INFO-H-515: BIG DATA ANALYTICS

Streaming analytics

Gianluca Bontempi

Machine Learning Group
Boulevard de Triomphe - CP 212
http://mlg.ulb.ac.be

HUMAN AS DATA STREAM PROCESSOR

- Human beings perceive each instant of their life through an array of sensory observations (visual, aural, nervous, etc).
- However, over the course of their life, they manage to abstract and store (in memory) only part of the observations,
- Humans may function adequately even if they **can not recall every detail** of each instant of their lives.

MAJOR TRENDS

1. Massive data streams (banking and credit transactions, satellite measurements, astronomical surveys, internet, sensor networks) due to real-time measurement technologies (e.g. wireless sensor networks, industry 4.0, IoT).
2. Streaming nature of data: impossibility of storing all the data or of processing a sample more than once (**one-pass only**).
3. Complex analytics: detecting outliers, extreme events, monitor complex correlations, track trends, pattern recognition (classification, forecasting)
4. Advances in hardware: parallel and distributed architectures, with multicore and cloud computing platforms providing access to hundreds or thousands of processors.

CHALLENGES

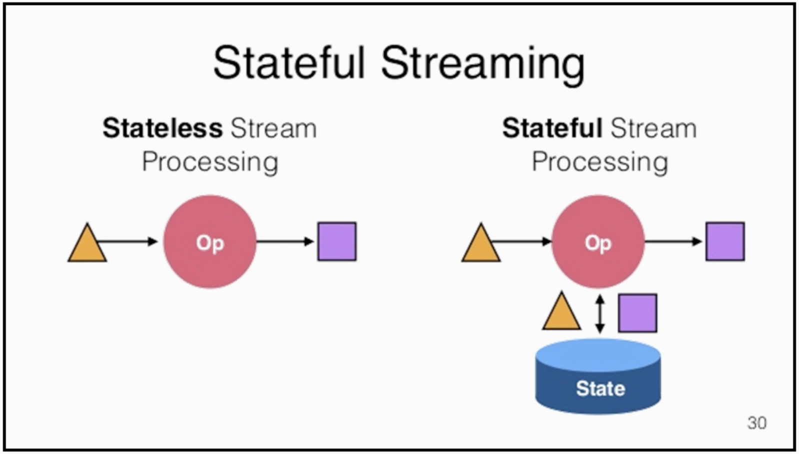
- Multiple passes on data are no more possible. One can process a data item **at most once**.
- Traditional approach of collecting, organising, storing, and analysing data is inadequate for applications where the reaction to events must be immediate.
- Asynchronous and fast arrival of stream elements (rate of arrival not under control).
- **Non stationarity**: data distribution may evolve with time.
- **Large dimensionality and distributed** nature of streams (sensor networks).
- Ad-hoc queries: arbitrary and **nonstandard** query about the stream.

STRATEGIES FOR STREAMING ANALYTICS

Possible strategies to move from batch to streaming analytics

- From exact solution to **approximation** (e.g. use of randomized algorithms): summarization techniques, e.g. sampling, windowing or budgeted storage.
- Iterative (online) versions of batch learning algorithms: from parallelization to **sequentialization** (e.g. least-squares).
- State estimation techniques: data are not meaningful by itself but as observations of some important hidden **state**.
- **Hierarchical** analysis: simple and faster analysis at low level (more data) and more sophisticated analysis at higher levels (few data), lambda architecture.

STATEFUL STREAM PROCESSING



ONLINE LEARNING

ONLINE LEARNING

- Goal: sequence of accurate predictions based on **error correction** and additional available information.
- Both supervised (including feature selection) and unsupervised learning tasks.
- Supervised learning; both for parametric identification (online learning of parameters) and/or structural identification (online learning of model structure).

ONLINE LEARNING

- It differentiates from batch learning since the training samples are treated in a sequential manner (with or without repetition): **one-pass learning**.
- First effective approach used to train neural networks (Hebb, 1949, Rosenblatt, 1957): it revives thanks to the big data setting and large scale tasks.
- During the early days it was mainly based on heuristics to deal with convergence issues. More recent theoretical analysis led to a deeper understanding of the algorithms.
- Note that it is the sequential treatment and the one-pass availability of the training set which determines the batch/online nature and not the iterative nature of the algorithm.

ONLINE VS BATCH LEARNING

- Online learning is typically simpler, easier to grasp since obtained as a **function of the latest estimate (state) and the current observation**.
- Batch learning typically faster for small datasets but more inefficient for large networks and large training sets
- Batch learning more prone to local minima
- Online learning more natural solution for dealing with **nonstationarity**.
- Online learning more sensitive to the choice of training parameters (e.g. learning rate)

STREAMING PARAMETER IDENTIFICATION

INCREMENTAL CALCULATIONS OF SAMPLE MEAN

- Sample average batch: $\mu_{(N)} = \frac{1}{N} \sum_{i=1}^{N} z_i$
- Since

$$\mu_{(N)} = \frac{\sum_{i=1}^{N-1} z_i + z_N}{N} = \frac{\sum_{i=1}^{N-1} z_i}{N-1} \frac{N-1}{N} + \frac{z_N}{N} = \frac{(N-1)\mu_{(N-1)}}{N} + \frac{z_N}{N}$$

- sequential (error correction) formulations:

$$\mu_{(N)} = \mu_{(N-1)} + \frac{1}{N}(z_N - \mu_{(N-1)}) = \mu_{(N-1)} + \alpha_{(N-1)}(z_N - \mu_{(N-1)})$$

- Note that $\alpha_{(N)} = 1/N \underset{N \to \infty}{\to} 0$

INCREMENTAL CALCULATIONS OF SAMPLE VARIANCE

- Sample variance:
  - batch formulation:
    $\sigma^2_{(N)} = \frac{1}{N}\sum_{i=1}^{N}(z_i - \mu_{(N)})^2 = \frac{S_{(N)}}{N} = \left(\frac{1}{N}\sum_{i=1}^{N} z_i^2\right) - \left(\frac{1}{N}\sum_{i=1}^{N} z_i\right)^2$
  - sequential formulation: $\sigma^2_{(N)} = S_{(N)}/N$ where

$$S_{(N)} = S_{(N-1)} + (z_N - \mu_{(N-1)})(z_N - \mu_{(N)})$$

    or

$$S_{(N)} = S_{(N-1)} + N(N-1)(\mu_{(N)} - \mu_{(N-1)})^2$$

- Recursive formulas for weighted means and variances exists too (look at T. Finch article in the course web page)

PARAMETER IDENTIFICATION LEARNING

- Core step in supervised learning.
- Solution of a **multivariate nonlinear optimization problem**.
- Cost function (to minimize) is training error

$$J_N(\theta) = \frac{\sum_{i=1}^{N}(y_i - h(x_i, \theta))^2}{N}$$

- Outcome: set of tuned parameters (e.g. neural network weights)

$$\hat{\theta} = \arg\min_{\theta} J_N(\theta)$$

- Batch learning: optimization is carried out with respect to the entire training set simultaneously
- Closed form solutions vs iterative solutions

ONLINE LINEAR LEARNING

BATCH LINEAR LEAST-SQUARES

The **least-squares estimator** $\hat{\boldsymbol{\beta}}$ minimizes the cost function

$$\hat{\beta} = \arg\min_{b} \sum_{i=1}^{N} (y_i - x_i^T b)^2 = \arg\min_{b} \left((Y - Xb)^T (Y - Xb)\right)$$

In the linear case it exists a closed form solution

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

where the $X^T X$ matrix is a symmetric $[p \times p]$ matrix.

In the MR class, we saw how to distribute the computation of the closed form solution. In what follow, we see how to solve it in a streaming fashion.

RECURSIVE LINEAR LEAST-SQUARES

Let us consider a multiple linear regression problem and a sequence of input-output samples x_i, y_i where $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$

$$\begin{cases} V_{(t)} &= V_{(t-1)} - \frac{V_{(t-1)} x_t^T x_t V_{(t-1)}}{1 + x_t V_{(t-1)} x_t^T} \\ \alpha_{(t)} &= V_{(t)} x_t^T \\ e &= y_t - x_t \hat{\beta}_{(t-1)} \\ \hat{\beta}_{(t)} &= \hat{\beta}_{(t-1)} + \alpha_{(t)} e \end{cases}$$

where $\hat{\beta}_{(t)}$ is the estimate after t observations.
Typical initialization

- $\hat{\beta}_{(0)}$ is usually put equal to a zero vector.
- $V_{(0)} = aI$, with $a > 0$ and I identity matrix since $V_{(0)}$ represents the variance of the estimator. By setting a equal to a large number the RLS algorithm will diverge very rapidly from the initialization $\hat{\beta}_{(0)}$.

RLS WITH FORGETTING FACTOR

Consider two situations

- the phenomenon underlying the data is linear but non stationary
- the phenomenon underlying the data is stationary and nonlinear but can be approximated by a linear model locally in time.

Solution: assign higher weights to more recent data (and forget older data).

Linear case: RLS techniques with forgetting factor $\nu < 1$.

RLS WITH FORGETTING FACTOR (II)

$$\begin{cases} V_{(t)} &= \frac{1}{\nu}\left(V_{(t-1)} - \frac{V_{(t-1)}x_t^T x_t V_{(t-1)}}{1+x_t V_{(t-1)} x_t^T}\right) \\ \alpha_{(t)} &= V_{(t)} x_t^T \\ e &= y_t - x_t \hat{\beta}_{(t-1)} \\ \hat{\beta}_{(t)} &= \hat{\beta}_{(t-1)} + \alpha_{(t)} e \end{cases}$$

- The smaller ν, the higher the forgetting.
- Note that for $\nu = 1$ we go back to the conventional RLS formulation.

ONLINE NONLINEAR LEARNING

BATCH PARAMETRIC IDENTIFICATION IN REGRESSION

Generic **nonlinear batch learner** with no closed form minimizer of the cost function $J_N(\theta)$.

Let

$$\hat{\theta}_N = \hat{\theta}(D_N) = \arg\min_{\theta \in \Lambda} \widehat{\text{MISE}}_{\text{emp}}(\theta) = \arg\min_{\theta \in \Lambda} J_N(\theta)$$

be the set of parameters which minimize the **empirical risk or training error**

$$J_N(\theta) = \frac{\sum_{i=1}^{N} L(z_i, \theta)}{N} = \frac{\sum_{i=1}^{N} (y_i - h(x_i, \theta))^2}{N}$$

ITERATIVE ALGORITHMS: FIRST ORDER APPROXIMATION

- **Iterative** algorithm builds a sequence $\hat{\theta}_{(t)}$ such that $J_N(\hat{\theta}_{(t)}) \leq J_N(\hat{\theta}_{(t-1)})$
- Let us consider the iteration

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} A \nabla_\theta J_N(\hat{\theta}_{(t-1)})$$

  where $\nabla_\theta J_N(\hat{\theta})$ is the gradient of the cost function, $\alpha_{(t)} > 0$ is the **learning rate**, and A is a **positive definite** matrix
- When the recursion converges, the gradient will be null so it will eventually converge to a minimum of the cost function (possibly a local minimum)
- If $A = I_n$ the recursion is the Steepest Descent algorithm
- N.B: online $\Rightarrow$ iterative but iterative $\not\Rightarrow$ online !

BATCH GRADIENT DESCENT ALGORITHM

Since $J_N(\theta) = \frac{\sum_{i=1}^{N} L(z_i,\theta)}{N}$, for $A = I$

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} \nabla_\theta J_N(\hat{\theta}) = \hat{\theta}_{(t-1)} - \alpha_{(t)} \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L(z_i, \hat{\theta}_{(t-1)})$$

- computation at each step of the average of the gradients of the loss function over the entire training set.
- if the learning rate (user designed unlike RLS) is small enough, the algorithm converges towards a local minimum of $J_N(\cdot)$
- Convergence speed-up is achieved by replacing the learning rate by a suitable definite positive matrix
- Note that this could be easily distributed in a Map Reduce form because of the summation form.

ONLINE GRADIENT DESCENT ALGORITHM

This is a simple modification of the batch version obtained by **dropping the averaging operation** and taking at random a sample $z_t \in D_N$

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} \nabla_\theta L(z_t, \hat{\theta}_{(t-1)})$$

- Assumption: estimation error due to **replacing the average with a single term** does not perturb the average behaviour.
- adaptive online setting where **no training set needs to be stored** and observations are processed immediately to improve performance

LEARNING RATE

- Very difficult to set
- if too small: update will be very slow and it will take very long time to achieve an acceptable loss
- if too large: the parameter will move all over the search space and may never achieve acceptable loss at all.
- High-dimensional non-convex tasks could lead to different sensitivity on each dimension: too small in some dimension and could be too large in another dimension.
- Hard to set a-priori different learning rates

ADA GRAD

- AdaGrad adaptively scales the learning rate for each dimension

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} g_{(t)} \odot \frac{1}{\sqrt{\sum_{\tau=1}^{t} g_{(\tau)}^2 + \epsilon}}$$

  where $\odot$ denotes the component-wise product and $g_{(t)} = \nabla_\theta L(z_t, \hat{\theta}_{(t-1)})$
- Better for sparse data since it decreases the learning rate faster for frequently changed parameters, and slower for parameters infrequently changed.
- Drawback: effective learning rate decreasing too fast because of the accumulation of the gradients: no more adaptation since learning rate is almost zero

- RMSProp: the update is

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} g_{(t)} \odot \frac{1}{\sqrt{s_{(t)} + \epsilon}}$$

  where $s_{(t)} = \beta s_{(t-1)} + (1-\beta) g^2_{(t)}$
- ADAM (Adaptive moment estimation): From

$$m_{(t)} = \beta_1 m_{(t-1)} + (1-\beta_1) g_{(t)}$$
$$s_{(t)} = \beta_2 s_{(t-1)} + (1-\beta_2) g^2_{(t)}$$

  the update becomes

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} - \alpha_{(t)} m_{(t)} \odot \frac{1}{\sqrt{s_{(t)} + \epsilon}}$$

  where m is the momentum terms taking into account previous gradients and allowing moving across flat spots of the search space.

PERCEPTRON RULE

- Binary classification task (class 1 or -1).
- Perceptron adapts the parameters of the threshold classifier

$$\hat{y} = \text{sgn}(x^T \theta)$$

only when a misclassification occurs and according to the rule

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} + \alpha_{(t)}\left(y_t - \text{sgn}(x_t^T \hat{\theta}_{(t-1)})\right)x_t$$

- So if $\hat{y}_t = y_t$ no update takes place. Otherwise, if a misclassification occurs, the following rule applies

$$\hat{\theta}_{(t)} = \hat{\theta}_{(t-1)} + 2\alpha_{(t)} y_t x_t$$

- Cost function reaches its minimal value zero when all examples are properly recognized or when the vector $\hat{\theta}$ is null.
- If the training set is linearly separable the algorithm finds a linear separation with probability one.

ONLINE K-MEANS

- Training set of N points $x_i \in \mathbb{R}^n, i = 1, \ldots, N$
- K-means is a clustering algorithm which for a fixed $K > 0$ returns the coordinates $\theta_k \in \mathbb{R}^n$ of K centroids such that the cost function $\sum_{i=1}^{N} L(x_i, \theta)$ is minimized where

$$L(x, \theta) = \min_{k=1}^{K} (x - \theta_k)^2 = (x - \theta^-)^2$$

  and θ^- the closest centroid to x.
- We can approximate the derivative of the loss with the derivative of the distance to the closest centroid
- Each time a new sample arrives, the centroid which is the closest to the new sample has its coordinates updated

$$\hat{\theta}^-_{(t)} = \hat{\theta}^-_{(t-1)} + \alpha_{(t)}(x_t - \hat{\theta}^-_{(t-1)})$$

STREAMING STRUCTURAL IDENTIFICATION

STREAMING STRUCTURAL IDENTIFICATION

- Model selection and validation is typically performed in a batch mode by use of cross-validation techniques
- Brute-force offline approach: estimate the generalization accuracy of the alternatives by means of a sufficiently large number of training and test phases. The candidate with the smallest estimated error is selected.
- Same computational resources are allocated to each model configuration: manifestly poor configurations are thoroughly tested to the same extent as the best ones are.

RACING ALGORITHMS

- Racing [6] **evaluates a set of model candidates** in a streaming fashion.
- As soon as statistical evidence is gathered against some candidates, those are eliminated and the race continues with the surviving ones.
- Racing uses one training sample at the time and generating a sequence of nested sets of candidates
- **Statistical tests** to discard alternatives that are significantly worse: Hoeffding test (non parametric and non blocking) and Friedman test (nonparametric and blocking) ([3])

HOEFFDING FORMULA

Let us consider a continuous random variable $\mathbf{z} \in Z$ such that the size of the interval domain Z is smaller than B. Given the expectation $\mu = E[\mathbf{z}]$ and the sample mean $\hat{\mu} = \sum_{i=1}^{N} z_i$ it can be shown that

$$\text{Prob}\left\{|\hat{\mu} - \mu| > \epsilon\right\} < 2\exp^{-2N\epsilon^2/B^2}$$

If we set

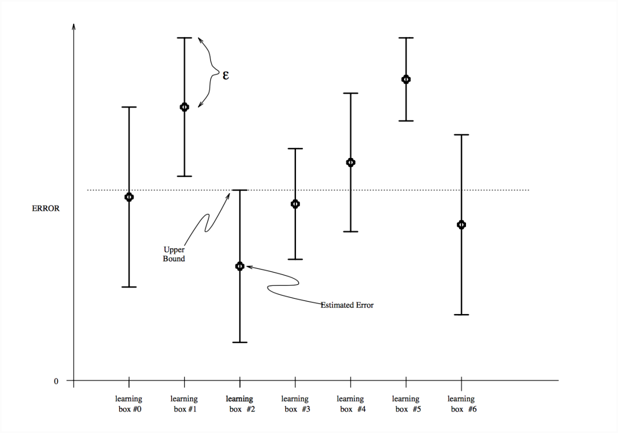
$$\delta = 2\exp^{-2N\epsilon^2/B^2} \Rightarrow \epsilon = \sqrt{\frac{B^2\log(2/\delta)}{2N}}$$

we have with confidence $1 - \delta$

$$|\hat{\mu} - \mu| < \sqrt{\frac{B^2\log(2/\delta)}{2N}}$$

We have seen before how the quantities $\hat{\mu}$ may be updated sequentially.

HOEFFDING RACE



From [6]

NONSTATIONARY AND EVOLVING ENVIRONMENTS

NONSTATIONARITY

A data generating process is stationary if the probabilistic process underlying the generation of data does not depend on time. In practice, a time-varying generating process is common in many settings like

- finance
- energy
- climate
- web
- environmental monitoring: sensor networks
- malware/spam/frauds
- recommendation systems

Causes:

- seasonality, periodicity effects
- changes in users' habits or preferences, changes in operating regimes
- faults
- natural evolution, drifts
- aging effects
- hidden state

TIME-SERIES VS. TIME VARYING

- A time-series is not necessarily time-varying
- For fixed values of a and b the process

$$y(t+1) = ay(t) + by(t-1) + \epsilon$$

  generates a time-series which is time-invariant.
- If a and b change with time the series becomes non-stationary

NONSTATIONARITY

- Assumption of identical distribution in training and test is then unrealistic.
- Existence of drifts. Probability distribution changes with time
- Nonadaptive model is suboptimal and bound to become obsolete.
- Evident importance of learning in nonstationary environments
- Two main approaches [4]
    - Active methods rely on an explicit detection of the change in the data distribution
    - Passive: continuously update the model over time (without explicit detection)
- Active methods preferable in abrupt changes
- Active methods sensitive to the false positive (false alarm) ratio
- Passive methods suitable in gradual drifts and recurring concepts settings.

PASSIVE APPROACHES

- Continuous updating the model without explicitly knowing whether a concept drift occurred or not
- Neither a priori nor derived information available about concept drift
- Strategies:
  - Single learner: online learning, forgetting strategies
  - Ensemble learning: for instance $\sum_{m=1}^{M} w_m h_m(x)$ where the weights of the aggregation are inversely proportional to the error of the component. For instance if E_m is validation error of the mth model we could set

    $$w_m = \frac{\frac{1}{E_m}}{\sum_{m=1}^{M} \frac{1}{E_m}}$$

    They manage the drift by adding or removing classifiers [5].
  - Racing strategies

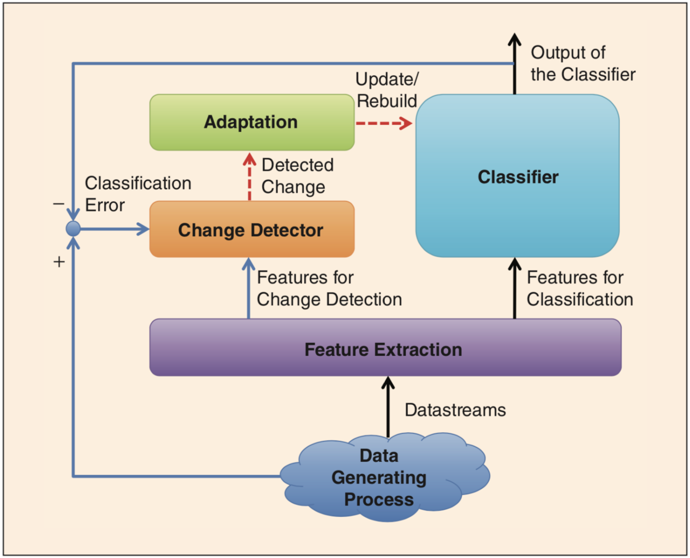
PASSIVE APPROACHES: ISSUES

- (+): no explicit decision then no risk of false alarms (like in active)
- (-): stability-plasticity dilemma: trade off between **stability** (i.e. retaining existing and still relevant knowledge) and **plasticity** (i.e. learning new knowledge)

ACTIVE APPROACHES

- They take advantage of a priori information or automatic systems able to detect changes in either input or conditional distributions
- Use of features (marginal and conditional) assumed to be stationary in stationary configurations.
- **Detect and react approach**: once an explicit detection of the change is done, obsolete knowledge is discarded and an adaptation mechanism is activated.
- Risk of false positives and false negatives.
- Statistical tools: (sequential) hypothesis tests.
- Example: change detection tests.

DETECT AND REACT



CHANGE DETECTION TESTS

- Designed to operate in a fully sequential manner
- Reduced computational complexity
- Typically relying on thresholds (Hoeffding bound, validation error).
- Difficult to set the threshold at design time; too low values induce many False Positives while too large values induce False Negatives.

ADAPTATION AFTER THE DETECTION

Three main strategies:

- Windowing: samples within a recent window are used to retrain the classifier whereas older ones are discarded
- Weighting: all samples are taken into consideration but suitably weighted according to recency or relevancy
- Biased reservoir sampling: e.g. reservoir sampling guaranteeing a uniform sampling also for very long streams. However a completely unbiased sample is undesirable since the data stream may evolve, and the vast majority of the points in the sample may represent the old history of the stream. Biased reservoir sampling techniques have been proposed in [2].

Charu C. Aggarwal.
Data Streams: Models and Algorithms (Advances in Database Systems).
Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

Charu C. Aggarwal.
On biased reservoir sampling in the presence of stream evolution.
In Proceedings of the 32Nd International Conference on Very Large Data Bases, VLDB '06, pages 607–618. VLDB Endowment, 2006.

Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp.
A racing algorithm for configuring metaheuristics.
In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

G. Ditzler, M. Roveri, C. Alippi, and R. Polikar.
Learning in nonstationary environments: A survey.
IEEE Computational Intelligence Magazine, 10(4):12–25, Nov 2015.

R. Elwell and R. Polikar.

Incremental learning of concept drift in nonstationary environments.
IEEE Transactions on Neural Networks, 22(10):1517–1531, Oct 2011.

O. Maron and A. Moore.
The racing algorithm: Model selection for lazy learners.
Artificial Intelligence Review, 11(1–5):193–225, 1997.