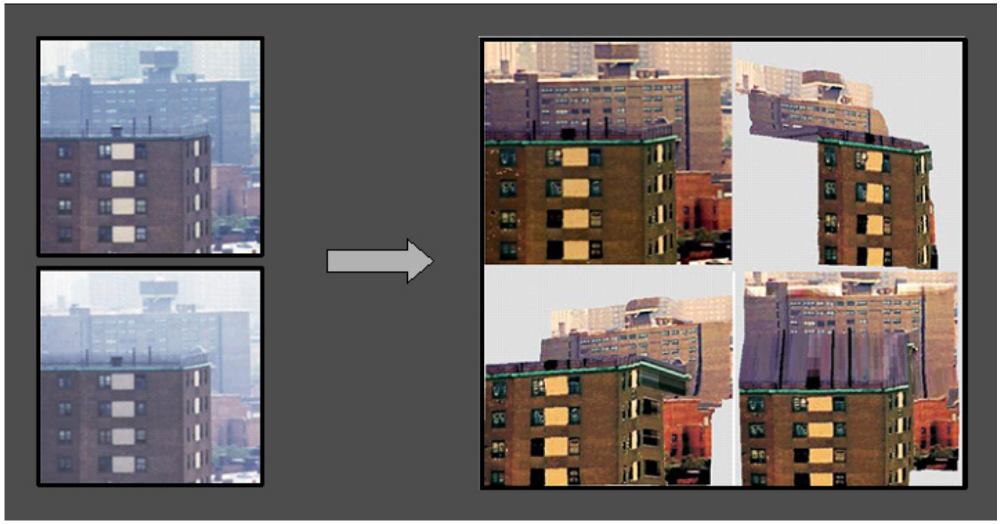
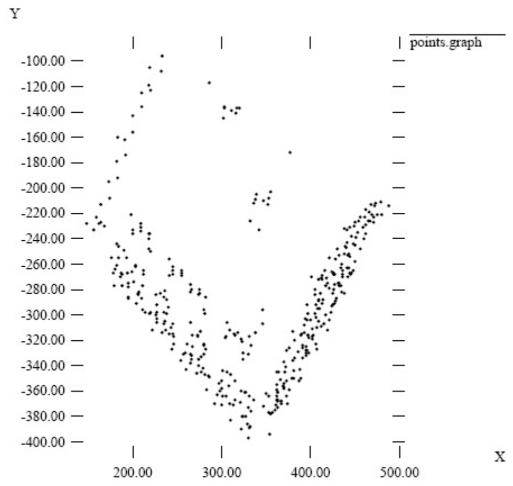
Recognition
Part I

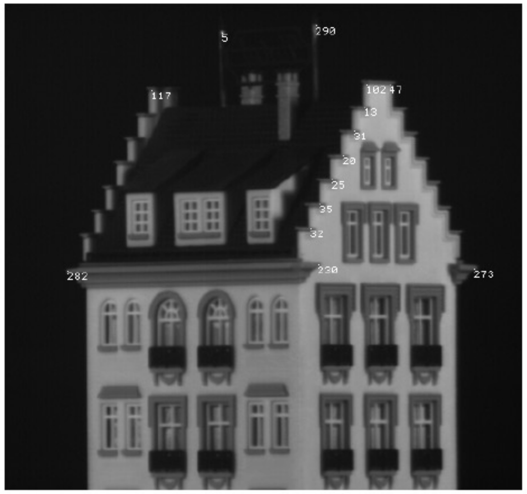
CSE 576

What we have seen so far: Vision as Measurement Device

Real-time stereo on Mars


Physics-based Vision





Structure from Motion

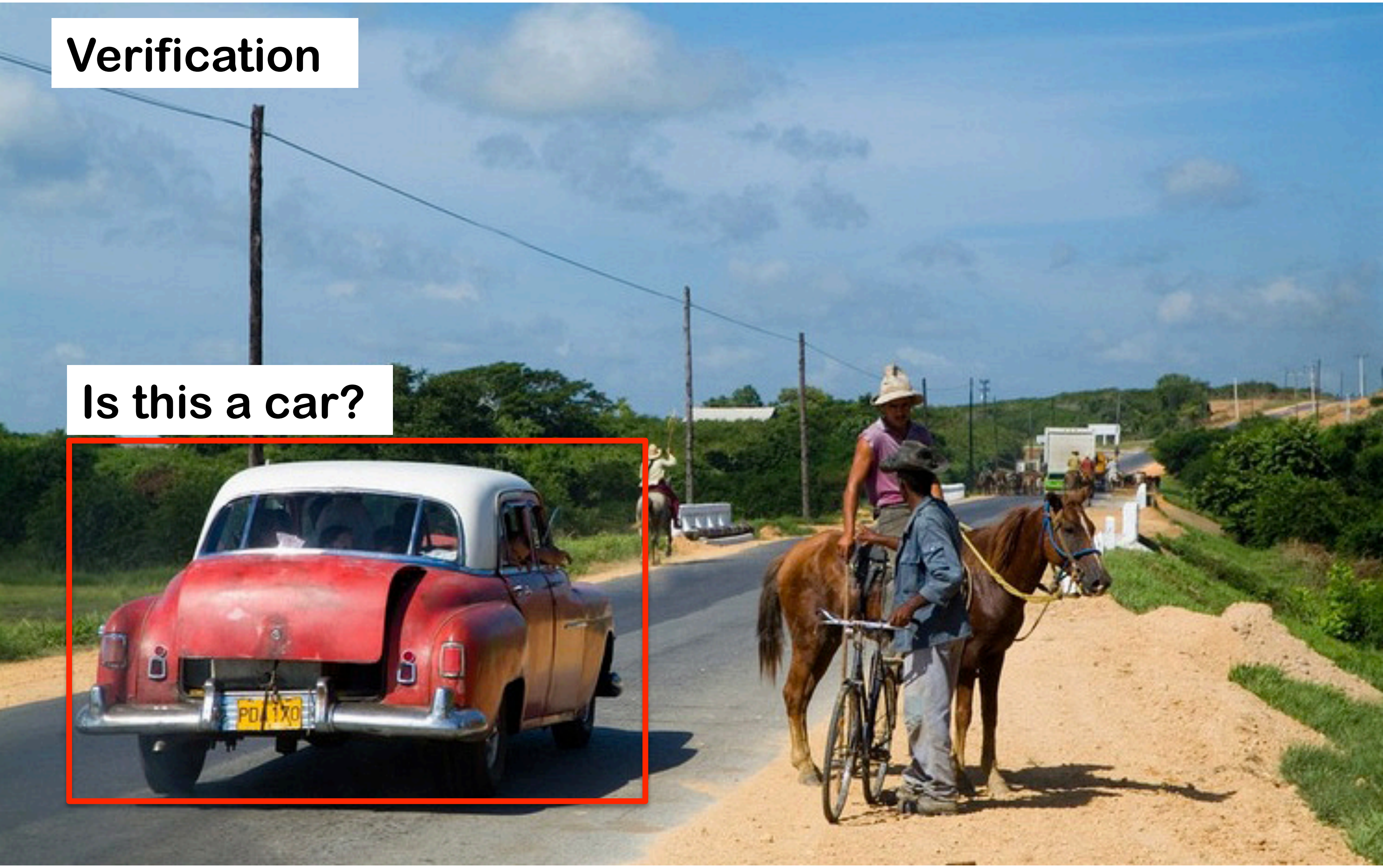
Virtualized Reality

Slide Credit: Alyosha

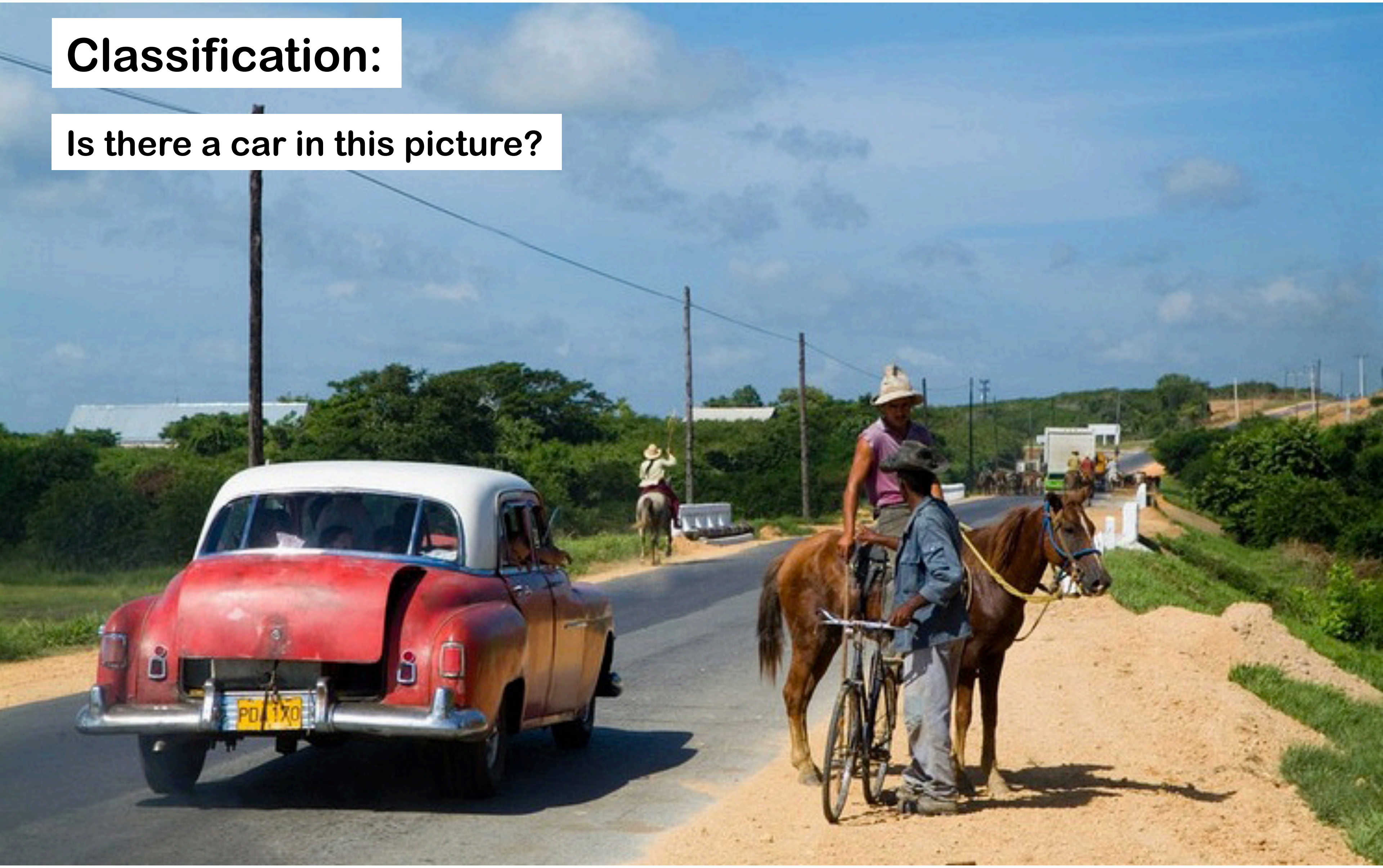
Visual Recognition

- What does it mean to “see”?
    - “What” is “where”, Marr 1982
- Get computers to “see”

Visual Recognition



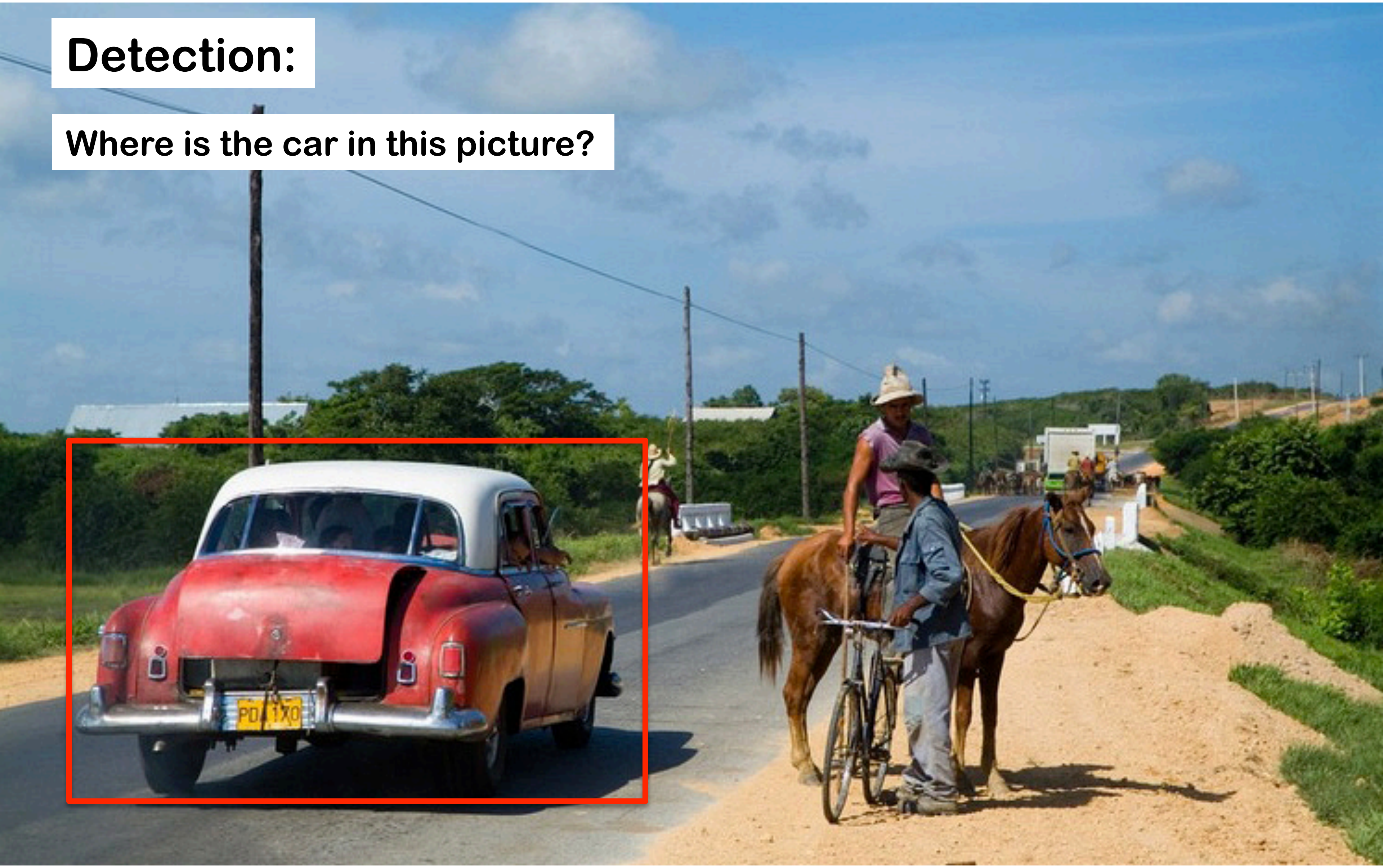
Visual Recognition



Classification:

Is there a car in this picture?

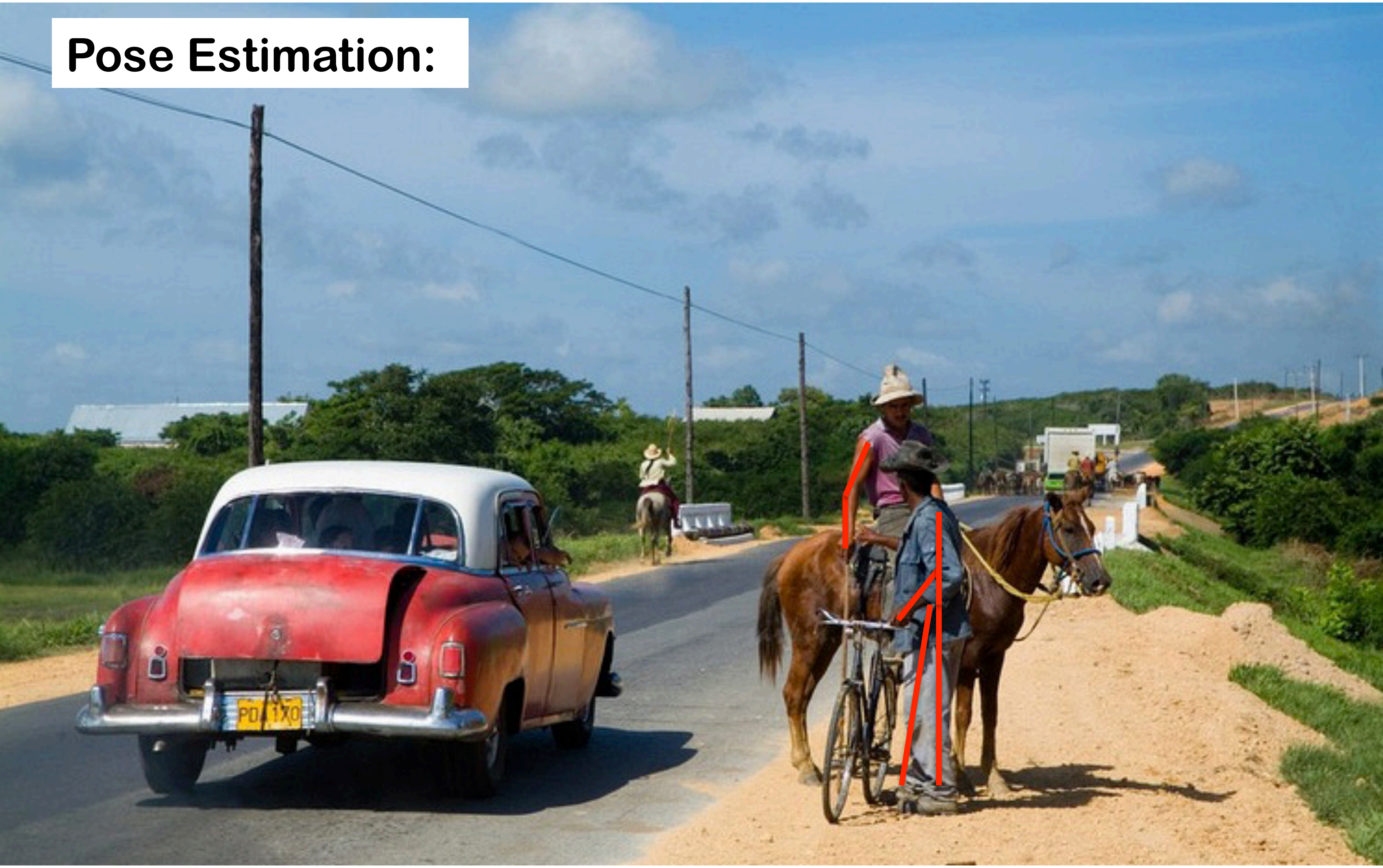
Visual Recognition



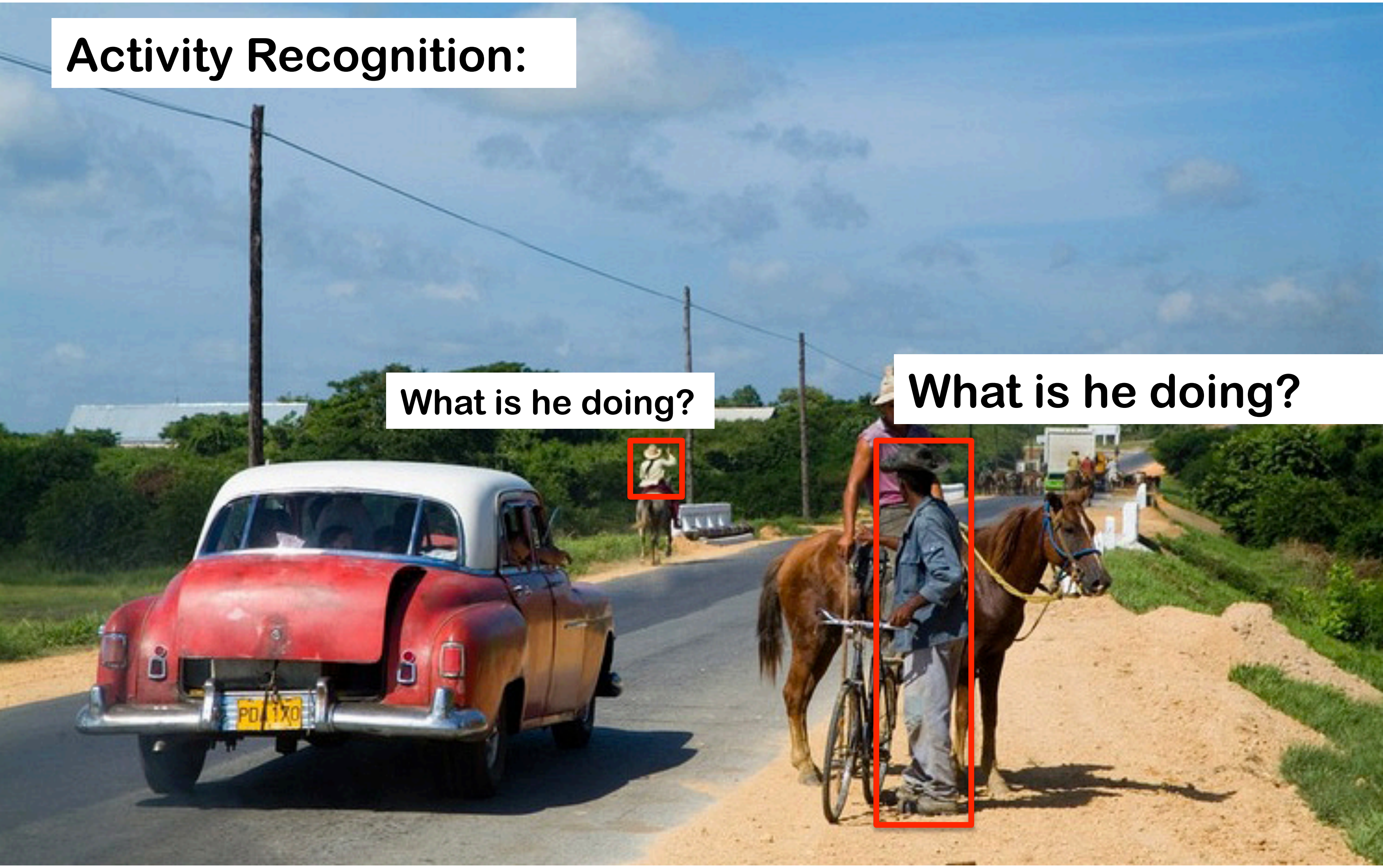
Detection:

Where is the car in this picture?

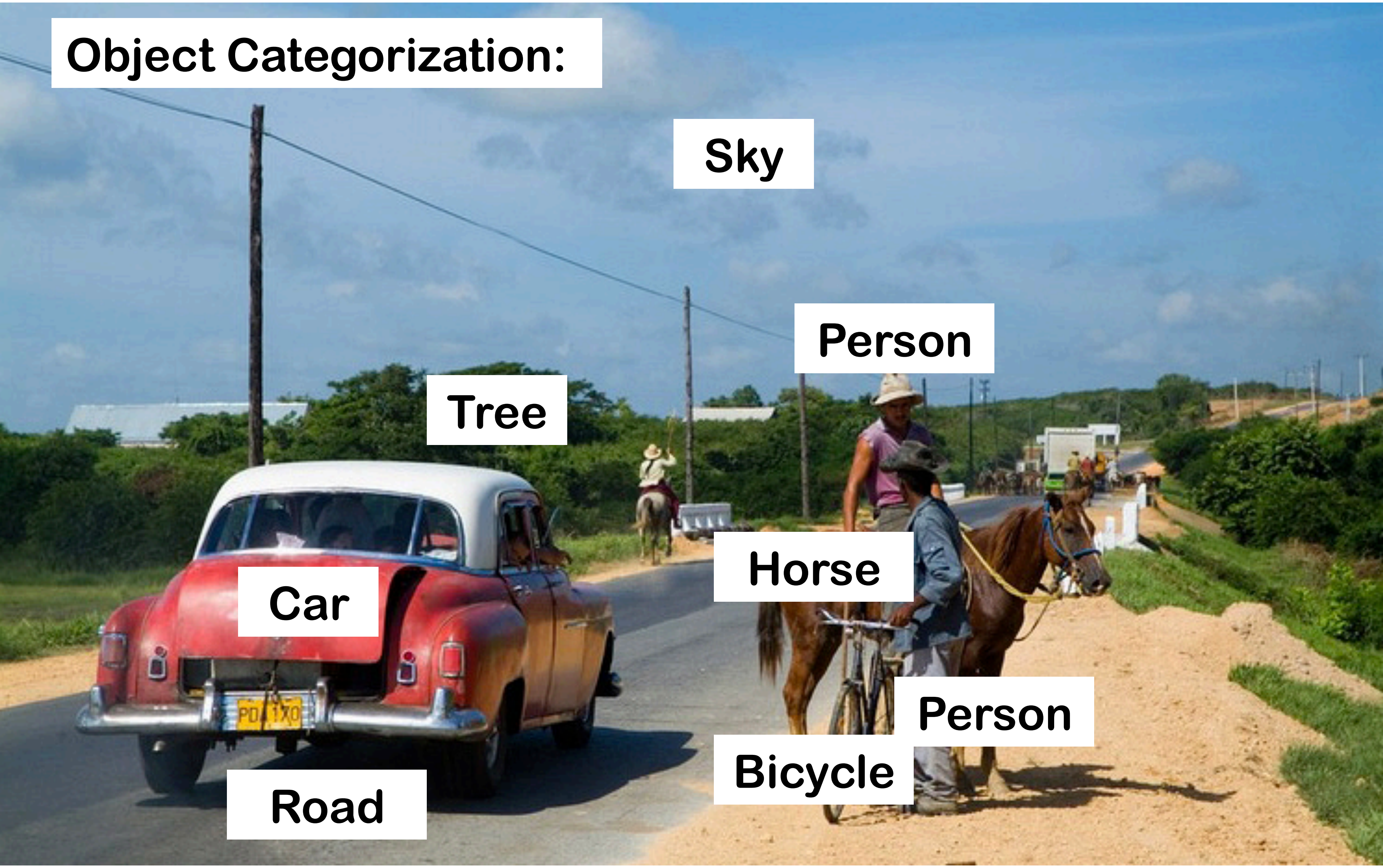
Visual Recognition



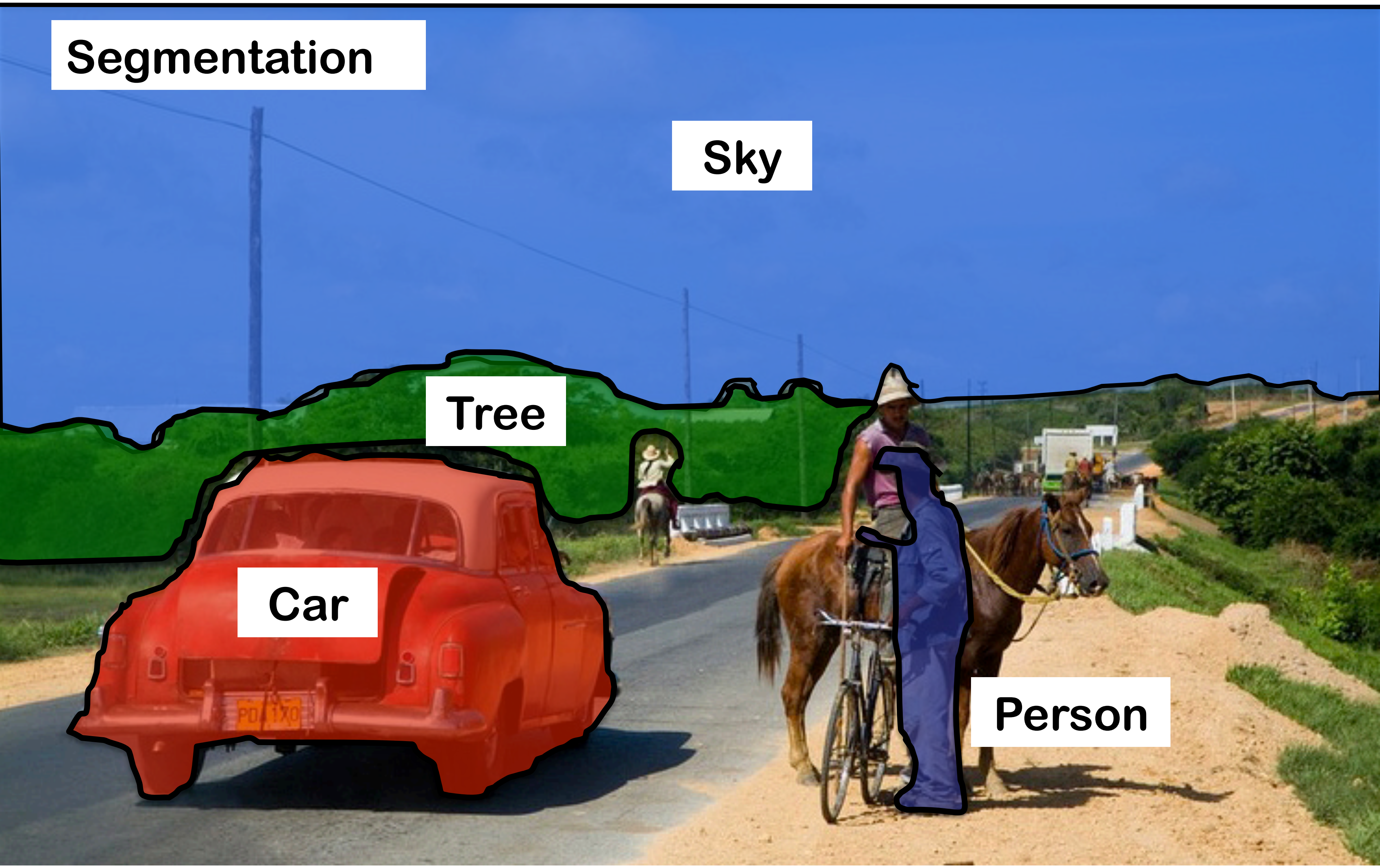
Visual Recognition



Visual Recognition

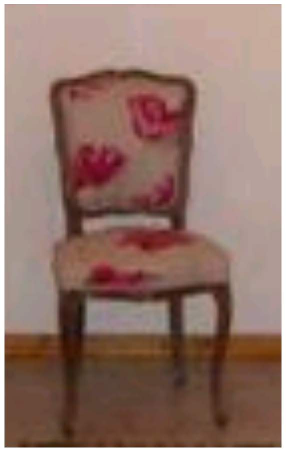


Visual Recognition

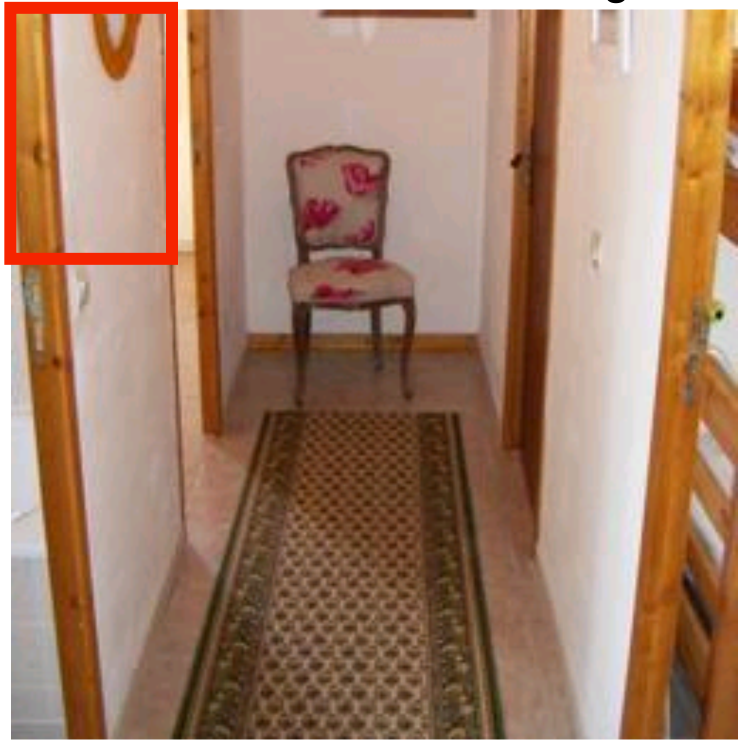


Object recognition
Is it really so hard?

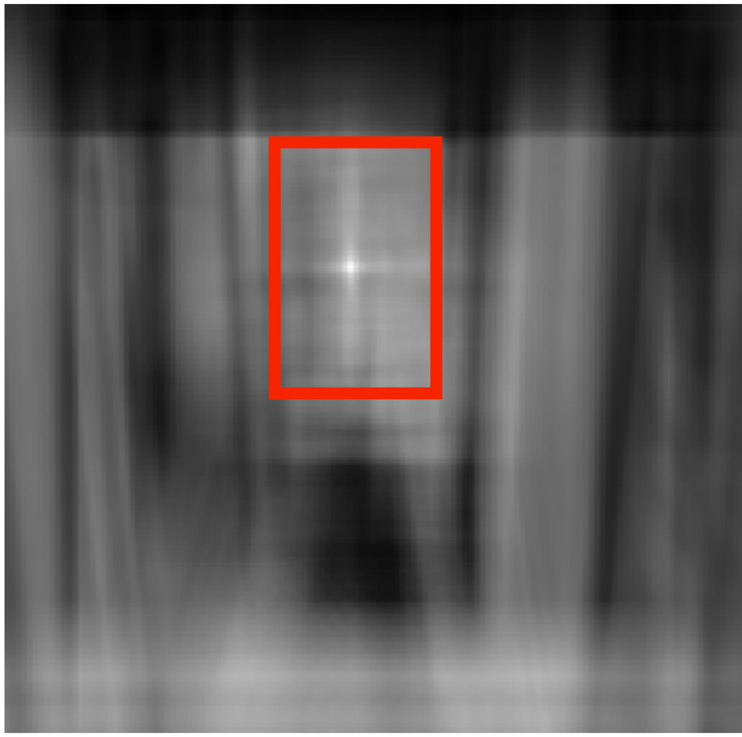
This is a chair

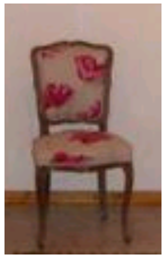


Find the chair in this image



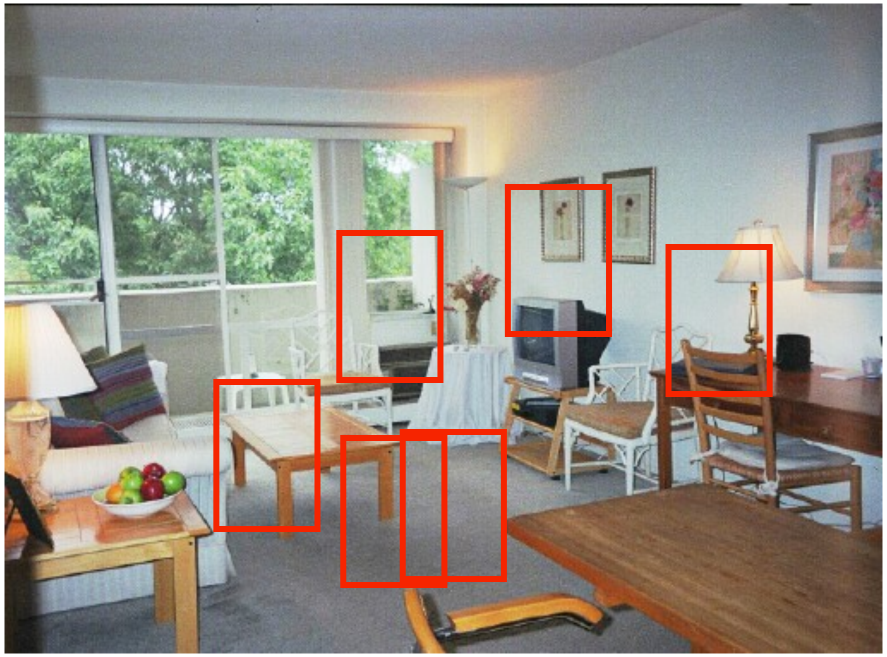
Output of normalized correlation

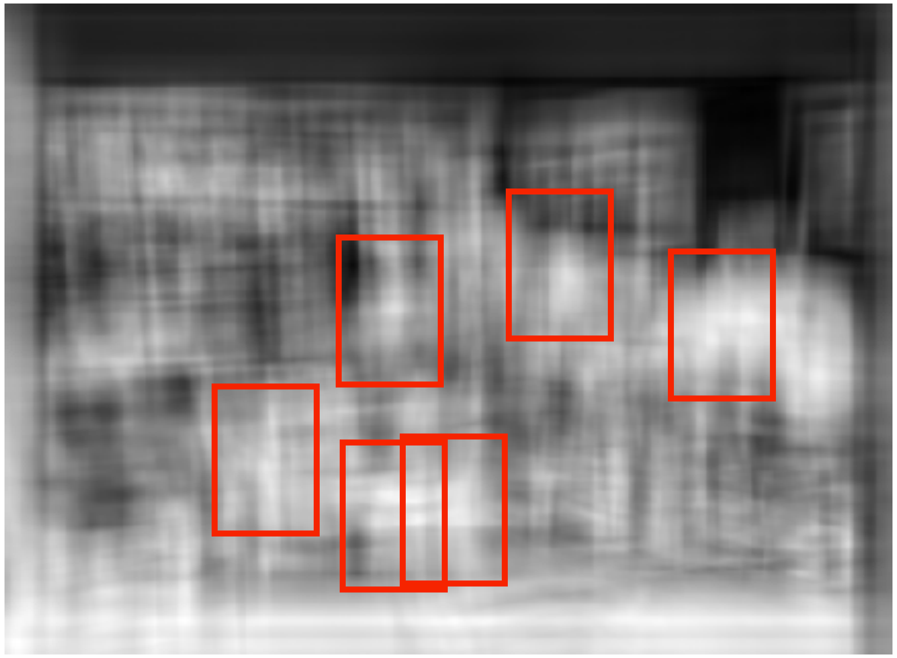




Object recognition
Is it really so hard?

Find the chair in this image





Pretty much garbage

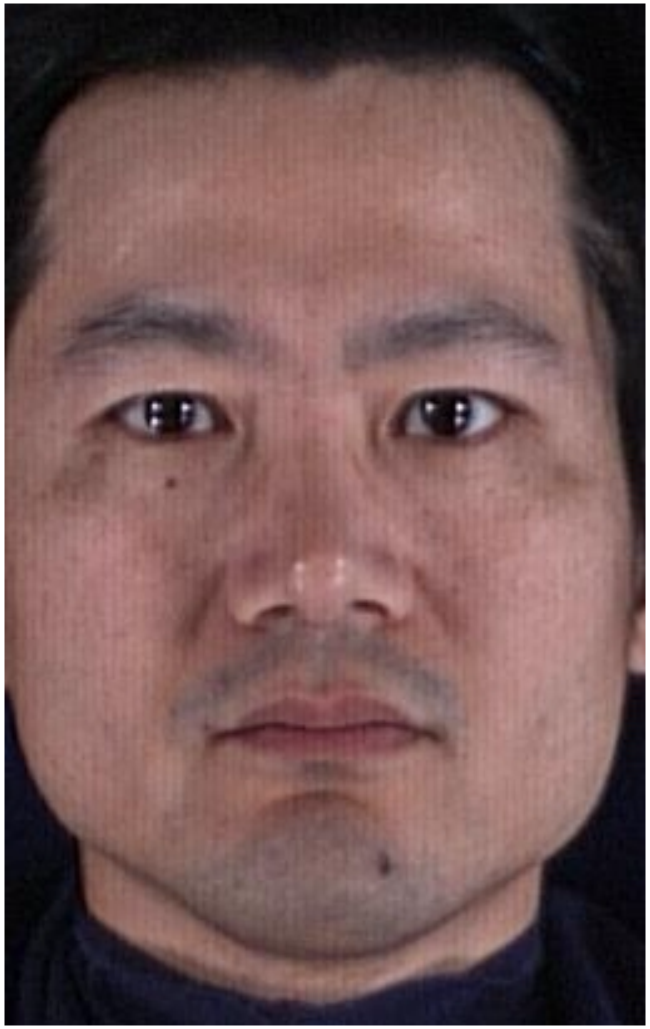
Simple template matching is not going to make it

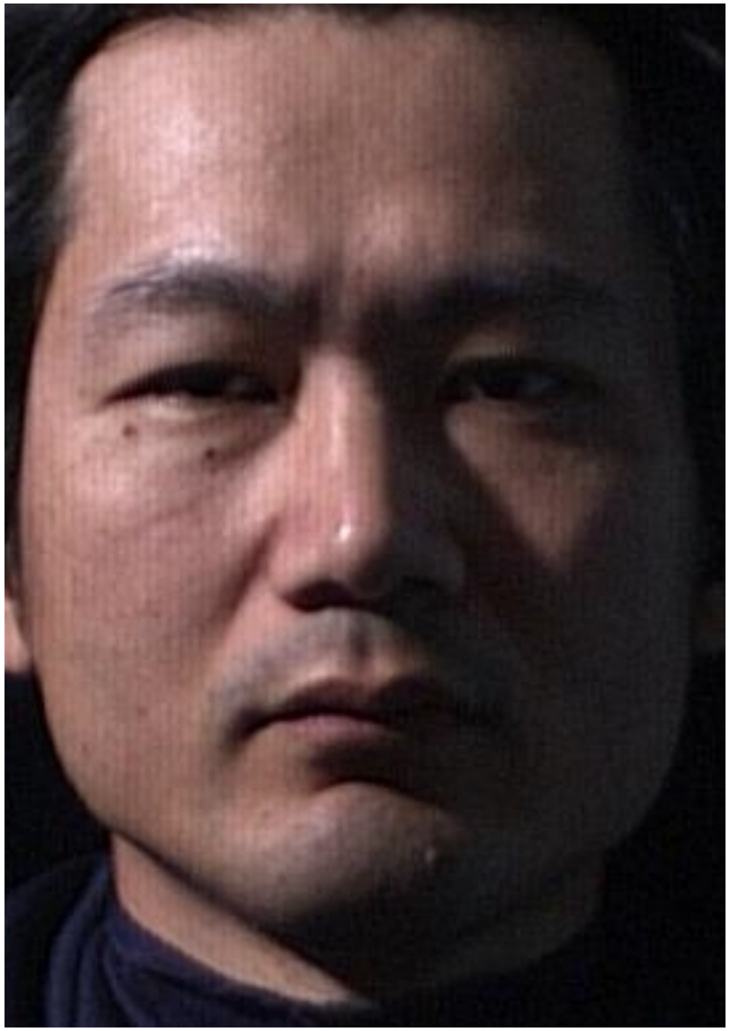
Challenges 1: view point variation

Michelangelo 1475-1564

slide by F

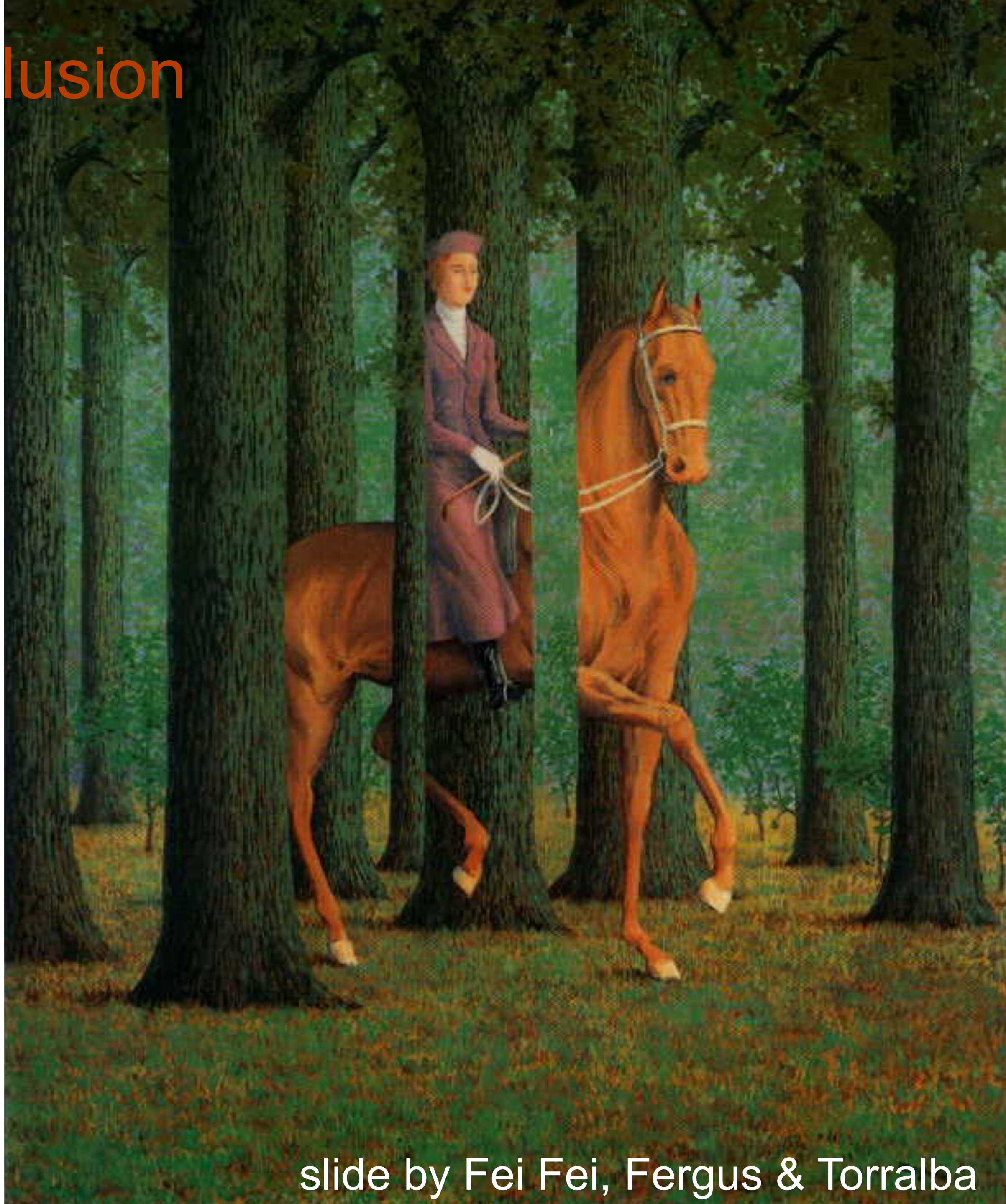
Challenges 2: illumination





slide credit: S. Ullman

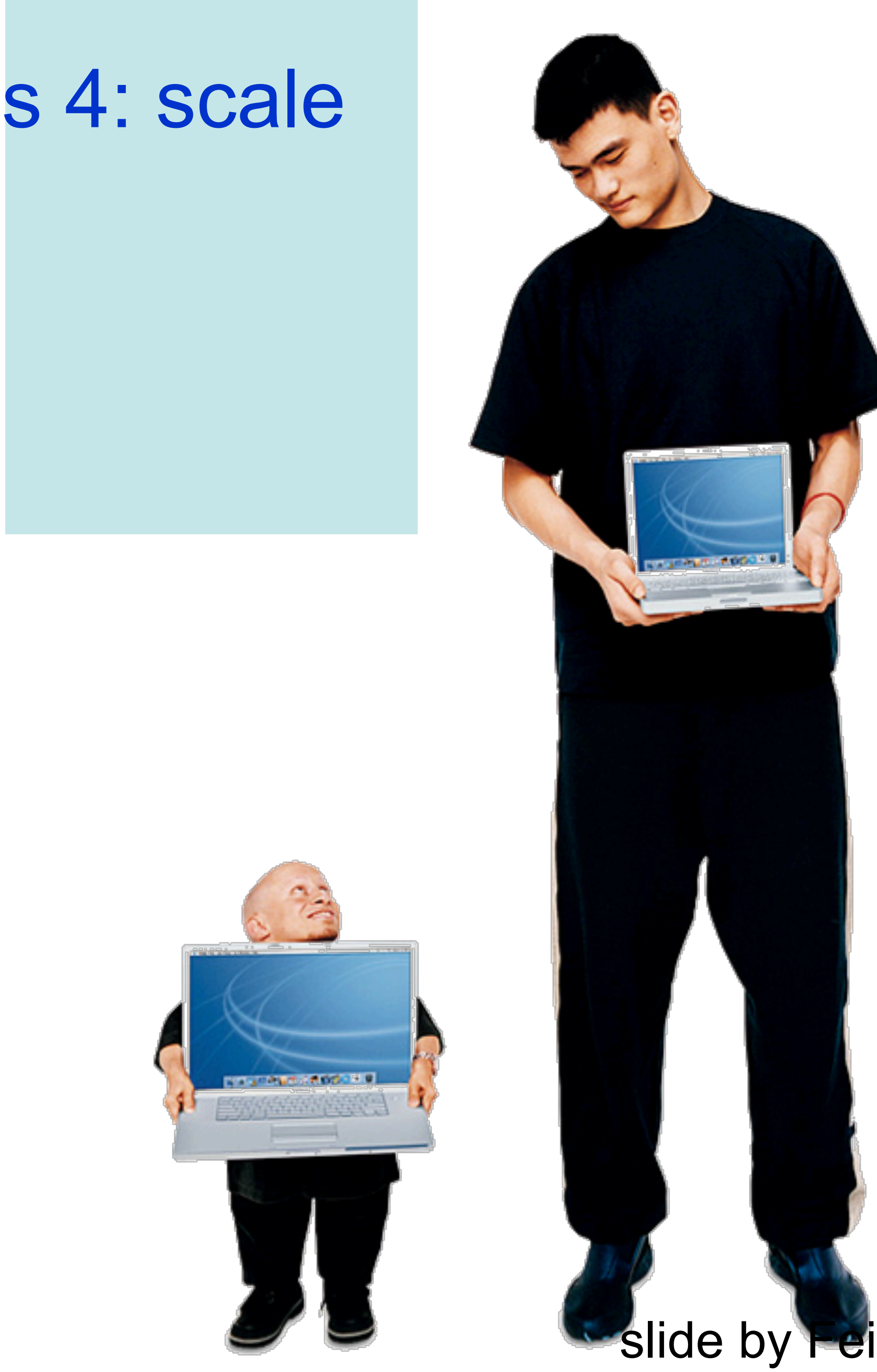
Challenges 3: occlusion



Magritte, 1957

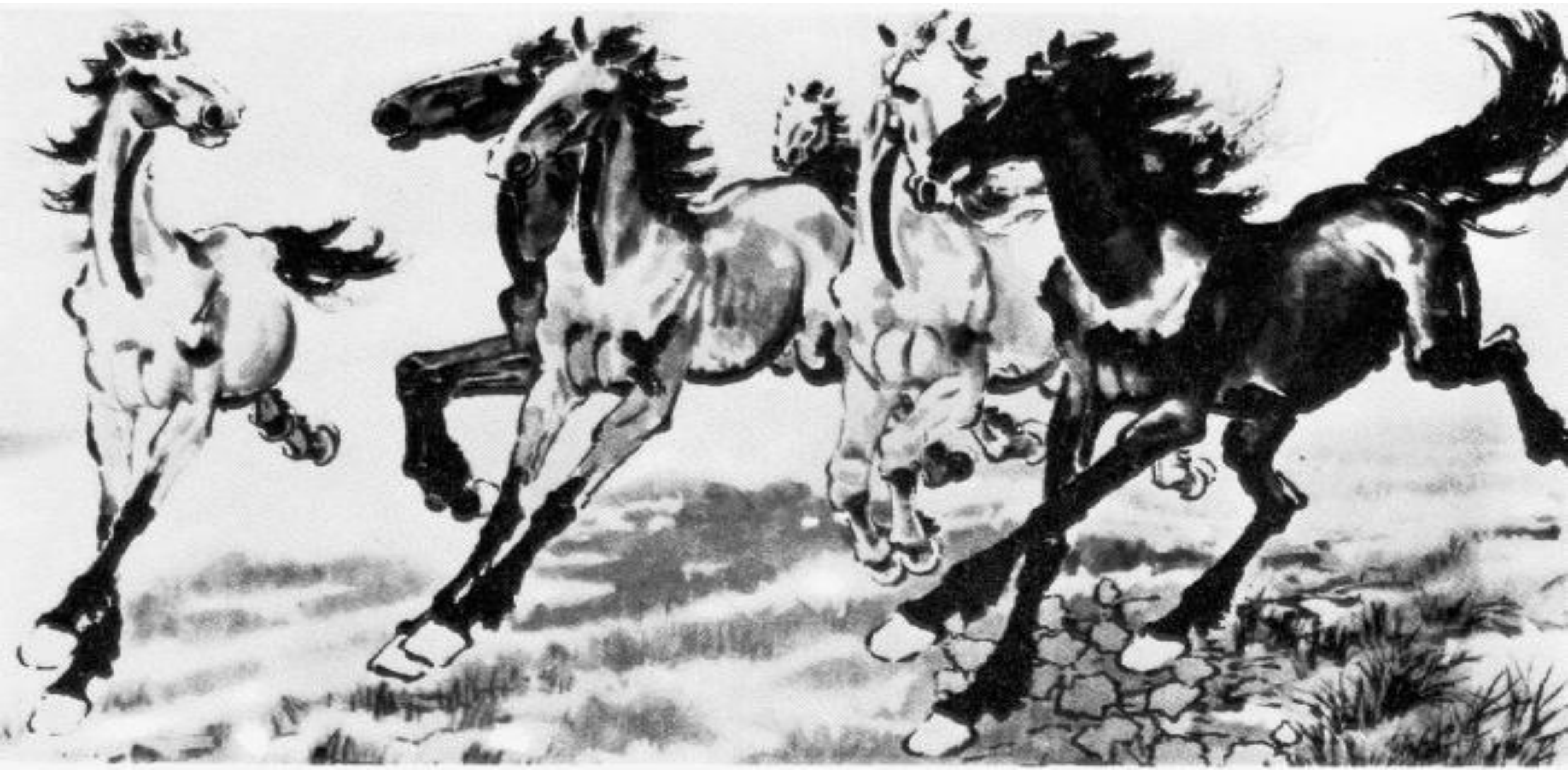
slide by Fei Fei, Fergus & Torralba

Challenges 4: scale



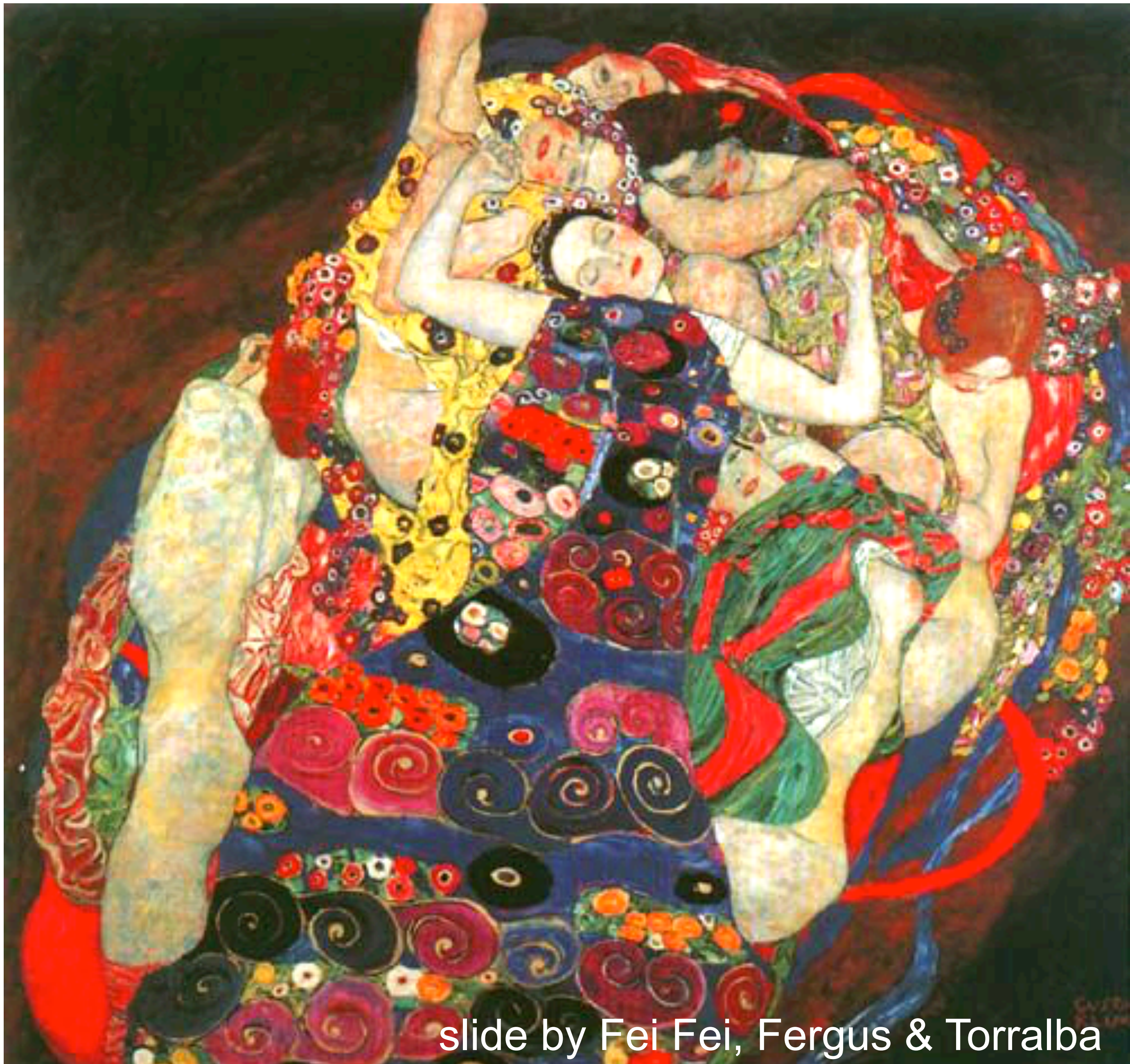
slide by Fei Fei, Fergus & Torralba

Challenges 5: deformation



Xu, Beihong 1943

Challenges 6: background clutter



Klimt, 1913

slide by Fei Fei, Fergus & Torralba

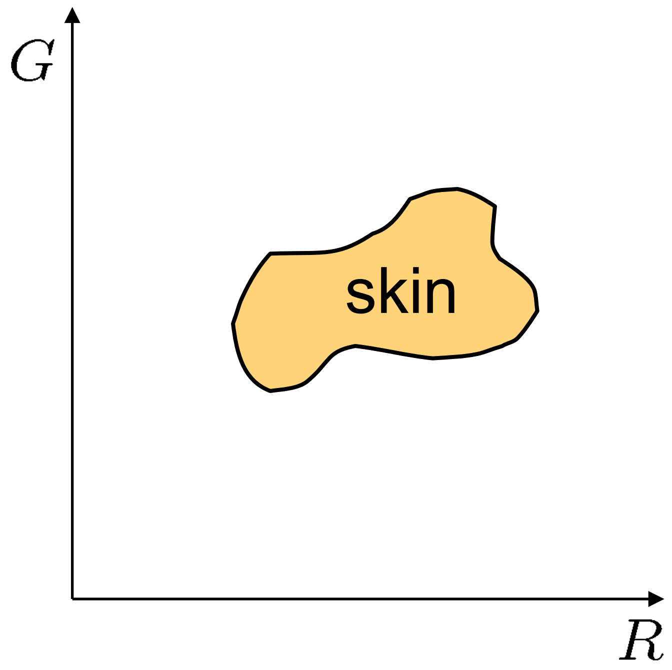
Challenges 7: object intra-class variation

slide by Fei-Fei, Fergus & Torralba

Let’s start with finding Faces

How to tell if a face is present?

One simple method: skin detection



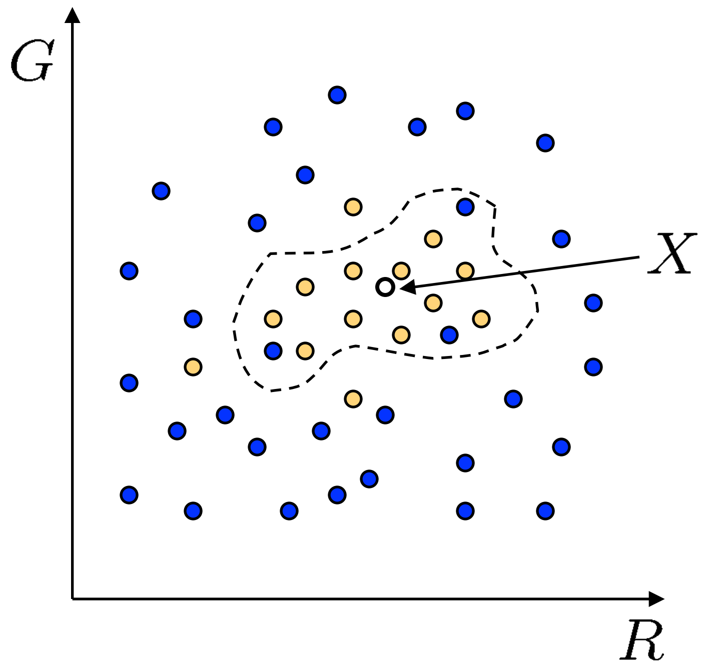
Skin pixels have a distinctive range of colors

- Corresponds to region(s) in RGB color space
  - for visualization, only R and G components are shown above

Skin classifier

- A pixel X = (R,G,B) is skin if it is in the skin region
- But how to find this region?

Skin detection



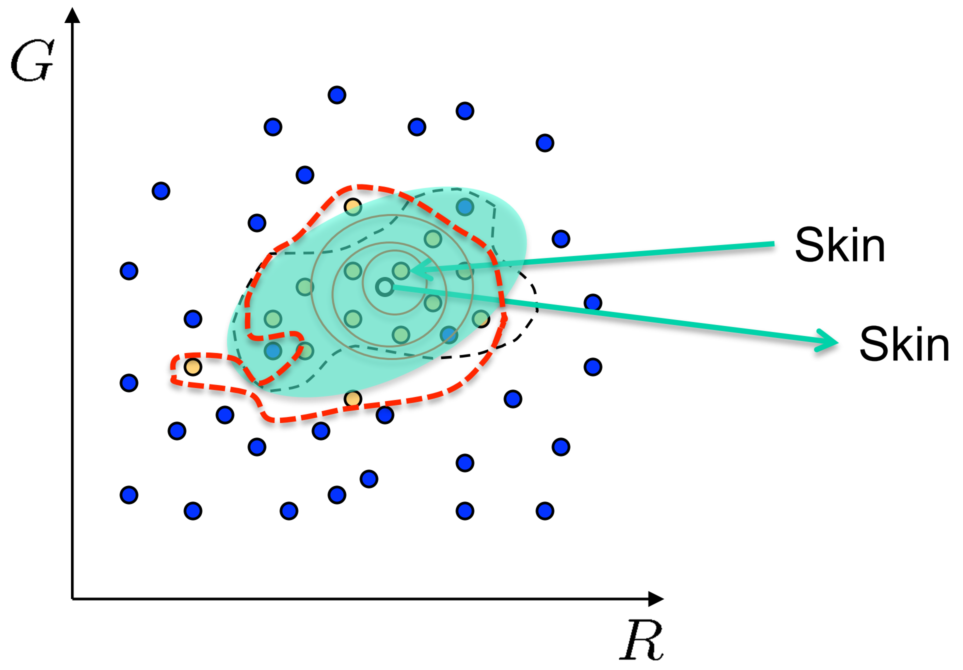
Learn the skin region from examples

- Manually label pixels in one or more “training images” as skin or not skin
- Plot the training data in RGB space
  - skin pixels shown in orange, non-skin pixels shown in blue
  - some skin pixels may be outside the region, non-skin pixels inside. Why?

Skin classifier

- Given X = (R,G,B): how to determine if it is skin or not?

Skin classification techniques



Skin classifier

- Given X = (R,G,B): how to determine if it is skin or not?
- Nearest neighbor
    - find labeled pixel closest to X
    - choose the label for that pixel
- Data modeling
    - Model the distribution that generates the data (Generative)
    - Model the boundary (Discriminative)

Classification

- Probabilistic
- Supervised Learning
- Discriminative vs. Generative
- Ensemble methods
- Linear models
- Non-linear models

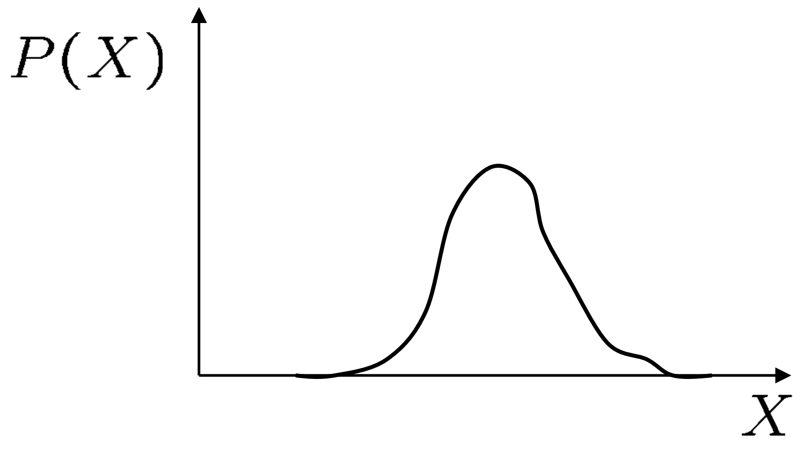
Let’s play with probability for a bit

Remembering simple stuff

Probability

Basic probability

- X is a random variable
- P(X) is the probability that X achieves a certain value



- $0 \le P(X) \le 1$
- $\int_{-\infty}^{\infty} P(X) dX = 1$ or $\sum P(X) = 1$

  continuous X discrete X

- Conditional probability: P(X | Y)
  - probability of X given that we already know Y

Thumbtack & Probabilities

P(Heads) = θ, P(Tails) = 1-θ

Flips are *i.i.d.*:

- Independent events $D = \{x_i \mid i = 1 \ldots n\}$, $P(D \mid \theta) = \Pi_i P(x_i \mid \theta)$
- Identically distributed according to Binomial distribution

Sequence *D* of α_H Heads and α_T Tails

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

Maximum Likelihood Estimation

Data: Observed set D of α_H Heads and α_T Tails

Hypothesis: Binomial distribution

Learning: finding θ is an optimization problem

- What's the objective function?

MLE: Choose θ to maximize probability of *D*

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \; P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \; \ln P(\mathcal{D} \mid \theta)\end{aligned}$$

Parameter learning

$$\begin{aligned}\widehat{\theta} &= \arg\max_{\theta} \ \ln P(\mathcal{D} \mid \theta) \\ &= \arg\max_{\theta} \ \ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\end{aligned}$$

Set derivative to zero, and solve!

$$\frac{d}{d\theta}\ln P(\mathcal{D} \mid \theta) = \frac{d}{d\theta}\left[\ln \theta^{\alpha_H}(1-\theta)^{\alpha_T}\right]$$

$$= \frac{d}{d\theta}\left[\alpha_H \ln \theta + \alpha_T \ln(1-\theta)\right]$$

$$= \alpha_H \frac{d}{d\theta}\ln \theta + \alpha_T \frac{d}{d\theta}\ln(1-\theta)$$

$$= \frac{\alpha_H}{\theta} - \frac{\alpha_T}{1-\theta} \quad = 0$$

$$\boxed{\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}}$$

But, how many flips do I need?

$$\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}$$

3 heads and 2 tails.

θ = 3/5, I can prove it!

What if I flipped 30 heads and 20 tails?

Same answer, I can prove it!

What's better?

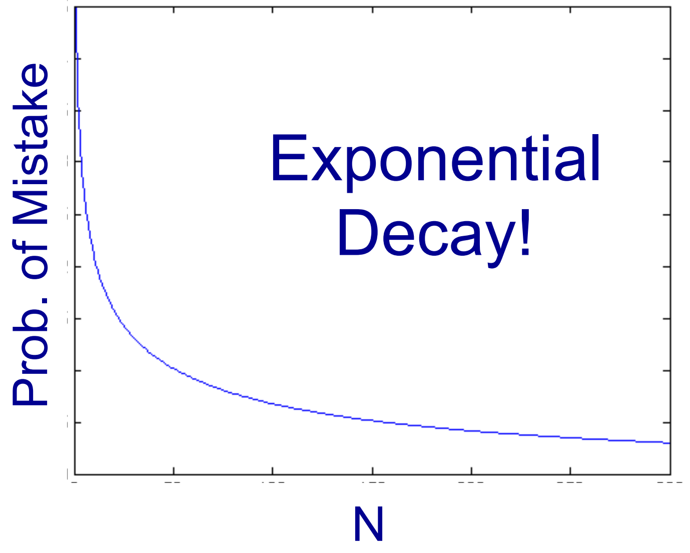
Umm… The more the merrier???

A bound (from Hoeffding's inequality)

For $N = \alpha_H + \alpha_T$, and
$$\widehat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T}$$

Let θ^* be the true parameter, for any $\varepsilon > 0$:

$$P(|\,\widehat{\theta} - \theta^*\,| \geq \epsilon) \;\leq\; 2e^{-2N\epsilon^2}$$

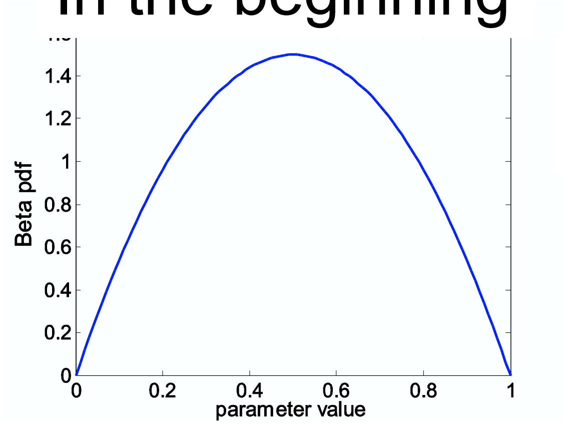


What if I have prior beliefs?

Wait, I know that the thumbtack is "close" to 50-50. What can you do for me now?

Rather than estimating a single θ, we obtain a distribution over possible values of θ

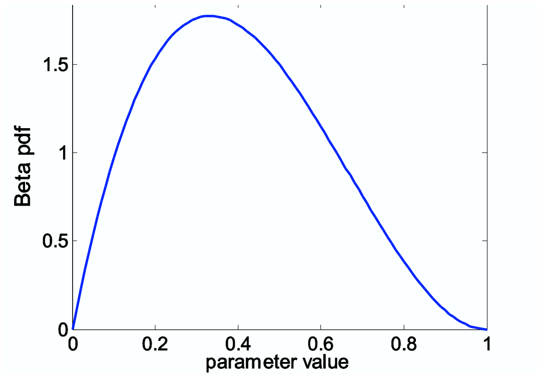
In the beginning



Observe flips
e.g.: {tails, tails}

$\longrightarrow$

After observations

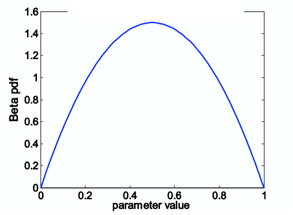


How to use Prior

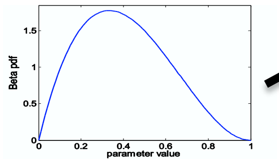
Use Bayes rule:

Data Likelihood

Prior



Posterior



$$P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta)P(\theta)}{P(\mathcal{D})}$$

Normalization

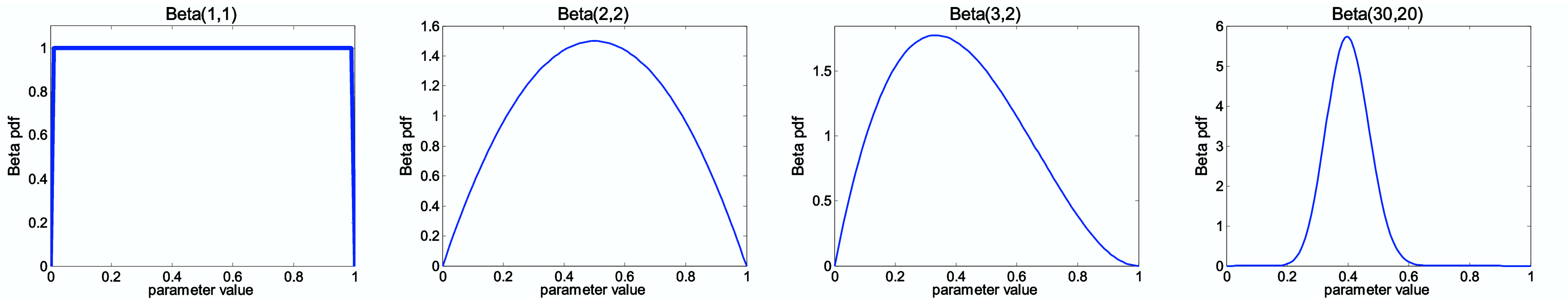
- Or equivalently: $P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta)P(\theta)$
- Also, for uniform priors:

  → reduces to MLE objective

$$P(\theta) \propto 1 \qquad P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta)$$

Beta prior distribution – P(θ)

$$P(\theta) = \frac{\theta^{\beta_H-1}(1-\theta)^{\beta_T-1}}{B(\beta_H,\beta_T)} \sim Beta(\beta_H,\beta_T)$$



Likelihood function:

$$P(\mathcal{D} \mid \theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$$

Posterior:

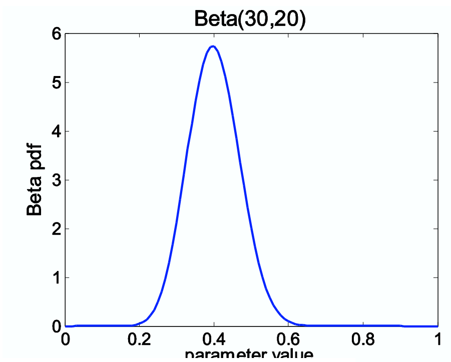
$$P(\theta \mid \mathcal{D}) \;\propto\; P(\mathcal{D} \mid \theta)P(\theta)$$

$$P(\theta \mid \mathcal{D}) \propto \theta^{\alpha_H}(1-\theta)^{\alpha_T}\;\theta^{\beta_H-1}(1-\theta)^{\beta_T-1}$$

$$= \theta^{\alpha_H+\beta_H-1}(1-\theta)^{\alpha_T+\beta_t+1}$$

$$= Beta(\alpha_H+\beta_H, \alpha_T+\beta_T)$$

MAP for Beta distribution

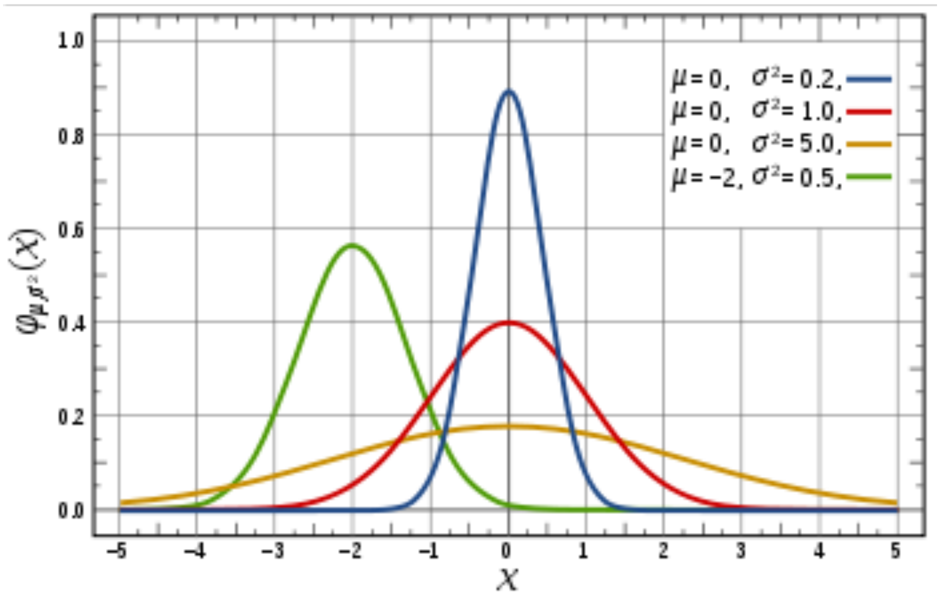


$$P(\theta \mid \mathcal{D}) = \frac{\theta^{\beta_H+\alpha_H-1}(1-\theta)^{\beta_T+\alpha_T-1}}{B(\beta_H+\alpha_H, \beta_T+\alpha_T)} \sim Beta(\beta_H+\alpha_H, \beta_T+\alpha_T)$$

MAP: use most likely parameter:

$$\widehat{\theta} = \arg\max_{\theta} P(\theta \mid \mathcal{D}) = \frac{\alpha_H+\beta_H-1}{\alpha_H+\beta_H+\alpha_T+\beta_T-2}$$

What about continuous variables?

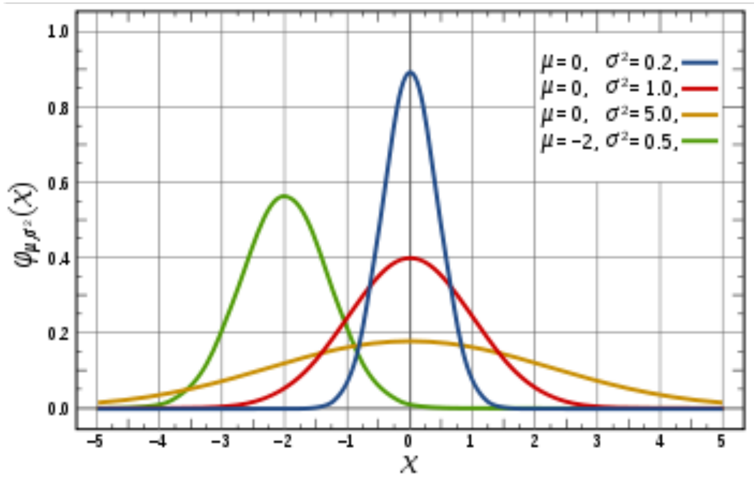


$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

We like Gaussians because

Affine transformation (multiplying by scalar and adding a constant) are Gaussian

- $X \sim N(\mu,\sigma^2)$
- $Y = aX + b \rightarrow Y \sim N(a\mu+b,a^2\sigma^2)$



Sum of Gaussians is Gaussian

- $X \sim N(\mu_X,\sigma^2_X)$
- $Y \sim N(\mu_Y,\sigma^2_Y)$
- $Z = X+Y \rightarrow Z \sim N(\mu_X+\mu_Y, \sigma^2_X+\sigma^2_Y)$

Easy to differentiate

Learning a Gaussian

- Collect a bunch of data
  - Hopefully, i.i.d. samples
  - e.g., exam scores
- Learn parameters
  - Mean: μ
  - Variance: σ

x_i $i =$	Exam Score
0	85
1	95
2	100
3	12
…	…
99	89

$$P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

MLE for Gaussian: $P(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

Prob. of i.i.d. samples $D=\{x_1,\ldots,x_N\}$:

$$P(\mathcal{D} \mid \mu, \sigma) = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{i=1}^{N} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$$

$$\mu_{MLE}, \sigma_{MLE} = \arg\max_{\mu,\sigma} P(\mathcal{D} \mid \mu, \sigma)$$

- Log-likelihood of data:

$$\begin{aligned} \ln P(\mathcal{D} \mid \mu, \sigma) &= \ln \left[\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^N \prod_{i=1}^{N} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}} \right] \\ &= -N \ln \sigma\sqrt{2\pi} - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\sigma^2} \end{aligned}$$

MLE for mean of a Gaussian

What's MLE for mean?

$$\begin{aligned}
\frac{d}{d\mu}\ln P(\mathcal{D}\mid\mu,\sigma) &= \frac{d}{d\mu}\left[-N\ln\sigma\sqrt{2\pi}-\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{2\sigma^2}\right] \\
&= \frac{d}{d\mu}\left[-N\ln\sigma\sqrt{2\pi}\right]-\sum_{i=1}^{N}\frac{d}{d\mu}\left[\frac{(x_i-\mu)^2}{2\sigma^2}\right] \\
&= -\sum_{i=1}^{N}\frac{(x_i-\mu)}{\sigma^2}=0 \\
&= -\sum_{i=1}^{N}x_i+N\mu=0
\end{aligned}$$

$$\widehat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N}x_i$$

MLE for variance

Again, set derivative to zero:

$$\begin{aligned}\frac{d}{d\sigma}\ln P(\mathcal{D}\mid\mu,\sigma) &= \frac{d}{d\sigma}\left[-N\ln\sigma\sqrt{2\pi}-\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{2\sigma^2}\right]\\ &= \frac{d}{d\sigma}\left[-N\ln\sigma\sqrt{2\pi}\right]-\sum_{i=1}^{N}\frac{d}{d\sigma}\left[\frac{(x_i-\mu)^2}{2\sigma^2}\right]\\ &= -\frac{N}{\sigma}+\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^3} \qquad = 0\end{aligned}$$

$$\boxed{\widehat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\widehat{\mu})^2}$$

Learning Gaussian parameters

MLE:

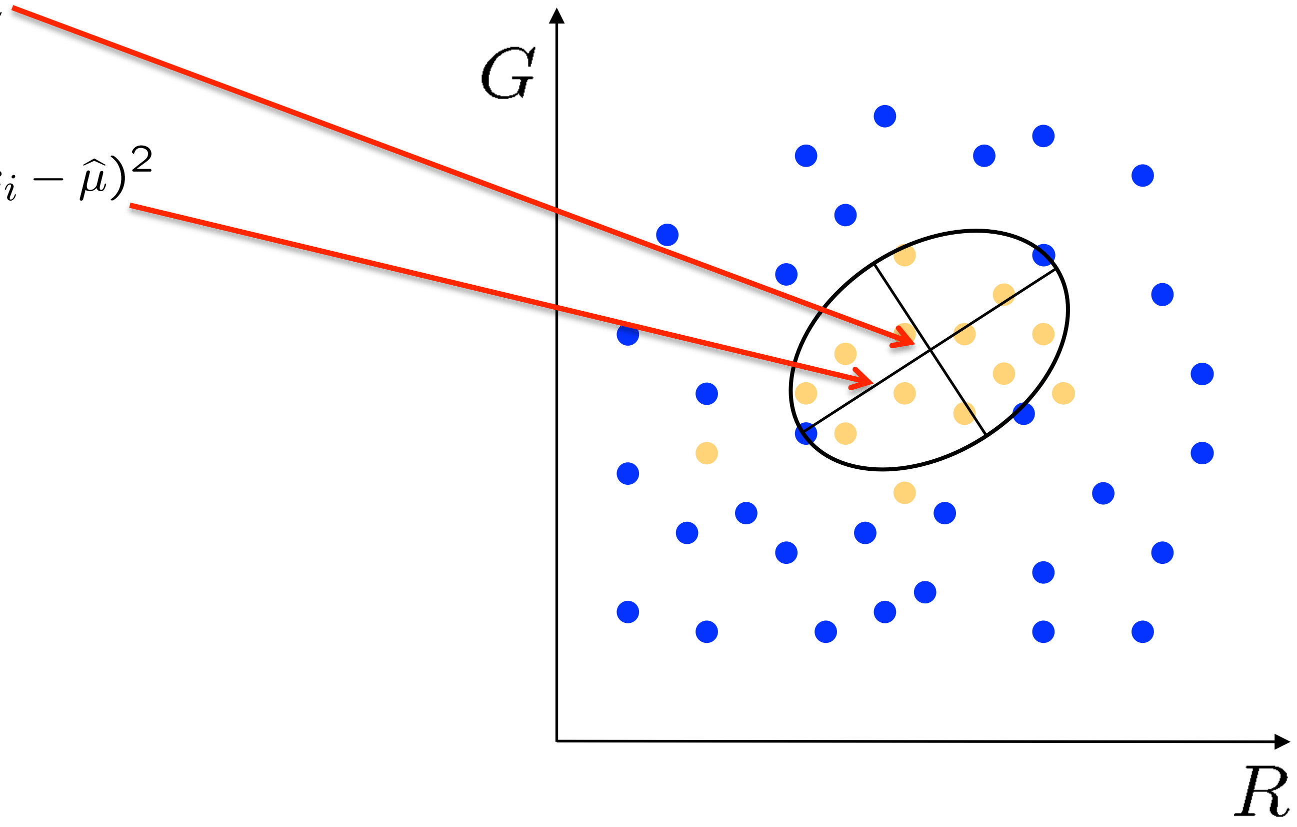
$$\widehat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\widehat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \widehat{\mu})^2$$

Fitting a Gaussian to Skin samples

$$\widehat{\mu}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\widehat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \widehat{\mu})^2$$



Skin detection results

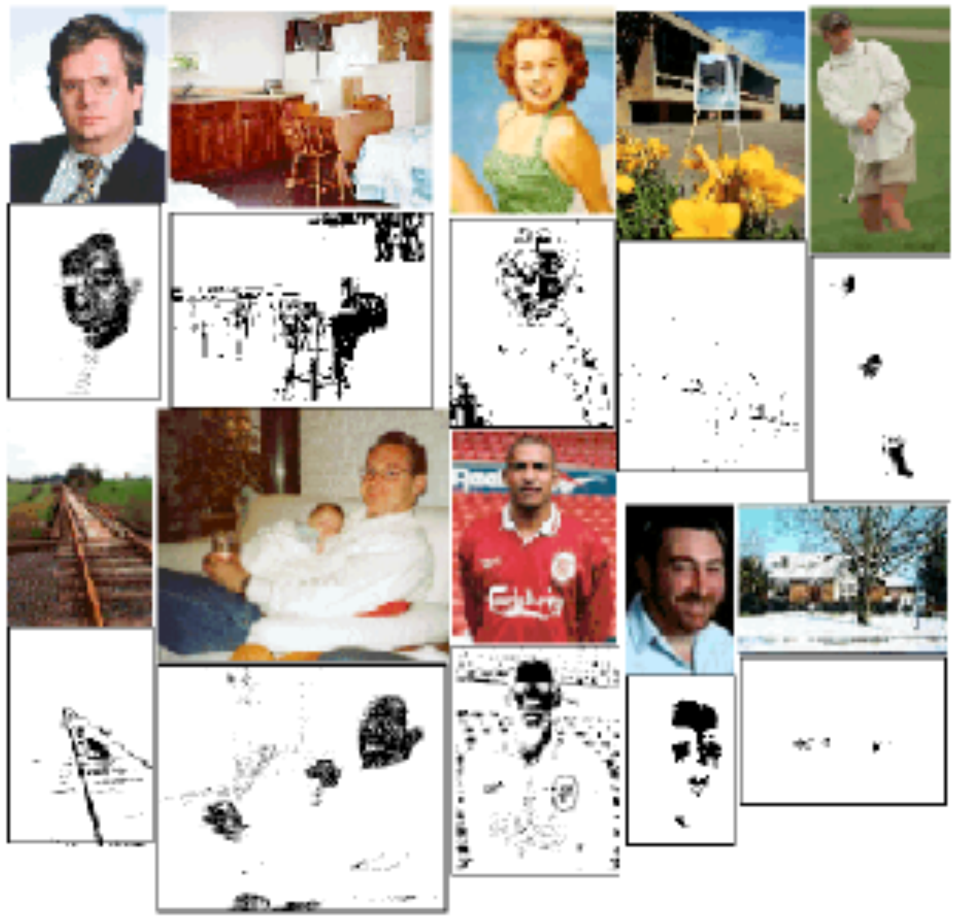


Figure 25.3. The figure shows a variety of images together with the output of the skin detector of Jones and Rehg applied to the image. Pixels marked black are skin pixels, and white are background. Notice that this process is relatively effective, and could certainly be used to focus attention on, say, faces and hands. *Figure from "Statistical color models with application to skin detection," M.J. Jones and J. Rehg, Proc. Computer Vision and Pattern Recognition, 1999 © 1999, IEEE*

Supervised Learning: find f

Given: Training set $\{(x_i, y_i) \mid i = 1 \ldots n\}$

Find: A good approximation to $f : X \rightarrow Y$

What is x?

What is y?

Simple Example: Digit Recognition

Input: images / pixel grids

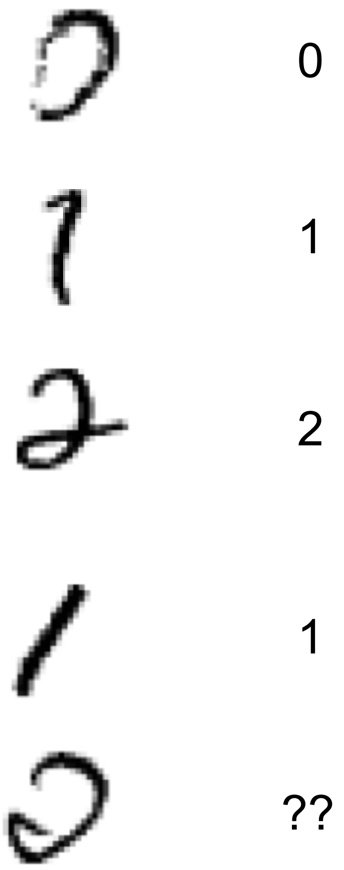
Output: a digit 0-9

Setup:

- Get a large collection of example images, each labeled with a digit
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future digit images

Features: ?

Screw You, I want to use Pixels :D



Lets take a probabilistic approach!!!

Can we directly estimate the data distribution P(X,Y)?

How do we represent these? How many parameters?

- Prior, P(Y):
  - Suppose Y is composed of *k* classes

- Likelihood, P(**X**|Y):
  - Suppose **X** is composed of *n* binary features

Conditional Independence

X is **conditionally independent** of Y given Z, if the probability distribution for X is independent of the value of Y, given the value of Z

e.g.,

$$(\forall i, j, k) P(X = i | Y = j, Z = k) = P(X = i | Z = k)$$

Equivalent to:

$$P(Thunder | Rain, Lightning) = P(Thunder | Lightning)$$

$$P(X, Y \mid Z) = P(X \mid Z) P(Y \mid Z)$$

Naïve Bayes

Naïve Bayes assumption:

- Features are independent given class:

$$\begin{aligned} P(X_1, X_2|Y) &= P(X_1|X_2, Y)P(X_2|Y) \\ &= P(X_1|Y)P(X_2|Y) \end{aligned}$$

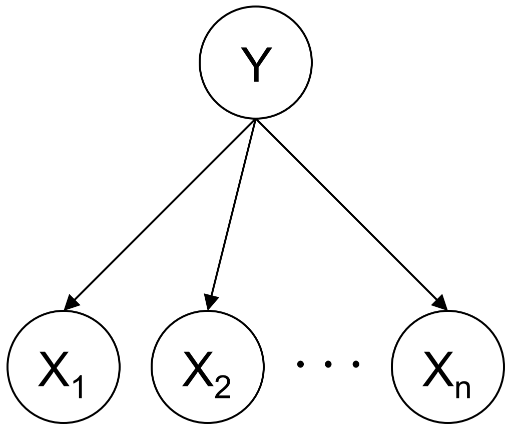
- More generally:

$$\boxed{P(X_1 \ldots X_n|Y) = \prod_i P(X_i|Y)}$$

The Naïve Bayes Classifier

Given:

- Prior P(Y)
- *n* conditionally independent features **X** given the class Y
- For each X_i, we have likelihood $P(X_i|Y)$

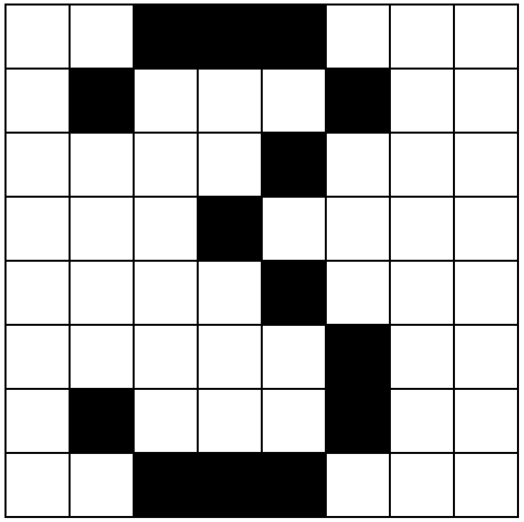


Decision rule:

$$\begin{aligned} y^* = h_{NB}(\mathbf{x}) &= \arg\max_y P(y) P(x_1, \ldots, x_n \mid y) \\ &= \arg\max_y P(y) \prod_i P(x_i|y) \end{aligned}$$

A Digit Recognizer

Input: pixel grids



Output: a digit 0-9

Naïve Bayes for Digits (Binary Inputs)

Simple version:

- One feature F_{ij} for each grid position <i,j>
- Possible feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
- Each input maps to a feature vector, e.g.

$$\rightarrow \langle F_{0,0} = 0 \;\; F_{0,1} = 0 \;\; F_{0,2} = 1 \;\; F_{0,3} = 1 \;\; F_{0,4} = 0 \;\; \ldots F_{15,15} = 0 \rangle$$

- Here: lots of features, each is binary valued

Naïve Bayes model:

$$P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$$

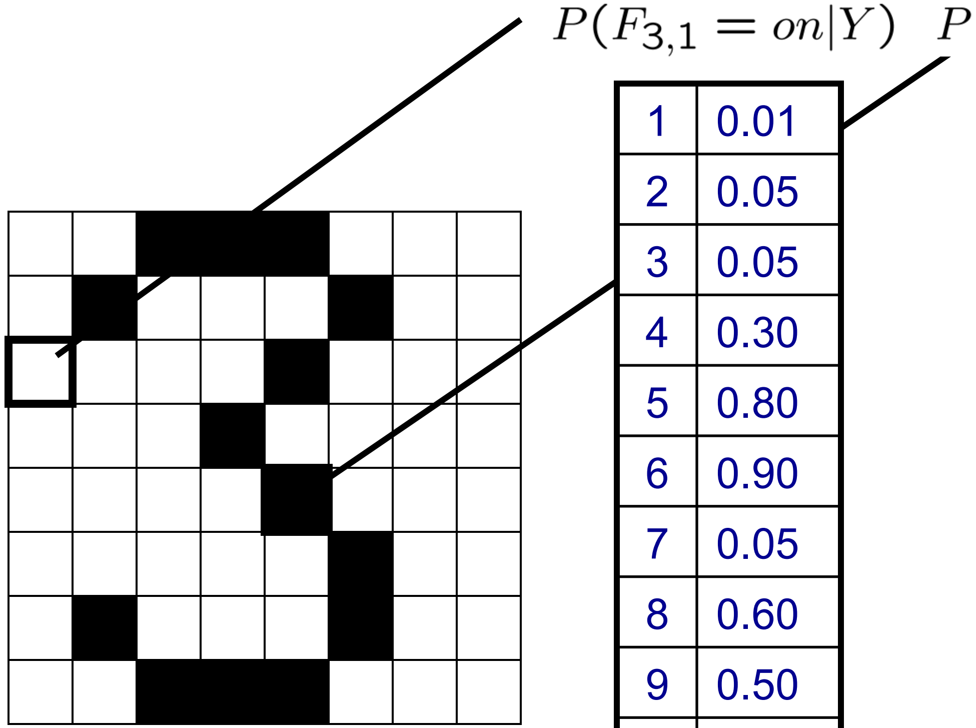
Are the features independent given class?

What do we need to learn?

Example Distributions

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

MLE for the parameters of NB

Given dataset

- Count(A=a,B=b) number of examples where A=a and B=b

MLE for discrete NB, simply:

- Prior:

$$P(Y=y) = \frac{Count(Y=y)}{\sum_{y'} Count(Y=y')}$$

- Likelihood:

$$P(X_i = x | Y = y) = \frac{Count(X_i = x, Y = y)}{\sum_{x'} Count(X_i = x', Y = y)}$$

Violating the NB assumption

Usually, features are not conditionally independent:

$$\boxed{P(X_1 ... X_n|Y) \neq \prod_i P(X_i|Y)}$$

- NB often performs well, even when assumption is violated
- [Domingos & Pazzani '96] discuss some conditions for good performance

Smoothing

$P(\text{features}, C = 2)$

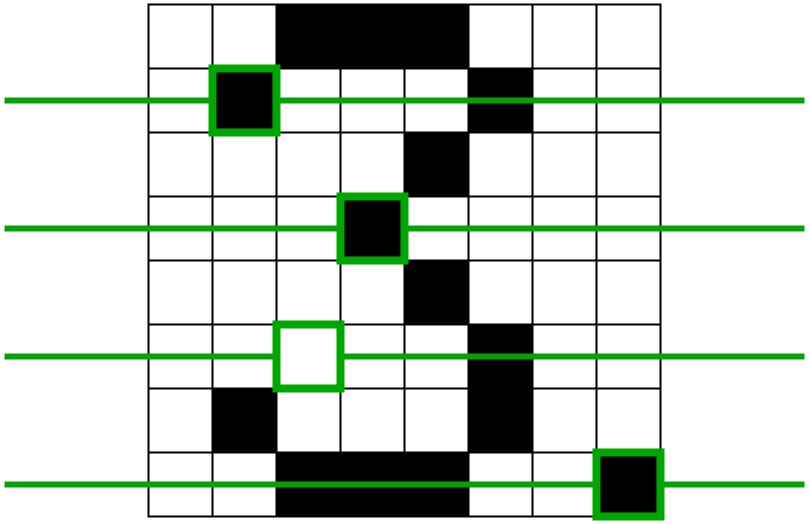
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

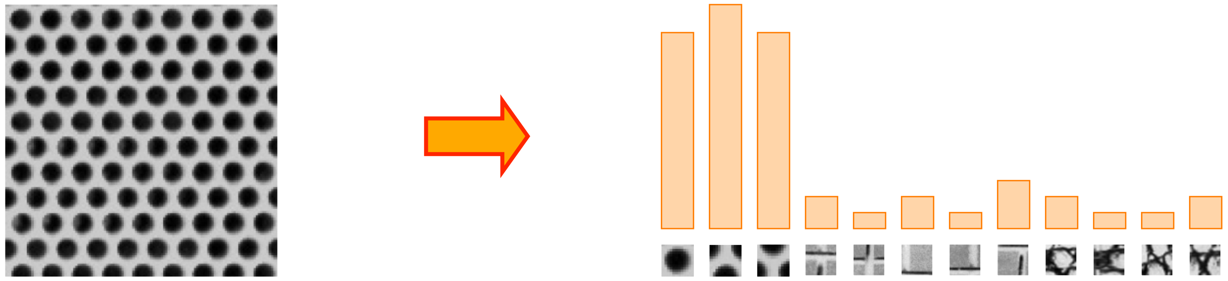
$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

2 wins!!

Does this happen in vision?

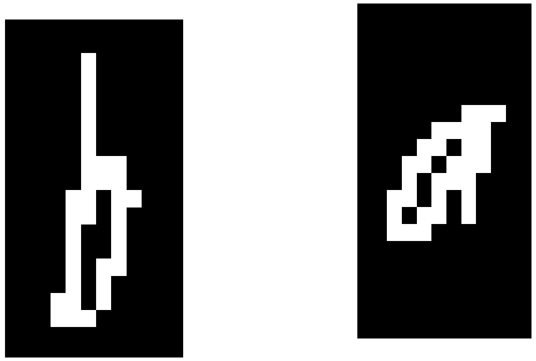
NB & Bag of words model



$$P(y) \quad \prod_{i=1} \quad P(x_i|y)$$

What about real Features? What if we have continuous X_i ?

Eg., character recognition: X_i is ith pixel



Gaussian Naïve Bayes (GNB):

$$P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Sometimes assume variance
is independent of Y (i.e., σ_i),
or independent of X_i (i.e., σ_k)
or both (i.e., σ)

Estimating Parameters

Maximum likelihood estimates:

Mean:

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

jth training example

δ(*x*)=1 if x true, else 0

Variance:

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k) - 1} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

another probabilistic approach!!!

Naïve Bayes: directly estimate the data distribution P(X,Y)!

- challenging due to size of distribution!
- make Naïve Bayes assumption: only need $P(X_i|Y)$!

But wait, we classify according to:

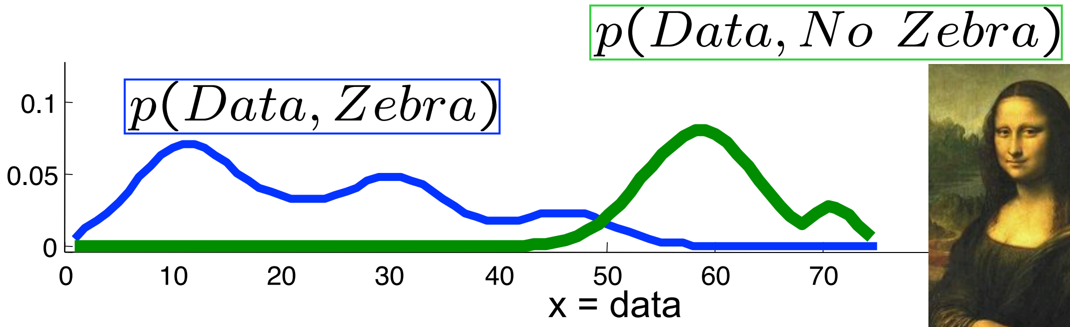
- $\max_Y P(Y|X)$

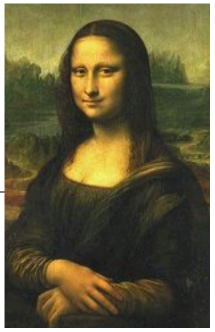
Why not learn P(Y|X) directly?

Discriminative vs. generative

- Generative model

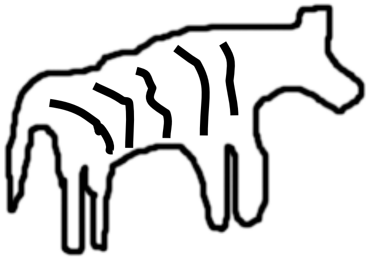
  (*The artist*)

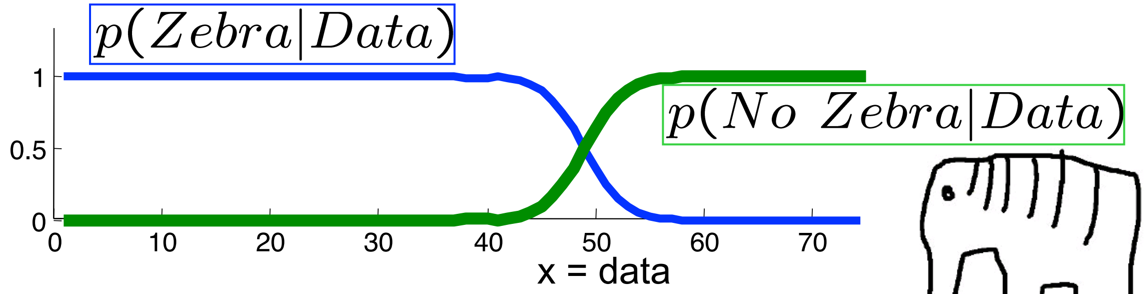




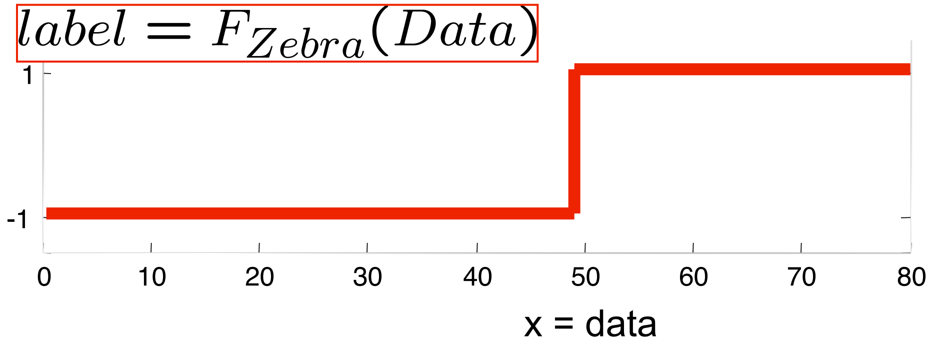
- Discriminative model

  (The lousy painter)





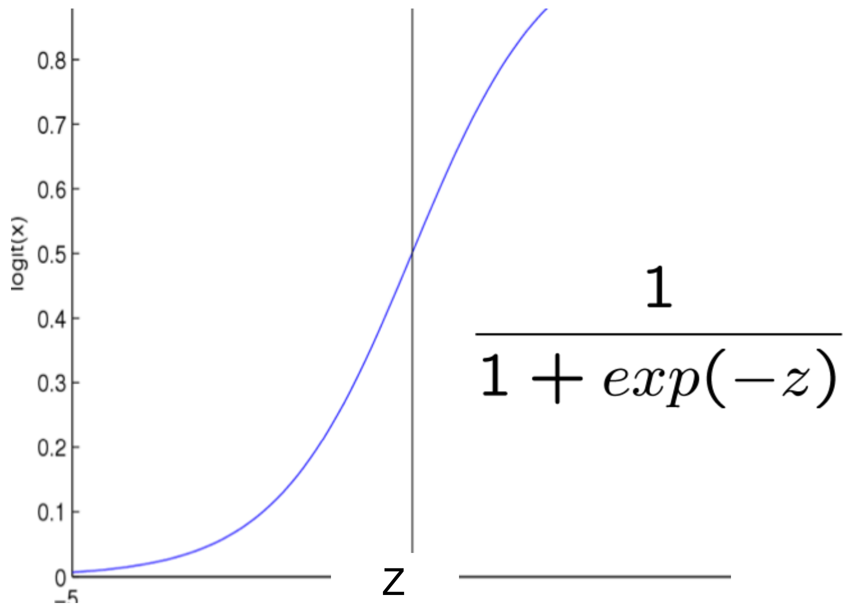
- Classification function



Logistic Regression

- Learn P(Y|**X**) directly!
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

Logistic function (Sigmoid):



$$\frac{1}{1+exp(-z)}$$

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)}$$

$$P(Y=0|X) = \frac{\exp(w_0+\sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0+\sum_{i=1}^{n} w_i X_i)}$$

Logistic Regression: decision boundary

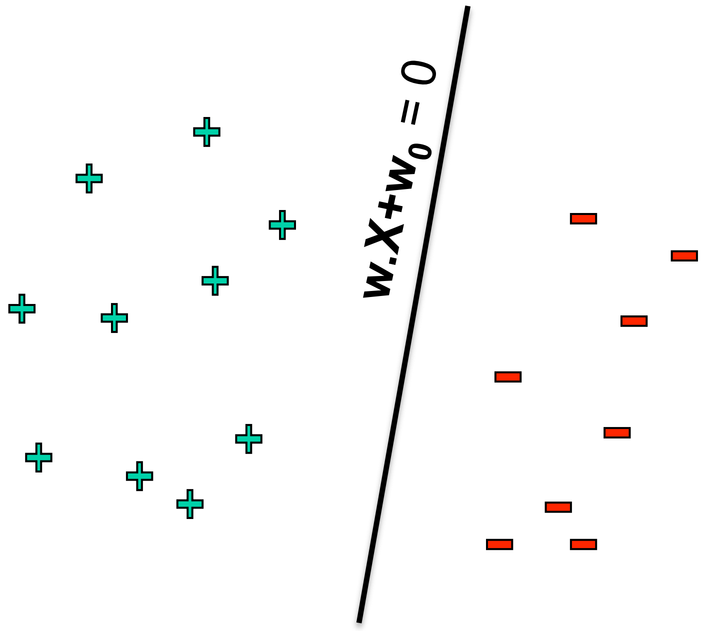
$$P(Y=1|X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^n w_i X_i)} \qquad P(Y=0|X) = \frac{\exp(w_0 + \sum_{i=1}^n w_i X_i)}{1+\exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

- Prediction: Output the Y with highest P(Y|X)
  - For binary Y, output Y=0 if

$$1 < \frac{P(Y=0|X)}{P(Y=1|X)}$$

$$1 < \exp(w_0 + \sum_{i=1}^n w_i X_i)$$

$$0 < w_0 + \sum_{i=1}^n w_i X_i$$



A Linear Classifier!

Loss functions / Learning Objectives: Likelihood v. Conditional Likelihood

Generative (Naïve Bayes) Loss function:

Data likelihood

$$\begin{aligned} \ln P(\mathcal{D} \mid \mathbf{w}) &= \sum_{j=1}^{N} \ln P(\mathbf{x}^j, y^j \mid \mathbf{w}) \\ &= \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w}) + \sum_{j=1}^{N} \ln P(\mathbf{x}^j \mid \mathbf{w}) \end{aligned}$$

But, discriminative (logistic regression) loss function:

Conditional Data Likelihood

$$\ln P(\mathcal{D}_Y \mid \mathcal{D}_{\mathbf{X}}, \mathbf{w}) = \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w})$$

- Doesn't waste effort learning P(X) – focuses on P(Y|**X**) all that matters for classification
- Discriminative models cannot compute P(**x**^{j}|**w**)!

Conditional Log Likelihood

$$P(Y=0|\mathbf{X},\mathbf{w}) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$l(\mathbf{w}) \equiv \sum_j \ln P(y^j|\mathbf{x}^j,\mathbf{w})$$

$$P(Y=1|\mathbf{X},\mathbf{w}) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

equal because y^j is in {0,1}

$$l(\mathbf{w}) = \sum_j y^j \ln P(y^j=1|\mathbf{x}^j,\mathbf{w}) + (1-y^j)\ln P(y^j=0|\mathbf{x}^j,\mathbf{w})$$

remaining steps: substitute definitions, expand logs, and simplify

$$= \sum_j y^j \ln \frac{e^{w_0+\sum_i w_i X_i}}{1+e^{w_0+\sum_i w_i X_i}} + (1-y^j)\ln \frac{1}{1+e^{w_0+\sum_i w_i X_i}}$$

Logistic Regression Parameter Estimation: Maximize Conditional Log Likelihood

$$\begin{aligned} l(\mathbf{w}) &\equiv \ln \prod_j P(y^j | \mathbf{x}^j, \mathbf{w}) \\ &= \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j)) \end{aligned}$$

Good news: *l*(**w**) is concave function of **w**

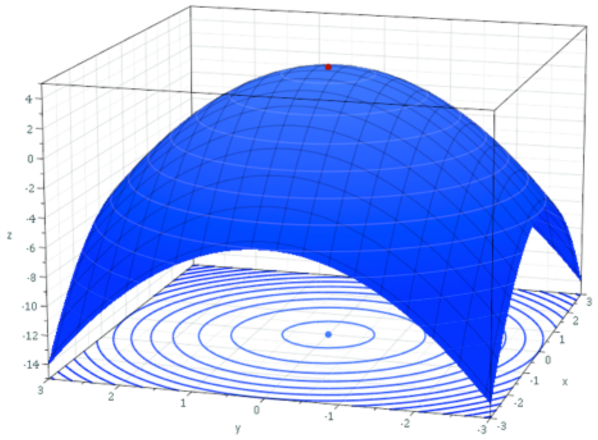
$\rightarrow$ no locally optimal solutions!

Bad news: no closed-form solution to maximize *l*(**w**)

Good news: concave functions "easy" to optimize

Optimizing concave function – Gradient ascent

Conditional likelihood for Logistic Regression is concave !



Gradient: $$\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$$

Update rule: $$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\mathbf{w})}{\partial w_i}$$

Gradient ascent is simplest of optimization approaches

- e.g., Conjugate gradient ascent much better

Maximize Conditional Log Likelihood: Gradient ascent

$$\boxed{P(Y=1|X,W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}}$$

$$l(\mathbf{w}) \;=\; \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j))$$

$$\frac{\partial l(w)}{\partial w_i} = \sum_j \left[\frac{\partial}{\partial w} y^j (w_0 + \sum_i w_i x_i^j) - \frac{\partial}{\partial w} \ln \left(1 + \exp(w_0 + \sum_i w_i x_i^j) \right) \right]$$

$$= \sum_j \left[y^j x_i^j - \frac{x_i^j \exp(w_0 + \sum_i w_i x_i^j)}{1 + \exp(w_0 + \sum_i w_i x_i^j)} \right]$$

$$= \sum_j x_i^j \left[y^j - \frac{\exp(w_0 + \sum_i w_i x_i^j)}{1 + \exp(w_0 + \sum_i w_i x_i^j)} \right]$$

$$\boxed{\frac{\partial l(w)}{\partial w_i} = \sum_j x_i^j \left(y^j - P(Y^j = 1 | x^j, w) \right)}$$

Gradient ascent for LR

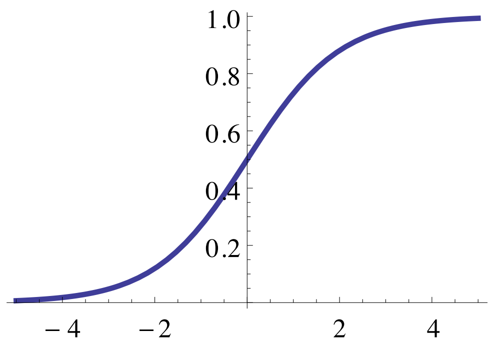
Gradient ascent algorithm: (learning rate η > 0)

```
do:
    $w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$
    For i=1…n: (iterate over weights)
        $w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]$
until "change" < ε
```

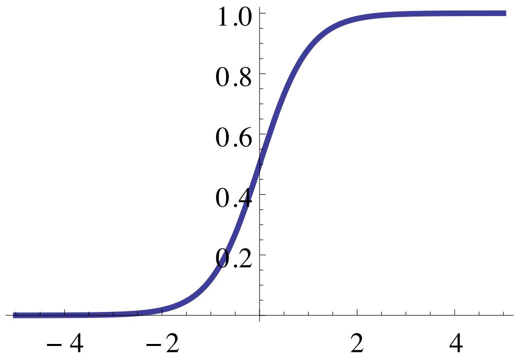
Loop over training examples!

Large parameters…

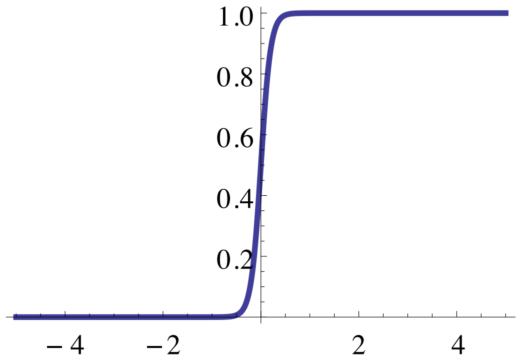
$$\boxed{\frac{1}{1+e^{-ax}}}$$



a=1



a=5



a=10

Maximum likelihood solution: prefers higher weights

- higher likelihood of (properly classified) examples close to decision boundary
- larger influence of corresponding features on decision
- *can cause overfitting!!!*

Regularization: penalize high weights

- again, more on this later in the quarter

How about MAP?

$$p(\mathbf{w} \mid Y, \mathbf{X}) \;\propto\; P(Y \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})$$

One common approach is to define priors on **w**

- Normal distribution, zero mean, identity covariance

$$p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} \; e^{\frac{-w_i^2}{2\kappa^2}}$$

Often called ***Regularization***

- Helps avoid very large weights and overfitting

MAP estimate:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

M(C)AP as Regularization

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right] \qquad p(\mathbf{w}) = \prod_i \frac{1}{\kappa\sqrt{2\pi}} \; e^{\frac{-w_i^2}{2\kappa^2}}$$

Add log p(w) to objective:

$$\ln p(w) \propto -\frac{\lambda}{2} \sum_i w_i^2 \qquad\qquad \frac{\partial \ln p(w)}{\partial w_i} = -\lambda w_i$$

- Quadratic penalty: drives weights towards zero
- Adds a negative linear term to the gradients

MLE vs. MAP

Maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[\prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})]}$$

Maximum conditional a posteriori estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[p(\mathbf{w}) \prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$\boxed{w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w})] \right\}}$$

Logistic regression v. Naïve Bayes

Consider learning f: X → Y, where

- X is a vector of real-valued features, < X_1 … X_n >
- Y is boolean

Could use a Gaussian Naïve Bayes classifier

- assume all X_i are conditionally independent given Y
- model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
- model P(Y) as Bernoulli(θ,1-θ)

What does that imply about the form of P(Y|X)?

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

Derive form for P(Y|X) for continuous X_i

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

up to now, all arithmetic

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

only for Naïve Bayes models

$$= \frac{1}{1 + \exp(\ (\ln \frac{1-\theta}{\theta}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

Looks like a setting for w_0?

Can we solve for w_i ?

- Yes, but only in Gaussian case

Ratio of class-conditional probabilities

$$\boxed{P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_i\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_i^2}}}$$

$$\ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)}$$

$$= \ln \left[\frac{\frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}}}{\frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x_i-\mu_{i1})^2}{2\sigma_i^2}}} \right]$$

$$= -\frac{(x_i-\mu_{i0})^2}{2\sigma_i^2} + \frac{(x_i-\mu_{i1})^2}{2\sigma_i^2}$$

$$\cdots$$

$$= \frac{\mu_{i0}+\mu_{i1}}{\sigma_i^2} x_i + \frac{\mu_{i0}^2+\mu_{i1}^2}{2\sigma_i^2}$$

Linear function! Coefficients expressed with original Gaussian parameters!

Derive form for P(Y|X) for continuous X_i

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \exp(\ (\ln \frac{1-\theta}{\theta}) + \boxed{\sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}}$$

$$\Downarrow$$

$$\boxed{\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)}$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

$$\boxed{w_0 = \ln \frac{1-\theta}{\theta} + \frac{\mu_{i0}^2 + \mu_{i1}^2}{2\sigma_i^2}}$$

$$\boxed{w_i = \frac{\mu_{i0} + \mu_{i1}}{\sigma_i^2}}$$

Gaussian Naïve Bayes vs. Logistic Regression

Set of Gaussian Naïve Bayes parameters (feature variance independent of class label)

$\longleftrightarrow$

Can go both ways, we only did one way

Set of Logistic Regression parameters

Representation equivalence

- **But only in a special case!!!** (GNB with class-independent variances)

But what's the difference???

LR makes no assumptions about P(**X**|Y) **in learning**!!!

Loss function!!!

- Optimize different functions ! Obtain different solutions

Naïve Bayes vs. Logistic Regression

Consider Y boolean, X_i continuous, $X=<X_1 \ldots X_n>$

Number of parameters:

Naïve Bayes: 4n +1

Logistic Regression: n+1

Estimation method:

Naïve Bayes parameter estimates are uncoupled

Logistic Regression parameter estimates are coupled

Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

Generative vs. Discriminative classifiers

Asymptotic comparison

(# training examples → infinity)

- when model correct
  - GNB (with class independent variances) and LR produce identical classifiers

- when model incorrect
  - LR is less biased – does not assume conditional independence
    - » **therefore LR expected to outperform GNB**

Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

Generative vs. Discriminative classifiers

Non-asymptotic analysis

- convergence rate of parameter estimates,
  (n = # of attributes in X)
  - Size of training data to get close to infinite data solution
  - Naïve Bayes needs *O*(log n) samples
  - Logistic Regression needs *O*(n) samples

- GNB converges more quickly to its (perhaps less helpful) asymptotic estimates

What you should know about Logistic Regression (LR)

Gaussian Naïve Bayes with class-independent variances representationally equivalent to LR

- Solution differs because of objective (loss) function

In general, NB and LR make different assumptions

- NB: Features independent given class ! assumption on P(**X**|Y)
- LR: Functional form of P(Y|**X**), no assumption on P(**X**|Y)

LR is a linear classifier

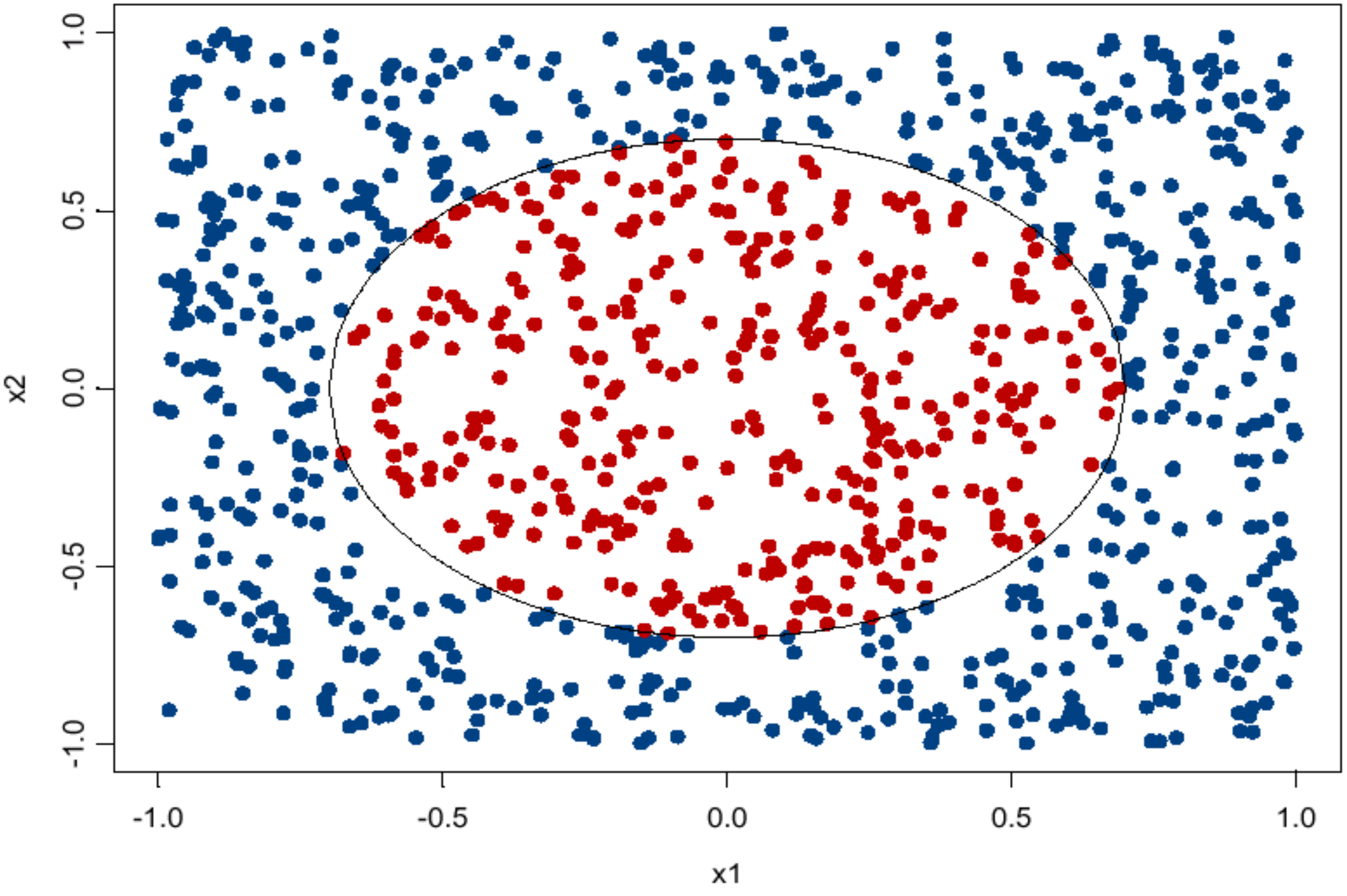
- decision rule is a hyperplane

LR optimized by conditional likelihood

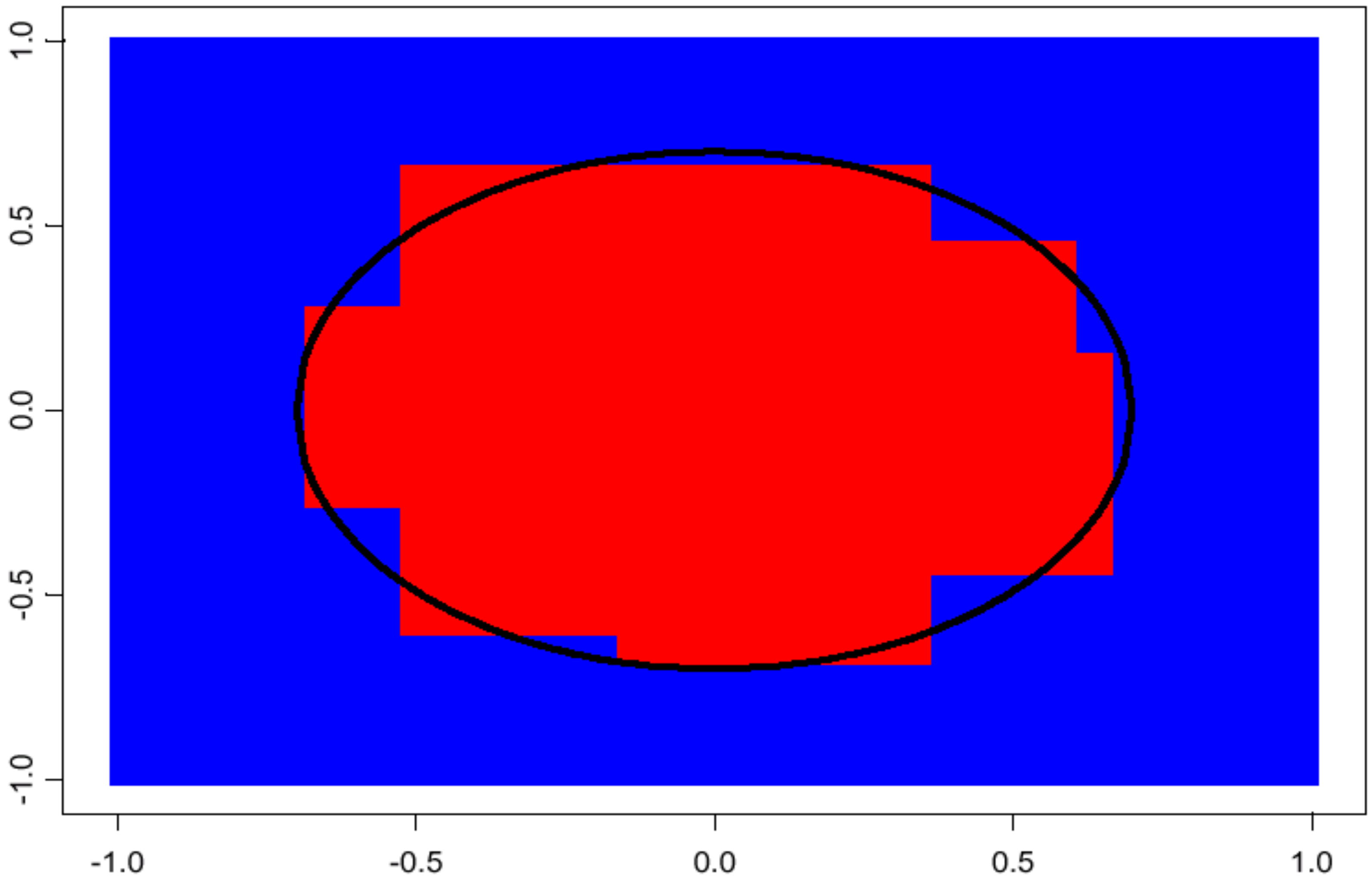
- no closed-form solution
- concave ! global optimum with gradient ascent
- Maximum conditional a posteriori corresponds to regularization

Convergence rates

- GNB (usually) needs less data
- LR (usually) gets to better solutions in the limit


x2
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0
x1

Decision Boundary



Voting (Ensemble Methods)

Instead of learning a single classifier, learn **many weak classifiers** that are **good at different parts of the data**

Output class: (Weighted) vote of each classifier

- Classifiers that are most “sure” will vote with more conviction
- Classifiers will be most “sure” about a particular part of the space
- On average, do better than single classifier!

But how???

- force classifiers to learn about different parts of the input space? different subsets of the data?
- weigh the votes of different classifiers?

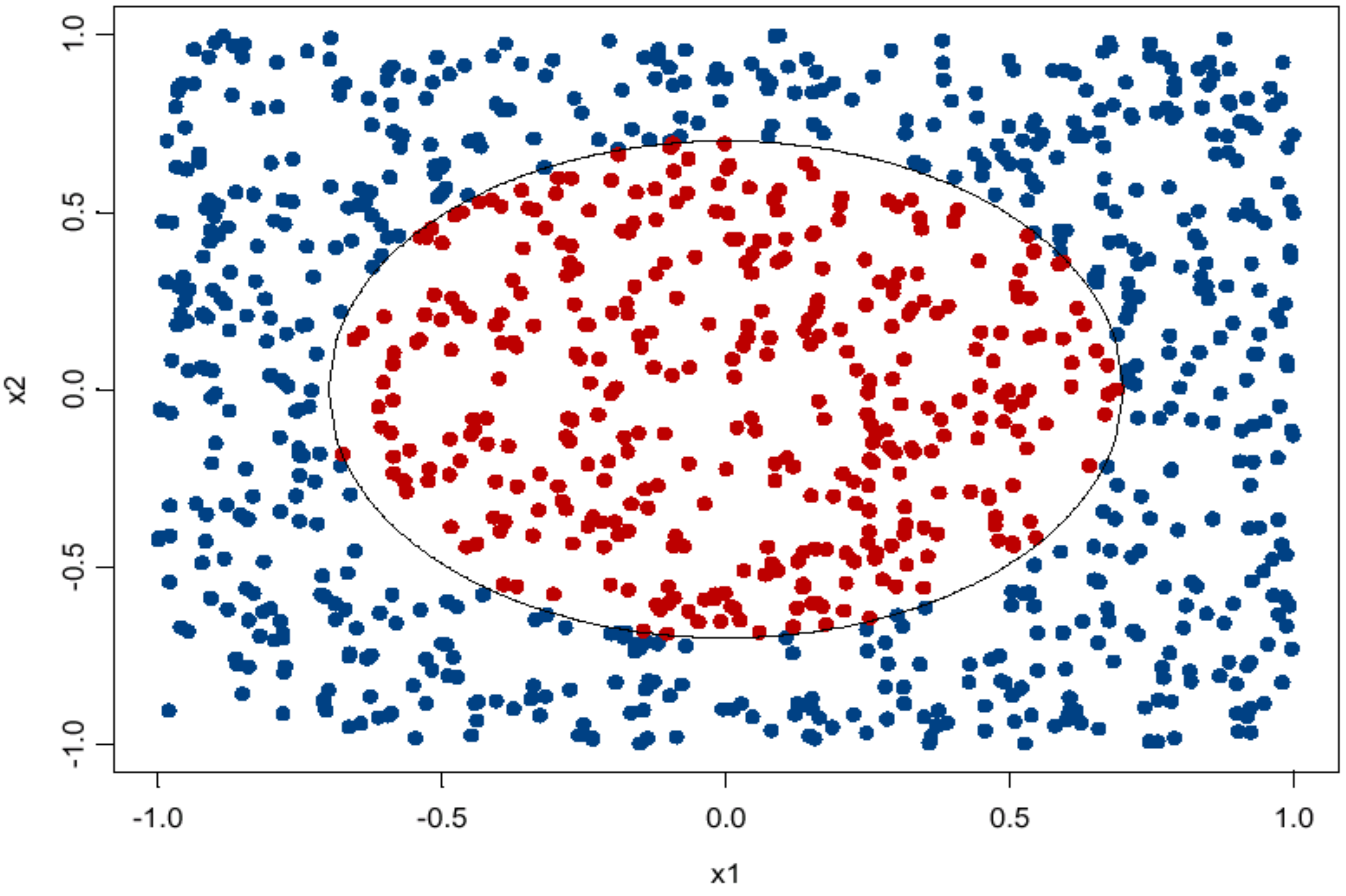
BAGGing = Bootstrap AGGregation

(Breiman, 1996)

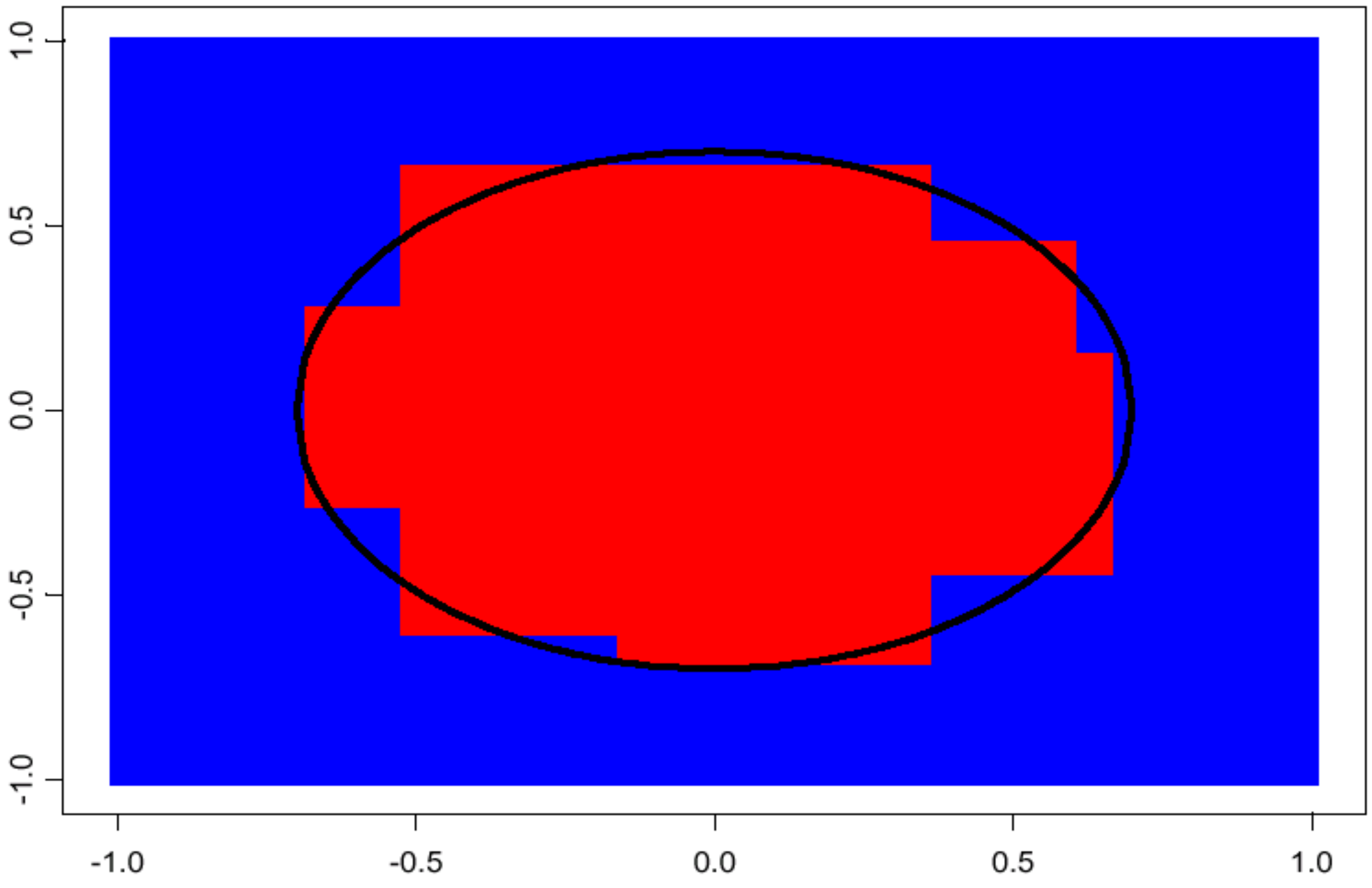
- for $i = 1, 2, \ldots, K$:
  - $T_i \leftarrow$ randomly select M training instances with replacement
  - $h_i \leftarrow \text{learn}(T_i)$ *[ID3, NB, kNN, neural net, …]*

- Now combine the T_i together with uniform voting ($w_i=1/K$ for all i)

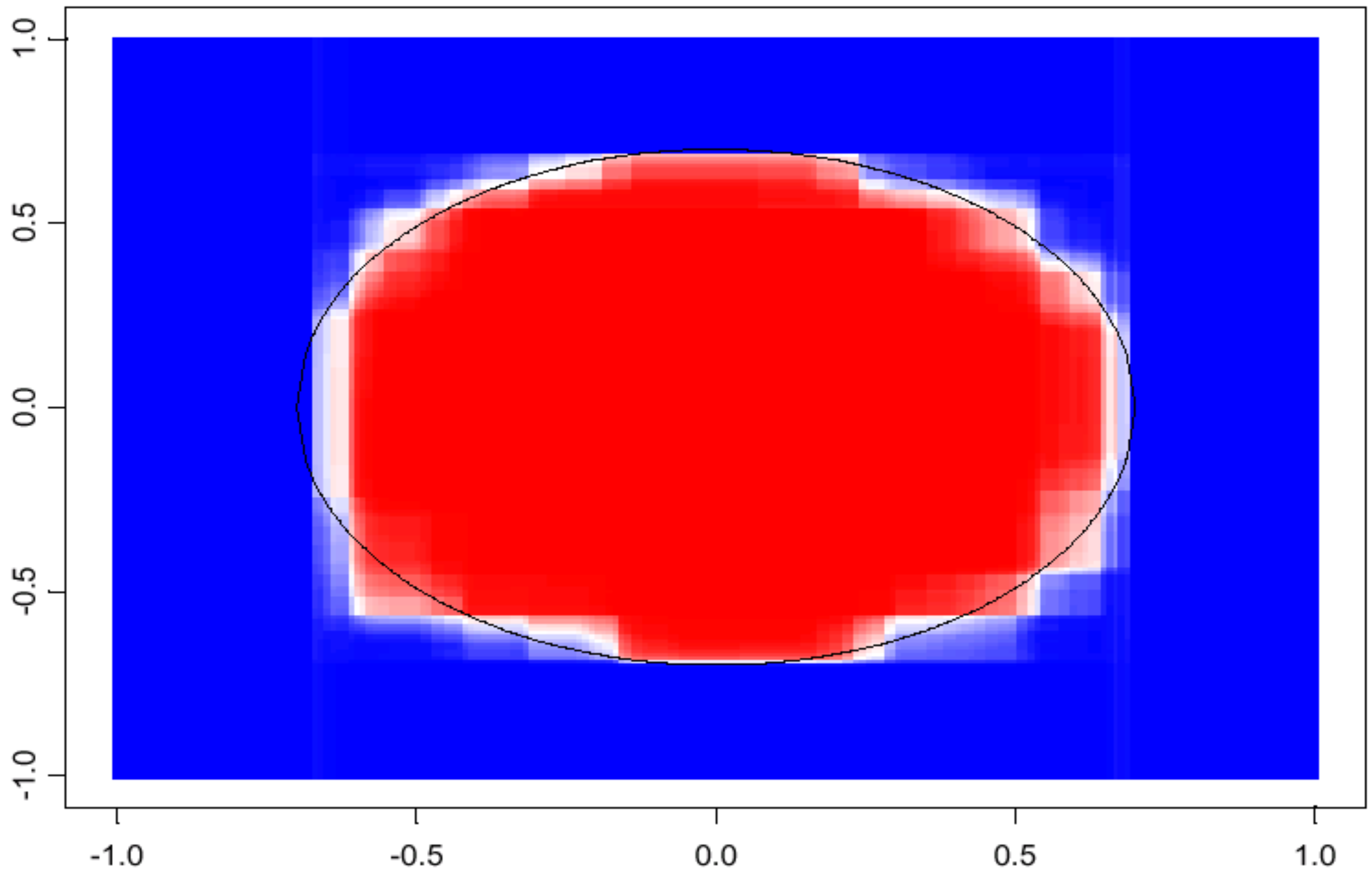
Bagging Example



Decision Boundary



100 bagged trees



shades of blue/red indicate strength of vote for particular classification

Fighting the bias-variance tradeoff

Simple (a.k.a. weak) learners are good

- e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
- Low variance, don’t usually overfit

Simple (a.k.a. weak) learners are bad

- High bias, can’t solve hard learning problems

Can we make weak learners always good???

- **No!!!**
- **But often yes…**

Boosting

[Schapire, 1989]

Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote

On each iteration *t*:

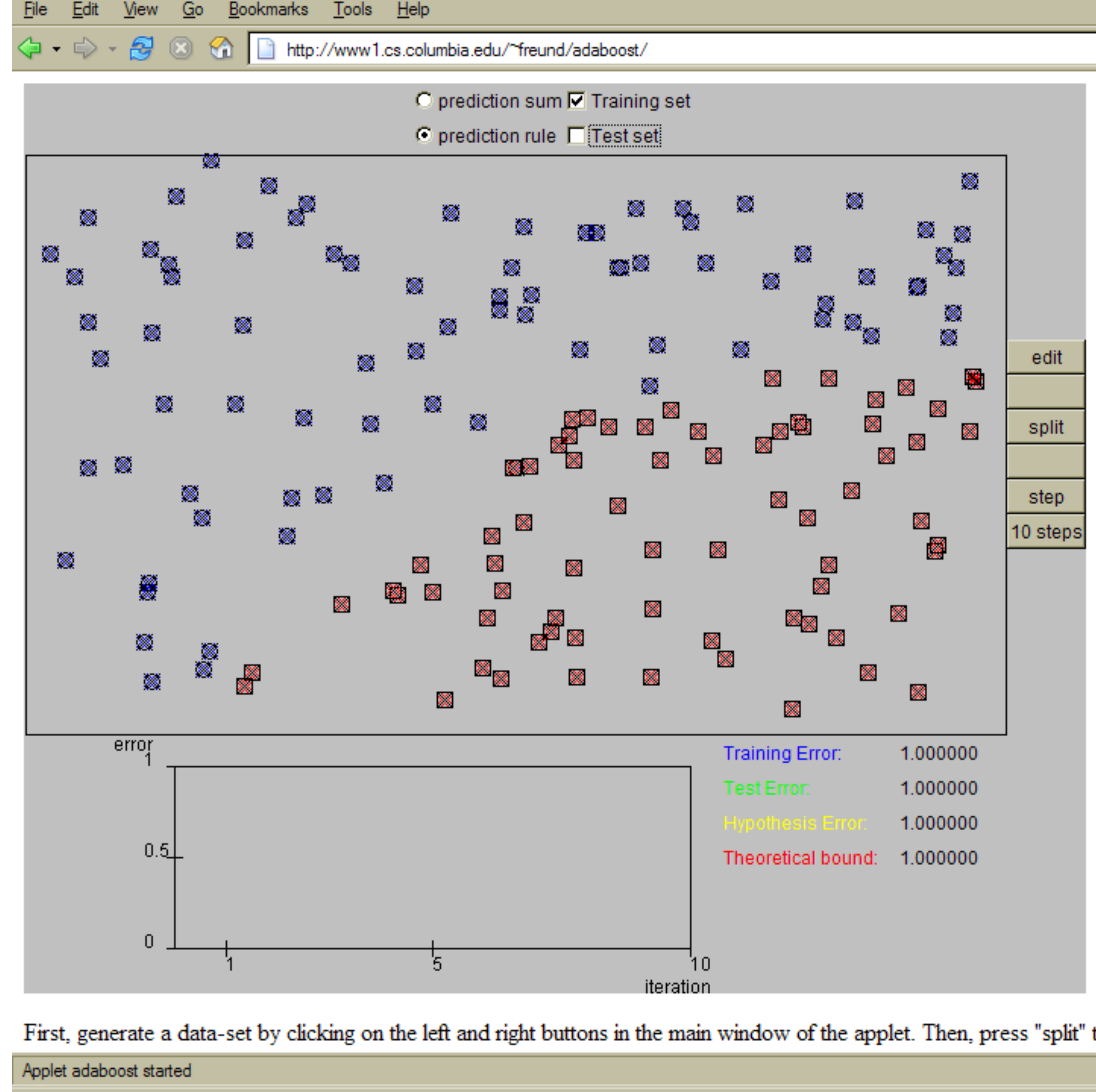
- weight each training example by how incorrectly it was classified
- Learn a hypothesis – h_t
- A strength for this hypothesis – α_t

Final classifier:

$$\boxed{h(x) = \text{sign}\left(\sum_i \alpha_i h_i(x)\right)}$$

Practically useful

Theoretically interesting



Training Error: 1.000000
Test Error: 1.000000
Hypothesis Error: 1.000000
Theoretical bound: 1.000000

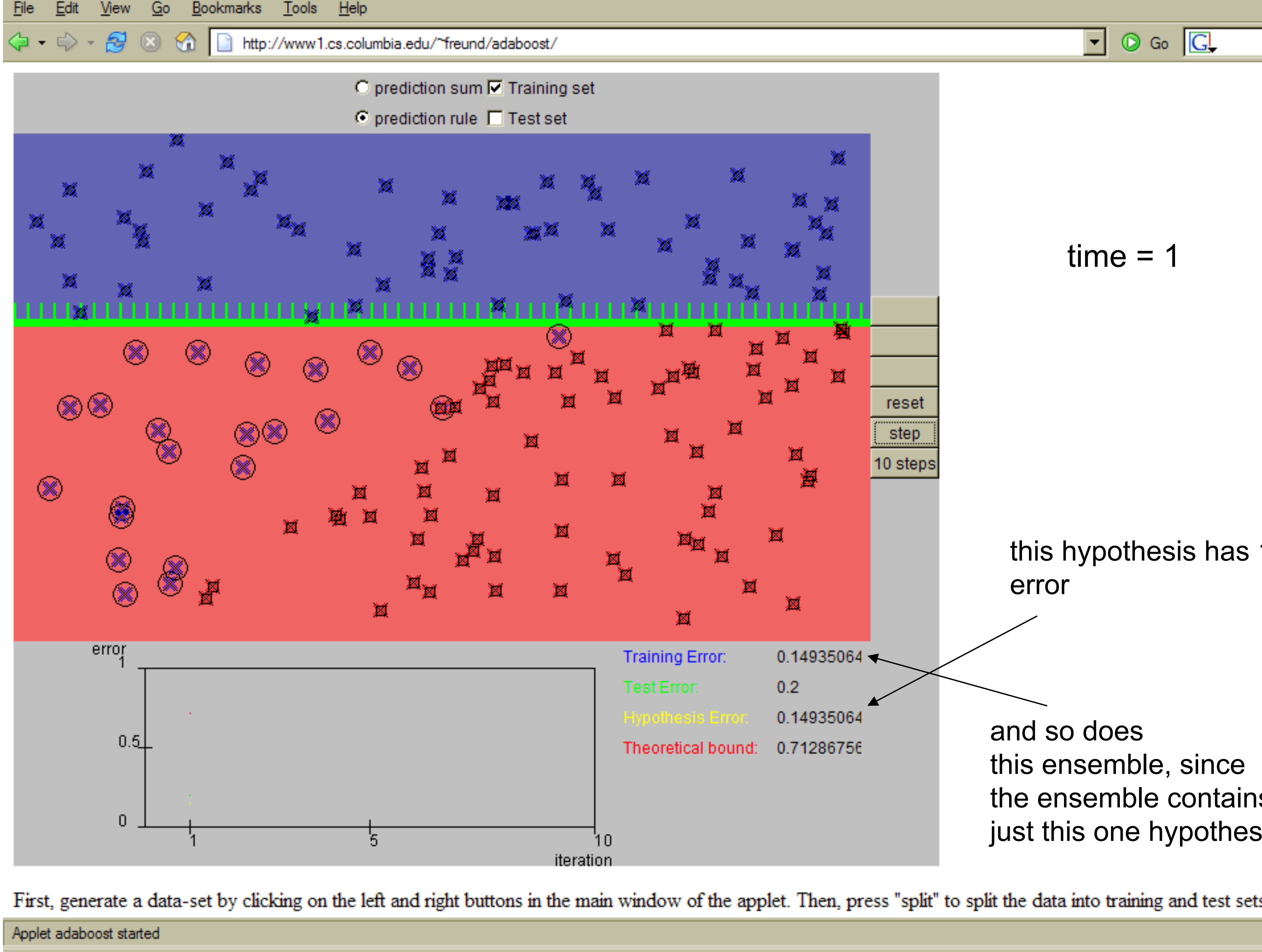
time = 0

blue/red = class

size of dot = weight

weak learner = Decision stub: horizontal or vertica

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets



Training Error: 0.14935064

Test Error: 0.2

Hypothesis Error: 0.14935064

Theoretical bound: 0.71286756

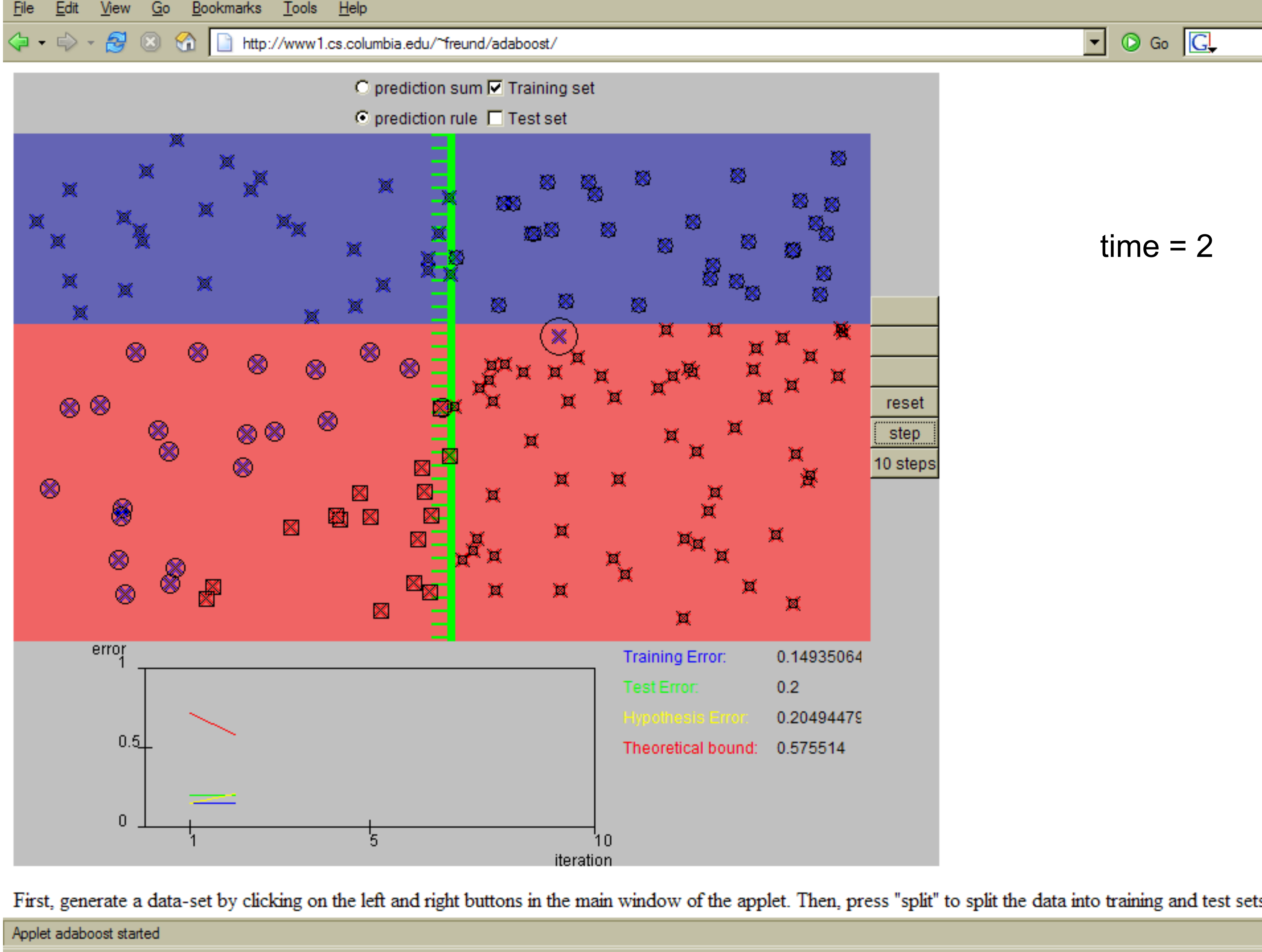
First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets

Applet adaboost started

time = 1

this hypothesis has
error

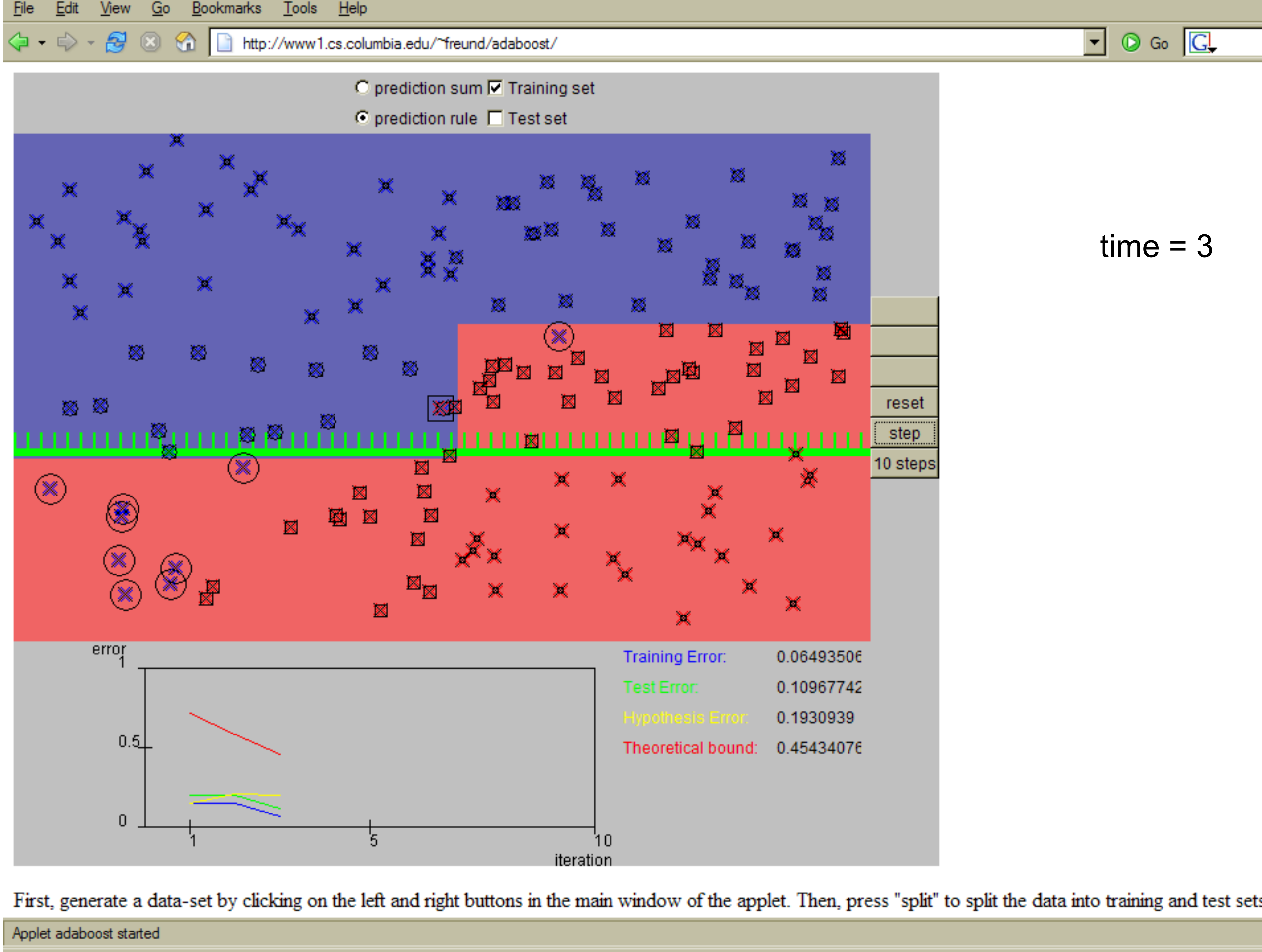
and so does
this ensemble, since
the ensemble contains
just this one hypothes



Training Error: 0.14935064
Test Error: 0.2
Hypothesis Error: 0.20494479
Theoretical bound: 0.575514

time = 2

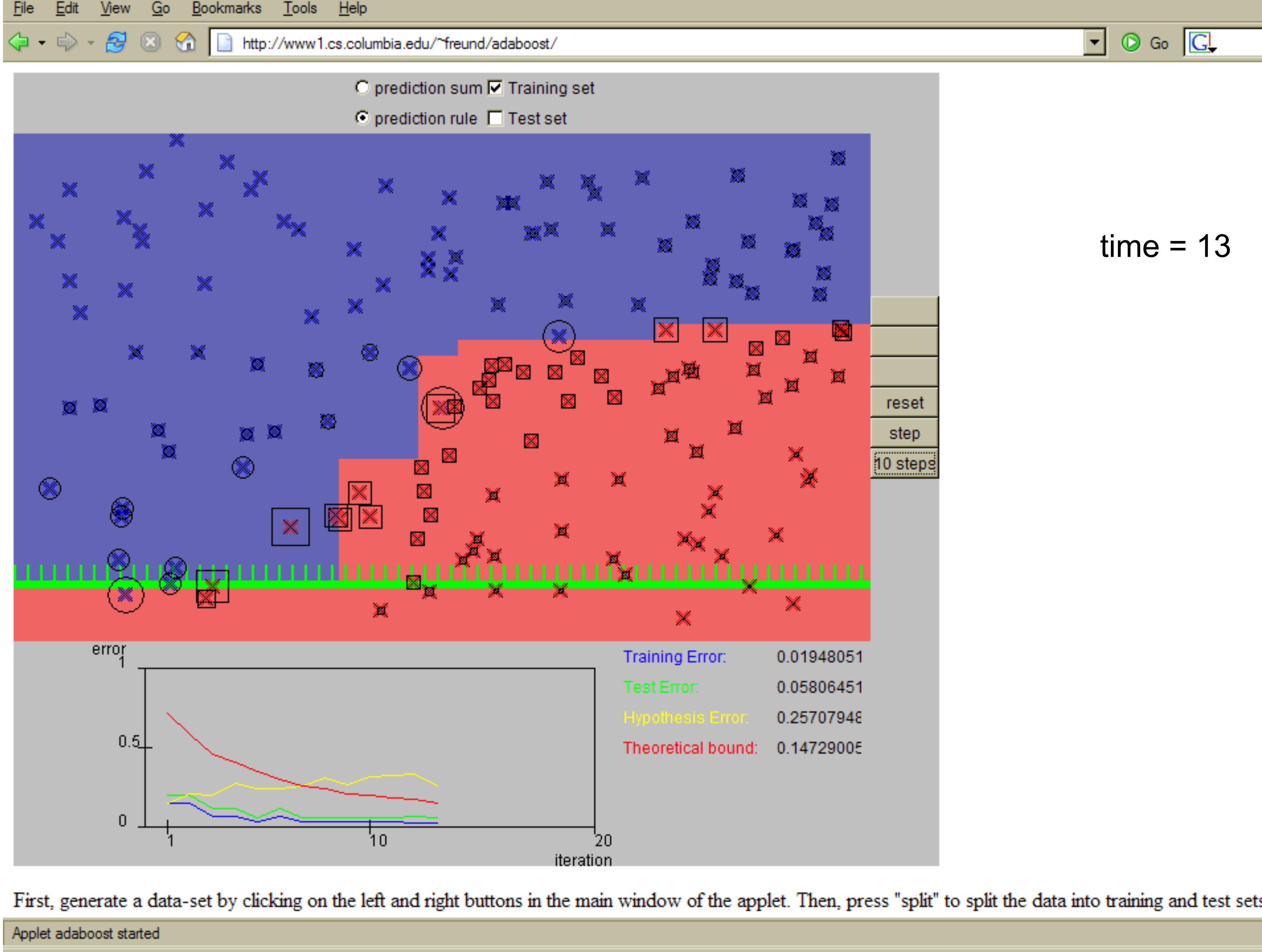
First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets



Training Error:	0.06493506
Test Error:	0.10967742
Hypothesis Error:	0.1930939
Theoretical bound:	0.45434076

time = 3

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets



prediction sum [✓ Training set]

[✓ prediction rule] Test set

Training Error: 0.01948051

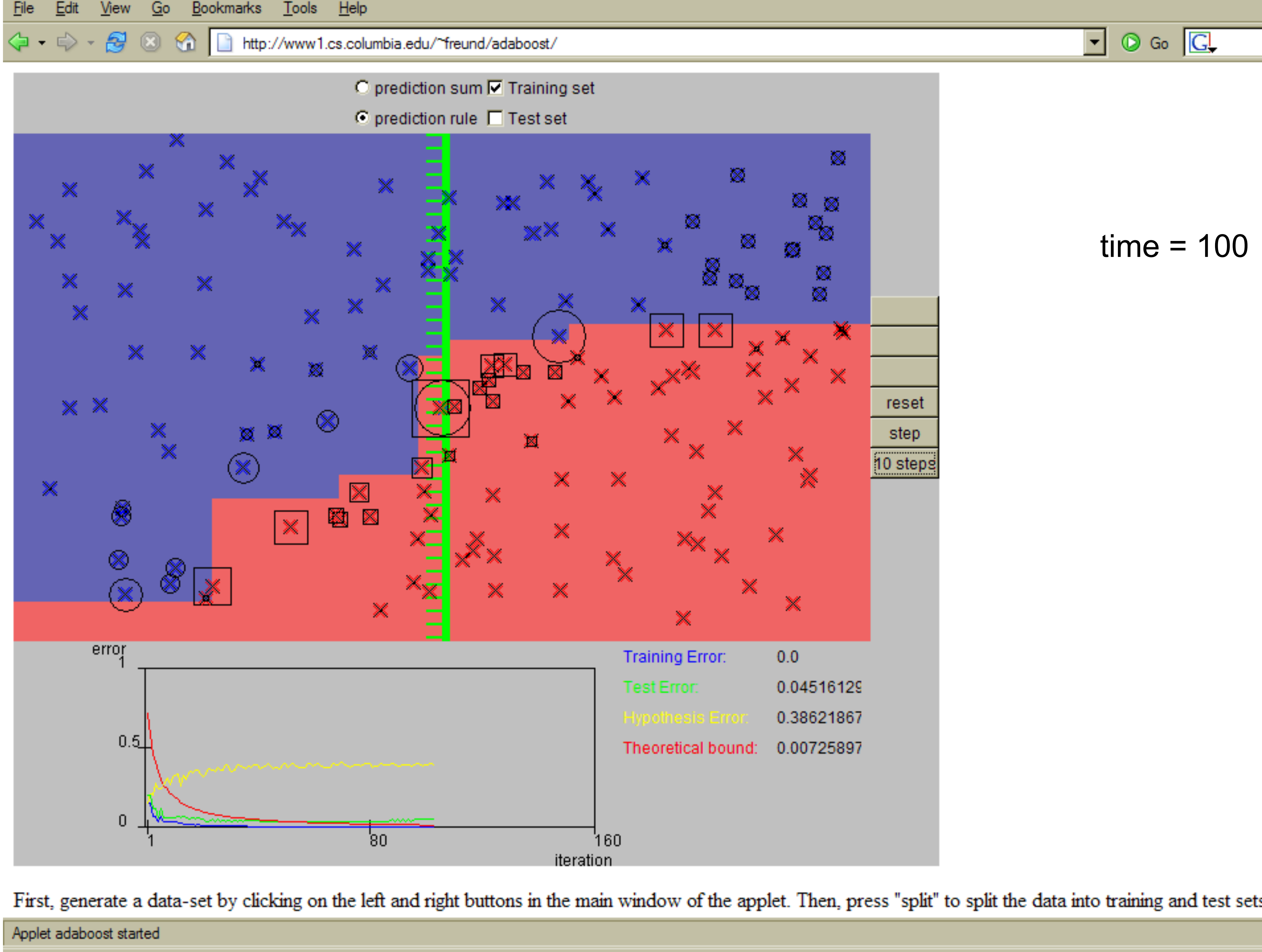
Test Error: 0.05806451

Hypothesis Error: 0.25707948

Theoretical bound: 0.14729005

time = 13

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets

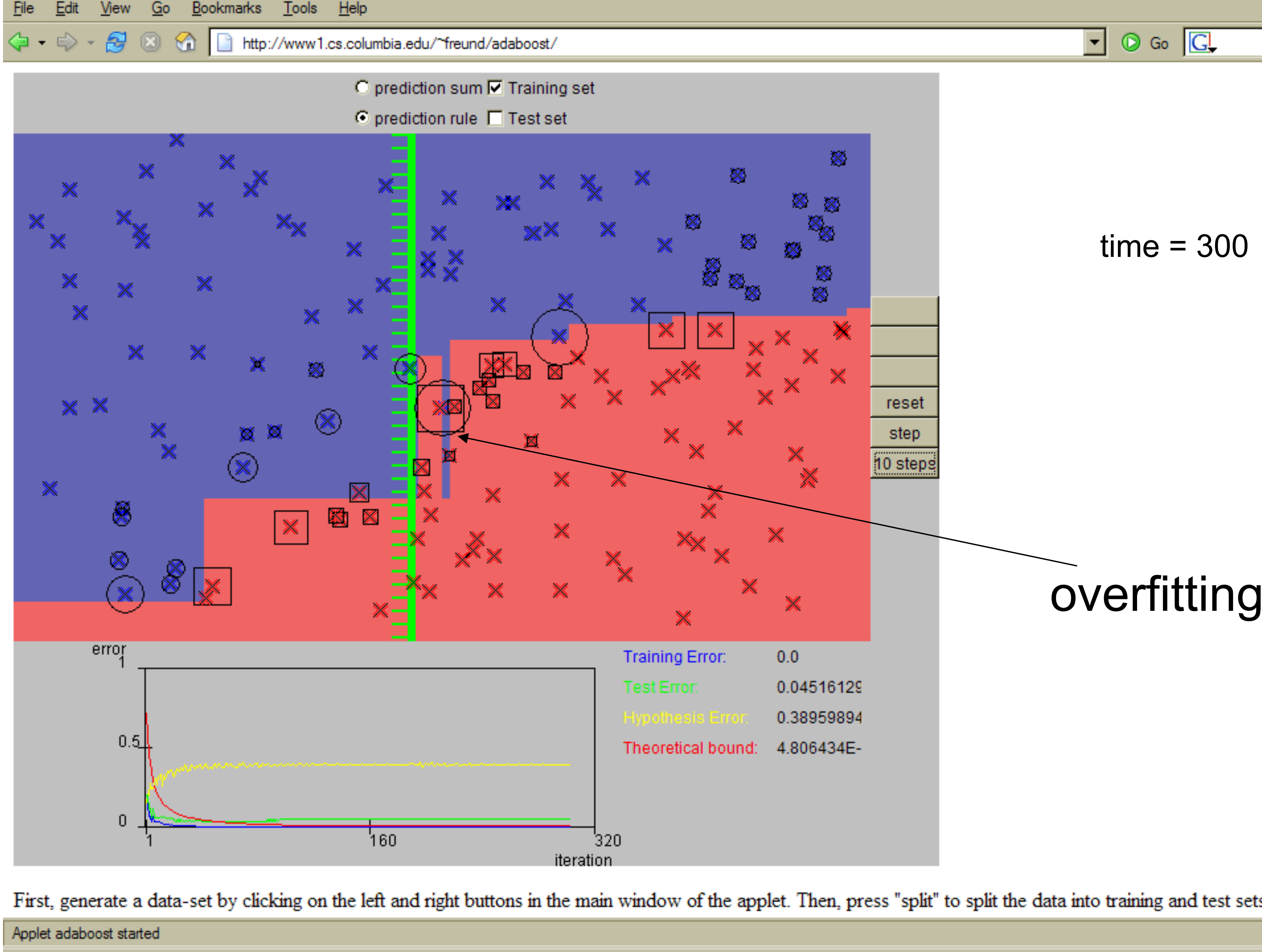


Training Error: 0.0
Test Error: 0.04516129
Hypothesis Error: 0.38621867
Theoretical bound: 0.00725897

time = 100

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets

Applet adaboost started



time = 300

overfitting

Training Error: 0.0

Test Error: 0.04516129

Hypothesis Error: 0.38959894

Theoretical bound: 4.806434E-

First, generate a data-set by clicking on the left and right buttons in the main window of the applet. Then, press "split" to split the data into training and test sets

Applet adaboost started

Learning from weighted data

Consider a weighted dataset

- D(i) – weight of *i* th training example ($\mathbf{x}^i$,y^i)
- Interpretations:
  - *i* th training example counts as if it occurred D(i) times
  - If I were to "resample" data, I would get more samples of "heavier" data points

Now, always do weighted calculations:

- e.g., MLE for Naïve Bayes, redefine *Count(Y=y)* to be weighted count:

$$Count(Y=y) = \sum_{j=1}^{n} D(j)\delta(Y^j = y)$$

- setting D(j)=1 (or any constant value!), for all j, will recreates unweighted case

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

How? Many possibilities. Will see one shortly!

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Final Result: linear sum of "base" or "weak" classifier outputs.

Figure 1: The boosting algorithm AdaBoost.

$$\boxed{\epsilon_t = P_{i\sim D_t(i)}[h_t(\mathbf{x}^i) \neq y^i]}$$

$$\boxed{\epsilon_t = \sum_{i=1}^{m} D_t(i)\delta(h_t(x_i) \neq y_i)}$$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$. $\longleftarrow$ $\boxed{\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$
- Update:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 1: The boosting algorithm AdaBoost.

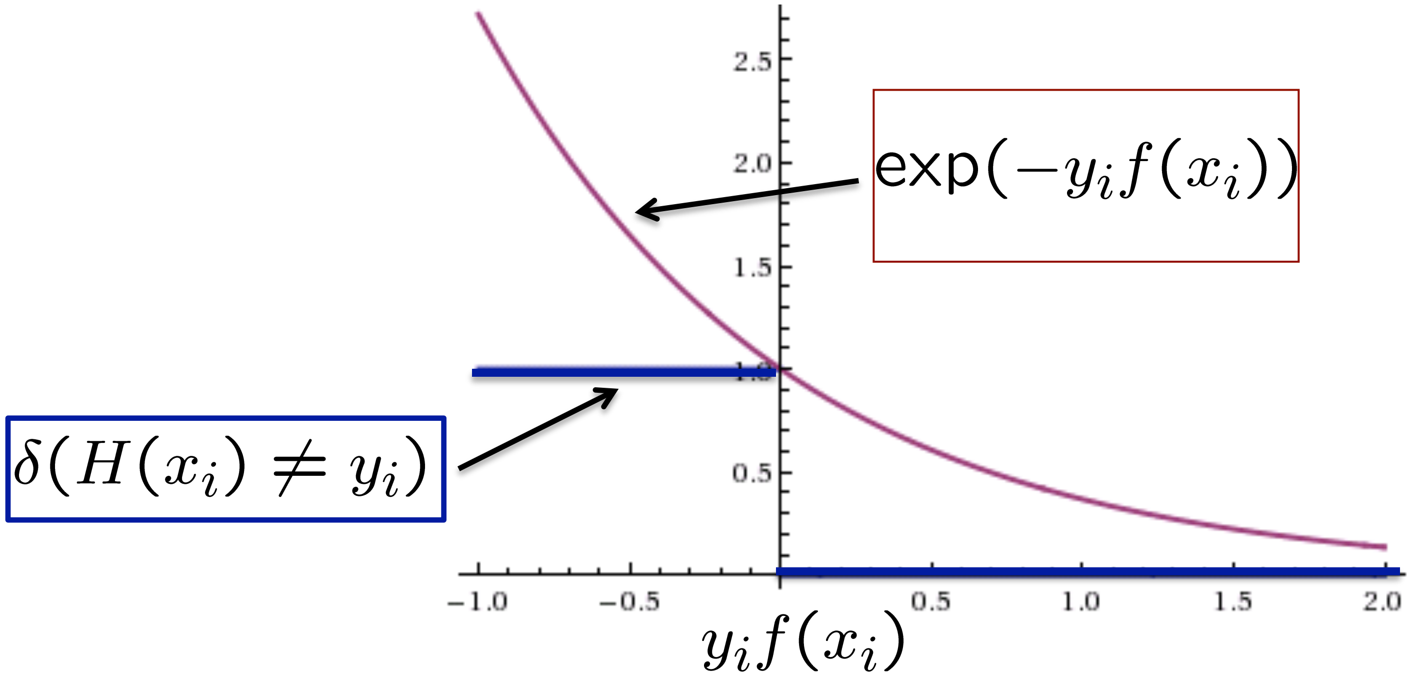
What α_t to choose for hypothesis h_t?

[Schapire, 1989]

Idea: choose α_t to minimize a bound on training error!

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i f(x_i))$$

Where $$f(x) = \sum_t \alpha_t h_t(x);\, H(x) = sign(f(x))$$



What α_t to choose for hypothesis h_t?

[Schapire, 1989]

Idea: choose α_t to minimize a bound on training error!

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i)\leq\frac{1}{m}\sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

Where

$$f(x)=\sum_t \alpha_t h_t(x);\ H(x)=sign(f(x))$$

And

$$Z_t=\sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

This equality isn't obvious! Can be shown with algebra (telescoping sums)!

If we minimize $\prod_t Z_t$, we minimize our training error!!!

We can tighten this bound greedily, by choosing α_t and h_t on each iteration to minimize Z_t.

h_t is estimated as a black box, but can we solve for α_t?

Summary: choose α_t to minimize *error bound*

[Schapire, 1989]

We can squeeze this bound by choosing α_t on each iteration to minimize Z_t.

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)\delta(h_t(x_i) \neq y_i)$$

For boolean Y: differentiate, set equal to 0, there is a closed form solution! [Freund & Schapire '97]:

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

Strong, weak classifiers

If each classifier is (at least slightly) better than random: $\varepsilon_t < 0.5$

Another bound on error:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \prod_t Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right)$$

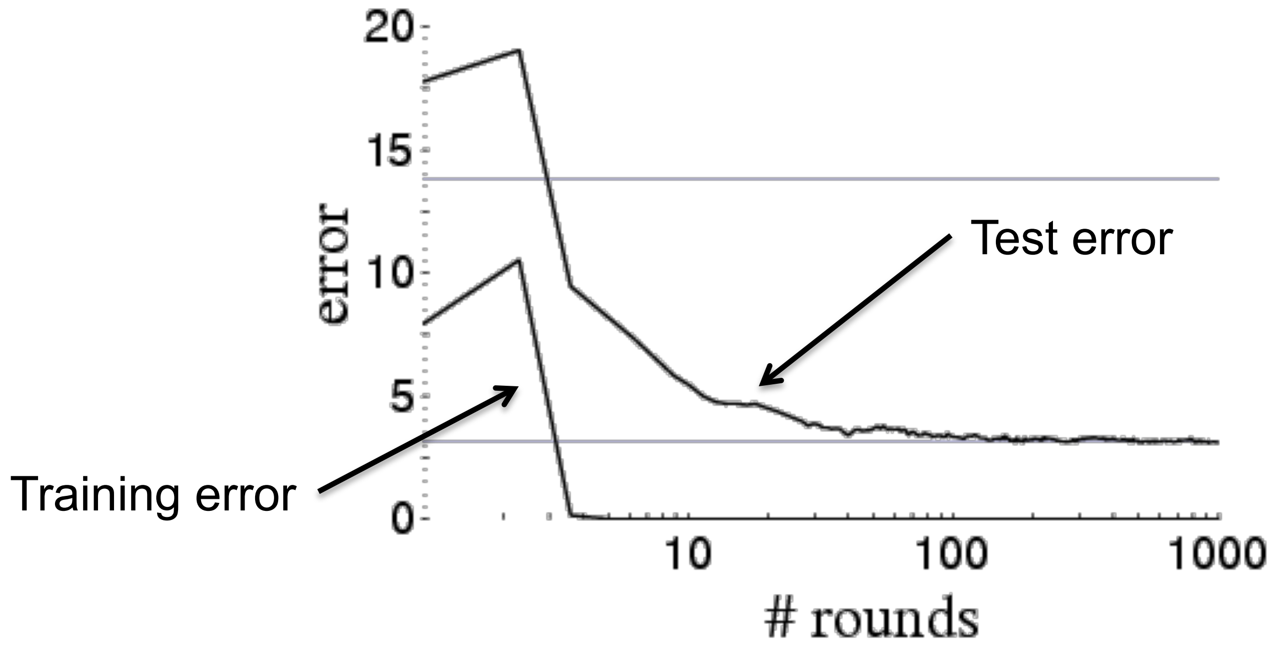
What does this imply about the training error?

- Will get there exponentially fast!

Is it hard to achieve better than random training error?

Boosting results – Digit recognition

[Schapire, 1989]



Boosting:

- Seems to be robust to overfitting
- Test error can decrease even after training error is zero!!!

Boosting generalization error bound

[Freund & Schapire, 1996]

$$error_{true}(H) \quad \leq \quad error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

Constants:

T: number of boosting rounds

- Higher T → Looser bound, *what does this imply?*

d: VC dimension of weak learner, measures complexity of classifier

- Higher d → bigger hypothesis space → looser bound

m: number of training examples

- more data → tighter bound

Boosting generalization error bound

[Freund & Schapire, 1996]

$$error_{true}(H) \quad \leq \quad error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

Constants:

- **Theory does not match practice**:
  - Robust to overfitting
  - Test set error decreases even after training error is zero
- **Need better analysis tools**
  - we'll come back to this later in the quarter

  - more data → tighter bound

Logistic Regression as Minimizing Loss

Logistic regression assumes:

$$P(Y=1|X) = \frac{1}{1+\exp(f(x))} \qquad f(x) = w_0 + \sum_i w_i h_i(x)$$

And tries to maximize data likelihood, for Y={-1,+1}:

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+e^{-y_i f(\mathbf{x}_i)}} \qquad \ln P(\mathcal{D}_Y \mid \mathcal{D}_{\mathbf{X}}, \mathbf{w}) = \sum_{j=1}^{N} \ln P(y^j \mid \mathbf{x}^j, \mathbf{w})$$

$$= -\sum_{i=1}^{m} \ln(1+\exp(-y_i f(x_i)))$$

Equivalent to minimizing *log loss*:

$$\sum_{i=1}^{m} \ln(1+\exp(-y_i f(x_i)))$$

Boosting and Logistic Regression

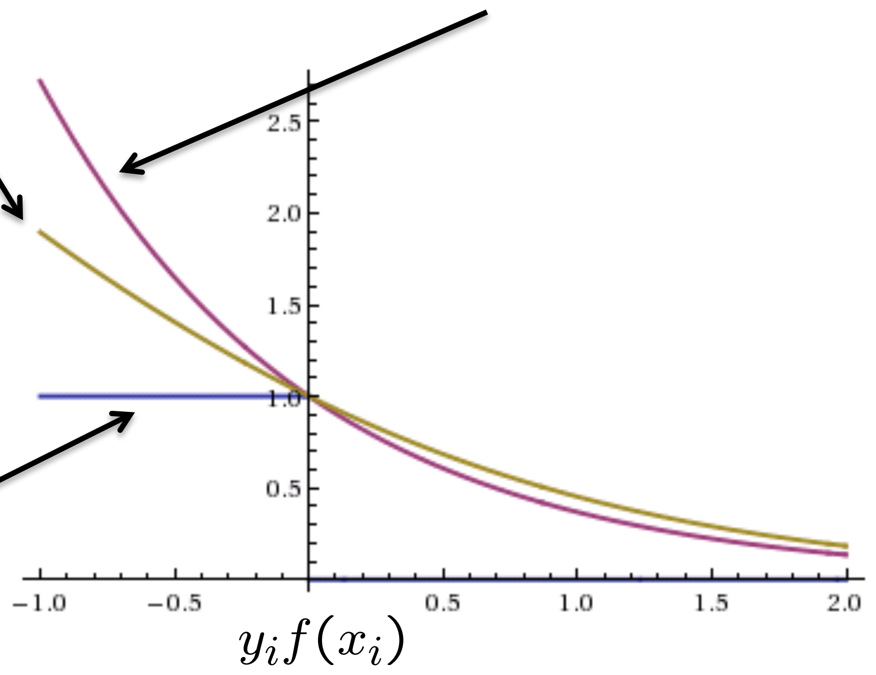
Logistic regression equivalent to minimizing log loss:

$$\boxed{\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))}$$

Boosting minimizes similar loss function:

$$\boxed{\frac{1}{m}\sum_{i} \exp(-y_i f(x_i)) = \prod_{t} Z_t}$$

$$\boxed{\delta(H(x_i) \neq y_i)}$$



Both smooth approximations of 0/1 loss!