CSE419 – Artificial Intelligence and Machine Learning 2021

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/CSE419_2021

Lecture 14 Part 1

Logistic Regression

Based on Asst. Prof. Dr. David Kauchak (Pomona College) Lecture Slides

Good Regression Videos

- An Introduction to Linear Regression Analysis > https://youtu.be/zPG4NjIkCjc
- How to calculate linear regression using least square method > https://youtu.be/JvS2triCgOY
- How to Calculate R Squared Using Regression Analysis > https://youtu.be/w2FKXOa0HGA
- Standard Error of the Estimate used in Regression Analysis (Mean Square Error) > https://youtu.be/r-txC-dpI-E

Good Regression Videos

- Statistics 101: Logistic Regression, An Introduction > https://youtu.be/zAULhNrnuL4
- Statistics 101: Logistic Regression Probability, Odds, and Odds Ratio > https://youtu.be/ckkiG-SDuV8
- Linear Regression Playlist > https://www.youtube.com/playlist?list=PLIeGtxpvyG-LoKUpV0fSY8BGKIMIdmfCi
- Logistic Regression Playlist > https://www.youtube.com/playlist?list=PLIeGtxpvyG-JmBQ9XoFD4rs-b3hkcX7Uu

Good Regression Videos

- StatQuest: Linear Models Pt.1 - Linear Regression > https://youtu.be/nk2CQITm_eo
- StatQuest: Linear Models Pt.1.5 - Multiple Regression > https://youtu.be/zITIFTsivN8
- StatQuest: Logistic Regression > https://youtu.be/yIYKR4sgzI8

Training revisited

From a probability standpoint, what we're really doing when we're training the model is selecting the Θ that maximizes:

$$p(q \mid data)$$

i.e.

$$\text{argmax}_q \, p(q \mid data)$$

That we pick the most likely model parameters given the data

Estimating revisited

We can incorporate a prior belief in what the probabilities might be

To do this, we need to break down our probability

$$p(q \mid data) = ?$$

(Hint: Bayes rule)

Estimating revisited

What are each of these probabilities?

$$p(q \mid data) = \frac{p(data \mid q)\,p(q)}{p(data)}$$

Priors

likelihood of the data
under the model

probability of different parameters,
call the prior

$$p(q \mid data) = \frac{p(data \mid q)\, p(q)}{p(data)}$$

probability of seeing the data
(regardless of model)

Priors

$$q = \operatorname{argmax}_q \frac{p(data \mid q)\,p(q)}{\underset{\uparrow}{p(data)}}$$

Does p(data) matter for the argmax?

Priors

likelihood of the data under the model

probability of different parameters, call the prior

$$q = \operatorname{argmax}_q p(data \mid q)p(q)$$

What does MLE assume for a prior on the model parameters?

Priors

likelihood of the data
under the model

probability of different parameters,
call the prior

$$q = \operatorname{argmax}_q p(data \mid q) p(q)$$

- Assumes a uniform prior, i.e. all Θ are equally likely!
- Relies solely on the likelihood

A better approach

$$q = \text{argmax}_q \, p(data \mid q) p(q)$$

$$likelihood(data) = \overset{n}{\underset{i=1}{\tilde{O}}} p_q(x_i)$$

We can use any distribution we'd like
This allows us to impart addition bias into the model

Another view on the prior

Remember, the max is the same if we take the log:

$$\theta = \text{argmax}_{\theta} \log(p(data \mid \theta)) + \log(p(\theta))$$

$$\log\text{-}\ likelihood = \sum_{i=1}^{n} \log(p(x_i))$$

We can use any distribution we'd like
This allows us to impart addition bias into the model

Does this look like something we've seen before?

Regularization vs prior

$$q = \text{argmax}_q \log(p(data \mid q)) + \log(p(q))$$

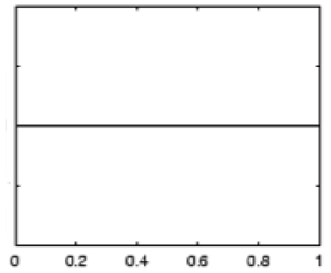
$$\text{fit}\left\{\begin{array}{ll}\text{likelihood based on the data} & \text{prior}\\ \text{loss function based on the data} & \text{regularizer}\end{array}\right\}\text{model bias}$$

$$\text{argmin}_{w,b} \overset{n}{\underset{i=1}{\text{å}}} loss(yy') + l\, regularizer(w)$$

Prior for NB

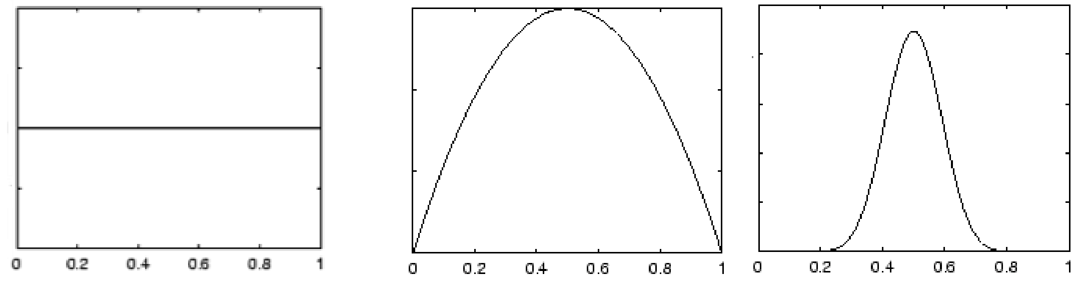
$$q = \operatorname{argmax}_q \log(p(data \mid q)) + \log(p(q))$$

Uniform prior



$$p(x_i \mid y) = \frac{count(x_i, y)}{count(y)}$$

Dirichlet prior



λ= 0 increasing

$$p(x_i \mid y) = \frac{count(x_i, y) + l}{count(y) + possible\_values\_of\_x_i * l}$$

Prior: another view

$$p(x_1, x_2, \ldots, x_m, y) = p(y) \prod_{j=1}^{m} p(x_i \mid y)$$

MLE: $p(x_i \mid y) = \dfrac{count(x_i, y)}{count(y)}$

What happens to our likelihood if, for one of the labels, we never saw a particular feature?

Goes to 0!

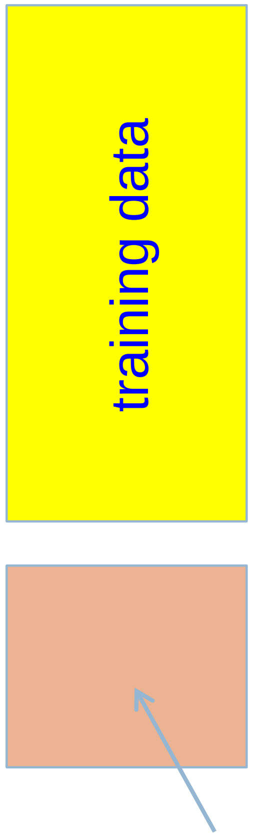
Prior: another view

$$p(x_i \mid y) = \frac{count(x_i, y)}{count(y)}$$

$$p(x_i \mid y) = \frac{count(x_i, y) + /}{count(y) + possible\_values\_of\_x_i * /}$$

Adding a prior avoids this!

Smoothing



$$p(x_i \mid y) = \frac{count(x_i, y)}{count(y)}$$

$$\Downarrow$$

$$p(x_i \mid y) = \frac{count(x_i, y) + \lambda}{count(y) + possible\_values\_of\_x_i * \lambda}$$

for each label, pretend like we've seen each feature value occur inλadditional examples

Sometimes this is also called smoothing because it is seen as smoothing or interpolating between the MLE and some other distribution

Basic steps for probabilistic modeling

Step 1: pick a model

Step 2: figure out how to estimate the probabilities for the model

Step 3 (optional): deal with overfitting

Probabilistic models

Which model do we use, i.e. how do we calculate p(*feature, label*)?

How do train the model, i.e. how to we we estimate the probabilities for the model?

How do we deal with overfitting?

Joint models vs conditional models

We've been trying to model the joint distribution (i.e. the data generating distribution):

$$p(x_1, x_2, \ldots, x_m, y)$$

However, if all we're interested in is classification, why not directly model the conditional distribution:

$$p(y \mid x_1, x_2, \ldots, x_m)$$

A first try: linear

$$p(y \mid x_1, x_2, \ldots, x_m) = x_1 w_1 + w_2 x_2 + \ldots + w_m x_m + b$$

Any problems with this?

- Nothing constrains it to be a probability
- Could still have combination of features and weight that exceeds 1 or is below 0

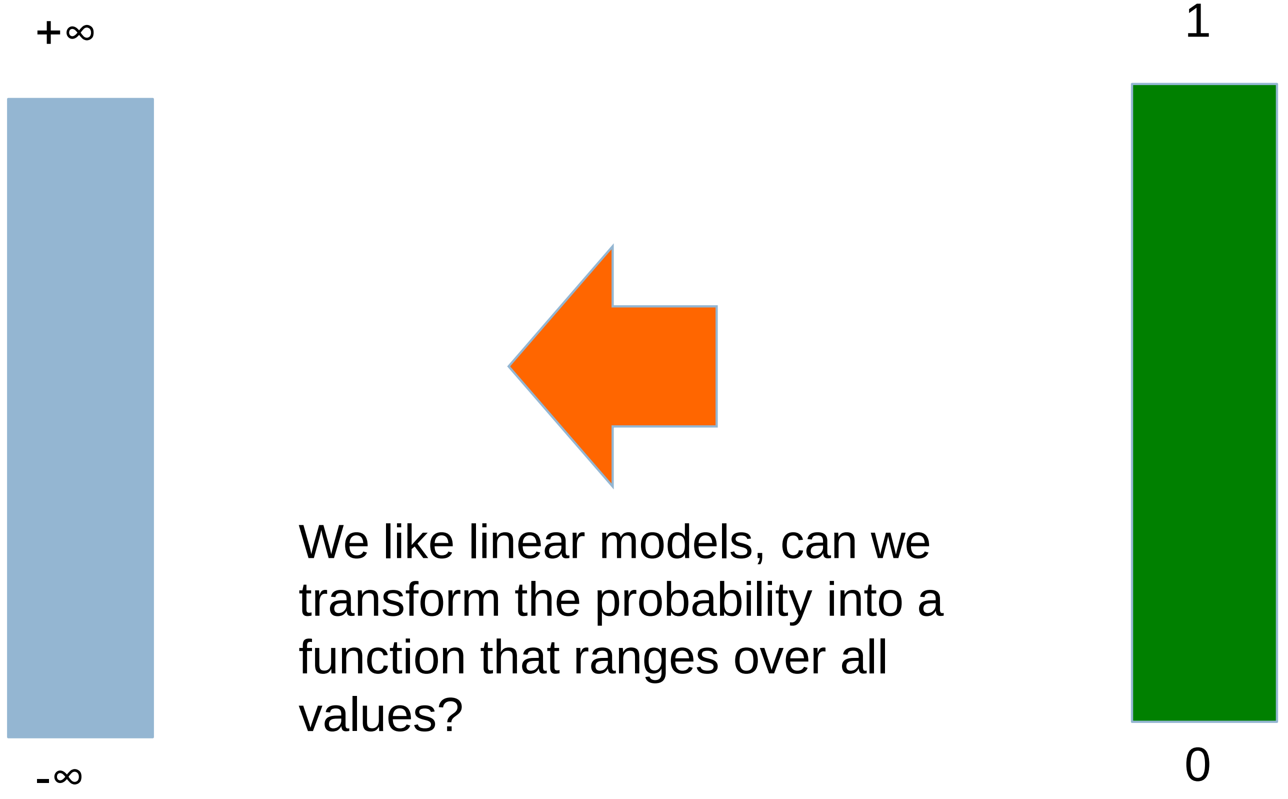
The challenge

$$x_1 w_1 + w_2 x_2 + \ldots + w_m x_m + b$$

$$\boxed{p(y \mid x_1, x_2, \ldots, x_m)}$$

Linear model

probability



We like linear models, can we transform the probability into a function that ranges over all values?

Odds ratio

Rather than predict the probability, we can predict the ratio of 1/0 (positive/negative)

Predict the **odds** that it is 1 (true): How much more likely is 1 than 0.

Does this help us?

$$\frac{P(1 \mid x_1, x_2, \ldots, x_m)}{P(0 \mid x_1, x_2, \ldots, x_m)} = \frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)} = x_1 w_1 + w_2 x_2 + \ldots + w_m x_m + b$$

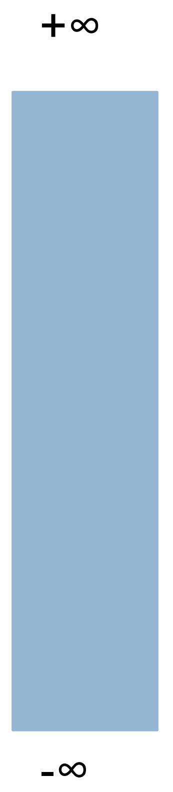
Odds ratio

$$x_1 w_1 + w_2 x_2 + \ldots + w_m x_m + b$$

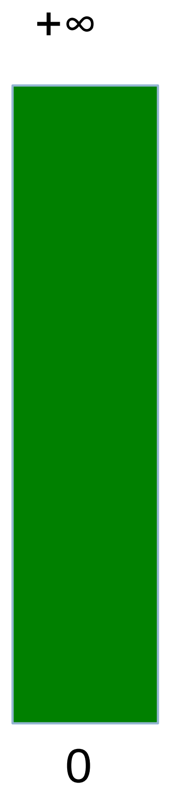
Linear model

$$\frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)}$$

odds ratio



Where is the dividing line between class 1 and class 0 being selected?



Odds ratio

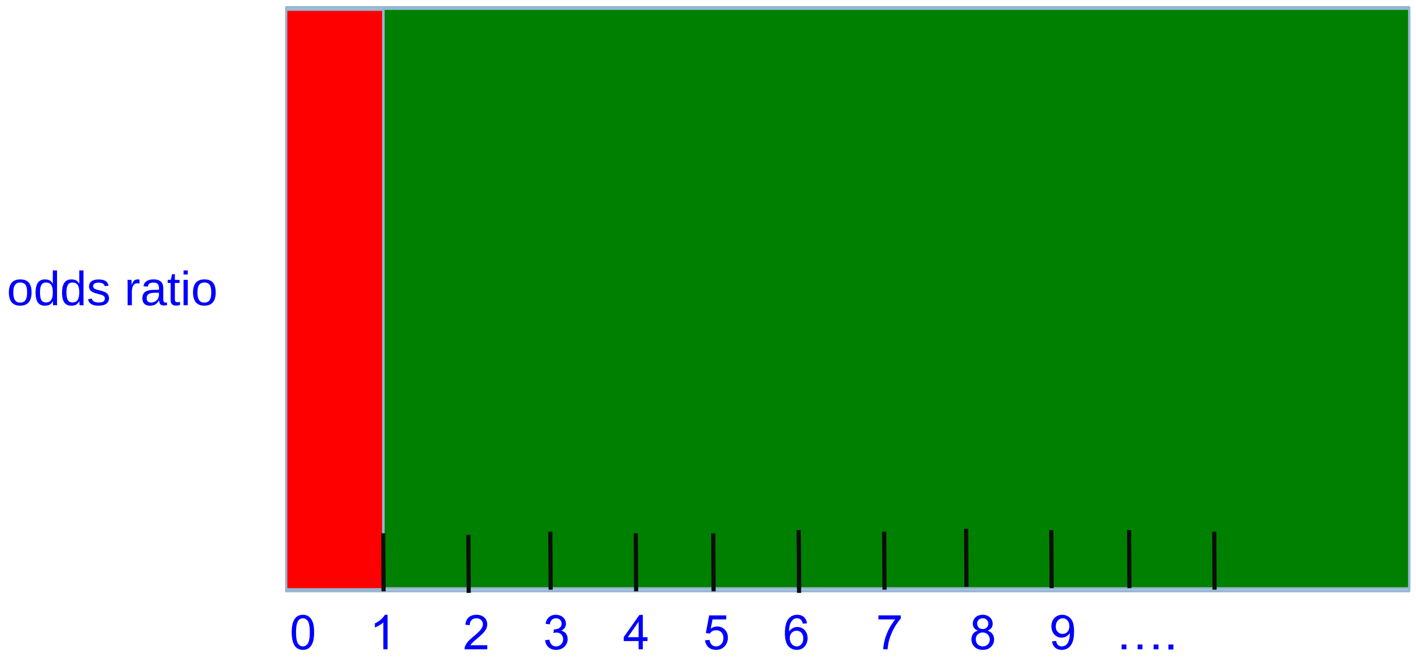
$$P(1|x_1,x_2,\ldots,x_m) > P(0|x_1,x_2,\ldots,x_m)$$

$$\frac{P(1|x_1,x_2,\ldots,x_m)}{1-P(1|x_1,x_2,\ldots,x_m)}$$

$$P(1|x_1,x_2,\ldots,x_m) > 1-P(1|x_1,x_2,\ldots,x_m)$$

We're trying to find some transformation that transforms the odds ratio to a number that is -∞ to +∞

Does this suggest another transformation?



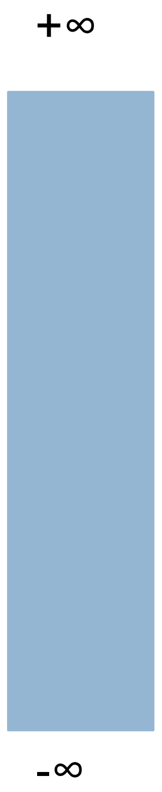
Log odds (logit function)

$$x_1 w_1 + w_2 x_2 + \ldots + w_m x_m + b$$

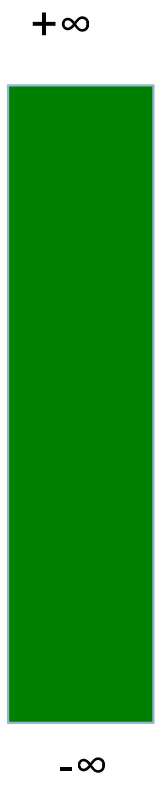
Linear regression

$$\log \frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)}$$

odds ratio



How do we get the probability of an example?



Log odds (logit function)

$$\log \frac{P(1|x_1, x_2, \ldots, x_m)}{1 - P(1|x_1, x_2, \ldots, x_m)} = w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b$$

$$\frac{P(1|x_1, x_2, \ldots, x_m)}{1 - P(1|x_1, x_2, \ldots, x_m)} = e^{w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b}$$

$$P(1|x_1, x_2, \ldots, x_m) = (1 - P(1|x_1, x_2, \ldots, x_m))e^{w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b}$$

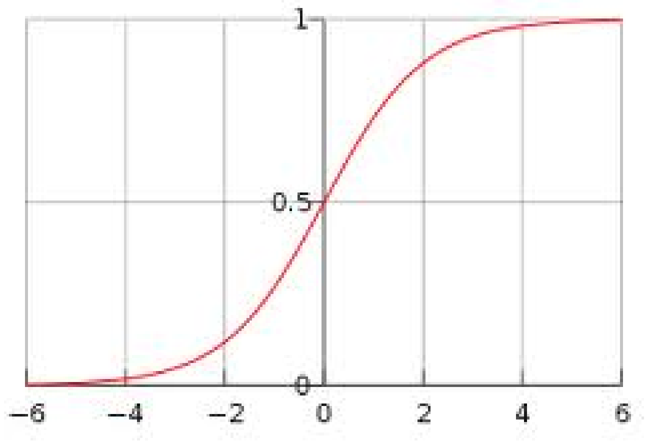
$$\ldots$$

$$P(1|x_1, x_2, \ldots, x_m) = \frac{1}{1 + e^{-(w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)}}$$

anyone recognize this?

Logistic function

$$\text{logistic} = \frac{1}{1 + e^{-x}}$$



Logistic regression

How would we classify examples once we had a trained model?

$$\log \frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)} = w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b$$

If the sum > 0 then p(1)/p(0) > 1, so positive

if the sum < 0 then p(1)/p(0) < 1, so negative

Still a *linear* classifier (decision boundary is a line)

Training logistic regression models

How should we learn the parameters for logistic regression (i.e. the w's)?

$$\log \frac{P(1 | x_1, x_2, \ldots, x_m)}{1 - P(1 | x_1, x_2, \ldots, x_m)} = w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b$$

parameters

$$P(1 | x_1, x_2, \ldots, x_m) = \frac{1}{1 + e^{-(w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)}}$$

MLE logistic regression

Find the parameters that maximize the likelihood (or log-likelihood) of the data:

$$\text{log-}\, likelihood = \sum_{i=1}^{n} \log(p(x_i))$$

$$= \sum_{i=1}^{n} \log\left(\frac{1}{1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)}} \right)$$

assume labels 1, -1

$$= \sum_{i=1}^{n} -\log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

MLE logistic regression

$$\text{log-}\,likelihood = \sum_{i=1}^{n} -\log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

We want to maximize, i.e.

$$MLE(data) = \operatorname{argmax}_{w,b} \text{log-}\,likelihood(data)$$

$$= \operatorname{argmax}_{w,b} \sum_{i=1}^{n} -\log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

$$= \operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

Look familiar? Hint: anybody read the book?

MLE logistic regression

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

Surrogate loss functions:

Zero/one:	$\ell^{(0/1)}(y,\hat{y}) = 1[y\hat{y} \le 0]$
Hinge:	$\ell^{(\text{hin})}(y,\hat{y}) = \max\{0, 1 - y\hat{y}\}$
Logistic:	$\ell^{(\log)}(y,\hat{y}) = \frac{1}{\log 2} \log(1 + \exp[-y\hat{y}])$
Exponential:	$\ell^{(\exp)}(y,\hat{y}) = \exp[-y\hat{y}]$
Squared:	$\ell^{(\text{sqr})}(y,\hat{y}) = (y - \hat{y})^2$

logistic regression: three views

$$\log \frac{P(1 \mid x_1, x_2, \ldots, x_m)}{1 - P(1 \mid x_1, x_2, \ldots, x_m)} = w_0 + w_1 x_2 + w_2 x_2 + \ldots + w_m x_m$$

linear classifier

$$P(1 \mid x_1, x_2, \ldots, x_m) = \frac{1}{1 + e^{-(w_0 + w_1 x_2 + w_2 x_2 + \ldots + w_m x_m)}}$$

conditional model
logistic

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i (w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)})$$

linear model
minimizing logistic
loss

Overfitting

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)})$$

If we minimize this loss function, in practice, the results aren’t great and we tend to overfit

Solution?

Regularization/prior

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)}) + \lambda\, regularizer(w, b)$$

or

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)}) - \log(p(w, b))$$

What are some of the regularizers we know?

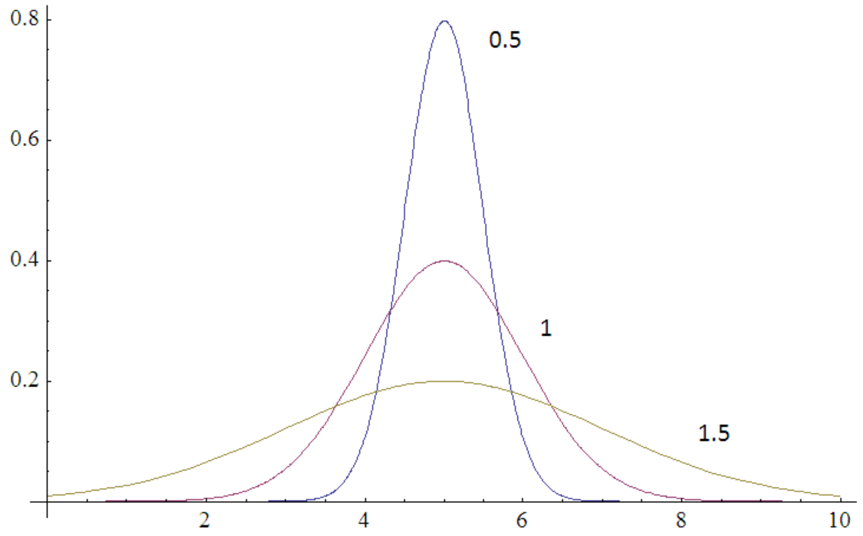
Regularization/prior

L2 regularization:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)}) + \lambda \|w\|^2$$

Gaussian prior:

p(w,b) ~



Regularization/prior

L2 regularization:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1x_2 + w_2x_2 + \ldots + w_mx_m + b)}) + \lambda \|w\|^2$$

Gaussian prior:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1x_2 + w_2x_2 + \ldots + w_mx_m + b)}) + \frac{1}{2\sigma^2}\|w\|^2$$

Does the λ make sense? $\lambda = \frac{1}{2\sigma^2}$

Regularization/prior

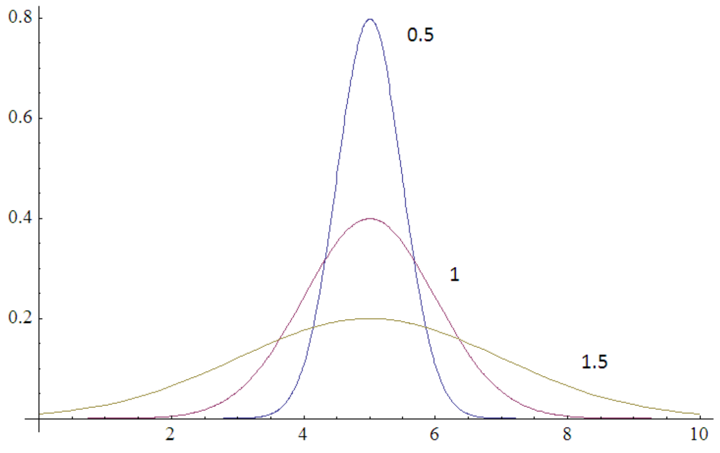
L2 regularization:

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)}) + \lambda \|w\|^2$$

Gaussian prior:

$$\operatorname{argmin}_{w,b} \sum_{i=1}^{n} \log(1+e^{-y_i(w_1x_2+w_2x_2+\ldots+w_mx_m+b)}) + \frac{1}{2\sigma^2}\|w\|^2$$

$$\lambda = \frac{1}{2\sigma^2}$$



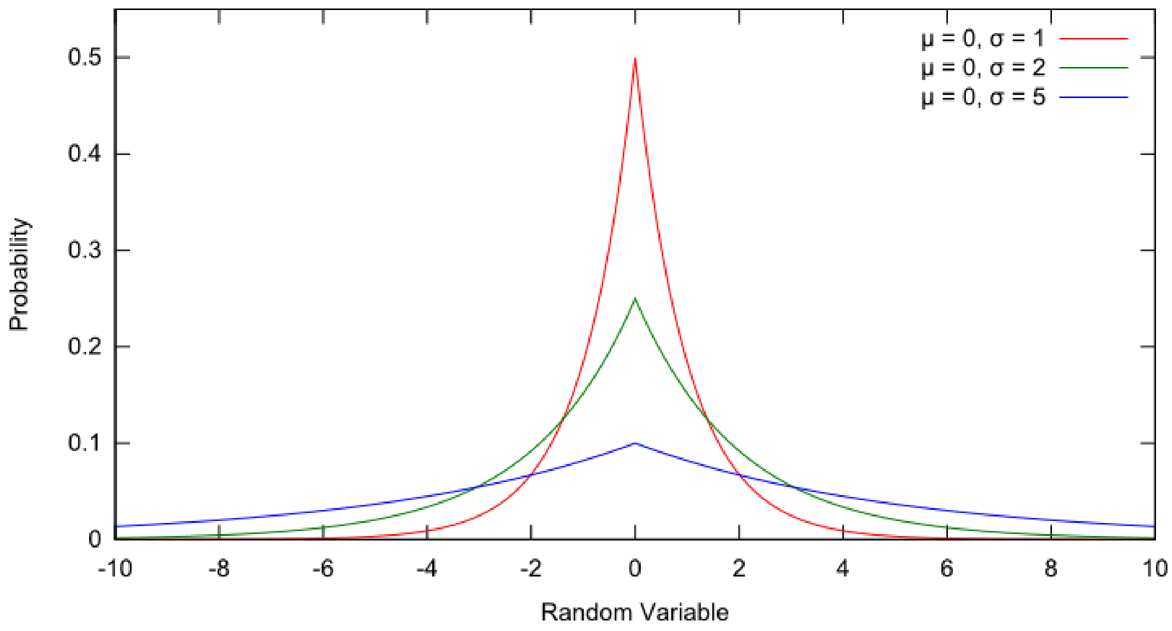
Regularization/prior

L1 regularization:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1 x_2 + w_2 x_2 + \ldots + w_m x_m + b)}) + \lambda \|w\|$$

Laplacian prior:

p(w,b) ~



Regularization/prior

L1 regularization:

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1x_2 + w_2x_2 + \ldots + w_mx_m + b)}) + \lambda \|w\|$$

Laplacian prior:

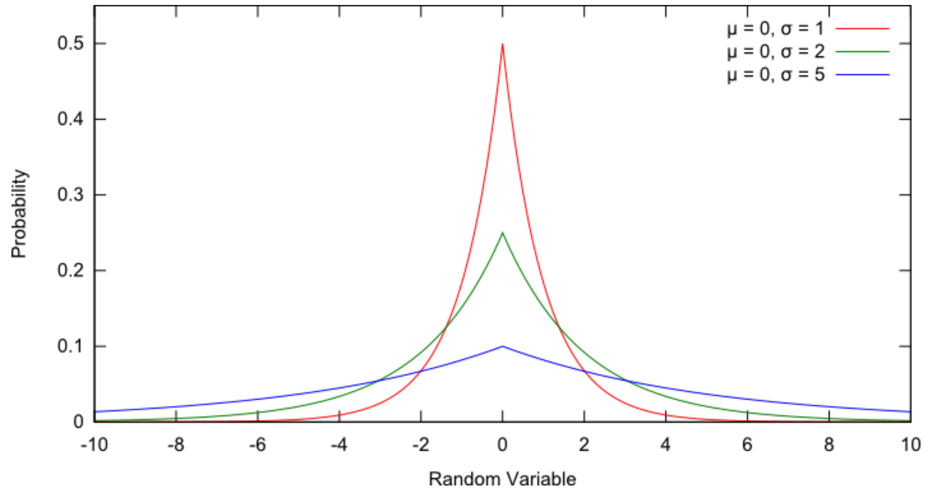
$$\text{argmin}_{w,b} \sum_{i=1}^{n} \log(1 + e^{-y_i(w_1x_2 + w_2x_2 + \ldots + w_mx_m + b)}) + \frac{1}{\sigma} \|w\|$$

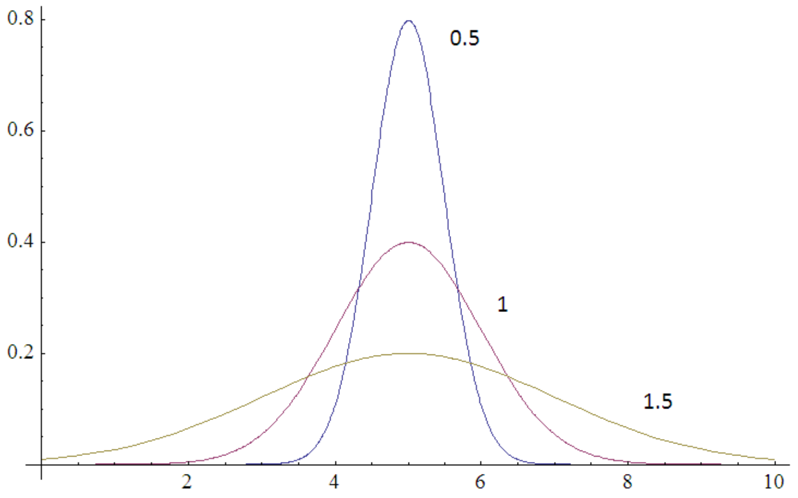
$$\lambda = \frac{1}{2\sigma^2}$$

L1 vs. L2

L1 = Laplacian prior

L2 = Gaussian prior

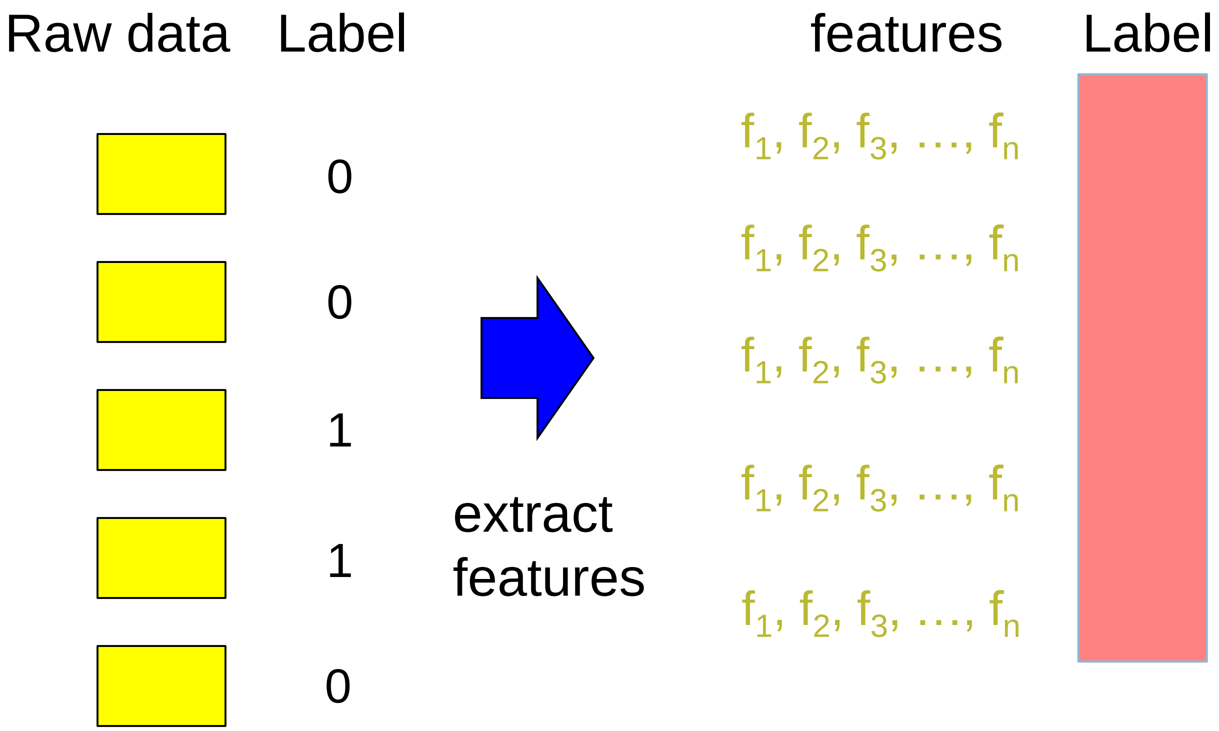




Logistic regression

Why is it called logistic regression?

A digression: regression vs. classification

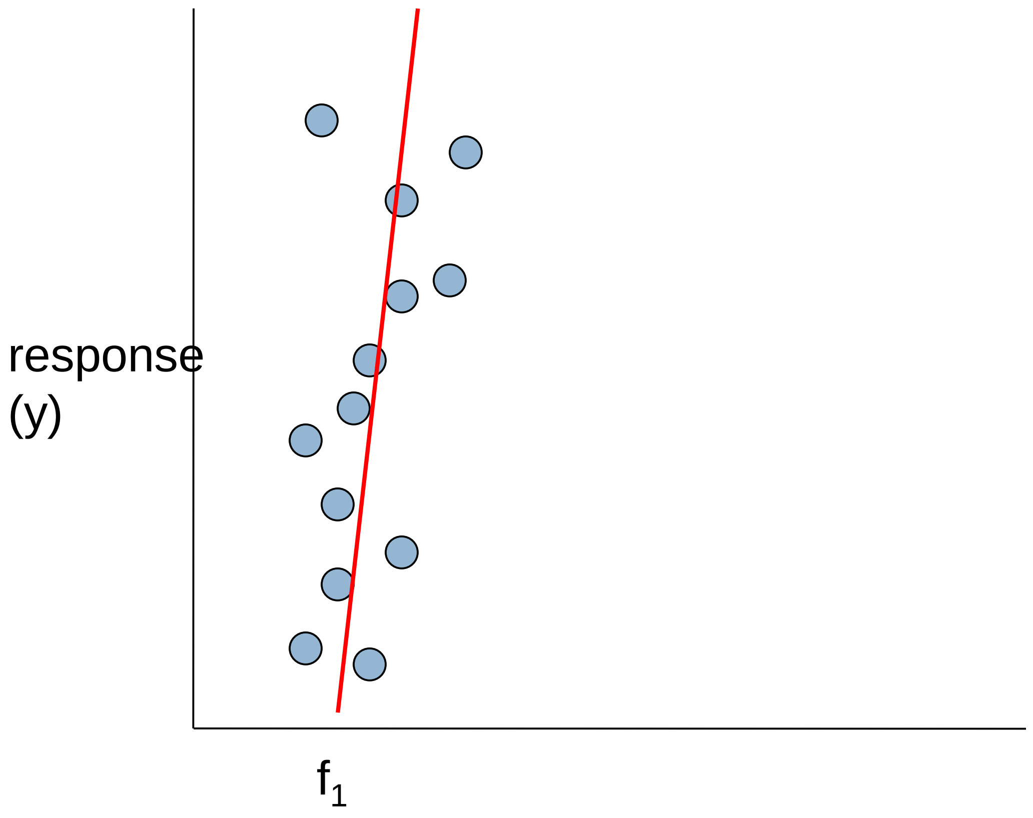


Raw data
Label
0
0
1
1
0
extract features
features
Label
$f_1, f_2, f_3, \ldots, f_n$
$f_1, f_2, f_3, \ldots, f_n$
$f_1, f_2, f_3, \ldots, f_n$
$f_1, f_2, f_3, \ldots, f_n$
$f_1, f_2, f_3, \ldots, f_n$

classification: discrete (some finite set of labels)

regression: real value

linear regression



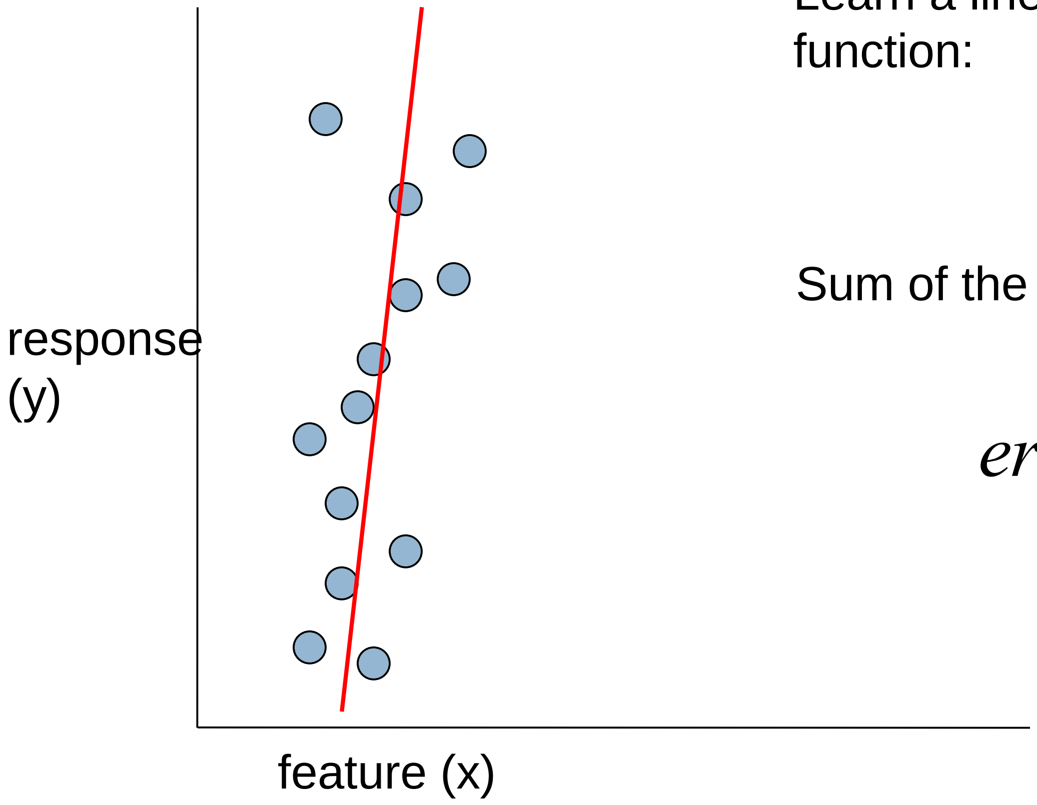
How can we find this line?

Given some points, find the ***line*** that best fits/explains the data

Our model is a line, i.e. we're assuming a linear relationship between the feature and the label value

$$h(y) = w_1 x_1 + b$$

Linear regression



Learn a line *h* that minimizes some loss/error function:

$$error(h) = ?$$

Sum of the individual errors:

$$error(h) = \sum_{i=1}^{n} \underbrace{\left| y_i - h(f_i) \right|}_{\text{0/1 loss!}}$$

Error minimization

How do we find the minimum of an equation?

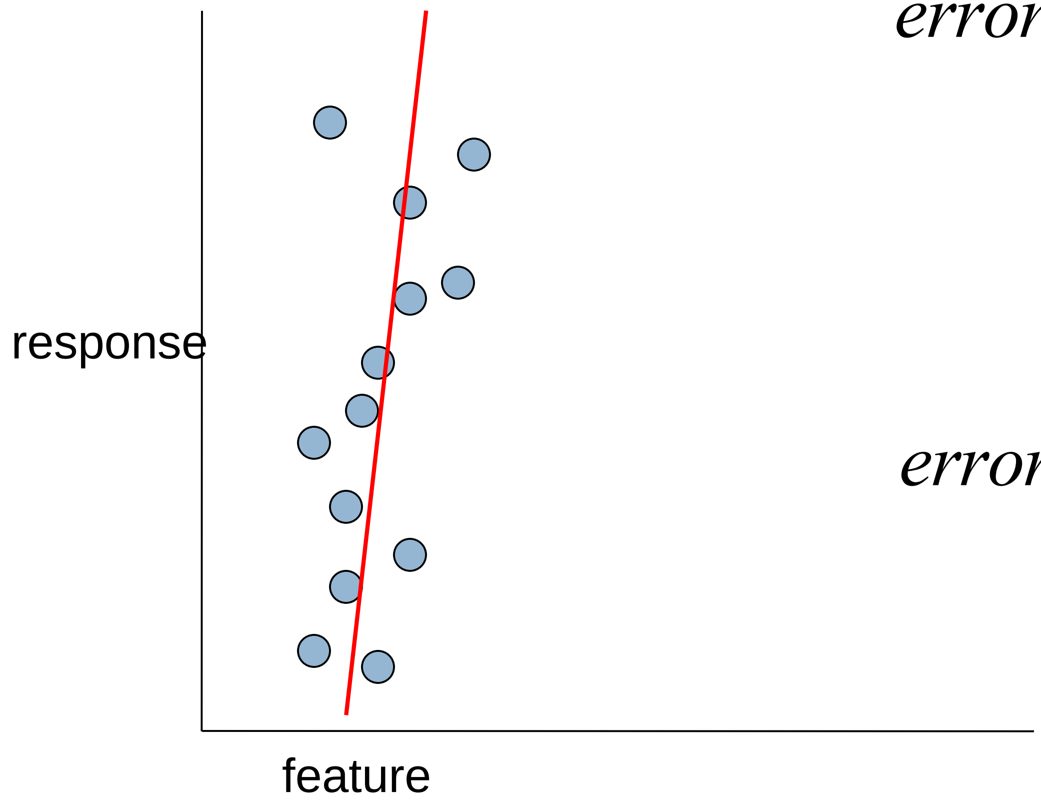
$$error(h) = \sum_{i=1}^{n} |y_i - h(f_i)|$$

Take the derivative, set to 0 and solve (going to be a min or a max)

Any problems here?

Ideas?

Linear regression



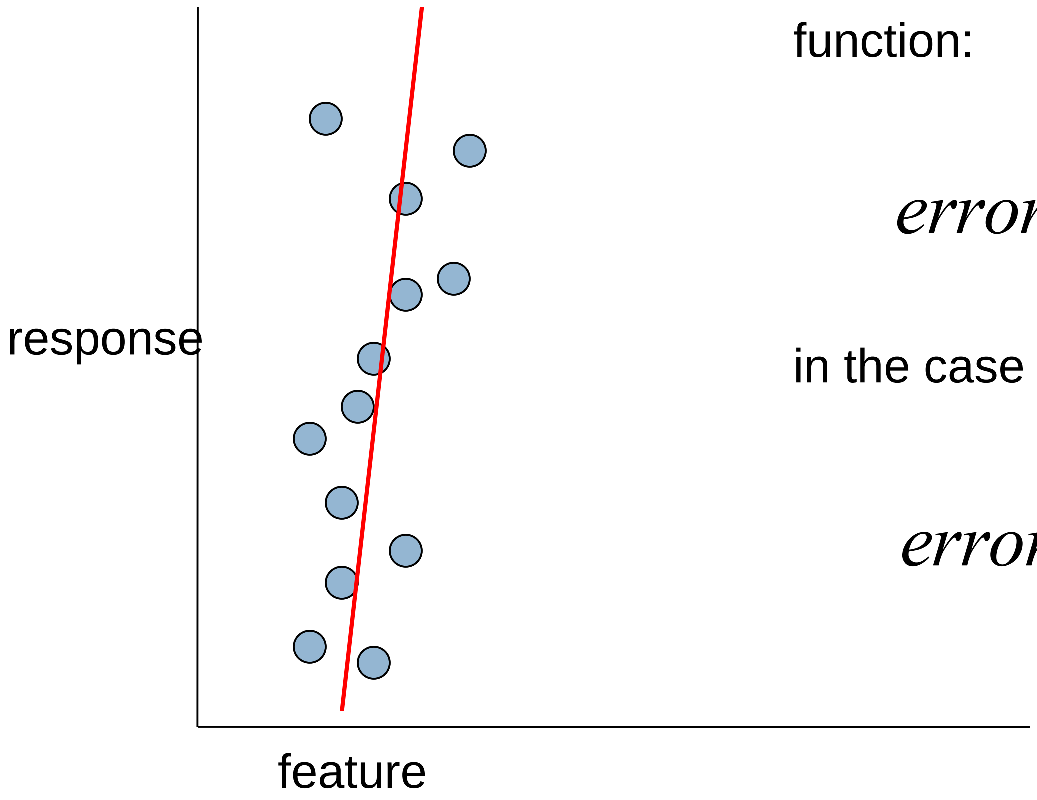
$$error(h) = \sum_{i=1}^{n} |y_i - h(f_i)|$$

$$\Downarrow$$

$$error(h) = \sum_{i=1}^{n} (y_i - h(f_i))^2$$

squared error is convex!

Linear regression



Learn a line *h* that minimizes an error function:

$$error(h) = \sum_{i=1}^{n} (y_i - h(f_i))^2$$

in the case of a 2d line:

$$error(h) = \sum_{i=1}^{n} (y_i - \underbrace{(w_1 x_1 + w_0)}_{\text{function for a line}})^2$$

Linear regression

We'd like to *minimize* the error

Find w_1 and w_0 such that the error is minimized

$$error(h) = \sum_{i=1}^{n} (y_i - (w_1 f_i + w_0))^2$$

We can solve this in closed form

Multiple linear regression

If we have m features, then we have a line in *m* dimensions

$$h(\bar{f}) = \underset{\uparrow}{w_0} + \underset{\uparrow}{w_1} f_1 + w_2 f_2 + \ldots + w_m f_m$$

weight
s

Multiple linear regression

We can still calculate the squared error like before

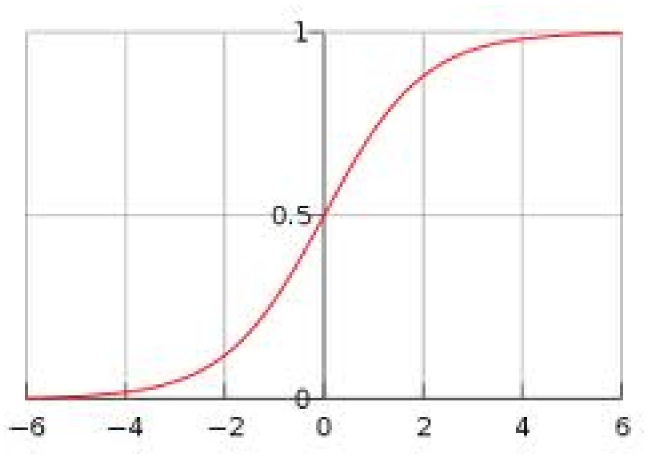
$$h(\bar{f}) = w_0 + w_1 f_1 + w_2 f_2 + \ldots + w_m f_m$$

$$error(h) = \sum_{i=1}^{n} (y_i - (w_0 + w_1 f_1 + w_2 f_2 + \ldots + w_m f_m))^2$$

Still can solve this exactly!

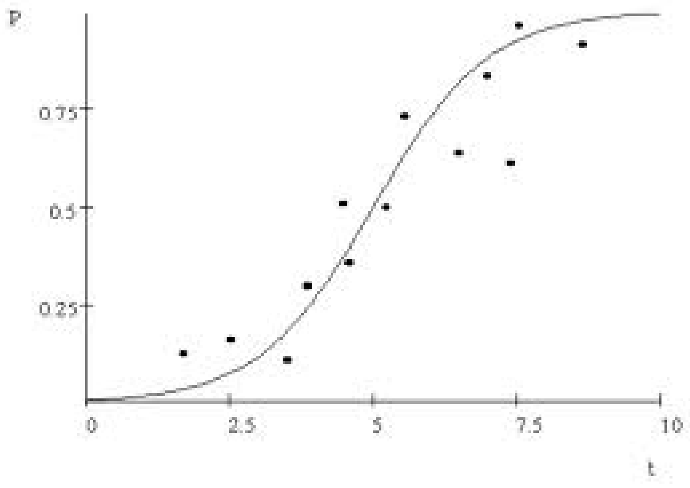
Logistic function

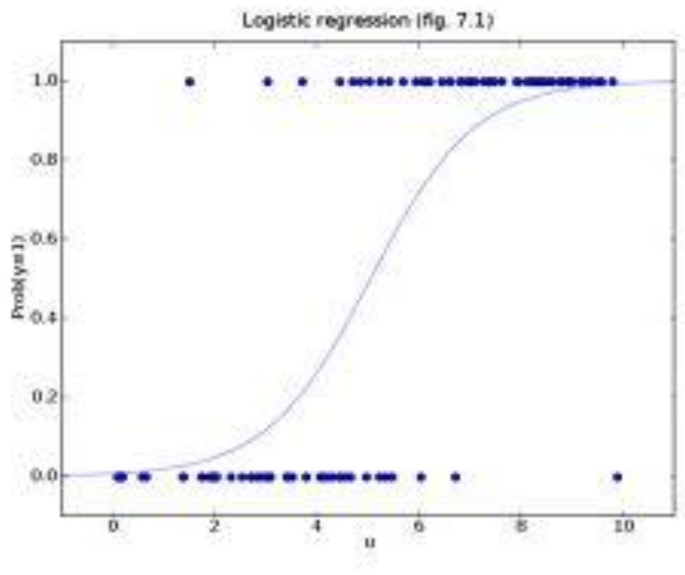
$$\text{logistic} = \frac{1}{1+e^{-x}}$$



Logistic regression

Find the best fit of the data based on a logistic





CSE419 – Artificial Intelligence and Machine Learning 2020

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/CSE419_2020

Lecture 14 Part 2

Ensemble Learning

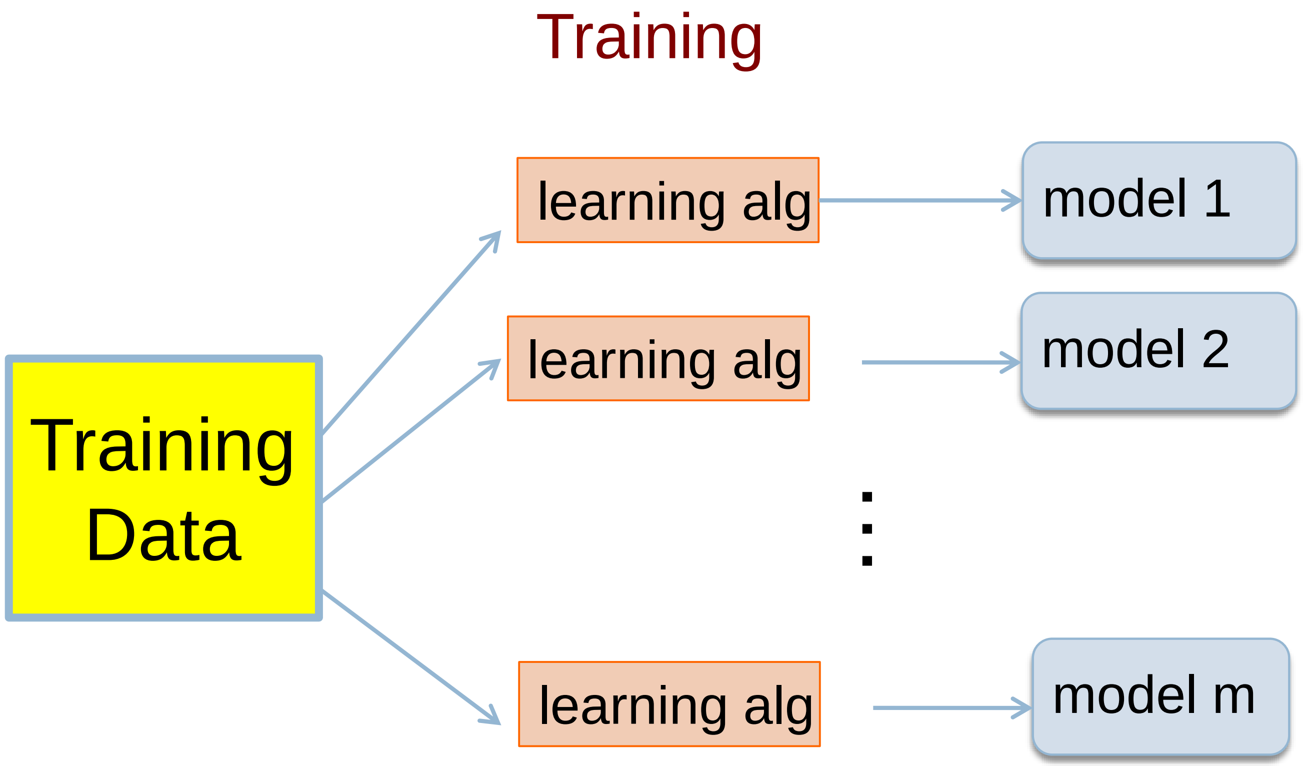
Based on Asst. Prof. Dr. David Kauchak (Pomona College) Lecture Slides

Ensemble learning

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

Ensemble learning

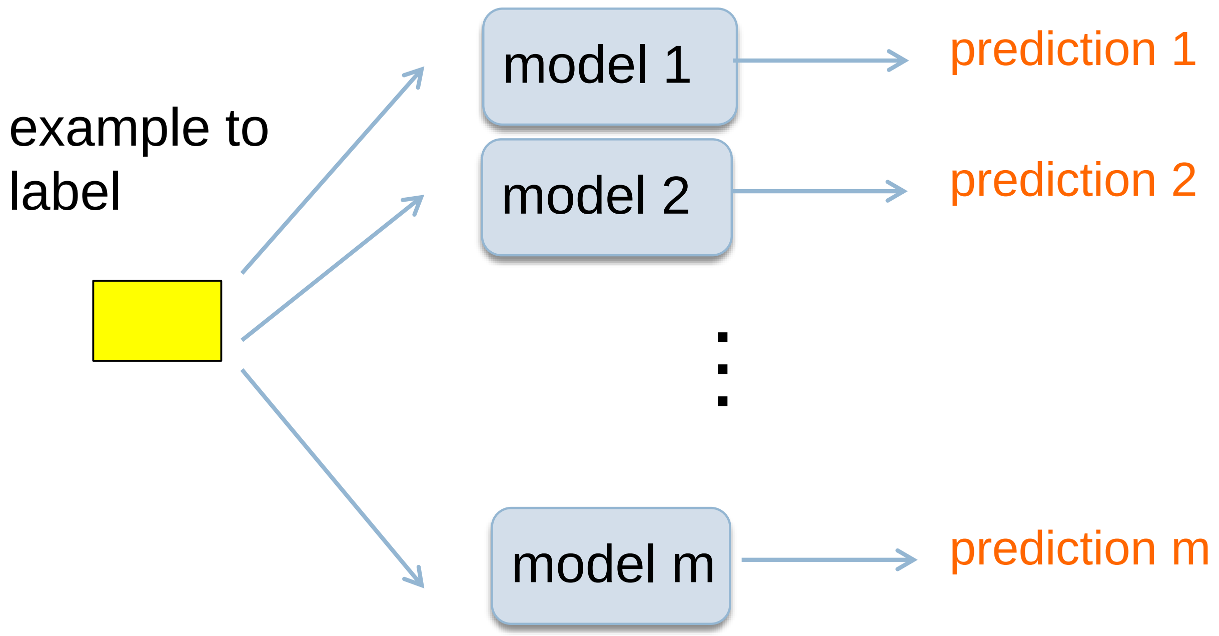
Basic idea: if one classifier works well, why not use multiple classifiers!



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 us?

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

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

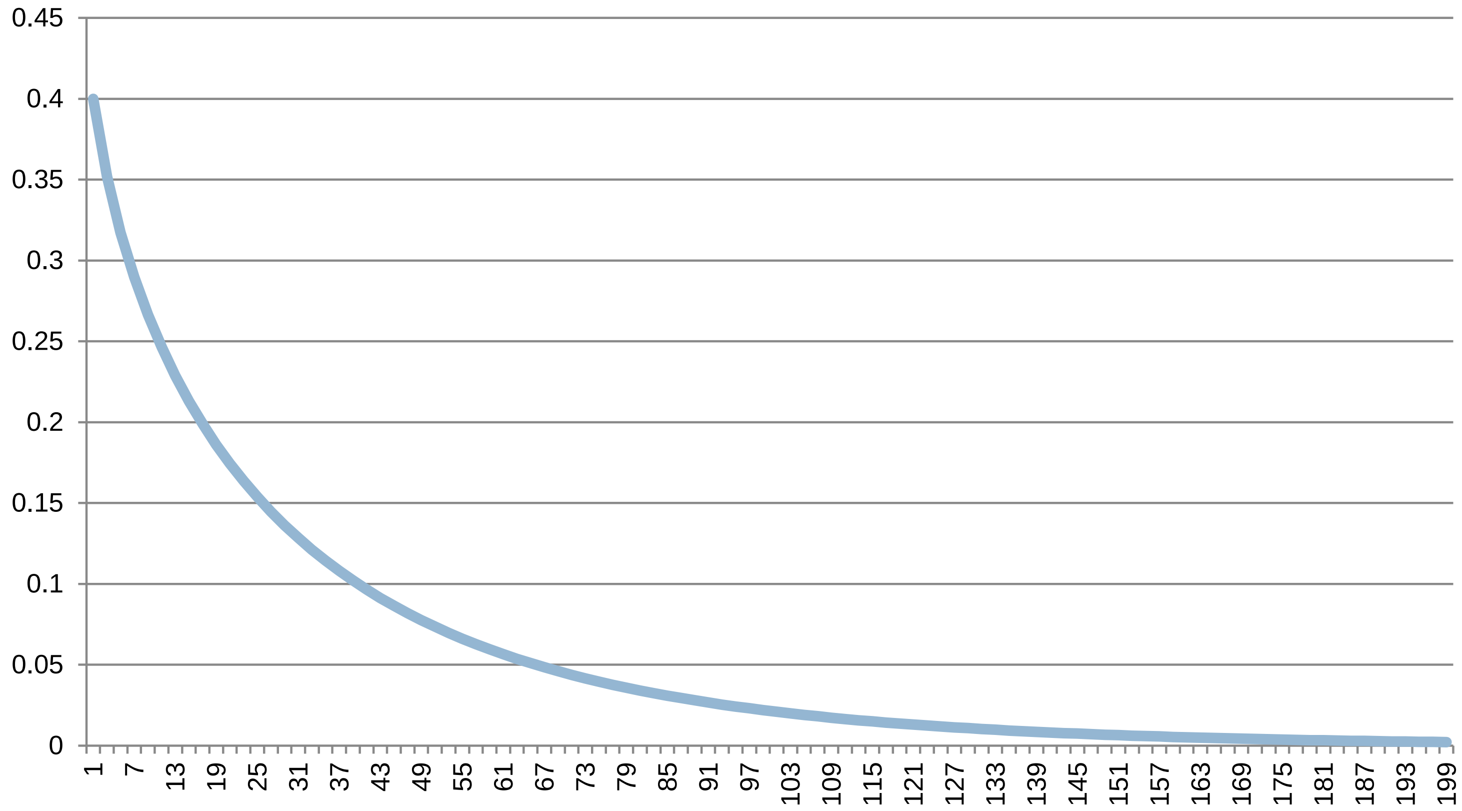
m classifiers in general, for r = probability of mistake for individual classifier:

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

(cumulative probability distribution for the binomial distribution)

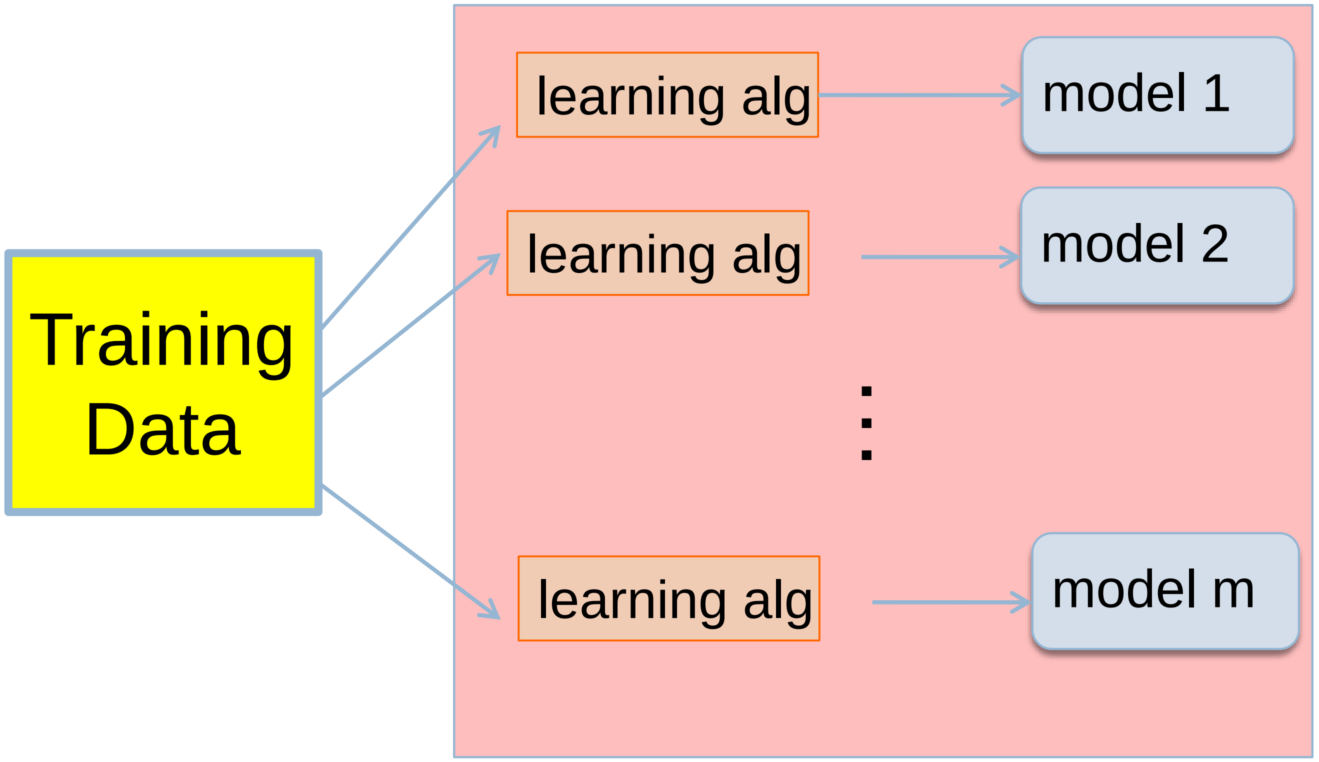
Given enough classifiers…

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



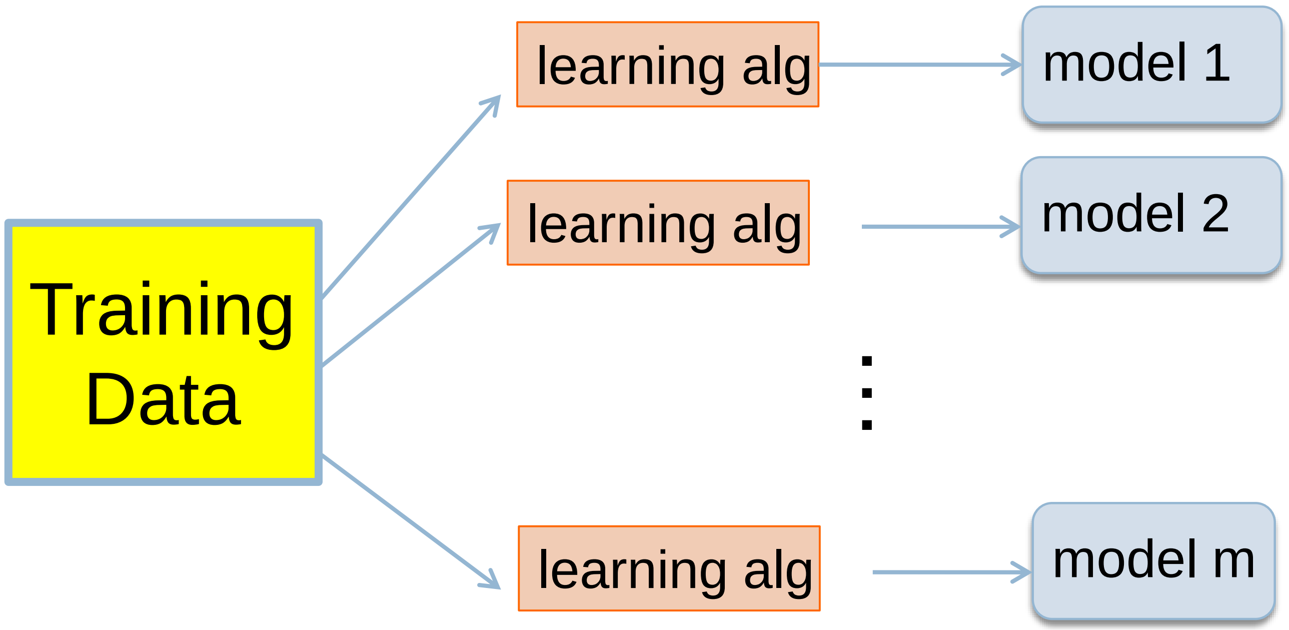
r = 0.4

Obtaining independent classifiers



Where to 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

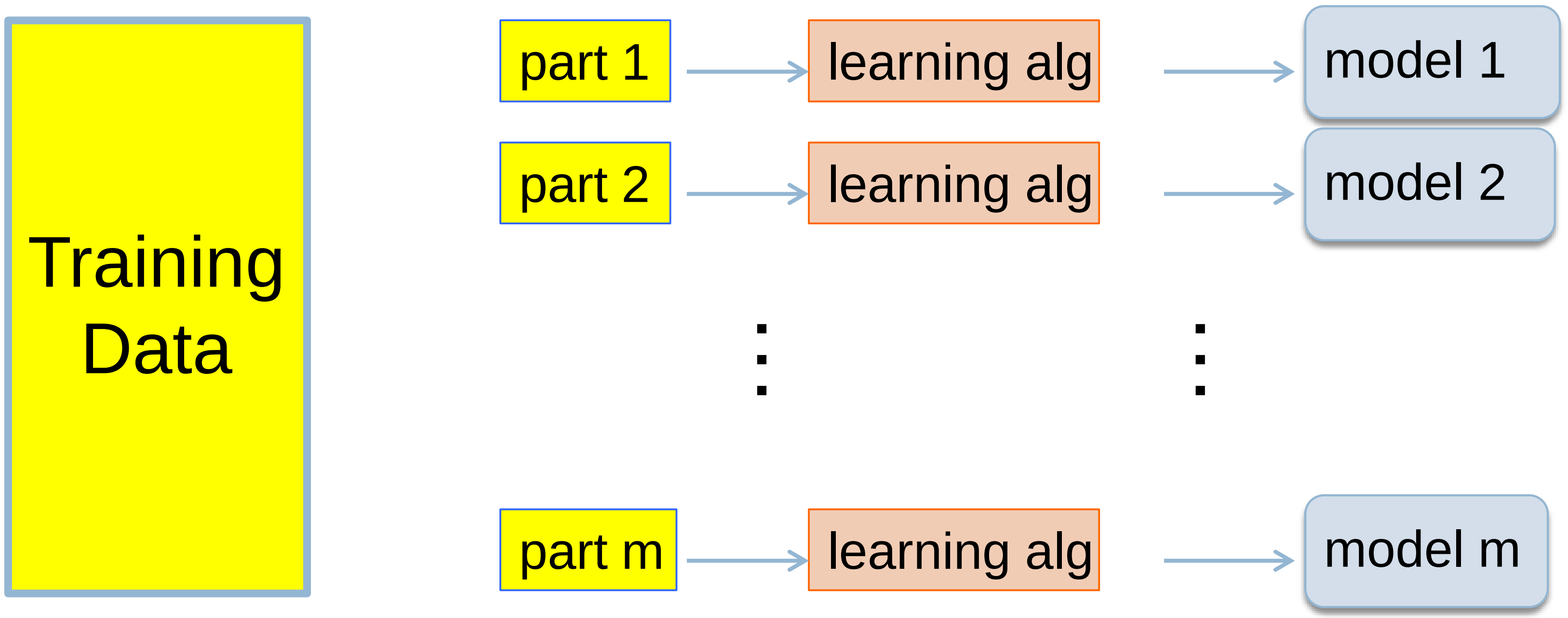
Pros:

- 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

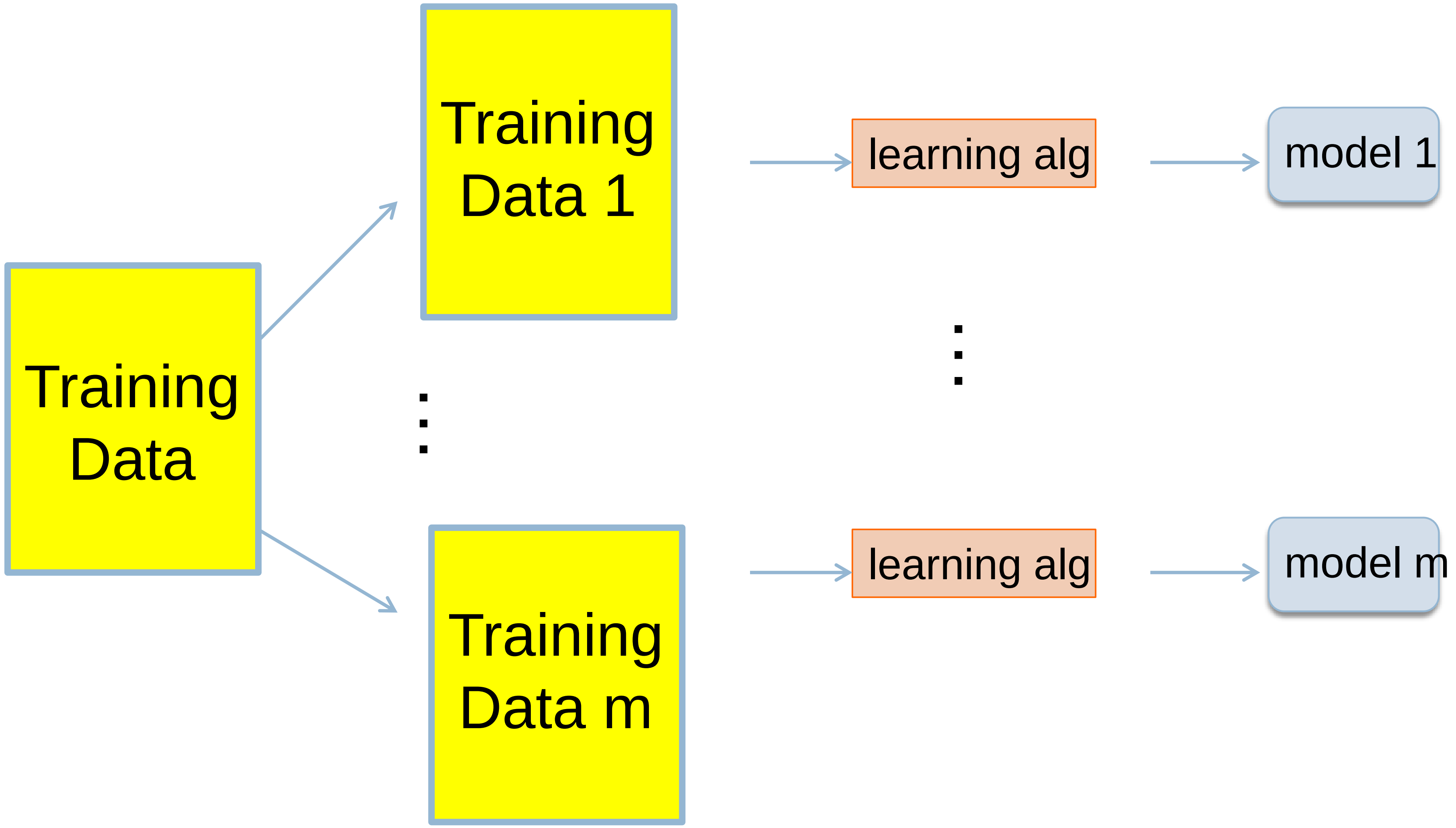
Pros:

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

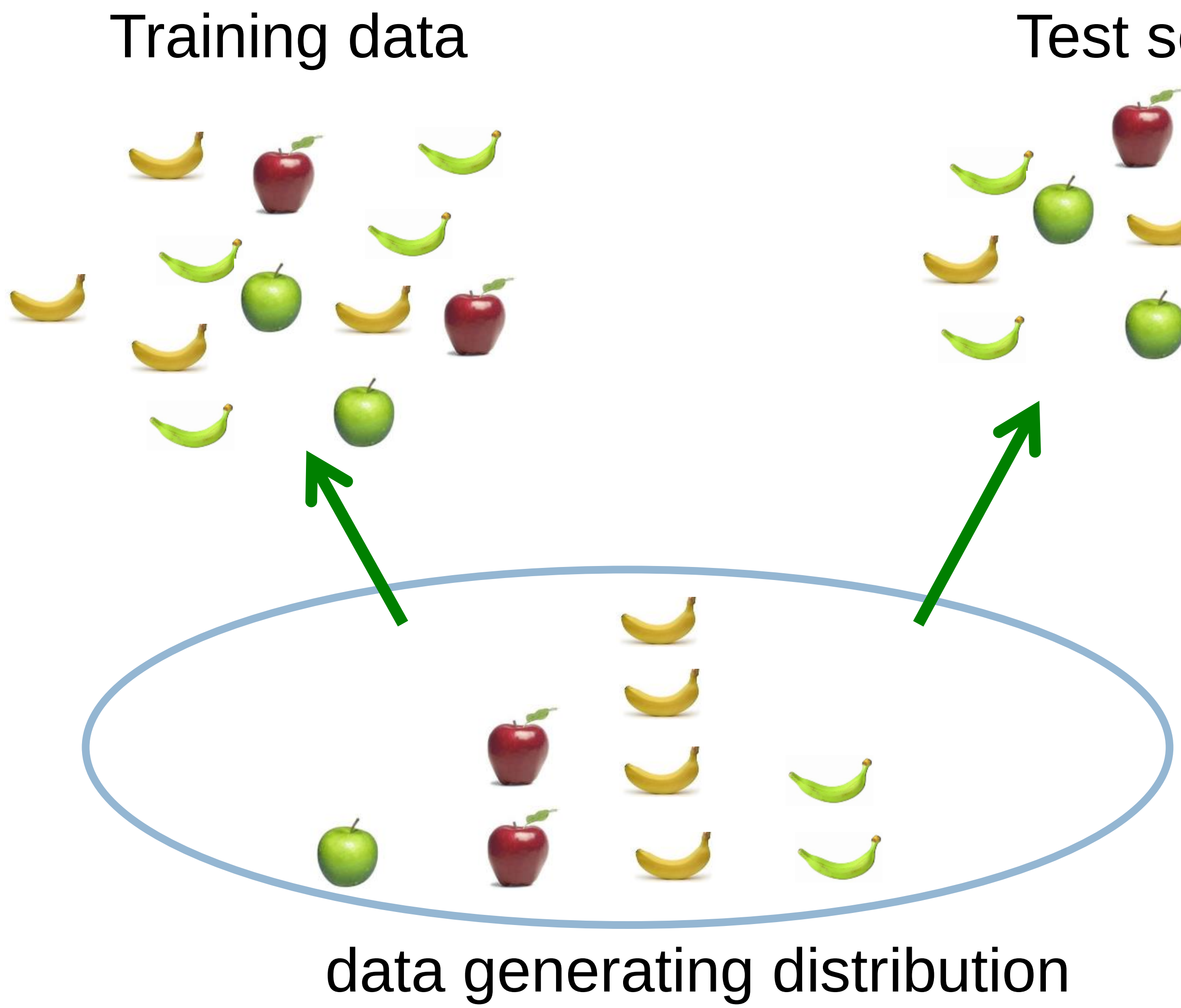
Cons/concerns:

- 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

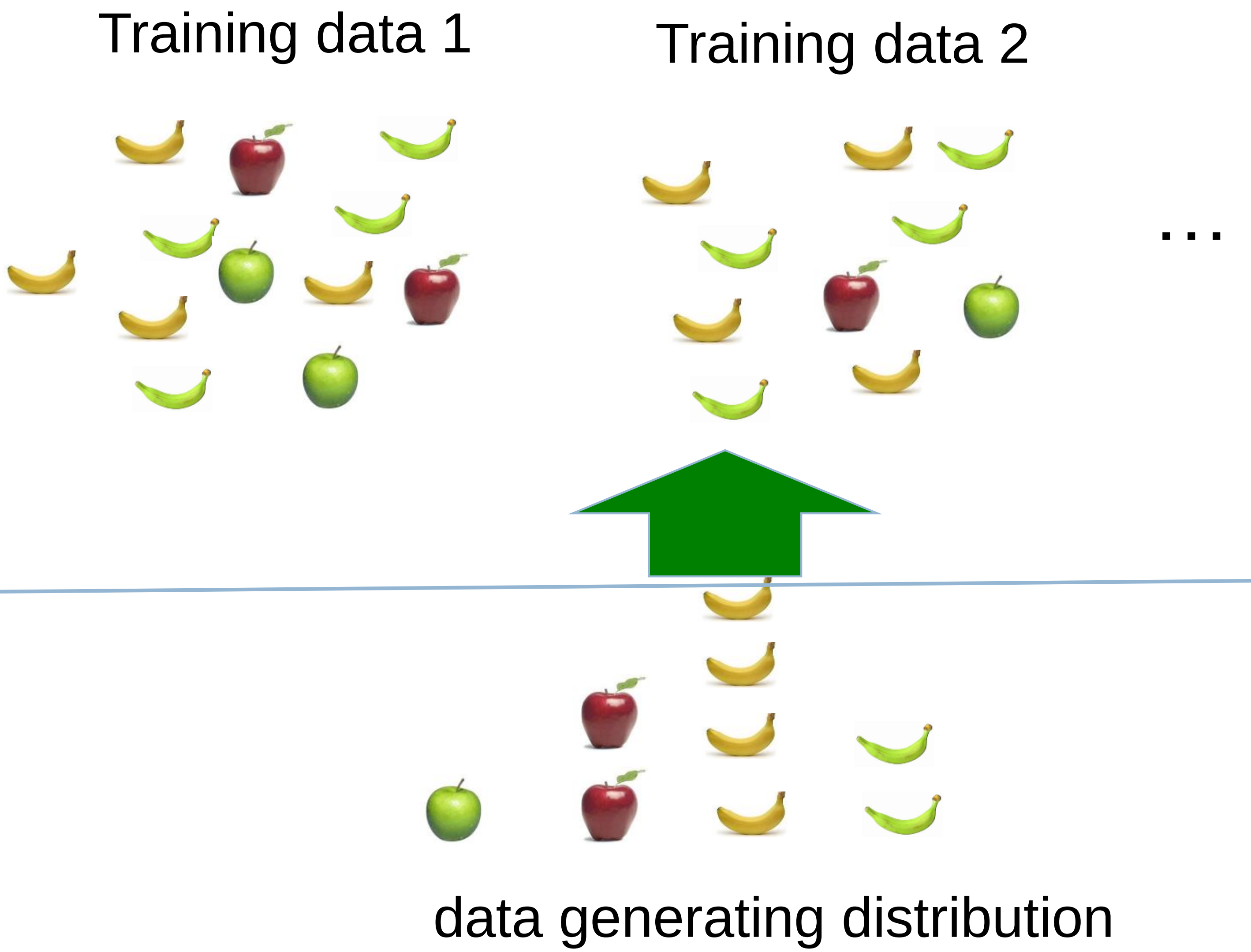
Idea 3: bagging



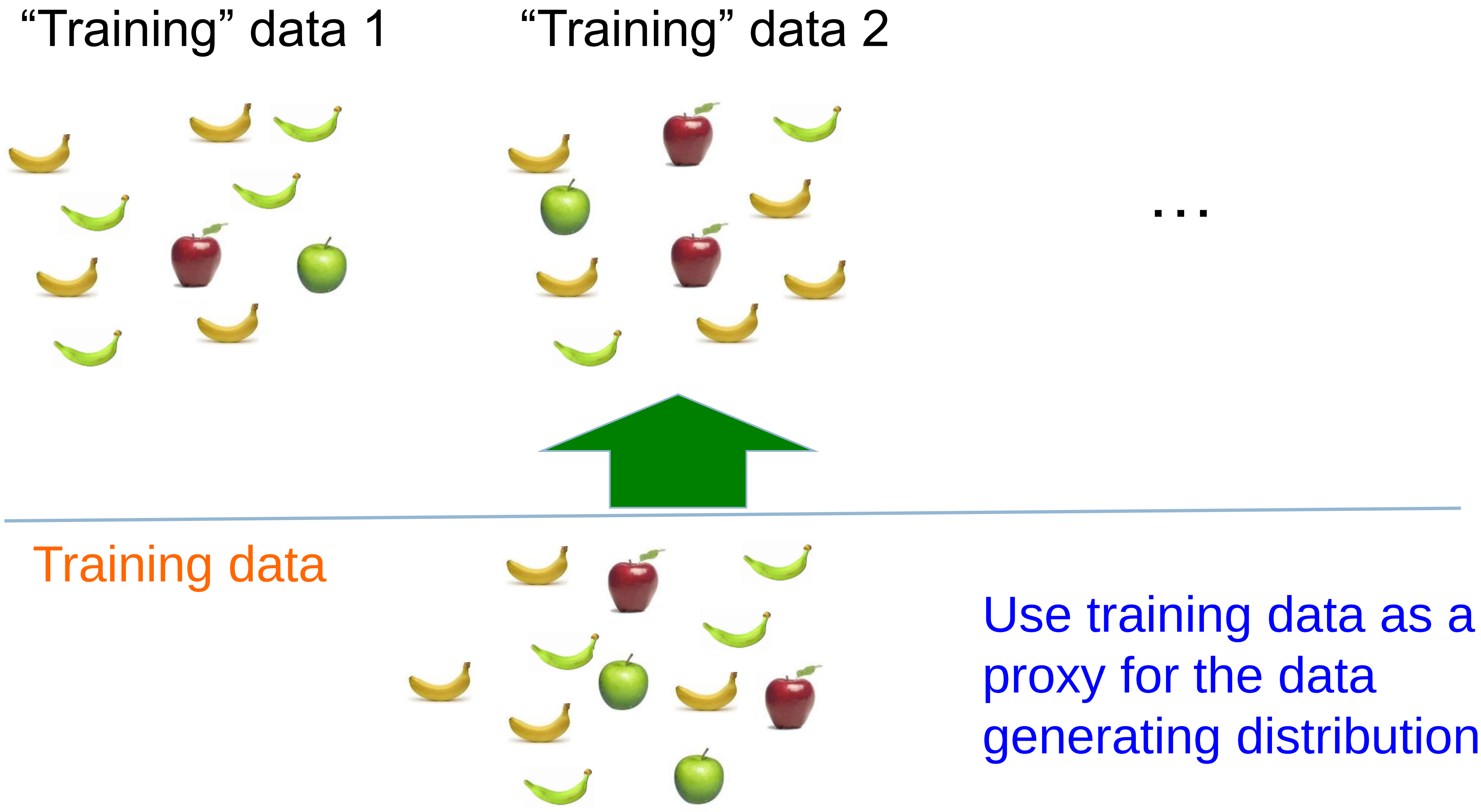
data generating distribution



Ideal situation



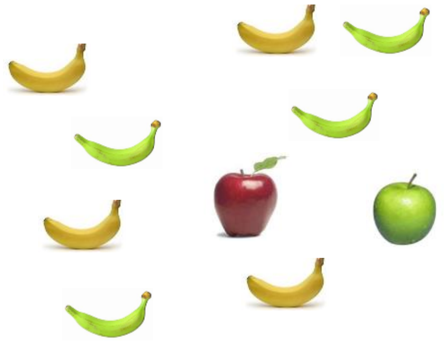
bagging



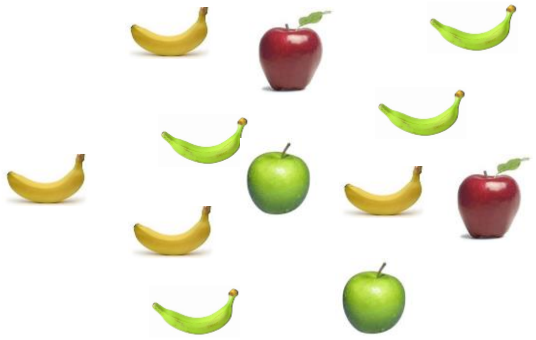
Use training data as a proxy for the data generating distribution

sampling with replacements

"Training" data 1



Training data

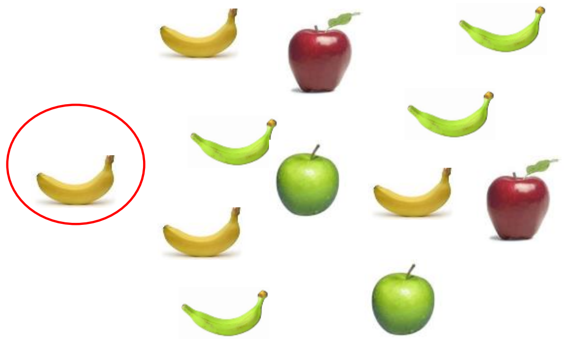


sampling with replacements

"Training" data 1

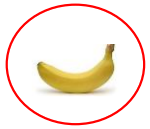
pick a random example from the real training data

Training data



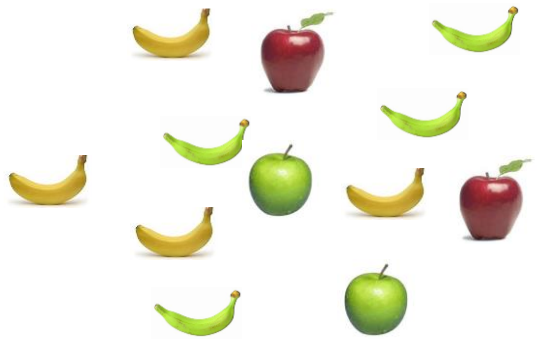
sampling with replacements

“Training” data 1



add it to the new “training” data

Training data

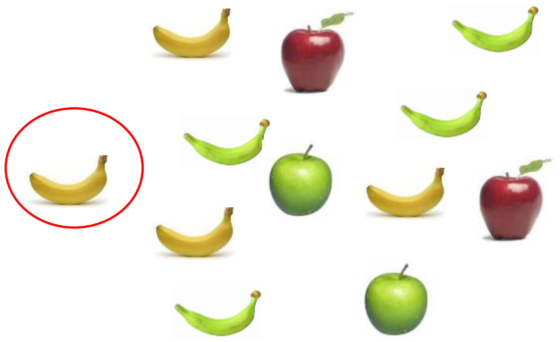


sampling with replacements

"Training" data 1

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

Training data

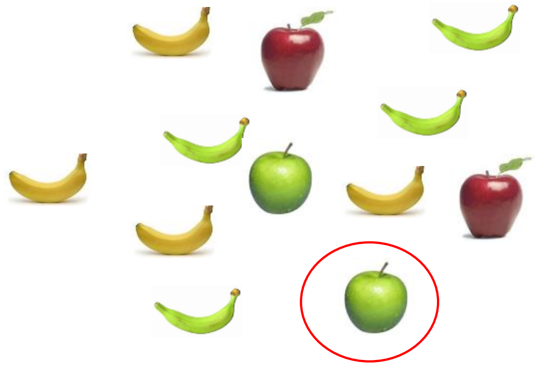


sampling with replacements

"Training" data 1

pick another random example

Training data

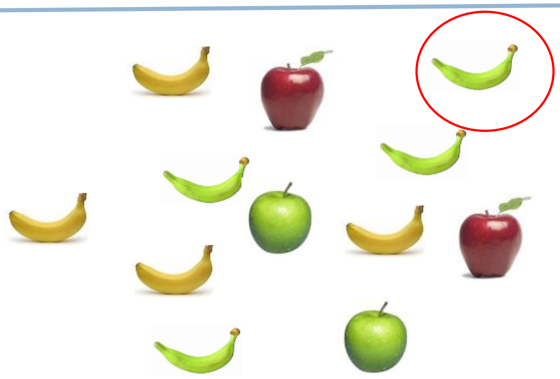


sampling with replacements

“Training” data 1

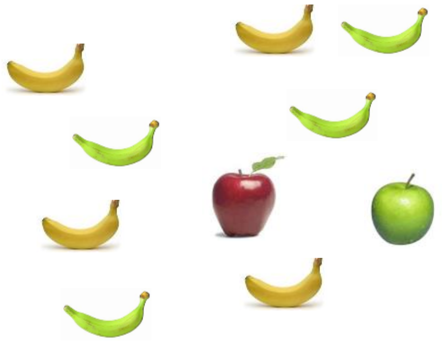
pick another random example

Training data



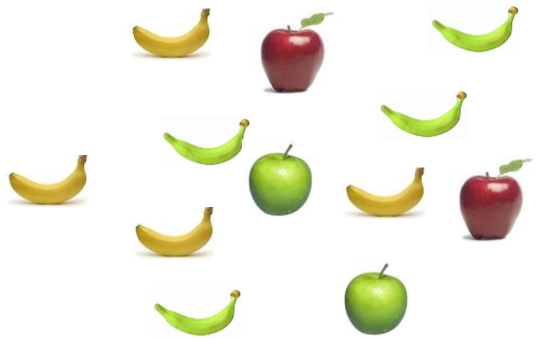
sampling with replacements

“Training” data 1



keep going until you’ve created a new “training” data set

Training data



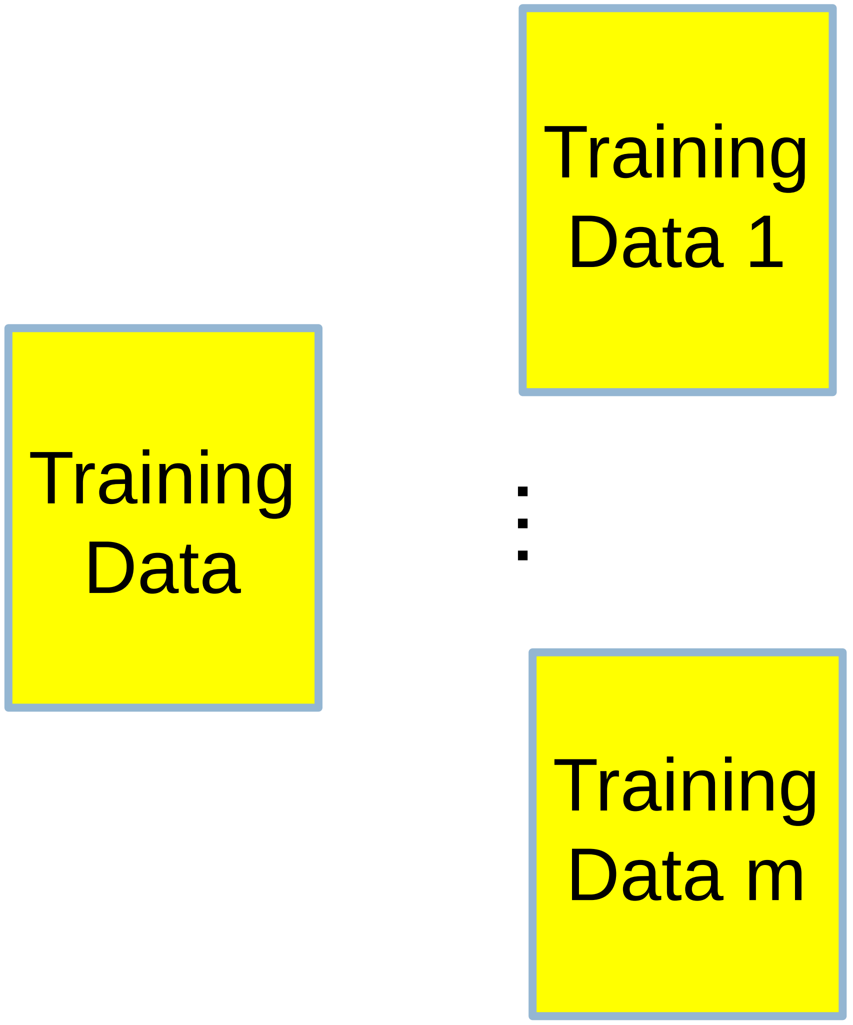
bagging

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

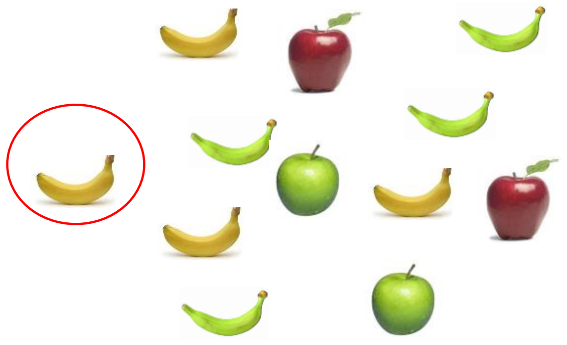


Won’t these all be basically the same?

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

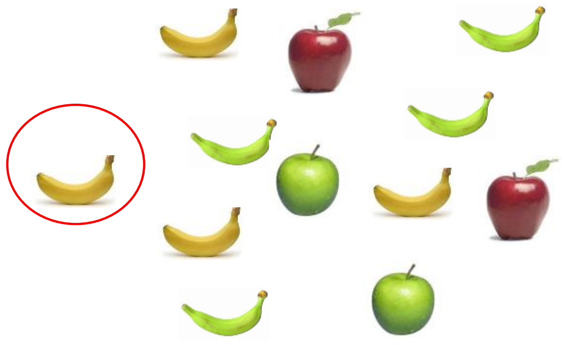


bagging concerns

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

$$1 - 1/n$$

Training data



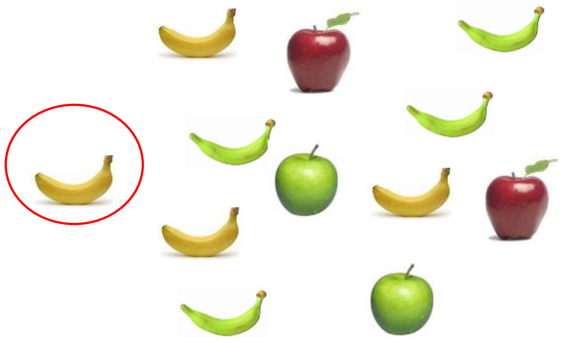
bagging concerns

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

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

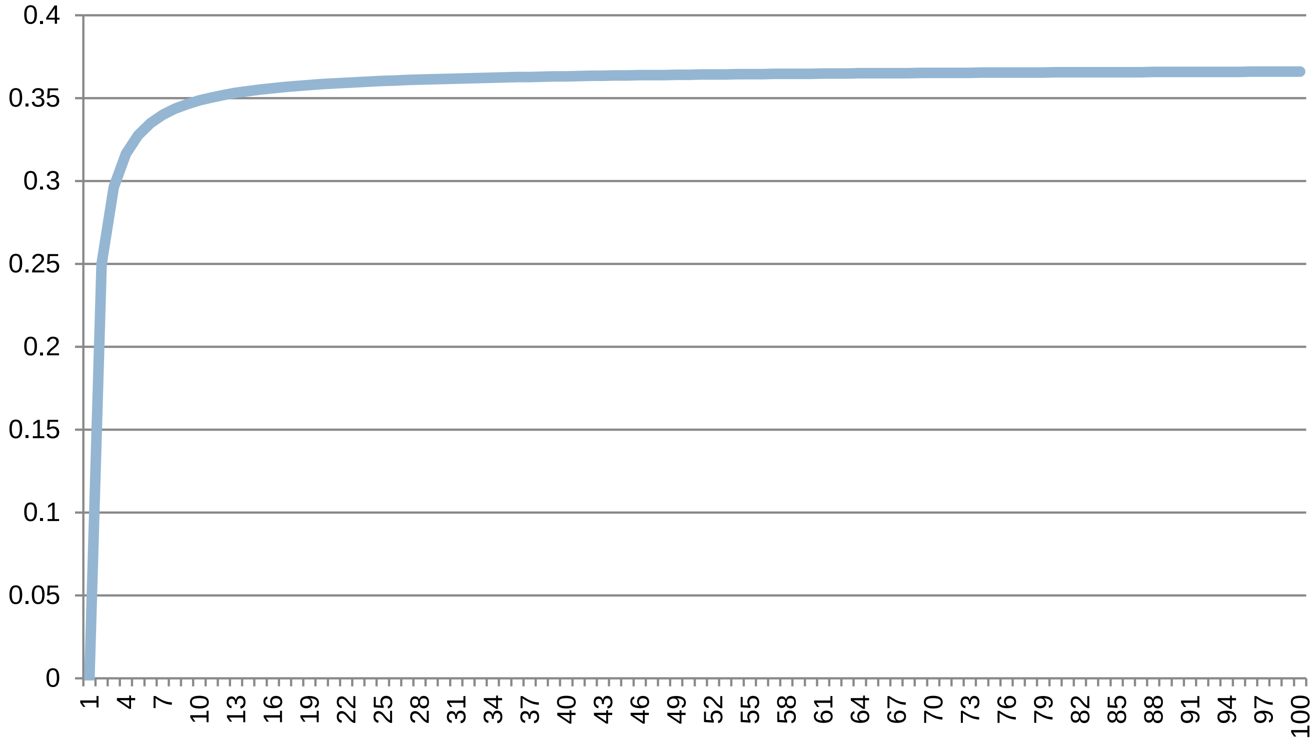
Each draw is independent
and has the same probability

Training data



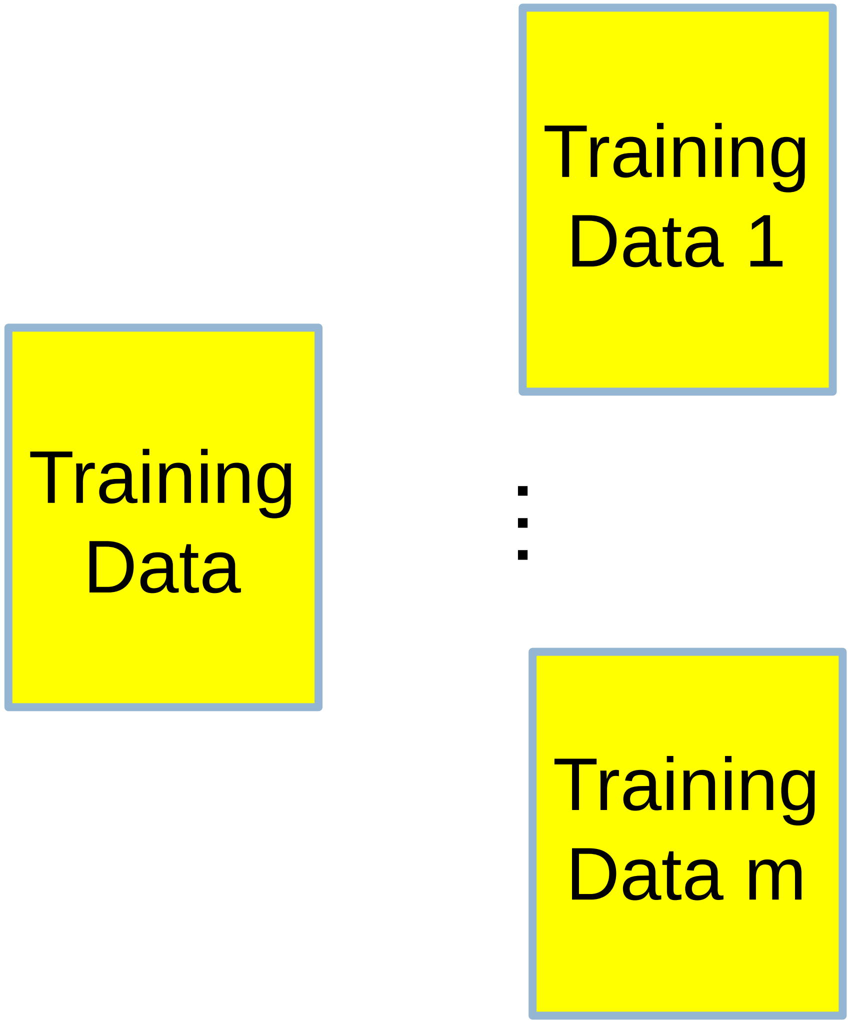
probability of overlap

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



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 a third 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

CSE419 – Artificial Intelligence and Machine Learning 2020

PhD Furkan Gözükara, Toros University

https://github.com/FurkanGozukara/CSE419_2020

Lecture 14 Part 3

Boosting

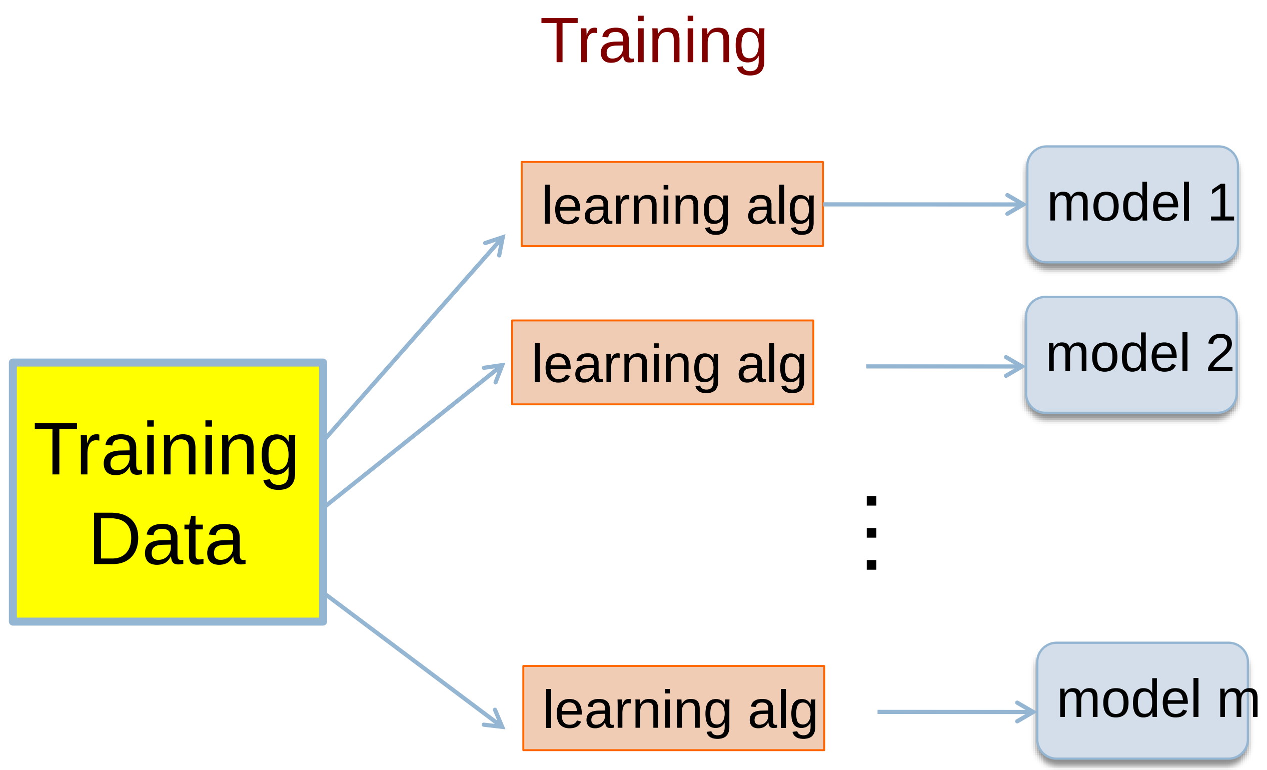
Based on Asst. Prof. Dr. David Kauchak (Pomona College) Lecture Slides

Ensemble learning

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

Ensemble learning

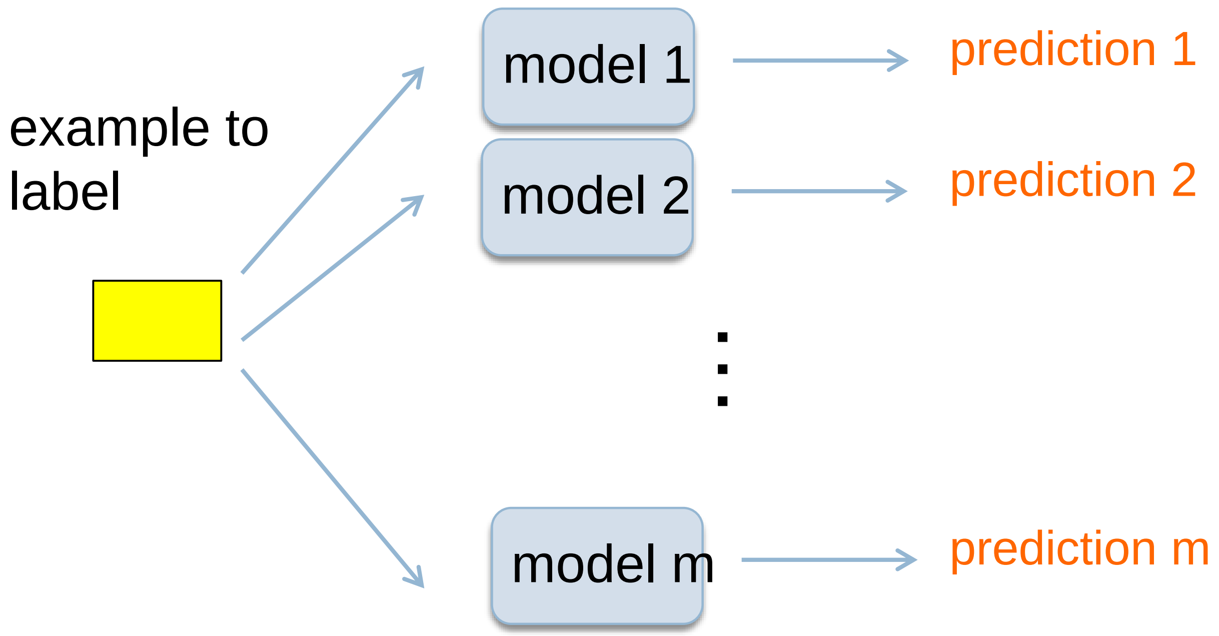
Basic idea: if one classifier works well, why not use multiple classifiers!



Ensemble learning

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

Testing



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

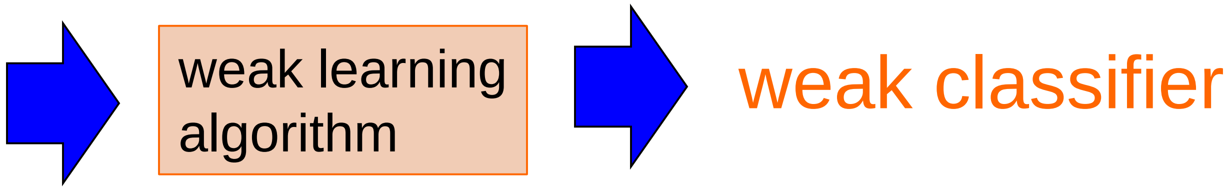
- a reasonable amount of training data
- a failure probability *p*

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



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:

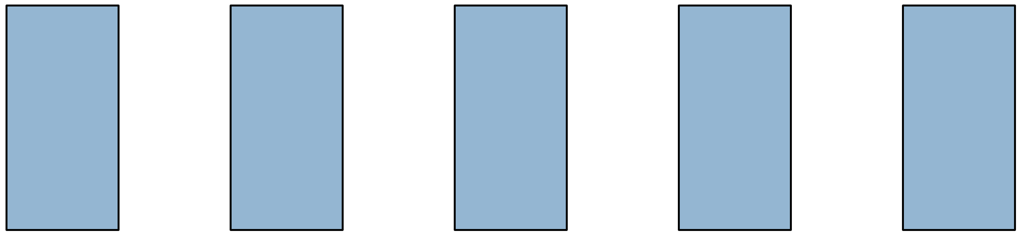
- learn a weak classifier and save
- change the example weights

Classify:

- get prediction from all learned weak classifiers
- weighted vote based on how well the weak classifier did when it was trained

boosting basics

Start with equal weighted examples

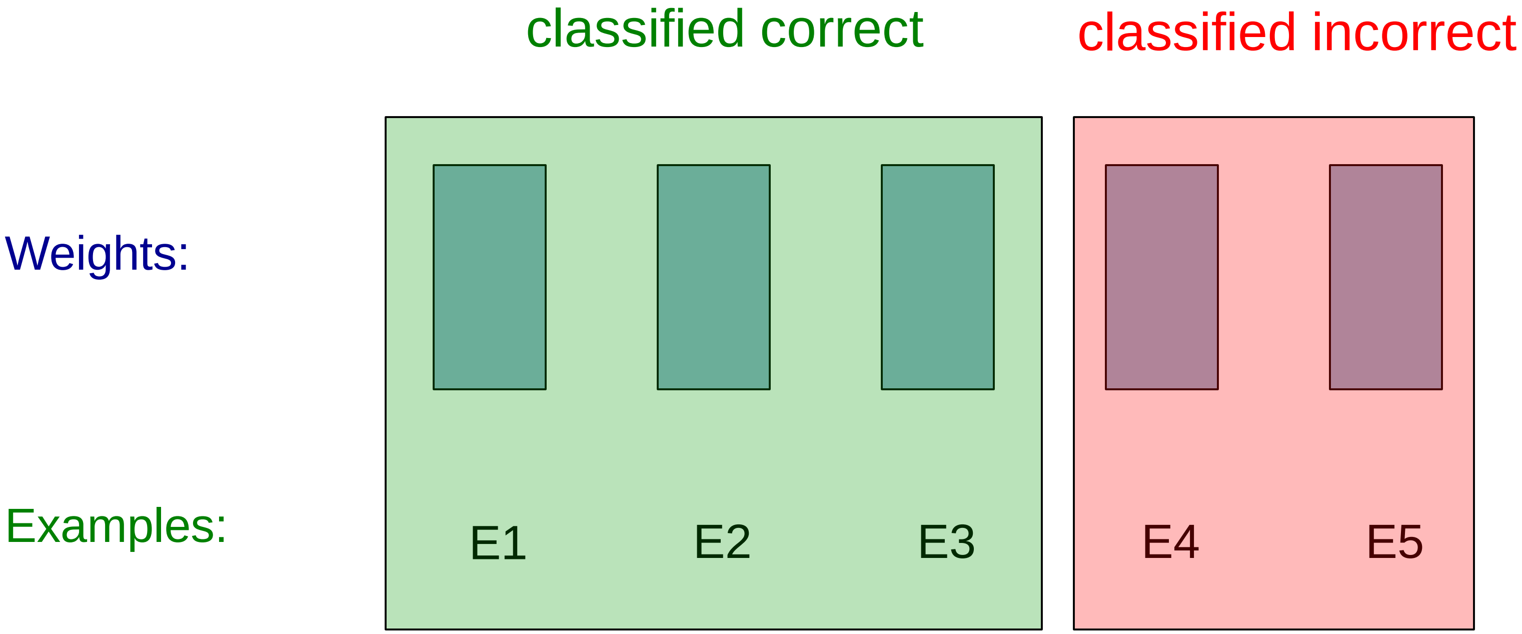


Weights:

Examples: E1 E2 E3 E4 E5

Learn a weak classifier

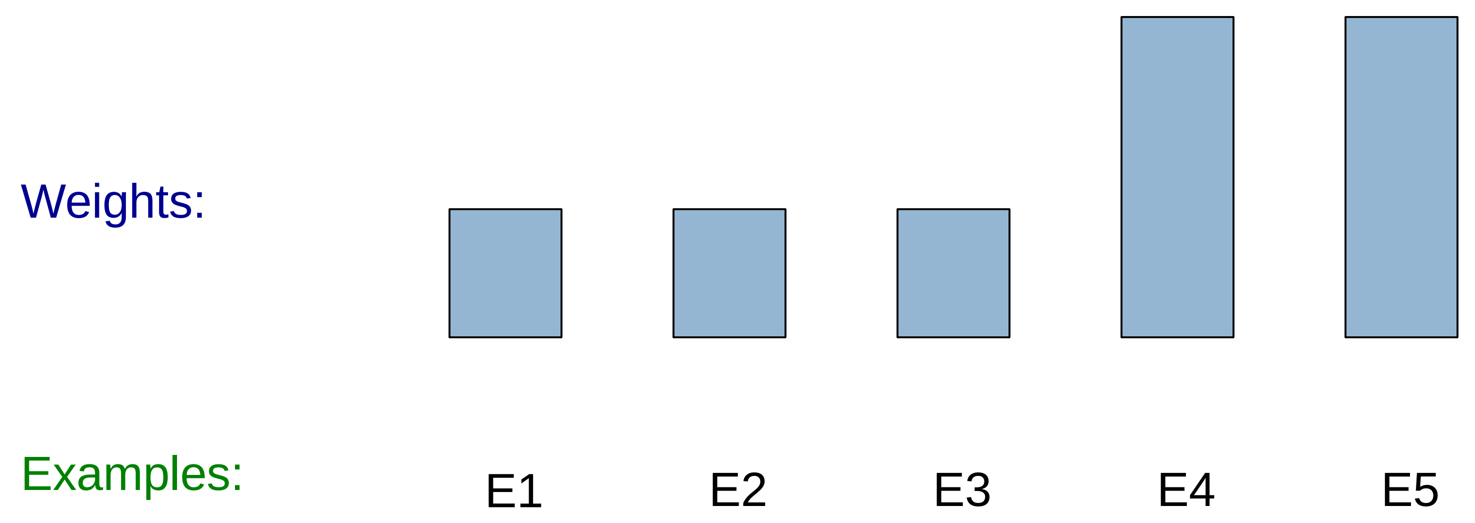
Boosting



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

How should we change the example weights?

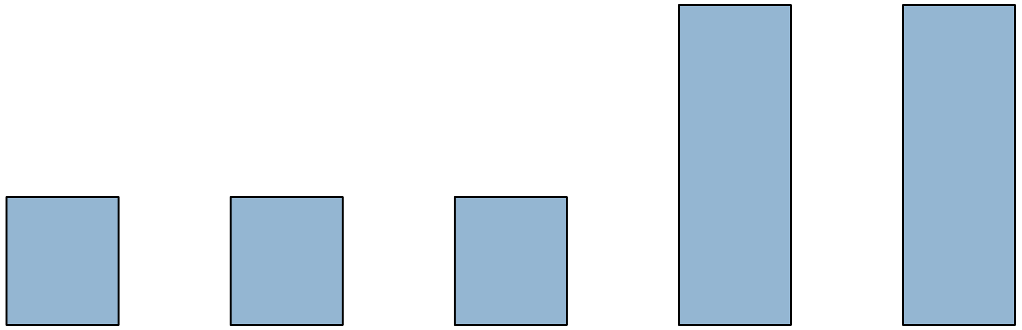
Boosting



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

Boosting

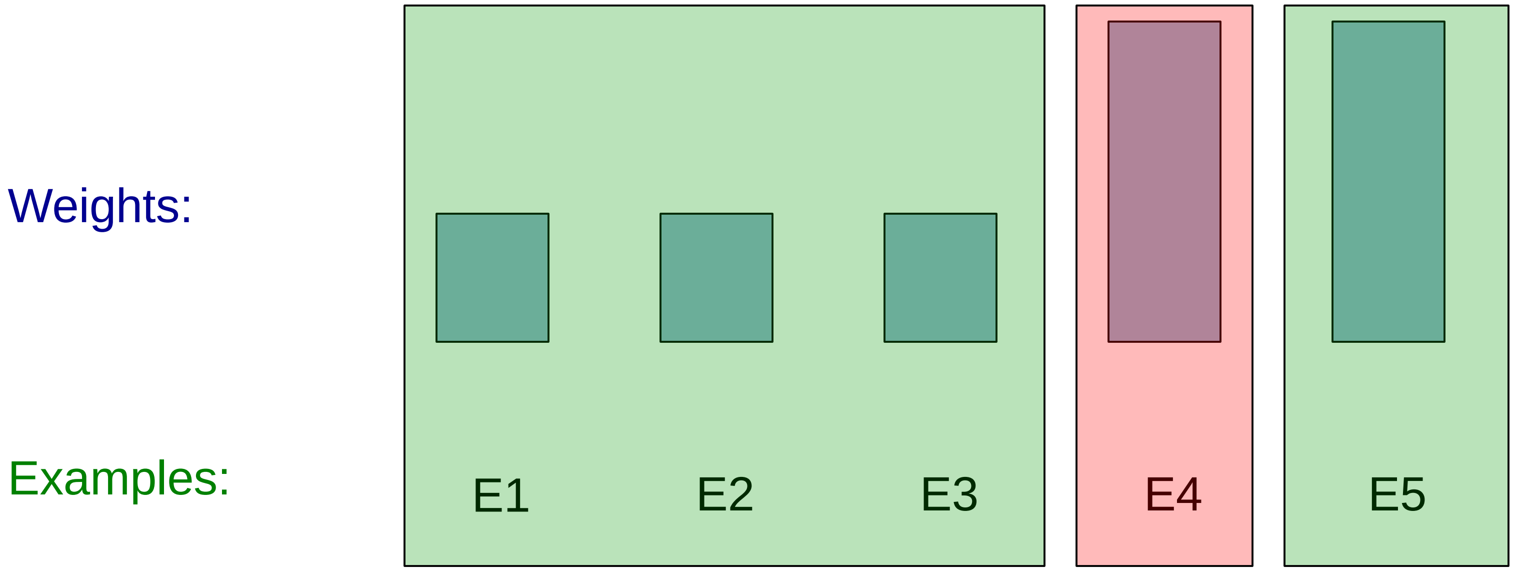
Weights:



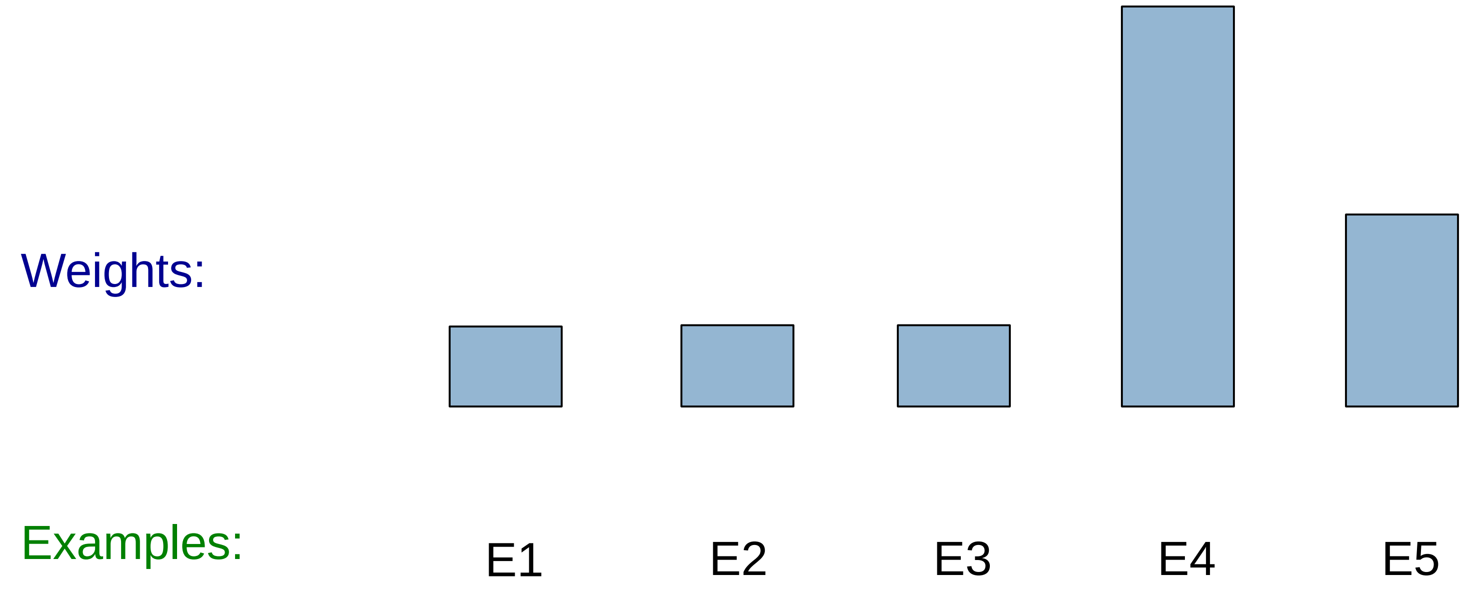
Examples: E1 E2 E3 E4 E5

Learn another weak classifier:

Boosting

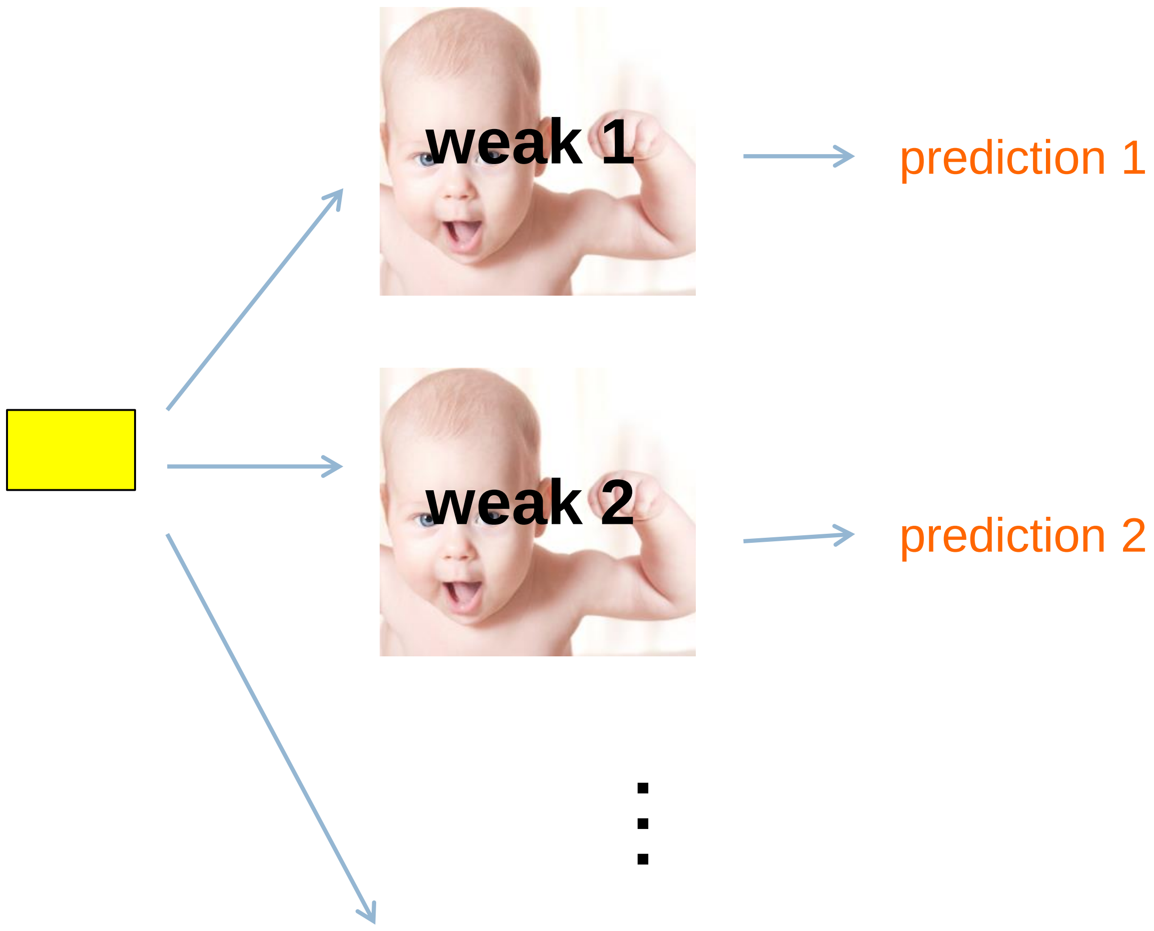


Boosting



- 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

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

- calculate "score" for this classifier:

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

- 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:

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

What does this say?

AdaBoost: train

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

weighted error for this classifier is:

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

What is the range of possible values?

AdaBoost: train

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

weighted error for this classifier is:

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

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

AdaBoost: train

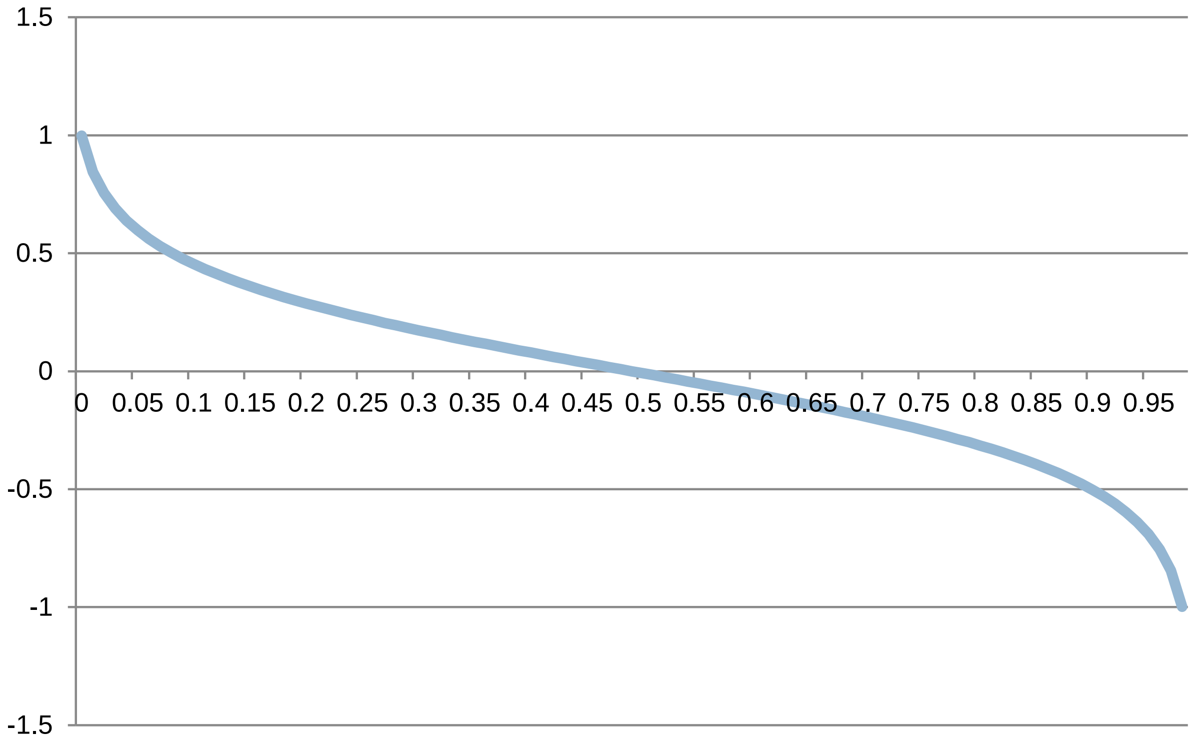
classifier$_k$ = learn a weak classifier based on weights

“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



$$a_k = \frac{1}{2}\log\left(\frac{1-e_i}{e_i}\right)$$

- 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 αvaries 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 αvaries from 1 to -1 training error)

We actually 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

$$w_i = \frac{1}{Z} w_i \underbrace{\exp\left(-\alpha_k * \underbrace{label_i * classifier_k(x_i)}_{\substack{\text{correct} \qquad \text{positive} \\ \text{incorrect} \quad \text{negative}}}\right)}_{\substack{\text{correct} \\ \text{incorrect}} \; ?}$$

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(-\alpha_k * \underbrace{\underbrace{label_i * classifier_k(x_i)}_{\text{correct positive; incorrect negative}}}_{\text{correct small value; incorrect large value}}\right)$$

correct positive
incorrect negative

correct value small
incorrect value large

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 αdo?

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 αdo?

If the classifier was good (<50% error)αis positive:
trust classifier output and move as normal
If the classifier was back (>50% error)αis negative
classifier is so bad, consider opposite prediction
of classifier

AdaBoost: train

update the example weights

$$w_i = \frac{1}{Z} w_i \exp\left(- \underbrace{\alpha_k * \underbrace{label_i * classifier_k(x_i)}_{\text{correct positive / incorrect negative}}}_{\text{correct small value / incorrect large value}}\right)$$

correct positive
incorrect negative

correct value small
incorrect value large

If the classifier was good (<50% error)αis positive
If the classifier was back (>50% error)αis negative

AdaBoost justification

update the example weights

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

Does this look like anything we’ve seen before?

AdaBoost justification

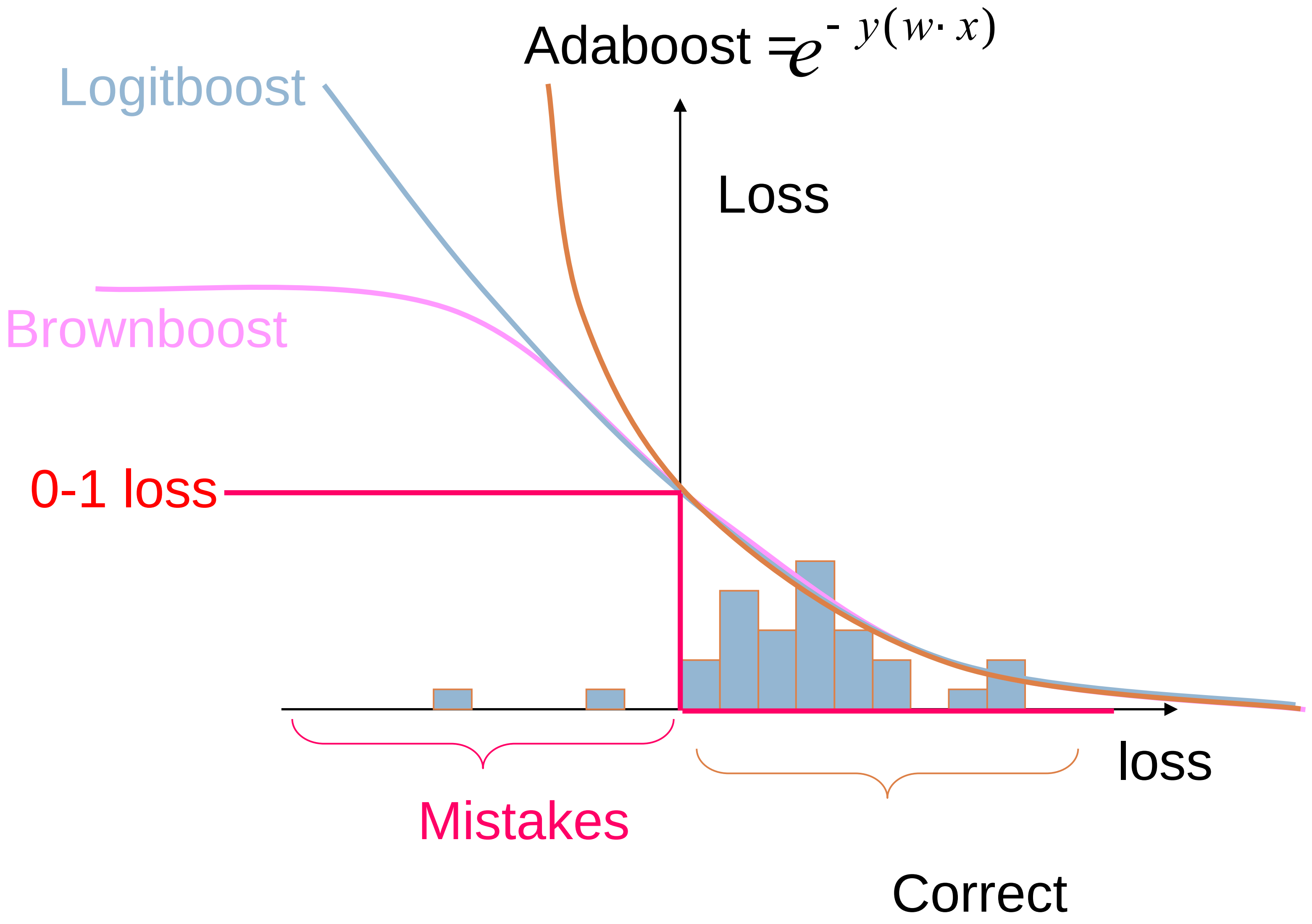
update the example weights

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

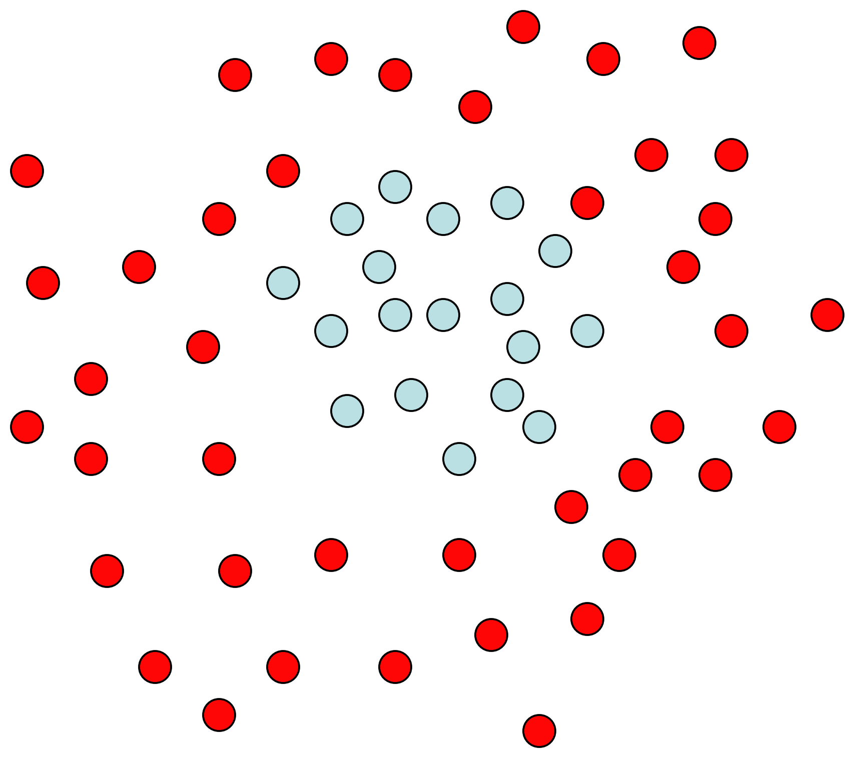
$$l(y, y') = \exp(-yy')$$

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

Other boosting variants

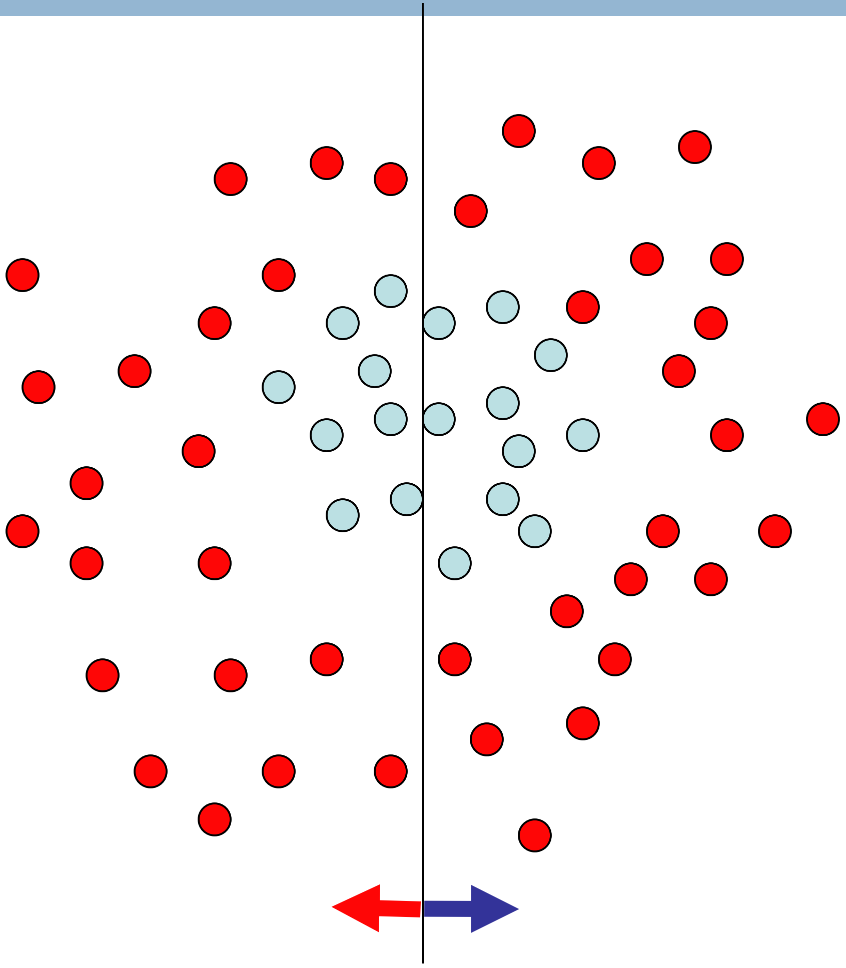


Boosting example



Start with equal weighted data set

Boosting example

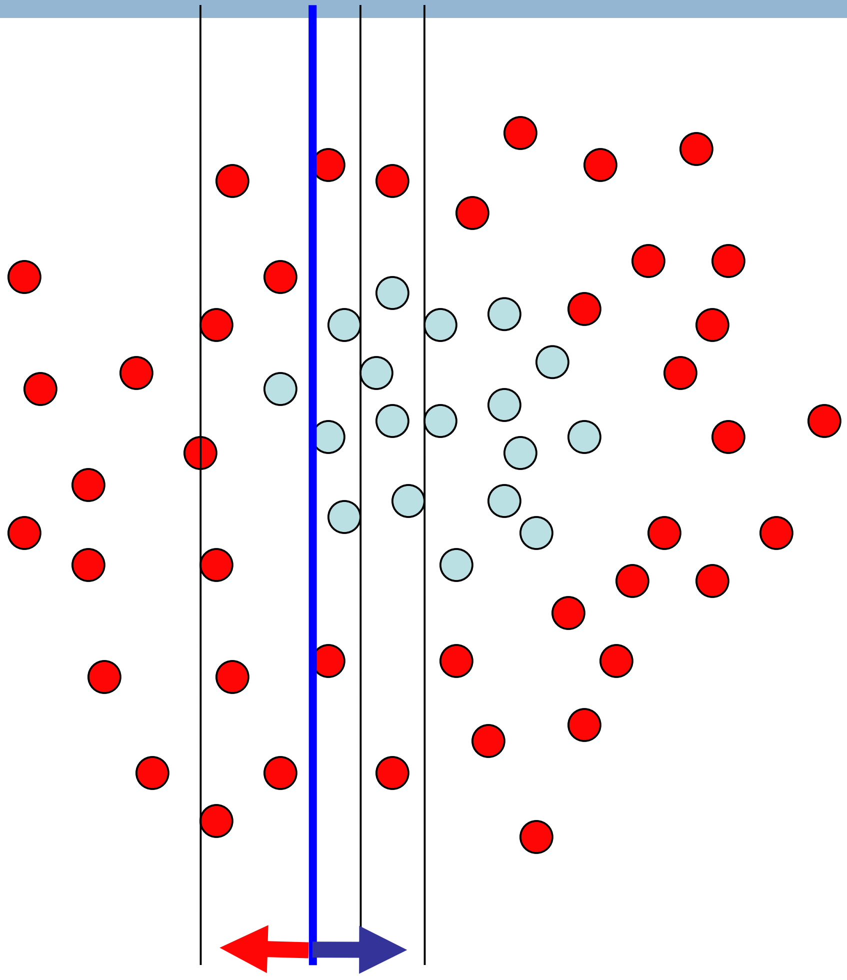


weak learner = line

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

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

Boosting example

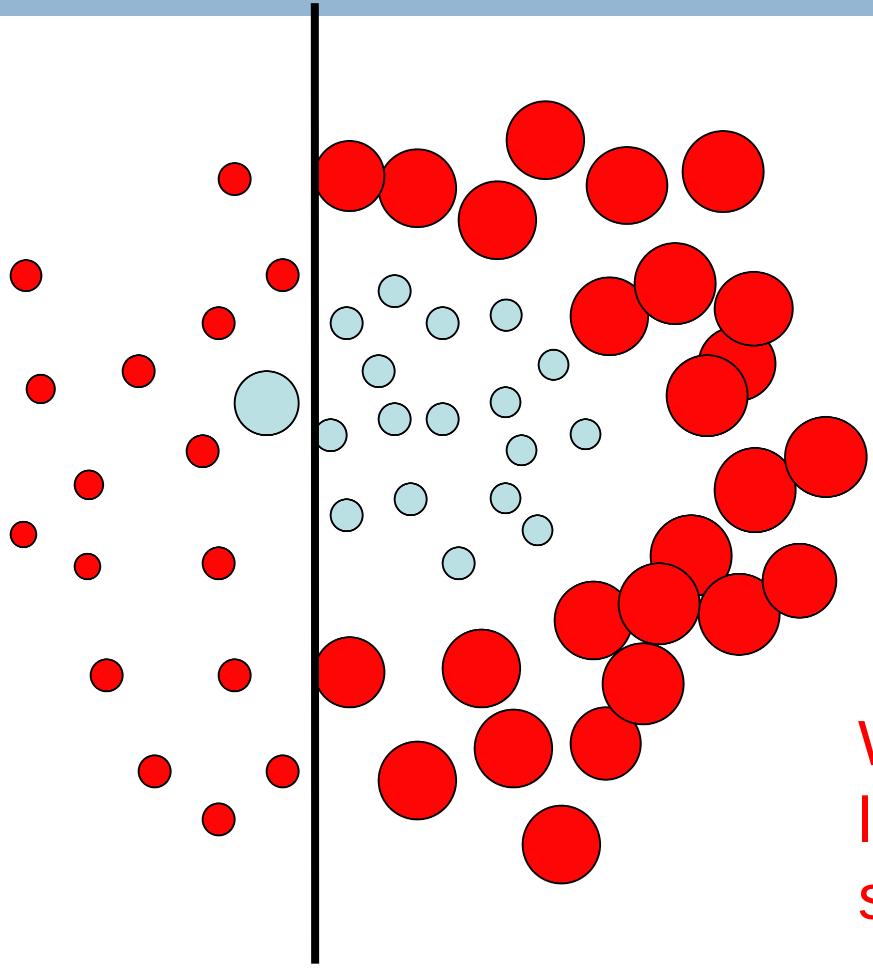


How should we reweight examples?

This one seems to be the best

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

Boosting example

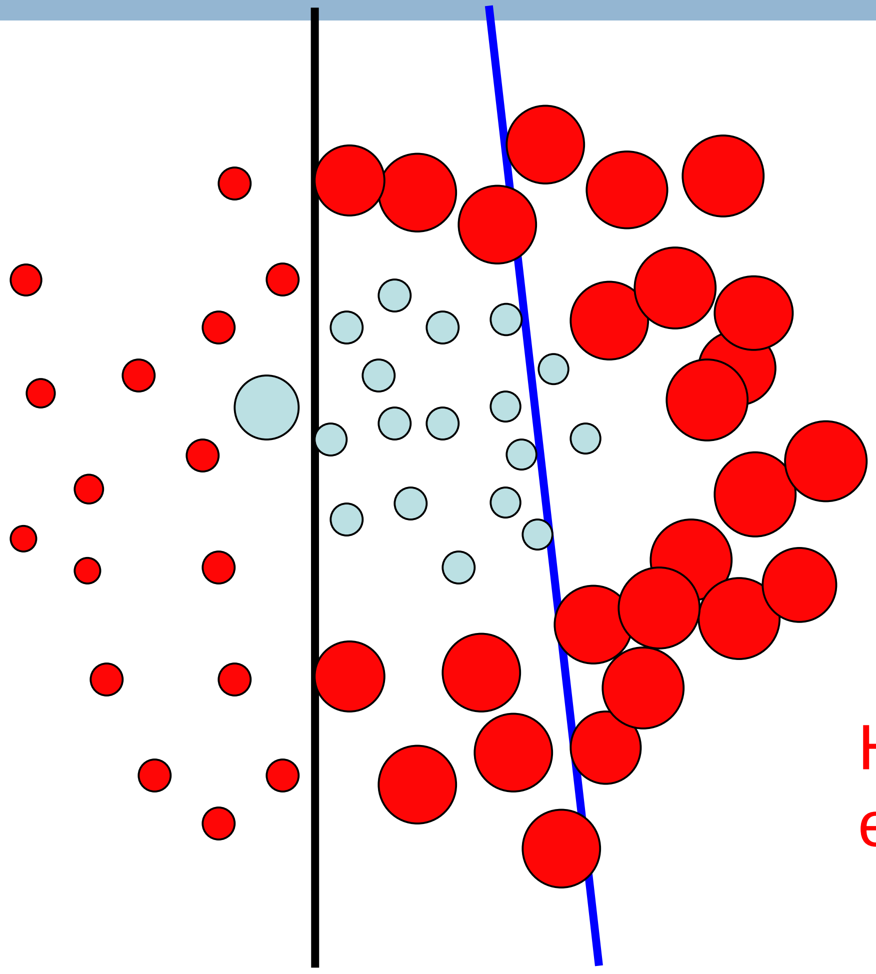


What would be the best line learned on this data set?

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

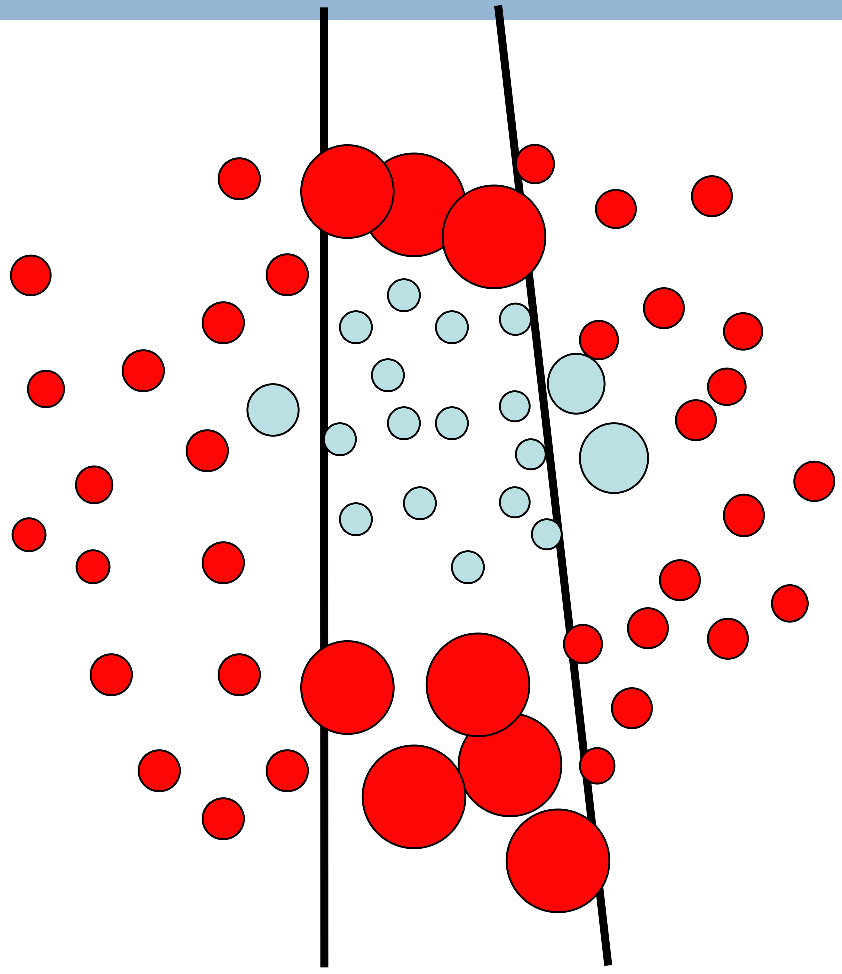
reds on this side get more weight
blues on this side get less weight

Boosting example



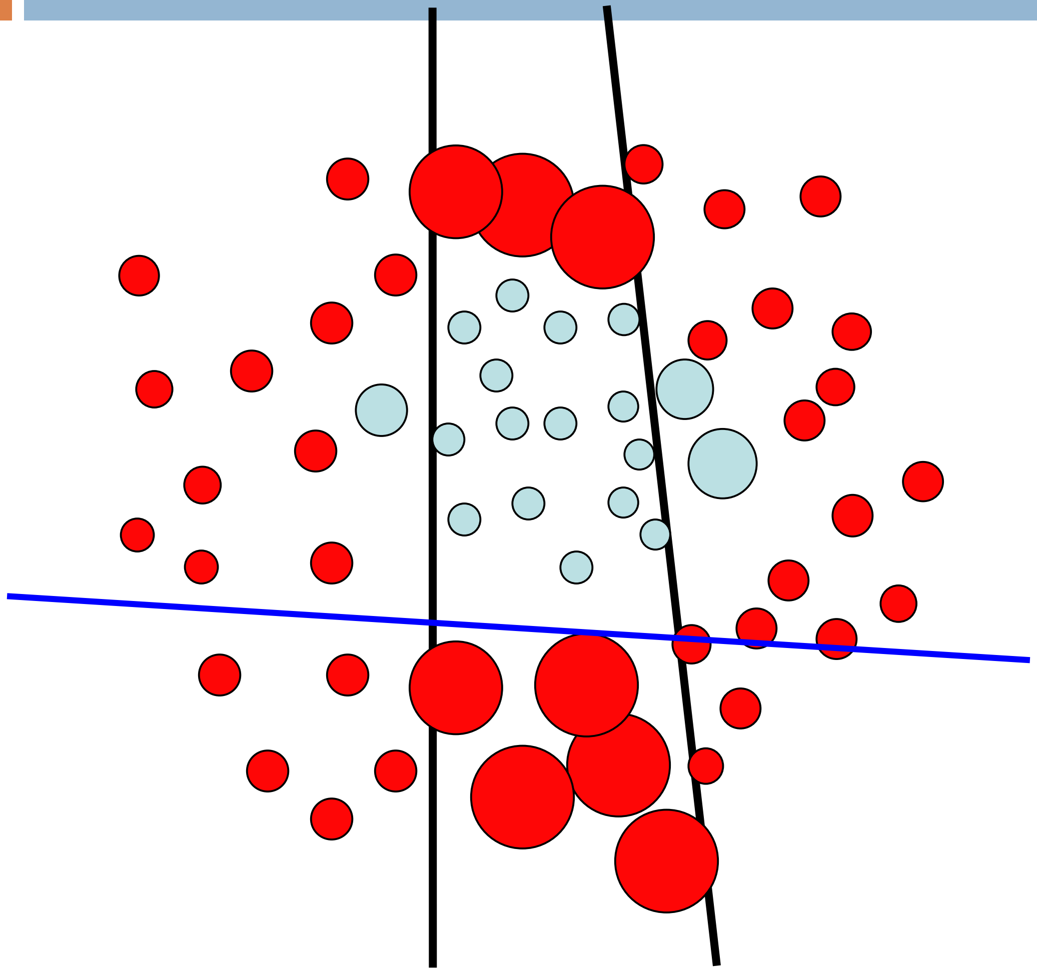
How should we reweight examples?

Boosting example

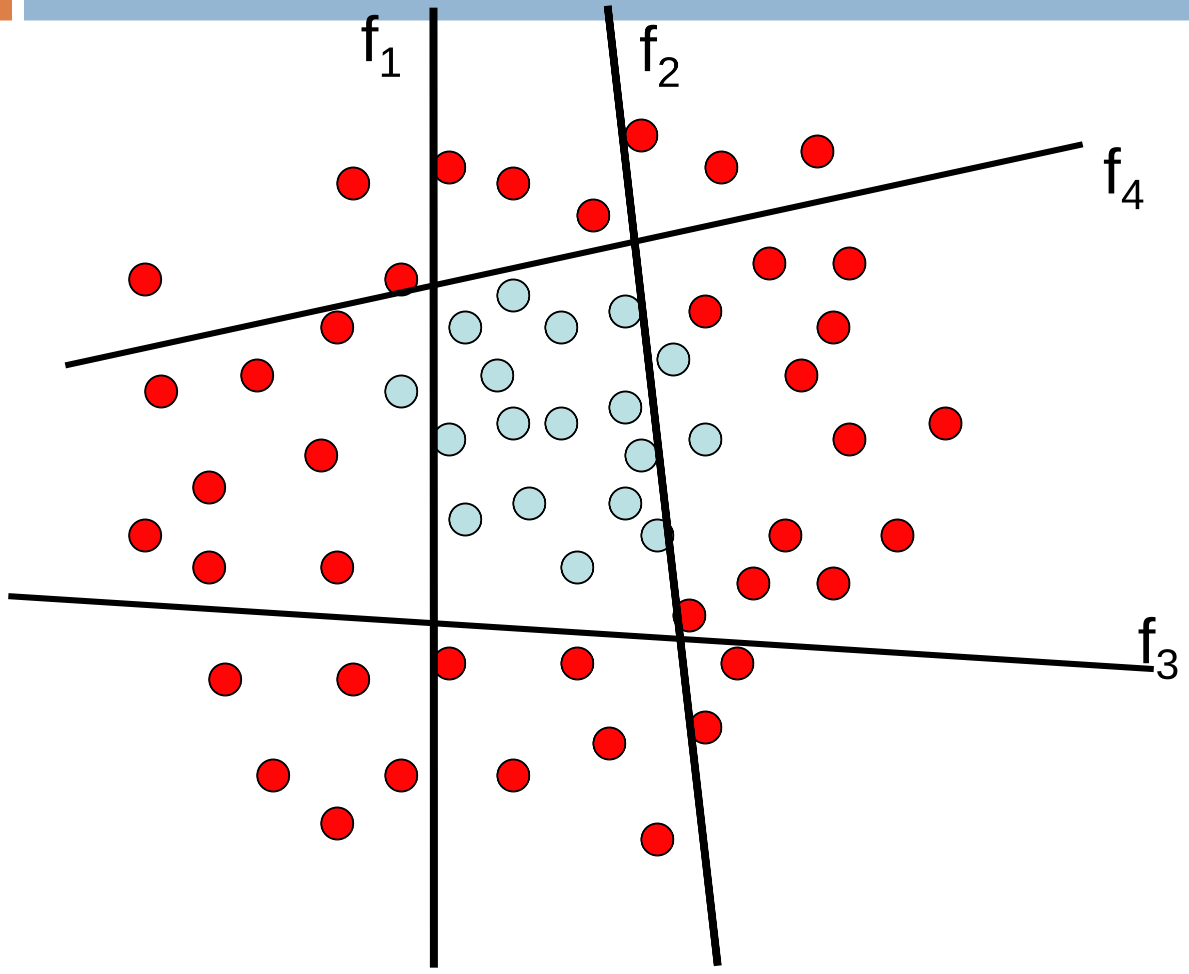


What would be the best line learned on this data set?

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:

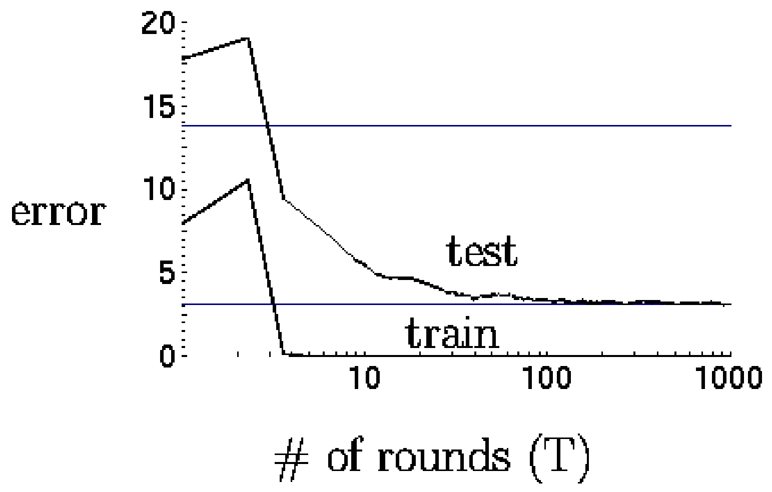
- can use a shallow (2-3 level tree)
- even more common is a 1-level tree
  - called a decision stump ☺
  - asks a question about a single feature

- 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

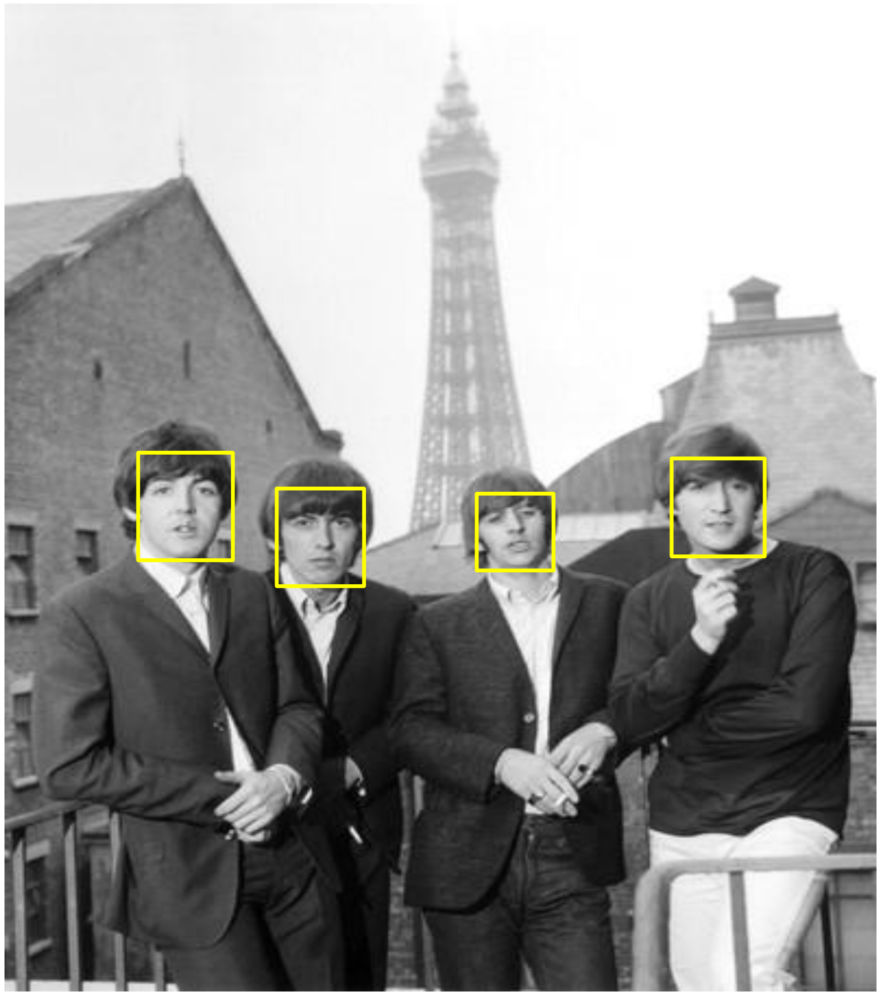
One of the keys is that boosting tends not to overfit, even for a large number of iterations



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

Adaboost application example: face detection

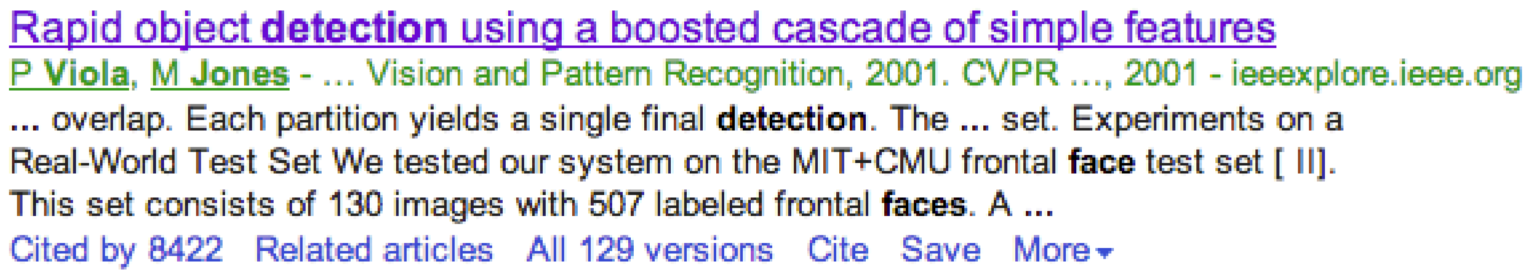
Adaboost application example: face detection



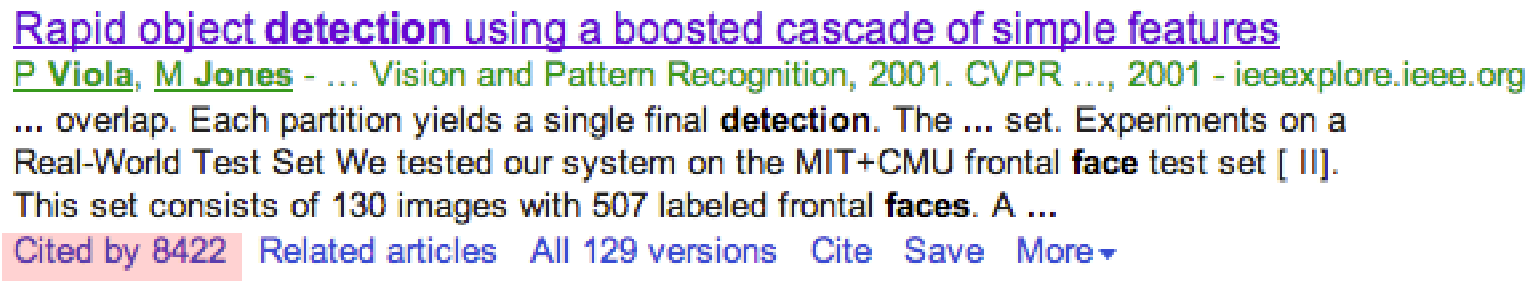
Rapid Object Detection using a Boosted Cascade of Simple Features

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

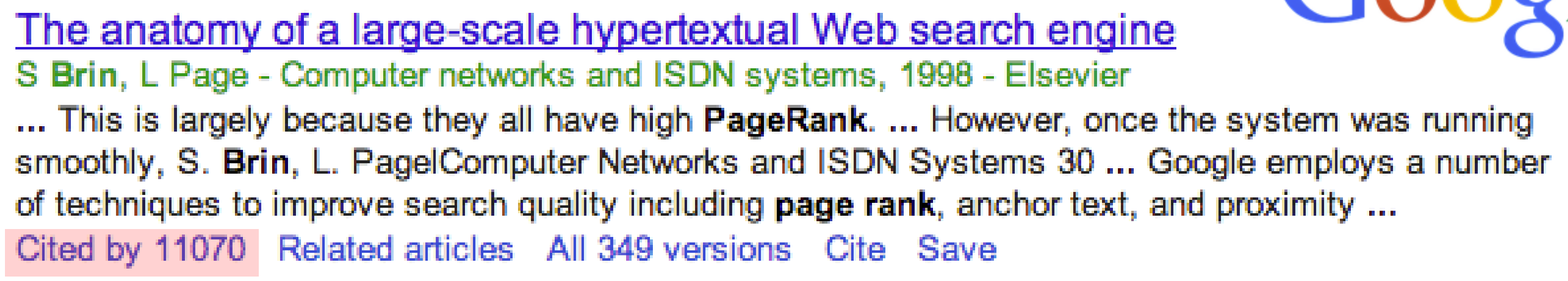
Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142



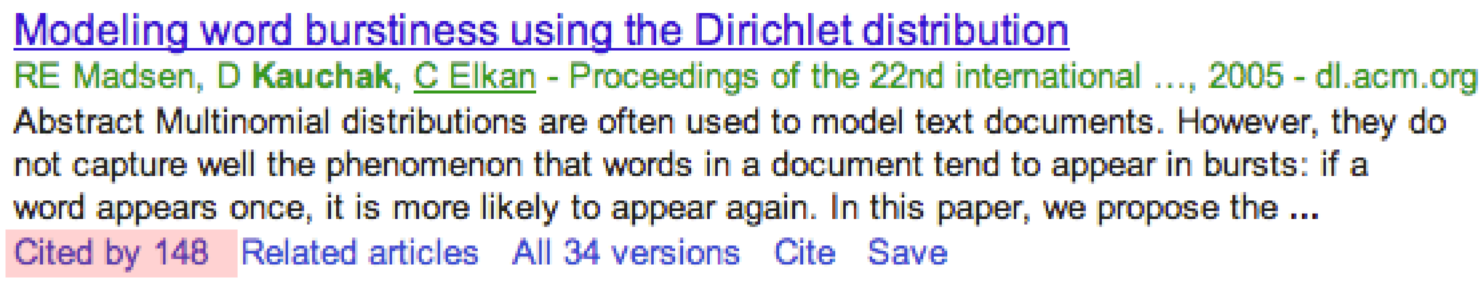
Rapid object **detection** using a boosted cascade of simple features
P **Viola**, M **Jones** - … Vision and Pattern Recognition, 2001. CVPR …, 2001 - ieeexplore.ieee.org
… overlap. Each partition yields a single final **detection**. The **…** set. Experiments on a Real-World Test Set We tested our 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


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

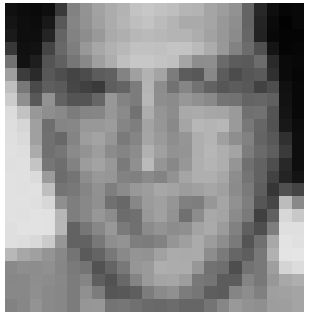
To give you some context of importance:


The anatomy of a large-scale hypertextual Web search engine
S **Brin**, L Page - Computer networks and ISDN systems, 1998 - Elsevier
… This is largely because they all have high **PageRank**. … However, once the system was running smoothly, S. **Brin**, L. PagelComputer Networks and ISDN Systems 30 … Google employs a number of techniques to improve search quality including **page rank**, anchor text, and proximity …
Cited by 11070 Related articles All 349 versions Cite Save

or:

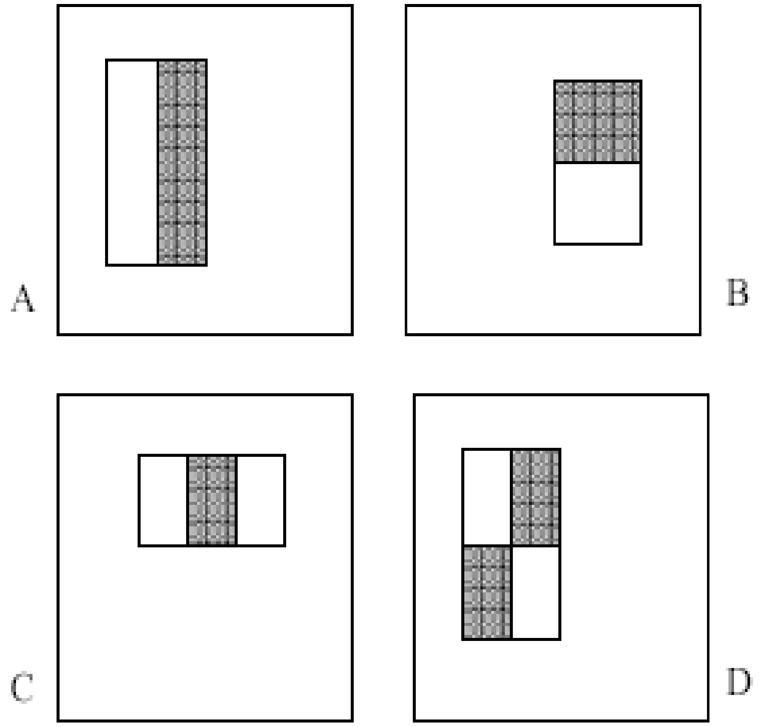

Modeling word burstiness using the Dirichlet distribution
RE Madsen, D **Kauchak**, C Elkan - Proceedings of the 22nd international …, 2005 - dl.acm.org
Abstract Multinomial distributions are often used to model text documents. However, they do not capture well the phenomenon that words in a document tend to appear in bursts: if a word appears once, it is more likely to appear again. In this paper, we propose the …
Cited by 148 Related articles All 34 versions Cite Save

“weak” learners



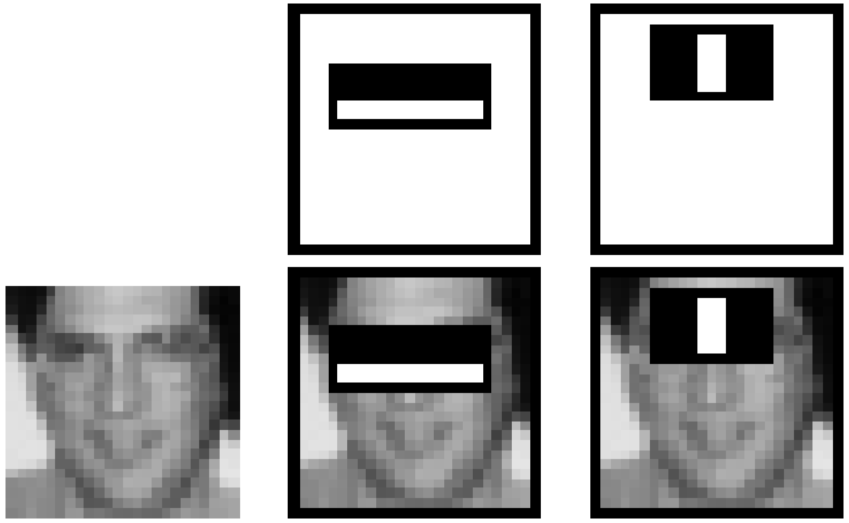
4 Types of “Rectangle filters”
(Similar to Haar wavelets
 Papageorgiou, et al.)

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



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

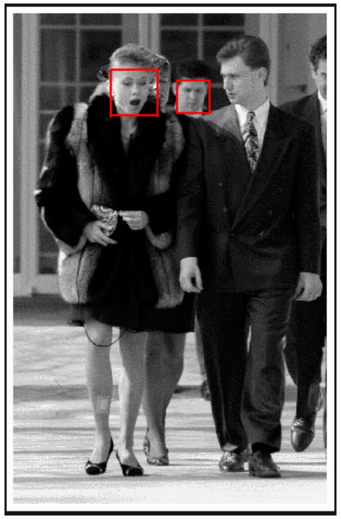
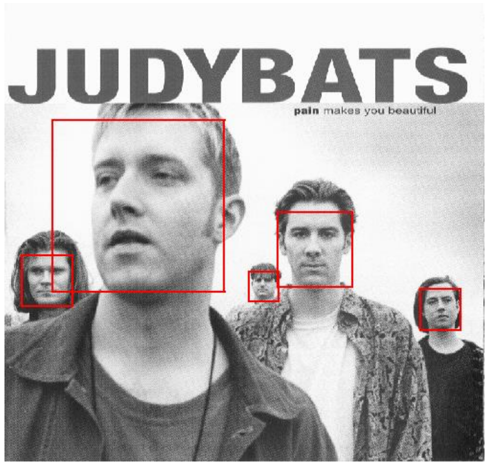
“weak” learners

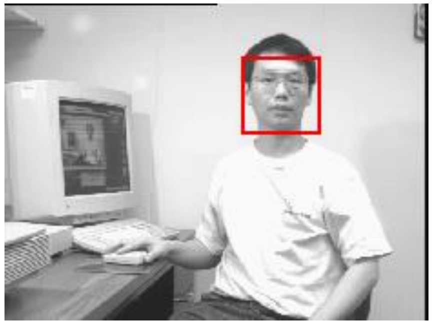
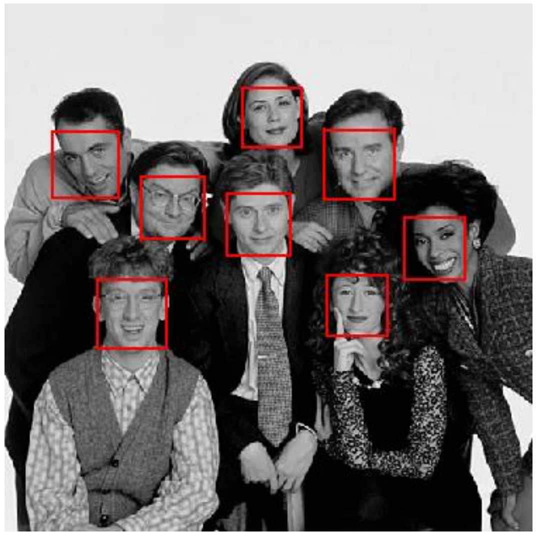
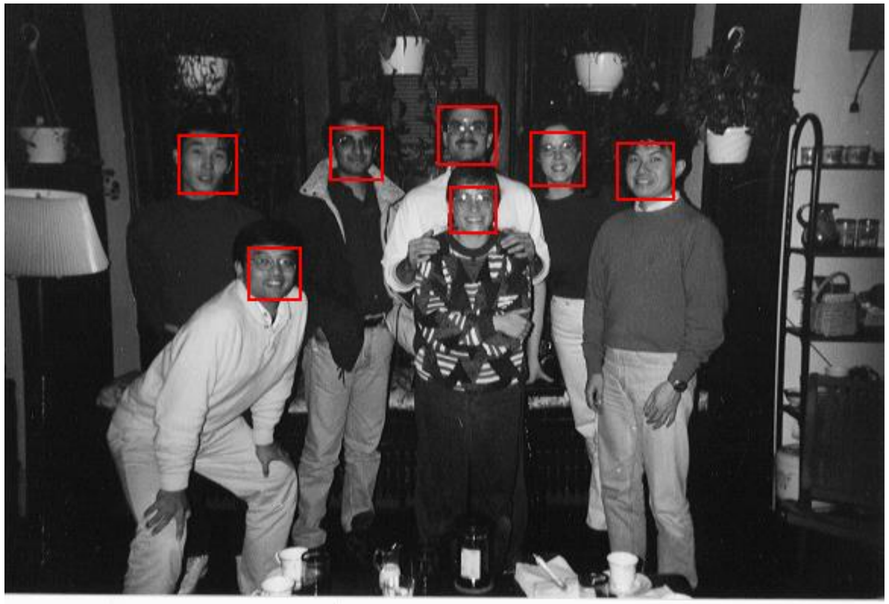


$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \ldots$$

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

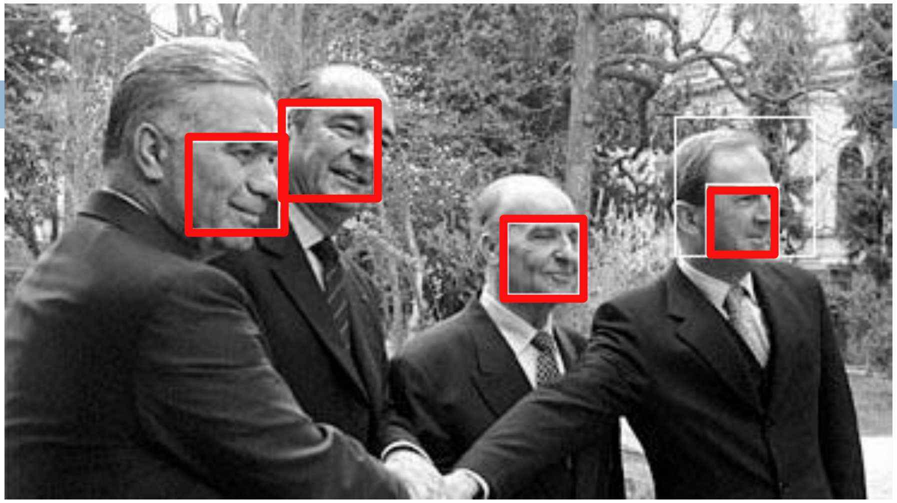
Example output

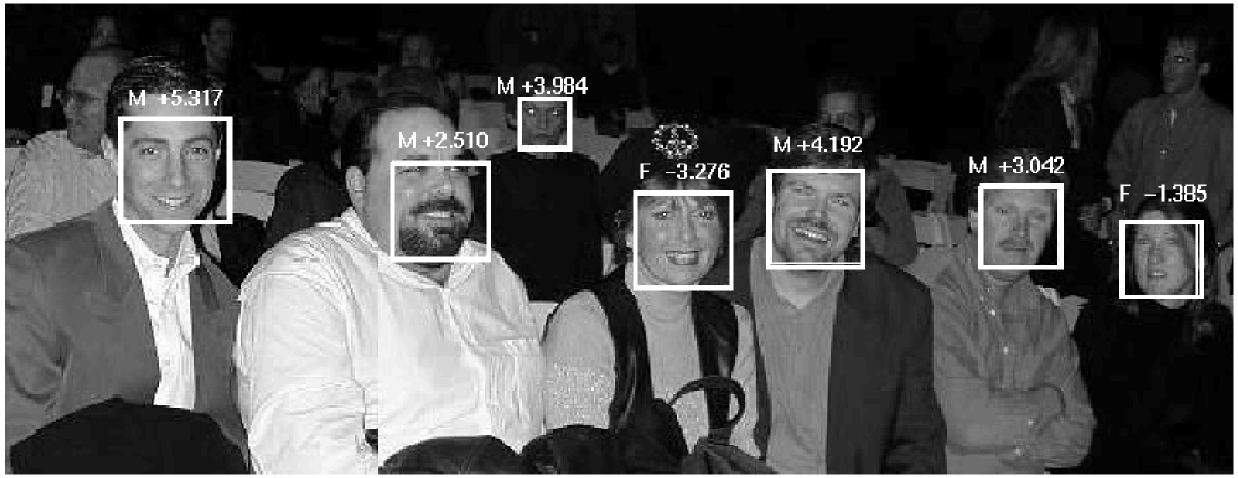
Solving other "Face" Tasks

Facial Feature Localization

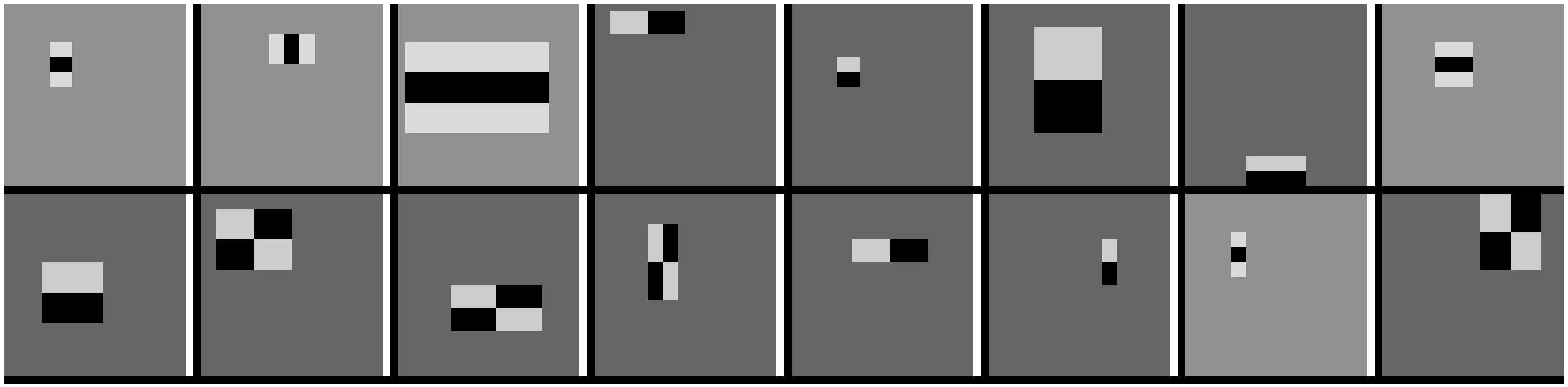


Profile Detection

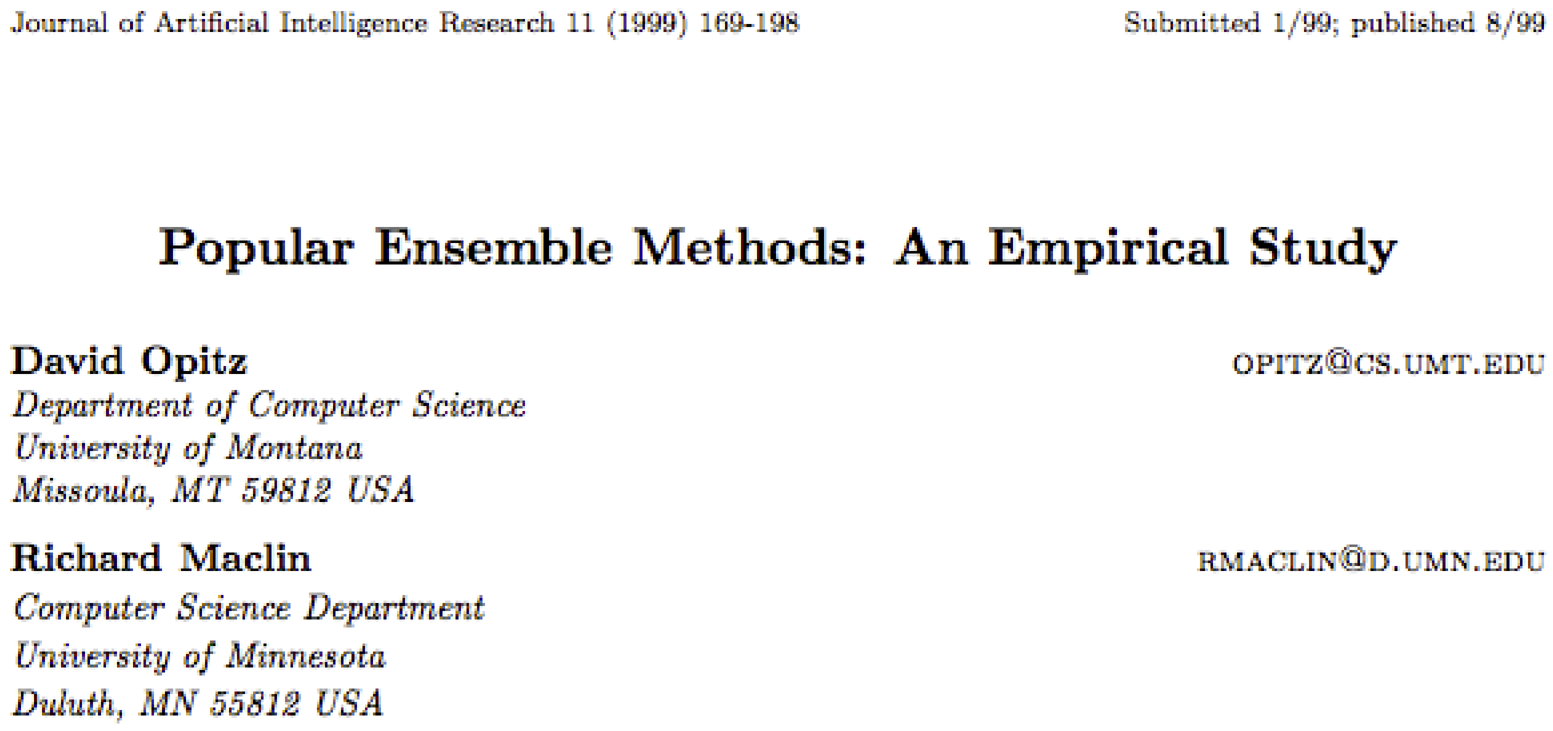
Demographic Analysis



“weak” classifiers learned



Bagging vs Boosting


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

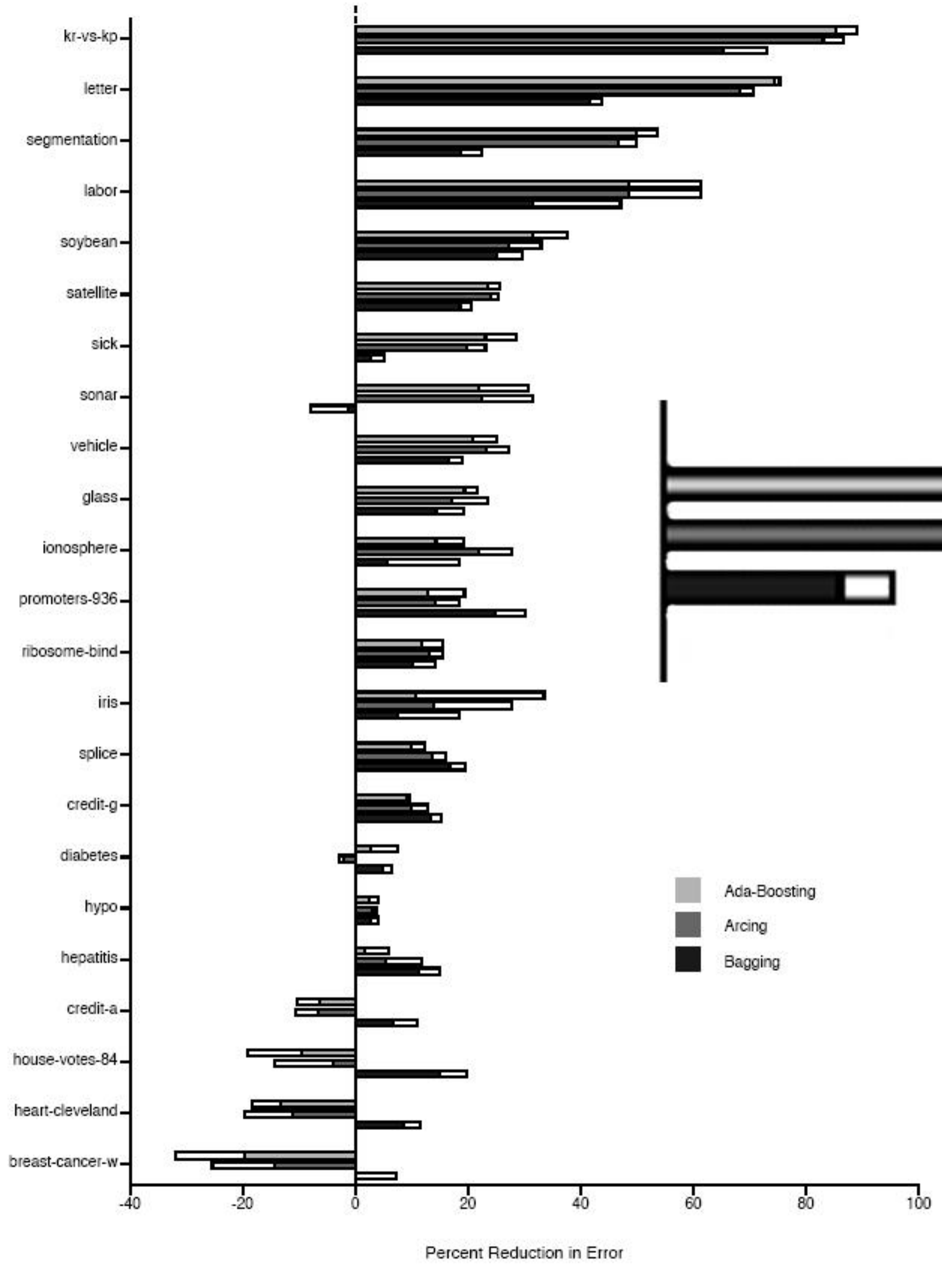
Popular Ensemble Methods: An Empirical Study

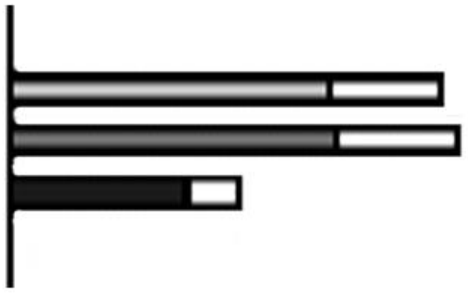
David Opitz OPITZ@CS.UMT.EDU
Department of Computer Science
University of Montana
Missoula, MT 59812 USA

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



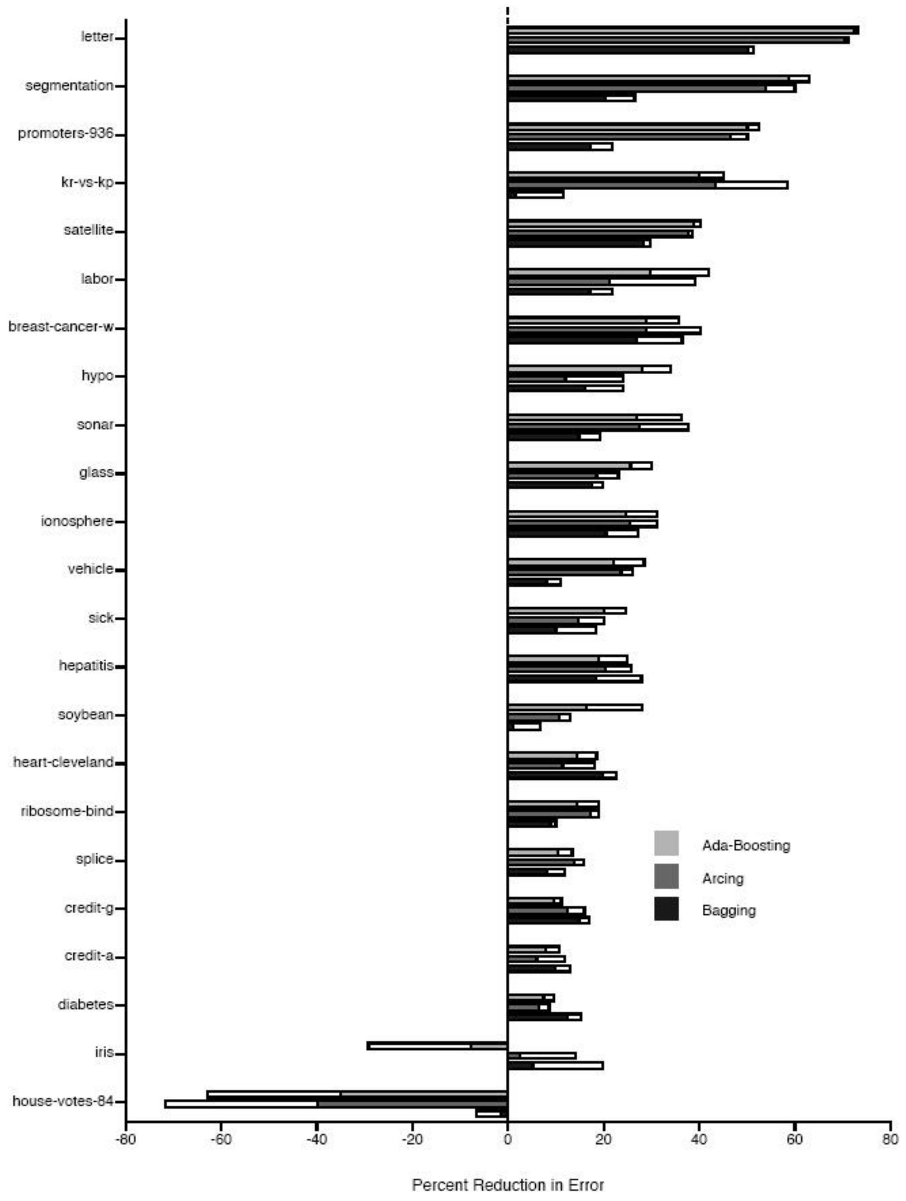


Change in error rate over standard classifier

Ada-Boosting
Arcing
Bagging

White bar represents 1 standard deviation

Boosting Decision Trees



Useful Videos

Extending Machine Learning Algorithms – AdaBoost Classifier > https://youtu.be/BoGNyWW9-mE

Ensembles (4): AdaBoost > https://youtu.be/ix6IvwbVpw0

Principles of Machine Learning | AdaBoost > https://youtu.be/-DUxtdeCiB4