Data Science and Scientific Computation
Track Core Course

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2016/17

Segment 1, Lecture 5

Overview

Date		no.	Topic
Feb 27	Mon	1	predictive models, least squares regression, model selection, regularization
Mar 1	Wed	2	real data, non-vectorial data, LASSO regression
Mar 6	Mon	3	missing data, nonlinear regression
Mar 8	Wed	4	robust regression, classification,
Mar 13	Mon	5	model evaluation, large-scale model learning
Mar 15	Wed	6	project Q&A
Mar 20	Mon	7	project Q&A
Mar 22	Wed	8	project presentations

Refresher

Summary

General form: regularized risk minimization

$$\min_f \quad \underbrace{\frac{1}{n}\sum_i \ell(\, y_i, f(x_i)\,)}_{\text{loss}} + \underbrace{\overbrace{\lambda}^{\text{reg.const.}}\ \Omega(f)}_{\text{regularizer}}$$

Find a prediction function that at the same time

- **explains the observed data well**, as measured by a loss function ℓ
  - $\ell(y, f(x)) = (f(x) - y)^2$: least squares regression
  - $\ell(y, f(x)) = |f(x) - y|$: robust regression
  - $\ell(y, f(x)) = \log(1 + e^{-yf(x)})$: logistic regression (=classification)
  - ...
- **is robust**, as measured by a regularizer Ω
  - $\Omega(f) = \|w\|^2$ for $f(x) = w^\top \phi(x)$: squared norm regularization
  - $\Omega(f) = \|w\|_{L^1}$ for $f(x) = w^\top \phi(x)$: LASSO (promotes sparsity)
  - ...

Other problems / other evaluation measures

Imbalanced data

Example

Classification problem: which pet care does a customer own?

- Training set: profile information of 8312 pet owners:
  - `dogs:` 4334, `cats:` 3611, `birds:` 367
- Observation: less than 5% of pet owners have `birds`
- Classifier might never predict `bird`, but have over 95% accuracy

What to do?

1) check if it is really a problem. if not: ignore
2) create more balanced training data
   - collect more training data of bird owners
   - reduce data of non-bird owners by random sub-sampling
   - re-weight data: bird owners count more than non-bird owners in loss

   note: this introduces a bias in the data statistics!

Other Ways to Evaluate Classifiers

Multi-class classifier $c : \mathcal{X} \to \{\texttt{dog}, \texttt{cat}, \texttt{bird}\}$

Confusion matrix, e.g.:

true \ predicted	dog	cat	bird
dog	3453	809	72
cat	561	2941	109
bird	27	21	319

$$\text{accuracy} = \frac{\text{sum of diagonal}}{\text{sum of all entries}} = 0.807 = 80.7\%$$

Alternatively, use *rates*, i.e. fraction of cases instead of total number

true \ predicted	dog	cat	bird
dog	0.415	0.097	0.009
cat	0.07	0.354	0.013
bird	0.003	0.003	0.038

$$\text{accuracy} = \text{sum of diagonal} = 0.807 = 80.7\%$$

Observation: rare classes contribute little to the overall accuracy.

Other Ways to Evaluate Classifiers

Multi-class classifier $c : \mathcal{X} \to \{\texttt{dog}, \texttt{cat}, \texttt{bird}\}$

Confusion matrix:

true \ predicted	dog	cat	bird
dog	3453	809	72
cat	561	2941	109
bird	27	21	319

$$\text{accuracy} = \frac{\text{sum of diagonal}}{\text{sum of all entries}} = 0.807 = 80.7\%$$

We can make all classes equally important: normalize each row by number of (true) cases: per-class accuracy

true \ predicted	dog	cat	bird
dog	0.797	0.187	0.016
cat	0.155	0.814	0.030
bird	0.073	0.057	0.869

$$\text{average per-class accuracy} = \text{average of diagonal} = 0.827 = 82.7\%$$

Other Ways to Evaluate Classifiers

Multi-class classifier $c : \mathcal{X} \to \{\texttt{dog}, \texttt{cat}, \texttt{bird}\}$

Confusion matrix:

true \ predicted	dog	cat	bird
dog	3489	845	0
cat	616	2995	0
bird	186	181	0

$$\text{accuracy} = \frac{\text{sum of diagonal}}{\text{sum of all entries}} = 0.780 = 78.0\%$$

If we consider all classes equally important:
normalize each row by number of cases: per-class accuracy

true \ predicted	dog	cat	bird
dog	0.805	0.195	0.000
cat	0.171	0.829	0.000
bird	0.507	0.493	0.000

$$\text{average per-class accuracy} = \text{average of diagonal} = 0.545 = 54.5\%$$

Other Ways to Evaluate Classifiers

Special case: binary classifier $c(x) = \text{sign}(w^\top x + b)$

Confusion matrix, e.g.:

true \ predicted	+1	−1
+1	53	9
−1	27	21

Other Ways to Evaluate Classifiers

Special case: binary classifier $c(x) = \text{sign}(w^\top x + b)$

Confusion matrix, e.g.:

true \ predicted	+1	−1
+1	53	9
−1	27	21

true \ predicted	+1	−1
+1	true positives (TP)	false negatives (FN)
−1	false positives (FP)	true negatives (TN)

Other Ways to Evaluate Classifiers

Special case: binary classifier $c(x) = \text{sign}(w^\top x + b)$

Confusion matrix, e.g.:

true \ predicted	+1	−1
+1	53	9
−1	27	21

true \ predicted	+1	−1
+1	true positives (TP)	false negatives (FN)
−1	false positives (FP)	true negatives (TN)

Observation:

- increasing the value of b can move cases from right column to left
  - FNs can become TPs, but TNs can become FPs
- decreasing the value of b has opposite effect: left column to right

Other Ways to Evaluate Classifiers

Special case: binary classifier $c(x) = \text{sign}(w^\top x + b)$

Confusion matrix, e.g.:

true \ predicted	+1	−1
+1	53	9
−1	27	21

true \ predicted	+1	−1
+1	true positives (TP)	false negatives (FN)
−1	false positives (FP)	true negatives (TN)

Observation:

- increasing the value of b can move cases from right column to left
  - FNs can become TPs, but TNs can become FPs
- decreasing the value of b has opposite effect: left column to right

Alternatively, use *rates*:

true \ pred	+1	−1
+1	$\text{TPR} = \frac{\text{TP}}{\text{TP+FN}}$	$\text{FNR} = \frac{\text{FN}}{\text{TP+FN}}$
−1	$\text{FPR} = \frac{\text{FP}}{\text{FP+TN}}$	$\text{TNR} = \frac{\text{TN}}{\text{FP+TN}}$

TPR and FPR grow monotonically with b,

Other Ways to Evaluate Classifiers: ROC curve

Binary classifier $c(x) = \text{sign}(w^\top x + b)$

Confusion matrix:

true \ predicted	yes	no
yes	0.43	0.09
no	0.27	0.21

acc. $= 64\%$

Adjust threshold a little: $c'(x) = \text{sign}(w^\top x + b'')$ for $\tilde{b}'' = b + 0.1$

Confusion matrix:

true \ predicted	yes	no
yes	0.47	0.05
no	0.28	0.20

acc. $= 67\%$

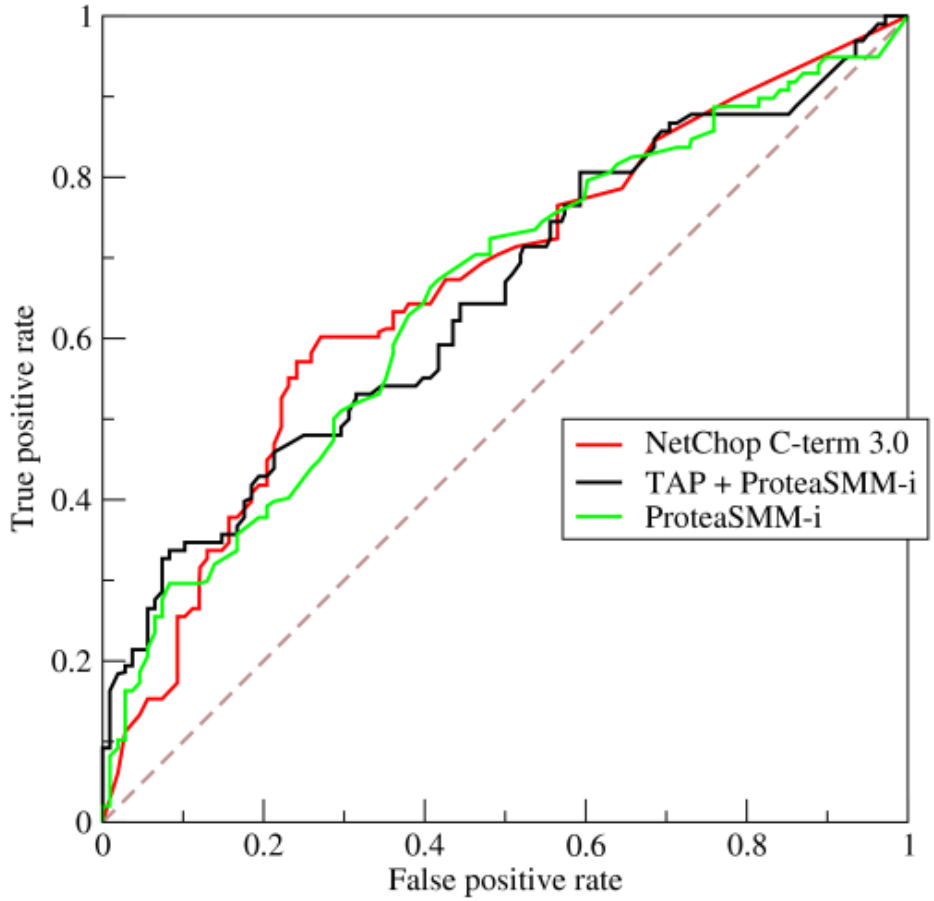
Adjust threshold a little: $c''(x) = \text{sign}(w^\top x + b')$ for $b' = b - 0.1$

Confusion matrix:

true \ predicted	yes	no
yes	0.42	0.10
no	0.17	0.31

acc. $= 73\%$

ROC-curve plots classifier characteristics across all thresholds/offsets:



- higher is better; best possible: $TP = 1, FP = 0$
- curves are monotonically increasing; diagonal is chance level
- if curves of different classifiers intersect: no clear winner
- if pressed for single quality value: **area-under-curve** (AUC)

Image: By BOR, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10714489

Other Ways to Evaluate Classifiers

Example

Binary classification problem: automatic test for BSE

- collect training data: 100000 cow brains
- number of BSE cases: 8

Accuracy of constant classifier, $f(x) = \texttt{"not BSE"}$: over 99.99%

But accuracy is not really what we care about. Primarily we want to

- reliably identify all `BSE` cases: $\rightarrow$ high "recall"

Secondarily, we want to

- return as few as possible `non-BSE` cases as `BSE`: $\rightarrow$ high "precision"

Other Ways to Evaluate Classifiers

Example

Binary classification problem: automatic test for BSE

- collect training data: 100000 cow brains
- number of BSE cases: 8

Accuracy of constant classifier, $f(x) = \texttt{"not BSE"}$: over 99.99%

But accuracy is not really what we care about. Primarily we want to

- reliably identify all `BSE` cases: $\rightarrow$ high "recall"

Secondarily, we want to

- return as few as possible `non-BSE` cases as `BSE`: $\rightarrow$ high "precision"

Many similar situation:

- credit card fraud, airport security checks, web search, ...

Other Ways to Evaluate Classifiers

Retrieval tasks: measure quality in terms of *precision* and *recall*:

Definition (Precision, Recall, F1-Score)

For a prediction function $c : \mathcal{X} \to \{\pm 1\}$ ("relevant"/"not relevant") and test set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \{\pm 1\}$.

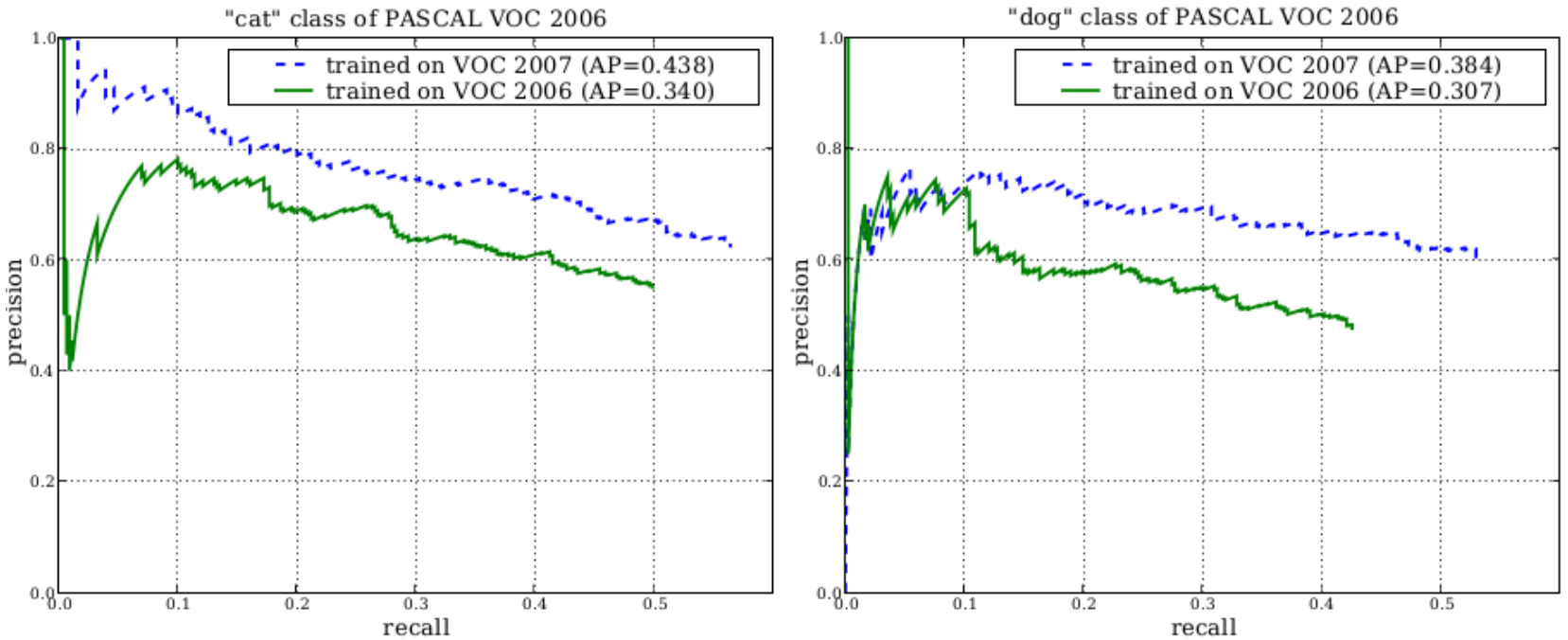
$$\textit{precision}(g) = \frac{\textit{number of samples with } c(x_j) = 1 \textit{ and } y_j = 1}{\textit{number of samples with } c(x_j) = 1} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FP}}$$

$$\textit{recall}(g) = \frac{\textit{number of samples with } c(x_j) = 1 \textit{ and } y_j = 1}{\textit{number of samples with } y_j = 1} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FN}}$$

$$\textit{F1-score}(g) = 2\frac{\textit{precision}(g) \cdot \textit{recall}(g)}{\textit{precision}(g) + \textit{recall}(g)}$$

Other Ways to Evaluate Classifiers: precision-recall curves

Precision-recall curve: retrieval characteristics across all thresholds



- curves are not strictly monotonic, but generally decreasing
- higher is better; best possible: precision $\equiv 1$ for all recall levels
- if curves of different classifiers intersect: no clear winner
- if pressed for single quality value: **average precision (AP)** (=AUC)

Large-Scale Model Learning

What is Large-Scale?

What is Large-Scale?

Working definition I

If the data does not fit into the working memory of your computer anymore.

- prediction problems at Google
- prediction problems at CERN
- many tasks in Bioinformatics
- many tasks involving neurophysiological recordings
- none of the problem we looked at so far

What is Large-Scale?

Working definition I

If the data does not fit into the working memory of your computer anymore.

- prediction problems at Google
- prediction problems at CERN
- many tasks in Bioinformatics
- many tasks involving neurophysiological recordings
- none of the problem we looked at so far

Working definition II

If the quality of the model you can learn is limited by the computational resources, not by the amount of data you have available.

- few prediction tasks with *linear* models (except Google/Facebook)
- many non-parametric or kernelized models
  for example, nonlinear digit classification with 60000 examples

Learning Models in Parallelism

If one computer is not fast enough, using multiple in parallel might help:

- 1) multi-core / shared memory architectures
  e.g., your laptop, your smartphone, our workstation
- 2) distributed computing / compute clusters
  e.g. Google BORG, or IST scientific computing clustering

But: don't start to optimize for speed too early, and beware of

Amdahl's law

The maximal speedup one can achieve by parallelizing a program is

$$\text{maximal speedup} = \frac{1}{1 - p + \frac{p}{s}}$$

where

- p is the percentage of executation time that can be parallelized
- s is the speedup one can achieve from its parallelization

For example: if 50% of a program can be parallelized, it's impossible to make the program more than $2\times$ faster.

Multi-Core / Shared Memory Parallelism

In the easiest case, you don't even notice.

- many system libraries automatically process data in parallel when run on multi-core CPUstim, e.g. matrix multiplications

```
from numpy.random import randn
X=randn(1000,100000)
X.dot(X.T)
```

Windows: system monitor?

Linux/MacOS:

- 'top' or 'htop' command show current CPU utilization
- 'time' shows how much runtime time a command uses

jupyter/ipython: '%time' measures how long a command takes

- wall time or real time: elapsed physical time
- user time or CPU time: performed CPU computations

If K CPUs are used, *user* time can be K times larger than *wall* time.

Otherwise, you can encourage/enforce parallel execution yourself using, e.g., `OpenMP`, `tbb` or other multi-threading libraries.

OpenMP:

```
double res[MAX];
int i;

#pragma omp parallel for
for(i=0; i<MAX; i++) {
  res[i] = some_slow_subroutine();
}
```

---

but be careful if function calls have side effects.

Generally, parallelization has overhead:

- starting a thread, shutting down a thread, ...

We can keep overhead small by parallelizing not more than necessary, and on an as-coarse-as-possible level (outer loops, not inner loops):

`double K[N,M];`
`int i,j;`
 	
`for (i=0...N) {`
`  parallel_for (j=0...M) {`
`  K[i,j] = kernel(i,j);`
`  }`
`}`
	`double K[N,M];`
	`int i,j;`
	
	`parallel_for (i=0...N) {`
	`  for (j=0...M) {`
	`  K[i,j] = kernel(i,j);`
	`  }`
	`}`
		`double K[N,M];`
		`int i,j;`
		
		`parallel_for (i=0...N) {`
		`  parallel_for (j=0...M) {`
		`  K[i,j] = kernel(i,j);`
		`  }`
		`}`

fine-grained	coarse-grained	too much parallelism
N times M threads 1 op. per thread	N threads M ops. per thread	NM threads 1 op. per thread

Shared Memory Parallelism

When learning predictive models, we often call the same programs/routine with different parameters, e.g. model selection.

This is perfect for parallelization! When running a program multiple times with different arguments, no information have to be exchanged and the danger of side effects is minimal.

"embarassingly parallel"

Running programs in parallel

We can run programs in parallel from the commandline

```
$ python ./myscript.py &
```

Returns immediately to the prompt. Script runs in the background.

`'screen'` allows sending a console to the background and accessing it again later, even if one has logged out inbetween and logged in again from another computer.

- `'screen'` opens a virtual console. Option `-L` logs all outputs
- `CTRL-a CTRL-d` sends virtual console to the background, programs keep running
- `'screen -r'` brings a virtual console back to the foreground
- multiple `screen` sessions can run in parallel

Similar functionality: `'tmux'`

Running programs in parallel

`parallel` (not always installed) runs multiple jobs in parallel while making sure that only a certain numbers of them are active at each time.

```
$ parallel -j3 python ./myscript.py  ::: 1 2 3 4 5
```

executes

```
./myscript.py 1
./myscript.py 2
./myscript.py 3
./myscript.py 4
./myscript.py 5
```

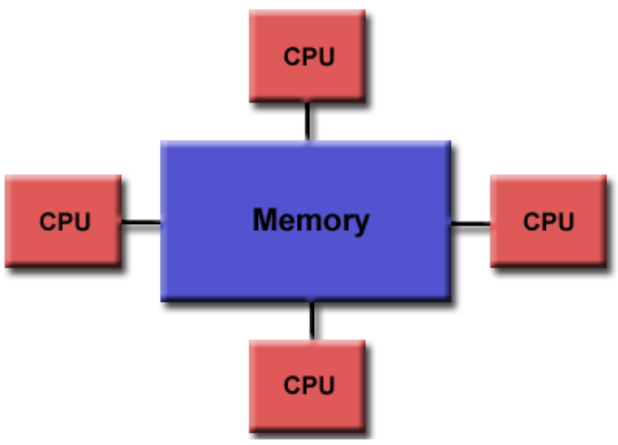
but not more than 3 at a time ('1', '2' and '3' are started in parallel, but '4' is only started once '1' has finished, etc.)

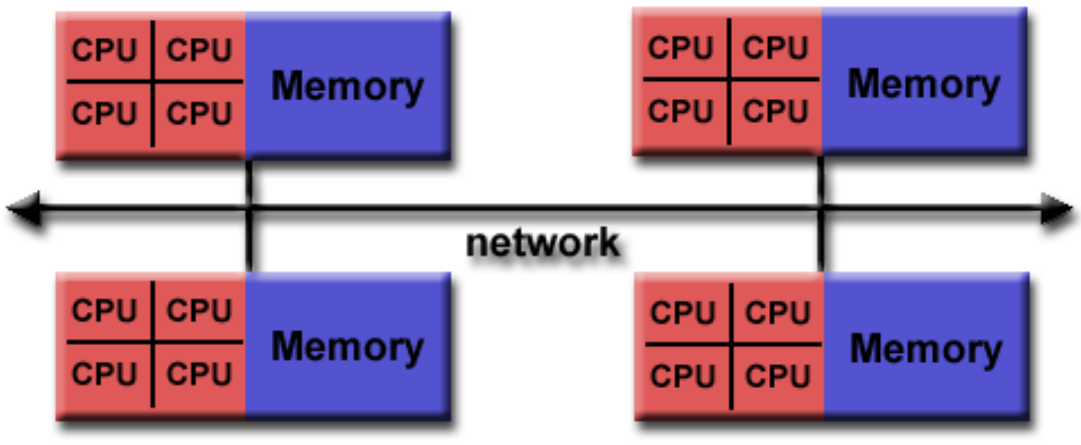
Very useful when one has to run more jobs than cores, or keep some capacity free for interacive work

Distributed Computing

Main difference between shared-memory and distributed is the speed by which information can be exchanged:

- shared-memory: via cache (very fast) or main memory (fast)
- distributed: via network (slow) or disk (very slow)





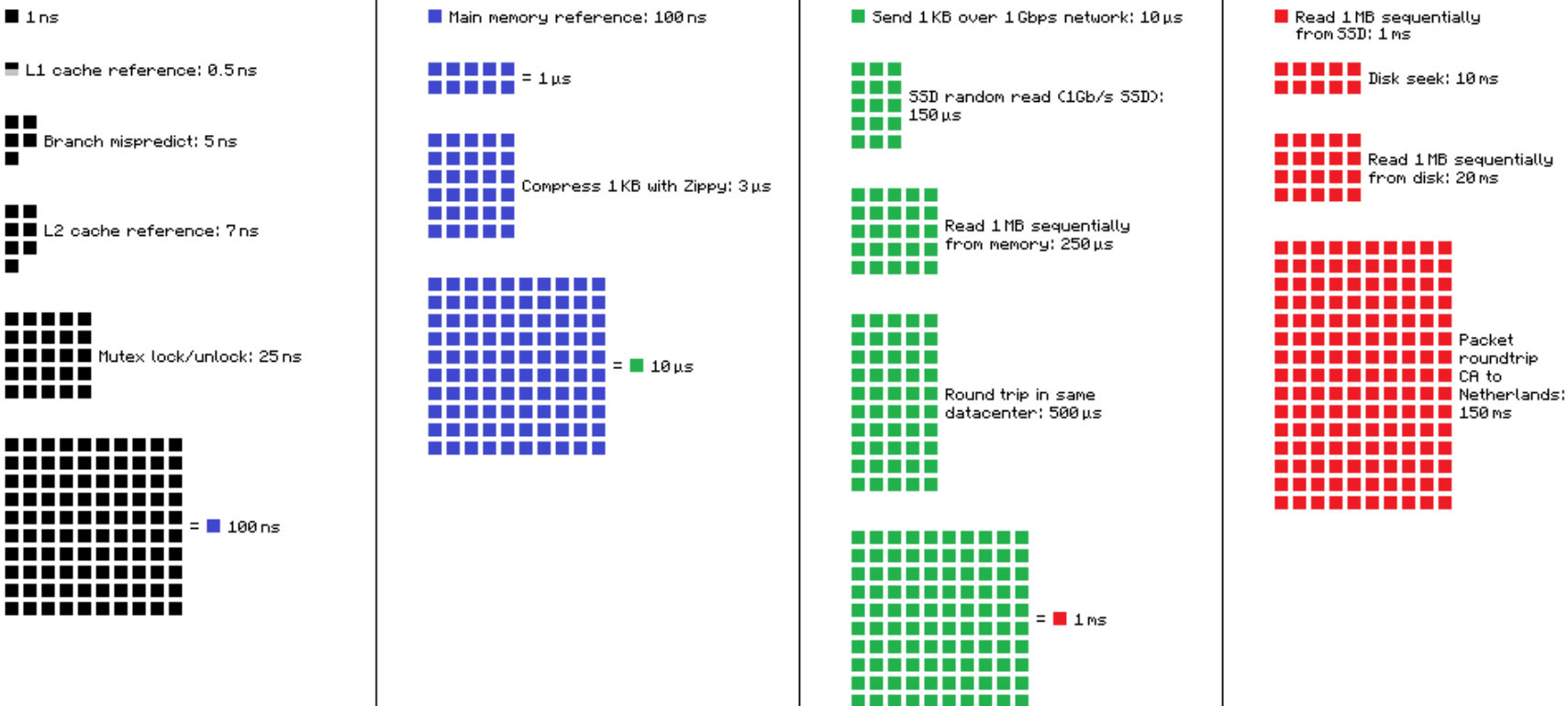
Images: http://shodor.org/ bplist/bwi/openmp.html

```
Latency Comparison Numbers
--------------------------
Arithmetic operation (e.g. multiplication)   0.5 ns
L1 cache reference                           0.5 ns
Branch mispredict                              5   ns
L2 cache reference                             7   ns
Mutex lock/unlock                             25   ns
Main memory reference                        100   ns
Compress 1K bytes with Zippy               3,000   ns        3 us
Send 1K bytes over 1 Gbps network         10,000   ns       10 us
Read 4K randomly from SSD*               150,000   ns      150 us
Read 1 MB sequentially from memory       250,000   ns      250 us
Round trip within same datacenter        500,000   ns      500 us
Read 1 MB sequentially from SSD*       1,000,000   ns    1,000 us    1 ms
(Spinning) disk seek                  10,000,000   ns   10,000 us   10 ms
Read 1 MB sequentially from disk      20,000,000   ns   20,000 us   20 ms
Send packet Austria->USA->Austria    150,000,000   ns  150,000 us  150 ms

Notes
-----
1 ns = 10^-9 seconds
1 us = 10^-6 seconds = 1,000 ns
1 ms = 10^-3 seconds = 1,000 us = 1,000,000 ns
```

Source: https://gist.github.com/jboner/2841832

Latency Numbers Every Programmer Should Know



Source: https://gist.github.com/2841832

Scientific Compute Cluster

IST has a compute cluster that is particularly easy to use for embarassingly parallel jobs.

- approx. 75 'compute nodes'
- approx. 1200 CPU cores, 4800 GB RAM (64 per node, 4 per core)
- approx 30TB storage, shared between all nodes
- parts reserved for special purpose, e.g. long-running fluid simuations
- most nodes can be accessed via *SUN Grid Engine (sge)*

Running principle (idealized):

- on the cluster, there is 1 *queue* for all jobs to be run
- whenever there is space on any node, the job highest ranked job(s) are removed from the queue and started on the free node
- when new jobs are submitted to the queue, they start at the bottom
- over time, jobs rise higher and higher in the queue until they start
- often: queue (almost) empty, jobs start almost immediately

Cluster Commands

- `ssh bjoern22` log into cluster
- `qstat` show your submitted jobs (running or waiting)
- `qstat -u \*` show submitted jobs of all users (long)
- `qstat -j 123` show information about job with ID 123
- `qsub ./myscript.sh` submit script to be executed
- `qdel 456` stop/delete job with ID 456

```
$ ssh nick01
$ for I in `seq 1 100`; do qsub ./myscript.sh $I ; done
```

Runs 100 instances of `myscript.sh` with arguments `1,2,3,...,100`.

How to get *outputs* from jobs? Write to disk with filename derived from argument and copy to local computer with `scp` (not elegant, but works).

Cluster Hygiene

The cluster shared amongst many users. One user who doesn't follow the rules can cause harm to many others. Don't be egoistic or lazy!

Cluster Hygiene

The cluster shared amongst many users. One user who doesn't follow the rules can cause harm to many others. Don't be egoistic or lazy!

- **Don't run jobs that take very long (e.g. longer than a week).** If you do have long-running computations, implement a functionality that stores intermediate results, allows terminating the jobs and restarts from where it was intererrupted when restarted.

Cluster Hygiene

The cluster shared amongst many users. One user who doesn't follow the rules can cause harm to many others. Don't be egoistic or lazy!

- **Don't run jobs that take very long (e.g. longer than a week).** If you do have long-running computations, implement a functionality that stores intermediate results, allows terminating the jobs and restarts from where it was intererrupted when restarted.
- **Don't use multi-threaded code** on the cluster, except if you explicitly reserve multiple cores. On *overbooked* nodes all jobs will slow down. Beware of multi-threaded libraries!

Cluster Hygiene

The cluster shared amongst many users. One user who doesn't follow the rules can cause harm to many others. Don't be egoistic or lazy!

- **Don't run jobs that take very long (e.g. longer than a week).** If you do have long-running computations, implement a functionality that stores intermediate results, allows terminating the jobs and restarts from where it was interrerrupted when restarted.
- **Don't use multi-threaded code** on the cluster, except if you explicitly reserve multiple cores. On *overbooked* nodes all jobs will slow down. Beware of multi-threaded libraries!
- **Specify how much memory a job will need** at most, so the scheduler software knows that the remaining memory is available for others. Jobs that needs little RAM can squeeze into nodes that don't have space for jobs with large RAM demand!

Cluster Hygiene

The cluster shared amongst many users. One user who doesn't follow the rules can cause harm to many others. Don't be egoistic or lazy!

- **Don't run jobs that take very long (e.g. longer than a week).** If you do have long-running computations, implement a functionality that stores intermediate results, allows terminating the jobs and restarts from where it was intererrupted when restarted.
- **Don't use multi-threaded code** on the cluster, except if you explicitly reserve multiple cores. On *overbooked* nodes all jobs will slow down. Beware of multi-threaded libraries!
- **Specify how much memory a job will need** at most, so the scheduler software knows that the remaining memory is available for others. Jobs that needs little RAM can squeeze into nodes that don't have space for jobs with large RAM demand!
- **If you use temporary files, delete them after use**. If the cluster filesystem runs full, everybody else's jobs can't write to disk anymore and might crash.

About Software Licences

Many commercial packages (Mathematica, Matlab, . . .) require licenses:

- *personal licenses*: tied to a specific user/computer
- *floating licenses*: a number of users on campus (or VPN) at any time
- *campus licenses*: arbitrary number of users on campus (or VPN)

Note: many of IST's licenses are *floating*

- users typically prefer *personal* or *campus* licenses
- *floating* licenses are typically more cost effective

Best practice:

- log out of the Matlab/Mathematica user interface, when not using it
- **compile Matlab code** before using it in a "production mode"
- for Matlab, use only *toolboxes* that you really need

What, if you try to use Matlab/Mathematica and no license is available?

- Contact IT. Do not download and install a pirated copy!

What if the problem is not runtime, but RAM?

Data Dimension Too High

- perform dimensionality reduction, e.g. Principal Component Analysis
- covariance matrix can be computed incrementally (if it fits)
- if covariance matrix does not fit into RAM, split features into blocks and select features from each block separately (approximate)

Too Many Examples

- use algorithm that allows *streaming* data access
- use only a subsample of examples (random or from clustering)
- build an ensemble: train several models on different subsets of the data and combine their outputs:
  - regression: by averaging
  - classification: majority votes

  (learning an ensemble is usually 'embarassingly parallel'!)


That's all Folks!