Project Proposal Comments (see Canvas' announcement)

First, there is significant variability in the proposals. Clearly, some people have spent quite a bit of time thinking about their project. Other proposals seemed to have been written at the last minute just to get through the proposal deadline. This is not a very good strategy. From experience, good proposals correlate with good projects. If you have not completely figured out what you want to do for your project, you are behind schedule.

After reading the proposals, let me just summarize some major comments that apply to almost everyone (Note: In these comments, I am not referring to specific proposals submitted to the class.) **Recall that the project will be evaluated by the criteria mentioned in** ProjectEvaluationGuidelines.pdf

Roughly speaking, the projects can be grouped in two major classes: **compare** vs. **improve**.

- **Compare projects** aim to implement and compare some techniques on a certain problem. Since these projects will not score highly in terms of creativity criterion, it is important that the comparison aspect is extensive. These days you can download lots of methods. E.g., CNNs for images include AlexNet, VGG, Inception, ResNet, etc. In the class quizzes, you are already asked to implement things on your own. For the project, you can use libraries. So, there is no reason for you not to do a very thorough job.
- **Improve projects** are projects that propose to apply machine learning to solve some new problem or advance some machine learning algorithm. These projects can score higher on creativity. However, keep in mind that most of the things that you find out there are things that we are aware of. We will not be impressed by a very cool demo that we know to be the straightforward application of code released with a recent paper. You will not get any extra creativity points for this. Which means that you should still compare different solutions for the problem or genuinely add some improvements (if this is a problem that you care about). Your paper should also make very clear what is the contribution beyond the original model, if you are to get any additional creativity points.

Note, also, that this is a **machine learning class**. You can use machine learning for anything you want, but what we care about is machine learning. Creating an impressive image morphing function is not machine learning, but image processing. Unless, of course, machine learning is at the center of the morphing technique. If you invent a new set of MFCC++ features for audio, this is audio, not machine learning. In this class, we are more interested in the comparison of the MFCC features to some features extracted by a CNN, boosting, or something like that. The point is that we will heavily disregard components that are not ML-based. Please do not complain later on about "I spend two weeks implementing this beautiful image morphing technique and you did not give me any credit." Of course, if the image morphing is something that you can add with minor work and benefits your results, by all means. But you will not get credit explicit for it.

The same is true for results. We are not interested in learning about the statistics of bitcoin usage. What we care about is what **ML techniques** enabled you to get to those conclusions and how. You could get a Nobel prize in economics from the paper that you wrote for this class and, if the paper does not contain anything interesting in terms of machine learning, still get a low score in the project. OK, maybe I am exaggerating a bit, but you get the point.

Some of you are focused on a particular model for the solution of a particular task. This is OK, but keep in mind that you will not get credit for advancing your thesis research if this is not ML. You still need to compare that model to other ML solutions or make machine learning improvements on the model. This is what you will get credit for.

For some of you, this is all that I am going to say. This means that your project direction looks fine at this point, but it is still your responsibility to think about the issues above and steer the project in the best possible direction. If you have any questions, please feel free to ask.

ECE 271B: Take-Home Quizzes Guidelines

By submitting your quiz solution, you agree to comply with the following.

1. The quiz should be treated as a **take-home test** and be an **INDIVIDUAL** effort. **NO collaboration is allowed**. The submitted work must be yours and must be original.
2. The work that you turn-in to be your own, using the resources that are available to **all** students in the class.
3. You are not allowed to consult or use resources provided by tutors, previous sludents in the class, or any websites that provide solutions or help in solving assignments and exams.
4. You will not upload your solutions or any other course materials to any websites or in some other way distribute them outside the class.
5. 0 points will be assigned to any problem that seems to violate these rules and, if recurrent, the incident(s) will be reported to the Academic Integrity Office.

With respect with quiz logistics, you should do the following.

1. Quizzes should be submit in PDF format on Gradescope by **11:59 pm of the due date**. Late submissions will be accepted within 24 hours, but will incur a 20% penalty. After that, there will be no credit.
2. If there are issues that need clarification, feel free to ask on Piazza. However, make sure **not to give the solutions away**. General questions that are not specifically about the problems can, of course, be discussed openly. It follows that if you can frame your question about the problem more generally, you will get a lot more feedback. In general, this also applies to the TAs office hours. If you are stuck in a problem, feel free to go see the TAs. However, TAs will not solve the problem for you. Make sure to ask the question more generally.
3. **Start early** because some problems might need non-trivial amounts of computer time.
4. Unless instructed otherwise, you have to **write all the code** (no packages allowed). If in doubt, ask on Piazza.
5. **All code used to solve the computer problems must be submitted with your quiz**. While we will not be grading code, the TA might need to check it up. If the code is not submitted, 0 points will be assigned to the computer problem.
6. Any request of **quiz regrading** must be submitted on Gradescope **within one week** after the release of the respective graded quiz.
7. Be considerate of the TAs that will be grading your quiz by **submitting a readable PDF document**. Be aware that there is no obligation on the part of the TAs to put effort into deciphering quizzes beyond what is reasonably expected. Typical problems for handwritten documents are: 1) poor handwriting; 2) student writes on both sides of the page and ink bleeds from the background; 3) documents "scanned" by a taking a picture, where there are issues of camera focus or perspective effects that compromise reading; 4) a PDF that is compiled with pages or images upside-down, out of order, or with a skewed perspective. These are issues that severely affect the ability of the TAs to do their job and can be easily avoided with some minimal amount of planning. Now that you are made aware of them, it should be fairly trivial to avoid them. If the TAs are faced with these issues, they can choose not to grade the problem. I give them that discretion.

Quiz #2 due **today** @ 11:59pm

Quiz #3 posted on Canvas
Due date: **Tuesday, 2/22**

ECE 271B – Winter 2022

Boosting (cont.)

Disclaimer:
This class will be recorded
and made available to students asynchronously.

Manuela Vasconcelos

ECE Department, UCSD

Boosting

- a procedure to **learn** ensemble learners

$$h(\mathbf{x}) = \text{sgn}[g(\mathbf{x})] \qquad g(\mathbf{x}) = \sum_i w_i\, \alpha_i(\mathbf{x})$$

  where the functions $\alpha_i(\mathbf{x})$ are called weak learners

- the question is:

  how do we <u>**learn**</u> the **"right" functions** and the **weights** w_i?

- as before, we consider a **loss/cost** $L[y, g(\mathbf{x})]$ of making a prediction $g(\mathbf{x})$ when the true value is y

- given the training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, the goal is to minimize the **empirical risk**

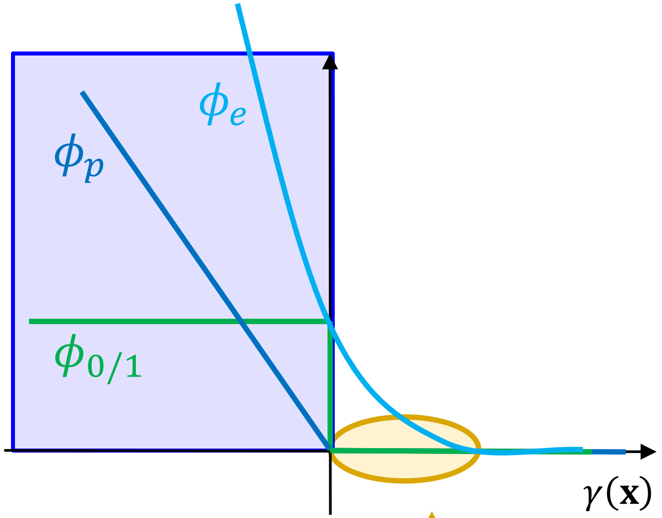
$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} L[y_i, g(\mathbf{x}_i)]$$

Loss Function

- boosting optimizes a margin loss

$$L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big) = \phi(\gamma(\mathbf{x}))$$

  that
  - besides penalizing errors
  - encourages large margins if there is no error



- this is achieved by using the exponential loss (AdaBoost)

$$L[y, g(\mathbf{x})] = \phi_e\big(yg(\mathbf{x})\big) = \exp\big(-yg(\mathbf{x})\big)$$

  which introduces a penalty for small positive margins

- losses with this property are called margin enforcing losses

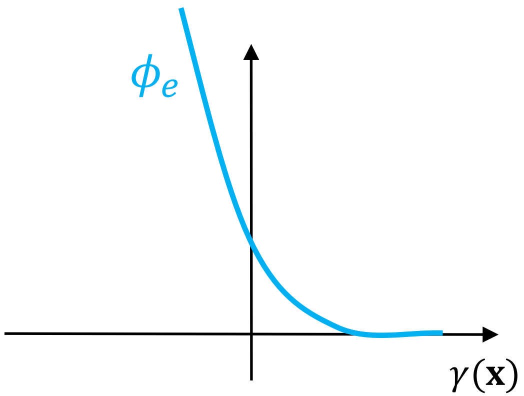
Boosting

- the goal is to find the **ensemble learner**

$$\boxed{g(\mathbf{x}) = \sum_i w_i \alpha_i(\mathbf{x})}$$

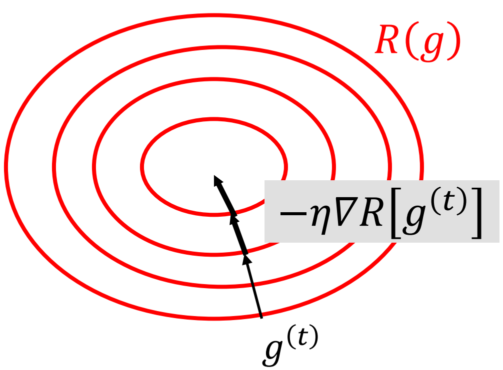
- that **minimizes the risk**

$$\boxed{R_{emp}[g(\mathbf{x})] = \frac{1}{n}\sum_{i=1}^{n} \phi_e\big(y_i g(\mathbf{x}_i)\big) = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]}$$



- note that $g(\mathbf{x})$ is a combination of functions $\alpha_i(\mathbf{x})$
- if we **can compute the gradient** $\nabla R_{emp}[g(\mathbf{x})]$ (**assuming** that **we can pick a different step size** η **per iteration** t), then we can **minimize the risk** by **gradient descent**

> - pick initial estimate $g^{(0)}$
> - follow the **negative gradient**
>
> $$g^{(t+1)} = g^{(t)} - \eta^{(t)} \nabla R_{emp}\left[g^{(t)}\right]$$



Boosting

assuming that we can
- compute the gradient
- pick a different step−size η per iteration

a g **learned** after $t+1$ iterations given by

$$\boxed{g^{(t+1)}} = g^{(t)} - \eta^{(t)} \nabla R_{emp}\left[g^{(t)}\right]$$
$$= g^{(t-1)} - \eta^{(t-1)} \nabla R_{emp}\left[g^{(t-1)}\right] - \eta^{(n)} \nabla R_{emp}\left[g^{(t)}\right]$$
$$= \cdots$$
$$= \boxed{-\sum_{i=1}^{t} \eta^{(i)} \nabla R_{emp}\left[g^{(i)}\right]}$$

(where we have assumed that $g^{(0)} = 0$)

- note that this is our ensemble learner

$$g(\mathbf{x}) = \sum_i w_i \alpha_i(\mathbf{x})$$

if we make the **equivalences**

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] \qquad w_t = \eta^{(t)}$$

Boosting

- last class, we show that the **gradient** along the **direction** (function) $u(\mathbf{x})$ is

$$\boxed{\nabla R_{emp}\left[g^{(t)}(\mathbf{x})\right] = \arg\min_{u} \sum_i y_i u(\mathbf{x}_i) \varpi_i}$$

  with

$$\boxed{\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i}$$

- ϖ_i can be seen as the weight of example $\mathbf{x}_i$ and does not depend on the direction u, just on the classifier already available at iteration t

- we do not optimize over all possible functions

- instead, we define a family U of functions and optimize over the elements of U

$$\nabla R_{emp}\left[g^{(t)}(\mathbf{x})\right] = \arg\min_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

- U can be **many** things (more on this later)

Boosting

- this leads to the <u>final form</u> of the **algorithm**

  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}\left[g^{(t)}\right]$ is decreasing
    - compute the weights

$$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

    - compute the negative gradient

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u \in U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

    - compute the step−size

$$w_t = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

    - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

Boosting: Weight

while $R_{emp}[g^{(t)}]$ is decreasing
- compute the weights

  $$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$
- compute the negative gradient

  $$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$
- compute the step size

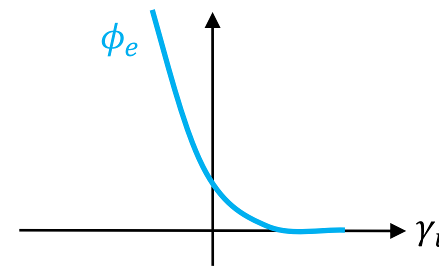
  $$w_t = \arg\min_{w} R_{emp}\left[g^{(t)} + w\alpha_t\right]$$
- update the learned function

  $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

- we can get some **intuition** by recalling that
  - the risk is

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \phi_e[y_i g(\mathbf{x}_i)] = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$

where $yg(\mathbf{x}_i) = \gamma_i$ is the margin of example $\mathbf{x}_i$



- hence, the **boosting weight** ϖ_i of $\mathbf{x}_i$

$$\boxed{\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)] = \phi_e(\gamma_i), \forall i}$$

depends on the margin γ_i of $\mathbf{x}_i$ under the **current** function $g^{(t)}(\mathbf{x}_i)$

  - it is large for the examples of large negative margin $\gamma_i \ll 0$ (these are examples $\mathbf{x}_i$ with **large error** under the **current** classifier)
  - it is approximately zero for the examples of **positive** margin $\gamma_i > 0$ (these are examples $\mathbf{x}_i$ **correctly** classified under the **current** classifier)
- **in summary**, the weighting mechanism makes **boosting** focus on the **hard** examples

Boosting

emphasizes "hard" examples

- initialize $t = 0,\ g^{(t)} = 0$
- while $R_{emp}\left[g^{(t)}\right]$ is decreasing
  - compute the weights

$$\boxed{\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i}$$

  - compute the negative gradient

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u\,\in\,U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

  - compute the step−size

$$w_t = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

  - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

note that this is a **generalization** of the **Perceptron**, which only considers errors, but weighs all errors equally

Perceptron Learning

set $k = 0, w_k = 0, b_k = 0$
set $R = \max_i \|\mathbf{x}_i\|$
do {
for $i = 1{:}n$ {
if $\boxed{y_i(\mathbf{w}_k{}^T\mathbf{x}_i + b_k) \leq 0}$ then {
- $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta y_i \mathbf{x}_i$
- $b_{k+1} = b_k + \eta y_i R^2$
- $k = k + 1$

}
}
} until $y_i(\mathbf{w}^T\mathbf{x}_i + b_k) > 0, \forall i$ (no errors)

Boosting : Gradient Step

while $R_{emp}[g^{(t)}]$ is decreasing

- compute the weights

$$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$

- compute the negative gradient

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

- compute the step size

$$w_t = \arg\min_w R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

- update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

- the gradient step
  - consists of selecting the "weak learner" u in U such that

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

  - note that

$$y_i\, u(\mathbf{x}_i) = \gamma_i'$$

is the example margin of $\mathbf{x}_i$ for classification by the weak learner $u(\mathbf{x})$

and

$$\sum_i y_i\, u(\mathbf{x}_i)\, \varpi_i = \sum_i \gamma_i' \varpi_i$$

(up to a scaling constant which makes no difference in the maximization)

is a weighted average of the margin over all examples $\mathbf{x}_i$,

where example $\mathbf{x}_i$ is weighted (ϖ_i) by how hard it is to classify

- **in summary**, boosting picks the weak learner of largest margin on the reweighted training set

Boosting

- what about the gradient step?

- initialize $t = 0,\ g^{(t)} = 0$
- while $R_{emp}\left[g^{(t)}\right]$ is decreasing
  - compute the weights

$$\boxed{\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i}$$

  - compute the negative gradient

$$\boxed{\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u \in U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i}$$

  - compute the step−size

$$w_t = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

  - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

emphasizes "hard" examples

picks weak learner of largest weighted margin

Boosting: Step–Size

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$

$$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$

- what about the **step–size**?

$$w_t = \arg\min_{w} \; R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

- since

$$R_{emp}\left[g^{(t+1)}\right] = R_{emp}\left[g^{(t)} + w\alpha_t\right] = \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i\left(g^{(t)}(\mathbf{x}_i) + w\alpha_t(\mathbf{x}_i)\right)\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i g^{(t)}(\mathbf{x}_i)\right] \exp[-y_i w\alpha_t(\mathbf{x}_i)]$$

$$= \frac{1}{n}\sum_{i=1}^{n} \varpi_i \exp[-y_i w\alpha_t(\mathbf{x}_i)]$$

and

$$\frac{d}{dw} R_{emp}\left[g^{(t+1)}\right] = \frac{1}{n}\sum_{i=1}^{n} \varpi_i \frac{d}{dw}\exp[-y_i w\alpha_t(\mathbf{x}_i)] = -\frac{1}{n}\sum_{i=1}^{n} \varpi_i\, y_i \alpha_t(\mathbf{x}_i)\exp[-y_i w\alpha_t(\mathbf{x}_i)]$$

- the <u>optimal</u> **step–size** must satisfy the **condition**

$$\boxed{\sum_{i=1}^{n} \varpi_i\, y_i \alpha_t(\mathbf{x}_i)\exp[-y_i w\alpha_t(\mathbf{x}_i)] = 0}$$

Boosting: Step–Size

optimal step–size must satisfy

$$\sum_{i=1}^{n} \varpi_i\, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w \alpha_t(\mathbf{x}_i)] = 0$$

while $R_{emp}(g^{(t)})$ is decreasing

- compute the weights

$$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$

- compute the negative gradient

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

- compute the step size

$$w_t = \arg\min_{w} R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

- update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

$$\begin{aligned}
\boxed{0} &= \sum_{i=1}^{n} \varpi_i\, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w \alpha_t(\mathbf{x}_i)] \\
&= \sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i) \exp\left[-y_i \left(g^{(t)}(\mathbf{x}_i) + w_t \alpha_t(\mathbf{x}_i)\right)\right] \\
&= \sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i) \exp[-y_i\, g^{(t+1)}(\mathbf{x}_i)] \\
&= \boxed{\sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i)\, \varpi_i^{(t+1)}}
\end{aligned}$$

$$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

$\gamma_i{}'$ – example margin of $\mathbf{x}_i$ for the selected (iteration t) weak learner

- this guarantees that the set of weights for the next iteration is "**balanced**"
- under the new weights (iteration $t+1$), the weak learner selected in the current iteration (t) has average margin equal to 0! (is "useless")
- the new weights are such that the weak learner just chosen (iteration t) has **no "confidence"** on the classification of the reweighted dataset ($t+1$)!
- "we **squeezed** all the juice out of weak learner selected at t"

Freund, Yoav; Schapire, Robert; A Decision–Theoretic Generalization of On–Line Learning and an Application to Boosting. *Journal of Computer and System Sciences* 55, 119–139,1997.

AdaBoost

- so far, we have considered ensemble classifiers

$$h(\mathbf{x}) = \operatorname{sgn}[g(\mathbf{x})] \qquad g(\mathbf{x}) = \sum_t w_t \alpha_t(\mathbf{x})$$

whose weak learners **can be <u>any</u> functions**

- what if we restrict the weak learners $\alpha_t(\mathbf{x})$ to be **<u>classifiers</u>** <u>themselves</u>?

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

$$\boxed{\alpha_t(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, t}$$

- in this case, the **ensemble rule**

$$\boxed{g(\mathbf{x}) = \sum_t w_t \alpha_t(\mathbf{x}) = \sum_{t|\alpha_t(\mathbf{x})=1} w_t - \sum_{t|\alpha_t(\mathbf{x})=-1} w_t}$$

is a **<u>true</u> voting procedure**

- $\alpha_t(\mathbf{x})$ **votes** for classes $+1$ or -1 with strength w_t
- the rule "**tallies**" the difference between the **strength** of positive and negative votes

AdaBoost

- and the optimal step–size condition is

$$\alpha_t(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, t$$

$$0 = \sum_i \varpi_i\, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w_t \alpha_t(\mathbf{x}_i)] = \sum_{i|y_i=\alpha_t(\mathbf{x}_i)} \varpi_i e^{-w_t} - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i e^{w_t}$$

and this holds if

$$e^{-w_t} \sum_{i|y_i=\alpha_t(\mathbf{x}_i)} \varpi_i = e^{w_t} \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i \iff e^{-w_t}\left(\sum_i \varpi_i - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i\right) = e^{w_t} \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i$$

$$\iff e^{2w_t} = \frac{\sum_i \varpi_i - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i} = \frac{1-\varepsilon}{\varepsilon} \quad \text{with} \quad \varepsilon = \frac{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

- hence, we have a **closed–form** for the step–size

$$\boxed{w_t = \frac{1}{2}\log\frac{1-\varepsilon}{\varepsilon}} \qquad \varepsilon = \frac{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

AdaBoost

- this is the AdaBoost algorithm
  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}[g^{(t)}]$ is decreasing
    - compute the weights

      $$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

    - compute the negative gradient

      $$\alpha_t = -\nabla R_{emp}[g^{(t)}] = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

    - compute the step−size

      $$w_t = \frac{1}{2}\log\frac{1-\varepsilon}{\varepsilon} \qquad \varepsilon = \frac{\sum_{i|y_i \neq \alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

    - update the learned function

      $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

emphasizes "hard" examples

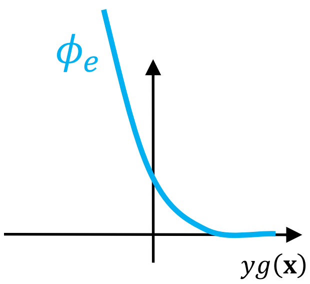
picks weak learner of largest weighted margin

there is **no** simpler ML algorithm that works!

AdaBoost

- because this is so **simple**, AdaBoost became **widely used** in machine learning
- however, it is not the **only** boosting algorithm
- recall that it assumes
  - exponential loss

$$L[y, g(\mathbf{x})] = \phi_e(yg(\mathbf{x})) = \exp[-yg(\mathbf{x})]$$



  - classification weak learners

$$\alpha_t(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, t$$

- there are other algorithms that **relax these assumptions**
  - one common variant is to consider other loss functions, more suitable to **different** types of problems
  - examples include LogitBoost, SavageBoost, tangentBoost, etc.

AdaBoost

- even if you restrict yourself to **AdaBoost**, you can implement many algorithms by choosing **different sets U of weak learners**
- this raises the questions: "**what weak learners can I use**" and "**why weak**"?
- to answer this,
  - recall that the derivative of the risk along the direction u is

$$D_u R_{emp}\left[g^{(t)}(\mathbf{x})\right] = -\frac{1}{n}\sum_{i=1}^{n} y_i u(\mathbf{x}_i)\varpi_i$$

  - hence, we can **make progress** as long as we **can find a direction u** such that

$$\sum_i y_i u(\mathbf{x}_i)\varpi_i > 0$$

  - for classification learners $u(\mathbf{x}) \in \{-1,1\}, \forall \mathbf{x}$, this happens if

$$\varepsilon = \frac{\sum_{i|y_i \neq u(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

$$\sum_{i|y_i=u(\mathbf{x}_i)} \varpi_i - \sum_{i|y_i\neq u(\mathbf{x}_i)} \varpi_i > 0 \Leftrightarrow \sum_i \varpi_i - 2\sum_{i|y_i\neq u(\mathbf{x}_i)} \varpi_i > 0 \Leftrightarrow 1 - 2\varepsilon > 0$$

AdaBoost

- in **summary**, we can **make progress** as long as we can find a direction u such that

$$\varepsilon < \frac{1}{2} \qquad \varepsilon = \frac{\sum_{i|y_i \neq \alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

  i.e., if the classifier $u(\mathbf{x})$ has **<u>less</u> than 50%** error ε in the weighted training set

- this means that
  - we can <u>**always**</u> progress if the set U is such that we can find a classifier $u(\mathbf{x})$ with less than 50% error for <u>**any**</u> set of weights ϖ_i
  - note that 50% error is a <u>very easy</u> condition to guarantee since **50% is the larger error that any classifier can have on a binary classification problem**
    - "if your error is 60%, I just say the opposite and have error of 40%!"
  - for this reason, the set U is called a set of "<u>weak learners</u>"

- this is **quite exciting**

  ➡ boosting can learn a <u>**strong classifier**</u> by combining an <u>**ensemble of very weak classifiers**</u>

Decision Stumps

- in practice, the weak learners tend to be <u>extremely simple</u>
- a <u>common</u> choice of weak learner is the family of <u>decision stumps</u>
  - let $\mathbf{x}$ be a $d-$dimensional vector $\mathbf{x} = (x_1, \dots, x_d)^T$
  - a decision stump picks <u>one</u> dimension of $\mathbf{x}$, say x_j, and <u>thresholds</u> it

$$u(\mathbf{x}; j, t) = \begin{cases} 1, & x_j \geq t \\ -1, & x_j < t \end{cases}$$

  - in this case, the set U is

$$U = \{u(\mathbf{x}; j, t) \mid j \in \{1, \dots, d\}, t \in T\}$$

  where T is a set of predefined <u>thresholds</u>

    - e.g., if there are 100 thresholds, U contains $100 \times d$ weak learners

Decision Stumps

$$u(\mathbf{x}; j, t) = \begin{cases} 1, & x_j \geq t \\ -1, & x_j < t \end{cases}$$

$$U = \{u(\mathbf{x}; j, t) \mid j \in \{1, \dots, d\}, t \in T\}$$

- since

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

and for <u>classification learners</u> $u(\mathbf{x}) \in \{-1,1\}, \forall \mathbf{x}$

$$\sum_i y_i u(\mathbf{x}_i) \varpi_i = \sum_{i \mid y_i = u(\mathbf{x}_i)} \varpi_i - \sum_{i \mid y_i \neq u(\mathbf{x}_i)} \varpi_i = \sum_i \varpi_i - 2 \sum_{i \mid y_i \neq u(\mathbf{x}_i)} \varpi_i$$

- it follows that the <u>weak learner selection</u> procedure is search over features j and thresholds t

$$\boxed{\alpha_t = \arg\min_{j,t} \sum_{i \mid y_i \neq u(\mathbf{x}_i; j, t)} \varpi_i}$$

- this is the weak learner of <u>minimum error</u> ε → $\varepsilon = \frac{\sum_{i \mid y_i \neq \alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$

Decision Stumps: Example

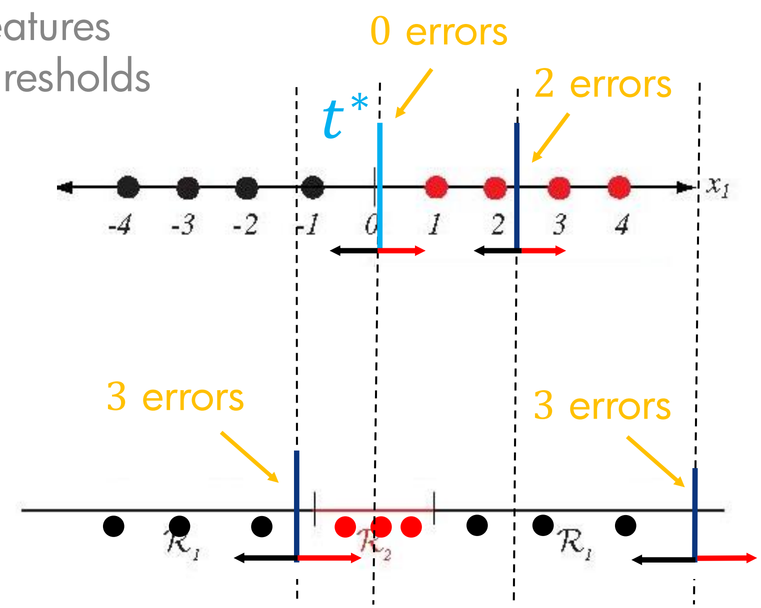
- decision stumps
  - simply cycle through <u>all</u> the features and, for each,
    - find optimal threshold
    - compute error ε

<u>example</u>

2 features
4 thresholds

feature 1 (x_1)

feature 2 (x_2)



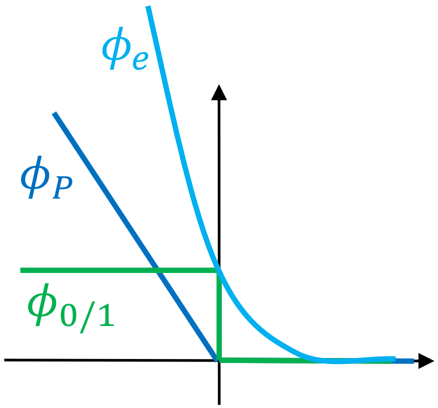
- pick the feature with overall <u>smallest</u> error ε and <u>best</u> threshold
- in this case, feature 1 and threshold t^*

Boosting

- in <u>summary</u>, there is a large number of boosting algorithms
- you need to choose
  - a margin loss function

$$\boxed{L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big)}$$

  - a set U of weak learners



- AdaBoost results from the choice of
  - exponential loss

$$\boxed{L[y, g(\mathbf{x})] = \exp\big(-yg(\mathbf{x})\big)}$$

  - classification weak learners

$$\boxed{\alpha_i(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, i}$$

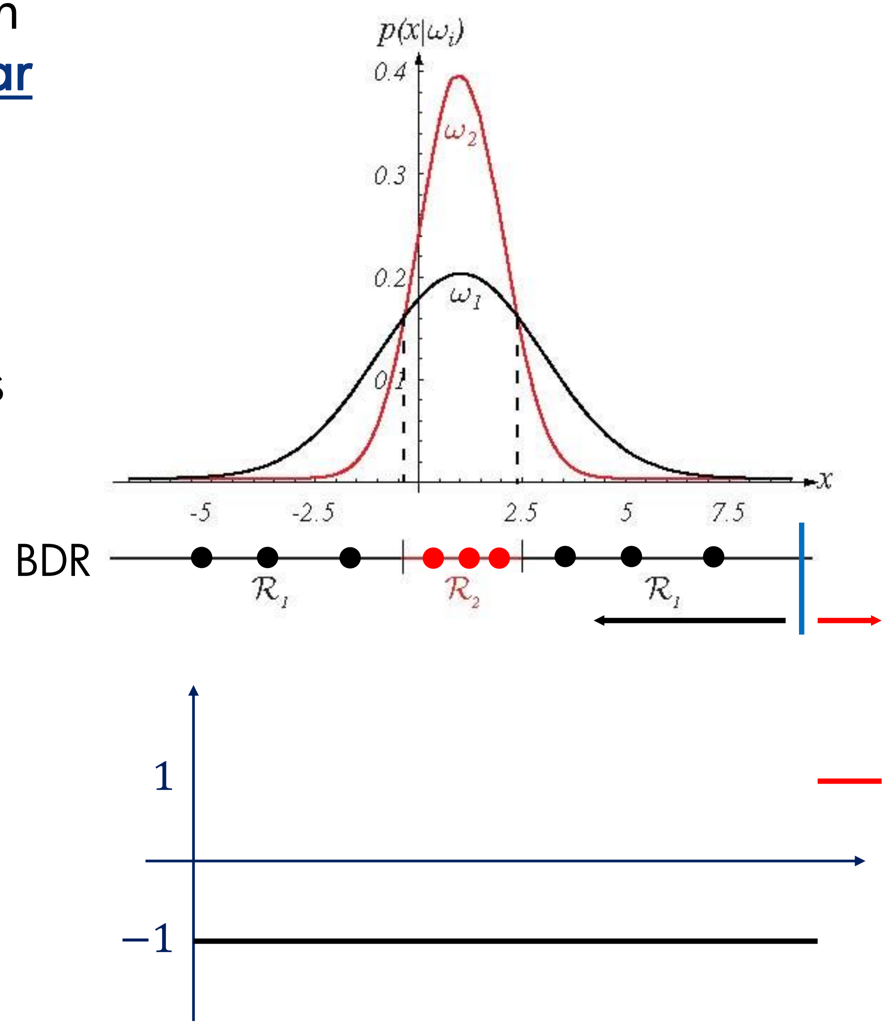
and these are frequently decision stumps

$$\boxed{U = \{u(\mathbf{x}; j, t) \mid j \in \{1, \dots, d\}, t \in T\}}$$

$$\boxed{u(\mathbf{x}; j, t) = \begin{cases} 1, & x_j \geq t \\ -1, & x_j < t \end{cases}}$$

Boosting at Work

- note that boosting works even when the boundaries are **quite** **non−linear**
- example
  - scalar x
  - Gaussian problem, different σ's
- **iteration 1:**



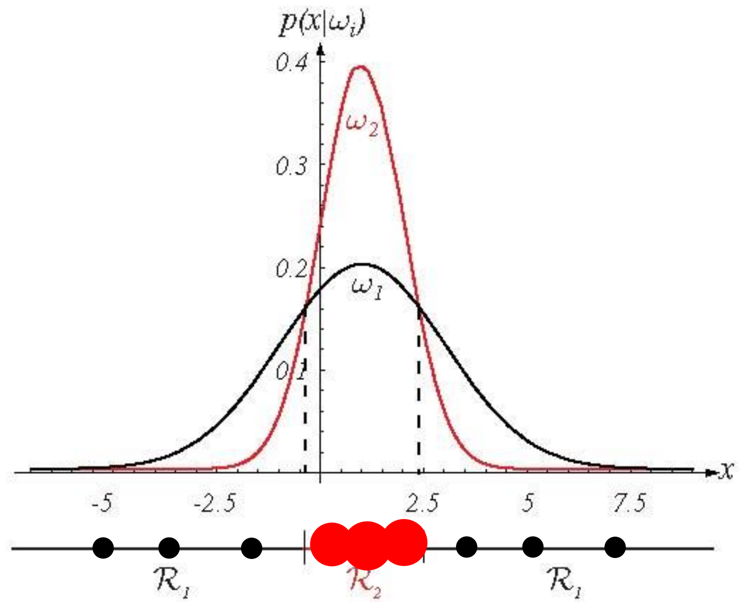
  - all points have **same weight**

$$\alpha_t = \arg\min_{j,t} \sum_{i|y_i \neq u(\mathbf{x}_i;j,t)} \varpi_i$$

  - minimum is 3 errors

Boosting at Work

- note that boosting works even when the boundaries are **quite** **non−linear**
- example
  - scalar x
  - Gaussian problem, different σ's



- **iteration 2:**
  - after weight updates

$$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

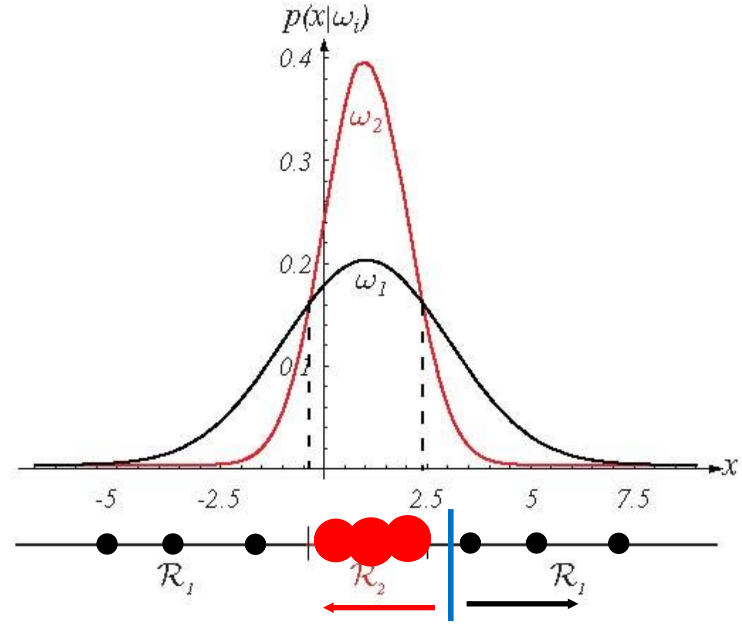
  - red points (points in error by current classifier) get heavier

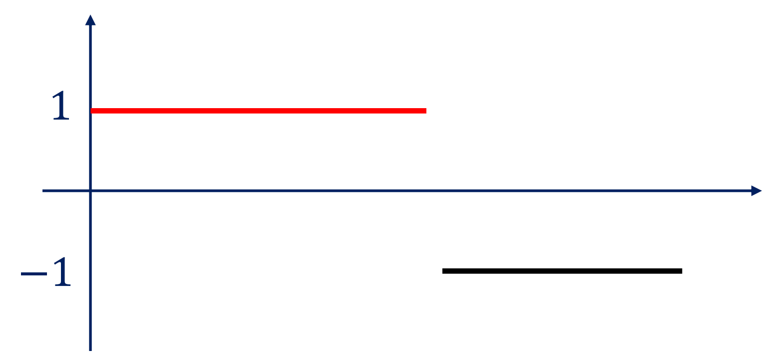
Boosting at Work

- note that boosting works even when the boundaries are **quite** <u>**non−linear**</u>
- <u>example</u>
  - scalar x
  - Gaussian problem, different σ's
- **iteration 2:**
  - assuming each <u>**black**</u> error count $1/3$,

$$\alpha_t = \arg\min_{j,t} \sum_{i|y_i \neq u(\mathbf{x}_i;j,t)} \varpi_i$$

  - minimum error is 1



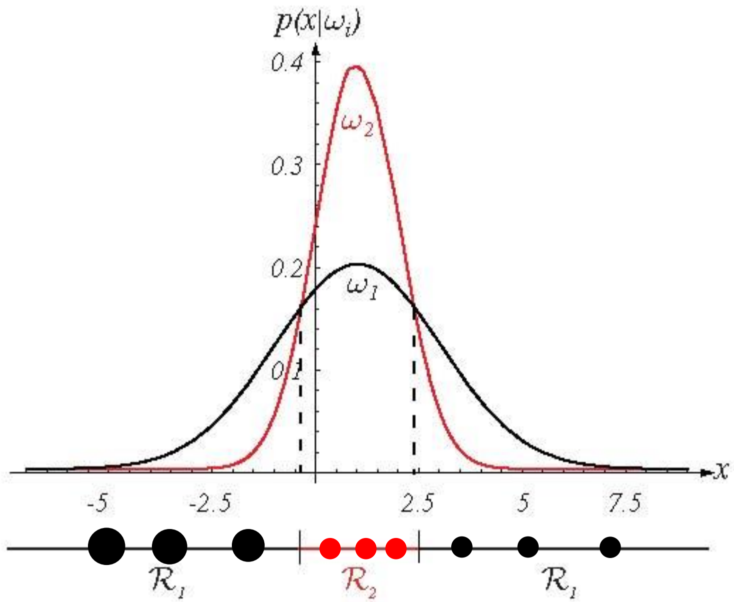


Boosting at Work

- note that boosting works even when the boundaries are **quite <u>non−linear</u>**
- <u>example</u>
  - scalar x
  - Gaussian problem, different σ's
- iteration 3:
  - after weight updates

$$\varpi_i = \exp\left[-y_i \, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

  - some <u>black points</u> get <u>heavier</u>



Boosting at Work

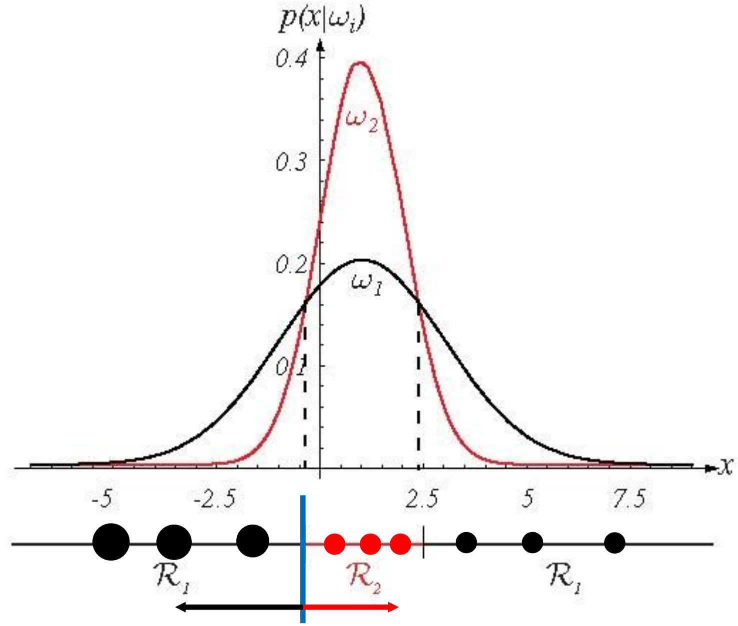
- note that boosting works even when the boundaries are **quite <u>non–linear</u>**
- <u>example</u>
  - scalar x
  - Gaussian problem, different σ's
- **iteration 3:**
  - we get the third threshold

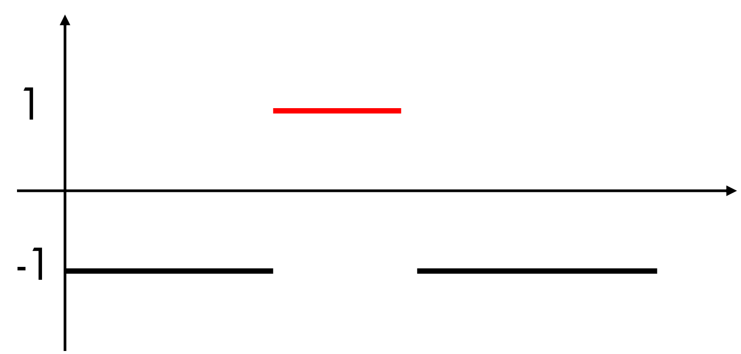
$$\alpha_t = \arg\min_{j,t} \sum_{i|y_i \neq u(\mathbf{x}_i;j,t)} \varpi_i$$

  - decision rule is something like this

$$g^{(3)}(\mathbf{x}) = g^{(2)}(\mathbf{x}) + w_2\alpha_2(\mathbf{x})$$

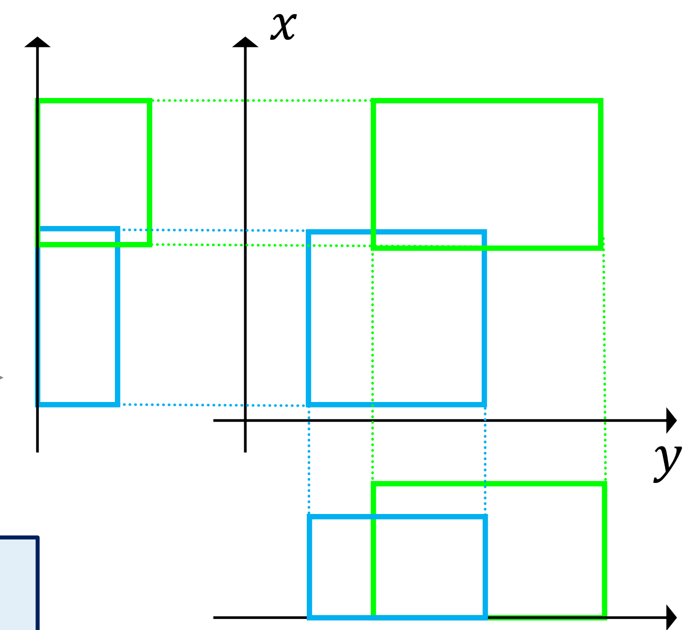
$$h(\mathbf{x}) = \mathrm{sgn}\left[g^{(3)}(\mathbf{x})\right]$$





Boosting as Feature Selection

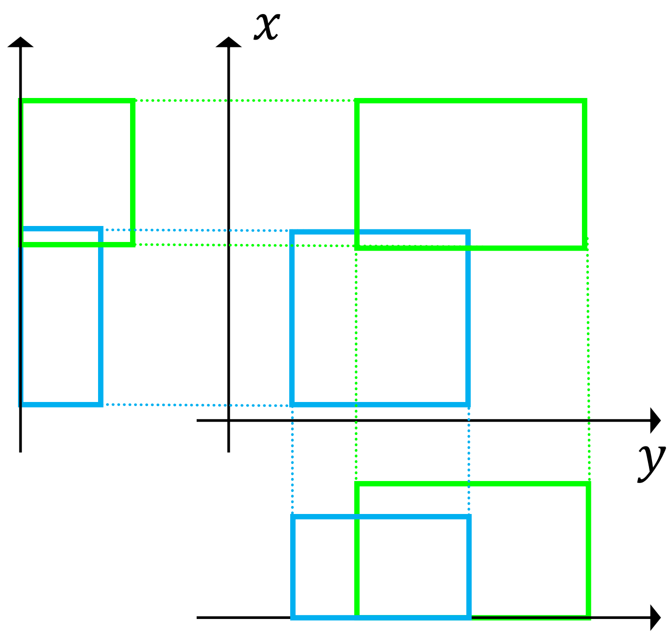
- AdaBoost with decision stumps can be seen as a feature selection method
- at each round
  - select the most discriminant feature (one that best separate the classes)
  - here, x would be selected first



- note that the feature selection is
  - performed jointly with classifier design
  - explicitly optimal in terms of minimizing classification error
- this is a significant advantage over classical methods (that select features and then design classifier)

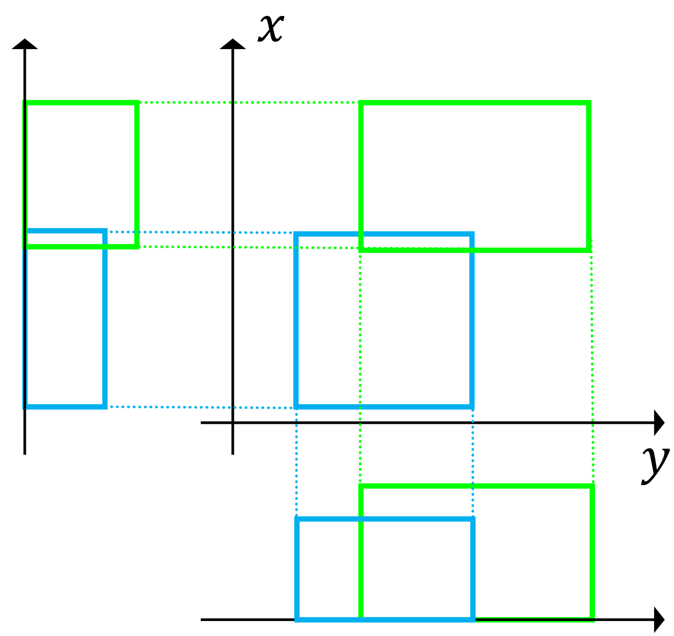
Boosting as Feature Selection

- in fact, boosting is <u>very</u> **smart** feature selection
- as we saw, <u>feature selection</u> requires
  - discrimination
  - independence
- how can we do this by looking at <u>one</u> feature at a time?
- we do <u>**not**</u> want copies of the **same** feature, even if it is **discriminant**
  - think of a problem with 500 features, 300 xs and 200 ys
  - once we **picked** x, there is <u>no</u> point in **picking** x **again**
  - it would <u>not</u> **add** anything to our classifier
  - more generally, we want the features to be as <u>independent</u> as possible



Boosting as Feature Selection

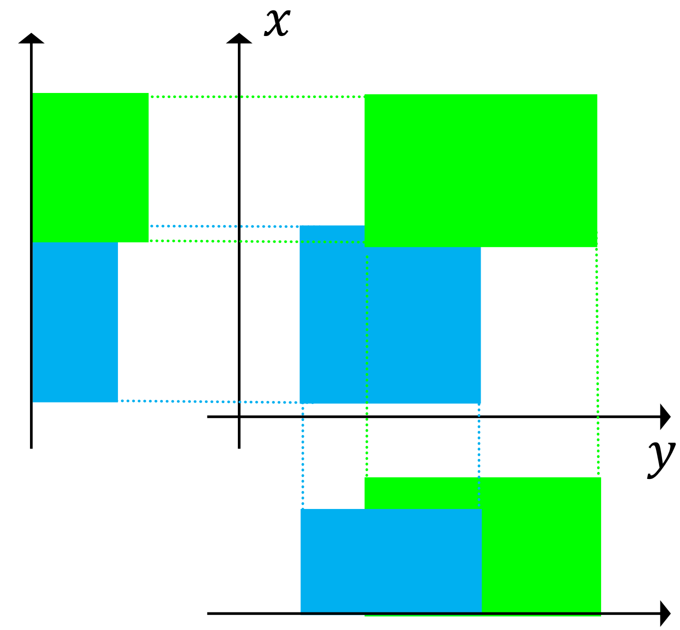
- hence
  - there is a tension
  - features correlated with the **most discriminant** are likely **to be discriminant**
  - they need to be penalized
  - this is really what the **reweighting** is accomplishing



- after the **first iteration**
  - all points **well** classified along 1st feature are **downgraded**
  - features correlated with 1st feature will **no** longer be **discriminant**
  - all the points **left** are points where the feature does **poorly**
- once again, this is done **optimally** with respect to **minimizing classification error**!

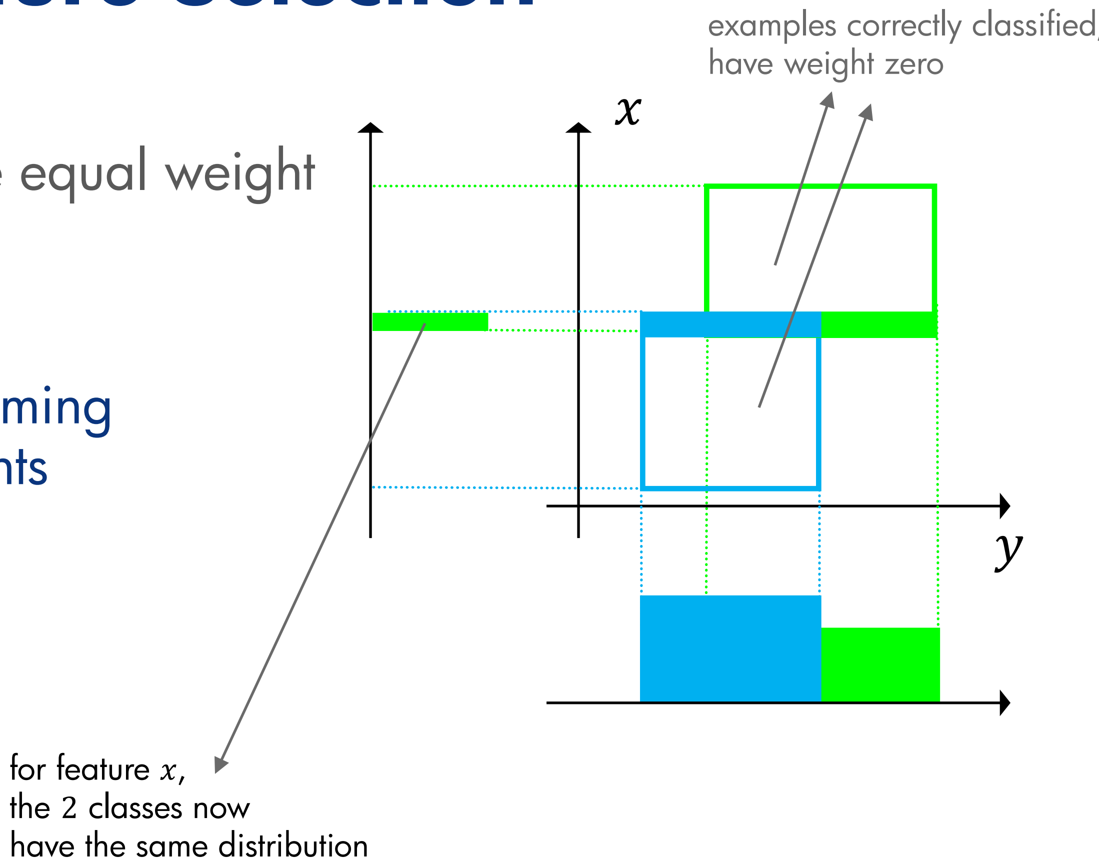
Boosting as Feature Selection

- in the example
  - initially, all points have equal weight
  - x is most discriminant, picked first



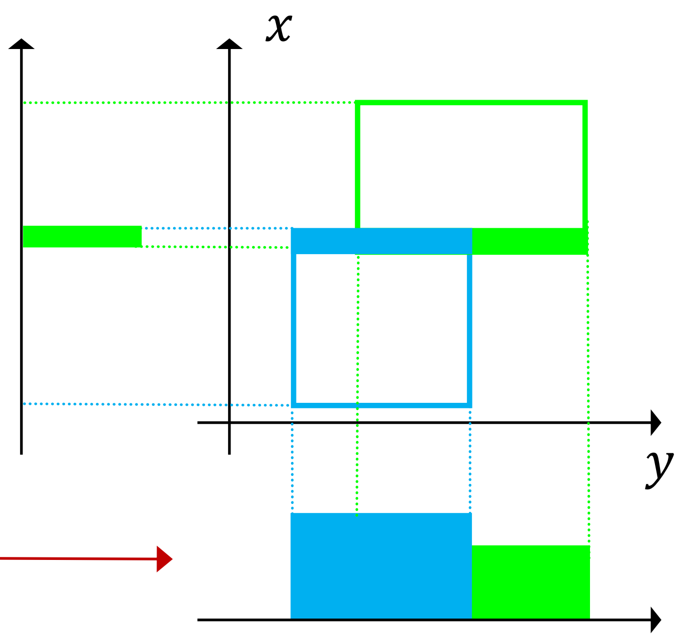
Boosting as Feature Selection

- in the example
  - initially, all points have equal weight
  - x is most discriminant, picked first
  - after reweighting (assuming correctly classified points get zero weight)



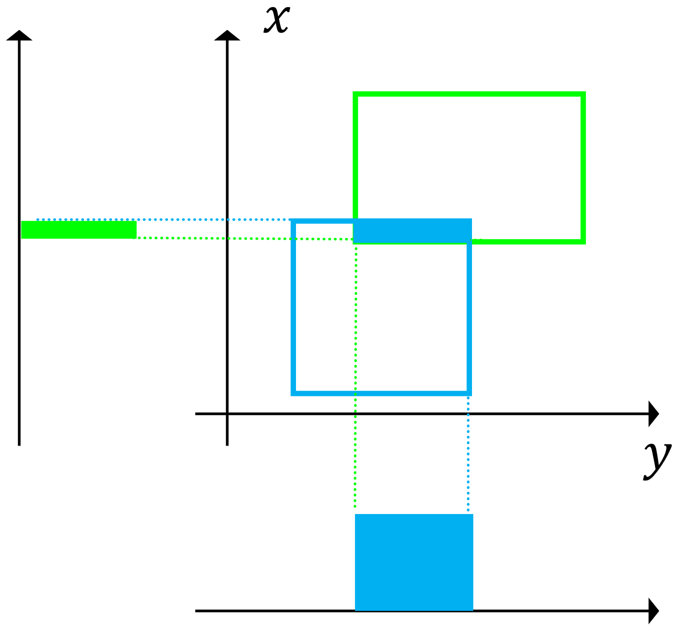
Boosting as Feature Selection

- in the example
  - initially, all points have equal weight
  - x is most discriminant, picked first
  - after reweighting (assuming correctly classified points get zero weight)
  - y is now more discriminating, and is picked as second feature



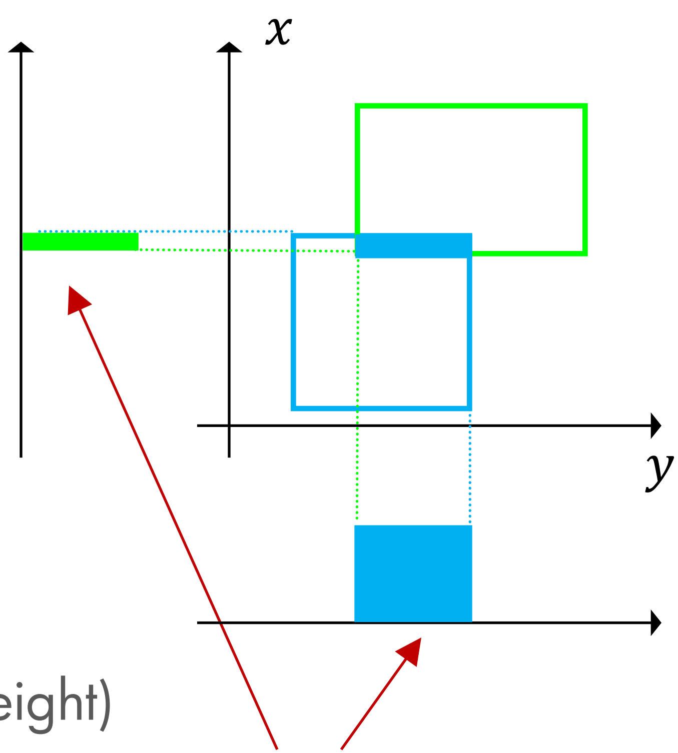
Boosting as Feature Selection

- in the example
  - initially, all points have equal weight
  - x is most discriminant, picked first
  - after reweighting (assuming correctly classified points get zero weight)
  - y is now more discriminating, and is picked as second feature
  - after reweighting (assuming correctly classified points get zero weight)



Boosting as Feature Selection

- in the example
  - initially, all points have equal weight
  - x is most discriminant, picked first
  - after reweighting (assuming correctly classified points get zero weight)
  - y is now more discriminating, and is picked as second feature
  - after reweighting (assuming correctly classified points get zero weight)
  - both features are now **equally bad**, not much more to choose, boosting will look for other features



- **overall:**
  - x is **always** available and could be picked up again
  - reweighting **penalizes the replicas**!

Boosting: Connections to Regularization

- what about regularization, SRM, and all that?
- boosting has no explicit regularizer, but an implicit one
- number M of weak learners (iterations)
  - as M increases, the classifier becomes more complex
  - without a limit on M, boosting will overfit
  - the limit on M can be seen as regularizer
  - this is really L_0 regularization

$$h(\mathbf{x}) = \operatorname{sgn}[g(\mathbf{x})]$$

$$g(\mathbf{x}) = \sum_{i=1}^{M} w_i \alpha_i(\mathbf{x})$$

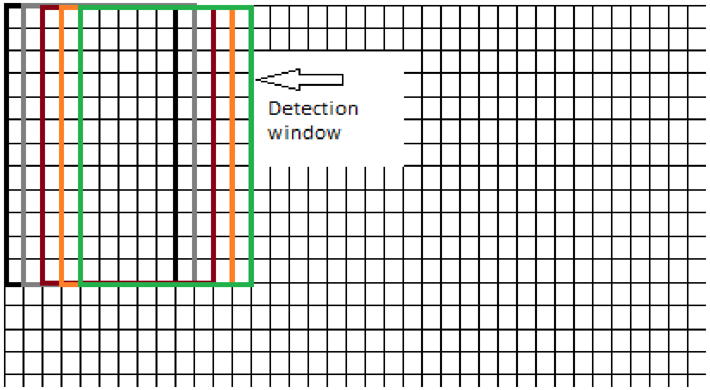
$$\|\mathbf{w}\|_0 = \# \{w_i > 0\}$$

  - by limiting the number of iterations M, we can effectively implement regularization
  - in practice, M is the "parameter" that you control
  - note there are no other parameters in boosting!
    (e.g., unlike NNs – how many layers, how many units, etc.?)

Detector Cascades

- boosting became extremely popular in **computer vision** due to its **success** in the problem of **object detection**

- this consists of
  - slide a window over the image
  - extract a patch at each location
  - use a classifier to detect the presence or absence of the object

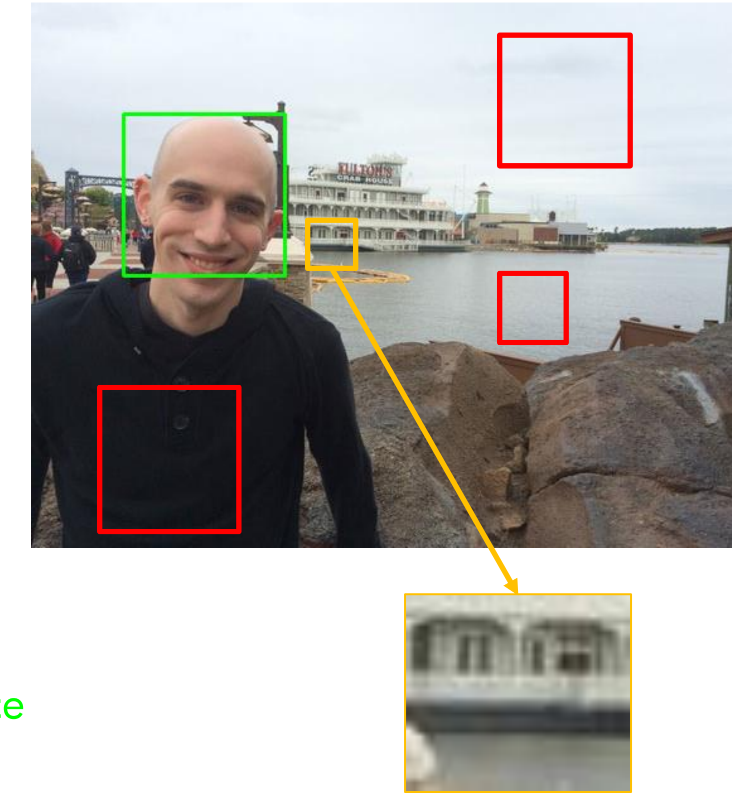


- difficulty
  - since we do **not** know where object is, must be **repeated** for many window sizes (object scales)
  - millions of images must be classified per image
  - for video, we need to do this $10 - 30$ frames per second
  - **Viola and Jones** (VJ) proposed a detector cascade

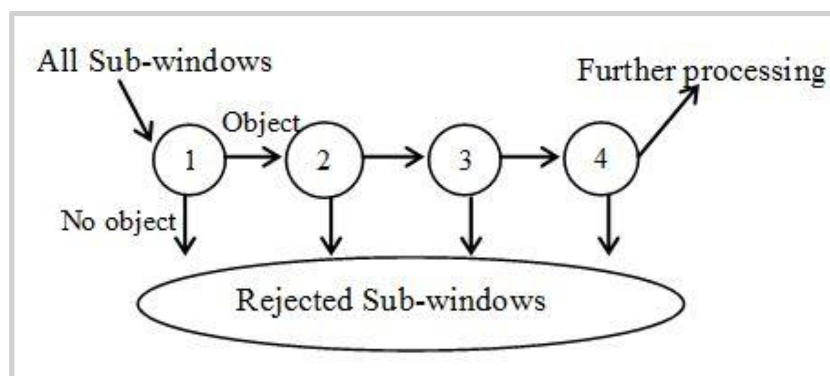
Viola, Paul; Jones, Michael; Robust Real–Time Object Detection. *International Journal of Computer Vision*, 2001.

Detector Cascades

- idea:
  - consider face detection
  - many windows can easily be classified as non–faces
    - a very simple classifier is sufficient to reject them
  - other windows are more face–like
    - rejecting them requires a more complicated classifier
  - finally, to be sure that a window contains a face
    - we need a really good classifier



- the detector cascade is implemented as a sequence of classifier stages
  - stage 1 has very **low complexity**
    - **rejects** obvious non–faces
  - complexity **increases** for later stages
  - note that final stages are **rarely** used
  - overall complexity is low!

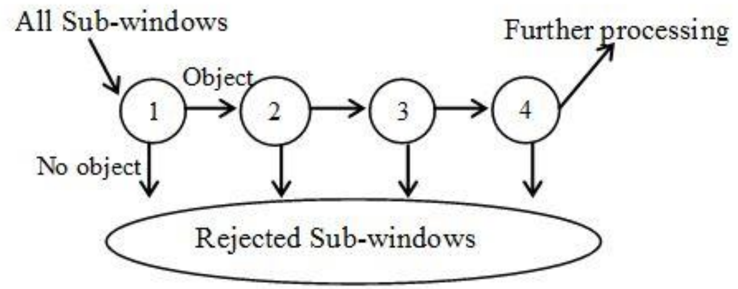


Detector Cascades

- an **ensemble classifier**

$$h(\mathbf{x}) = \text{sgn}[g(\mathbf{x})]$$

$$g(\mathbf{x}) = \sum_i w_i \alpha_i(\mathbf{x})$$



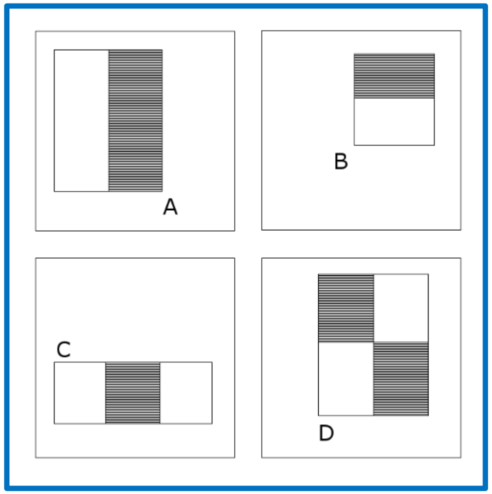
- is great for this because we can just create intermediate "exit points"
- this consists of creating a **sequence of classifiers**

$$\boxed{h_k(\mathbf{x}) = \text{sgn}[g_k(\mathbf{x})]} \qquad \boxed{g_k(\mathbf{x}) = \sum_{i=1}^{N_k} w_i \alpha_i(\mathbf{x})}$$

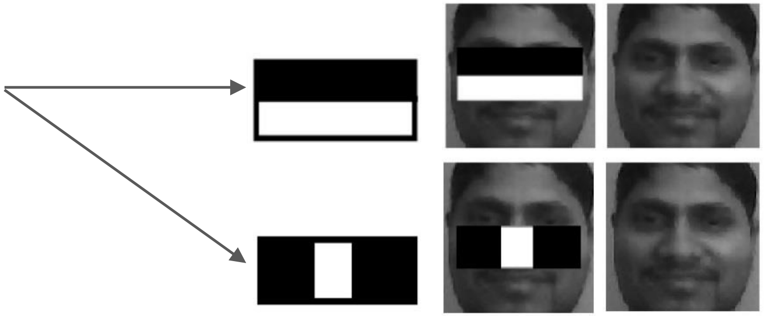
- the k^{th} stage is a classifier based on the first N_k weak learners
- e.g., $N_1 = 1, N_2 = 5, N_3 = 10,$ implements a cascade where
  - stage 1 has 1 weak learner, stage 2 has 5, and stage 3 has 10
  - assume that the stages 1, 2 reject 50%, 90% of windows, respectively
  - complexity is (.1x10+.5x5+1x1) C = 4.5C < 16C, where C = complexity of WL

Detector Cascades

- in the VJ cascade
  - weak learners are decision stumps on 4 type of Haar features
  - these features are very efficient to compute
  - they are boxes of value $+1$ or -1
  - feature evaluation consists of summing pixels inside positive boxes and subtracting pixels inside negative ones
  - VJ introduced an image processing trick – the **integral image** – that allows the computation of the summations inside each box with 4 additions
  - hence, the features can be implemented with 8 (A and B), 12 (C), and 16 (D) additions
  - note that this does not depend on the window size!



- these features are also very good for face detection
  - they match patterns like eyes and nose
  - a detector with two weak learners (20 additions) can confidently reject many non–faces
  - on average, it can detect 100% of the faces, while rejecting 50% of the non–faces
  - cascade of a **few stages** can reject 90% of the non–faces without loosing any face



Detector Cascades

- the VJ cascade detector achieves **high accuracy** for real–time classification

(original at https://www.youtube.com/watch?v=pZi9o-3ddq4)

- it has became **widely used** (your smart phone is probably running it)