ENSEMBLE LEARNING

David Kauchak
CS158 – Fall 2019

Admin

Class last Tuesday?

Assignment grading

Assignment 9

Midterm 2

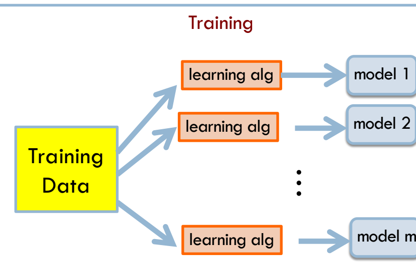
Final project
- 11/27 (Wed) submit project proposal

Ensemble learning

Basic idea: if one classifier works well, why not use multiple classifiers!

Ensemble learning

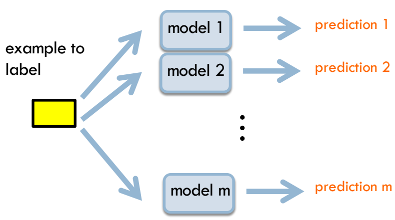
Basic idea: if one classifier works well, why not use multiple classifiers!



Ensemble learning

Basic idea: if one classifier works well, why not use multiple classifiers!

Testing



How do we decide on the final prediction?

Ensemble learning

Basic idea: if one classifier works well, why not use multiple classifiers!

Testing

prediction 1

prediction 2

⋮

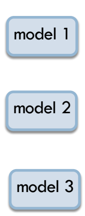
prediction m

- take majority vote
- if they output probabilities, take a weighted vote

How does having multiple classifiers help?

Benefits of ensemble learning

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)



Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e. error rate) with three classifiers for a binary classification problem?

Benefits of ensemble learning

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

model 1	model 2	model 3	prob
C	C	C	.6*.6*.6=0.216
C	C	I	.6*.6*.4=0.144
C	I	C	.6*.4*.6=0.144
C	I	I	.6*.4*.4=0.096
I	C	C	.4*.6*.6=0.144
I	C	I	.4*.6*.4=0.096
I	I	C	.4*.4*.6=0.096
I	I	I	.4*.4*.4=0.064

Benefits of ensemble learning

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

model 1	model 2	model 3	prob
C	C	C	.6*.6*.6=0.216
C	C	I	.6*.6*.4=0.144
C	I	C	.6*.4*.6=0.144
C	I	I	.6*.4*.4=0.096
I	C	C	.4*.6*.6=0.144
I	C	I	.4*.6*.4=0.096
I	I	C	.4*.4*.6=0.096
I	I	I	.4*.4*.4=0.064

0.096+
0.096+
0.096+
0.064 =
35% error!

Benefits of ensemble learning

3 classifiers in general, for r = probability of mistake for individual classifier:

$$p(error) = 3r^2(1-r) + r^3$$

binomial distribution

r	p(error)
0.4	0.35
0.3	0.22
0.2	0.10
0.1	0.028
0.05	0.0073

Benefits of ensemble learning

5 classifiers in general, for r = probability of mistake for individual classifier:

$$p(error) = 10r^3(1-r)^2 + 5r^4(1-r) + r^5$$

r	p(error) 3 classifiers	p(error) 5 classifiers
0.4	0.35	0.32
0.3	0.22	0.16
0.2	0.10	0.06
0.1	0.028	0.0086
0.05	0.0073	0.0012

Benefits of ensemble learning

m classifiers in general, for r = probability of mistake for individual classifier:

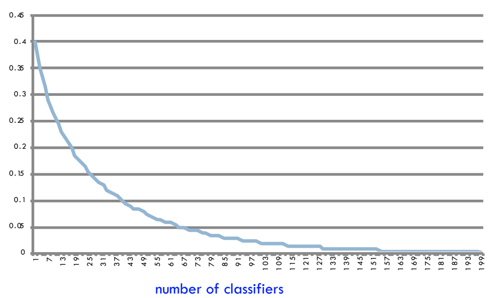
$$p(error) = \sum_{i=(m+1)/2}^{m} \binom{m}{i} r^i (1-r)^{m-i}$$

(cumulative probability distribution for the binomial distribution)

Given enough classifiers…

$$p(error) = \sum_{i=(m+1)/2}^{m} \binom{m}{i} r^i (1-r)^{m-i}$$

r = 0.4



What is the catch?

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

model 1

model 2

model 3

Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e. error rate) with three classifiers for a binary classification problem?

What is the catch?

Assume each classifier makes a mistake with some probability (e.g. 0.4, that is a 40% error rate)

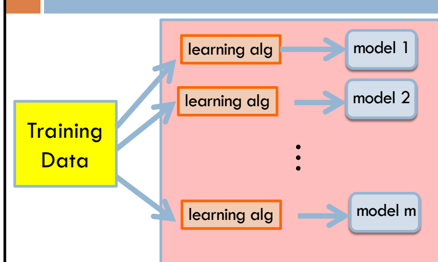
model 1

model 2

model 3

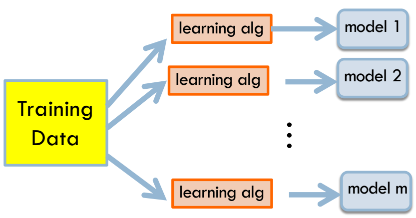
Assuming the decisions made between classifiers are independent, what will be the probability that we make a mistake (i.e. error rate) with three classifiers for a binary classification problem?

Obtaining independent classifiers



Where do we get *m* independent classifiers?

Idea 1: different learning methods



decision tree

k-nn

perceptron

naïve bayes

gradient descent variant 1

gradient descent variant 2

⋮

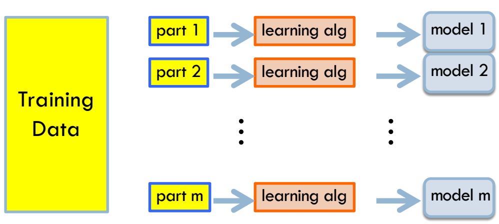
Pros/cons?

Idea 1: different learning methods

Pros:
- Lots of existing classifiers already
- Can work well for some problems

Cons/concerns:
- Often, classifiers are not independent, that is, **they make the same mistakes!**
  - e.g. many of these classifiers are linear models
  - voting won't help us if they're making the same mistakes

Idea 2: split up training data

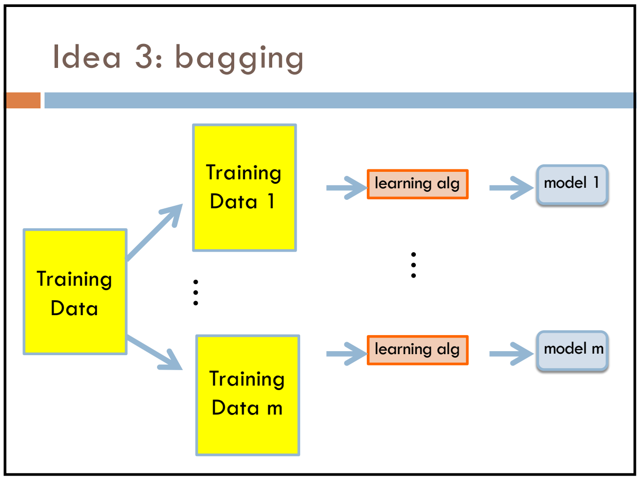


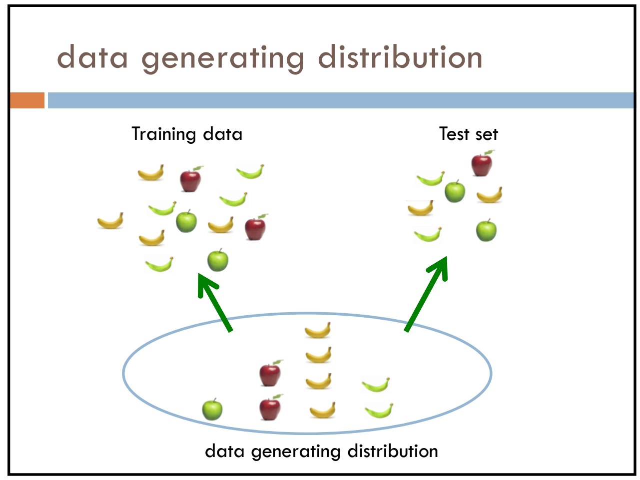
Use the same learning algorithm, but train on different parts of the training data

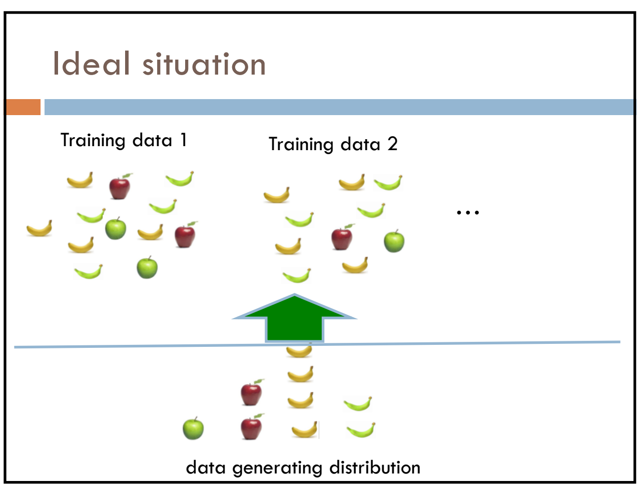
Idea 2: split up training data

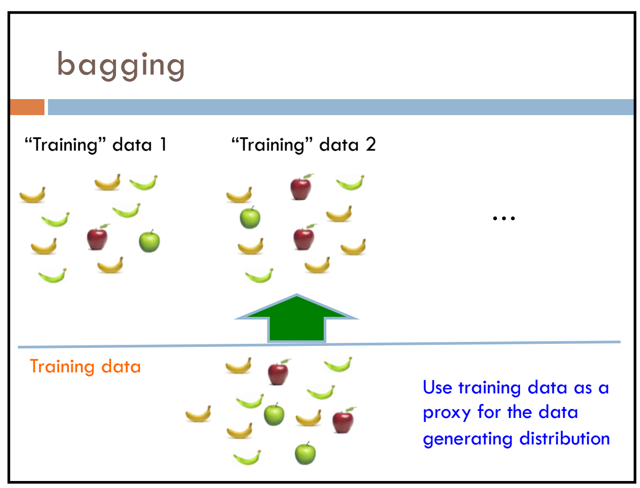
Pros:
- Learning from different data, so can't overfit to same examples
- Easy to implement
- fast

Cons/concerns:
- Each classifier is only training on a small amount of data
- Not clear why this would do any better than training on full data and using good regularization


Idea 3: bagging
Training Data
Training Data 1
Training Data m
learning alg
learning alg
model 1
model m


data generating distribution
Training data
Test set
data generating distribution

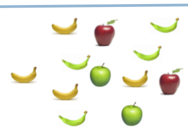

Ideal situation
Training data 1
Training data 2
…
data generating distribution


bagging
"Training" data 1
"Training" data 2
…
Training data
Use training data as a proxy for the data generating distribution

sampling with replacements

"Training" data 1

Training data

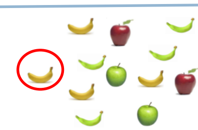


sampling with replacements

"Training" data 1

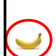
pick a random example from the real training data

Training data



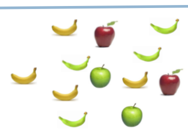
sampling with replacements

"Training" data 1



add it to the new "training" data

Training data

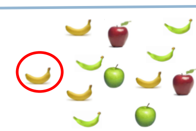


sampling with replacements

"Training" data 1

put it back (i.e. leave it) in the original training data

Training data

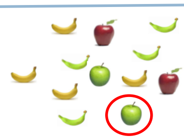


sampling with replacements

"Training" data 1

pick another random example

Training data

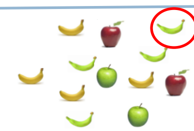


sampling with replacements

"Training" data 1

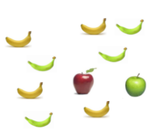
pick another random example

Training data



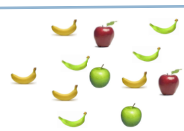
sampling with replacements

"Training" data 1



keep going until you've created a new "training" data set

Training data



bagging

create m "new" training data sets by sampling with replacement from the original training data set (called *m* "bootstrap" samples)

train a classifier on each of these data sets

to classify, take the majority vote from the *m* classifiers

bagging concerns

Training Data

Training Data 1

⋮

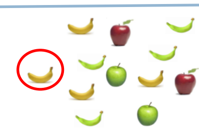
Training Data m

Won't these all be basically the same?

bagging concerns

For a data set of size n, what is the probability that a given example will **NOT** be select in a "new" training set sampled from the original?

Training data

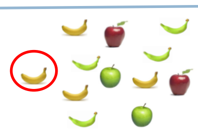


bagging concerns

What is the probability it isn't chosen the first time?

$$1 - 1/n$$

Training data



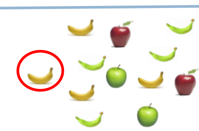
bagging concerns

What is the probability it isn't chosen the ***any*** of the n times?

$$(1 - 1/n)^n$$

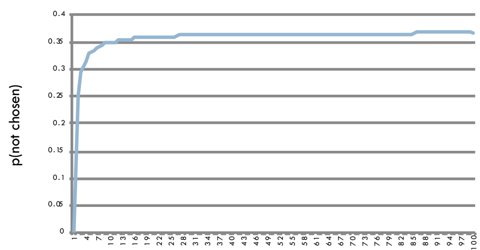
Each draw is independent and has the same probability

Training data



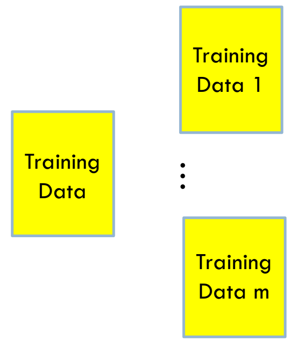
probability of overlap

$$(1-1/n)^n$$



Converges very quickly to 1/e ≈ 63%

bagging overlap



Won't these all be basically the same?

On average, a randomly sampled data set will only contain 63% of the examples in the original

When does bagging work

Let's say 10% of our examples are noisy (i.e. don't provide good information)

For each of the "new" data set, what proportion of noisy examples will they have?

- They'll still have ~10% of the examples as noisy
- However, these examples will only represent about two-thirds of the original noisy examples

For some classifiers that have trouble with noisy classifiers, this can help

When does bagging work

Bagging tends to reduce the *variance* of the classifier

By voting, the classifiers are more robust to noisy examples

Bagging is most useful for classifiers that are:

- Unstable: small changes in the training set produce very different models
- Prone to overfitting

Often has similar effect to regularization

Idea 4: boosting

training data

Data	Label	Weight
	0	0.2
	0	0.2
	1	0.2
	1	0.2
	0	0.2

"training" data 2

Data	Label	Weight
	0	0.1
	0	0.1
	1	0.4
	1	0.1
	0	0.3

"training" data 3

Data	Label	Weight
	0	0.05
	0	0.2
	1	0.2
	1	0.05
	0	0.5

"Strong" learner

Given

- a reasonable amount of training data
- a target error rate ε
- a failure probability *p*

A **strong learning algorithm** will produce a classifier with error rate <ε with probability 1-p

"Weak" learner

Given

- a reasonable amount of training data
- a failure probability *p*

A **weak learning algorithm** will produce a classifier with error rate < 0.5 with probability 1-p

Weak learners are much easier to create!

weak learners for boosting

Data	Label	Weight
	0	0.2
	0	0.2
	1	0.2
	1	0.2
	0	0.2

weak learning algorithm → weak classifier

Which of our algorithms can handle weights?

Need a weak learning algorithm that can handle **weighted** examples

boosting: basic algorithm

Training:

start with equal example weights

for some number of iterations:

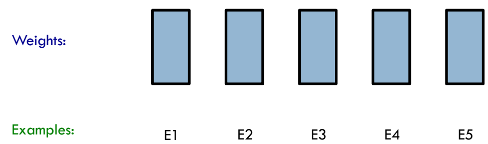
- learn a weak classifier and save
- change the example weights

Classify:

- get prediction from all learned weak classifiers
- weighted vote based on how well the weak classifier did when it was trained (i.e. in relation to training error)

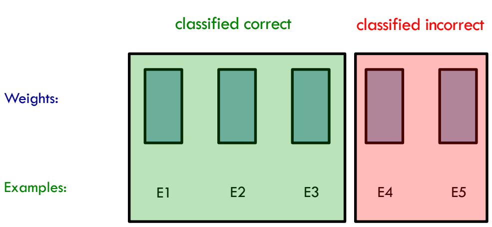
boosting basics

Start with equal weighted examples



Learn a weak classifier:

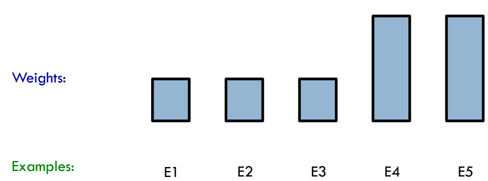
Boosting



We want to reweight the examples and then learn another weak classifier

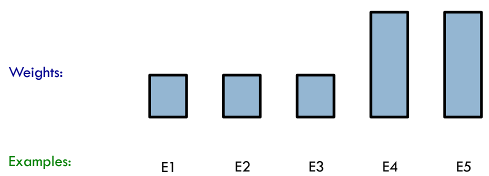
How should we change the example weights?

Boosting



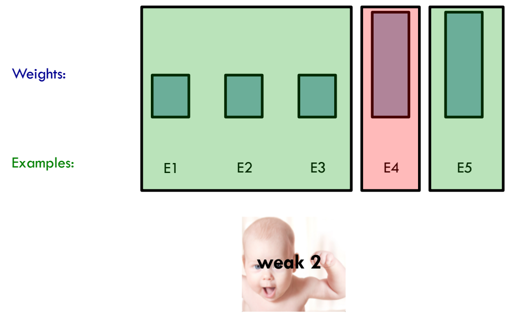
- decrease the weight for those we're getting correct
- increase the weight for those we're getting incorrect

Boosting

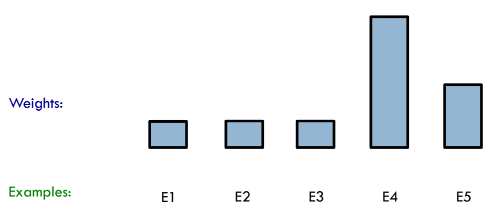


Learn another weak classifier:

Boosting

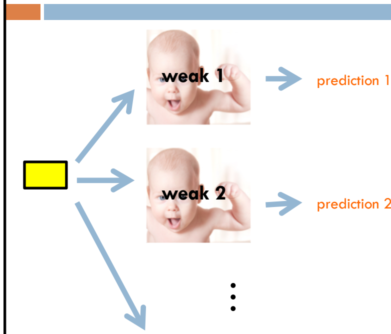


Boosting



- decrease the weight for those we're getting correct
- increase the weight for those we're getting incorrect

Classifying



weighted vote based on how well they classify the training data

weak_2_vote > weak_1_vote since it got more right

Notation

x_i example i in the training data

w_i weight for example *i*, we will enforce:

$$w_i \geq 0$$

$$\sum_{i=1}^{n} w_i = 1$$

classifier$_k(x_i)$ +1/-1 prediction of classifier *k* example *i*

AdaBoost: train

for k = 1 to *iterations*:

- classifier$_k$ = learn a weak classifier based on weights
- calculate weighted error for this classifier

$$\varepsilon_k = \sum_{i=1}^{n} w_i * 1[label_i \neq classifier_k(x_i)]$$

- calculate "score" for this classifier:

$$\alpha_k = \frac{1}{2}\log\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- change the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

AdaBoost: train

classifier$_k$ = learn a weak classifier based on weights

weighted error for this classifier is:

$$\varepsilon_k = \sum_{i=1}^{n} w_i * 1[label_i \neq classifier_k(x_i)]$$

What does this say?

AdaBoost: train

classifier$_k$ = learn a weak classifier based on weights

weighted error for this classifier is:

$$\varepsilon_k = \sum_{i=1}^{n} \underbrace{w_i * \underbrace{1[label_i \neq \underbrace{classifier_k(x_i)}_{\text{prediction}}]}_{\text{did we get the example wrong}}}_{\text{weighted sum of the errors/mistakes}}$$

What is the range
of possible values?

AdaBoost: train

classifier$_k$ = learn a weak classifier based on weights

weighted error for this classifier is:

$$\varepsilon_k = \underbrace{\sum_{i=1}^{n} w_i * \underbrace{1[label_i \neq \underbrace{classifier_k(x_i)}_{\text{prediction}}]}_{\text{did we get the example wrong}}}_{\text{weighted sum of the errors/mistakes}}$$

Between 0 (if we get all examples right) and 1 (if we get them all wrong)

AdaBoost: train

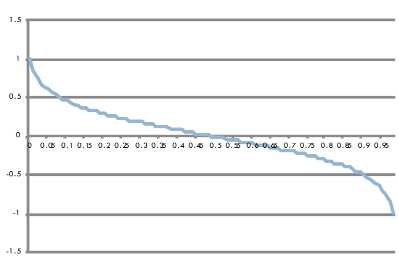
classifier$_k$ = learn a weak classifier based on weights

“score” or weight for this classifier is:

$$\alpha_k = \frac{1}{2}\log\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

What does this look like (specifically for errors between 0 and 1)?

AdaBoost: train



$$\alpha_k = \frac{1}{2}\log\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- ranges from +∞ to -∞
- for most reasonable values: ranges from ~1 to -1
- errors of 50% = 0

AdaBoost: classify

$$classify(x) = sign\left(\sum_{k=1}^{iterations} \alpha_k * classifier_k(x)\right)$$

What does this do?

AdaBoost: classify

$$classify(x) = sign\left(\sum_{k=1}^{iterations} \alpha_k * classifier_k(x)\right)$$

The weighted vote of the learned classifiers weighted by α (remember α generally varies from ~1 to -1 training error)

What happens if a classifier has error >50%

AdaBoost: classify

$$classify(x) = sign\left(\sum_{k=1}^{iterations} \alpha_k * classifier_k(x)\right)$$

The weighted vote of the learned classifiers weighted by α (remember α generally varies from ~1 to -1 training error)

We vote the opposite!

AdaBoost: train, updating the weights

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

Remember, we want to enforce:

$$w_i \geq 0$$

$$\sum_{i=1}^{n} w_i = 1$$

Z is called the normalizing constant. It is used to make sure that the weights sum to 1

What should it be?

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

Remember, we want to enforce:

$$w_i \geq 0$$

$$\sum_{i=1}^{n} w_i = 1$$

normalizing constant (i.e. the sum of the "new" w_i):

$$\boxed{Z = \sum_{i=1}^{n} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)}$$

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k \boxed{* \, label_i * classifier_k(x_i)}\right)$$

What does this do?

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

correct → positive
incorrect → negative

correct ?
incorrect

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

correct → positive
incorrect → negative

correct → small value
incorrect → large value

Note: only change weights based on current classifier (not all previous classifiers)

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * label_i * classifier_k(x_i)\right)$$

What does the α do?

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp(-\alpha_k * label_i * classifier_k(x_i))$$

What does the α do?

If the classifier was good (<50% error) α is positive:
- trust classifier output and move as normal

If the classifier was bad (>50% error) α is negative
- classifier is so bad, consider opposite prediction of classifier

AdaBoost justification

update the example weights

$$w_i = \frac{1}{Z} w_i \underbrace{\exp(-\alpha_k * label_i * classifier_k(x_i))}_{\text{Does this look like anything we've seen before?}}$$

AdaBoost justification

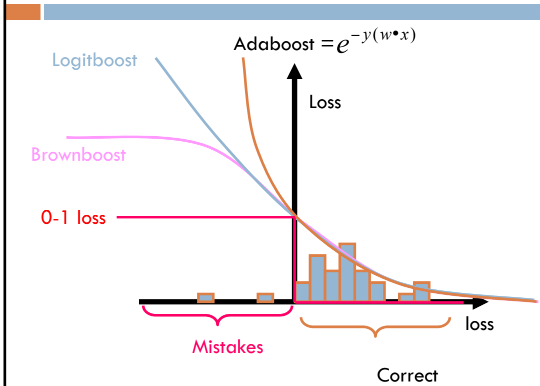
update the example weights

$$w_i = \frac{1}{Z} w_i \underbrace{\exp(-\alpha_k * label_i * classifier_k(x_i))}_{\text{Exponential loss!}}$$

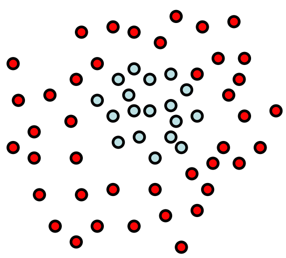
$$l(y, y') = \exp(-yy')$$

AdaBoost turns out to be another approach for minimizing the exponential loss!

Other boosting variants



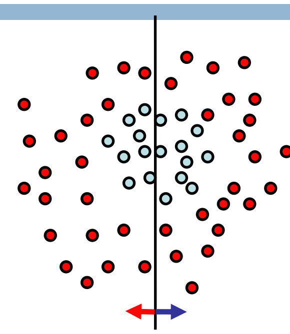
Boosting example



Start with equal weighted data set

Boosting example

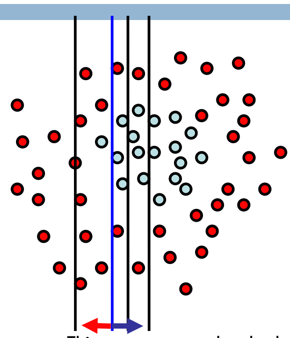
weak learner = line



What would be the best line learned on this data set?

h => p(error) = 0.5 it is at chance

Boosting example

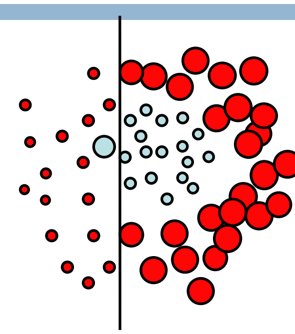


How should we reweight examples?

This one seems to be the best

This is a '**weak classifier**': It performs slightly better than chance.

Boosting example

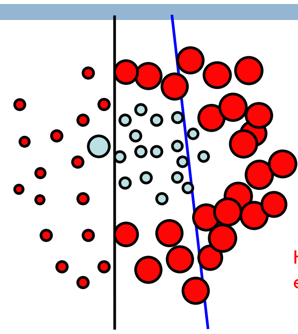


What would be the best line learned on this data set?

reds on this side get less weight
blues on this side get more weight

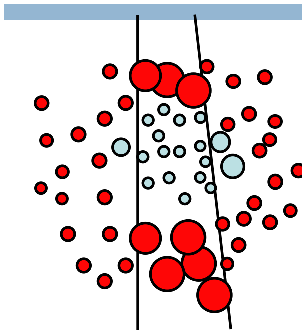
reds on this side get more weight
blues on this side get less weight

Boosting example



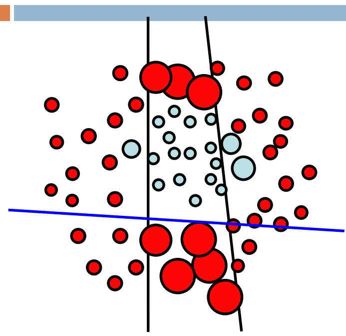
How should we reweight examples?

Boosting example

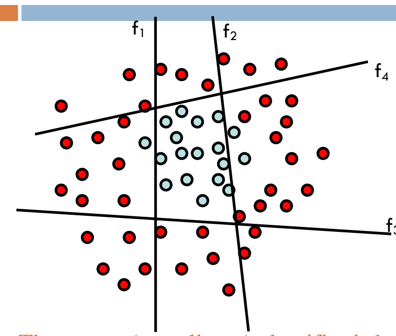


What would be the best line learned on this data set?

Boosting example



Boosting example



The strong (non- linear) classifier is built as the combination of all the weak (linear) classifiers.

AdaBoost: train

for k = 1 to *iterations*:

- classifier_k = learn a weak classifier based on weights
- weighted error for this classifier is:
- "score" or weight for this classifier is:
- change the example weights

What can we use as a classifier?

AdaBoost: train

for k = 1 to *iterations*:

- classifier_k = learn a weak classifier based on weights
- weighted error for this classifier is:
- "score" or weight for this classifier is:
- change the example weights

- Anything that can train on weighted examples
- For most applications, must be fast!

  Why?

AdaBoost: train

for k = 1 to *iterations*:

- classifier_k = learn a weak classifier based on weights
- weighted error for this classifier is:
- "score" or weight for this classifier is:
- change the example weights

- Anything that can train on weighted examples
- For most applications, must be fast!
  - Each iteration we have to train a new classifier

Boosted decision stumps

One of the most common classifiers to use is a decision tree:

- can use a shallow (2-3 level tree)
- even more common is a 1-level tree
  - called a decision stump ☺
  - asks a question about a single feature

What does the decision boundary look like for a decision stump?

Boosted decision stumps

One of the most common classifiers to use is a decision tree:

- can use a shallow (2-3 level tree)
- even more common is a 1-level tree
  - called a decision stump ☺
  - asks a question about a single feature

What does the decision boundary look like for boosted decision stumps?

Boosted decision stumps

One of the most common classifiers to use is a decision tree:

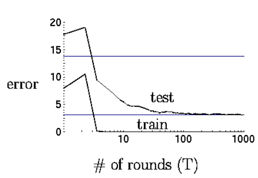
- can use a shallow (2-3 level tree)
- even more common is a 1-level tree
  - called a decision stump ☺
  - asks a question about a single feature

- Linear classifier!
- Each stump defines the weight for that dimension
  - If you learn multiple stumps for that dimension then it's the weighted average

Boosting in practice

Very successful on a wide range of problems

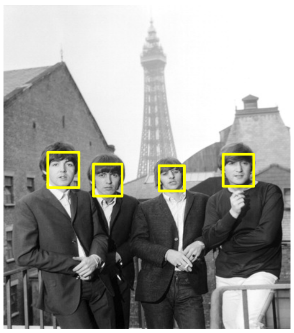
One of the keys is that boosting tends not to overfit, even for a large number of iterations



Using <10,000 training examples can fit >2,000,000 parameters!

Adaboost application example: face detection

Adaboost application example: face detection



Rapid Object Detection using a Boosted Cascade of Simple Features

Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

Rapid object **detection** using a boosted cascade of simple features
P **Viola**, M **Jones** - … Vision and Pattern Recognition, 2001. CVPR …, 2001 - ieeexplore.ieee.org
… overlap. Each partition yields a single final **detection**. The **…** set. Experiments on a Real-World Test Set We tested our system on the MIT+CMU frontal **face** test set [II]. This set consists of 130 images with 507 labeled frontal **faces**. A **…**
Cited by 8422 Related articles All 129 versions Cite Save More

[PDF] **Rapid object detection using** a **boosted** cascade of simple features
P Viola, M Jones - CVPR (1), 2001 - researchgate.net
This paper describes a machine learning approach for visual **object detection** which is capable of processing images extremely **rapidly** and achieving high **detection** rates. This work is distinguished by three key contributions. The first is the introduction of a new image …
Cited by 19976 Related articles All 101 versions

To give you some context of importance:

Google

The anatomy of a **large-scale** hypertextual web search engine
S Brin, L Page - Computer networks and ISDN systems, 1998 - Elsevier
In this paper, we present Google, a prototype of a **large-scale** search engine which makes heavy use of **the** structure present in hypertext. Google is designed to crawl and index **the** Web efficiently and produce much more satisfying search results than existing systems. **The** …
Cited by 18931 Related articles All 220 versions Web of Science: 4067

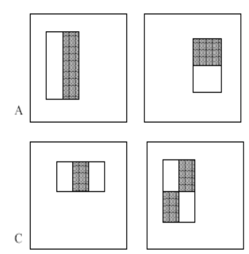
or:

Modeling word burstiness using the Dirichlet distribution
RE Madsen, D Kauchak, C Elkan - Proceedings of the 22nd international …, 2005 - dl.acm.org
Multinomial distributions are often used to model text documents. However, they do not capture well the phenomenon that words in a document tend to appear in bursts: if a word appears once, it is more likely to appear again. In this paper, we propose the Dirichlet compound multinomial model (DCM) as an alternative to the multinomial. The DCM model has one additional degree of freedom, which allows it to capture burstiness. We show experimentally that the DCM is substantially better than the multinomial at modeling text …
Cited by 289 Related articles All 27 versions

"weak" learners

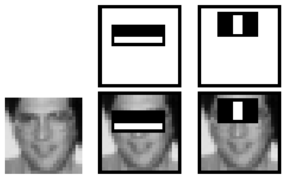
4 Types of "Rectangle filters" (Similar to Haar wavelets Papageorgiou, et al.)

Based on 24x24 grid:
160,000 features to choose from



g(x) = sum(WhiteArea) - sum(BlackArea)

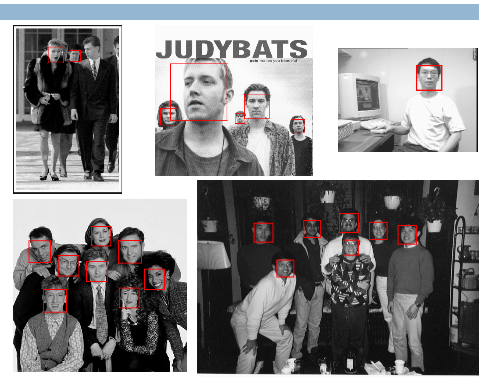
"weak" learners



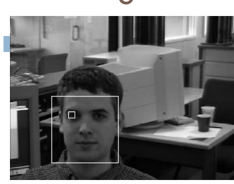
$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \ldots$$

$$f_i(x) = \begin{cases} 1 & \text{if } g_i(x) > \theta_i \\ -1 & \text{otherwise} \end{cases}$$

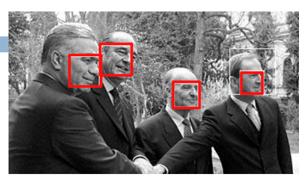
Example output



Solving other "Face" Tasks

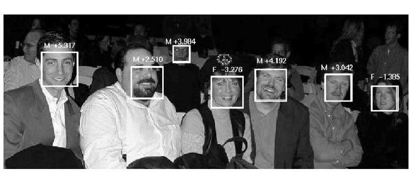


Facial Feature Localization

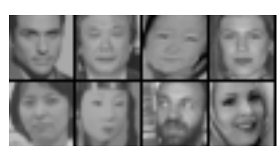


Profile Detection

Demographic Analysis



"weak" classifiers learned



Bagging vs Boosting

Journal of Artificial Intelligence Research 11 (1999) 169-198 Submitted 1/99; published 8/99

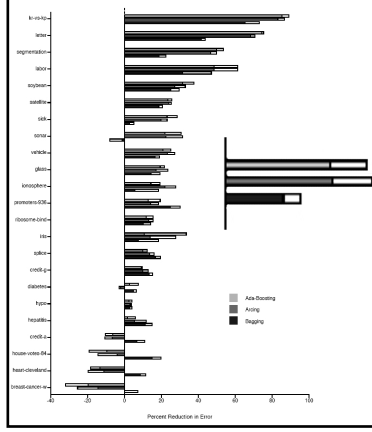
Popular Ensemble Methods: An Empirical Study

David Opitz OPITZ@CS.UMT.EDU
Department of Computer Science
University of Montana
Missoula, MT 59812 USA

Richard Maclin RMACLIN@D.UMN.EDU
Computer Science Department
University of Minnesota
Duluth, MN 55812 USA

http://arxiv.org/pdf/1106.0257.pdf

Boosting Neural Networks



Change in error rate over standard classifier

Ada-Boosting
Arcing
Bagging

White bar represents 1 standard deviation

Boosting Decision Trees