Recognition
Part I: Machine Learning

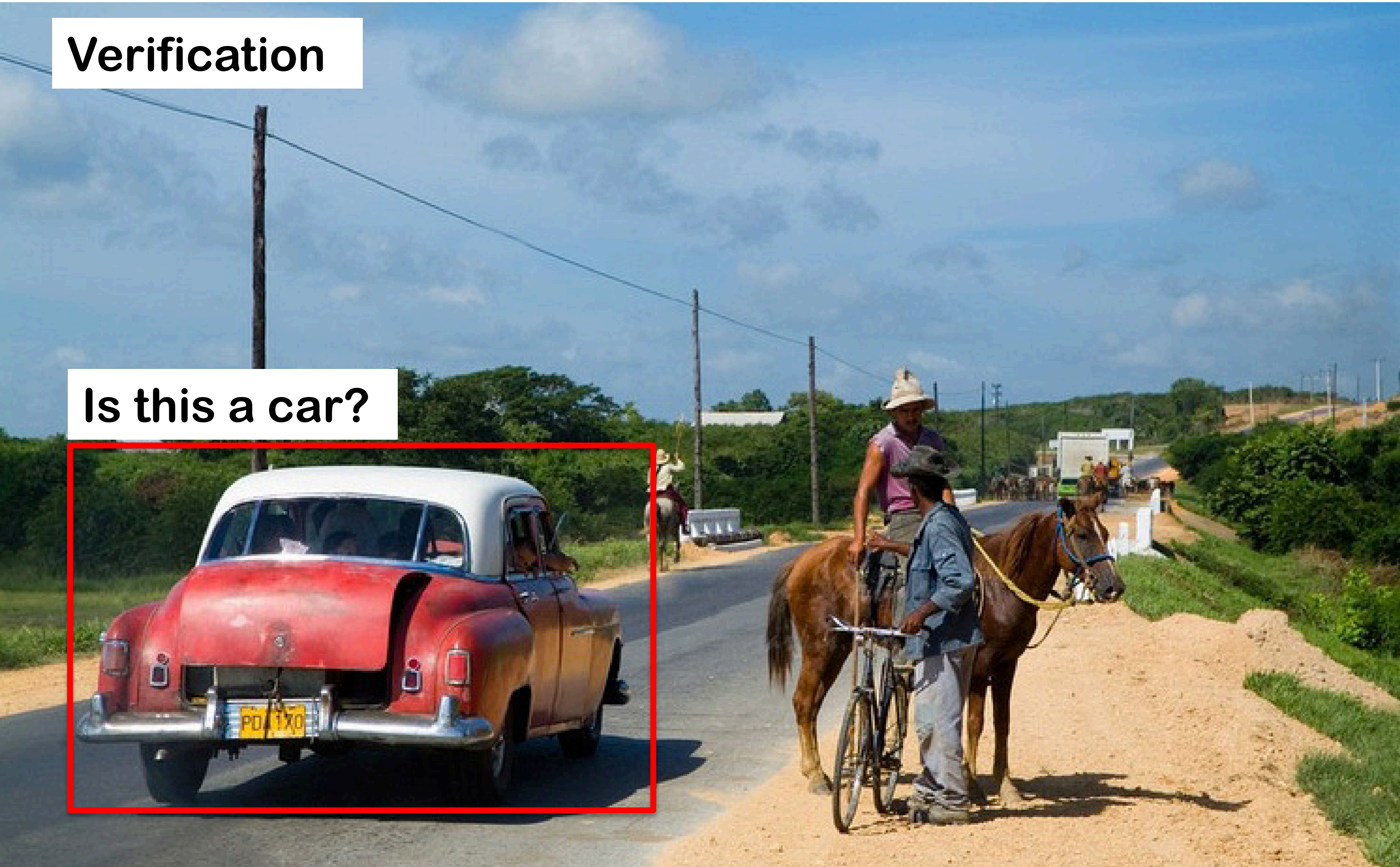
CSE 455

Linda Shapiro

Visual Recognition

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

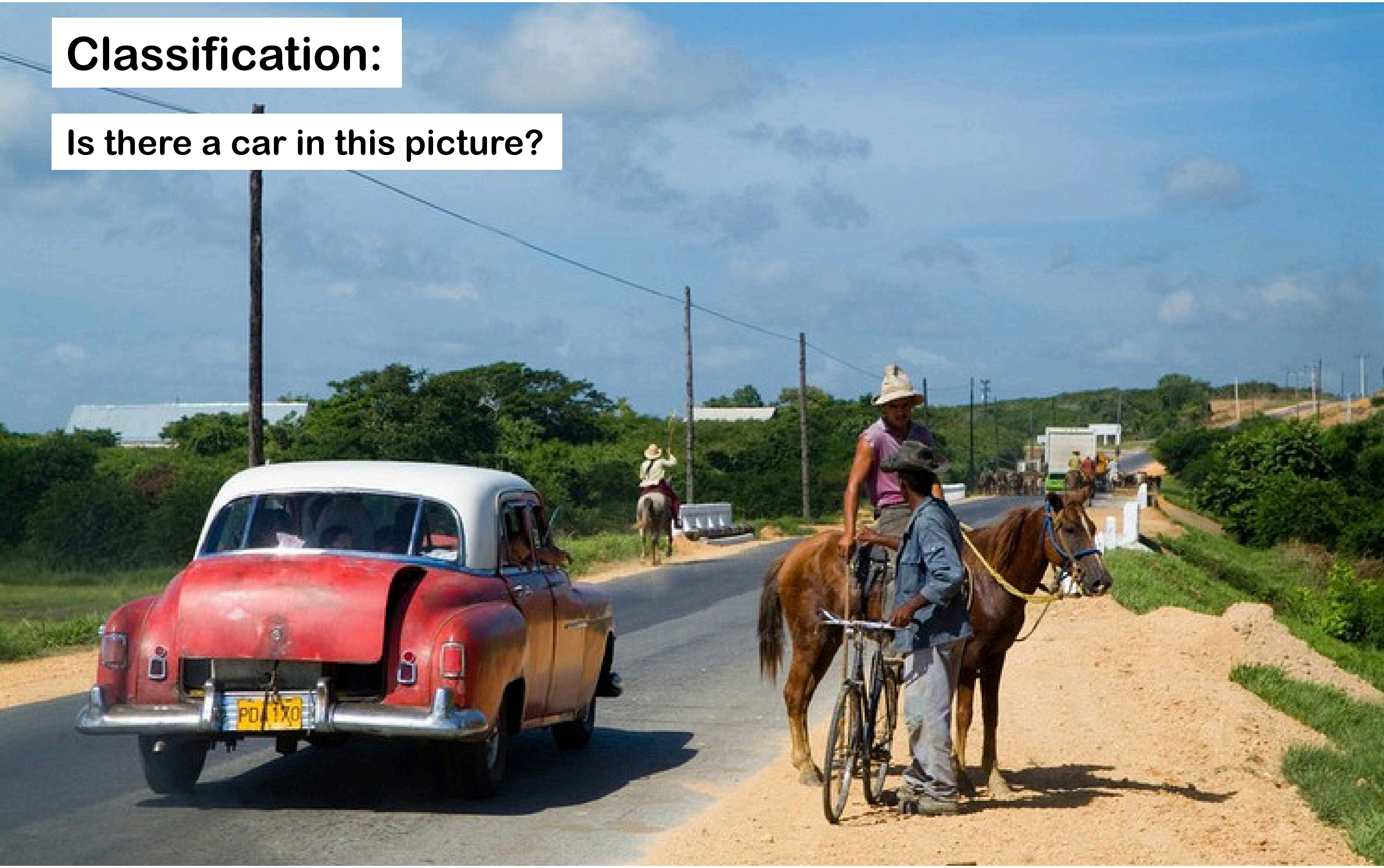
Visual Recognition



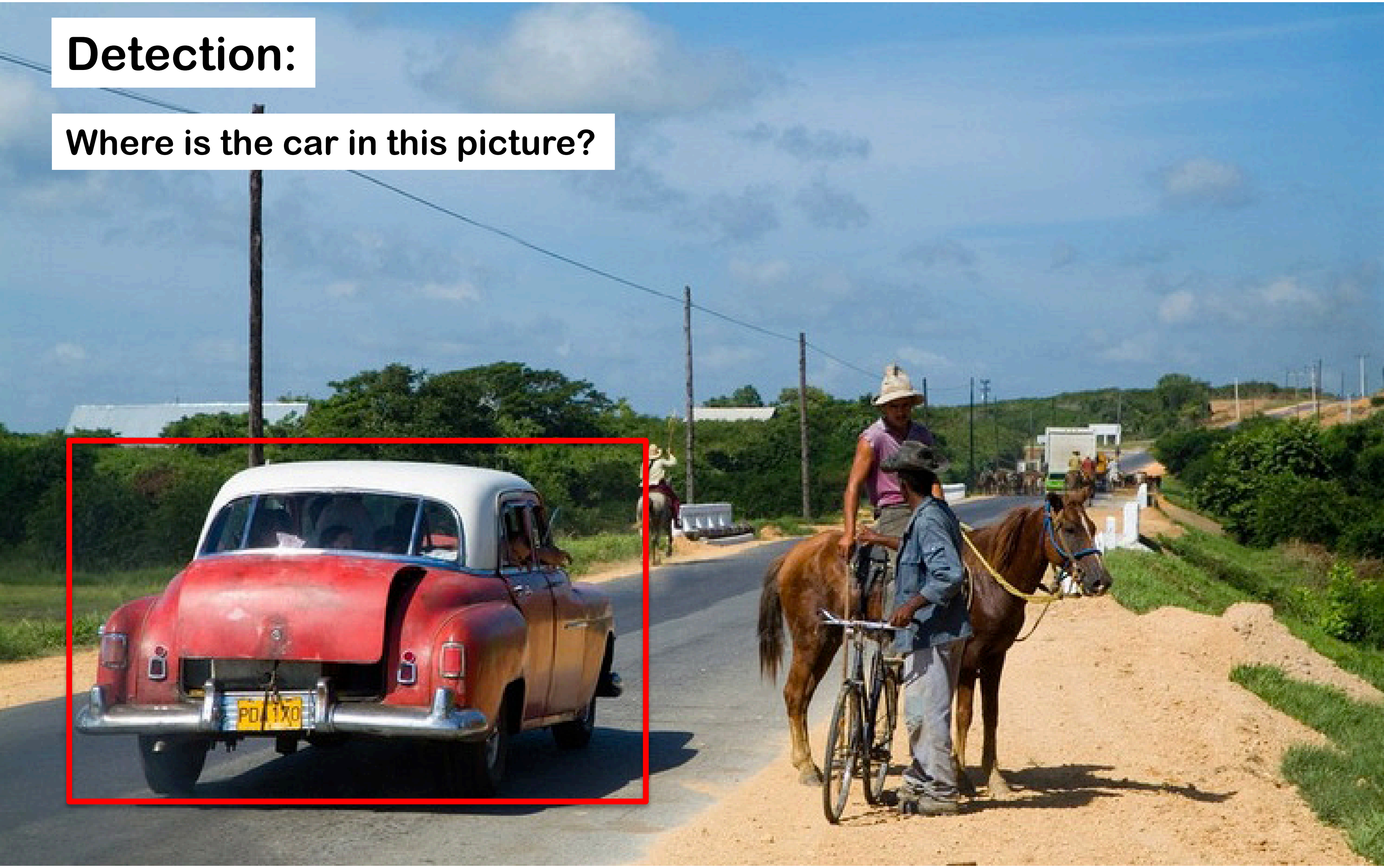
Visual Recognition

Classification:

Is there a car in this picture?

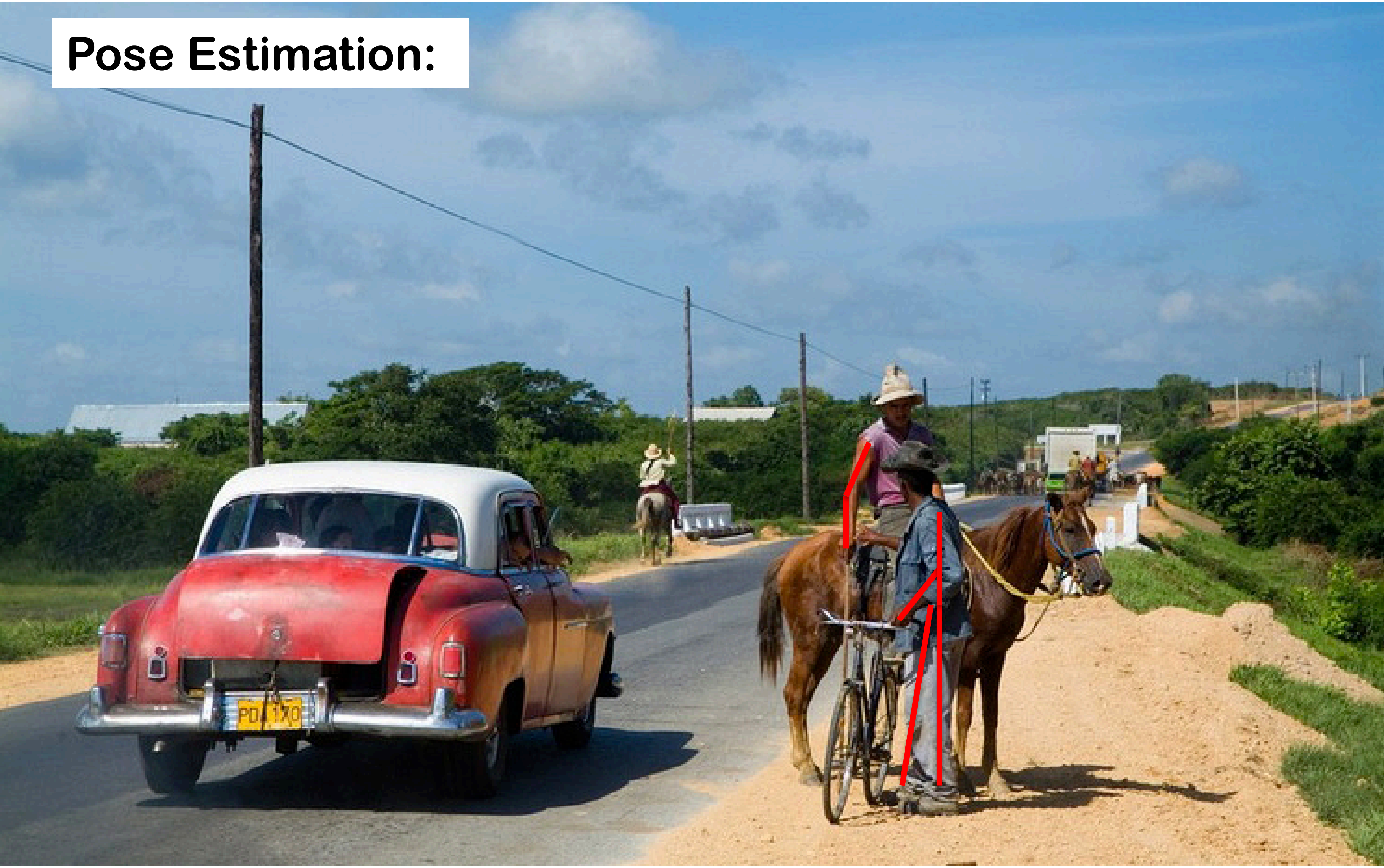


Visual Recognition

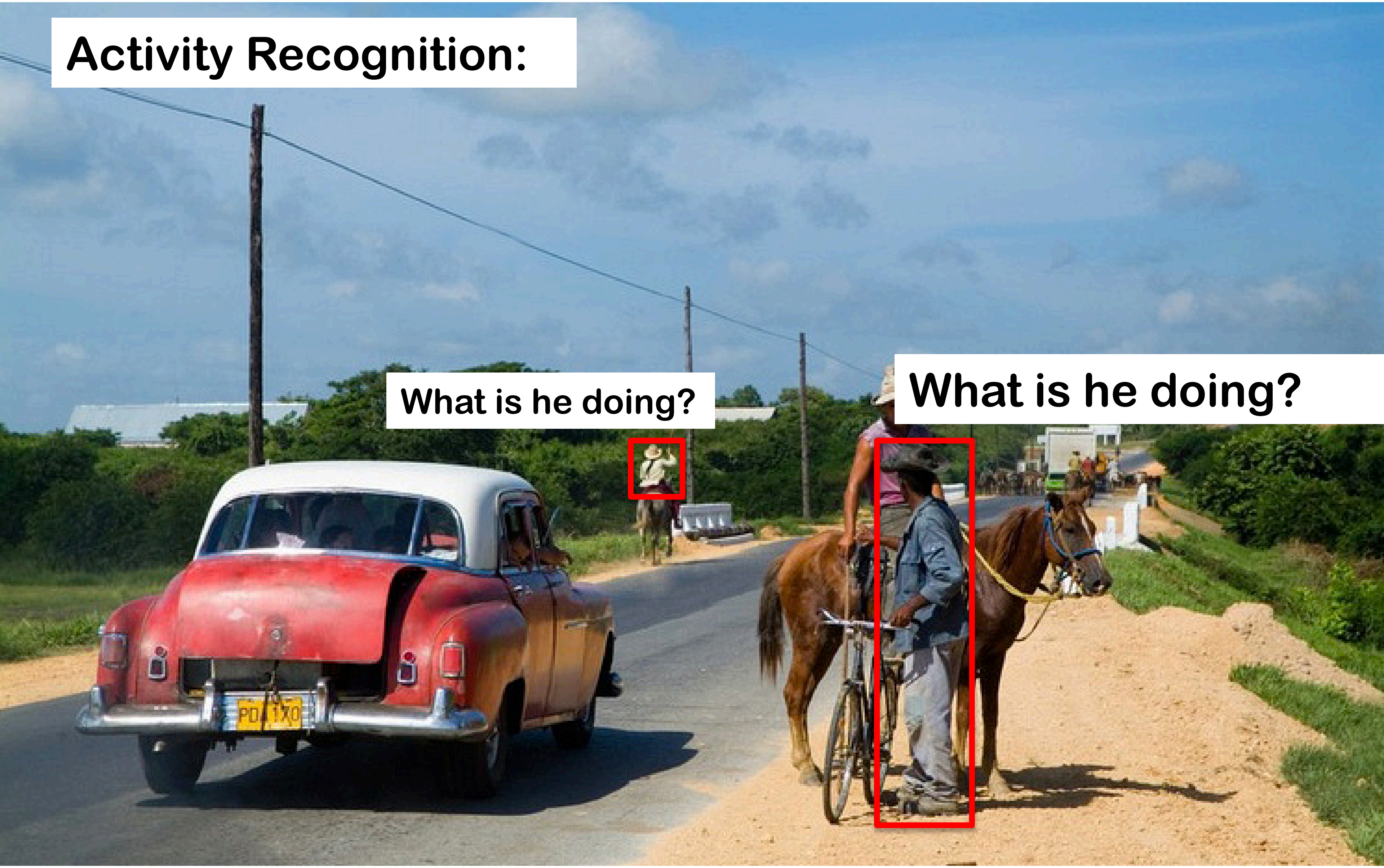


Visual Recognition

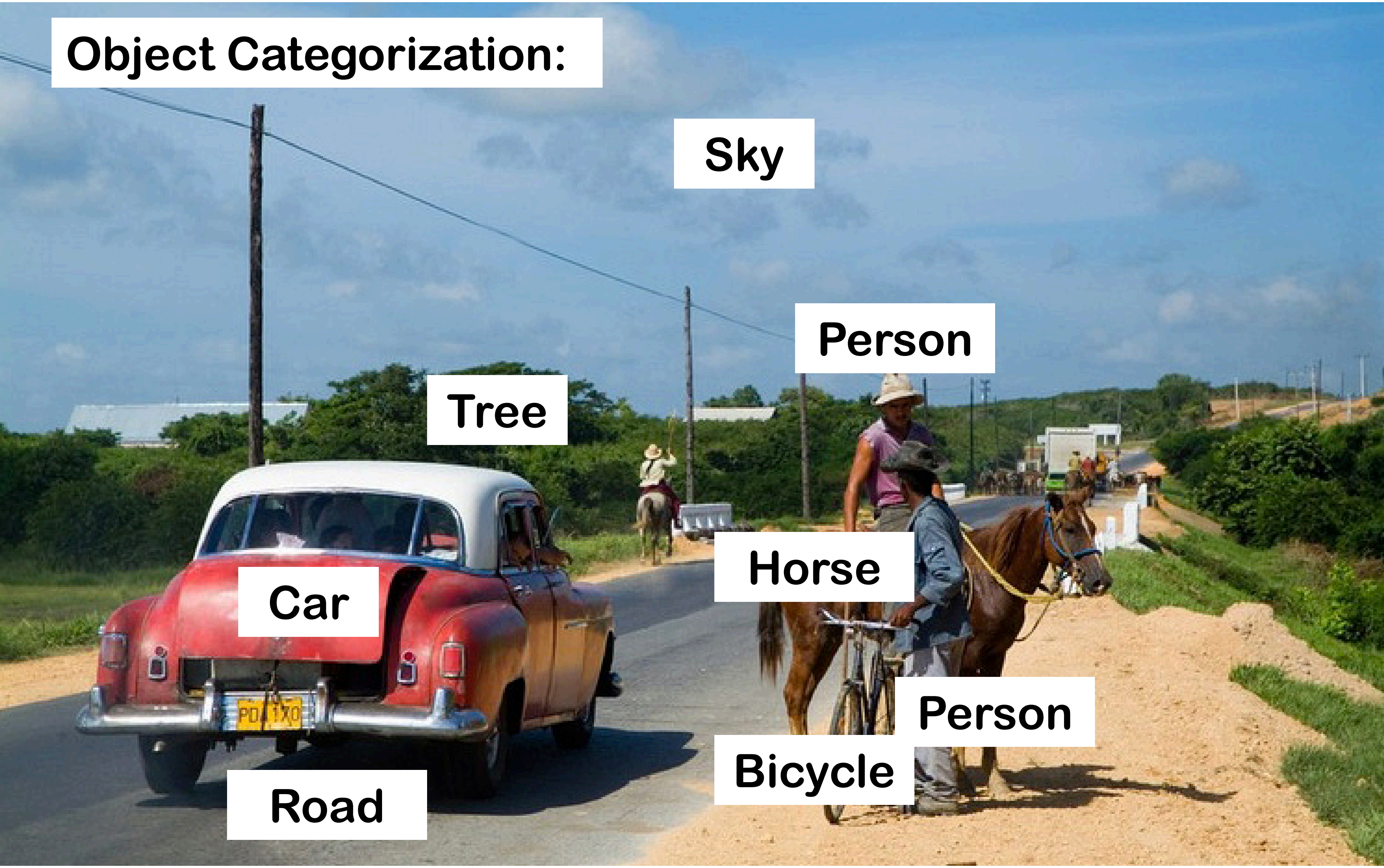
Pose Estimation:



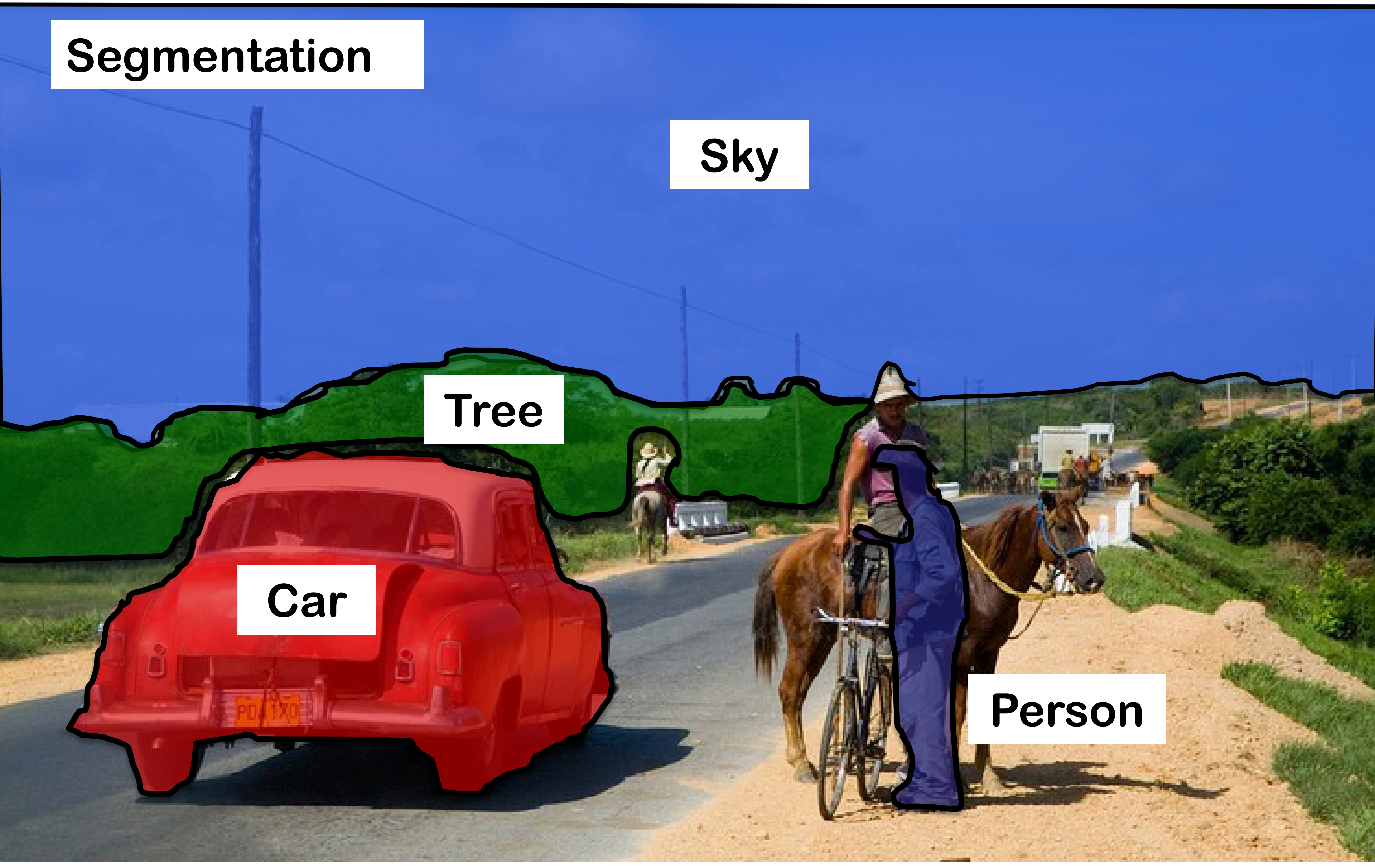
Visual Recognition



Visual Recognition



Visual Recognition

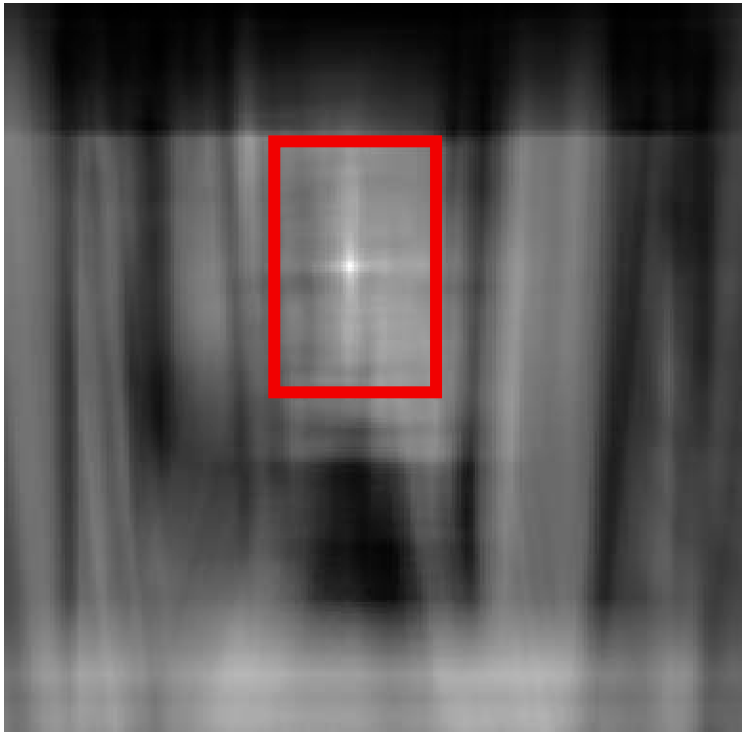


Object recognition
Is it really so hard?

This is a chair

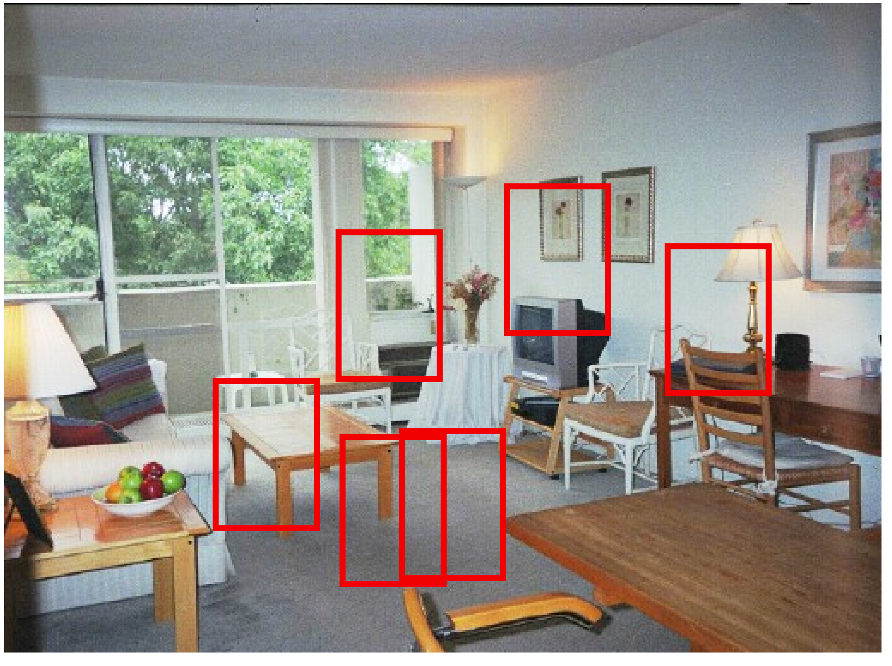
Find the chair in this image

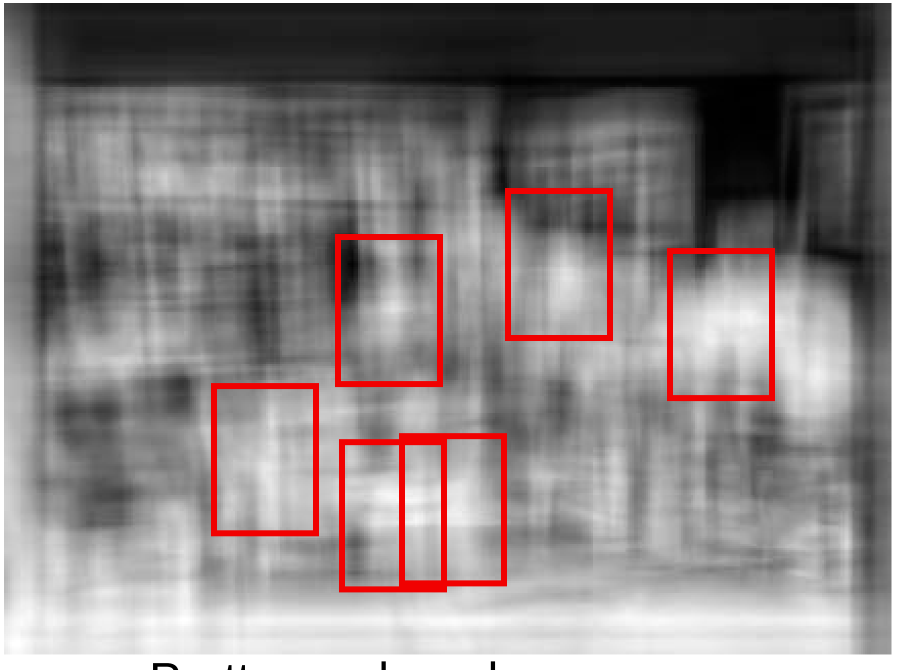
Output of normalized correlation



Object recognition
Is it really so hard?

Find the chair in this image





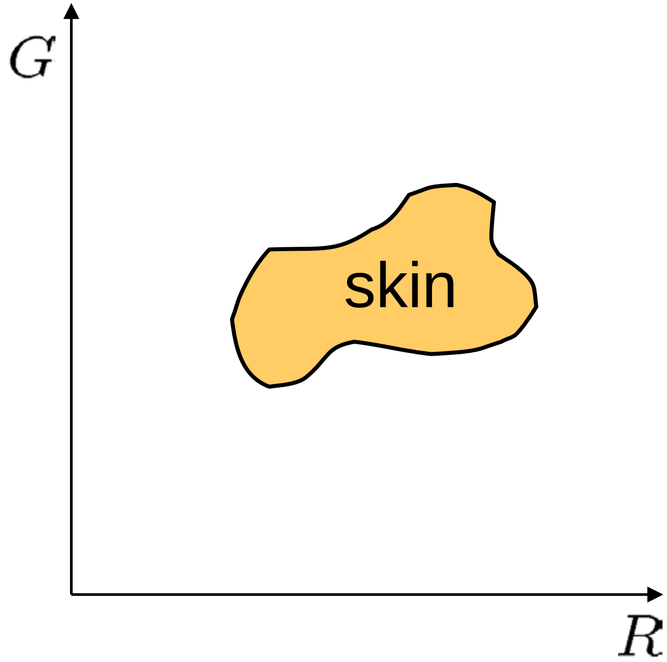
Pretty much garbage

Simple template matching is not going to make it

Let’s start with
Face detection

How to tell if a face is present?

One simple method: skin detection



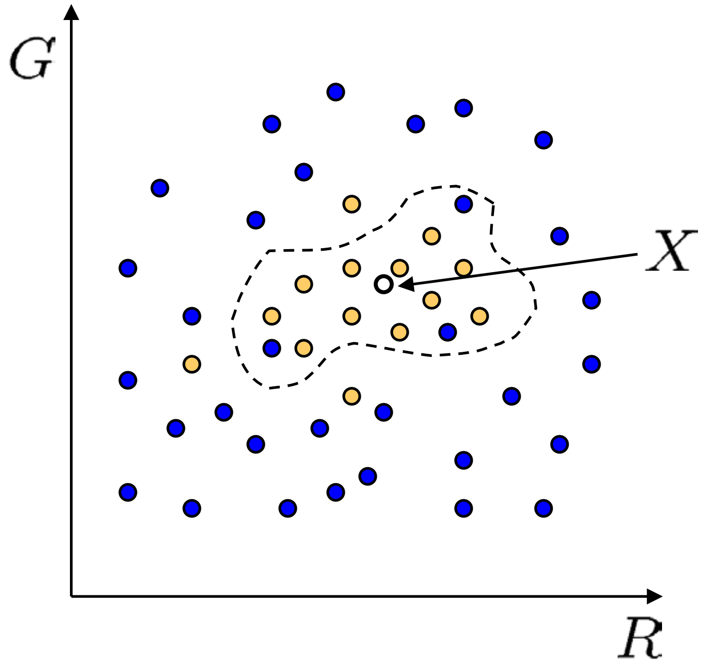
Skin pixels have a distinctive range of colors

- 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



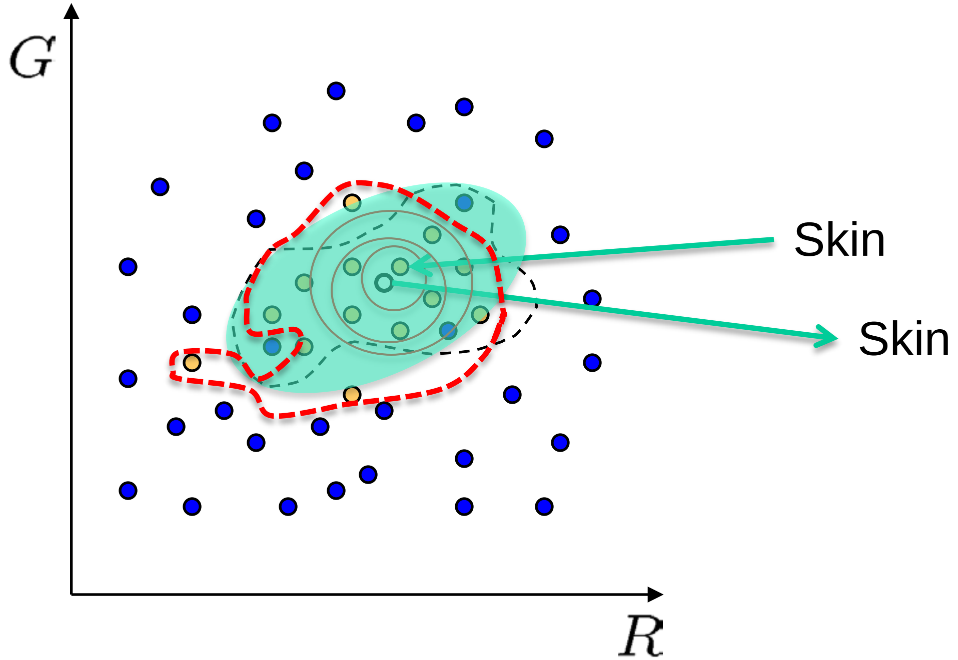
Learn the skin region from examples

- 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 classifier**
  - 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)

Generative vs. Discriminative

- Generative Model: We learn the parameters of a distribution that fit the object class we are trying to learn. Most common is the Gaussian distribution.

- Discriminative Model: We learn a classifier that can predict whether a sample is in our class or not.

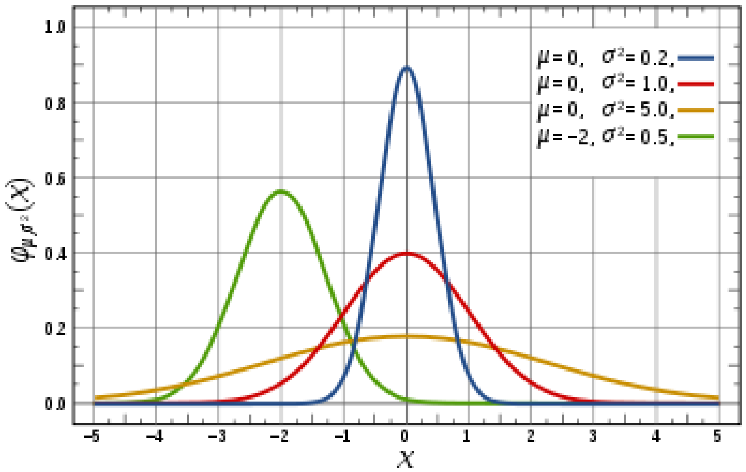
We like Gaussians because

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

These refer to the true mean and variance of the underlying distribution.

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 mean and variance of a Gaussian

The maximum likelihood estimate (MLE) for the mean of a Gaussian distribution is its sample mean.

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

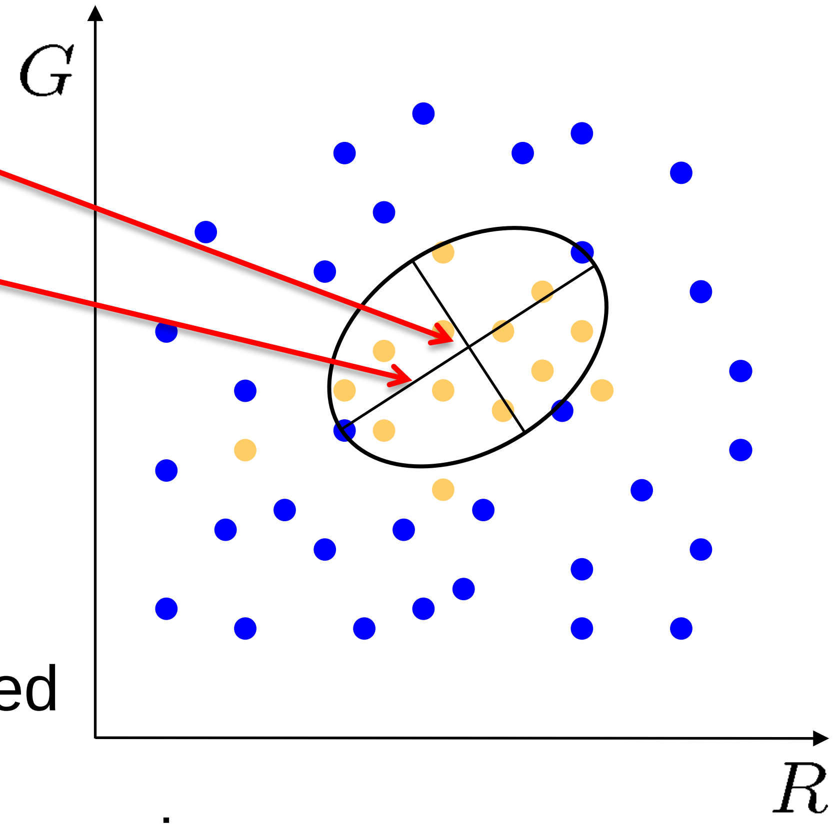
The maximum likelihood estimate for the variance of a Gaussian is its sample variance.

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

Fitting a Gaussian to Skin samples

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

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



What is being over simplified here?

This is a 2D distribution, so we need

- a mean of vectors
- a covariance matrix Σ instead of a variance

Skin detection results

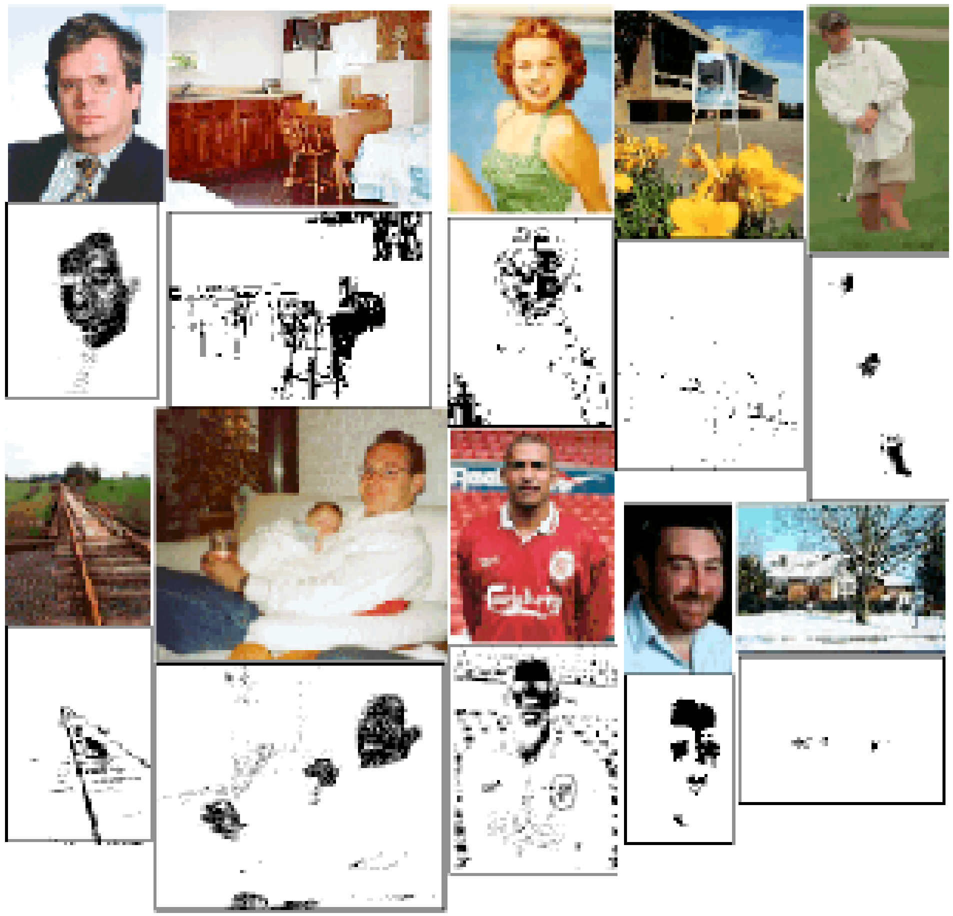


Figure 25.3. The figure shows a variety of images together with the output of the skin detector of Jones and Rehg applied to the image. Pixels marked black are skin pixels, and white are background. Notice that this process is relatively effective, and could certainly be used to focus attention on, say, 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*

Generative vs. Discriminative

- Generative Model: We learn the parameters of a distribution that fit the object class we are trying to learn. Most common is the Gaussian distribution. We look at the **probability of a sample belonging to that distribution**.

- **Discriminative Model**: We learn a classifier that can predict whether a sample is in our class or not.

- What is a classifier?

- Mathematically, it is a function f that when given a sample can predict its class.

Classification

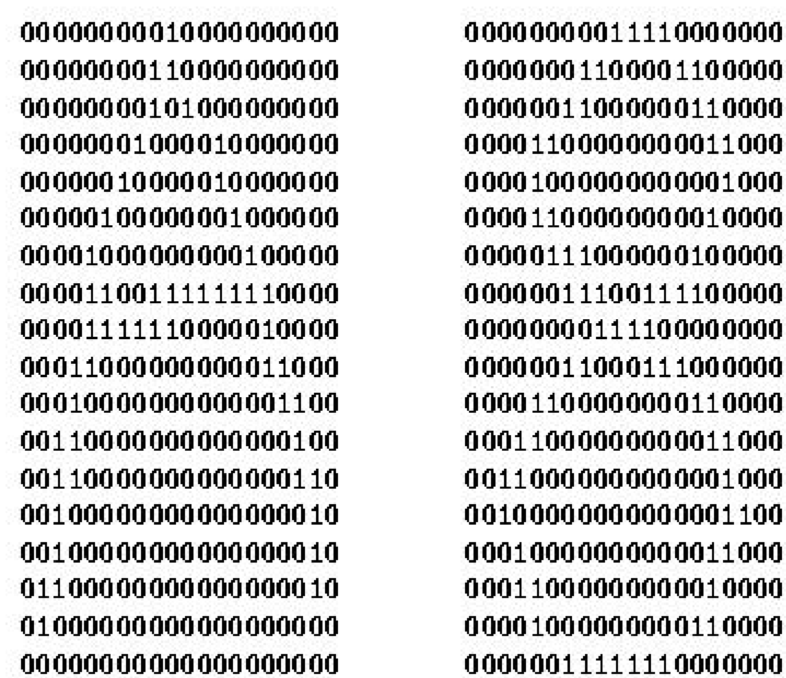
- A class is a set of objects having some important properties in common.

- A feature extractor is a program that inputs the data (image) and extracts features that can be used in classification.

- A classifier is a program that inputs the feature vector and assigns it to one of a set of designated classes or to the “reject” class.

Feature Vector Representation

- $X=[x_1, x_2, \ldots , x_n]$, each
  x_j a real number
- x_j may be an object
  measurement
- x_j may be a count of
  object parts



Example: [area, height, width, #holes, #strokes, cx, cy]

Some Terminology

Classes: set of m known categories of objects

(a) might have a known description for each

(b) might have a set of samples for each

Reject Class:

a generic class for objects not in any of the designated known classes

Classifier:

Assigns object to a class based on features

Supervised Learning: find f

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

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

What is each X_i?

What is y_i?

Naive Bayes Classifier

- Uses Bayes rule for classification
- One of the simpler classifiers
- Part of the free WEKA suite of classifiers

Bayes Rule

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

Which is shorthand for:

$$P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i)P(Y = y_i)}{P(X = x_j)}$$

This slide and those following are from Tom Mitchell's course in Machine Learning.

Bayes Theorem

Does patient have cancer or not?

A patient takes a lab test and the result comes back positive. The test returns a correct positive result in only 98% of the cases in which the disease is actually present, and a correct negative result in only 97% of the cases in which the disease is not present. Furthermore, .008 of the entire population have this cancer.

$$P(cancer) = .008 \qquad P(\neg cancer) = .992$$
$$P(+|cancer) = .980 \qquad P(-|cancer) = .020$$
$$P(+|\neg cancer) = .030 \qquad P(-|\neg cancer) = .970$$

Basic Formulas for Probabilities

- *Product Rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B:

$$P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$$

- *Sum Rule*: probability of a disjunction of two events A and B:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Theorem of total probability*: if events $A_1, \ldots, A_n$ are mutually exclusive with $\sum_{i=1}^{n} P(A_i) = 1$, then

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Naive Bayes Classifier

V is set of possible classes

Assume target function $f : X \rightarrow V$, where each instance x described by attributes $\langle a_1, a_2 \ldots a_n \rangle$.
Most probable value of $f(x)$ is:

MAP: maximum a posteriori probability.

most probable class

$$v_{MAP} = \operatorname*{argmax}_{v_j \in V} P(v_j | a_1, a_2 \ldots a_n)$$

$$= \operatorname*{argmax}_{v_j \in V} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

by Bayes Rule

$$= \operatorname*{argmax}_{v_j \in V} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Assume $P(a_1,...,a_n)$ same for all $a_1,...a_n$.

Naive Bayes assumption:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

Conditional independence

which gives

Naive Bayes classifier: $v_{NB} = \operatorname*{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$

Naive Bayes Algorithm

Naive_Bayes_Learn($examples$)

For each target value v_j for each class

$\hat{P}(v_j) \leftarrow$ estimate $P(v_j)$ estimate its probability

For each attribute value a_i of each attribute a

$\hat{P}(a_i|v_j) \leftarrow$ estimate $P(a_i|v_j)$ and estimate the probability of that class for each attribute value

Classify_New_Instance(x)

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j)$$

Elaboration

The set of examples is actually a set of preclassified feature vectors called the **training set**.

From the training set, we can estimate the a priori probability of each class:

P(C) = # training vectors from class C / total # of training vectors

For each class **C**, attribute **a,** and possible value for that attribute **a_i**, we can estimate the conditional probability:

$P(a_i | C_j)$ = # training vectors from class C_j in which value(a) = a_i

Training Examples

features | class | some estimates

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

P(y) = 9/14

P(n) = 5/14

P(sun | y) = 2/9

P(cool | y) = 3/9

P(high | y) = 3/9

P(strong | y) = 3/9

(probability of class Yes times product of probabilities of certain values for each of its attributes.)

P(y)P(sun | y)P(cool | y)P(high | y)P(strong | y) =
(9/14) * (2/9) * (3/9) * (3/9) * (3/9) =
.005

Naive Bayes: Example

Consider *PlayTennis* again, and new instance

$\langle Outlk = sun, Temp = cool, Humid = high, Wind = strong$

Want to compute:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

v_1 is y
v_2 is n

$P(y)\ P(sun|y)\ P(cool|y)\ P(high|y)\ P(strong|y) = .005$

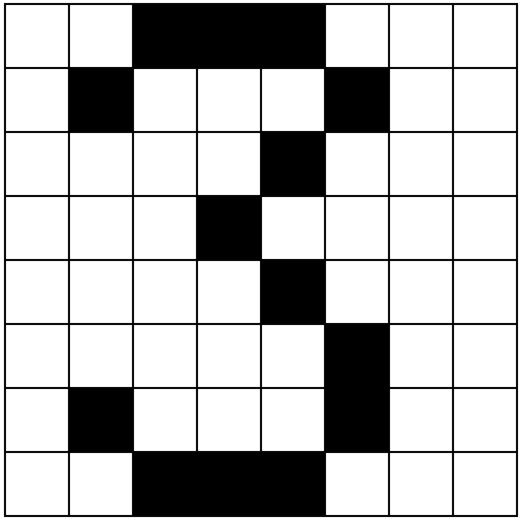
$P(n)\ P(sun|n)\ P(cool|n)\ P(high|n)\ P(strong|n) = .021$

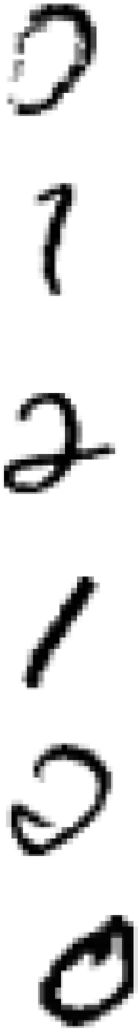
$$\rightarrow v_{NB} = n$$

This is a prediction. If it is sunny, cool, highly humid, and strong wind, it is more likely that we won't play tennis than that we will.

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:

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

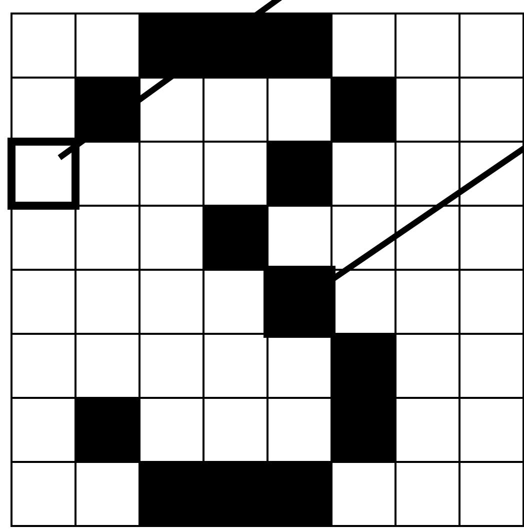
Are the features independent given class?

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:

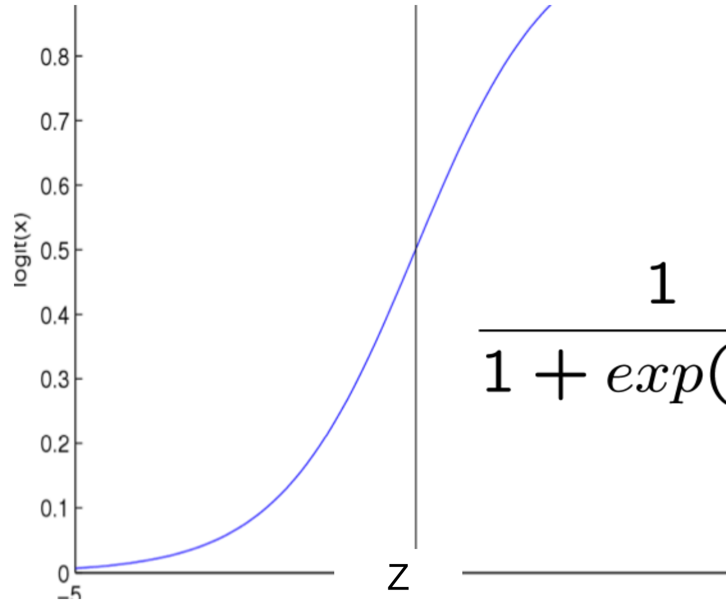
$$P(X_1 ... X_n | Y) \neq \prod_i P(X_i | Y)$$

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

But it's not the only trick in our bag.

Logistic Regression

Logistic function (Sigmoid):



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

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

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

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

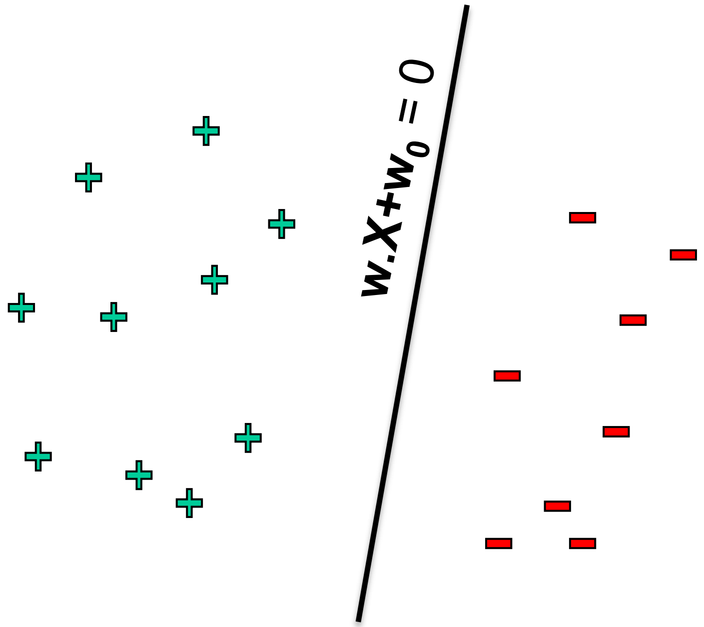
Logistic Regression: decision boundary

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

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

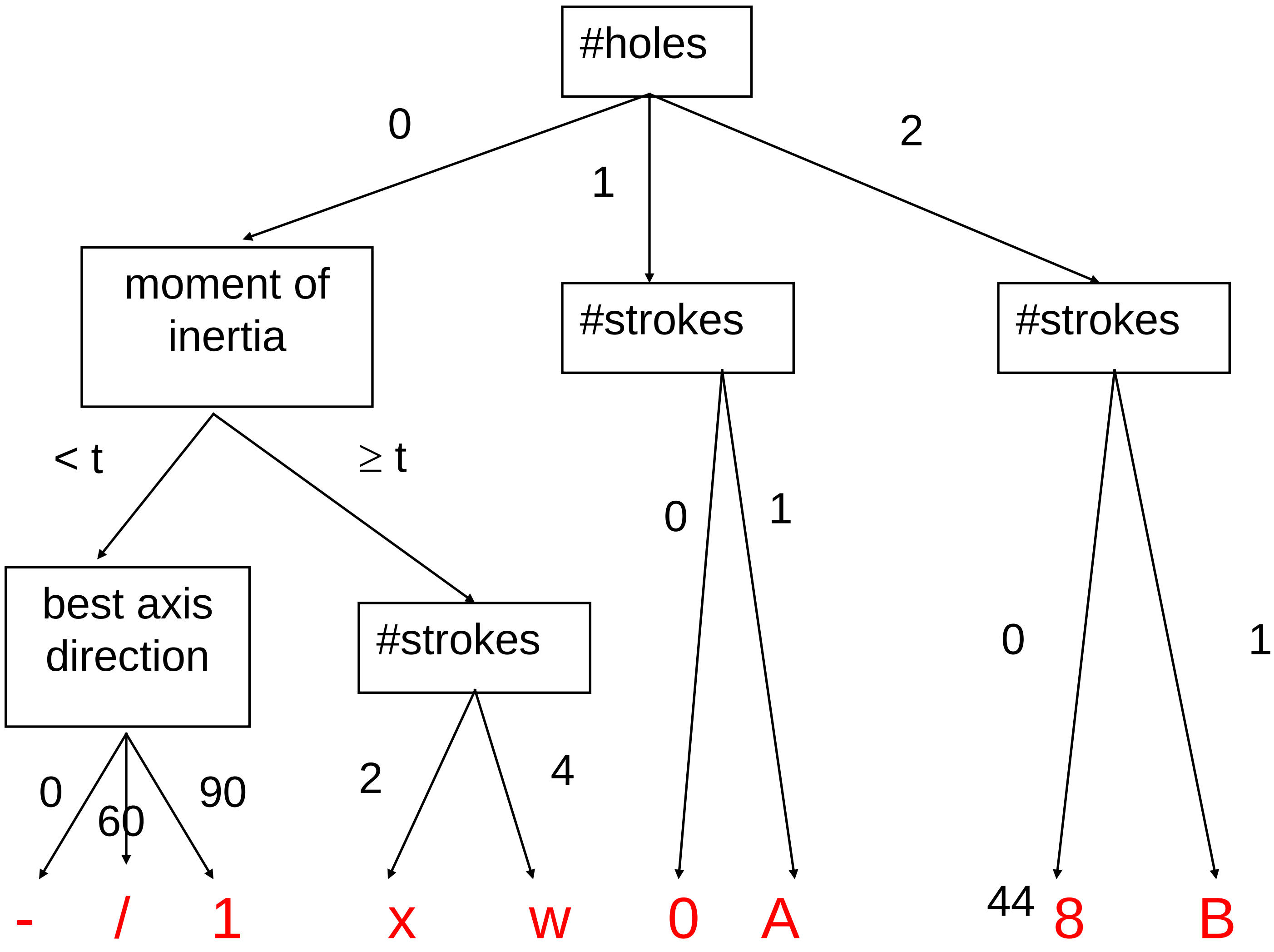
$$1 < \frac{P(Y=0|X)}{P(Y=1|X)}$$

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



A Linear Classifier!

Decision Trees



Decision Tree Characteristics

1. Training

   How do you construct one from training data?
   Entropy-based Methods

2. Strengths

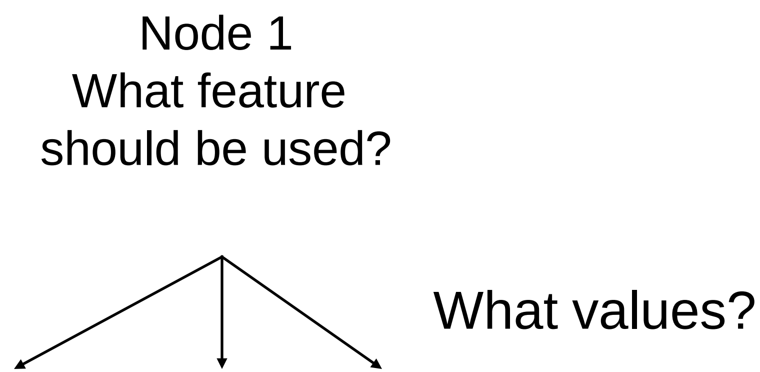
   Easy to Understand

3. Weaknesses

   Overfitting (the classifier fits the training data very well, but not new unseen data)

Entropy-Based Automatic Decision Tree Construction

Training Set S
x1=(f11,f12,…f1m)
x2=(f21,f22, f2m)
.
.
xn=(fn1,f22, f2m)



Quinlan suggested information gain in his ID3 system and later the gain ratio, both based on entropy.

Entropy

Given a set of training vectors S, if there are c classes,

$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \log_2 (p_i)$$

Where p_i is the proportion of category i examples in S.

If all examples belong to the same category, the entropy is 0 (no discrimination).

The greater the discrimination power, the larger the entropy will be.

Information Gain

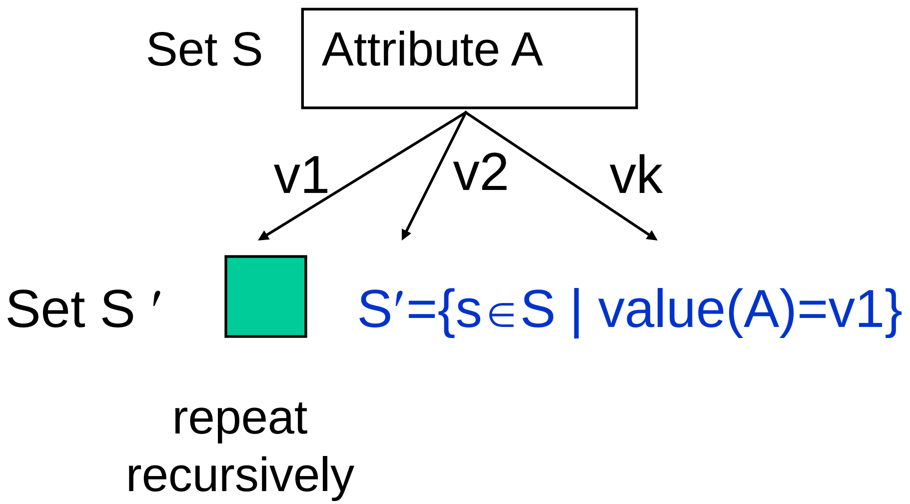
The information gain of an attribute A is the expected reduction in entropy caused by partitioning on this attribute.

$$\text{Gain}(S,A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|Sv|}{|S|} \text{Entropy}(S_v)$$

where S_v is the subset of S for which attribute A has value v.

Choose the attribute A that gives the maximum information gain.

Information Gain (cont)



The attribute A selected at the top of the tree is the one with the highest information gain.

Subtrees are constructed for each possible value vi of attribute A.

The rest of the tree is constructed in the same way.

Summary: Decision Trees

Limitations

- Often produce noisy (bushy) or weak (stunted) classifiers.
- Do not generalize too well.
- Training data fragmentation:
  - As tree progresses, splits are selected based on less and less data.
- Overtraining and undertraining:
  - Deep trees: fit the training data well, will not generalize well to new test data.
  - Shallow trees: not sufficiently refined.
- Stability
  - Trees can be very sensitive to details of the training points.
  - If a single data point is only slightly shifted, a radically different tree may come out!

  $\Rightarrow$ Result of discrete and greedy learning procedure.
- Expensive learning step
  - Mostly due to costly selection of optimal split.

Randomized Decision Trees (Amit & Geman 1997)

Decision trees: main effort on finding good split

- Training runtime: $O(DN^2 \log N)$
- This is what takes most effort in practice.
- Especially cumbersome with many attributes (large D).

Idea: randomize attribute selection

- No longer look for globally optimal split.
- Instead randomly use subset of K attributes on which to base the split.
- Choose best splitting attribute e.g. by maximizing the information gain (= reducing entropy):

Randomized Decision Trees

Randomized splitting

- Faster training: $O(K N^2 \log N)$.
- Use very simple binary feature tests.
- Typical choice
  - $K = 10$ for root node.
  - $K = 100d$ for node at level d.

Effect of random split

- Of course, the tree is no longer as powerful as a single classifier…
- But we can compensate by building several trees.

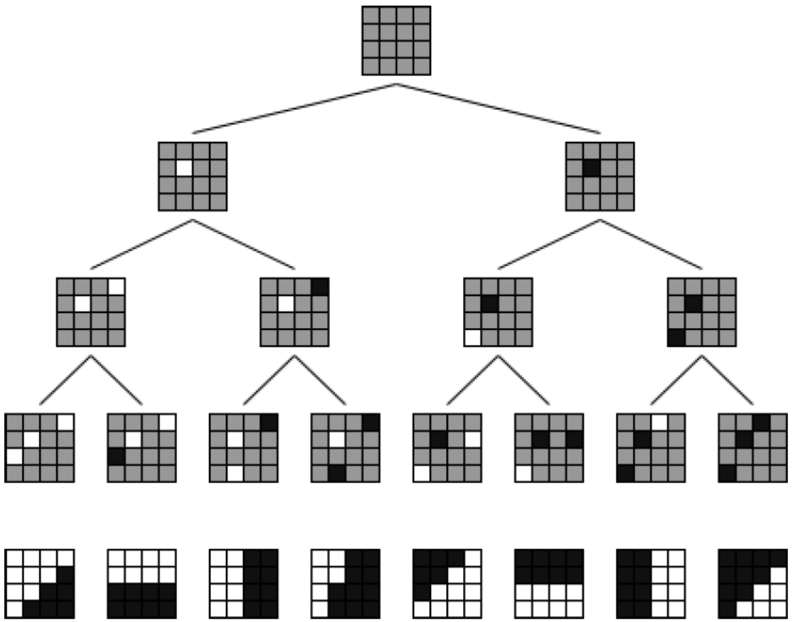
Applications

Computer Vision: Optical character recognition

- Classify small (14x20) images of hand-written characters/digits
  into one of 10 or 26 classes.

Simple binary features

- Tests for individual binary pixel values.
- Organized in randomized tree.



Y. Amit, D. Geman, Shape Quantization and Recognition with Randomized Trees, *Neural Computation*, Vol. 9(7), pp. 1545-1588, 1997.

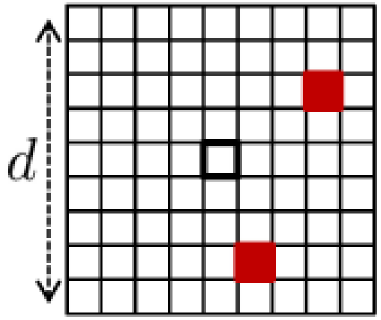
Applications

Computer Vision: fast keypoint detection

- Detect keypoints: small patches in the image used for matching
- Classify into one of ~200 categories (visual words)

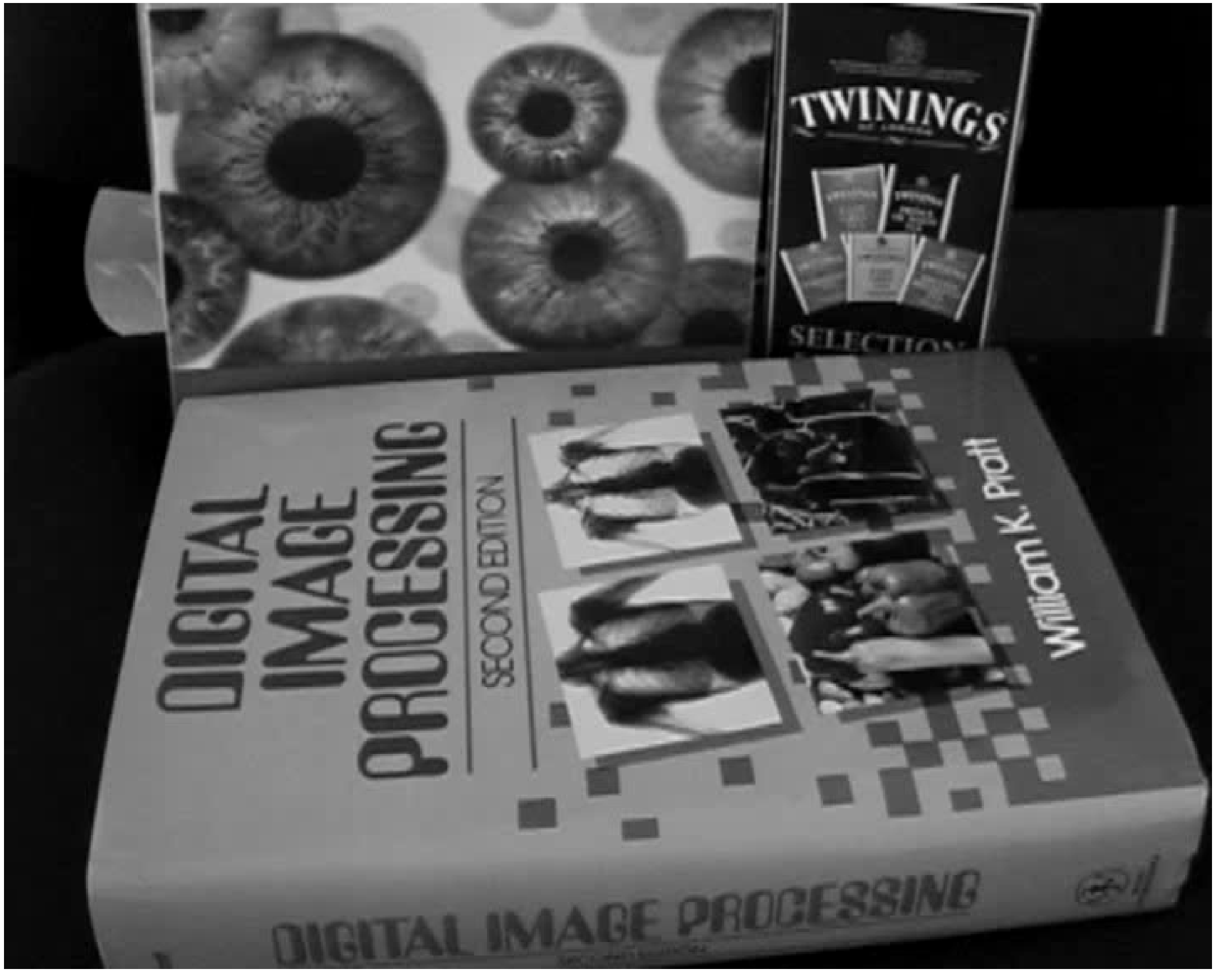
Extremely simple features

- E.g. pixel value in a color channel (CIELab)
- E.g. sum of two points in the patch
- E.g. difference of two points in the patch
- E.g. absolute difference of two points



Create forest of randomized decision trees

Application: Fast Keypoint Detection



M. Ozuysal, V. Lepetit, F. Fleuret, P. Fua, Feature Harvesting for Tracking-by-Detection. In *ECCV'06*, 2006.

Random Forests (Breiman 2001)

General ensemble method   multiple classifiers

- Idea: Create “forest” of many (very simple) trees.

Empirically very good results

Standard decision trees: main effort on finding good split

- Random Forests trees put very little effort in this.
- Each split is only made based on a random subset of the available attributes.
- Trees are grown fully (important!).

Main secret

- Injecting the “right kind of randomness”. See next slide.

Random Forests – Algorithmic Goals

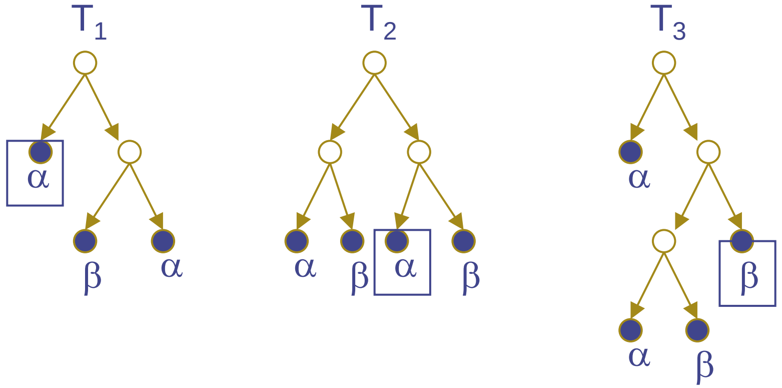
Create many trees (50 – 1,000)

Inject randomness into trees such that

- Each tree has maximal strength
  - I.e. a fairly good model on its own
- Each tree has minimum correlation with the other trees.
  - I.e. the errors tend to cancel out.

Ensemble of trees votes for final result

- Simple majority vote for category.



Other Important Classifiers

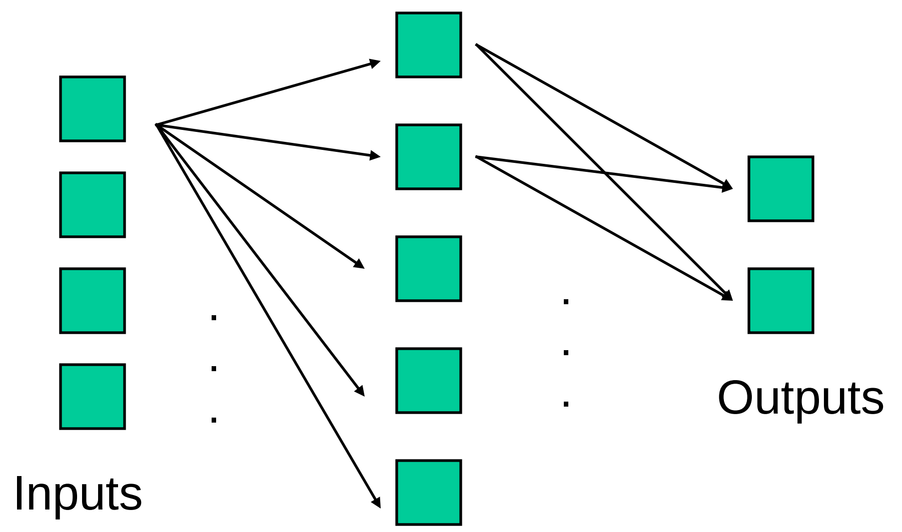
- Neural Nets: We will look at these in detail in the lecture on deep neural nets, so only a quick look now

- Support Vector Machines: These are important in certain object recognition systems, so we will look at them only briefly now and more thoroughly later

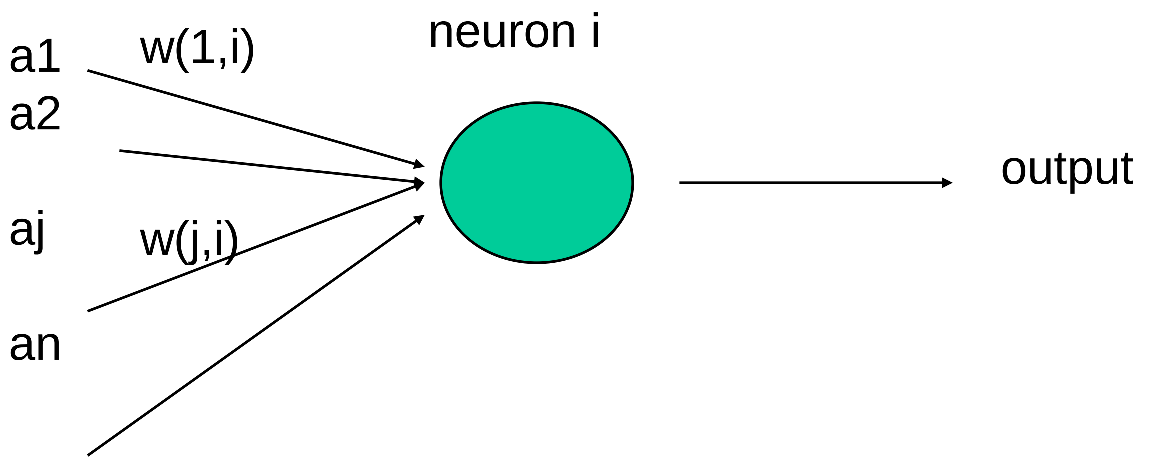
Artificial Neural Nets

Artificial Neural Nets (ANNs) are networks of artificial neuron nodes, each of which computes a simple function.

An ANN has an input layer, an output layer, and "hidden" layers of nodes.



Node Functions



output = g (∑ aj * w(j,i))

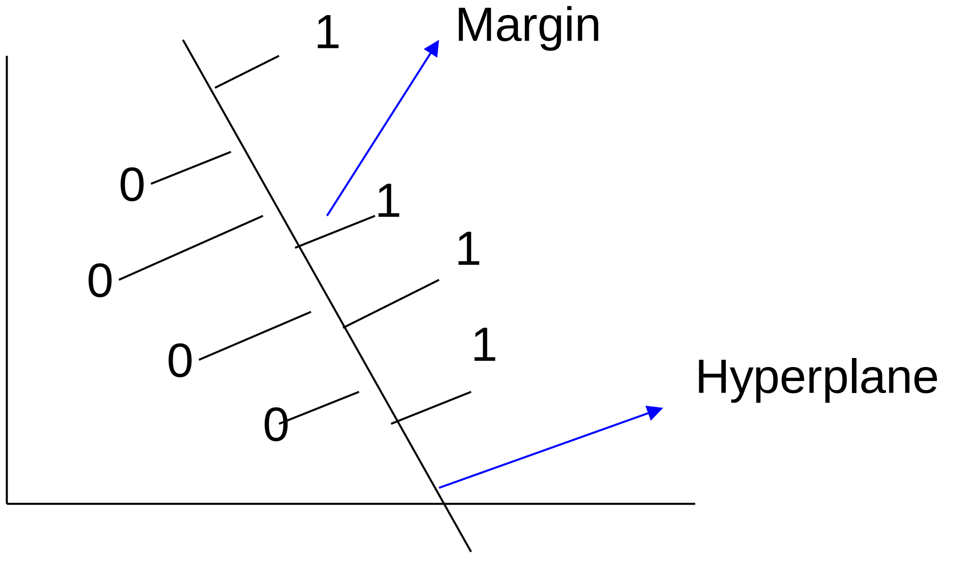
Function g is commonly a step function, sign function, or sigmoid function.

Support Vector Machines (SVM)

Support vector machines are learning algorithms that try to find a hyperplane that separates the differently classified data the most. They are based on two key ideas:

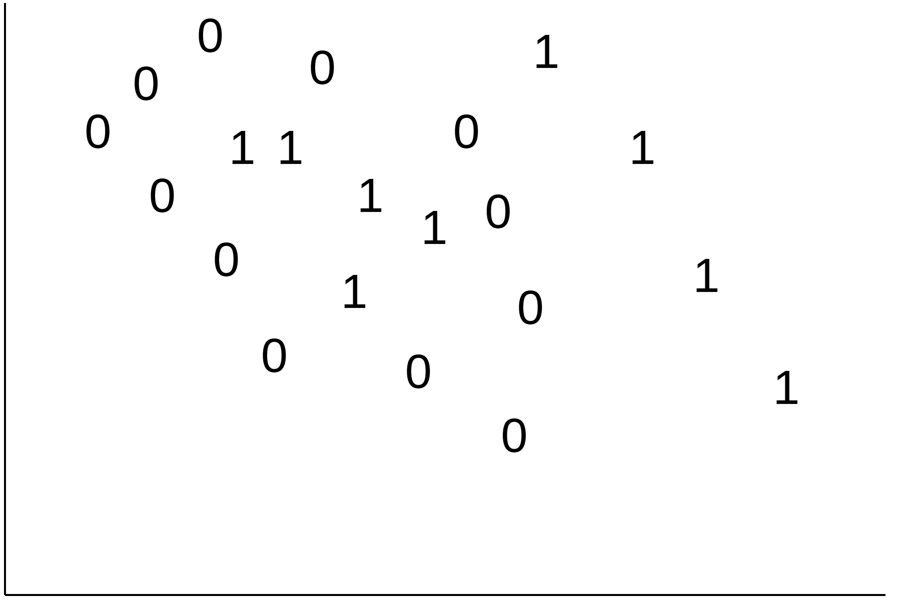
- Maximum margin hyperplanes
- A kernel 'trick'.

Maximal Margin



Find the hyperplane with maximal margin for all the points. This originates an optimization problem which has a unique solution (convex problem).

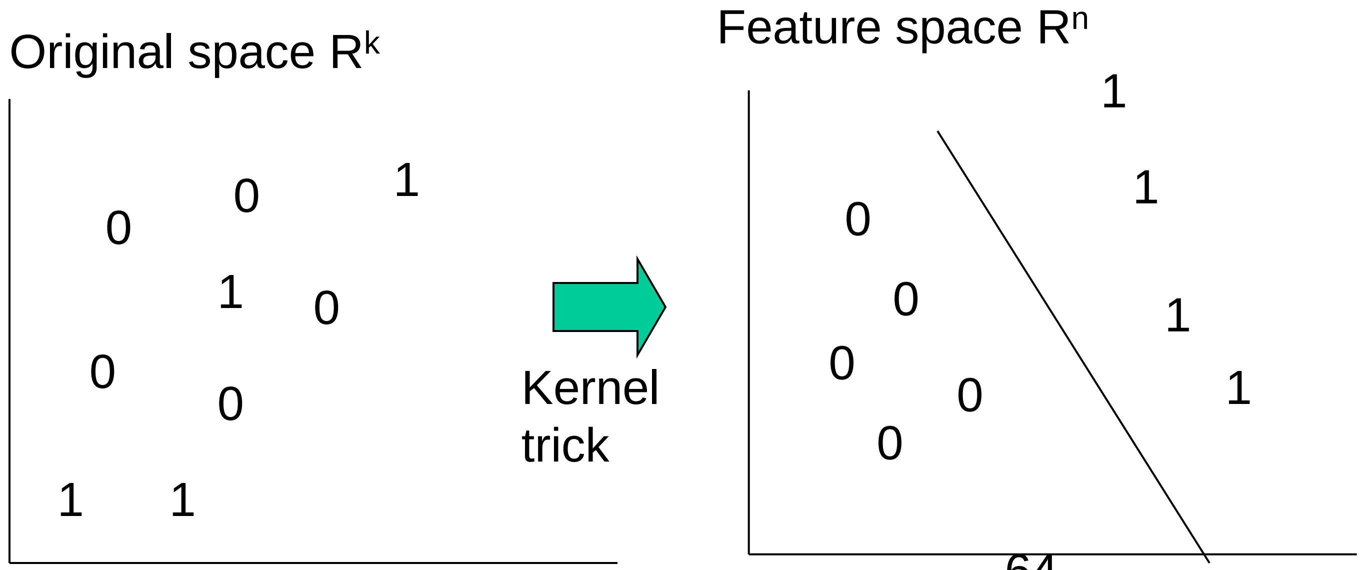
Non-separable data



What can be done if data cannot be separated with a hyperplane?

The kernel trick

The SVM algorithm maps the original data to a
a different feature space in which data (which is not separable in the original space) becomes separable in the feature space.



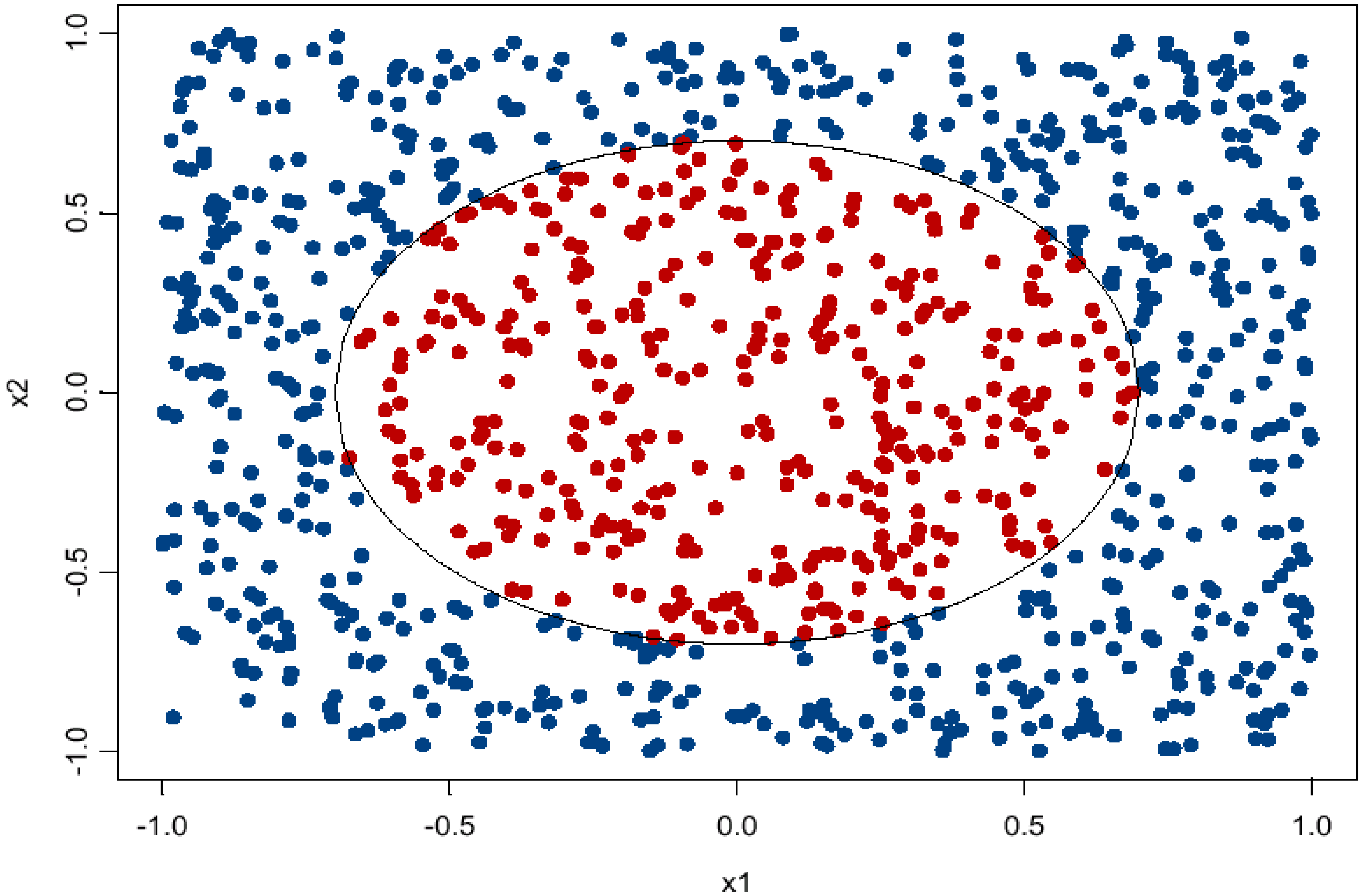
Ensembles

- When single classifiers alone are not good enough, we turn to ensembles.

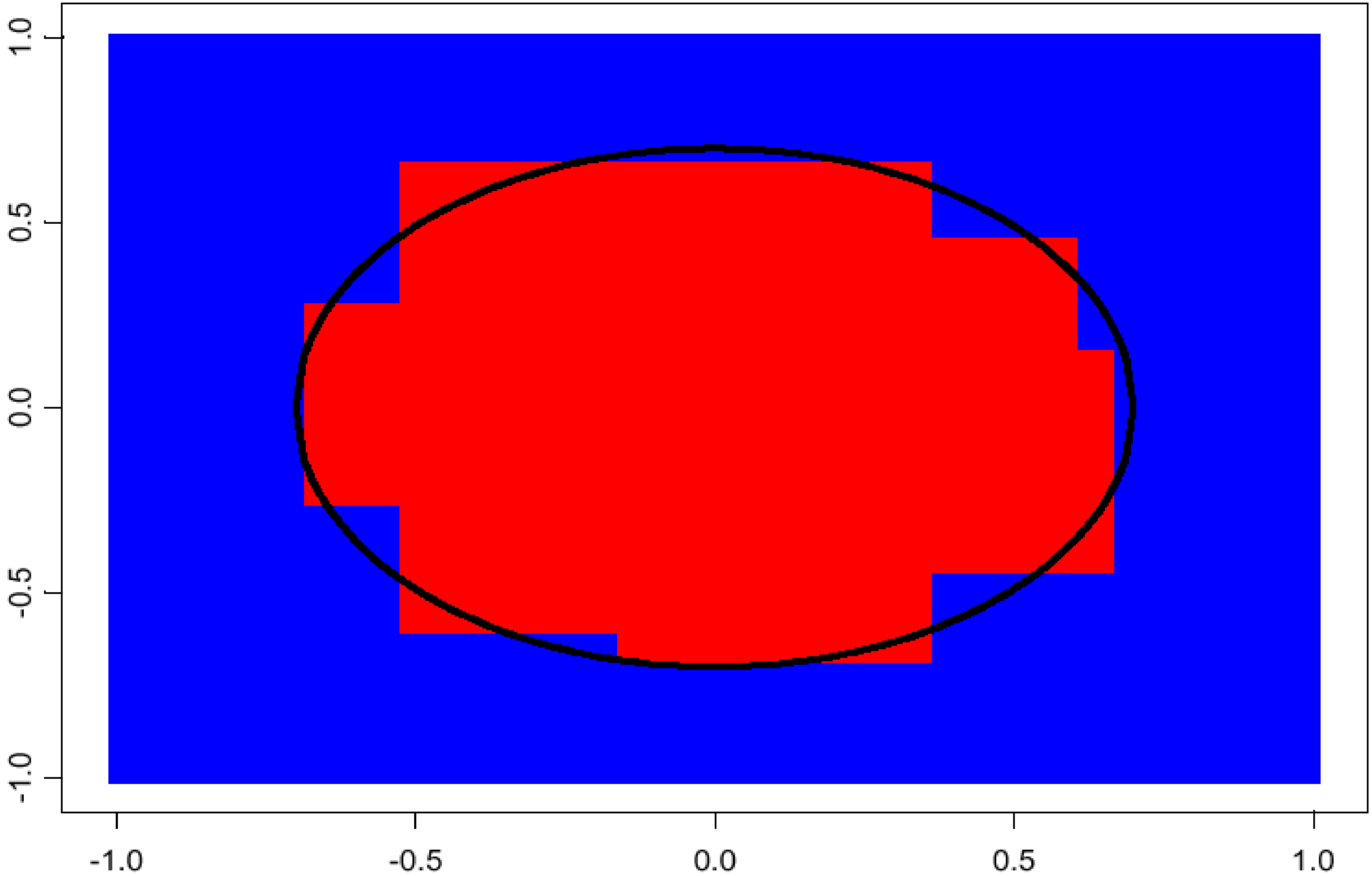
- An ensemble is a set of classifiers that together produce the final decision.

- There are multiple different ways of arranging the classifiers and of combining the results.

Nonlinear Classification Problem



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?

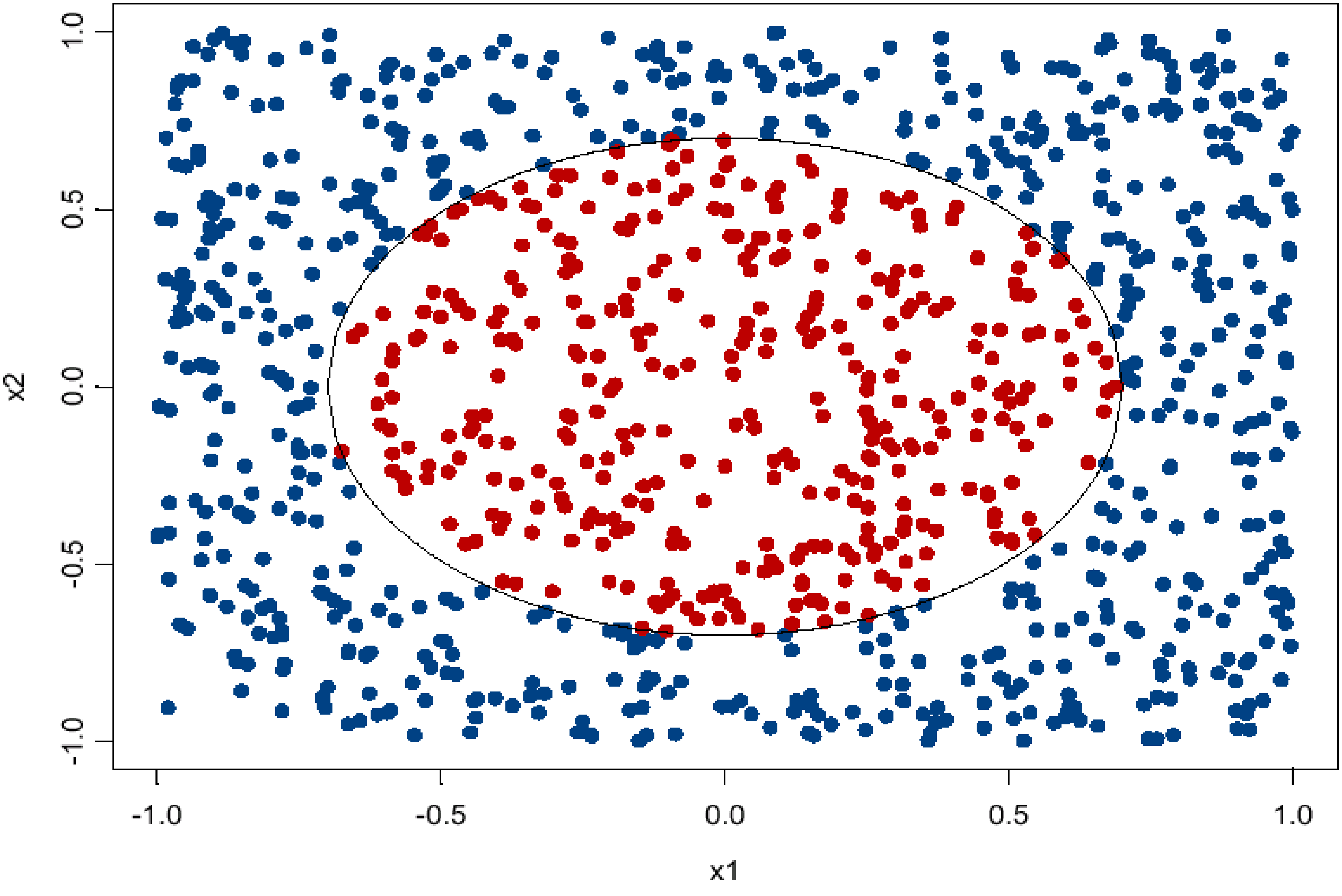
BAGGing = Bootstrap AGGregation

(Breiman, 1996)

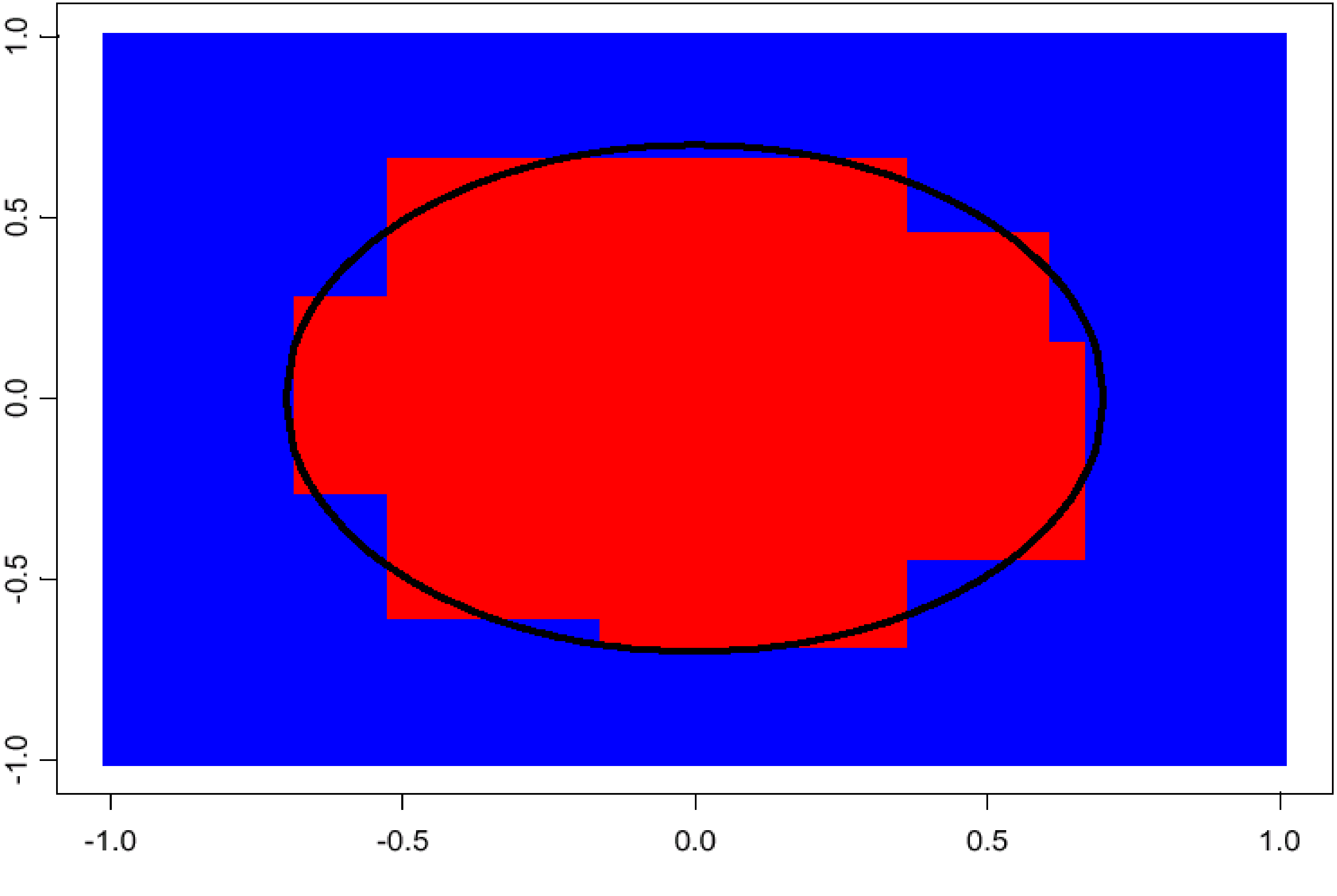
- for $i = 1, 2, \ldots, K$:
  - $T_i \leftarrow$ randomly select M training instances with replacement
  - $h_i \leftarrow \text{learn}(T_i)$ *[ID3, NB, kNN, neural net, …]*

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

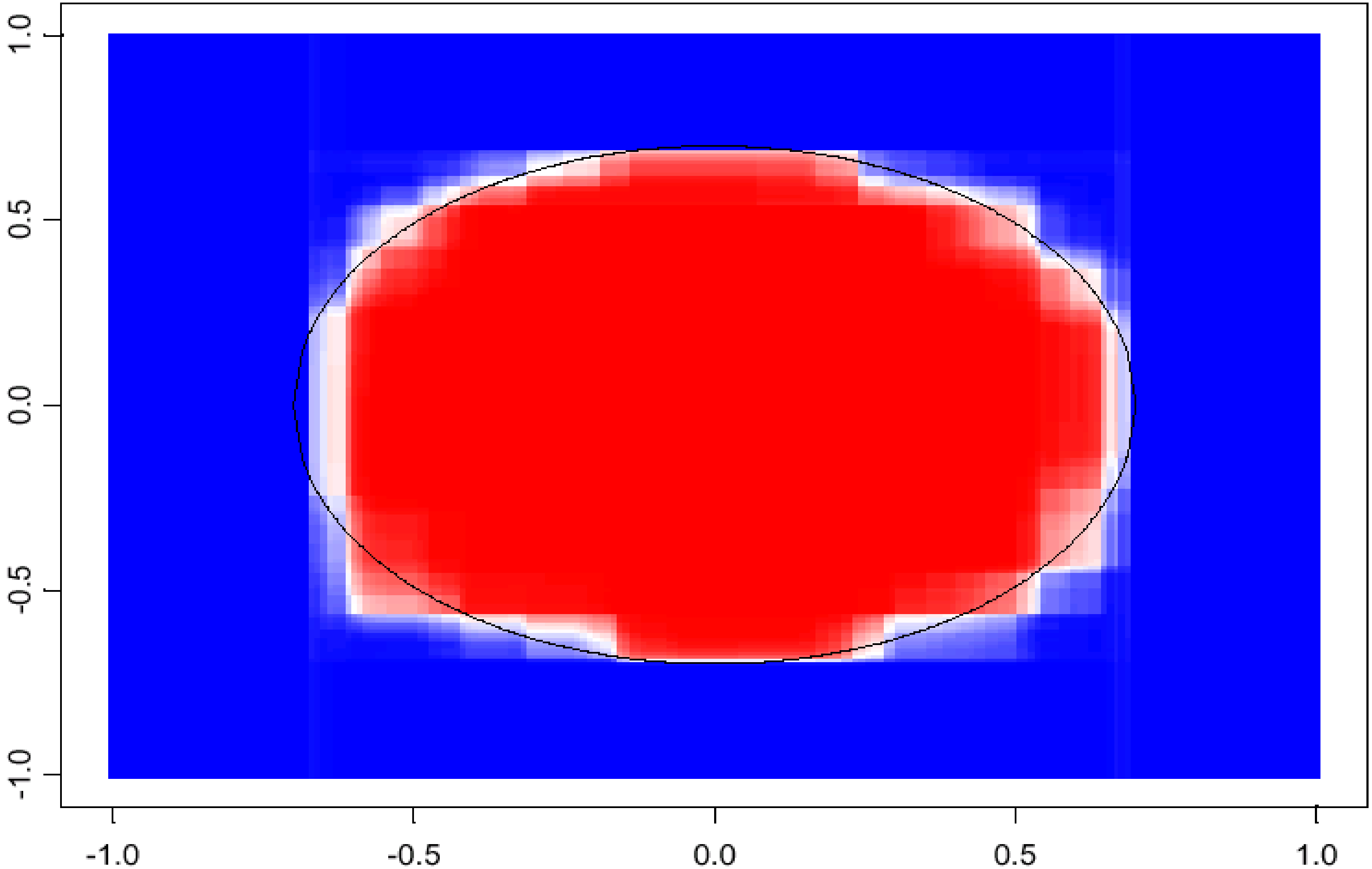
Bagging Example



Decision Boundary



100 bagged trees



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

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:

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

Practically useful

Theoretically interesting

AdaBoost

- A very popular boosting algorithm
- Can boost learning with an kind of weak learners, ie. decision stumps, decision trees, neural nets, SVMs
- Theoretically proven to boost results if the weak learners are good enough (> 50%)
- Used in the face detection algorithm for HW 4