AdaBoost, Gradient boosting, Viola-Jones Face detector

Main idea of boosting

„Can a set of **weak learners** create a single **strong learner**?”

- asked Kearns and Valiant (1988, 1989)

Main idea of boosting

„Can a set of **weak learners** create a single **strong learner**?”

- asked Kearns and Valiant (1988, 1989)

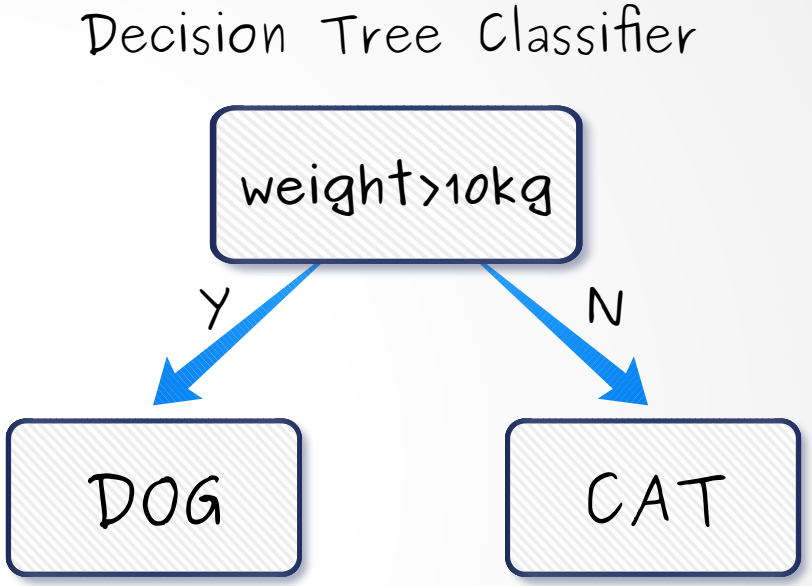
„Yes.”

- said Robert Schapire
in „The Strength of Weak Learnability” (1990)

Note:
Robert Schapire, together with Yoav Freund, invented the AdaBoost algorithm in 1996. They both received the Gödel prize in 2003 for this work.

Weak Learners

- Slightly better than random guess,
- for example shallow decision tree classifier (stump) is often used,
- but it can be any other algorithm.



But:
It works best, if the choosen algorithm has high bias and its errors are non repetetive.

Strong Learner

A classifier that is arbitrarily well-correlated with the true classification

In our case, a strong learner is built from many pieces - weak learners. Final classifier is then expressed as their linear combination.

Random Forest differences

Boosting	Bagging (e.g. Random Forest)
• We compensate **high bias** by using many weak learners, • classifiers standalone should be slightly better than random guess, • we construct them incrementaly, where each one depends on the previous one.	• We compensate **high variance** by training many classifiers and by averaging their classifications, • rather powerful classifiers - each trained on a boostrap sample, • we construct them all at once.

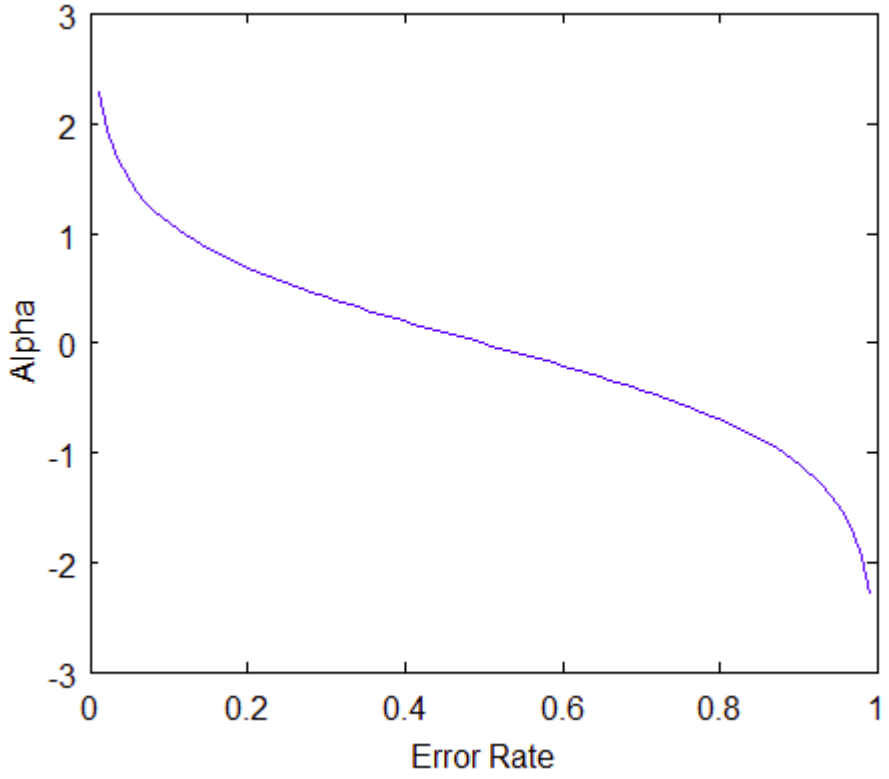
Adaptive Boosting Algorithm

Overview of algorithm:

1) Create simple classifier,
2) Compute the weight of the classifier based on its accuracy,
3) Update training examples weights *(in order to focus on harder examples)*,
4) Sample training dataset using weights as a probability distribution,
5) Go back to *1)*.

Weight of a classifier - update equation

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$$



Weight of the examples - update equation

$$D_{t+1}(i) = \frac{D_t \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Z_t - normalization term, all weights have to sum up to 1

Those weights will be a distribution from which we will choose the training examples for the next iteration classifier!

Note:

By including alpha in this term, we are also incorporating the classifier's effectiveness into consideration. If a weak classifier misclassifies an input, we don't take that as seriously as a strong classifier's mistake.

Gradient Boosting Algorithm

Overview of algorithm:

1) Create simple regressor,
2) Compute the (pseudo) residuals of it based on its predictions,
3) Replace ground truth values with these *(in order to focus on lacks of previous reg.)*,
4) Go back to *1)*.

Introduction:

problem of least squares regression - teach a model F to predict values

$$Loss = \frac{1}{n}\sum_i (F(x_i) - y_i)^2$$

In each stage of gradient boosting it may be assumed that there's an imperfect model. The algorithm improves it, by adding new estimator:

$$F_{m+1}(x) = F_m(x) + h(x)$$

To find this new estimator ***h***, we need to see that perfect ***h*** would imply:

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

$$h(x) = y - F_m(x)$$

Keep in mind, as in other boosting variants, next generations are trying to correct the errors of its predecessor.

Observation: **residulas** for a given model are in fact negative gradients of the squared error loss function:

$$-\frac{\partial}{\partial F(x)}\frac{1}{2}(y - F(x))^2 = y - F(x)$$

In general, the gradient boosting classifier is described as a weighted sum of estimators (weak learners) ***h***:

$$\hat{F}(x) = \sum_{i=1}^{M} \lambda_i h_i(x) + C$$

Algorithm:

$\{(x_i, y_i)\}_{i=1}^{n}$: training set,

$L(F(x), y)$: differentable loss function,

and let ***M*** be the number of iterations (also classifiers).

Step 1:

Initialize the model with a constant value

$$F_0(x) = \arg\min_{\lambda} \sum_{i=1}^{n} L(y_i, \lambda)$$

Step 2:

for *m=1* to *M*:

1. $r_{im} = \frac{\partial}{\partial F(x_i)} L(F(x_i), y_i)$, for $i = 1, ..., n$.

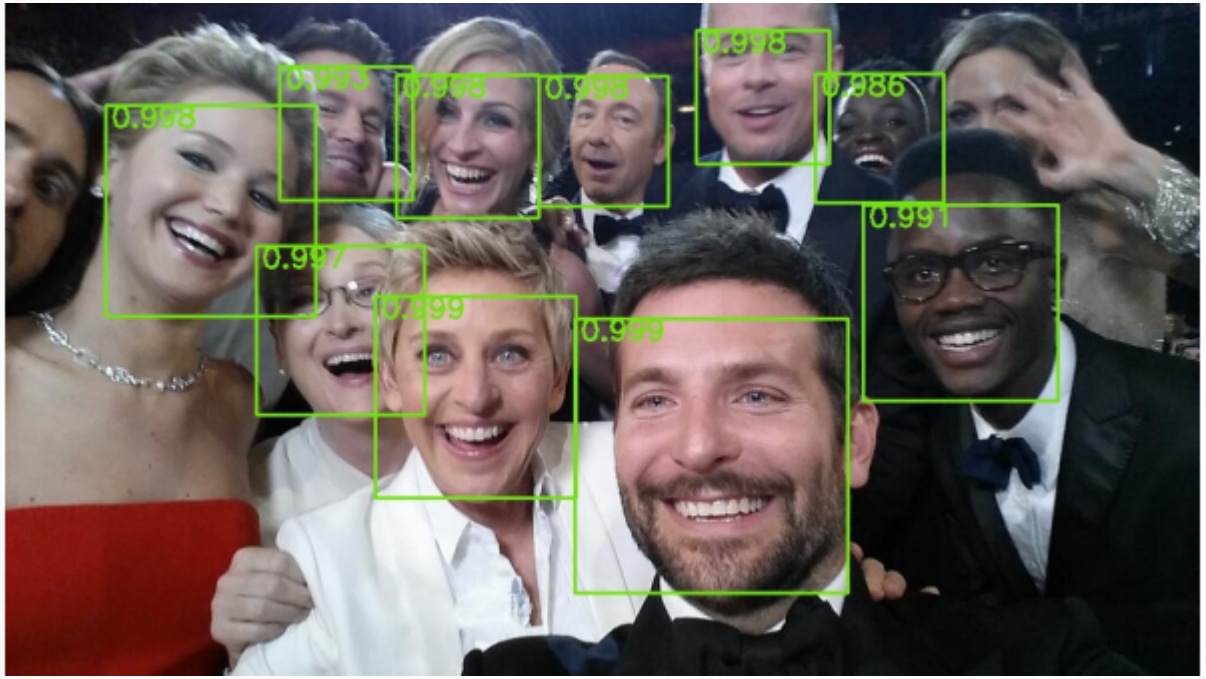
2. fit a learner $h_m(x)$ using $\{(x_i, r_{im})\}_{i=1}^{n}$ as training set

3. $\lambda_m = \arg\min_{\lambda} \sum_{i=1}^{m} L(F_{m-1}(x_i) + \lambda h_m(x_i), y_i)$

4. $F_m(x) = F_{m-1}(x) + \lambda_m h_m(x)$

Viola–Jones face detection

The first object detection framework to provide competitive object detection rates **in real-time** proposed in **2001** by Paul Viola and Michael Jones.



Viola–Jones face detection

Characteristics:

- robust - very high detection rate (true-positive rate) & very low false-positive rate always,
- real time - for practical applications at least 2 frames per second must be processed,
- face detection only (not recognition) - the goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

Viola–Jones face detection

Characteristics:

- robust - very high detection rate (true-positive rate) & very low false-positive rate always,
- real time - for practical applications at least 2 frames per second must be processed,
- face detection only (not recognition) - the goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

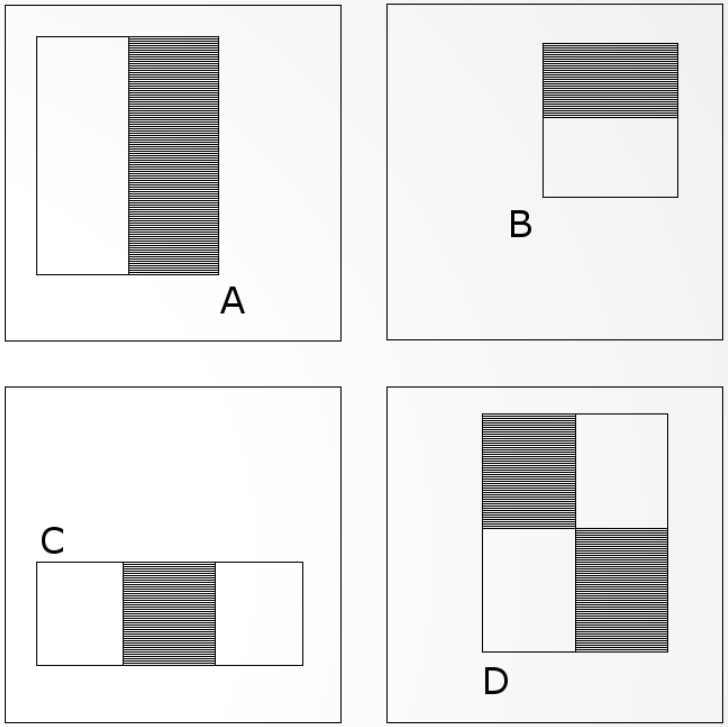
Overview of algorithm:

1) Haar Feature Selection
2) Creating an Integral Image
3) AdaBoost Training
4) Cascading Classifiers

Haar Feature selection

We use very simple, rectangular features. The value of any given feature is the sum of the pixels within clear rectangles subtracted from the sum of the pixels within shaded rectangles.

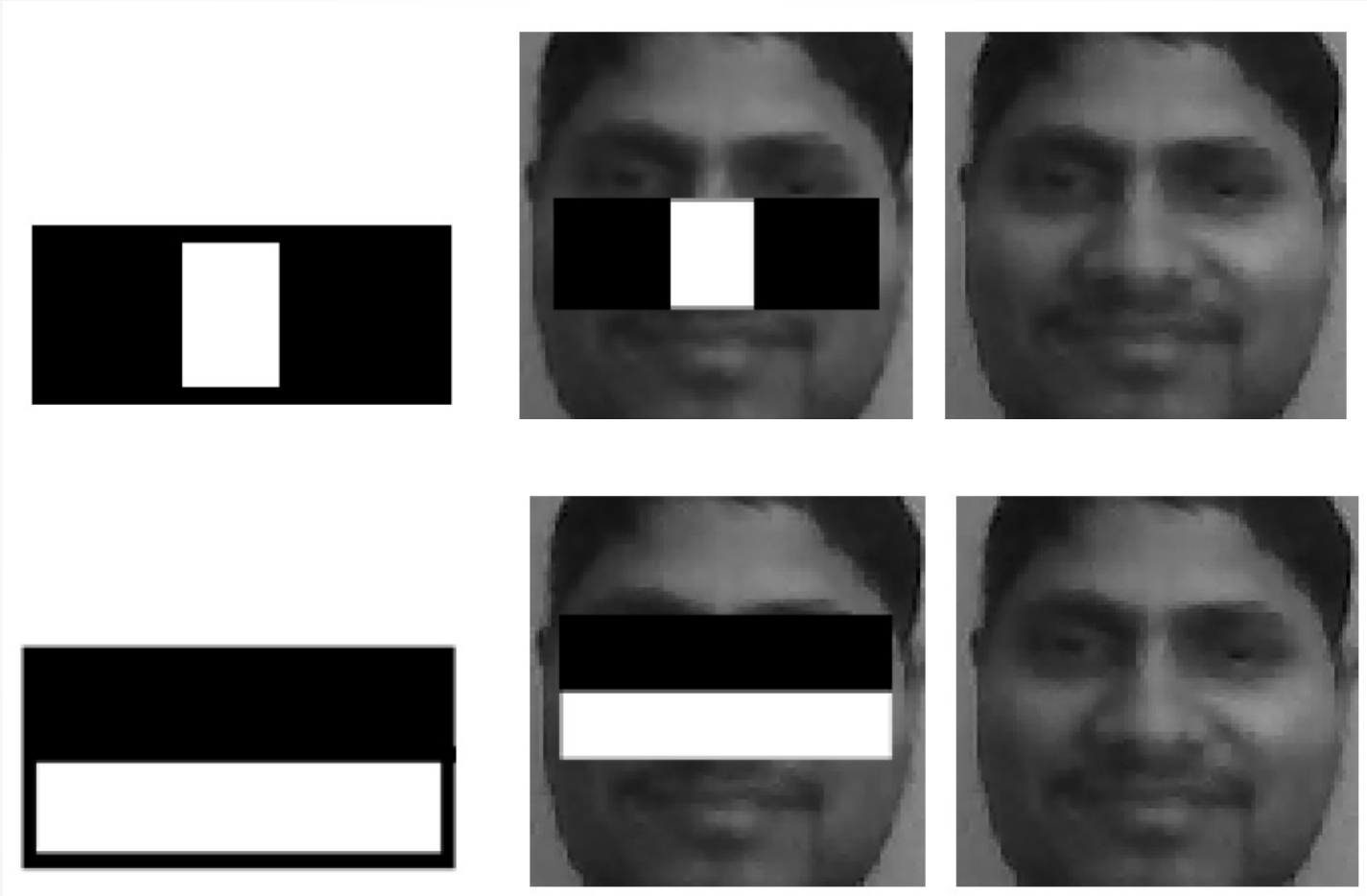
$$\sum \text{pixels in white area} - \sum \text{pixels in dark area}$$



Example rectangle features shown relative to the enclosing detection window

Haar Feature selection

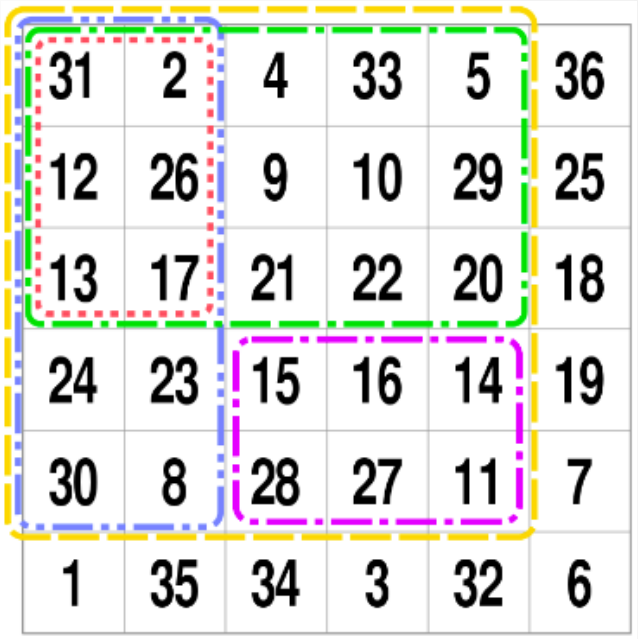
It works becasue
all human faces share some similar properties



Integral Image

We want to compute sum of pixels in down-right square for many different masks. Given we have many different masks, it can be too expensive.

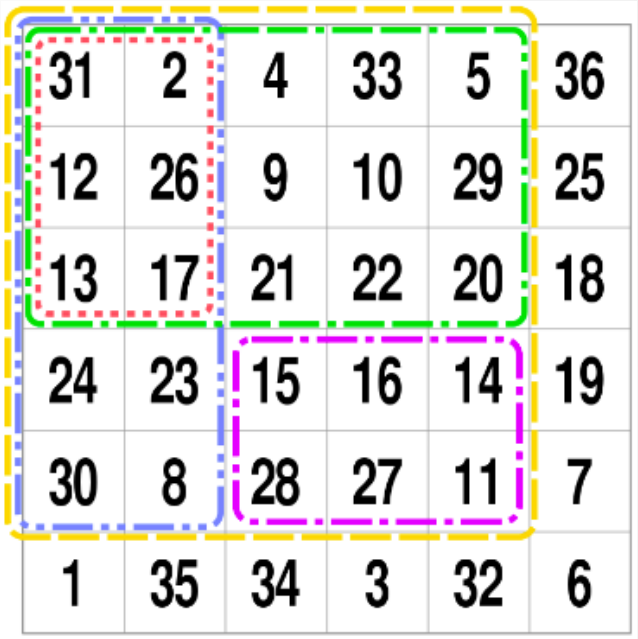
To speed things up, we introduce the **intergal image**, aka. **summed-area table**.



Integral Image

We want to compute sum of pixels in down-right square for many different masks. Given we have many different masks, it can be too expensive.

To speed things up, we introduce the **intergal image**, aka. **summed-area table**.



For any pixel value in image, the corresponding value in integral image is:

$$I(x,y) = \sum_{x' \leq x} \sum_{y' \leq y} i(x', y')$$

AdaBoost

We will combine many of Haar (-like) features into the final classifier.

For now let's observe that these features are family of highly biased, weak learners! We can create many features in different locations and shapes, and ask AdaBoost to make a strong learner from them.

Let's now remind the algorithm...

AdaBoost

AdaBoost for Viola Jones Detector:

1) Initialize uniform distribution on training images,

Repeat for each feature:

2) Find optimal threshold for feature that minimizes weighted classification error,
3) Assign a weight to the feature based on its accuracy,
4) Assign proper weights to training images *(in order to focus more on harder examples),*

5) Set the final classifier to the linear combination of the features.

Cascade architecture

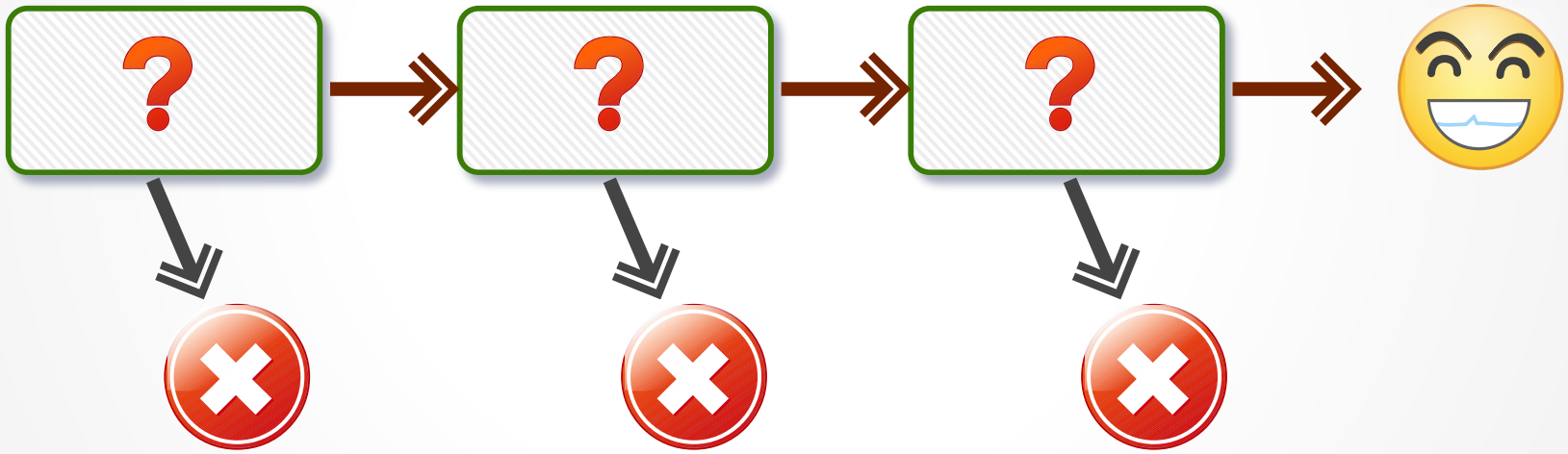
Facts:

- On average only 0.01% of all sub-windows are positive (faces)
- We’d like to spend time only on potentially positive sub-windows

Cascade architecture

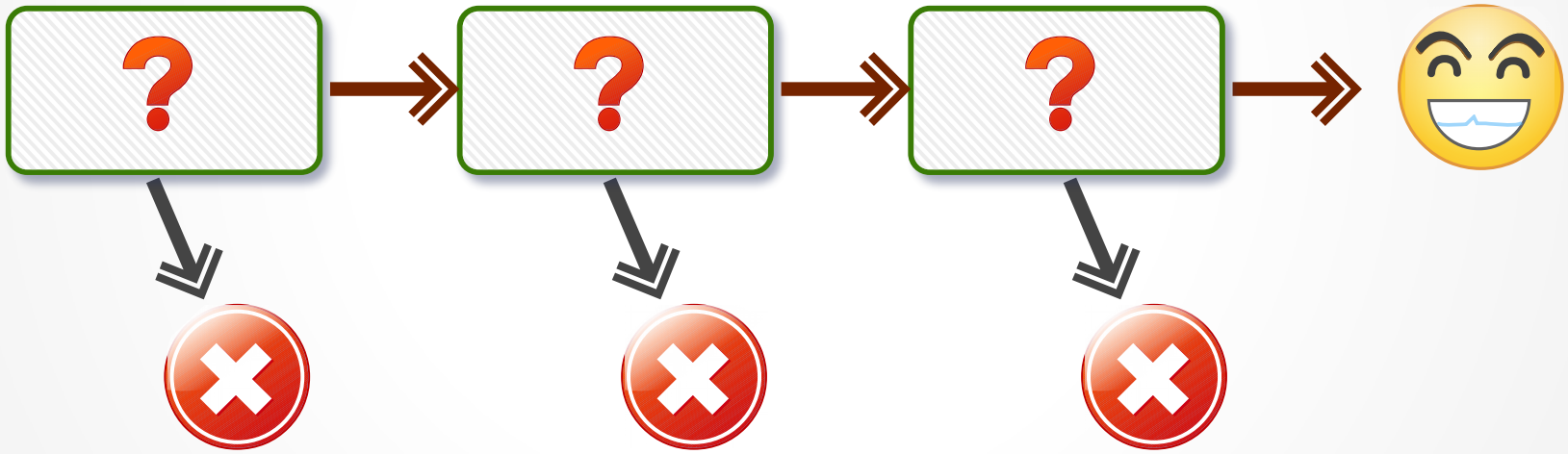
Facts:

- On average only 0.01% of all sub-windows are positive (faces)
- We'd like to spend time only on potentially positive sub-windows



Cascade architecture

The classifiers are arranged in a cascade in **order of complexity.** The first classifiers should be quicker, since it will be used every time. With the cascade, the final classifier has **extremely small false positive rate**.



XGBoost library

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.

Except for optimization, there were many tricks used:

- regularization,
- early stopping,
- column and row subsampling,
- parameters tuning,
- and more!

It is also easy to download and use in Python.

But consider alternatives - e.g. LightGBM.

AdaBoost, Gradient boosting, Viola-Jones Face detector

Thank you!