Online Machine Learning

Simone Filice
filice.simone@gmail.com
University of Roma Tor Vergata

Motivations

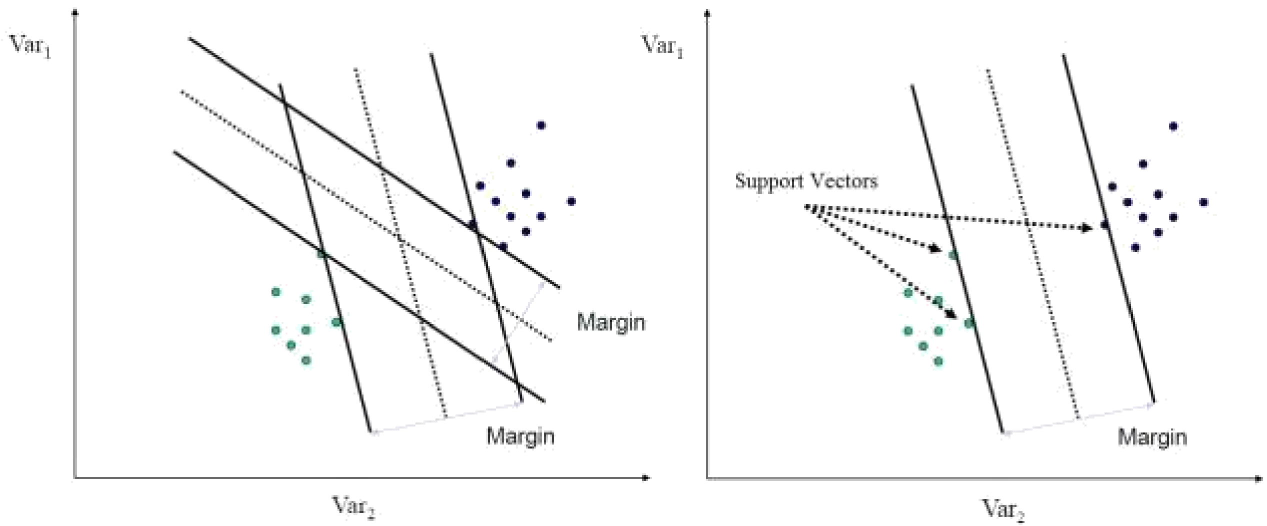
- Common ML algorithms simultaneously exploit a whole dataset. This process, referred as *batch learning*, is not practical when:
  - New data naturally arise over the time: exploiting new data means building from scratch a new model → usually not feasible!
  - The dataset is too large to be efficiently exploited: memory and computational problems!
  - The concept we need to learn changes over the time: batch learning provide a static solution that will surely degrade as time goes by

Online Machine Learning

- Incremental Learning Paradigm:
  - Every time a new example is available, the learned hypothesis is updated
- Inherent Appealing Characteristics:
  - The model does not need to be re-generated from scratch when new data is available
  - Capability of tracking a Shifting Concept
  - Faster training process if compared to batch learners (e.g. SVM)

Perceptron

- Perceptron is a simple discriminative classifier
  - Instances are feature vectors $\boldsymbol{x}' \in \mathbb{R}^d$ with label $y \in [-1, +1]$
  - Classification function is an hyperplane in $\mathbb{R}^d$: $f(\boldsymbol{x}') = \boldsymbol{w}' \cdot \boldsymbol{x}' + b$



- Compact notation: $\boldsymbol{w} = \{b, w'_1, w'_2, \ldots, w'_d\}$, $\boldsymbol{x} = \{1, x'_1, x'_2, \ldots, x'_d\}$

Batch Perceptron

- IDEA : adjust the hyperplane until no training errors are done (input data must be linearly separable)
- Batch perceptron learning procedure:

Start with $\boldsymbol{w}_1 = 0$

do

    errors=false

    For all t=1…T

        Receive a new sample $\boldsymbol{x_t}$

        Compute $y = \boldsymbol{w}_t \cdot \boldsymbol{x_t}$

        if $y \cdot y_t < \beta_t$ then $\boldsymbol{w}_{t+1} = \gamma_t \boldsymbol{w}_t + \alpha_t y_t \boldsymbol{x}_t$ with $\alpha_t > 0$
            errors=true

        else

            $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$

while(errors)

return $\boldsymbol{w}_{T+1}$

Online Learning Perceptron

- IDEA : adjust the hyperplane after each classification ($\boldsymbol{w}_t$ = weight vector at time *t*) and never stop learning

- Online perceptron learning procedure:

```
Start with w_1 = 0
For all t=1…
      Receive a new sample x_t
      Compute y = w_t · x_t
      Receive a feedback y_t
      if y · y_t < β_t then w_{t+1} = γ_t w_t + α_t y_t x_t   with α_t > 0
      else   w_{t+1} = w_t
endfor
```

Start with $\boldsymbol{w}_1 = 0$

For all t=1…

Receive a new sample $\boldsymbol{x_t}$

Compute $y = \boldsymbol{w}_t \cdot \boldsymbol{x_t}$

Receive a feedback y_t

if $y \cdot y_t < \beta_t$ then $\boldsymbol{w}_{t+1} = \gamma_t \boldsymbol{w}_t + \alpha_t y_t \boldsymbol{x}_t$ with $\alpha_t > 0$

else $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$

endfor

Shifting Perceptron

- IDEA: weak dependance from the past in order to obtain a tracking ability

- Shifting Perceptron learning procedure (Cavallanti et al 2006):

Start with $\boldsymbol{w}_1 = 0$, k=0
For all t=1…
    Receive a new sample $\boldsymbol{x_t}$
    Compute $y = \mathrm{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x_t})$
    Receive a feedback y_t
    if $y \neq y_t$ then
        $\lambda_k = \frac{\lambda}{\lambda + k}$ with $\lambda > 0$
        $\boldsymbol{w}_{t+1} = (1 - \lambda_k)\boldsymbol{w}_t + \lambda_k y_t \boldsymbol{x}_t$
        k=k+1
    else $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t$
endfor

Online Linear Passive Aggressive (1/3)

- IDEA: Every time a new example $\langle x_t, y_t \rangle$ is available the current classification function is modified as less as possible to correctly classify the new example

- Passive Aggressive learning procedure (Crammer et al 2006):

Start with $\boldsymbol{w}_1 = 0$, k=0

For all t=1…

  Receive a new sample $\boldsymbol{x_t}$

  Compute $y = \mathrm{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x_t})$

  Receive a feedback y_t

  Measure a classification loss (divergence between y_t and y)
  Modify the model to get zero loss, preserving what was learned from previous examples

Online Linear Passive Aggressive (2/3)

- Loss measure:

  Hinge loss: $l(\boldsymbol{w};(\boldsymbol{x}_t, y_t)) = \max(0; 1 - y_t(\boldsymbol{w} \cdot \boldsymbol{x}_t))$

- Model variation:

  $$\|\boldsymbol{w}_{t+1} - \boldsymbol{w}_t\|^2$$

- Passive Aggressive Optimization Problem:

  $$\boldsymbol{w}_{t+1} = argmin_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}_t\|^2 \text{ such that } l(\boldsymbol{w};(\boldsymbol{x}_t, y_t)) = 0$$

- Closed form solution:

  $$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t y_t \boldsymbol{x}_t \text{ where } \tau_t = \frac{l(\boldsymbol{w}_t;(\boldsymbol{x}_t, y_t))}{\|\boldsymbol{x}_t\|^2}$$

Online Linear Passive Aggressive (3/3)

- The previous formulation is a hard margin version that has a problem:
  - a single outlier could produce a high hyperplane shifting, making the model forget the previous learning
- Soft version solution:
  - control the algorithm aggressiveness through a parameter C

- PA-I formulation:

$$\boldsymbol{w}_{t+1} = argmin_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + C\xi \text{ s.t. } l\big(\boldsymbol{w}; (\boldsymbol{x}_t, y_t)\big) \leq \xi \text{ with } \xi \geq 0$$

$$\Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t y_t \boldsymbol{x}_t \text{ where } \tau_t = \min\left\{C; \frac{l(\boldsymbol{w}_t; (\boldsymbol{x}_t, y_t))}{\|\boldsymbol{x}_t\|^2}\right\}$$

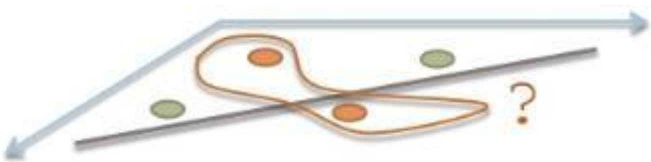
- PA-II model:

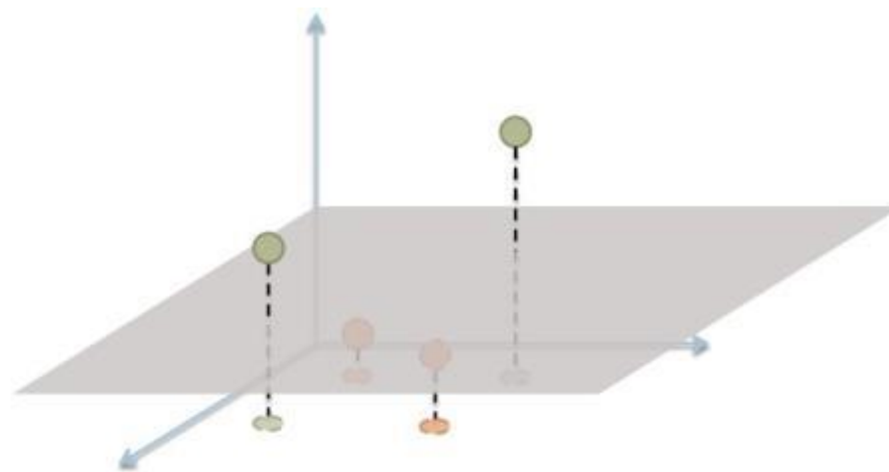
$$\boldsymbol{w}_{t+1} = argmin_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w} - \boldsymbol{w}_t\|^2 + C\xi^2 \text{ s.t. } l\big(\boldsymbol{w}; (\boldsymbol{x}_t, y_t)\big) \leq \xi \text{ with } \xi \geq 0$$

$$\Rightarrow \boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t y_t \boldsymbol{x}_t \text{ where } \tau_t = \frac{l(\boldsymbol{w}_t; (\boldsymbol{x}_t, y_t))}{\|\boldsymbol{x}_t\|^2 + \frac{1}{2}C}$$

Data Separability

- Training data could not be separable
- Possible solutions:
  - Use a more complex classification function → Risk of overfitting!
  - Define a new set of feature that makes the problem linearly separable

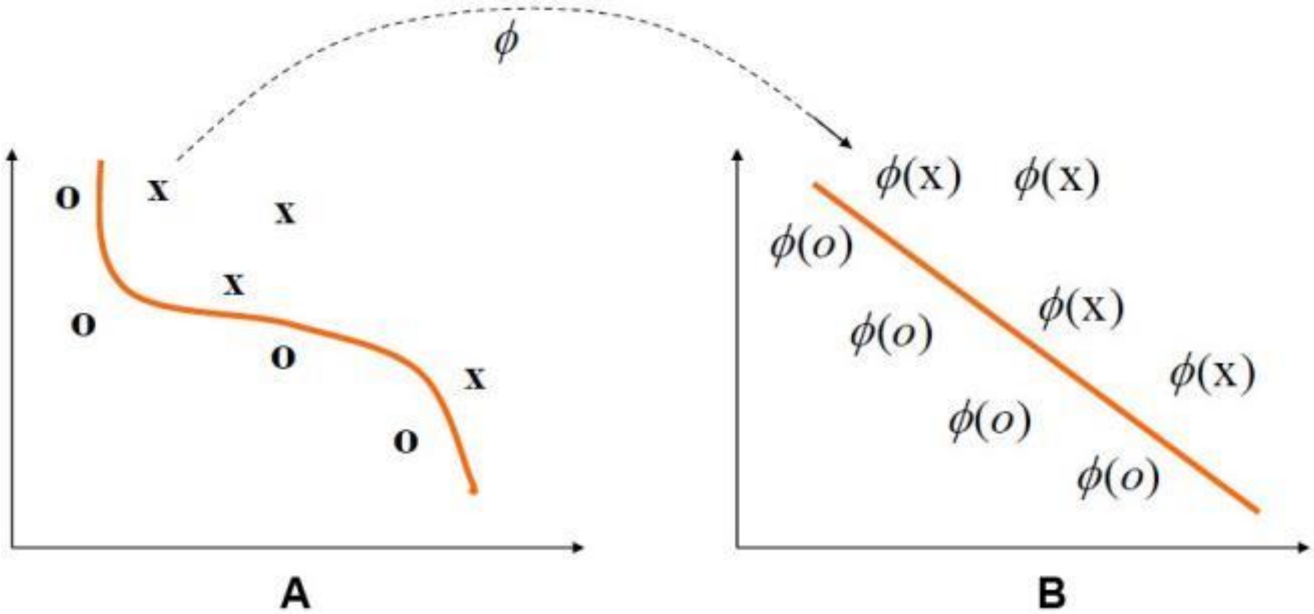




  - Project the current examples in a space in which they are separable…

Kernel Methods

- Training data can be projected in a space in which they are more easily separable



- Kernel Trick: any kernel function K performs the dot product in the kernel space without explicitly project the input vectors in that space
- Structured data (tree, graph, high order tensor…) can be exploited

Kernelized Passive Aggressive

- In kernelized Online Learning algorithms a new support vector is added every time a misclassification occurs

LINEAR VERSION	KERNELIZED VERSION
Classification function	
$f_t(\boldsymbol{x}) = \boldsymbol{w}_t{}^T \boldsymbol{x}$	$f_t(x) = \sum_{i \in S} \alpha_i k(x, x_i)$
Optimization Problem (PA-I)	
$\boldsymbol{w}_{t+1} = argmin_{\boldsymbol{w}} \frac{1}{2} \lVert \boldsymbol{w} - \boldsymbol{w}_t \rVert^2 + C\xi$ Such that $1 - y_t \mathrm{f}_t(\boldsymbol{x}_t) \leq \xi, \xi \geq 0$	$f_{t+1}(x) = \mathrm{argmin}_{\mathrm{f}} \frac{1}{2} \lVert f(x) - f_t(x) \rVert^2{}_{\mathcal{H}} + C\xi$ Such that $1 - y_t f_t(x_t) \leq \xi, \xi \geq 0$
Closed form solution	
$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \tau_t y_t \boldsymbol{x}_t$ where $\tau_t = \min\left\{C; \frac{\max(0, 1 - y_t f_t(\boldsymbol{x}_t))}{\lVert \boldsymbol{x}_t \rVert^2}\right\}$	$f_{t+1}(x) = \mathrm{f}_t(x) + \alpha_t k(x, x_t)$ where $\alpha_t = y_t \cdot \min\left\{C; \frac{\max(0, 1 - y_t f_t(x_t))}{\lVert x_t \rVert^2{}_{\mathcal{H}}}\right\}$

Linear Vs Kernel Based Learning

LINEAR VERSION	KERNELIZED VERSION
Classification function	
explicit hyperlplane in the original space ☹ Only linear functions can be learnt	implicit hyperplane in the RKHS ☺ Non linear functions can be learnt
Example form	
☹ Only feature vectors can be exploited	☺ Structured representations can be exploited
Computational complexity	
☺ A classification is a single dot product	☹ A classification involves $\|S\|$ kernel computations
Memory usage	
☺ Only a the explicit hyperplane must be stored	☹ All the support vectors and their weights must be stored

Learning on a Budget

- In kernelized online learning algorithm the set of support vectors can grow without limits
- Possible solution: Limit the number of support vector, defining a budget *B*
- This solution has the following advantages:
  - The memory occupation is upperbounded by *B* support vectors
  - Each classification needs at most *B* kernel computations
  - In shifting concept tasks, budget algorithms can outperform non-budget counterparts because they are faster in adapting

Limit the number of Support Vectors

- In order to respect the budget B, different policies can be formulated:
  - Stop learning when budget is exceeded: *Stoptron*
  - Delete a random support vector: *Randomized Perceptron*
  - Delete the more redundant support vector: *Fixed Budget Conscious Perceptron*
  - Delete the oldest support vector: *Least recent Budget Perceptron* and *Forgetron*
  - Modify the Support Vectors weights in order to adapt the classification hypothesis to the new sample: *Projectron*
  - *Online Passive-Aggressive on a Budget*

Stoptron

- Baseline of the online learning on a budget algorithms: Fix a budget *B* and stop learning when the number of support vectors is equal to *B*

- Stoptron algorithm (Orabona et al 2008):

Start with $S = \emptyset$

For all t=1…

    Receive a new sample $\boldsymbol{x_t}$

    Compute $y = \sum_{i \in S} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}_t)$

    Receive a feedback y_t

    if $yy_t < \beta$ and $|S| < B$ then

        $S = S \cup \{t\}$

        $\alpha_t = 1$

    endif

endfor

Randomized Perceptron

- Simplest deleting policy: when the budget B is exceeded remove a random support vector
- Randomized Perceptron algorithm (Cavallanti et al 2007):

```
Start with S = ∅
For all t=1…
    Receive a new sample x_t
    Compute   y = Σ_{i∈S} α_i y_i K(x_i, x_t)
    Receive a feedback y_t
    if  y y_t < β
        if |S| = B
            select randomly s ∈ S, S = S \ {s}
        endif
        S = S ∪ {t}  α_t = 1
    endif
endfor
```

Forgetron

- <u>Deleting policy:</u> Every time a new support vector is added, the weights of the others are reduced. Thus SVs lose weight with aging and removing the older SV should assure a minimum impact to the classification function.
- Forgetron algorithm (Dekel et al 2008):

```
Start with S = ∅
For all t=1…
    Receive a new sample x_t
    Compute  y = Σ_{i∈S} α_i y_i K(x_i, x_t)
    Receive a feedback y_t
    if  y y_t < β
        if |S| = B
            S=S\min{S} //the oldest Support vector is removed
        endif
        S = S ∪ {t}  α_t = 1,  α_i = φ_t α_i  ∀i ∈ S \ {t} //adding a new Sv and shrinking
    endif
endfor
```

Summary

- Online learning methods can:
  - Incrementally learn from new samples
  - Dinamically adapt to problem variations
  - Reduce the computational cost of building a new model

- Online learning methods can be used with kernels but they suffer from the "*curse of kernelization*":
  - The number of support vectors can grow without bounds

- Several number of budgeted solutions have been proposed