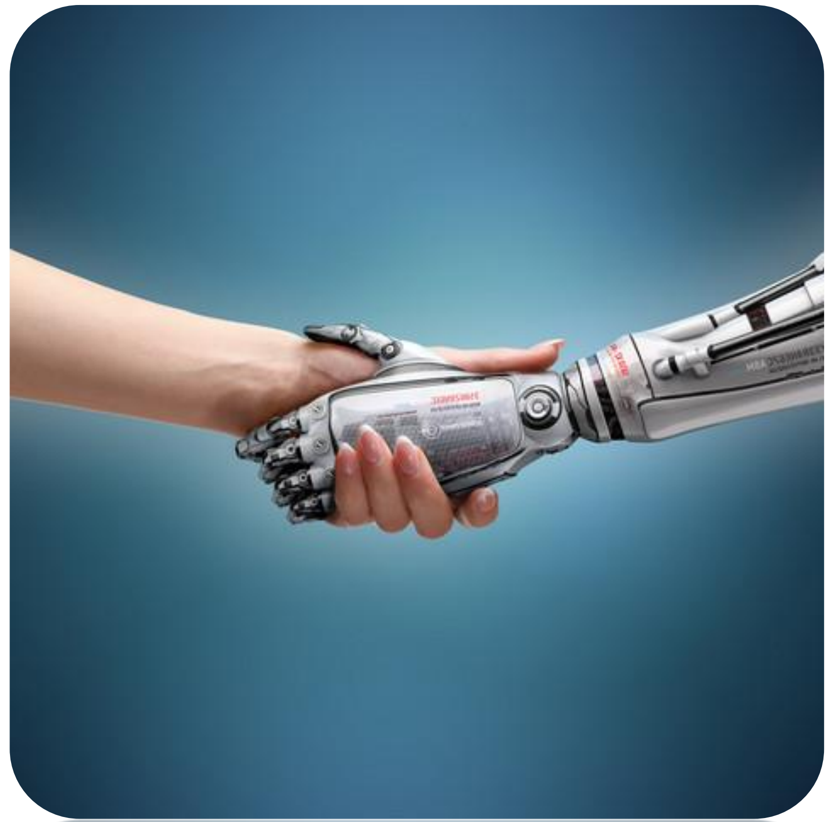
Zurich University
of Applied Sciences

Artificial Intelligence
V09: Ensemble Learning

Ensembles of classifiers
Boosting
A pattern recognition example



Based on material by
- Jin Tian, Iowa State University
- Cheng Li, Northeastern University
- Jason Brownlee, Machine Learning Mastery
- Yu Wen, Tatung University

Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

Educational objectives

- **Remember** the **AdaBoost** algorithm **and** its distinction from **Bagging**
- **Explain** how **Boosting** can be seen **as** a form of **gradient descent** and what benefit this viewpoint has
- **Use** **current implementations** of decision tree ensembles (e.g., Random Forest®, XGBoost) for machine learning tasks

„In which we see that combining many weak agents can result in a very strong one.“

➔ Reading: AIMA, ch. 18.10-18.12

1. ENSEMBLES OF CLASSIFIERS

Ensembles
Combining many weak agents to form a strong one

Ensembles in a nutshell

same or different $\mathcal{H}$

- Goal: **Combining** multiple **complementary classifiers** to increase performance
- Idea: Build different "experts", and **let them vote**

Pros & cons

✓ Very **effective** in practice
✓ Good **theoretical guarantees**
✓ **Easy** to **implement**, not too much parameter tuning
✗ The result is not so transparent (**black box**)
✗ **Not** a **compact** representation

Formal problem description

- Given T binary classification hypotheses $(h_1,\ldots,h_T)$, find a combined classifier with better performance of the form

$$\hat{h}(x) = sgn\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

assuming classes -1/+1

majority vote

individual weight

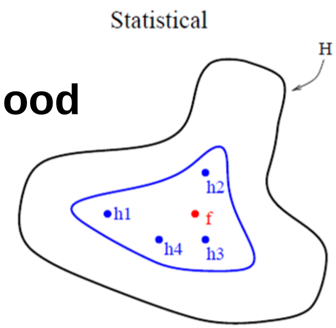
Why do they work?

Three fundamental reasons why ensembles *may* be beneficial

We *cannot* **know** the best ➔ so we average

Statistical

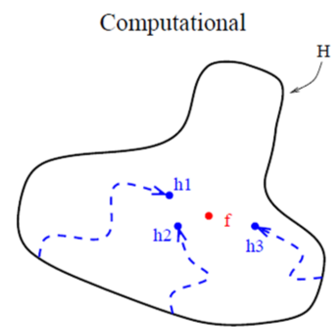
- Given **finite** amount of **data**, many hypothesis typically **appear equally good**
- **Averaging** may be a better approximation to the true f



We *may not* **find** the best ➔ so we average

Computational

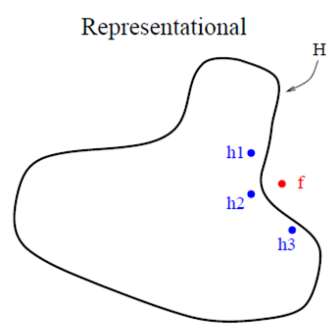
- **Search** for h **is heuristic** due to interesting $\mathcal{H}$'s being huge/infinite
- Strategy to **avoid local minima**:
  repeat with random restarts, construct an ensemble



We *cannot* **find** the best ➔ so we average

Representational

- The desired **target function may not be realizable** using individual classifiers from $\mathcal{H}$
- It may be **approximated by** ensemble **averaging**



Example: Bagging
Majority vote over bootstrapped training data

Bootstrap Aggregating [Breiman, 1996]

- Idea: Design the ensemble to be **as diverse as possible**
  ➔ Assert that this ensures **complementary** learners
- Almost always **improves** results if base learner is **unstable**
  (i.e., classification changes with slightly different training data)

Algorithm

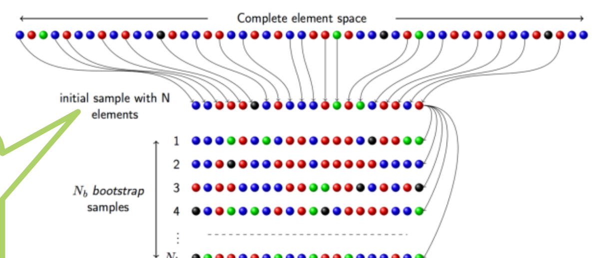
- The process is remarkably **simple** (also to implement)

```
for t := 1..number of ensemble members T
    X_t := sample i.i.d. from training data X with replacement #bootstrap samples
    h_t := train any algorithm on X_t
return ĥ := sgn(∑_{t=1}^{T} 1 · h_t(x))  #majority vote
```

$$\hat{h} := sgn\left(\sum_{t=1}^{T} 1 \cdot h_t(x)\right)$$

- Usually, **the more** ensemble members, **the better**

Statistical reasoning: *"If the [training data] is a good approximation of the population, the bootstrap method will provide a good* ***approximation of*** *the sampling distribution* [~**variability** of a statistic **in different samples** from population]*".* → see R. Vitillo, https://robertovitillo.com/2015/03/, 2015

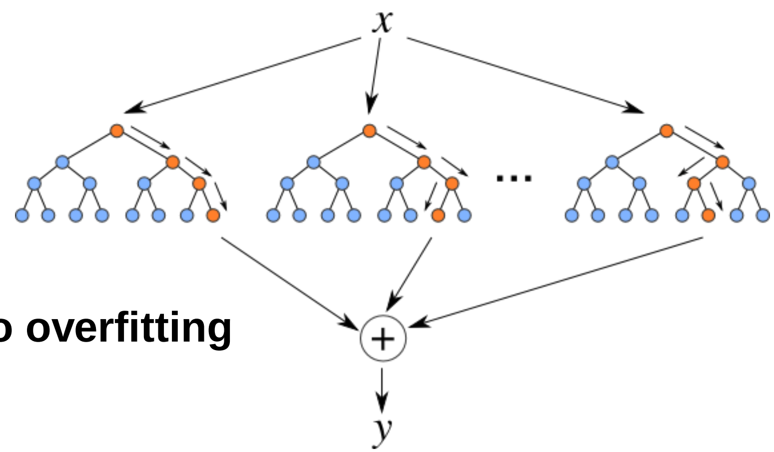


Zurich University of Applied Sciences

Random Forest®
A brief description

Build a majority-voting **ensemble of decision trees**; for each tree,

- Choose a **stratified training set** of n out of N instances by sampling **with replacement**
- At every level,
  - choose a **random feature set** (**with replacement**) of m out the p attributes
  - choose **the best** split among those attributes
- **No pruning** of the branches takes place



Advantages

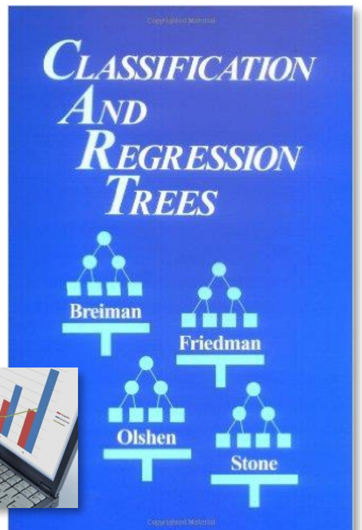
- Fast training, parallelizable application
- High independence of base classifiers ➔ nearly **no overfitting**
- Few hyper parameters
- Applicable to large quantities of N, p and #classes

➔ **Very good out-of-the-box method**

Further reading

- [Breiman 2001]: *«Random Forests»*. Machine Learning 45 (1), 5-32

2. BOOSTING



1983: BI

2016: Data Science

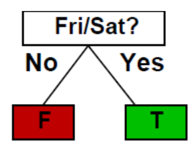
Foundations of boosting

General idea

- **Boost** the performance of weak learners (error slightly >chance)
- **Make** currently **misclassified examples** more **important**, then combine hypotheses
  ➔ Each **stage** (additively) corrects shortcomings of previous stage by **reweighting**, then **majority vote**
- Origins in computer science: [Kearns & Valiant, 1988] (as opposed to Bagging: statistics)

Adaptive Boosting algorithm [Freund & Schapire, 1997]

- Weak learner: decision stump (=decision tree of height 1; but generalizable to others)
  ➔ Important: weak learners have skill but remain weak (to not lose the ensemble effect)



initialize weights: $w_i \coloneqq \frac{1}{N}$ #each sample gets same weight

for $t := 1..T$

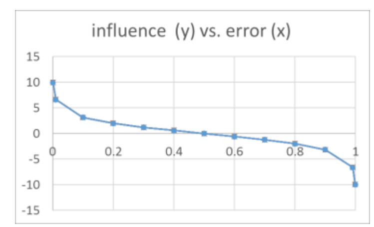
  $h_t \coloneqq$ train decision stump on the x_i, weighted by the w_i

  $\varepsilon_t \coloneqq \frac{\sum_{i=1}^{N} w_i \cdot I(y_i \neq h_t(x_i))}{\sum_{i=1}^{N} w_i}$ #compute **error**; $I()$ is the identity function

  $\alpha_t \coloneqq \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ #compute **influence** of weak learner

  $w_i \coloneqq w_i \cdot e^{\alpha_t \cdot I(y_i \neq h_t(x_i))}$ #increase weight by exp(influence) in case of error

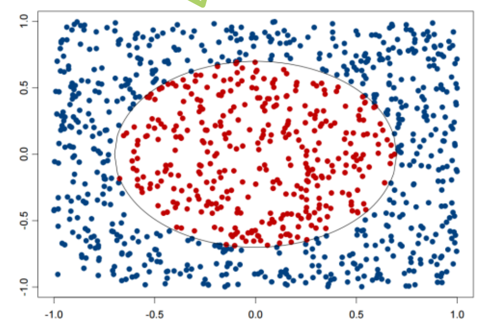
return $\hat{h} \coloneqq sgn(\sum_{t=1}^{T} \alpha_t \cdot h_t(x))$ #weighted majority vote

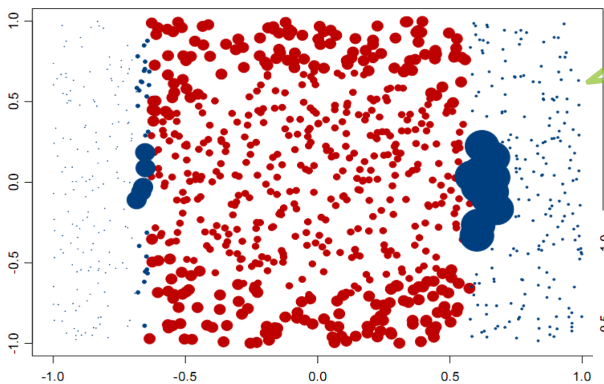


AdaBoost in practice

Based on Seni & Elder, *«From Trees to Forests and Rule Sets – A Unified Overview of Ensemble Methods», KDD 2007*

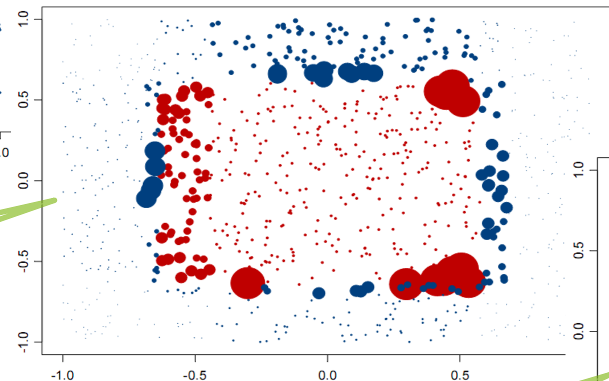
Goal



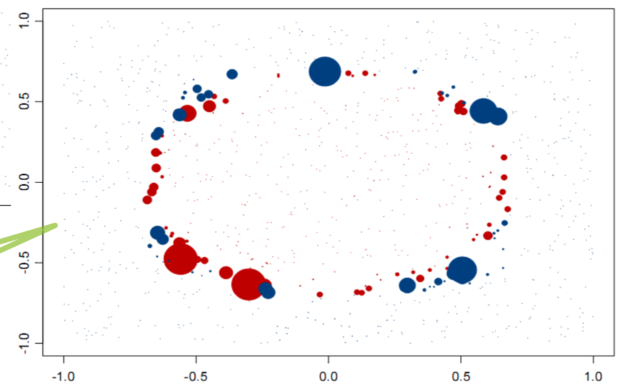


AdaBoost's classifications (colors) and weights (size) after **1 iteration** ➔ still looks like a single decision tree with rectangular decision boundary

3 iterations



20 iterations



Pros & cons

- ✓ Very **little code**
- ✓ **Improves capacity** of weak (base) learner (thus combating underfitting)
- ✓ Still learns when others overfit ➔ **margin optimization**
- × Sensitive to noise and outliers

From AdaBoost to gradient boosting

Recall: In **AdaBoost**, ”**shortcomings**" are identified by high-**weight** data points

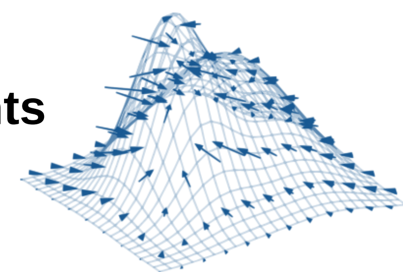
A brief history of modern boosting (selective, shortened)

1. Invention: **AdaBoost**, the **first** successful boosting **algorithm**
   [Freund et al., 1996], [Freund & Schapire, 1997]
2. Translation: **Formulation as gradient descent** with special loss function (→ compare V04)
   [Breiman et al., 1998], [Breiman, 1999]
3. Generalization: Gradient boosting in order **to handle** a **variety of loss functions**
   [Friedman et al., 2000], [Friedman, 2001]

➔ For a great example of cross-disciplinary fertilization, see
Breiman, *“Arcing classifiers (with discussion and a rejoinder by the author)”*, 1998

In **gradient boosting**, “**shortcomings**" are identified by **gradients**

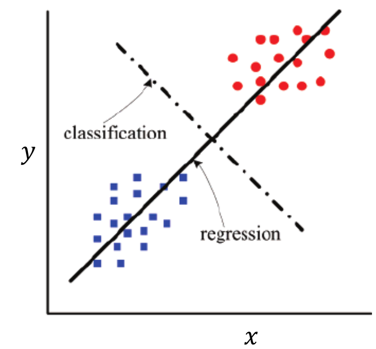
- Gradients of what? Why? → see next slides



Intuition for gradient boosting

Setup

- For ease of discussion we change the setting from (binary) classification to regression (i.e., real-valued labels)
- Results are again applicable to classification
(but not intuitively as straight-forward)



Let's play a game

- You are given data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ and **the task to fit model** $\hat{h}(x)$
➔ minimize squared loss $\ell\big(y, h(x)\big) = \frac{1}{2}(y - h(x))^2$
- Suppose a friend helps by giving you an **initial model** $F(x)$ (a regression tree)
➔ You check his model and find the model is good but not perfect (e.g. $F(x_1) = 0.8$ while $y_1 = 0.9$)
- Rule: $F(x)$ must **not be changed** in any way, but another model might be added
➔ i.e. $\hat{h}(x) = F(x) + h(x)$
- How to train $h(x)$?

We want this to be true

Equivalently, we can fit the new regression tree h to:

$$F_1(x_1) + h(x_1) = y_1 \quad ➔ \quad h(x_1) = y_1 - F(x_1)$$

$$\vdots$$

$$F(x_N) + h(x_N) = y_N \quad ➔ \quad h(x_N) = y_N - F(x_N)$$

Intuition for gradient boosting (contd.)

Simple ensemble solution

- The $y_i - F(x_i)$'s are called residuals
  ➔ These are the parts that the initial model F cannot do well
  ➔ The role of h is to compensate the shortcomings of F
- If the new model $F + h$ is still not satisfactory, we can add another regression tree...

How is this related to gradient descent?

w.r.t J's parameters θ

- Gradient Descent in general: **minimize** a function J **by moving** into **opposite direction** of **gradient**

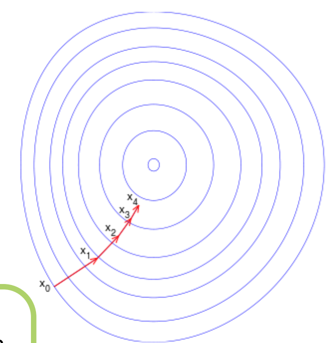
$$\theta_i^{new} = \theta_i^{old} - \alpha \frac{\partial J}{\partial \theta_i^{old}}$$

i.e., $J = L$

assuming mean squared error loss, the standard for regression

- Earlier: wanted to **minimize loss** function $\mathrm{L} = \sum_{i=1}^{N} \ell\big(y_i, F(x_i)\big) = \sum_{i=1}^{N} \frac{1}{2}\big(y_i - F(x_i)\big)^2$
  ➔ $F(x_i)$ is the parameter of L, so we take derivatives w.r.t. $F(x_i)$:

$$\frac{\partial L}{\partial F(x_i)} = \frac{\partial \ell\big(y_i, F(x_i)\big)}{\partial F(x_i)} = F(x_i) - y_i$$



- That is: We can **interpret residuals** for $h(x)$ **as negative gradients** for improving $F(x_i)$
  ➔ $y_i - F(x_i) = -\frac{\partial L}{\partial F(x_i)}$

compare
$$\begin{aligned} F^{new}(x_i) &= F^{old}(x_i) + h(x_i) \\ &= F^{old}(x_i) + y_i - F^{old}(x_i) \\ &= F^{old}(x_i) - 1 \cdot \frac{\partial L}{\partial F^{old}(x_i)} \end{aligned}$$

with: $\theta_i^{new} = \theta_i^{old} - \alpha \frac{\partial J}{\partial \theta_i^{old}}$

Improving $F(x_i)$ with $h(x_i)$ or by gradient descent on its parameters is equivalent ➔ doing many iterations of gradient descent is equivalent to many rounds of boosting as well.

Gradient boosting of regression trees
(Multiclass classification → see appendix)

Algorithm

- Gradient boosting for regression

```
Start with an initial model, e.g. F = (∑_{i=1}^N y_i)/N (always predict mean value)
repeat until convergence
    −g(x_i) = −∂ℓ(y_i,F(x_i))/∂F(x_i)
    fit regression tree h to −g(x_i)
    F ≔ F + αh #α is a tunable learning rate, e.g. = 1
```

True for $\ell =$ squared loss

- Residual ⇔ negative gradient
- Fit h_i to residual ⇔ fit h_i to negative gradient
- Update h_i based on residual ⇔ update h_i based on negative gradient

➔ So we **are actually** updating our model **using gradient descent**!

Advantage of gradient descent formulation

- Allows **consider**ing **other loss** functions (e.g. more outlier-robust, domain-specific, …)
  ➔ Derive the corresponding algorithms in the same way

XGBoost: A scalable tree boosting system
[Chen & Guestrin, 2016]

A **skillfully engineered**, **highly optimized** implementation
- Used by $17/29$ **winning** teams on **Kaggle** 2015
- Open source (Python, R, Spark, …): https://github.com/dmlc/xgboost
- Scalable: $10\times$ faster than usual implementations, scales to $\sim 10^9$ training points
  - Massive use of parallelization/distribution (e.g. on Hadoop/Spark, but also on desktop)

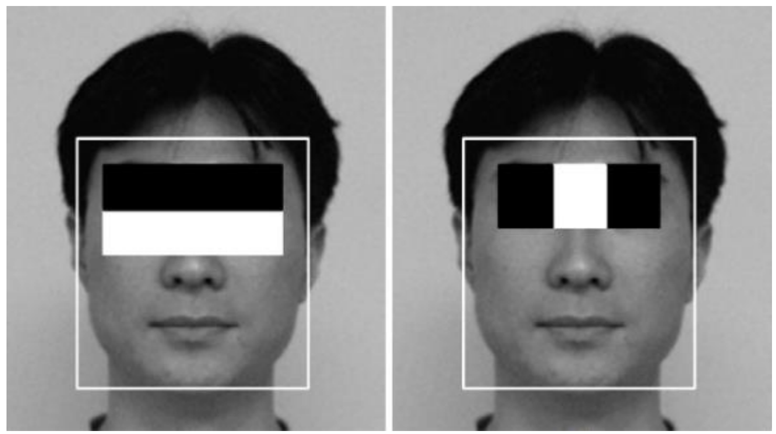
Both types of novelties **purely increase** the **computational performance**, not learning in general

Algorithmic novelties
- Distributed **approximate best split** finding („weighted quantile sketch" using quantile statistics)
- **Exploit sparsity** (induced by missing values/one-hot encoding → via default directions for branching)

Parallelization Cache-aware access (for gradient statistics)
- Efficient **out-of-core computation** (i.e., computation on data not fitting into main memory)

General tricks for tree boosting
- Use aggressive sub-sampling (e.g., selecting only 50% of the data)
- Using column sub-sampling prevents over-fitting even more so than row sub-sampling

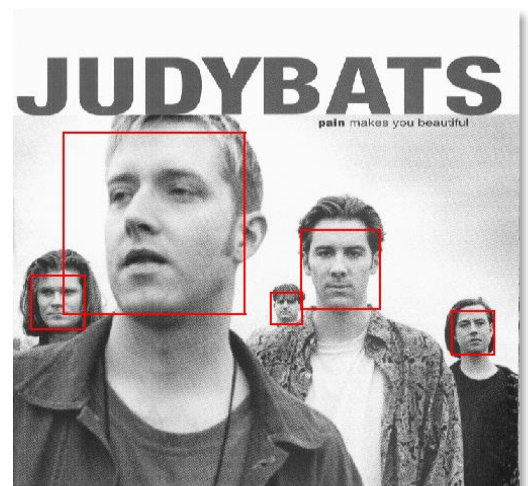
3. A PATTERN RECOGNITION EXAMPLE



AdaBoost for face detection
A detailed example of a boosted decision stumps application

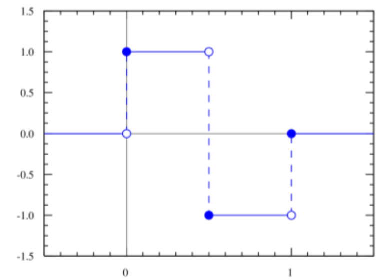
Challenges
- **Slide** a **window** across image and evaluate a face model **at every location & scale**
  ➔ Sliding window detector must evaluate tens of thousands of location/scale combinations
- Faces are **rare**: 0–10 per image
  - For **computational efficiency**, we should try spending as little time as possible on non-face windows
    ➔ A megapixel image has $\sim 10^6$ pixels and a comparable number of candidate face locations
  - To avoid having a false positive in every image, the **false positive rate has** to be **less** than $\mathbf{10^{-6}}$

The Viola-Jones face detector [Viola & Jones, 2001]
- A seminal approach to **real-time object detection**
  ➔Training is slow, but detection is very fast
- Key ideas
  - Integral images for fast feature evaluation
  - **Boosting for** feature selection amongst $\sim 10^5$ candidates
  - Attentional cascade for fast & accurate rejection of non-face windows
    → see appendix



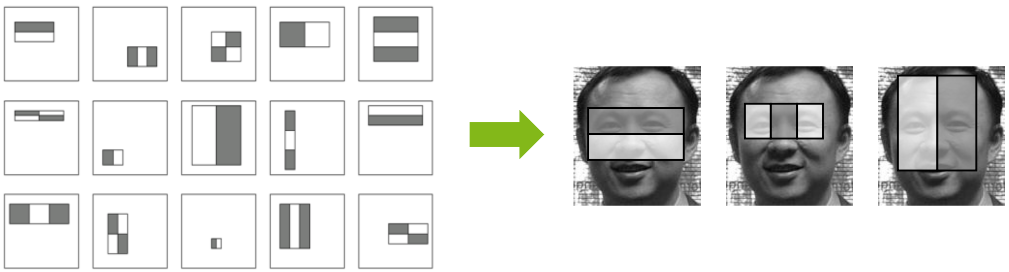
Rectangular facial features

…and their efficient calculation via the integral image



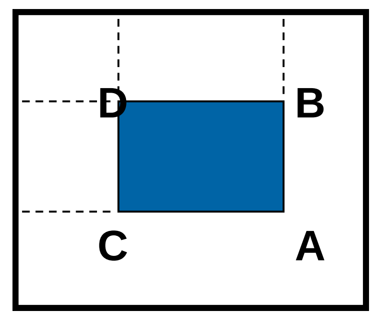
Pixel-based features for face detection

- Reminiscent of Haar wavelets
- Simple **sum of pixel intensities** within rectangular regions resemble typical shading patterns of faces



Integral images (ii)

- Let **each pixel** be the **sum** of all pixels **left** and **above**



Computing sums of pixels within a rectangle using ii

- $sum = ii_A - ii_B - ii_C + ii_D$
- Needs **only 3 additions** for any size of rectangle (**constant time**)

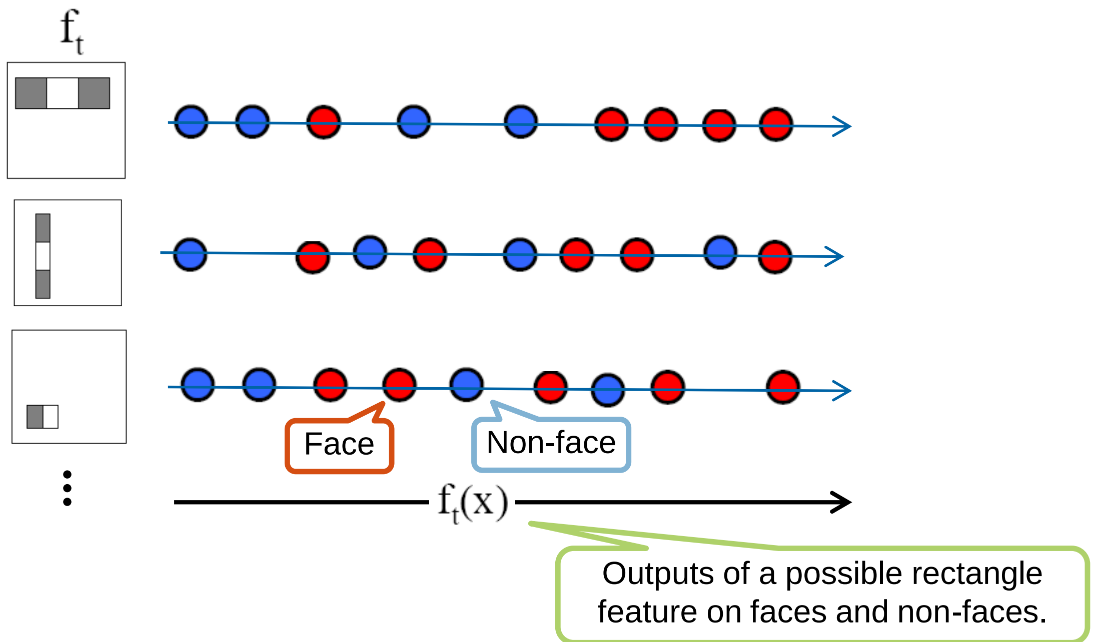
Feature selection via AdaBoost
Slide adapted from Grauman & Leibe's AAAI'08 tutorial

Size of feature space

- Ca. **160'000 distinct** rectangular features **per** detection **window** (via scaling/translation)
  ➔ Which ones are **good?** What is a good **subset?**

Finding a good succession of features

- **Start**: Select the **single** rectangle **feature** & **threshold** that best separates faces/non-faces



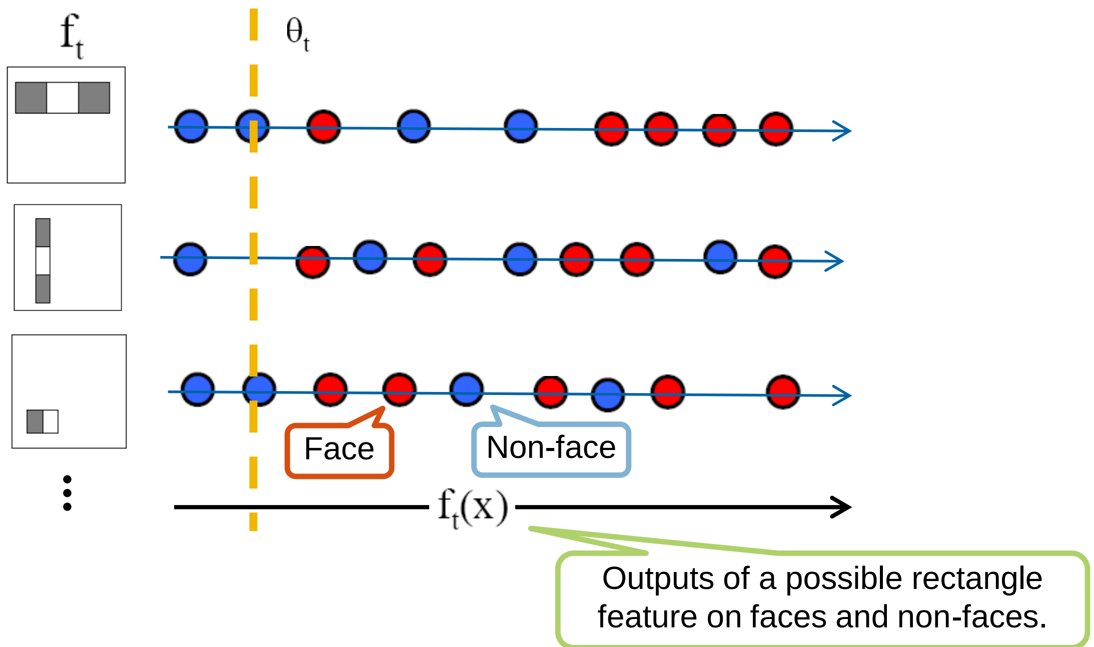
Feature selection via AdaBoost
Slide adapted from Grauman & Leibe's AAAI'08 tutorial

Size of feature space

- Ca. **160'000 distinct** rectangular features **per** detection **window** (via scaling/translation)
  ➔ Which ones are **good?** What is a good **subset?**

Finding a good succession of features

- **Start**: Select the **single** rectangle **feature** & **threshold** that best separates faces/non-faces



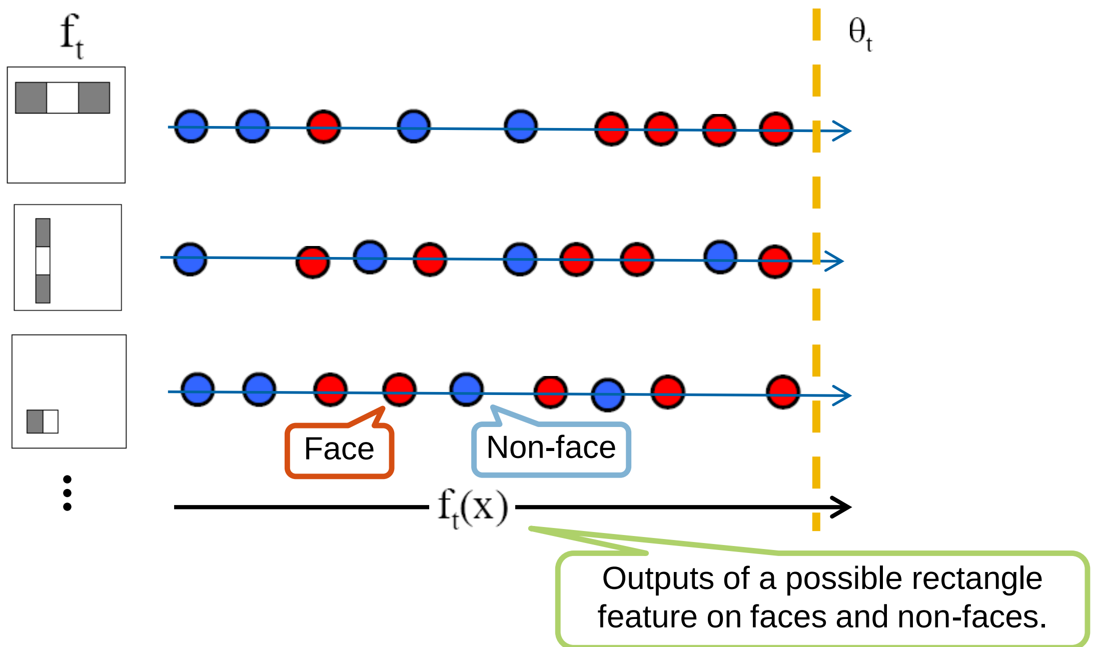
Feature selection via AdaBoost
Slide adapted from Grauman & Leibe’s AAAI’08 tutorial

Size of feature space

- Ca. **160‘000 distinct** rectangular features **per** detection **window** (via scaling/translation)
  ➔ Which ones are **good?** What is a good **subset?**

Finding a good succession of features

- **Start**: Select the **single** rectangle **feature** & **threshold** that best separates faces/non-faces



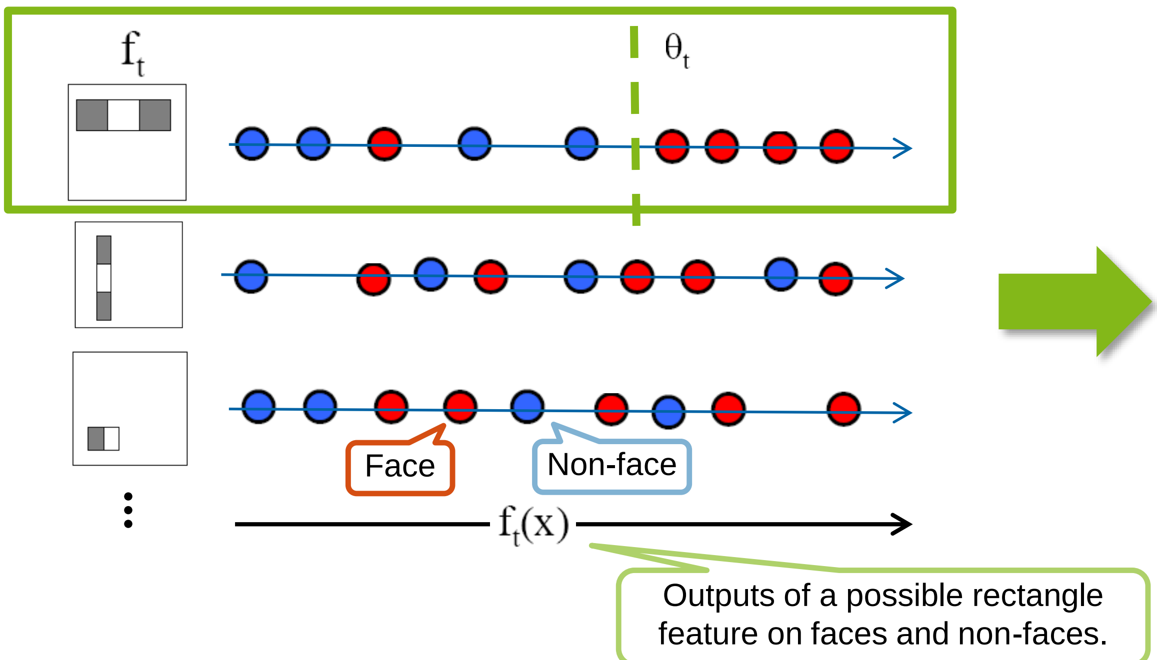
Feature selection via AdaBoost
Slide adapted from Grauman & Leibe's AAAI'08 tutorial

Size of feature space

- Ca. **160'000 distinct** rectangular features **per** detection **window** (via scaling/translation)
  ➔ Which ones are **good?** What is a good **subset?**

Finding a good succession of features

- **Start**: Select the **single** rectangle **feature** & **threshold** that best separates faces/non-faces



Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

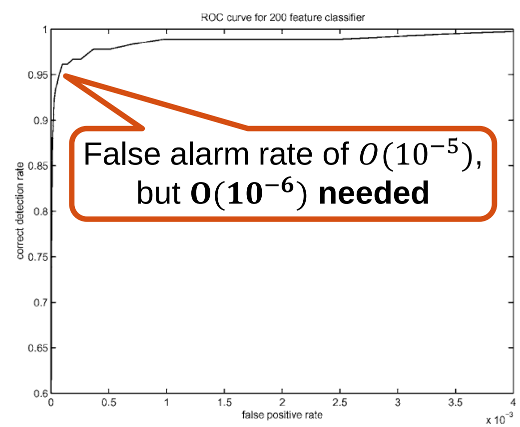
➔ **Continue** using **AdaBoost**

Training the boosting classifier
Incorporating feature selection

Training set contains face and non-face examples
- **5000 faces** (frontal, many variations among illumination/pose, rescaled to 24×24)
- **300 million non-faces** (extracted from $9'500$ non-face images)
- Faces are normalized (scale, translation)
- Initially, all have equal weights

For **each round** of boosting:
- Evaluate **each** rectangle **filter** on **each example**, select best threshold
- **Select best filter/threshold** combination
- Reweight examples

➔ Computational complexity: $O(rounds \times examples \times features)$



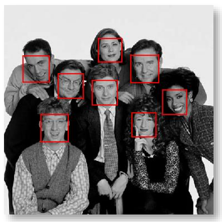
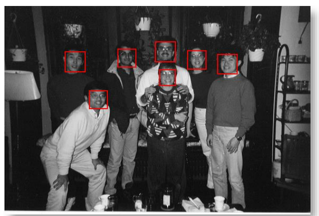
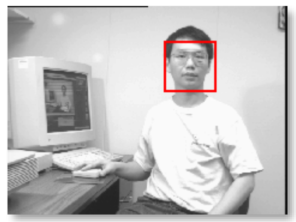
Result
- A 200-feature classifier can yield **95% detection rate** and a false positive rate of 1 in 14084

➔ **Not** yet **good** enough for practice!

Final result of Viola-Jones face detection

After some more engineering…

- Attentional **cascade** for improved false positive rate (→ see appendix)
- **Variance normalization** of pixel intensities to cope with different lighting
- Merging multiple detections
- Multi-scale detection by **scaling** the **detector** (factor of 1.25 yields good resolution)

Lasting effect

- Got **applied to more** visual detection problems
  ➔ facial feature localization, profile faces, male/female image classification, audio fingerprinting, …
- **Solved** the problem of face detection **in real time** (e.g. for digicams)
  ➔ available in OpenCV (http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html)
- One of the first mind-blowing computer vision applications before deep learning trend

Where’s the intelligence?
Man vs. machine

- Boosting emerged as an answer for theoretical problems in computational learning theory
  ➔ solves the „function learning“ problem quite well!

- Trees still need hand-crafted feature engineering
  ➔ solved by deep learning models (on different kinds of data)

- Building strong agents by combining the **„wisdom of a crowd“** of just barely useful agents is again a powerful principle from real life, thus taken over to AI
  ➔ We see a pattern here: Single **good (& simple) ideas** are taken over as singletons, but are **yet disconnected**

Review

- Ensembles **combine** many **weak learners** (same or different $\mathcal{H}$) **to** a **strong** classifier by **weighted majority voting**

- **Bagging** uses **bootstrap resampling** to construct diversity (∼complementarity)
  - **Random Forest**® uses bagging for row subsampling as well as columns (feature) subsampling → **very good out-of-the-box** model, nearly **parameter-free**
- **AdaBoost** subsequently adds new models that **focus on the harder** (previously misclassified) examples via **reweighting**
  - A seminal example is the **Viola-Jones** face detector application using Boosting for **feature selection as well as** the foundation for **classification**
- AdaBoost can be **generalize**d to **gradient boosting** (a form of gradient descent)
  - Intuition (in a regression setting) comes by deriving that the residuals (errors) of the last round are equivalent to the negative gradient of the squared error loss function
  - **XGBoost** is a **computationally** highly **optimized** & scalable implementation

- [Opinion] Ensembles make decision trees fashionable again in my eyes (i.e., not just usable on BI-like problems)

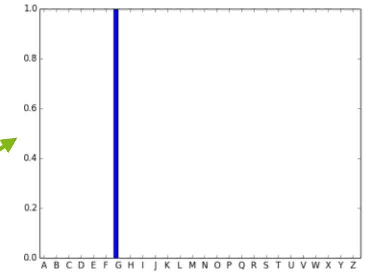
APPENDIX

Gradient boosting extension to (multiclass) classification

Model

- Each class c has its own model $F_c(x)$ (binary classification tree, emitting $0/1$)
- Use outputs to compute class probabilities: $P_c(x) = \frac{e^{F_c(x)}}{\sum_i e^{F_i(x)}}$ (softmax)

  ➔ Final classification = class with highest probability

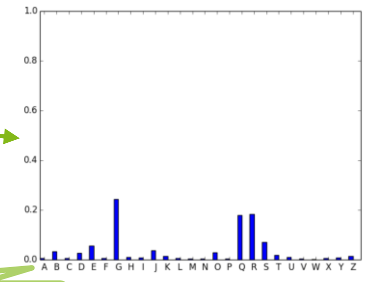


Loss function per data point

- Turn the label y_i into a (true) probability distribution $Y_c(x_i)$
- Calculate predicted probability distribution $P_c(x_i)$

  → Based on current models $F_c(x_i)$
- Calculate difference between true and predicted probability distribution

  → Use e.g. KL-divergence as loss



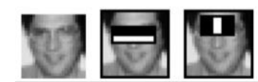
Example: Letter (A-Z) classification

Overall objective

- Do gradient descent to make true and predicted distribution as close as possible $\forall x_i$
- We achieve this goal by adjusting our models F_c

Removing false alarms while retaining high detection rate

Attentional Cascade



- **Start** with a **simple** classifier (2 features)
  ➔ **Rejecting many** of the **negative** sub-windows **while detecting** almost **all** positive sub-windows
- **Positive** response from the first classifier **triggers** the evaluation the **next** classifier, etc.
  ➔ Subsequent classifiers get more complex, hence longer runtime but lower false alarm rate
- A negative outcome at any point leads to the immediate rejection of the sub-window
- Training:
  - Keep **adding features** to **current stage until** its target rates (**TP, FP**) have been **met**
  - If overall **FP** is **not low** enough, then **add** another **stage**
  - Use false positives from current stage as the negative training examples for the next stage

Detection rate (TP) vs. false alarm rate (FP) for chained classifiers

- Found by multiplying the respective rates of the individual stages
  ➔ **TP of $\mathbf{0.9}$ and FP of $\mathbf{\sim 10^{-6}}$** can be **achieved with**
  a **$\mathbf{10}$-stage** cascade: each stage having
  - TP of 0.99 ($0.99^{10} \approx 0.9$)
  - FP of ~ 0.3 ($0.3^{10} \approx 6 \times 10^{-6}$)