Forests	96.47%	10.55%	1.78%
CNN (*logcosh*)	**98.79%**	36.62%	0.63%
CNN (*quantile99*)	94.59%	**0.68%**	**0.12%**

95% - 5% split	Accuracy	Underest.	Underest. $< \times 10$
Random Forests	96.79%	9.64%	1.50%
CNN (*logcosh*)	**99.03%**	24.83%	0.50%
CNN (*quantile99*)	94.69%	**0.94%**	**0.09%**

Conclusion:

- Accuracy and underestimation rate seem high enough for practical applications
- Investigated prediction errors mostly are outliers and ambiguous cases
- CNN with *logcosh* is the most accurate
- CNN with *quantile99* may be the most useful in practice
- In practice, a quorum of the multiple algorithms could be used

[5] Percentage of predictions within a factor $\times 10$ from the truth

CEA/DAM

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019