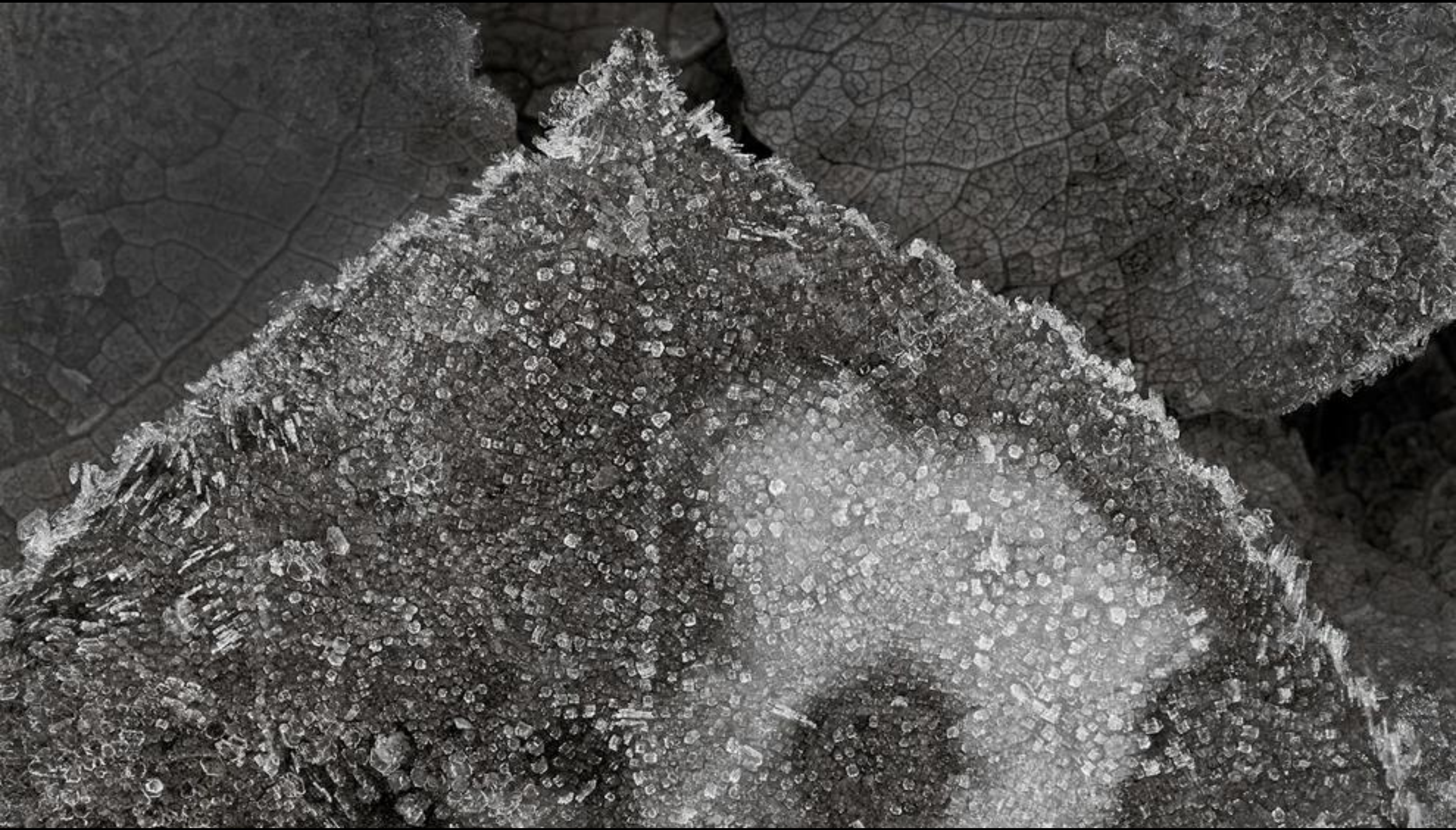
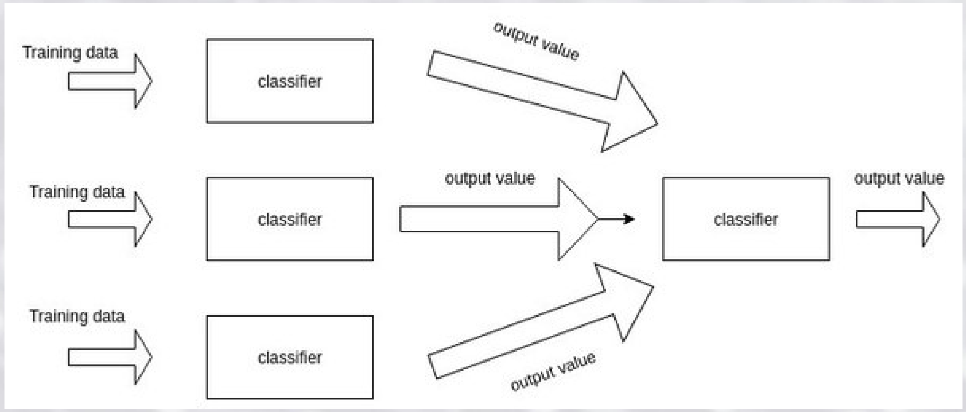


Ensemble Learning
CS 446/546

Outline



- Overview of Ensemble Learning
- Generating Diverse Learners
- Model Combination Schemes:

  Voting, Bagging, Boosting, Mixture of Experts, Cascading
- *AdaBoost* Algorithm
- Viola-Jones Face Detection Algorithm

Overview

- The **"no free lunch" theorem** states that there is no single learning algorithm that in any domain always induces the most accurate learner. The usual approach in ML is to try many different models and to choose the one that performs best on a separate *validation set*.

- Each learning algorithm dictates a specific model (e.g. SVMs, NNs, probabilistic models) that comes with a set of inherent assumptions.

- This **inductive bias** renders an error if the assumption do not hold for the data.

- Oftentimes we attempt to mitigate the effect of these assumptions by adopting a methodology that makes as few *a priori* assumptions as possible (see: *the principle of insufficient reason*); using **multiple learners**, however, presents an alternative solution.

Overview

- The performance of a learned model may be fine-tuned to get the highest possible accuracy on a validation set, but this fine-tuning is a complex task and still there are instances on which even the best learner is inaccurate.

- By suitably combining multiple learned models as an *ensemble*, accuracy can be improved.

This leads to (2) fundamental questions for ensemble learning:

(1) How do we generate *base-learners* (i.e. learned models using the same data/learning algorithm type) that complement each other?

(2) How do we <u>combine the outputs of the base-learners for maximum accuracy</u>?

(*) Note that <u>not all model combinations will necessarily increase test accuracy</u>, but learning extra models will always increase run-time and space complexity.

Generating Diverse Learners

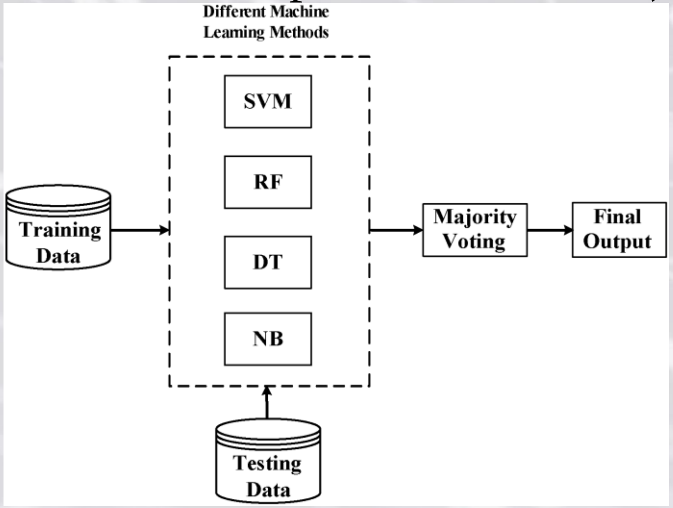
• There are <u>two basic tensions</u> when it comes to generating diverse learning models.

(*) On the one hand, we want *dissimilar models* so that they differ (in a complementary way) in their decisions; at the same time, we want a gain in overall accuracy, so the different learners need to be *accurate* (at least in their domain of expertise).

Here are some of the different ways to achieve the generation of diverse learners:

(i) **Use different algorithms**: We can use different learning algorithms to train different base-learners. Different algorithms make different assumptions about the data and yield different classifiers (e.g. parametric model + non-parametric model).

Combining multiple learners in this fashion frees us (potentially) over committing to one particular learning model (i.e. a stricter inductive bias).



Generating Diverse Learners

(ii) Use different hyperparameters: We can use the same learning algorithm but with different hyperparameters (e.g. different size NNs, different momentum/decay parameters, different regularization parameters, different initial weights, etc.).

• When we train multiple base-learners with different hyperparameter values, we average over this factor and thus reduce variance (and predictive error).

Generating Diverse Learners

(ii) Use different hyperparameters: We can use the same learning algorithm but with different hyperparameters (e.g. different size NNs, different momentum/decay parameters, different regularization parameters, different initial weights, etc.).

When we train multiple base-learners with different hyperparameter values, we average over this factor and thus reduce variance (and predictive error).

(iii) **Use different input representations**: Separate base-learners may use different representations of the same input object, making it possible to integrate different types of data modalities.

• Different representations render different characteristics of the data explicit (i.e. some embedded patterns become more prominent than others). Choosing different data representations could involve using disparate *feature selection* or *feature engineering* processes.

• One simple approach in this vein is to **choose random subsets of the data** (*random subspace method*). This has the effect that different learners will look at the same problem from different perspectives, generating a robust method that is potentially less susceptible to the curse of dimensionality.

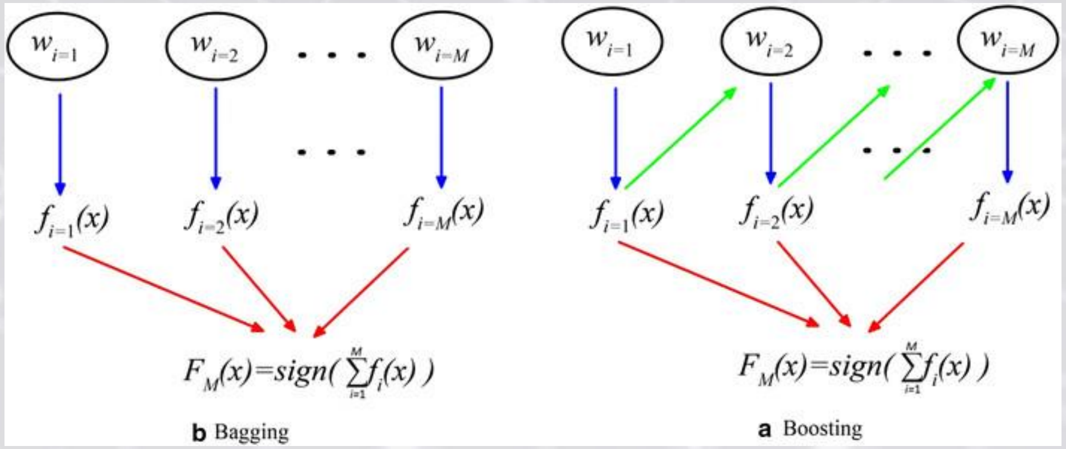
Generating Diverse Learners

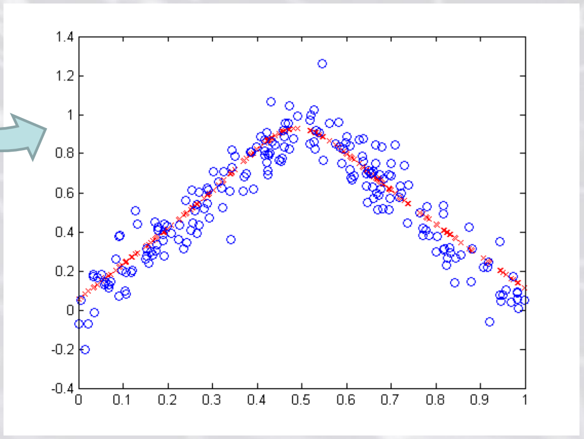
(iv) Use different training sets: Another option is to train different base-learners using different subsets of the training set.

This can be done by drawing random training sets from a given sample – this is known as **bagging**.

Or, alternatively, learners can be trained serially so that instances on which the preceding base-learners are not accurate are given more emphasis in training later base-learners – this is generally known as **boosting** (additional related method: **cascading**). The explicit advantage of this approach is that we <u>actively try to generate complementary learners</u>, instead of leaving this to chance.

Lastly, if we partition the training data based on locality in the input space and train a base-learner on instances in a certain local part of the input space, this produces a **mixture of experts**.





Generating Diverse Learners

(*) Diversity vs. Accuracy: It is important to note that when we generate multiple base-learners, we want them to be reasonably accurate but we don't require them to be very accurate individually.

• Consequently, our base-learners are generally not optimized separately for best accuracy. The base-learners are instead chosen for their *simplicity* (remember that we intend to train many of them), and not their overall accuracy.

• We do nonetheless <u>require the base-learners to be diverse</u>, that is to say: accurate on different instances, thereby specializing in subdomains of the problem.

Bottom line: the <u>final accuracy when the base-learners are combined is paramount</u>.

(*) Note that this implies that the required accuracy and diversity of the learners also depend on **how their decisions are combined**, which we discuss next.

Model Combination Schemes

Next we consider (5) of the most common model combination schemes used in ensemble learning for ML:

(1) Voting
(2) Bagging
(3) Boosting
(4) Mixture of Experts
(5) Cascading

First, let's begin with a simple and intuitive example of ensemble learning.

Ensemble Learning with Voting: Example

Given three hypotheses, h_1, h_2, $h_{3,}$ with $h_i(\mathbf{x}) \in \{-1,1\}$

Suppose each h_{i} has 60% generalization accuracy, and assume errors are independent.

Now suppose $H(\mathbf{x})$ is the majority vote of h_1, h_2, and h_3. What is probability that H is correct?

h_1	h_2	h_3	***H***	*probability*
C	C	C	**C**	
C	C	I	**C**	
C	I	I	I	
C	I	C	**C**	
I	C	C	**C**	
I	I	C	I	
I	C	I	I	
I	I	I	I	
				Total probability correct:

h_1	h_2	h_3	***H***	*probability*
C	C	C	**C**	.216
C	C	I	**C**	.144
C	I	I	I	.096
C	I	C	**C**	.144
I	C	C	**C**	.144
I	I	C	I	.096
I	C	I	I	.096
I	I	I	I	.064
				Total probability correct: .648

Ensemble Learning with Voting: Example

Again, given three hypotheses, h_1, h_2, $h_{3.}$

Suppose each h_i has 40% generalization accuracy, and assume errors are independent.

Now suppose we classify **x** as the majority vote of h_1, h_2, and h_3 . What is probability that the classification is correct?

h_1	h_2	h_3	***H***	*probability*
C	C	C	**C**	
C	C	I	**C**	
C	I	I	I	
C	I	C	**C**	
I	C	C	**C**	
I	I	C	I	
I	C	I	I	
I	I	I	I	

h_1	h_2	h_3	***H***	*probability*
C	C	C	**C**	.064
C	C	I	**C**	.096
C	I	I	I	.144
C	I	C	**C**	.096
I	C	C	**C**	.096
I	I	C	I	.144
I	C	I	I	.144
I	I	I	I	.261
				Total probability correct: .352

Q: In general, if hypotheses $h_1, ..., h_M$ all have generalization accuracy **A**, what is probability that a majority vote will be correct? (you should work this out)

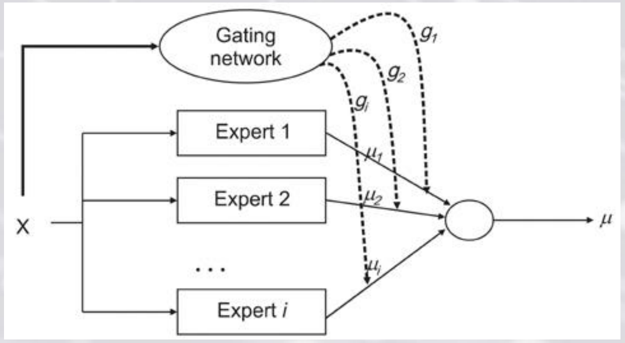
Model Combination Schemes

"Big picture" considerations for model combination schemes:

(*) *Multi-expert combination* methods have base-learners that work in parallel:

(i) In the **global approach** to multi-expert combination methods, given an input, <u>all</u> base-learners generate an output and all these outputs are used for prediction (e.g., voting).

(ii) In the **local approach**, there is a *gating model*, which looks at the input and chooses one (or very few) of the learners as responsible for generating the output (e.g. mixture of experts).



Model Combination Schemes

“Big picture” considerations for model combination schemes:

(*) *Multi-expert combination* methods have base-learners that work in parallel (e.g. global approach and local approach).

(*) *Multistage combination*: apply a serial approach where the next base-learner is trained with or tested on only the instances where the previous base-learners are not sufficiently accurate (e.g., boosting).

• The idea is that the base-learners are sorted according to increasing complexity so that a complex base-learner is not used unless the preceding, simpler base-learners are not confident.

Model Combination Schemes

"Big picture" considerations for model combination schemes:

(*) *Multi-expert combination* methods have base-learners that work in *parallel.*

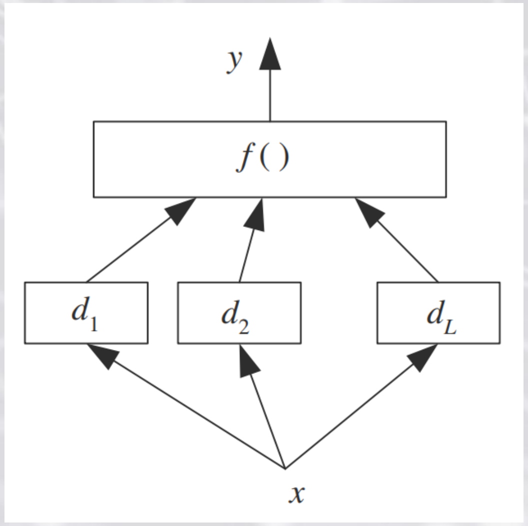
(*) *Multistage combination*: apply a *serial approach.*

Suppose that we have L base-learners. Denote the prediction of *jth* base-learner for input x as $d_j(x)$.

The **final prediction** is calculated from the predictions of the base learners:

$$y = f\left(d_1, d_2, \ldots, d_L \mid \theta\right)$$

Where f(·) is the *combining function* (also called the *fusion function*) with parameters θ.



Model Combination Schemes: Voting

- The simplest way to combine multiple classifiers is by **voting** (see previous example), which corresponds to taking a linear combination of the learners:

$$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$$

Where w_j is the combination weight and d_{ji} denotes the *jth* (of say L total base-learners) using the *ith* input representation of the data.

Model Combination Schemes: Voting

$$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$$

• In the case where all learners are given equal weight this yields *simple voting* (i.e. the answer is the average of all the base learners).

In fact there are many plausible different *ensemble rules* one can use:

Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = median_j\, d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

(*) Sum rule is intuitive and commonly used in practice; median is more robust to outliers; min and max are pessimistic/optimistic, respectively.

Model Combination Schemes: Voting

$$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$$

• For classification tasks, one ordinarily uses a *majority vote criterion* (i.e. winning class gets the most votes).

• Another possibility is to learn the weights in the fusion function themselves (see: *stacked generalization*).

• Still an additional alternative is to **be Bayesian**.

Model Combination Schemes: Voting

$$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$$

- Still an additional alternative is to **be Bayesian.**

For example, in classification if we have: $w_j = P(M_j)$, $d_{ji} = P(C_i | x, M_j)$, where M_j is the *jth* base-learner this yields the following posterior calculation:

$$P(C_i | x) = \sum_{\text{all models } M_j} P(C_i | x, M_j) P(M_j)$$

(*) Note that in this scheme the weight correlates with the prior, so that a cogent model has a higher weight, and the prediction for each base-learner is the class posterior for the given base-learner.

Model Combination Schemes: Voting

• The fundamental idea with voting is that it has the <u>effect of smoothing the functional space</u> and can be accordingly thought of as a **regularizer** with a **smootheness assumption** on the true function.

(*) We vote over models with high variance and low bias so that <u>after combining base-learners, the bias remains small and we reduce the variance by averaging</u>. Even if the individual models are biased, the decrease in variance may offset this bias and still decrease error.

Model Combination Schemes: Voting

(*) We vote over models with high variance and low bias so that after combining base-learners, the bias remains small and we reduce the variance by averaging. Even if the individual models are biased, the decrease in variance may offset this bias and still decrease error.

More formally: consider the expected value and variance of the output prediction $y_i = \sum_j w_j d_{ji}$, rendered by a simple average, so w_j=1/l (we also assume the d_j are IID for simplicity).

$$E[y] = E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L E[d_j] = E[d_j]$$

Why?

(*) This formula shows that the expected value doesn't change (when we compare the ensemble combination to a single base-learner); thus **the bias for the ensemble model doesn't change**.

Model Combination Schemes: Voting

$$E[y] = E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L E\left[d_j\right] = E\left[d_j\right]$$

(*) This formula shows that the expected value doesn't change (when we compare the ensemble combination to a single base-learner); thus **the bias for the ensemble model doesn't change**.

$$Var[y] = Var\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L^2} Var\left[\sum_j d_j\right] = \frac{1}{L^2} L Var\left[d_j\right] = \frac{1}{L} Var\left[d_j\right]$$

Why? Why?

(*) Crucially, this formula demonstrates that <u>variance decreases as the number of voters L increases</u>.

Again, illustrating the **key point**: We vote over models with high variance and low bias so that <u>after combining base-learners, the bias remains small and we reduce the variance by averaging</u>.

Model Combination Schemes: Bagging

• **Bagging** (short for **bootstrap aggregating**) is a voting method whereby base-learners are made different by training them over slightly different training sets -- among other applications, it is frequently used with decision trees.

• Generating *L* slightly different samples from a given sample is done by **bootstrap** (i.e. *sampling with replacement*): Given a training set *X* of size *N* we draw *N* instances randomly from *X* with replacement.

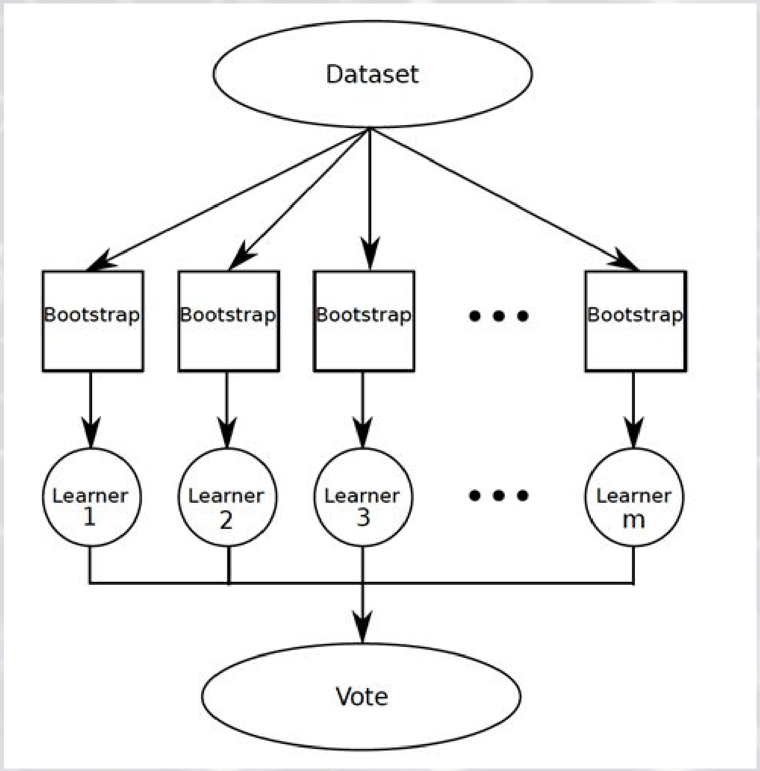
• The effect of bootstrapping is to generate L samples (X_j, j=1, …, L) that are similar because they are all drawn from the same original sample, but they are nevertheless slightly different due to chance.

Each base learner d_j is thus trained with these L samples X_j.

Model Combination Schemes: Bagging

• In summary: Bagging generates L training sets using bootstrapping, trains L base-learners, and then applies a voting procedure to the L base-learners (in order to reduce variance of the overall model).

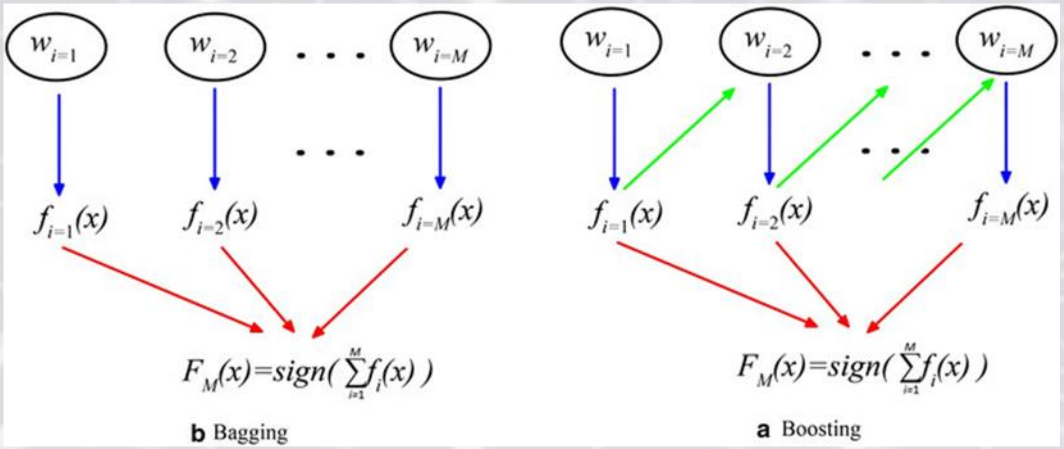
• Bagging can be used for both regression and classification tasks; in the case of regression it is more common to use the median (as opposed to the mean) for improved robustness.



Model Combination Schemes: Boosting

• With bagging, generating complementary base-learners is left to chance through boostrapping; in **boosting**, we *actively* try to generate complementary base-learners by training the learners *sequentially*, so that the next learner trains on the mistakes of the previous learners.

• Boosting combines complementary *weak learners* (meaning their accuracy is above chance, but they are nonetheless relatively inexpensive to train).



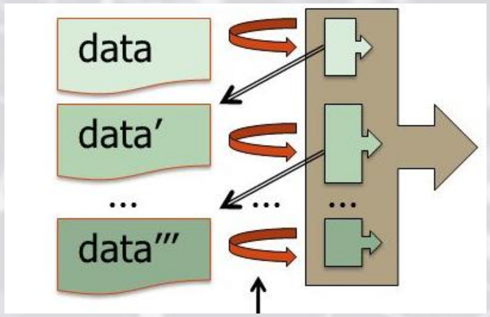
"Intro to Boosting": https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

Model Combination Schemes: Boosting

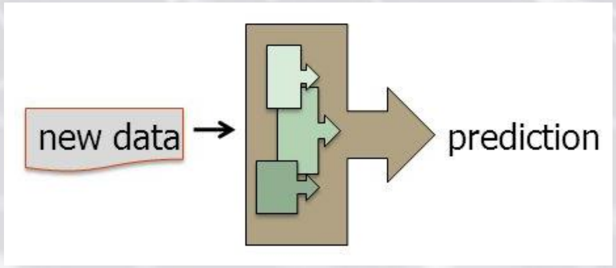
• As a basic schematic for boosting, consider a boosting algorithm (this is how the original 1990 Schapire paper worked) that combines three weak learners to generate a strong learner.

• Given a training set, we randomly partition it into three subsets: X_1, X_2 and X_3; use X_1 to train d_1. Then take X_2 and feed it to d_1. Next we u se every instance misclassified by d_1 in combination with many instances on which d_1 is correct from X_2, and together form the training set for d_2.

• Lastly, we take X_3 and feed it to d_1 and d_2; the instances on which d_1 and d_2 disagree form the training set of d_3.



(*) During testing, we take a datum and give it to d_1 and d_2; if they agree this is the prediction; otherwise, the response of d_3 is taken as the output.



Model Combination Schemes: AdaBoost

• A very popular boosting method known as **AdaBoost** (short for *adaptive boosting*) was developed by Freund and Schapire in 1996 (later won the *Gödel prize*).

(*) Adaboost uses the same training set over and over and thus the data set need not be large, but the classifiers should be simple so that they do no overfit. AdaBoost can also combine an arbitrary number of base learners – not just three.

(*) Adaboost combines different weak learners (i.e. hypotheses), where the training error is close but less than 50%, to produce a strong learner (i.e. with training error close to zero).

AdaBoost: Algorithm Sketch

Given examples S and learning algorithm L, with $| S | = N$

- Initialize probability distribution over examples $\mathbf{w}_1(i) = 1/N$.
- Repeatedly run L on training sets $S_t \subset S$ to produce $h_1, h_2, \ldots , h_K$.
  - At each step, derive S_t from S by choosing examples probabilistically according to probability distribution $\mathbf{w}_t$. Use S_t to learn h_t.
- At each step, derive $\mathbf{w}_{t+1}$ by giving more probability to examples that were misclassified at step t.
- The final ensemble classifier H is a weighted sum of the h_t's, with each weight being a function of the corresponding h_t's error on its training set.

AdaBoost: Algorithm

- Given $S = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ where $\mathbf{x} \in \boldsymbol{X}, y_i \in \{+1, -1\}$

- Initialize $\mathbf{w}_1(i) = 1/N$. (Uniform distribution over data)

AdaBoost: Algorithm

- For $t = 1, \ldots, K$:
  - Select new training set S_t from S with replacement, according to $\mathbf{w}_t$
  - Train L on S_t to obtain hypothesis h_t
  - Compute the training error ε_t of h_t on $\boldsymbol{S}$:

$$\varepsilon_t = \sum_{j=1}^{N} \mathbf{w}_t(j)\, \delta(y_j \neq h_t(\mathbf{x}_j)), \text{ where}$$

$$\delta(y_j \neq h_t(\mathbf{x}_j)) = \begin{cases} 1 \text{ if } y_j \neq h_t(\mathbf{x}_j) \\ 0 \text{ otherwise} \end{cases}$$

  - Compute coefficient

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

AdaBoost: Algorithm

– Compute new weights on data:

For $i = 1$ to N

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that $\mathbf{w}_{t+1}$ will be a probability distribution:

$$Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

AdaBoost: Algorithm

- At the end of K iterations of this algorithm, we have

  $h_1, h_2, \ldots, h_K$

  We also have

  $\alpha_1, \alpha_2, \ldots, \alpha_K$, where

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

- Ensemble classifier:

$$H(\mathbf{x}) = \operatorname{sgn} \sum_{t=1}^{K} \alpha_t h_t(\mathbf{x})$$

- Note that hypotheses with higher accuracy on their training sets are weighted more strongly.

AdaBoost: Data Example

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ are class +1

$\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ are class −1

$t = 1$:

$\mathbf{w}_1 = \{1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8\}$

$S_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ (notice some repeats)

Train classifier on S_1 to get h_1

Run h_1 on **S**. Suppose classifications are: {**1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**}

- Calculate error: $\epsilon_1 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j)) = \frac{1}{8}(3) = .375$

AdaBoost: Data Example

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ are class +1

$\{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ are class −1

$t = 1$:

$\mathbf{w}_1 = \{1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8, 1/8\}$

$S_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$ (notice some repeats)

Train classifier on S_1 to get h_1

Run h_1 on **S**. Suppose classifications are: {**1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**, **−1**}

- Calculate error: $\varepsilon_1 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j)) = ?$

Calculate *α's*:

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) =$$

Calculate new **w***'s*:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$$\hat{\mathbf{w}}_2(1) =$$

$$\hat{\mathbf{w}}_2(2) =$$

$$\hat{\mathbf{w}}_2(3) =$$

$$\hat{\mathbf{w}}_2(4) =$$

$$\hat{\mathbf{w}}_2(5) =$$

$$\hat{\mathbf{w}}_2(6) =$$

$$\hat{\mathbf{w}}_2(7) =$$

$$\hat{\mathbf{w}}_2(8) =$$

$$\mathbf{w}_2(1) =$$

$$\mathbf{w}_2(2) =$$

$$\mathbf{w}_2(3) =$$

$$\mathbf{w}_2(4) =$$

$$\mathbf{w}_2(5) =$$

$$\mathbf{w}_2(6) =$$

$$\mathbf{w}_2(7) =$$

$$\mathbf{w}_2(8) =$$

$$Z_1 = \sum_i \hat{\mathbf{w}}_2(i) =$$

Calculate α's:

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = .255$$

Calculate new **w**'s:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$$\hat{\mathbf{w}}_2(1) = (.125)\exp(-.255(1)(1)) = 0.1$$
$$\hat{\mathbf{w}}_2(2) = (.125)\exp(-.255(1)(-1)) = 0.16$$
$$\hat{\mathbf{w}}_2(3) = (.125)\exp(-.255(1)(-1)) = 0.16$$
$$\hat{\mathbf{w}}_2(4) = (.125)\exp(-.255(1)(-1)) = 0.16$$
$$\hat{\mathbf{w}}_2(5) = (.125)\exp(-.255(-1)(-1)) = 0.1$$
$$\hat{\mathbf{w}}_2(6) = (.125)\exp(-.255(-1)(-1)) = 0.1$$
$$\hat{\mathbf{w}}_2(7) = (.125)\exp(-.255(-1)(-1)) = 0.1$$
$$\hat{\mathbf{w}}_2(8) = (.125)\exp(-.255(-1)(-1)) = 0.1$$

$$\mathbf{w}_2(1) = 0.1/.98 = 0.102$$
$$\mathbf{w}_2(2) = 0.163$$
$$\mathbf{w}_2(3) = 0.163$$
$$\mathbf{w}_2(4) = 0.163$$
$$\mathbf{w}_2(5) = 0.102$$
$$\mathbf{w}_2(6) = 0.102$$
$$\mathbf{w}_2(7) = 0.102$$
$$\mathbf{w}_2(8) = 0.102$$

$$Z_1 = \sum_i \hat{\mathbf{w}}_2(i) = .98$$

$t = 2$

$w_2 = \{0.102, 0.163, 0.163, 0.163, 0.102, 0.102, 0.102, 0.102\}$

$S_2 = \{x_1, x_2, x_2, x_3, x_4, x_4, x_7, x_8\}$

Learn classifier on S_2 to get h_2

Run h_2 on S. Suppose classifications are: {1, 1, 1, 1, 1, 1, 1, 1}

Calculate error:

$$\epsilon_2 = \sum_{j=1}^{N} \mathbf{w}_t(j)\,\delta(y_j \neq h_t(\mathbf{x}_j))$$

$$= (.102) \times 4 = 0.408$$

Calculate α'*s*:

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = .186$$

Calculate **w**'*s*:

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

$$\hat{\mathbf{w}}_3(1) = (.102)\exp(-.186(1)(1)) = 0.08$$
$$\hat{\mathbf{w}}_3(2) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(3) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(4) = (.163)\exp(-.186(1)(1)) = 0.135$$
$$\hat{\mathbf{w}}_3(5) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(6) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(7) = (.102)\exp(-.186(-1)(1)) = 0.122$$
$$\hat{\mathbf{w}}_3(8) = (.102)\exp(-.186(-1)(1)) = 0.122$$

$$\mathbf{w}_3(1) = 0.08/.973 = 0.082$$
$$\mathbf{w}_3(2) = 0.139$$
$$\mathbf{w}_3(3) = 0.139$$
$$\mathbf{w}_3(4) = 0.139$$
$$\mathbf{w}_3(5) = 0.125$$
$$\mathbf{w}_3(6) = 0.125$$
$$\mathbf{w}_3(7) = 0.125$$
$$\mathbf{w}_3(8) = 0.125$$

$$Z_2 = \sum_i \hat{\mathbf{w}}_2(i) = .973$$

t =3

w_3 = {0.082, 0.139, 0.139, 0.139, 0.125, 0.125, 0.125, 0.125}

S_3 = {x_2, x_3, x_3, x_3, x_5, x_6, x_7, x_8}

Run classifier on S_3 to get h_3

Run h_3 on S. Suppose classifications are: {1, 1, −1, 1, −1, −1, 1, −1}

Calculate error:

$$\varepsilon_3 = \sum_{j=1}^{N} \mathbf{w}_t(i)\,\delta(y_j \neq h_t(\mathbf{x}_j))$$

$$= (.139) + (.125) = 0.264$$

- Calculate *α's*:

$$\alpha_3 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = .512$$

- Ensemble classifier:

$$H(\mathbf{x}) = \operatorname{sgn}\sum_{t=1}^{K} \alpha_t h_t(\mathbf{x})$$

$$= \operatorname{sgn}\left(.255 \times h_1(\mathbf{x}) + .186 \times h_2(\mathbf{x}) + .512 \times h_3(\mathbf{x})\right)$$

Exampl e	Actual class	h_1	h_2	h_3
$\mathbf{x}_1$	1	1	1	1
$\mathbf{x}_2$	1	−1	1	1
$\mathbf{x}_3$	1	−1	1	−1
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	−1	−1	1	−1
$\mathbf{x}_6$	−1	−1	1	−1
$\mathbf{x}_7$	−1	1	1	1
$\mathbf{x}_8$	−1	−1	1	−1

Recall the training set:

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8,\}$$

where $\{ x_1, x_2, x_3, x_4 \}$ are class +1

$\{x_5, x_6, x_7, x_8 \}$ are class −1

$$H(\mathbf{x}) = \operatorname{sgn} \sum_{t=1}^{T} a_t h_t(\mathbf{x})$$

$$= \operatorname{sgn}\left(.255 \times h_1(\mathbf{x}) + .186 \times h_2(\mathbf{x}) + .512 \times h_3(\mathbf{x})\right)$$

What is the accuracy of *H* on the training data?

AdaBoost: Summary

- Given S = $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in X$, $y_i \in \{+1, -1\}$

- Initialize $w_1(i) = 1/N$. (Uniform distribution over data)

- For $t = 1, \ldots, K$:

  1. Select new training set S_t from S with replacement, according to $\mathbf{w}_t$

  1. Train L on S_t to obtain hypothesis h_t

  1. Compute the training error ε_t of h_t on $\boldsymbol{S}$:

$$\varepsilon_t = \sum_{j=1}^{N} \mathbf{w}_t(j)\, \delta(y_j \neq h_t(\mathbf{x}_j)),$$

$$\text{where } \delta(y_j \neq h_t(\mathbf{x}_j)) = \begin{cases} 1 \text{ if } y_j \neq h_t(\mathbf{x}_j) \\ 0 \text{ otherwise} \end{cases}$$

If $\varepsilon_t > 0.5$, abandon h_t and go to step 1

4. Compute coefficient:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

5. Compute new weights on data:

For $i = 1$ to N

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

where Z_t is a normalization factor chosen so that $\mathbf{w}_{t+1}$ will be a probability distribution:

$$Z_t = \sum_{i=1}^{N} \mathbf{w}_t(i)\ \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$$

- At the end of K iterations of this algorithm, we have $h_1, h_2, \ldots, h_K$, and $\alpha_1, \alpha_2, \ldots, \alpha_K$

- Ensemble classifier:

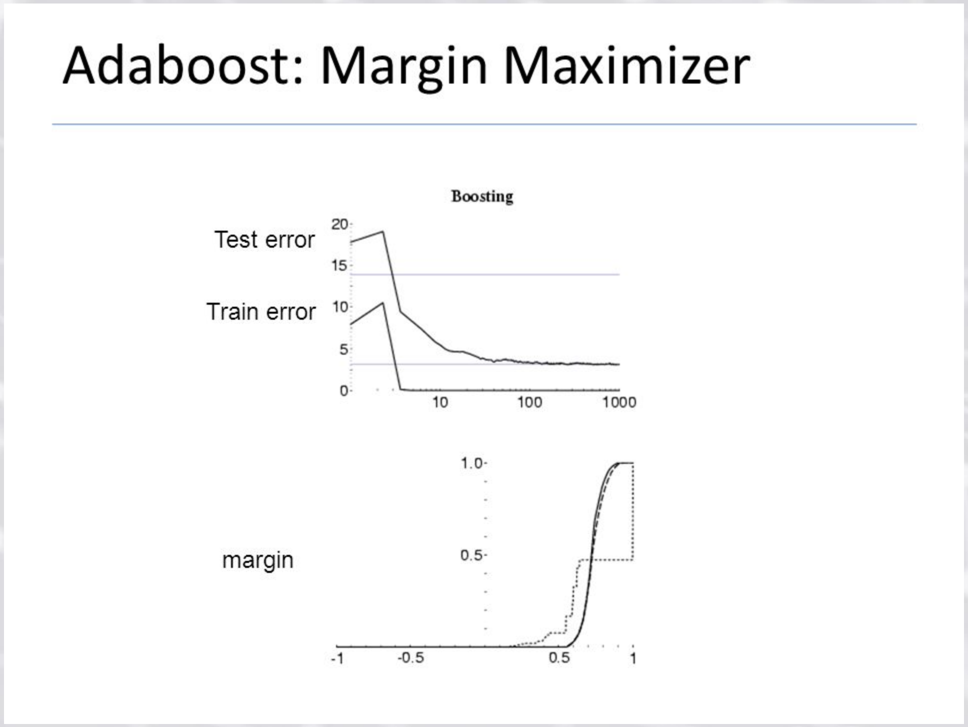
$$H(\mathbf{x}) = \operatorname{sgn}\sum_{t=1}^{K} \alpha_t h_t(\mathbf{x})$$

AdaBoost: Overview

- Adaboost seems to reduce both bias and variance and it does not seem to overfit for increasing K.

Why does it work?

Schapire et al. explain that the success of AdaBoost is *due to its property of increase the margin*. Recall from SVMs, that if the margin increases, the training instances are better separated and an error is less likely.



Adaboost: Margin Maximizer

AdaBoost: Overview

•In AdaBoost, although different base-learners have slightly different training sets, <u>this difference is not left to chance as in bagging</u>, but is a function of the error of the previous base-learner. The actual performance of boosting on a particular problem is naturally dependent on the data and base-learner.

• In order to be effective, there should be enough training data and the base-learner should be weak but not too weak, as boosting is particularly susceptible to noise and outliers (since boosting focuses on examples are hard to classify).

• For this reason boosting can be used to identify outliers and noise in a dataset.

(*) AdaBoost has also been generalized to regression.

Case Study of Adaboost: Viola-Jones Face Detection Algorithm

- P. Viola and M. J. Jones, *Robust real-time face detection*. International Journal of Computer Vision, 2004.

- First face-detection algorithm to work well in real-time (e.g., on digital cameras); it has been very influential in computer vision (16k+ citations).

Viola:
MIT/Amazon

https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

Viola-Jones: Training Data

- Positive: Faces scaled and aligned to a base resolution of 24 by 24 pixels.

- Negative: Much larger number of non-faces.

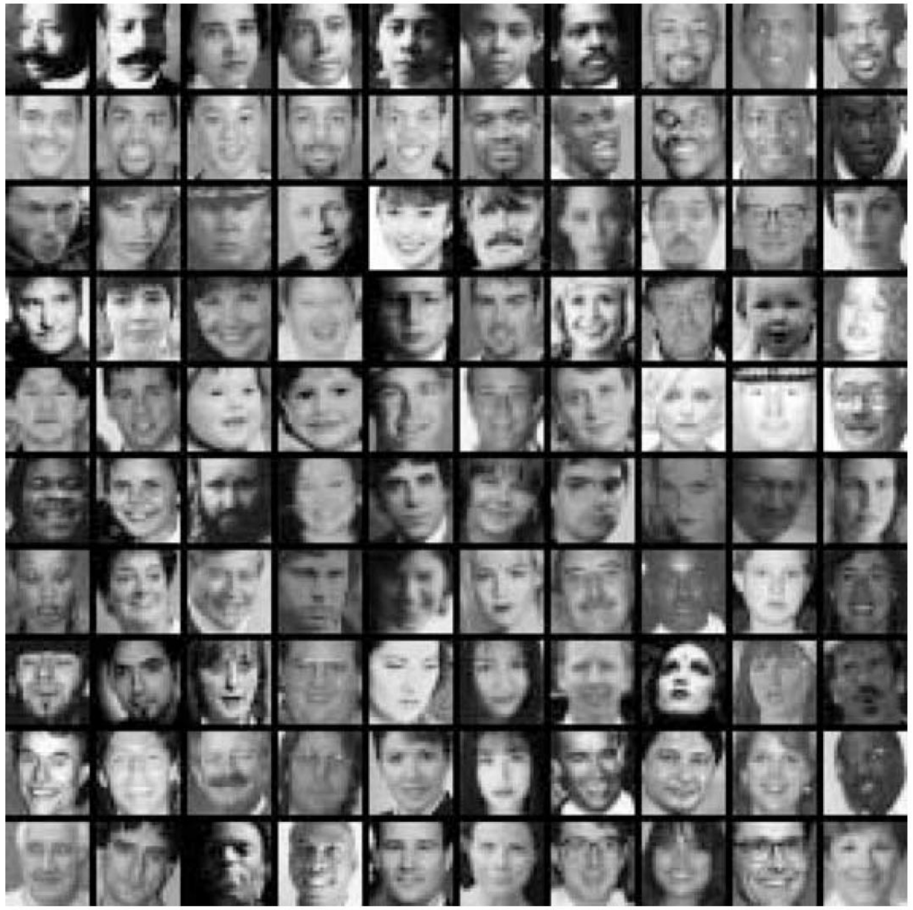
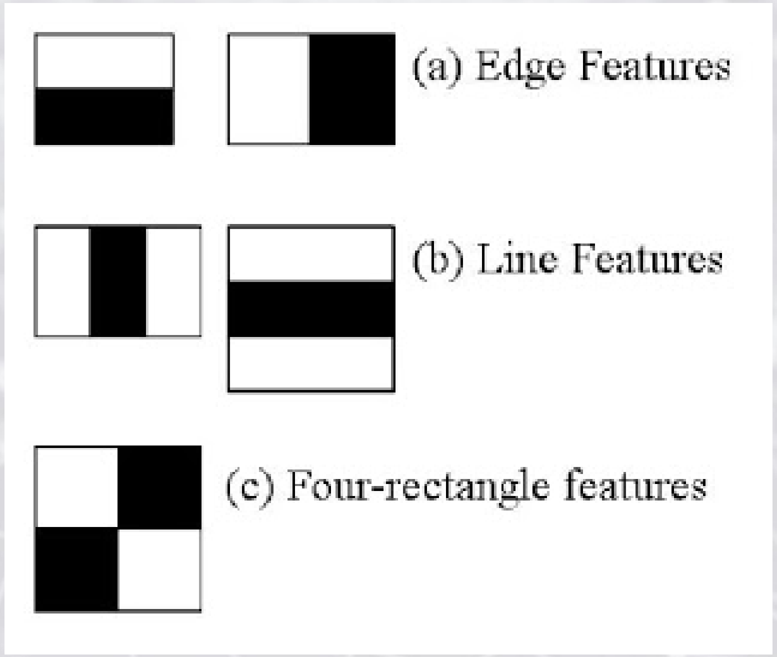
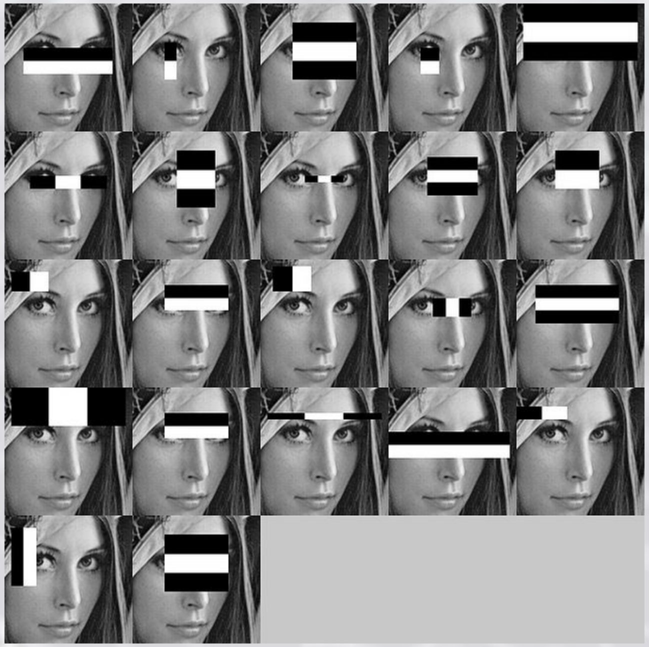


Figure 8. Example of frontal upright face images used for training.

Features




From http://makematics.com/research/viola-jones

Use rectangle features at multiple sizes and location in an image subwindow (candidate face).

For each feature f_j :

$$f_j = \sum_{b \in \text{black pixels}} \text{intensity(pixel } b) - \sum_{w \in \text{white pixels}} \text{intensity(pixel } w)$$

Possible number of features per 24 x 24 pixel subwindow > 180,000.

Detecting faces

Given a new image:

- Scan image using subwindows at all locations and at different scales
- For each subwindow, compute features and send them to an ensemble classifier (learned via boosting). If classifier is positive ("face"), then detect a face at this location and scale.

Viola-Jones Face Detection Algorithm

Preprocessing: Viola & Jones use a clever pre-processing step that allows the rectangular features to be computed very quickly. (See their paper for description.)

They use a variant of AdaBoost to both select a small set of features and train the classifier.

Viola-Jones Face Detection Algorithm

<u>Base ("weak) classifiers</u>:

For each feature f_j ,

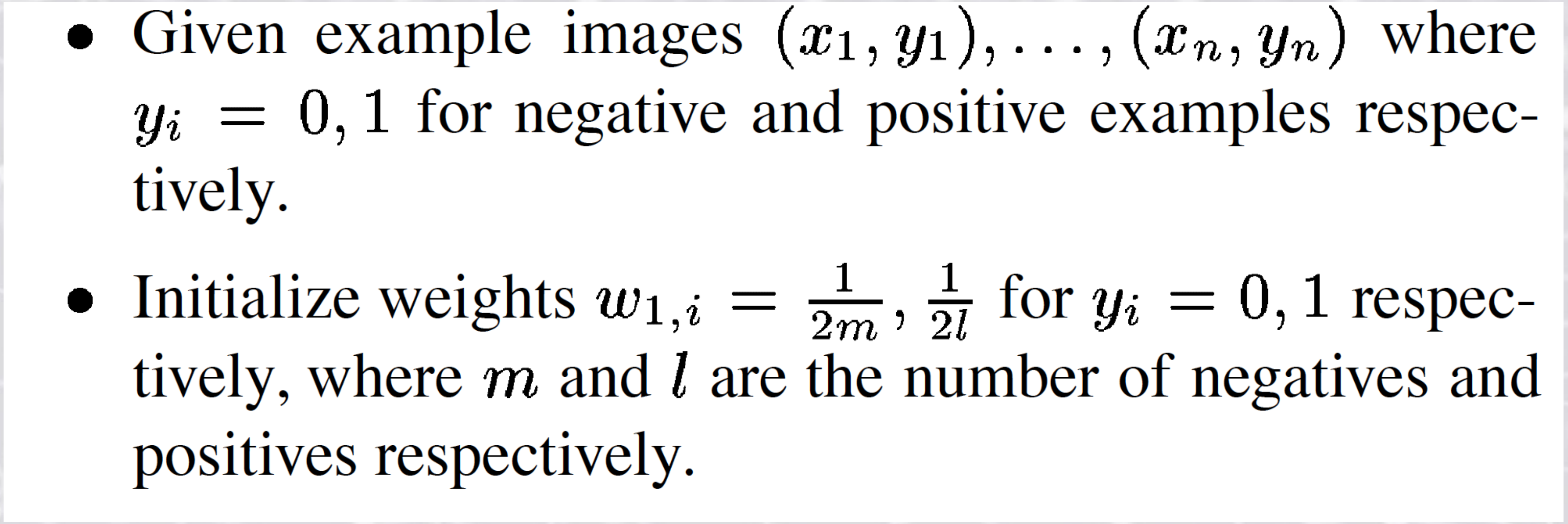
$$h_j = \left\{ \begin{array}{c} 1 \text{ if } p_j f_j(x) < p_j q_j \\ -1 \text{ otherwise} \end{array} \right\}$$

where x is a 24 x 24-pixel subwindow of an image, θ_j is the threshold that best separates the data using feature f_j , and p_j is either -1 or 1.

Such features are called **decision stumps**.

Viola-Jones Face Detection Algorithm

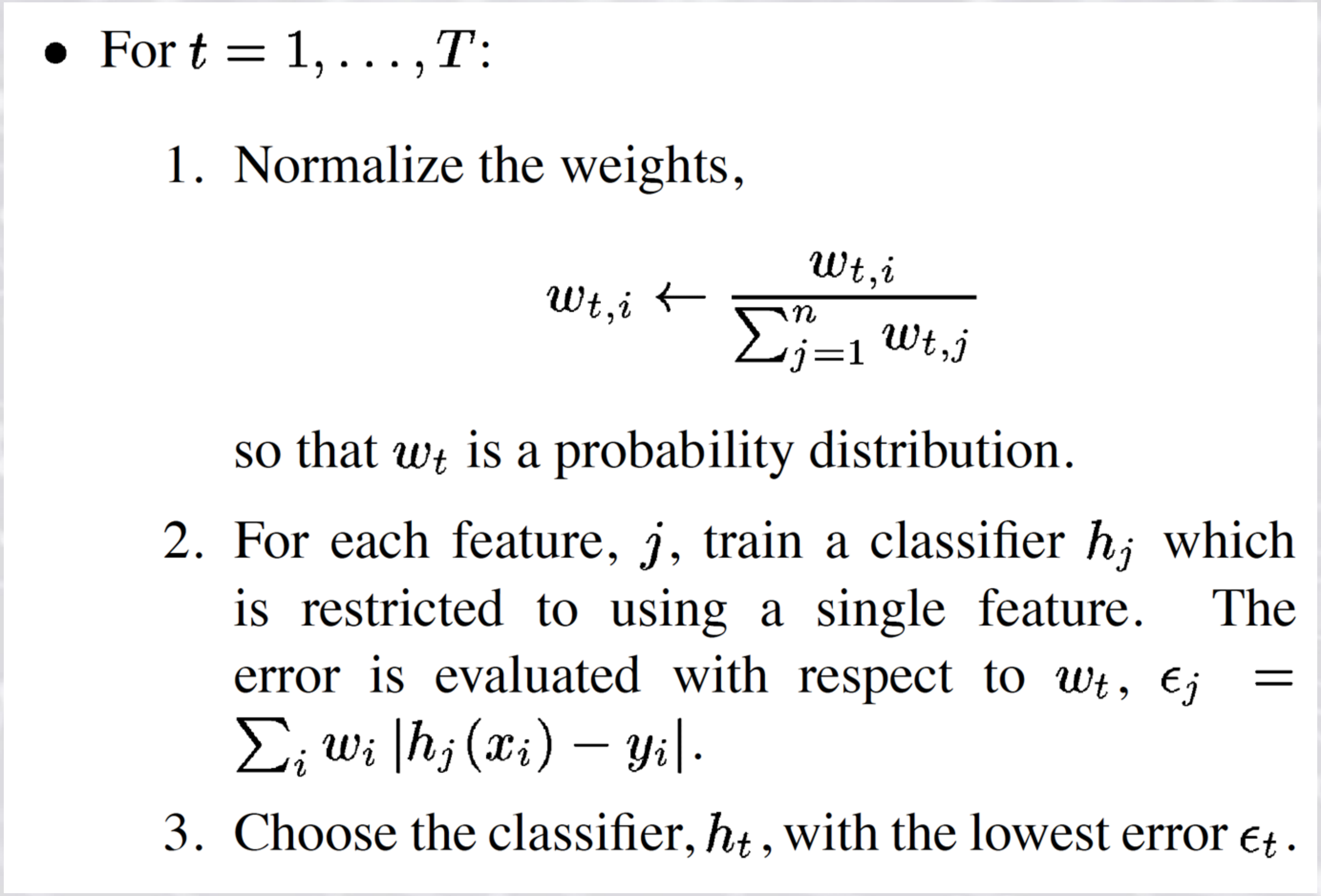
Boosting algorithm:



- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.

Viola-Jones Face Detection Algorithm

Boosting algorithm:

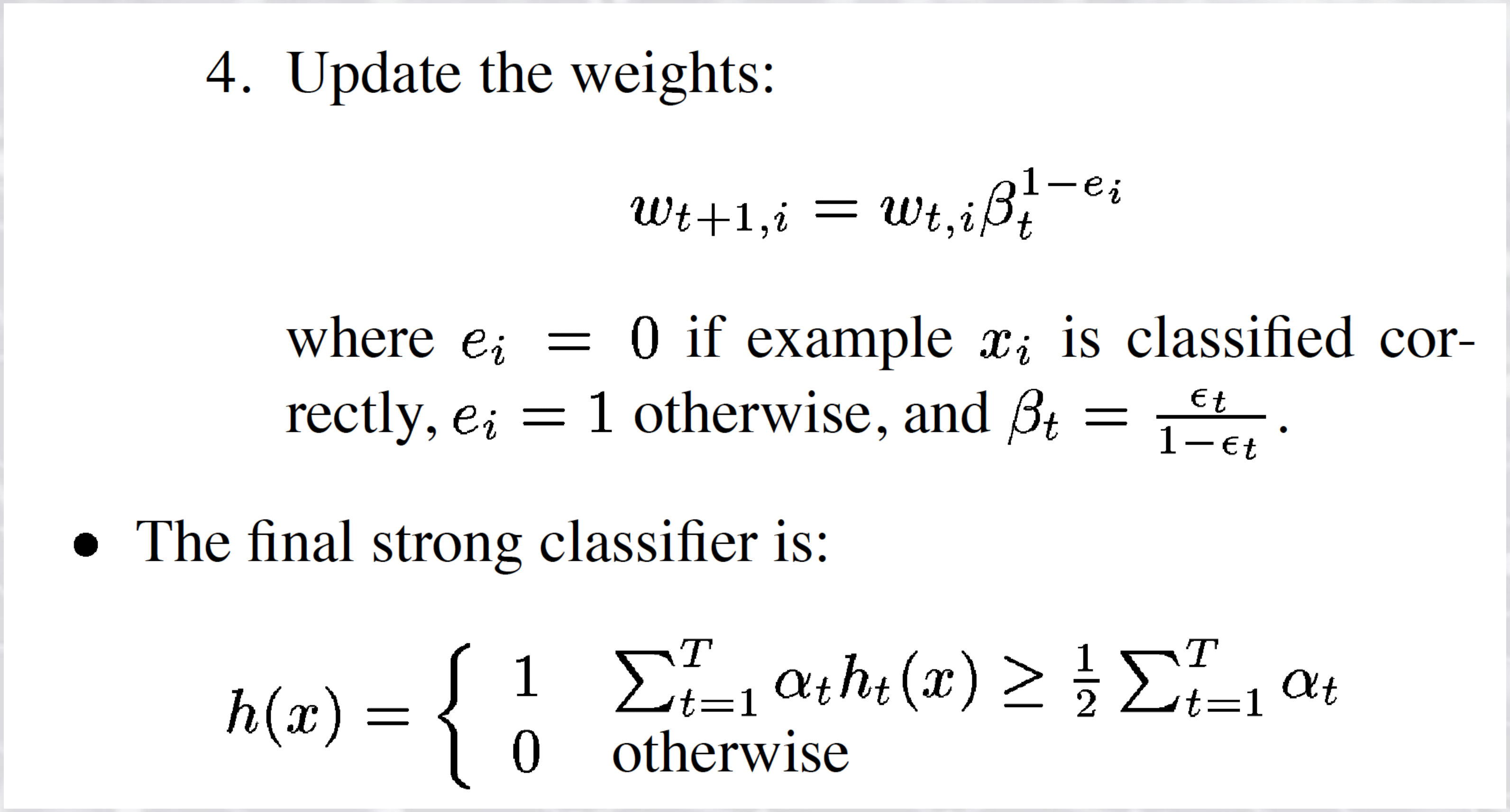


- For $t = 1, \ldots, T$:

  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.

Viola-Jones Face Detection Algorithm

Boosting algorithm:



4. Update the weights:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

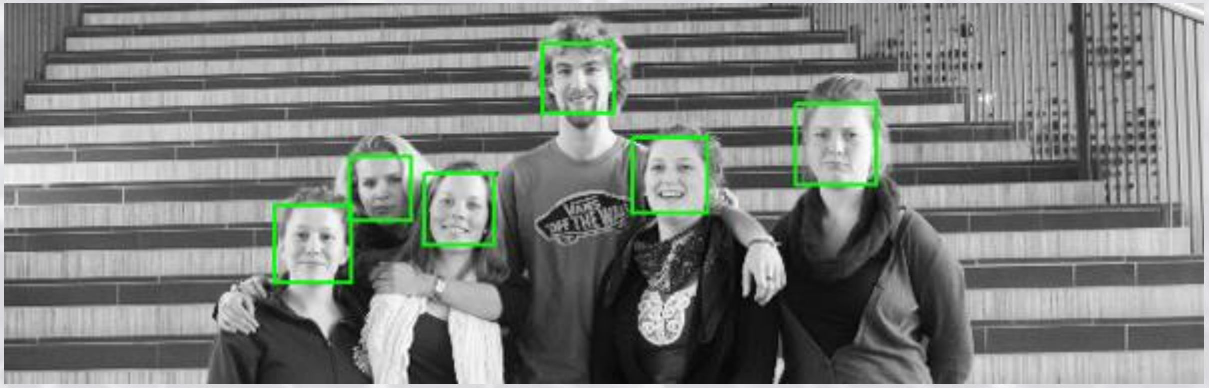
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $a_t = \ln \frac{1}{b_t}$

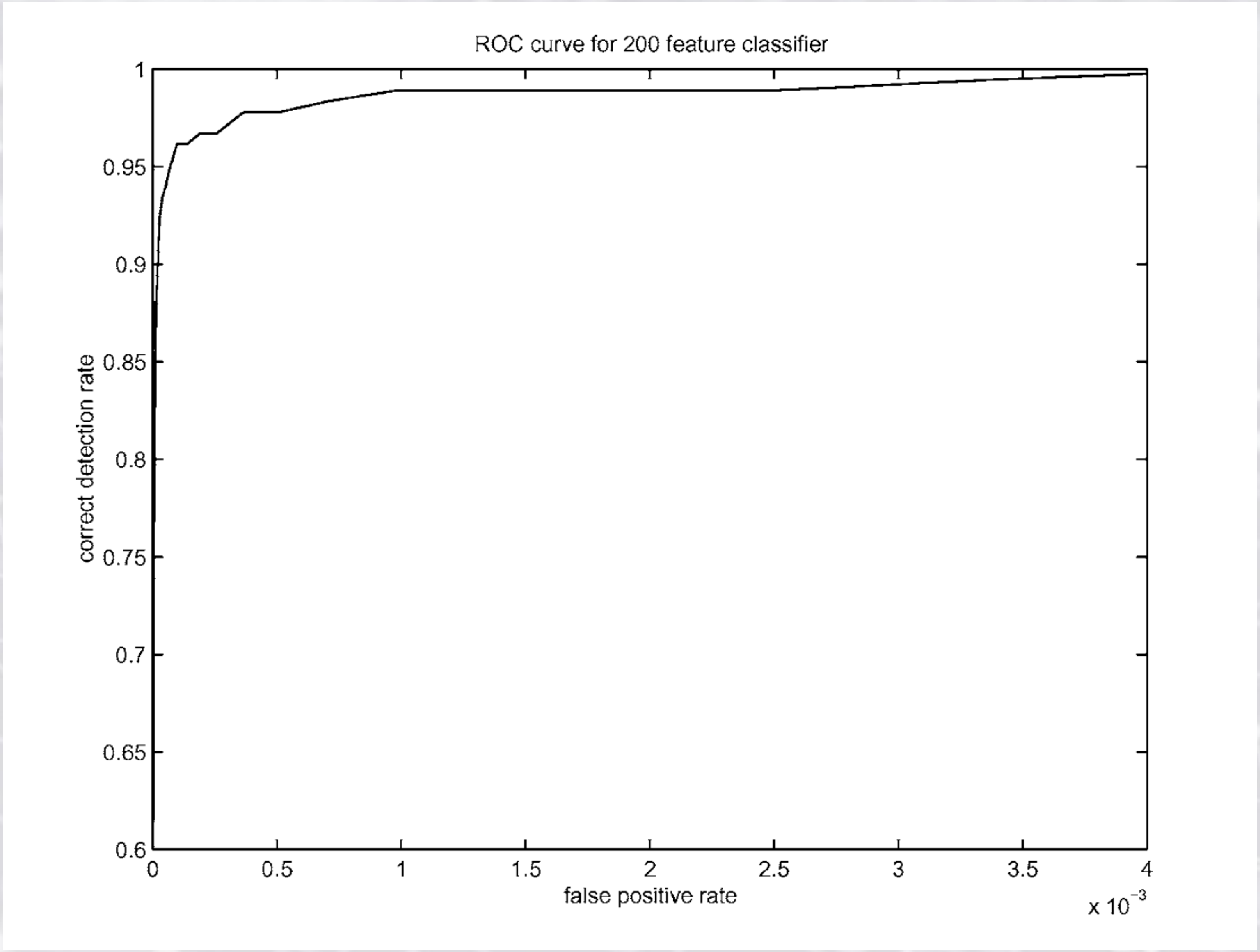
Note that only the top T features are used.

Viola-Jones Face Detection Algorithm



https://www.youtube.com/watch?v=k3bJUP0ct08

https://www.youtube.com/watch?v=c0twACIJYm8


ROC curve for 200 feature classifier
correct detection rate
false positive rate
x 10^-3
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
0
0.5
1
1.5
2
2.5
3
3.5
4

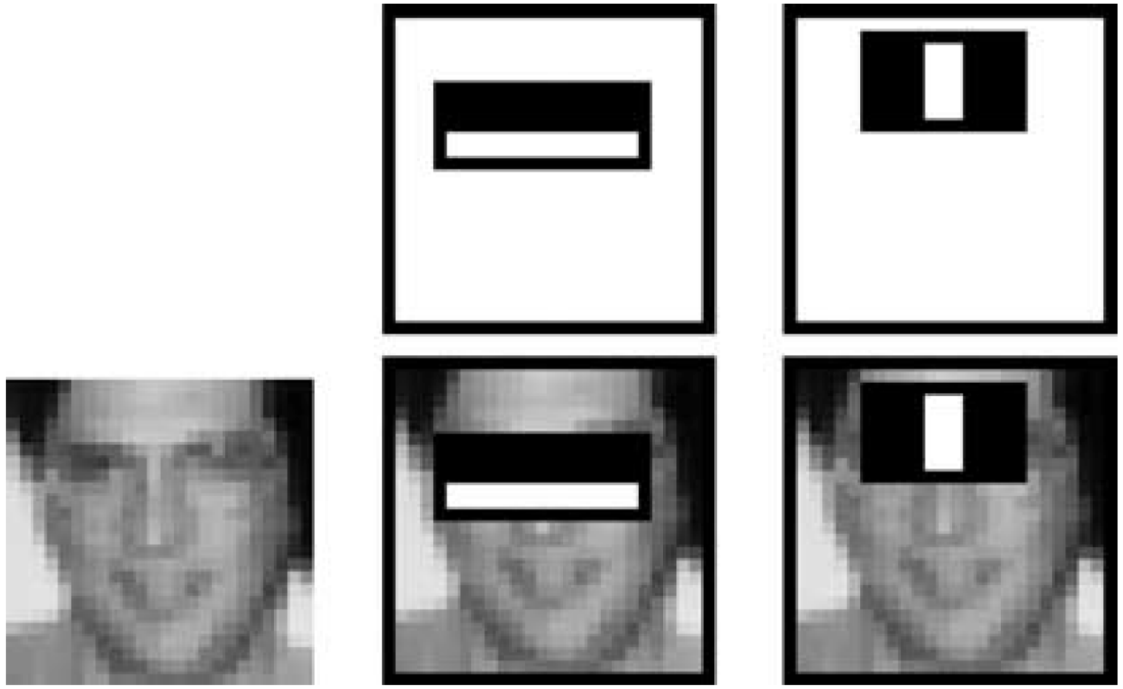


Figure 5. The first and second features selected by AdaBoost. The two features are shown in the top row and then overlayed on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

Viola-Jones Face Detection Algorithm

Advantages

- Fast feature computation
- Efficient feature selection
- Scale and Location Invariant detector
- Instead of scaling image, scales features
- Can be trained for other types of objects

Disadvantages

- Detector is most effective only on frontal images of face
- Sensitive to lighting conditions
- Can generate multiple detections for the same face

Model Combination Schemes: Mixture of Experts

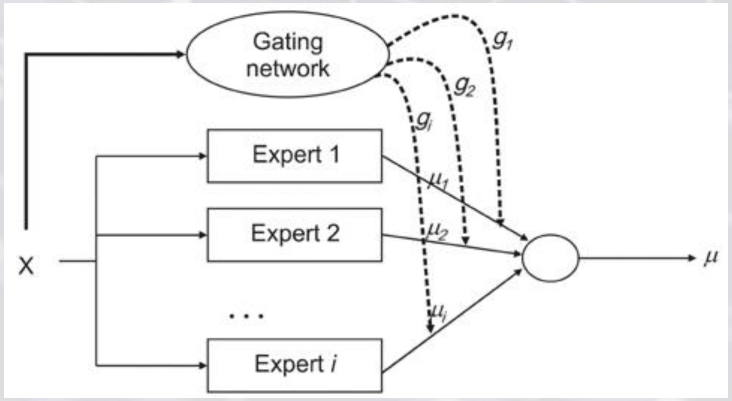
• In voting, the weights w_j are <u>constant over the input space</u> (e.g. $w_j=1/L$ for all j).

• By contrast, in the **mixture of experts** (Jordan *et al.*, 1994)architecture (a *local method*), there is a **gating function/network**, *g*, whose outputs are the weights of the experts. This gating function is dependent upon the input, hence <u>the weights of the experts need not be constant over the input space</u> in this regime.

• Mixture of experts can be learned via competitive learning algorithm so that <u>each base-learner becomes an expert in a different part of the input space</u>; to wit, if $g_j(x)= w_j(x)$ (for base-learner d_j when $x \in$ domain(d_j)), then the final output is a weighted average as in voting:

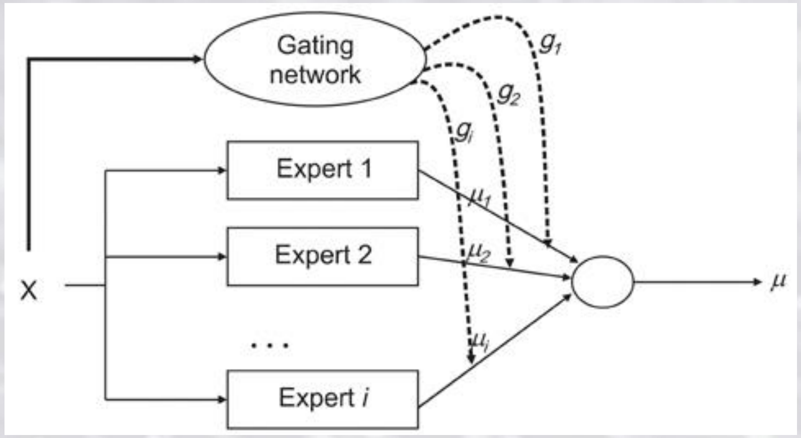
$$y = \sum_j g_j(x) d_j$$

(*) In practice, the weight $w_j(x)$ is close to 1 in base-learner j's region of expertise.



Jordan: UCB

Model Combination Schemes: Mixture of Experts



• In the mixture of experts architecture, experts are biased but negatively correlated.

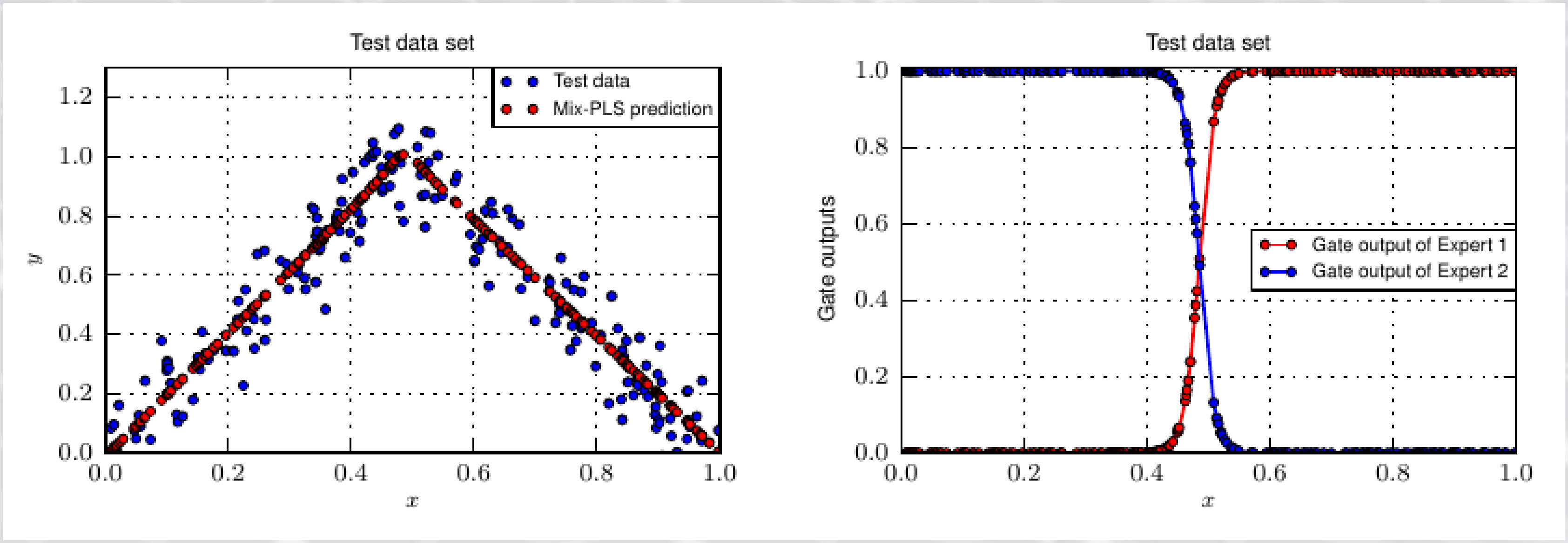
(*) As training proceeds, bias decreases and expert variances increase but at the same time as <u>experts localize in different parts of the input space</u>, their covaraiances get more negative, which <u>decreases the total variance, and hence the overall error</u>.

Model Combination Schemes: Mixture of Experts

• Learning a mixture of experts (MoE) architecture boils down to: (1) learning the parameters of individual learners and (2) learning parameters of the gating network.

(*) The most common approach to learning an MoE model is to use a variant of the EM (here the E-step updates the gating function and mixing weights, while the M-step maximizes the individual learner parameters).

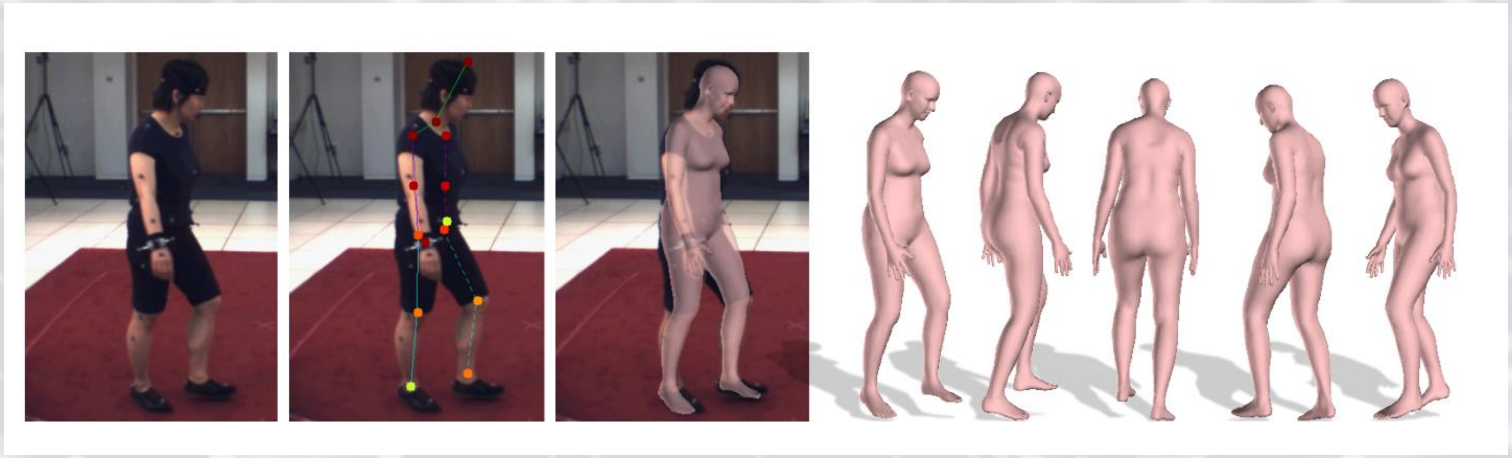
(*) Alternatively, one can also use *dynamic programming* to solve **segmented regression** and related problems.



Model Combination Schemes: Mixture of Experts

- Another example from computer vision research:

Bogo, et al., "Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image", *ECCV*, 2016.



(*) Researchers developed the first method to automatically estimate the 3D pose of the human body as well as its 3D shape from a single 2D image. Methodology: Use CNN to predict 2D joint body locations, then use a mixture of experts (3D Gaussians) for pose estimation.

http://files.is.tue.mpg.de/black/papers/BogoECCV2016.pdf

Model Combination Schemes: Cascading

• The idea in cascaded classifiers is to have a sequence of base-classifiers d_j sorted in terms of their space or time complexity, or the cost of the representation they use, so that d_{j+1} is sotlier than dj.

• Cascading is a multistage method, and we use d_j only if all the preceding learners, d_k, $k<j$ are not confident.

• Each learner has an associated confidence weight (w_j) such that we say d_j is confident of its output and can be used if $w_j > \theta_j$ (or some such thresholding criterion) and for each confidence threshold it holds that: $\theta_j \le \theta_{j+1} < 1$.

• In classification, the confidence function is set to the highest posterior: $w_j = \max_i d_{ji}$.

• We use learner d_j if all the preceding learners are not confident: $y_i = d_{ji}$ if $w_j > \theta_j$ and for all k, $k < j$, $w_k < \theta_k$.

Model Combination Schemes: Cascading

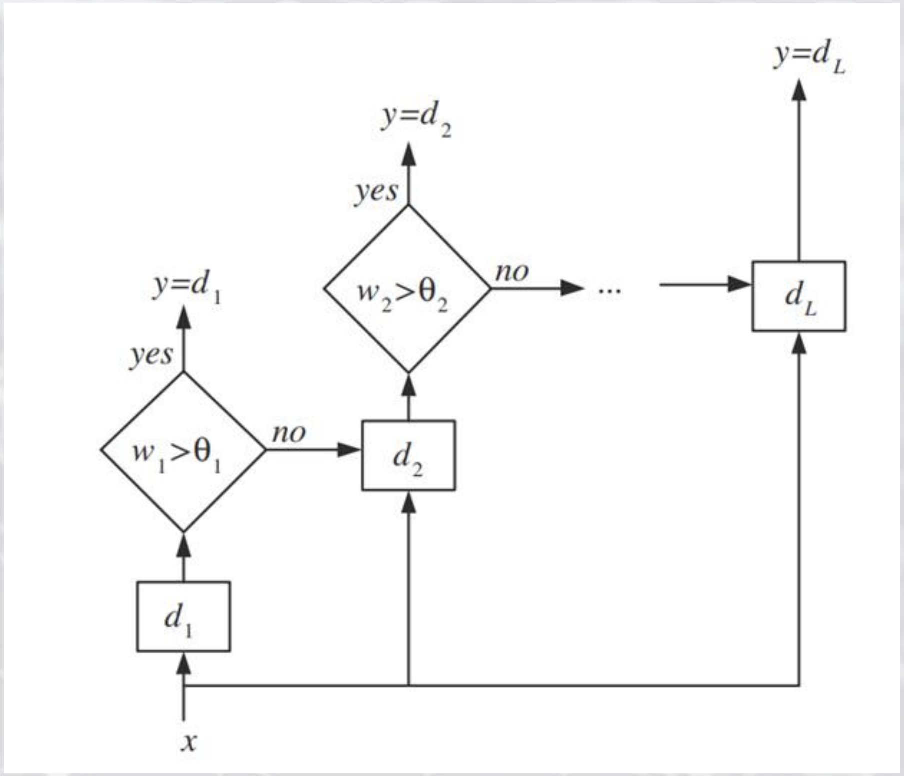
• Starting with j=1, given a training set, we train d_j. Then we find al instances from a separate validation set on which d_j is not confident, and these constitute the training set of d_{j+1}. Note that unlike AdaBoost, we choose not only the misclassified instances but the ones for which the previous base-learner is not confident.

• This then covers misclassified instances as well as the instances for which the posterior is not high enough; these are instances on the right side of the boundary but for which the distance to the discriminant, namely, the *margin*, is not large enough.

(*) **Basic idea**: early simple classifier handles the majority of instances, and a more complex classifier is used only for a small percentage of the data, thereby not significantly increasing the overall model complexity.

(*) Note that this procedure differs from other multi-expert methods such as voting where all base-learners generate their output for any instance.

Model Combination Schemes: Cascading

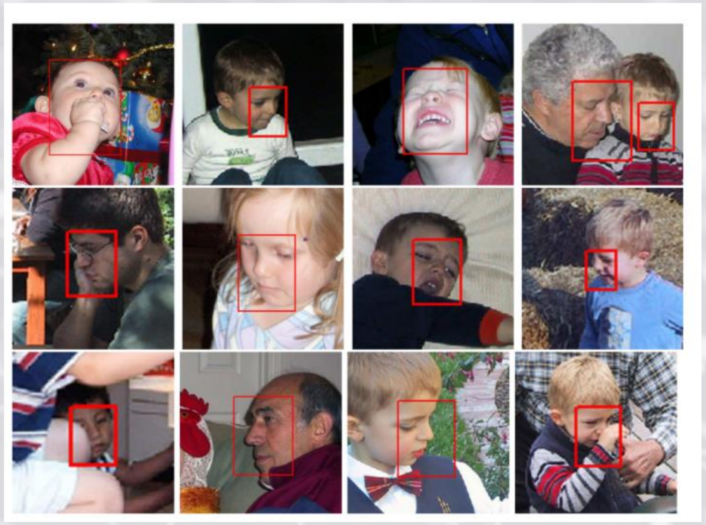


• The **inductive bias** of cascading is that the classes can be explained by a small number of "rules" in increasing complexity.

(*) Cascading represents a <u>middle ground between two extremes of parametric and non-parametric classification</u>. Whereas, say, a regression model (parametric model) applies a single rule to all data, k-NN, for example (non-parametric model), stores the entire training set in memory without generating any parsimonious "rules," cascading instead generates rules to explain most of the data as cheaply as possible (and usually stores the rest of the data as exceptions).

Model Combination Schemes: Robust Object Detection with Cascading

• L. Bourdev, et al., "Robust Object Detection via Soft Cascade", CVPR, 2005.



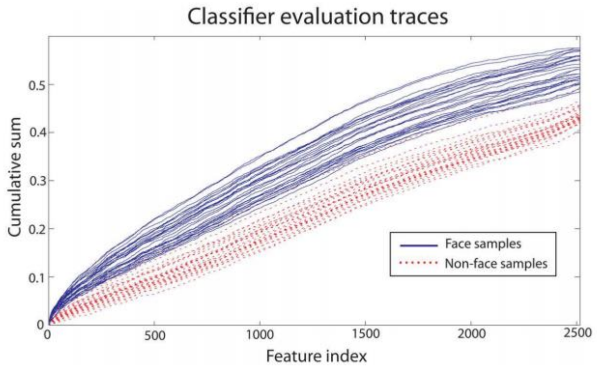


Figure 2: *Sample traces*

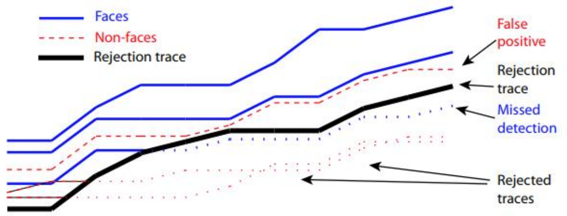


Figure 3: *Sample evaluation using Soft Cascade*

(*) Researchers developed a method for training object detectors using a generalization of the cascade architecture, resulting in a detection rate and speed comparable to state-of-the-art. Specifically, the classifier is an ensemble consisting of a sum of thresholded classifiers selected during AdaBoost training scaled by the associated weights; the partial sums of the classifiers generate a *sample trace* (shown); the cumulative sums for the two classes (face/non-face) separate as the evaluation progresses.
http://lubomir.org/academic/softcascade.pdf

Fin