Lecture 16: Boosting

Reading: Sections 10.1 - 10.7

GU4241/GR5241 Statistical Machine Learning

Linxi Liu

April 12, 2019

Boosting

- Arguably the most popular (and historically the first) ensemble method.
- Weak learners can be trees, Perceptrons, etc.
- Requirement: It must be possible to train the weak learner on a *weighted* training set.

Overview

- Boosting adds weak learners one at a time.
- A weight value is assigned to each training point.
- At each step, data points which are currently classified correctly are weighted down (i.e. the weight is smaller the more of the weak learners already trained classify the point correctly).
- The next weak learner is trained on the *weighted* data set: In the training step, the error contributions of misclassified points are multiplied by the weights of the points.
- Roughly speaking, each weak learner tries to get those points right which are currently not classified correctly.

Training With Weights

Example: Decision stump

A decision stump classifier for two classes is defined by

$$f(\mathbf{x} \,|\, j, t) := \begin{cases} +1 & x^{(j)} > t \\ -1 & \text{otherwise} \end{cases}$$

where $j \in \{1, \ldots, d\}$ indexes an axis in $\mathbb{R}^d$.

Weighted data

- Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.
- With each data point $\mathbf{x}_i$ we associate a weight $w_i \geq 0$.

Training on weighted data

Minimize the *weighted* misclassifcation error:

$$(j^*, t^*) := \arg\min_{j,t} \frac{\sum_{i=1}^{n} w_i \mathbb{I}\{y_i \neq f(\mathbf{x}_i | j, t)\}}{\sum_{i=1}^{n} w_i}$$

AdaBoost

Input

- Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$
- Algorithm parameter: Number M of weak learners

Training algorithm

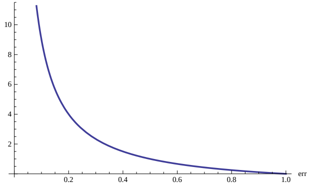
1. Initialize the observation weights $w_i = \frac{1}{n}$ for $i = 1, 2, ..., n$.
2. For $m = 1$ to M:
   2.1 Fit a classifier $g_m(x)$ to the training data using weights w_i.
   2.2 Compute
   $$\text{err}_m := \frac{\sum_{i=1}^{n} w_i \mathbb{I}\{y_i \neq g_m(x_i)\}}{\sum_i w_i}$$
   2.3 Compute $\alpha_m = \log(\frac{1-\text{err}_m}{\text{err}_m})$
   2.4 Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$ for $i = 1, 2, ..., n$.
3. Output
   $$f(x) := \text{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$

AdaBoost

Weight updates

$$\alpha_m = \log\left(\frac{1-\mathsf{err}_m}{\mathsf{err}_m}\right)$$

$$w_i^{(\mathsf{m})} = w_i^{(\mathsf{m-1})} \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq g_m(x_i)))$$



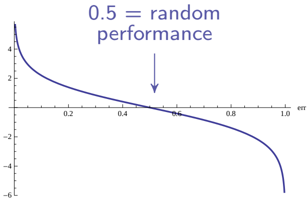
$\frac{1-\mathsf{err}_m}{\mathsf{err}_m}$

Hence:

$$w_i^{(\mathsf{m})} = \begin{cases} w_i^{(\mathsf{m-1})} & \text{if } g_m \text{ classifies } x_i \text{ correctly} \\ w_i^{(\mathsf{m-1})} \cdot \frac{1-\mathsf{err}_m}{\mathsf{err}_m} & \text{if } g_m \text{ misclassifies } x_i \end{cases}$$

Weighted classifier

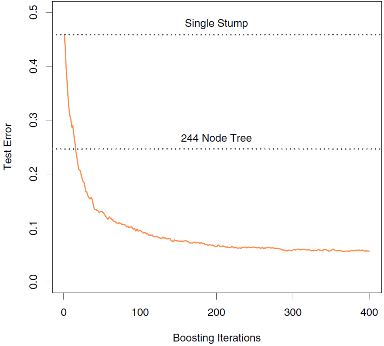
$$f(x) = \mathsf{sign}\left(\sum_{m=1}^{M} \alpha_m g_m(x)\right)$$



$\alpha_m = \log\left(\frac{1-\mathsf{err}_m}{\mathsf{err}_m}\right)$

Example

AdaBoost test error (simulated data)



- Weak learners used are decision stumps.
- Combining many trees of depth 1 yields much better results than a single large tree.

Properties

- AdaBoost is one of most widely used classifiers in applications.
- Decision boundary is non-linear.
- Can handle multiple classes if weak learner can do so.

Test vs. training error

- Most training algorithms (e.g. decision trees) terminate when training error reaches minimum.
- AdaBoost weights keep changing even if training error is minimal.
- Interestingly, the *test error* typically keeps decreasing even *after* training error has stabilized at minimal value.
- It can be shown that this behavior can be interpreted in terms of a margin:
  - Adding additional classifiers slowly pushes overall f towards a maximum-margin solution.
  - May not improve training error, but improves generalization properties.
- This does *not* imply that boosting magically outperforms SVMs, only that minimal training error does not imply an optimal solution.

Boosting and Feature Selection

AdaBoost with Decision Stumps

- Once AdaBoost has trained a classifier, the weights α_m tell us which of the weak learners are important (i.e. classify large subsets of the data well).
- If we use Decision Stumps as weak learners, each f_m corresponds to one axis.
- From the weights α, we can read off which axis are important to separate the classes.

Boosting and Feature Selection

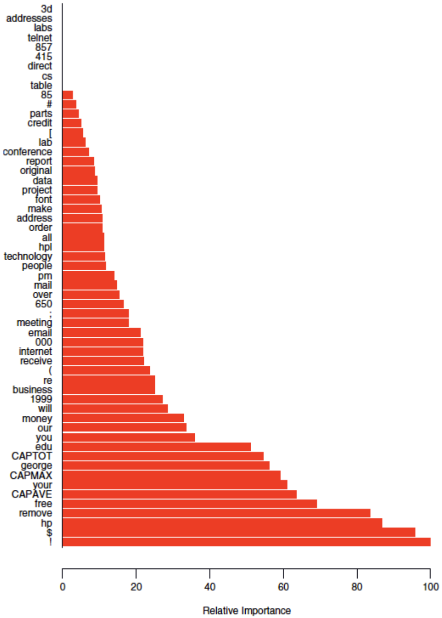
AdaBoost with Decision Stumps

- Once AdaBoost has trained a classifier, the weights α_m tell us which of the weak learners are important (i.e. classify large subsets of the data well).
- If we use Decision Stumps as weak learners, each f_m corresponds to one axis.
- From the weights α, we can read off which axis are important to separate the classes.

Terminology

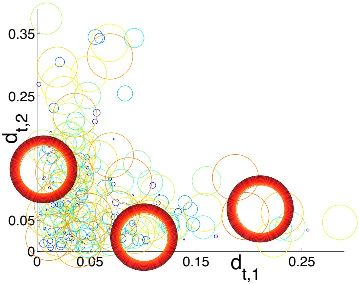
The dimensions of $\mathbb{R}^d$ (= the measurements) are often called the **features** of the data. The process of selecting features which contain important information for the problem is called **feature selection**. Thus, AdaBoost with Decision Stumps can be used to perform feature selection.

Spam Data



- Tree classifier: 9.3% overall error rate
- Boosting with decision stumps: 4.5%
- Figure shows feature selection results of Boosting.

Cycles



- An odd property of AdaBoost is that it can go into a cycle, i.e. the same sequence of weight configurations occurs over and over.
- The figure shows weights (called d_t by the authors of the paper, with t=iteration number) for two weak learners.
- Circle size indicates iteration number, i.e. larger circel indicates larger t.

Face Detection

Searching for faces in images

Two problems:

- **Face detection** Find locations of all faces in image. Two classes.
- **Face recognition** Identify a person depicted in an image by recognizing the face. One class per person to be identified + background class (all other people).

Face detection can be regarded as a solved problem. Face recognition is not fully solved.

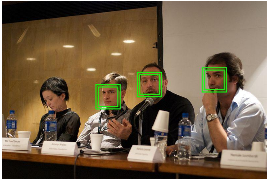
Face detection as a classification problem

- Divide image into patches.
- Classify each patch as "face" or "not face"

Classifier Cascades

Unbalanced Classes

- Our assumption so far was that both classes are roughly of the same size.
- Some problems: One class is much larger.
- Example: Face detection.
  - Image subdivided into small quadratic patches.
  - Even in pictures with several people, only small fraction of patches usually represent faces.



Standard classifier training

Suppose positive class is very small.

- Training algorithm can achieve good error rate by classifiying *all* data as negative.
- The error rate will be precisely the proportion of points in positive class.

Classifier Cascades

Addressing class imbalance

- We have to change cost function: False negatives (= classify face as background) expensive.
- Consequence: Training algorithm will focus on keeping proportion of false negatives small.
- Problem: Will result in many false positives (= background classified as face).

Cascade approach

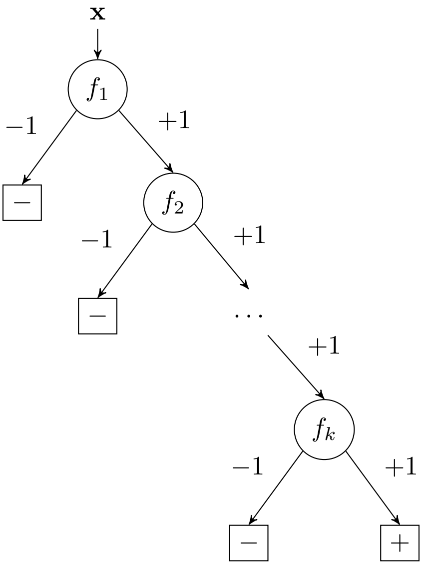
- Use many classifiers linked in a chain structure ("cascade").
- Each classifier eliminates part of the negative class.
- With each step down the cascade, class sizes become more even.

Classifier Cascades

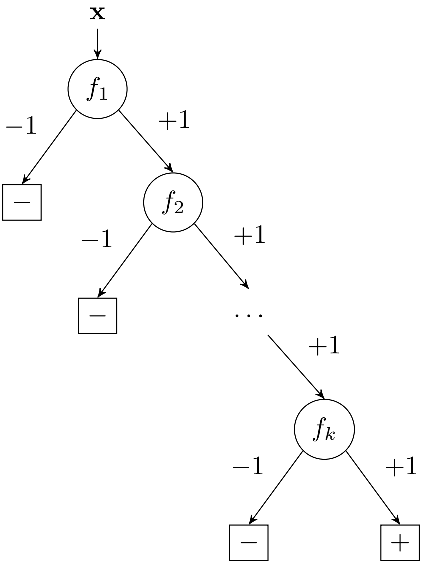
Training a cascade

Use imbalanced loss (very low false negative rate for each f_j).

1. Train classifier f_1 on entire training data set.
2. Remove all $\mathbf{x}_i$ in negative class which f_1 classifies correctly from training set.
3. On smaller training set, train f_2.
4. ...
5. On remaining data at final stage, train f_k.



Classifier Cascades



Classifying with a cascade

- If any f_j classifies $\mathbf{x}$ as negative, $f(\mathbf{x}) = -1$.
- Only if all f_j classify $\mathbf{x}$ as positive, $f(\mathbf{x}) = +1$.

Why does a cascade work?

We have to consider two rates

false positive rate $\quad \text{FPR}(f_j) = \dfrac{\#\text{negative points classified as "+1"}}{\#\text{negative training points at stage } j}$

detection rate $\quad \text{DR}(f_j) = \dfrac{\#\text{correctly classified positive points}}{\#\text{positive training points at stage } j}$

We want to achieve a low value of $\text{FPR}(f)$ and a high value of $\text{DR}(f)$.

Class imbalance

In face detection example:

- Number of faces classified as background is (size of face class) $\times$ $(1 - \text{DR}(f))$
- We would like to see a decently high detection rate, say 90%
- Number of background patches classified as faces is (size of background class) $\times$ $(\text{FPR}(f))$
- Since background class is huge, $\text{FPR}(f)$ has to be *very* small to yield roughly the same amount of errors in both classes.

Why does a cascade work?

Cascade detection rate

The rates of the overall cascade classifier f are

$$\mathsf{FPR}(f) = \prod_{j=1}^{k} \mathsf{FPR}(f_j) \qquad \mathsf{DR}(f) = \prod_{j=1}^{k} \mathsf{DR}(f_j)$$

- Suppose we use a 10-stage cascade ($k = 10$)
- Each DR(f_j) is 99% and we permit FPR(f_j) of 30%.
- We obtain DR$(f) = 0.99^{10} \approx 0.90$ and FPR$(f) = 0.3^{10} \approx 6 \times 10^{-6}$

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.

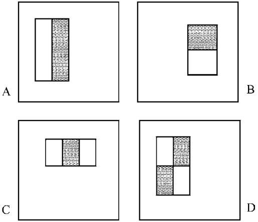
Feature Extraction

Extraction method

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

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

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

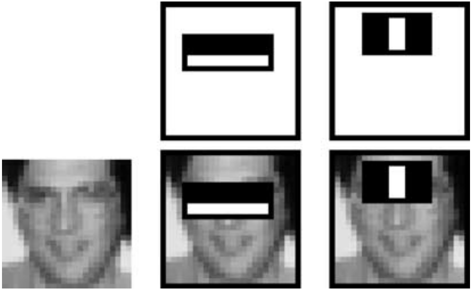


The dimension is huge

- 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

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

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

- 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,

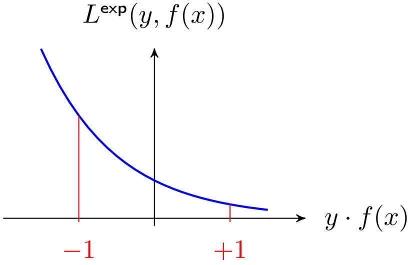
$$(\alpha_m, g_m) := \mathsf{arg} \min_{\alpha'_m, g'_m} \hat{R}_n(F^{(\mathsf{m-1})} + \alpha'_m g'_m)$$

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

Exponential Loss

Definition

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



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},

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

Relation to AdaBoost

It can be shown that the classifier obtained by solving

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

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

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

at each step m, we have to choose:

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

This is precisely equivalent to what AdaBoost does.

AdaBoost as Additive Model

AdaBoost approximates the 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 effectively optimizes *exponential* loss risk.