Viola-Jones Detector

Objectives

- Classification step should be computationally efficient.
- Expensive training affordable.

Strategy

- Extract very large set of measurements (features), i.e. d in $\mathbb{R}^d$ large.
- Use Boosting with decision stumps.
- From Boosting weights, select small number of important features.
- Class imbalance: Use Cascade.

Classification step

Compute only the selected features from input image.

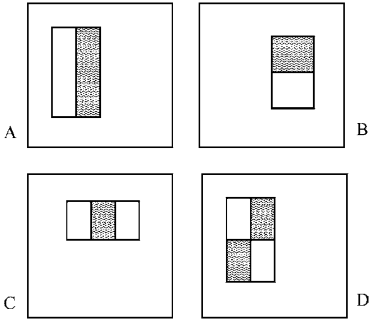
Feature Extraction

Extraction method

1. Enumerate possible windows (different shapes and locations) by $j = 1, \dots, d$.
2. For training image i and each window j, compute

$$x_{ij} := \text{average of pixel values in gray block(s)} - \text{average of pixel values in white block(s)}$$

3. Collect values for all j in a vector $\mathbf{x}_i := (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$.

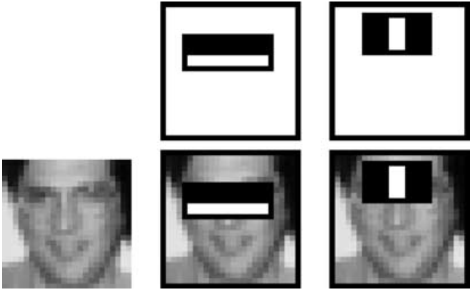


The dimension is huge

- One entry for (almost) every possible location of a rectangle in image.
- Start with small rectangles and increase edge length repeatedly by 1.5.
- In Viola-Jones paper: Images are 384×288 pixels, $d \approx 160000$.

Selected Features

First two selected features



200 features are selected in total.

Training the Cascade

Training procedure

1. User selects acceptable rates (FPR and DR) for each level of cascade.
2. At each level of cascade:
   - Train boosting classifier.
   - Gradually increase number of selected features until rates achieved.

Use of training data

Each training step uses:

- All positive examples (= faces).
- Negative examples (= non-faces) misclassified at previous cascade layer.

EXAMPLE RESULTS

Table 3. Detection rates for various numbers of false positives on the MIT + CMU test set containing 130 images and 507 faces.

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2%	93.7%	–
Rowley-Baluja-Kanade	83.2%	86.0%	–	–	–	89.2%	90.1%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–

Additive View of Boosting

Basis function interpretation

The boosting classifier is of the form

$$f(\mathbf{x}) = \operatorname{sgn}(F(\mathbf{x})) \qquad \text{where} \qquad F(\mathbf{x}) := \sum_{m=1}^{M} \alpha_m g_m(\mathbf{x}) \; .$$

- A linear combination of functions $g_1, \ldots, g_m$ can be interpreted as a representation of F using the **basis functions** $g_1, \ldots, g_m$.
- We can interpret the linear combination $F(\mathbf{x})$ as an approximation of the decision boundary using a basis of weak classifiers.
- To understand the approximation, we have to understand the coefficients α_m.

Boosting as a stage-wise minimization procedure

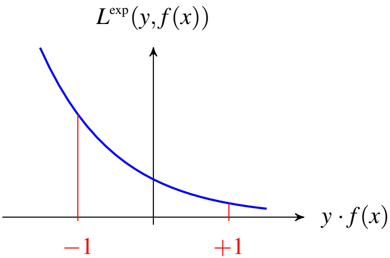
It can be shown that α_m is obtained by minimizing a risk,

$$(\alpha_m, g_m) := \arg\min_{\alpha'_m, g'_m} \hat{R}_n(F^{\text{(m-1)}} + \alpha'_m g'_m)$$

under a specific loss function, the **exponential loss**. Notation: $F^{\text{(m)}} := \sum_{j \leq m} \alpha_m g_m$.

Definition

$$L^{\exp}(y, f(x)) := \exp(-y \cdot f(x))$$



Relation to indicator function

$$y \cdot f(x) = \begin{cases} +1 & x \text{ correctly classified} \\ -1 & x \text{ misclassified} \end{cases}$$

This is related to the indicator function we have used so far by

$$-y \cdot f(x) = 2 \cdot \mathbb{I}\{f(x) \neq y\} - 1$$

Additive Perspective

Exponential loss risk of additive classifier

Our claim is that AdaBoost minimizes the empirical risk under L^{exp},

$$\hat{R}_n(F^{(m-1)} + \beta_m g_m) = \frac{1}{n}\sum_{i=1}^{n} \exp(\overbrace{-y_i F^{(m-1)}}^{\text{fixed in } m\text{th step}} - \underbrace{y_i \beta_m g_m(\mathbf{x}_i)}_{\text{we only have to minimize here}})$$

Relation to AdaBoost

It can be shown that the classifier obtained by solving

$$\arg\min_{\beta_m, g_m} \hat{R}_n(F^{(m-1)} + \beta_m g_m)$$

at each step m yields the AdaBoost classifier.

AdaBoost as Additive Model

More precisely, it can be shown:

If we build a classifier $F(\mathbf{x}) := \sum_{m=1}^{M} \beta_m g_m(\mathbf{x})$ which minimizes

$$\hat{R}_n(F^{\text{(m-1)}}(\mathbf{x}) + \beta_m g_m(\mathbf{x}))$$

at each step m, we have to choose:

- g_m as the classifier which minimizes the weighted misclassifiation rate.
- $\beta_m = \frac{1}{2} \log \frac{1-\text{err}_m}{\text{err}_m} = \frac{1}{2}\alpha_m$
- $w_i^{\text{(m+1)}} := w_i^{\text{(m)}} \exp(-y_i \beta_m g_m(\mathbf{x}_i))$

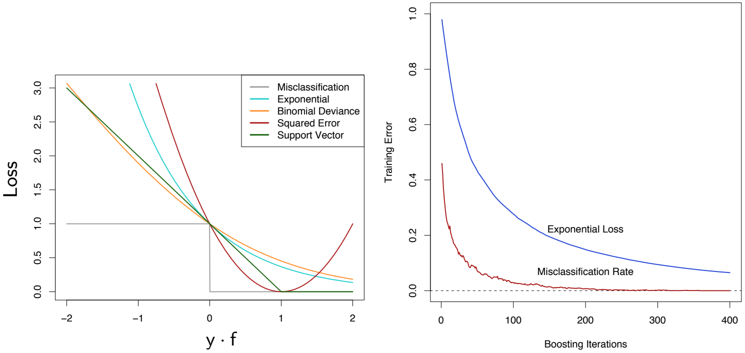
This is precisely equivalent to what AdaBoost does.

In other words

AdaBoost approximates the Bayes-optimal classifier (under exponential loss) using a basis of weak classifiers.

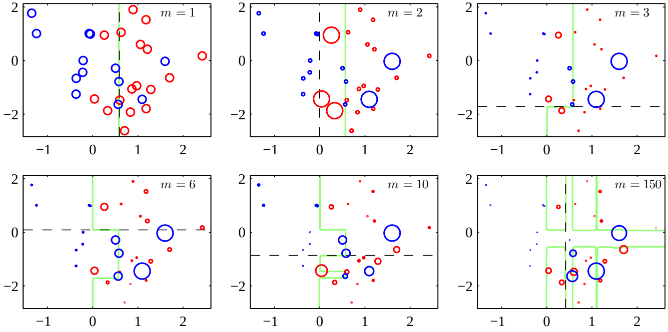
- Since we do not know the true risk, we approximate by the empirical risk.
- Each weak learner optimizes 0-1 loss on *weighted* data.
- Weights are chosen so that procedure overall optimizes *exponential* loss risk.

Loss functions



- The right figure shows the misclassification rate and the average exponential loss on the same data as number of weak learners increases.
- From the additive model perspective, the exponential loss helps explain why prediction error continues to improve when training error is already optimal.

ILLUSTRATION



Circle = data points, circle size = weight.

Dashed line: Current weak learner. Green line: Aggregate decision boundary.

Bagging and Random Forests

Background: Resampling Techniques

We briefly review a technique called bootstrap on which Bagging and random forests are based.

Bootstrap

Bootstrap (or **resampling**) is a technique for improving the quality of estimators.

Resampling = sampling from the empirical distribution

Application to ensemble methods

- We will use resampling to generate weak learners for classification.
- We discuss two classifiers which use resampling: Bagging and random forests.
- Before we do so, we consider the traditional application of Bootstrap, namely improving estimators.

BOOTSTRAP: BASIC ALGORITHM

Given

- A sample $\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n$.
- An estimator $\hat{S}$ for a statistic S.

Bootstrap algorithm

1. Generate B **bootstrap samples** $\mathcal{B}_1, \dots, \mathcal{B}_B$. Each bootstrap sample is obtained by sampling n times with replacement from the sample data. (Note: Data points can appear multiple times in any $\mathcal{B}_b$.)
2. Evaluate the estimator on each bootstrap sample:

$$\hat{S}_b := \hat{S}(\mathcal{B}_b)$$

   (That is: We estimate S pretending that $\mathcal{B}_b$ is the data.)

3. Compute the **bootstrap estimate** of S by averaging over all bootstrap samples:

$$\hat{S}_{\mathrm{BS}} := \frac{1}{B} \sum_{b=1}^{B} \hat{S}_b$$

Example: Variance Estimation

Mean and Variance

$$\mu := \int_{\mathbb{R}^d} \mathbf{x}\, p(\mathbf{x}) d\mathbf{x} \qquad \sigma^2 := \int_{\mathbb{R}^d} (\mathbf{x} - \mu)^2 p(\mathbf{x}) d\mathbf{x}$$

Plug-in estimators for mean and variance

$$\hat{\mu} := \frac{1}{n} \sum_{i=1}^{n} \tilde{\mathbf{x}}_i \qquad \hat{\sigma}^2 := \frac{1}{n} \sum_{i=1}^{n} (\tilde{\mathbf{x}}_i - \hat{\mu})^2$$

Bootstrap Variance Estimate

Bootstrap algorithm

1. For $b = 1, \dots, B$, generate a boostrap sample $\mathcal{B}_b$. In detail:
   For $i = 1, \dots, n$:
   - Sample an index $j \in \{1, \dots, n\}$.
   - Set $\tilde{\mathbf{x}}_i^{(b)} := \tilde{\mathbf{x}}_j$ and add it to $\mathcal{B}_b$.
2. For each b, compute mean and variance estimates:

$$\hat{\mu}_b := \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i^{(b)} \qquad\qquad \hat{\sigma}_b^2 := \frac{1}{n}\sum_{i=1}^{n} (\tilde{\mathbf{x}}_i^{(b)} - \hat{\mu}_b)^2$$

3. Compute the bootstrap estimate:

$$\hat{\sigma}_{\mathrm{BS}}^2 := \frac{1}{B}\sum_{b=1}^{B} \hat{\sigma}_b^2$$

How Often Do We See Each Sample?

Sample $\{\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n\}$, bootstrap resamples $\mathcal{B}_1, ..., \mathcal{B}_B$.

In how many sets does a given $\mathbf{x}_i$ occur?

Probability for $\mathbf{x}_i$ *not* to occur in n draws:

$$\Pr\{\tilde{\mathbf{x}}_i \notin \mathcal{B}_b\} = (1 - \frac{1}{n})^n$$

For large n:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679$$

- Asymptotically, any $\tilde{\mathbf{x}}_i$ will appear in $\sim 63\%$ of the bootstrap resamples.
- Multiple occurrences possible.

How often is $\tilde{\mathbf{x}}_i$ expected to occur?

The *expected* number of occurences of each $\tilde{\mathbf{x}}_i$ is B.

Bootstrap estimate averages over reshuffled samples.

Bootstrap: Applications

Estimate variance of estimators

- Since estimator $\hat{S}$ depends on (random) data, it is a random variable.
- The more this variable scatters, the less we can trust our estimate.
- If scatter is high, we can expect the values $\hat{S}_b$ to scatter as well.
- In previous example, this means: Estimating the variance of the variance estimator.

Variance reduction

- Averaging over the individual bootstrap samples can reduce the variance in $\hat{S}$.
- In other words: $\hat{S}_{\text{BS}}$ typically has lower variance than $\hat{S}$.
- This is the property we will use for classicifation in the following.

As alternative to cross validation

To estimate prediction error of classifier:

- For each b, train on $\mathcal{B}_b$, estimate risk on points not in $\mathcal{B}_b$.
- Average risk estimates over bootstrap samples.

Idea

- Recall Boosting: Weak learners are deterministic, but selected to exhibit high variance.
- Strategy now: Randomly distort data set by resampling.
- Train weak learners on resampled training sets.
- Resulting algorithm: **Bagging** (= **B**ootstrap **agg**regation)

REPRESENTATION OF CLASS LABELS

For Bagging with K classes, we represent class labels as vectors:

$$\mathbf{x}_i \text{ in class } k \qquad \text{as} \qquad y_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \longleftarrow k\text{th entry}$$

This way, we can average together multiple class labels:

$$\frac{1}{n}(y_1 + \ldots + y_n) = \begin{pmatrix} p_1 \\ \vdots \\ p_k \\ \vdots \\ p_K \end{pmatrix}$$

We can interpret p_k as the probability that one of the n points is in class k.

Bagging: Algorithm

Training

For $b = 1, \dots, B$:

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from training data.
2. Train a classifier f_b on $\mathcal{B}_b$.

Classification

- Compute

$$f_{\text{avg}}(\mathbf{x}) := \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$$

  This is a vector of the form $f_{\text{avg}}(\mathbf{x}) = (p_1(\mathbf{x}), \dots, p_k(\mathbf{x}))$.

- The Bagging classifier is given by

$$f_{\text{Bagging}}(\mathbf{x}) := \arg\max_k \{p_1(\mathbf{x}), \dots, p_k(\mathbf{x})\} \ ,$$

  i.e. we predict the class label which most weak learners have voted for.