Learning

Chapter 18 and Parts of Chapter 20

- AI systems are complex and may have many parameters.
- It is impractical and often impossible to encode all the knowledge a system needs.
- Different types of data may require very different parameters.
- Instead of trying to hard code all the knowledge, it makes sense to learn it.

Learning from Observations

- Supervised Learning – learn a function from a set of training examples which are preclassified feature vectors.

feature vector (shape,color)	class
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

Given a previously unseen feature vector, what is the rule that tells us if it is in class I or class II?

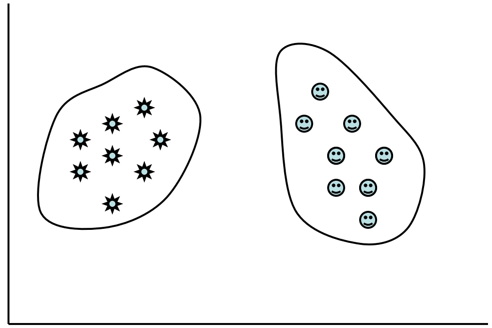
(circle, green)	?
(triangle, blue)	?

Real Observations

%Training set of Calenouria and Dorenouria
@DATA
0,1,1,0,0,0,0,0,0,1,1,2,3,0,1,2,0,0,0,0,0,0,0,0,1,0,0,1,
0,2,0,0,0,0,1,1,1,0,1,8,0,7,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
3,3,4,0,2,1,0,1,1,1,0,0,0,0,1,0,0,1,1,cal 0,1,0,0,0,1,0,0,0,4,1,2
,2,0,1,0,0,0,0,0,1,0,0,3,0,2,0,0,1,1,0,0,1,0,0,0,1,0,1,6,1,8,2,0,0,
0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,2,0,5,0,0,0,0,0,0,0,1,3,0,0,0,0,
0,cal
0,0,1,0,1,0,0,1,0,1,0,0,1,0,3,0,1,0,0,2,0,0,0,0,1,3,0,0,0,0,0,0,1,0,
2,0,2,0,1,8,0,5,0,1,0,1,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0,0,3,0,0,2,1,1,
5,0,0,0,2,1,3,2,0,1,0,0,cal 0,0,0,0,0,0,0,0,2,0,0,1,2,0,1,1,0,0,0,1
,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,3,0,0,4,1,8,0,0,0,1,0,0,0,0,0,1,0,1
,0,1,0,0,0,0,0,0,4,2,0,2,1,1,2,1,1,0,0,0,0,2,0,0,2,2,cal
...

Learning from Observations

- Unsupervised Learning – No classes are given. The idea is to find patterns in the data. This generally involves clustering.



- Reinforcement Learning – learn from feedback after a decision is made.

Topics to Cover

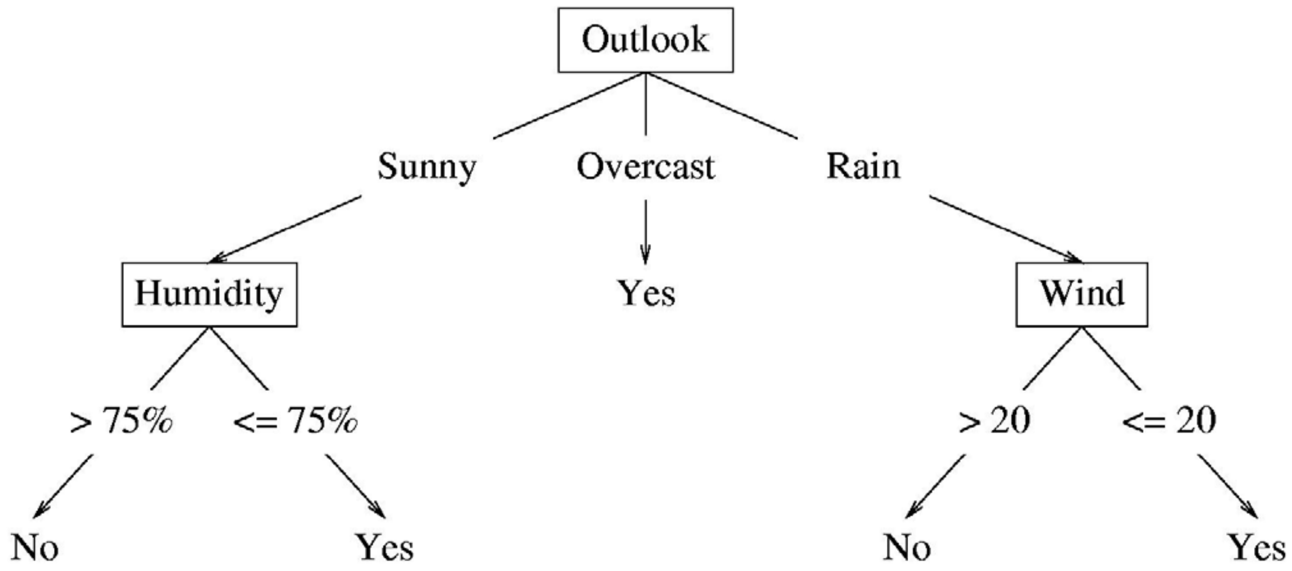
- Inductive Learning
  - decision trees
  - ensembles
  - regresion
  - neural nets
  - kernel machines
- Unsupervised Learning
  - K-Means Clustering
  - Expectation Maximization (EM) algorithm

Decision Trees

- Theory is well-understood.
- Often used in pattern recognition problems.
- Has the nice property that you can easily understand the decision rule it has learned.

Decision Tree Hypothesis Space

If the features are continuous, internal nodes may test the value of a feature against a threshold.



Classic ML example: decision tree for
“Shall I play tennis today?”

from Tom Mitchell’s ML book

A Real Decision Tree (WEKA)

Calenouria

Dorenouria

```
part23 < 0.5
|   part29 < 3.5
|   |   part34 < 0.5
|   |   |   part8 < 2.5
|   |   |   |   part2 < 0.5
|   |   |   |   |   part63 < 3.5
|   |   |   |   |   |   part20 < 1.5 : dor (53/12) [25/8]
|   |   |   |   |   |   part20 >= 1.5
|   |   |   |   |   |   |   part37 < 2.5 : cal (6/0) [5/2]
|   |   |   |   |   |   |   part37 >= 2.5 : dor (3/1) [2/0]
|   |   |   |   |   part63 >= 3.5 : dor (14/0) [3/0]
|   |   |   |   part2 >= 0.5 : cal (21/8) [10/4]
|   |   |   part8 >= 2.5 : dor (14/0) [14/0]
|   |   part34 >= 0.5 : cal (38/12) [18/4]
|   part29 >= 3.5 : dor (32/0) [10/2]
part23 >= 0.5
|   part29 < 7.5 : cal (66/8) [35/12]
|   part29 >= 7.5
|   |   part24 < 5.5 : dor (9/0) [4/0]
|   |   part24 >= 5.5 : cal (4/0) [4/0]
```

Evaluation

Correctly Classified Instances	281	73.5602 %
Incorrectly Classified Instances	101	26.4398 %
Kappa statistic	0.4718	
Mean absolute error	0.3493	
Root mean squared error	0.4545	
Relative absolute error	69.973 %	
Root relative squared error	90.7886 %	
Total Number of Instances	382	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.77	0.297	0.713	0.77	0.74	0.747	cal
	0.703	0.23	0.761	0.703	0.731	0.747	dor
Wg Avg.	0.736	0.263	0.737	0.736	0.735	0.747	

=== Confusion Matrix ===

a	b	<-- classified as
144	43	a = cal
58	137	b = dor

Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

$$\text{F-Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Properties of Decision Trees

- They divide the decision space into axis parallel rectangles and label each rectangle as one of the k classes.
- They can represent Boolean functions.
- They are variable size and deterministic.
- They can represent discrete or continuous parameters.
- They can be learned from training data.

Learning Algorithm for Decision Trees

```
Growtree(S)    /* Binary version */
   if (y==0 for all (x,y) in S) return newleaf(0)
   else if (y==1 for all (x,y) in S) return newleaf(1)
   else
      choose best attribute x_j
      S_0 = (x,y) with x_j = 0
      S_1 = (x,y) with x_j = 1
      return new node(x_j, Growtree(S_0), Growtree(S_1))
end
```

How do we choose the best attribute?

What should that attribute do for us?

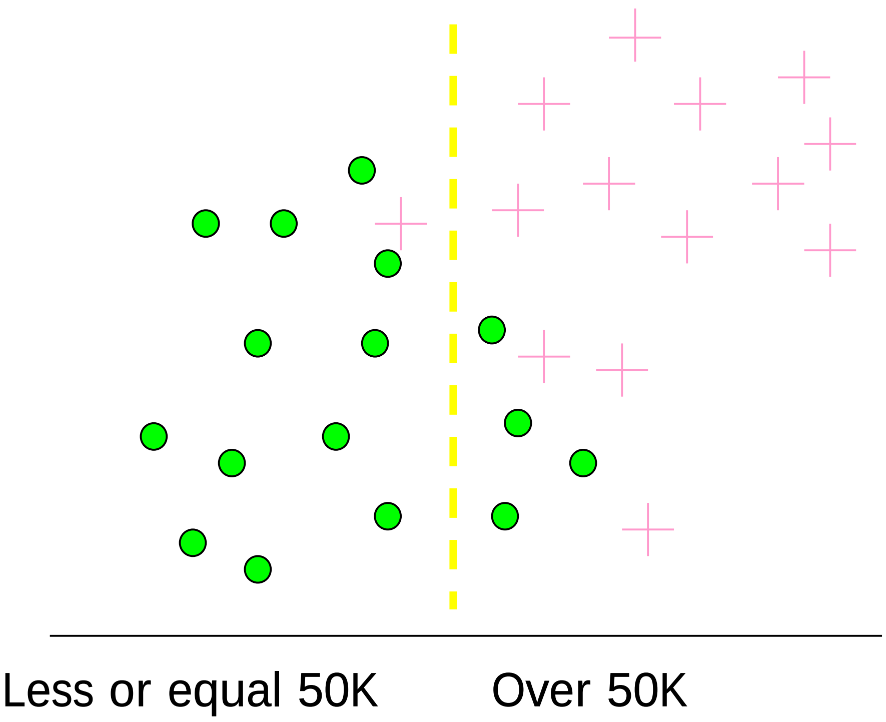
Criterion for attribute selection

- Which is the best attribute?
  - The one that will result in the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Need a good measure of purity!
  - Maximal when?
  - Minimal when?

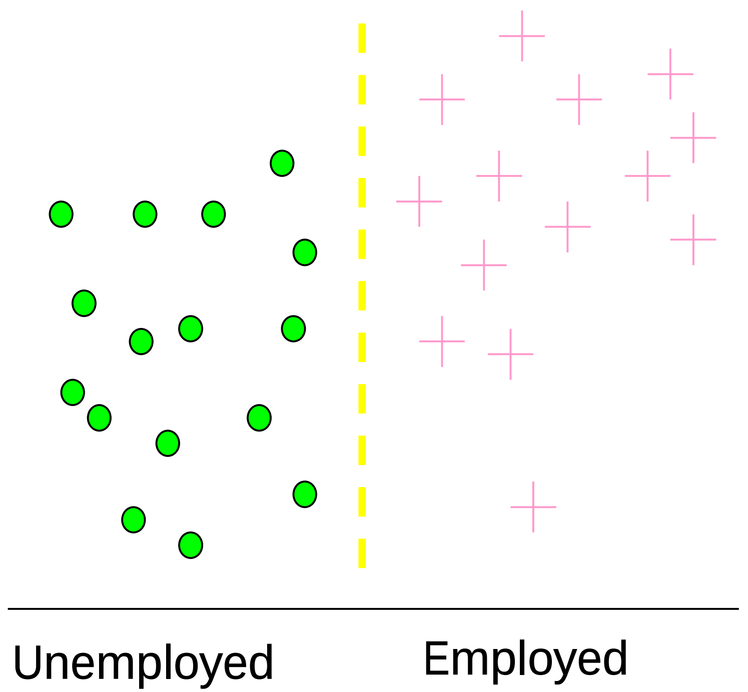
Information Gain

Which test is more informative?

Split over whether Balance exceeds 50K

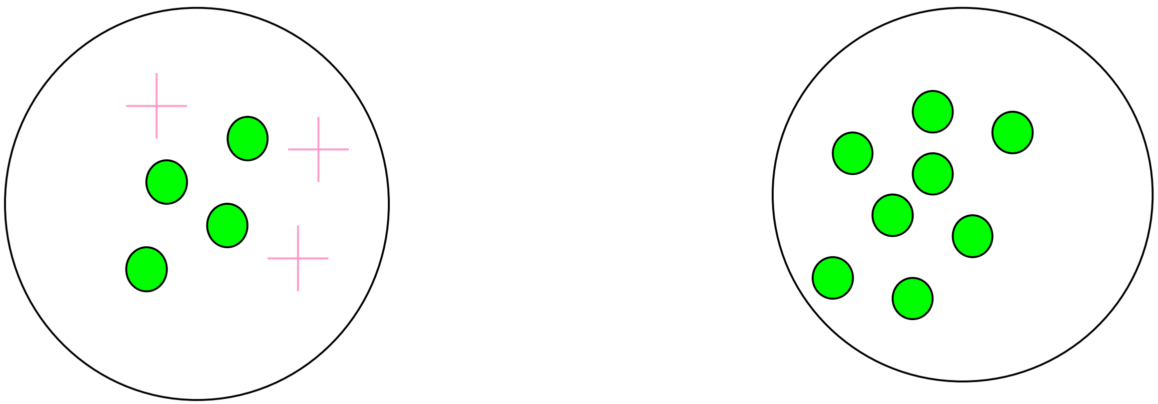


Split over whether applicant is employed



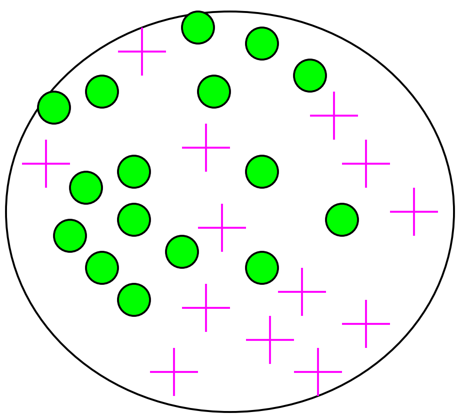
Impurity/Entropy (informal)

– Measures the level of **impurity** in a group of examples

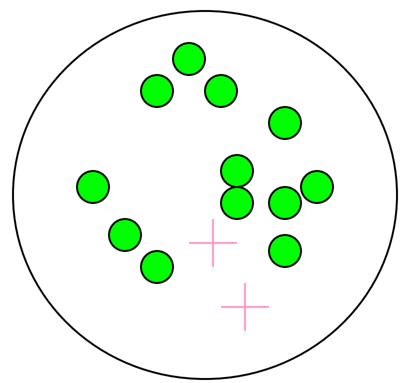


Impurity

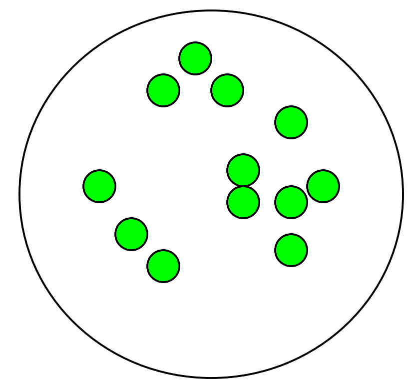
Very impure group



Less impure

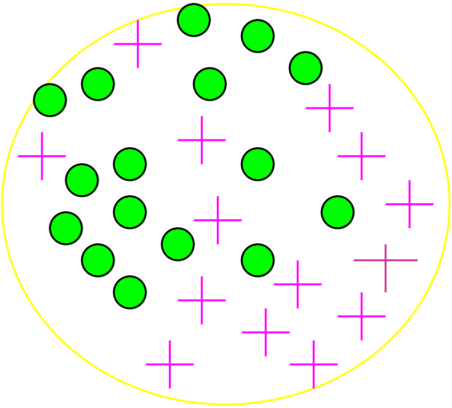


Minimum impurity



Entropy: a common way to measure impurity

- Entropy = $\sum_{i} - p_i \log_2 p_i$



p_i is the probability of class i

Compute it as the proportion of class i in the set.

16/30 are green circles; 14/30 are pink crosses
$\log_2(16/30)$ = -.9; $\log_2(14/30)$ = -1.1
Entropy = -(16/30)(-.9) –(14/30)(-1.1) = .99

- Entropy comes from information theory. The higher the entropy the more the information content.

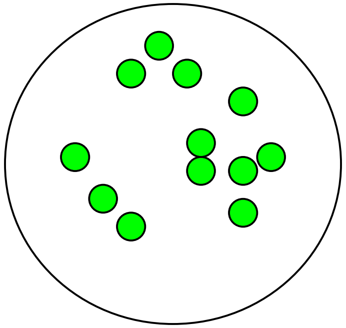
What does that mean for learning from examples?

2-Class Cases:

- What is the entropy of a group in which all examples belong to the same class?
  - entropy = $- 1 \log_2 1 = 0$

  not a good training set for learning

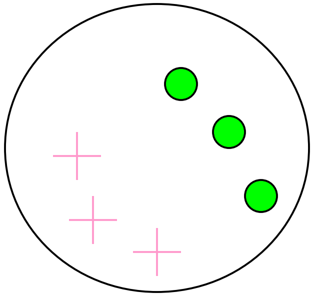
Minimum impurity



- What is the entropy of a group with 50% in either class?
  - entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

  good training set for learning

Maximum impurity



Information Gain

- We want to determine which attribute in a given set of training feature vectors is most useful for discriminating between the classes to be learned.

- Information gain tells us how important a given attribute of the feature vectors is.

- We will use it to decide the ordering of attributes in the nodes of a decision tree.

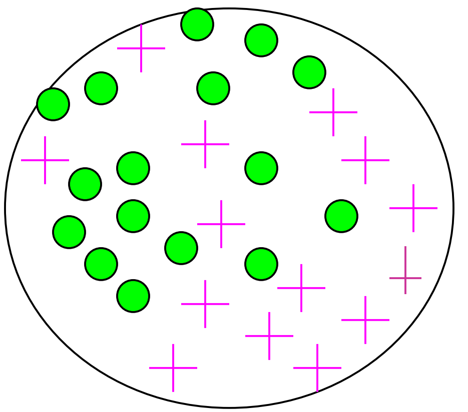
Calculating Information Gain

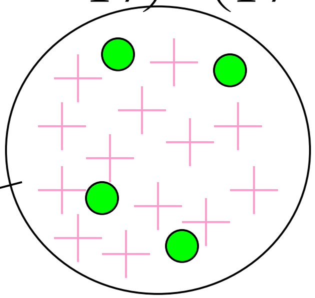
Information Gain = entropy(parent) – [average entropy(children)]

parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$

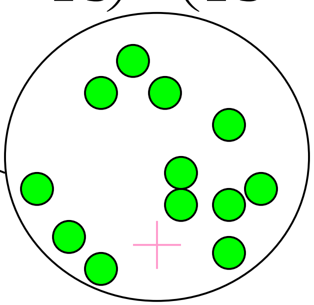
Entire population (30 instances)





17 instances

child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$



13 instances

(Weighted) Average Entropy of Children =

$$\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$$

Information Gain= 0.996 - 0.615 = 0.38 for this split

Entropy-Based Automatic Decision Tree Construction

Training Set S

$x_1=(f_{11},f_{12},\ldots f_{1m})$

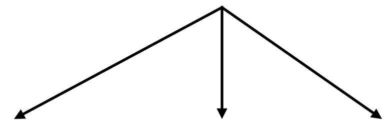
$x_2=(f_{21},f_{22}, \quad f_{2m})$

.

.

$x_n=(f_{n1},f_{22}, \quad f_{2m})$

Node 1
What feature
should be used?



What values?

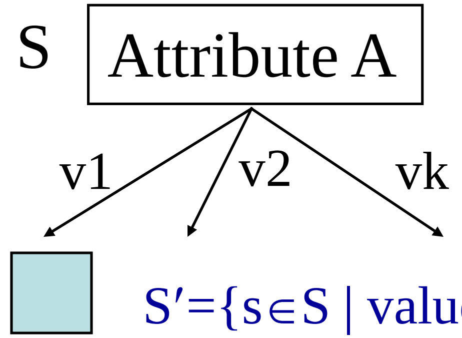
Quinlan suggested information gain in his ID3 system and later the gain ratio, both based on entropy.

Using Information Gain to Construct a Decision Tree

1

Choose the attribute A with highest information gain for the full training set at the root of the tree.

Full Training Set S



2

Construct child nodes for each value of A. Each has an associated subset of vectors in which A has a particular value.

Set S′

$S' = \{ s \in S \mid value(A) = v1 \}$

3

repeat recursively till when?

Simple Example

Training Set: 3 features and 2 classes

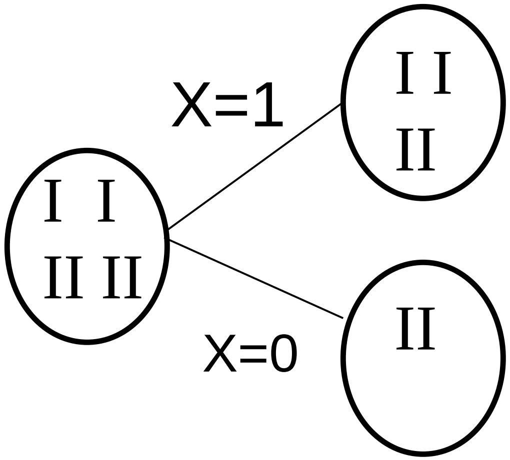
X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

How would you distinguish class I from class II?

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Eparent= 1

Split on attribute X



If X is the best attribute,
this node would be further split.

$$E_{child1} = -(1/3)\log_2(1/3)-(2/3)\log_2(2/3)$$
$$= .5284 + .39$$
$$= .9184$$

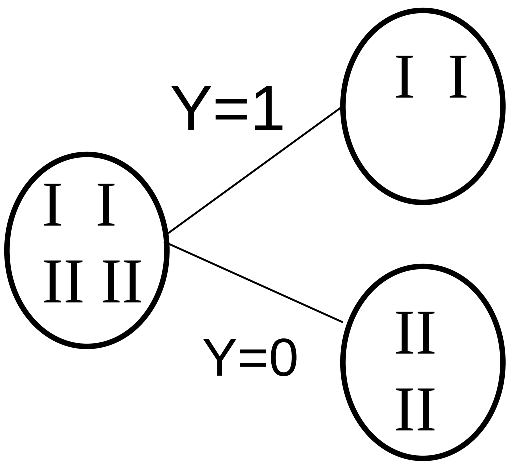
$$E_{child2} = 0$$

GAIN = 1 – (3/4)(.9184) – (1/4)(0) = .3112

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Eparent= 1

Split on attribute Y



$E_{child1}= 0$

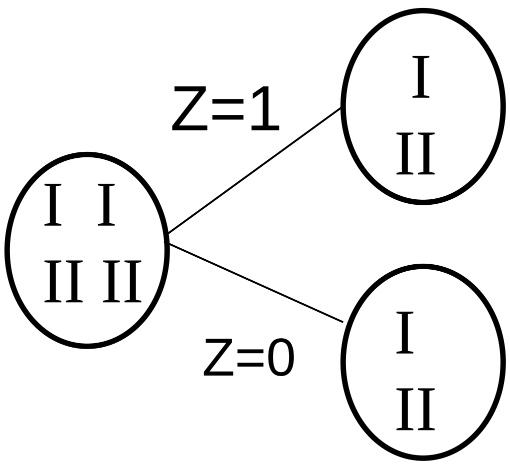
$E_{child2}= 0$

GAIN = 1 –(1/2) 0 – (1/2)0 = 1; BEST ONE

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Eparent= 1

Split on attribute Z



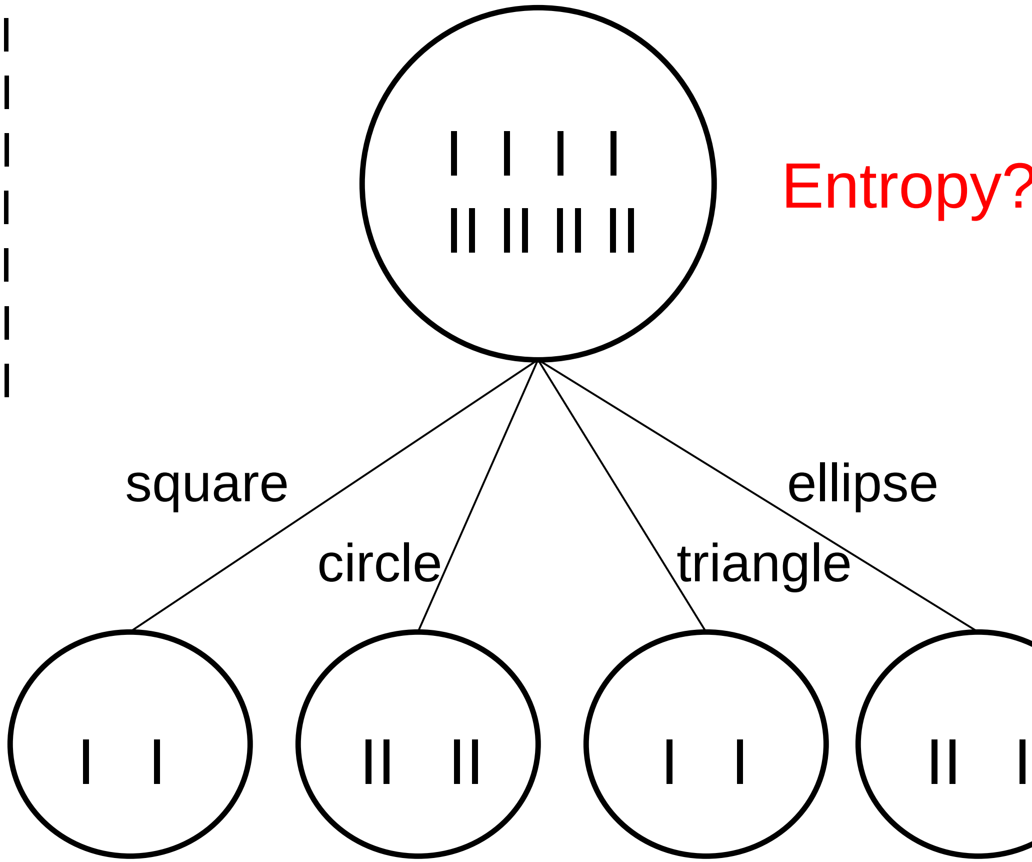
$E_{child1} = 1$

$E_{child2} = 1$

GAIN = 1 – (1/2)(1) – (1/2)(1) = 0 ie. NO GAIN; WORST

feature vector	class
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

Try the shape feature

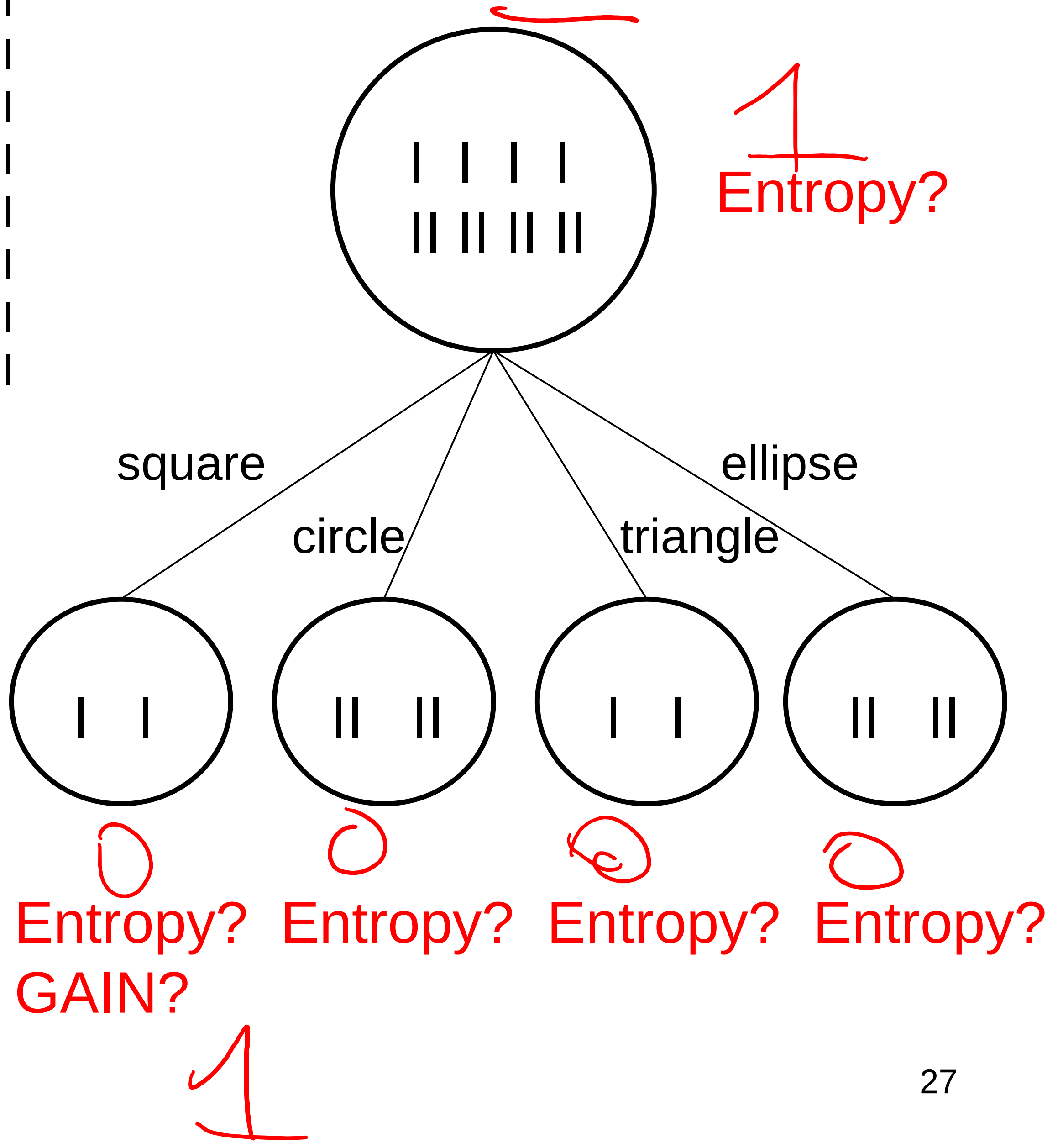


Entropy? Entropy? Entropy? Entropy?

GAIN?

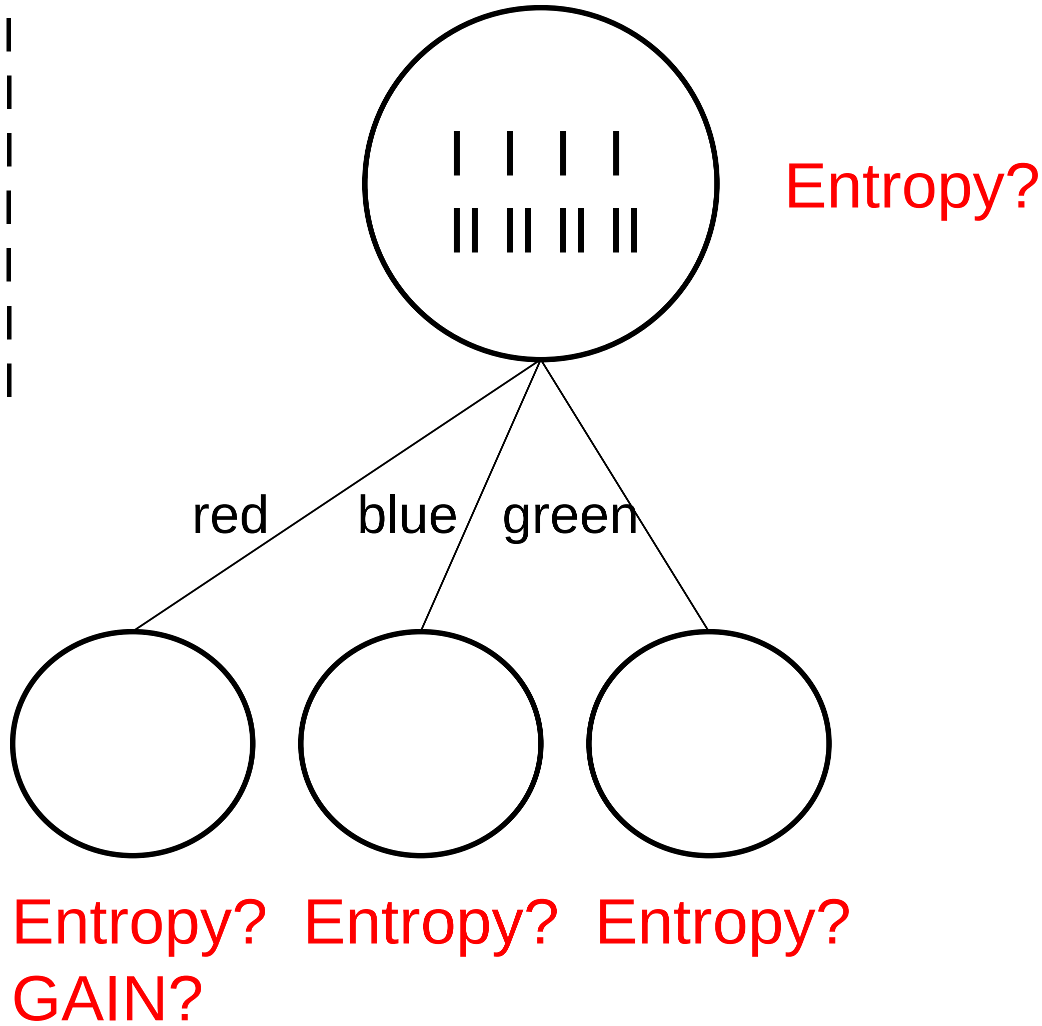
feature vector	class
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

Try the shape feature



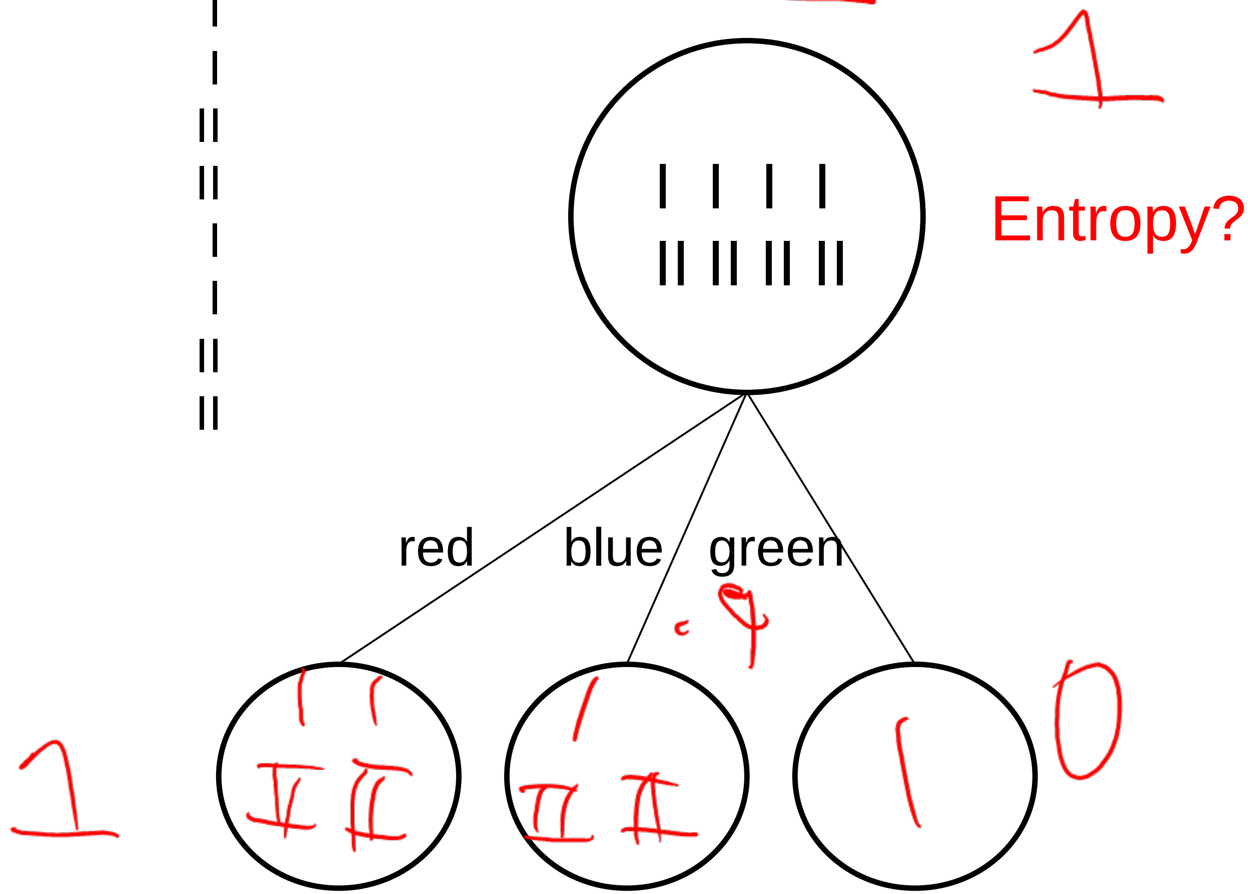
feature vector	class
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

Try the color feature



feature vector	class
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

Try the color feature



Entropy? Entropy? Entropy?

GAIN?

$$1 - \left[\frac{4}{8}(1) + \frac{3}{8}(.9) + 0\right] = .18$$

Many-Valued Features

- Your features might have a large number of discrete values.

Example: pixels in an image have (R,G,B)

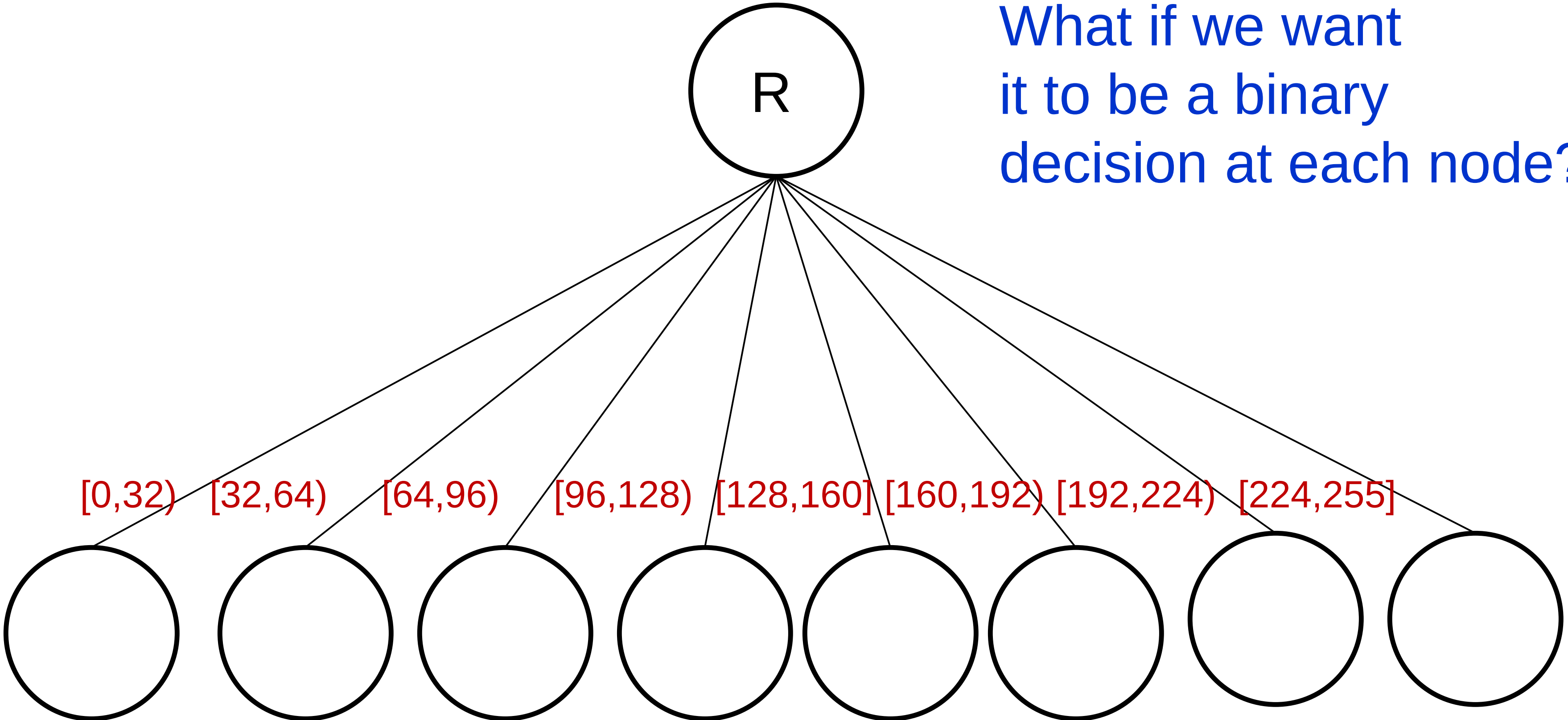
which are each integers between 0 and 255.

- Your features might have continuous values.

Example: from pixel values, we compute gradient magnitude, a continuous feature

One Solution to Both

- We often group the values into bins



What if we want it to be a binary decision at each node?

Training and Testing

- Divide data into a training set and a separate testing set.
- Construct the decision tree using the training set only.
- Test the decision tree on the training set to see how it's doing.
- Test the decision tree on the testing set to report its real performance.

Measuring Performance

- Given a test set of labeled feature vectors

e.g. (square,red) I

- Run each feature vector through the decision tree
- Suppose the decision tree says it belongs to class X and the real label is Y
- If (X=Y) that's a correct classification
- If (X<>Y) that's an error

Measuring Performance

- In a 2-class problem, where the classes are positive or negative (ie. for cancer)
  - # true positives TP
  - # true negatives TN
  - # false positives FP
  - # false negatives FN
- Accuracy = #correct / #total = (TP +TN) / (TP + TN + FP + FN)
- Precision = TP / (TP + FP)

  How many of the ones you said were cancer really were cancer?
- Recall = TP / (TP + FN)

  How many of the ones who had cancer did you call cancer?

More Measures

- F-Measure = 2*(Precision * Recall) / (Precision + Recall)

Gives us a single number to represent both precision and recall.

In medicine:

- Sensitivity = TP / (TP + FN) = Recall

The sensitivity of a test is the proportion of people who have a disease who test positive for it.

- Specificity = TN / (TN + FP)

The specificity of a test is the number of people who DON'T have a disease who test negative for it.

Measuring Performance

- For multi-class problems, we often look at the confusion matrix.

assigned class

true class

	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

C(i,j) = number of times (or percentage) class i is given label j.

Overfitting

- Suppose the classifier h has error (1-accuracy) of $error_{train}(h)$
- And there is an alternate classifier (hypothesis) h' that has $error_{train}(h')$
- What if $error_{train}(h) < error_{train}(h')$
- But $error_D(h) > error_D(h')$ for full set D
- Then we say h **overfits** the training data

What happens as the decision tree gets bigger and bigger?

Error on training data goes down, on testing data goes up

Reduced Error Pruning

- Split data into training and validation sets
- Do until further pruning is harmful
  1. Evaluate impact on validation set of pruning each possible node (and its subtree)
  2. Greedily remove the one that most improves validation set accuracy
- Then you need an additional independent testing set.

On training data it looks great.

But that’s not the case for the test data.

The tree is pruned back to the red line where it gives more accurate results on the test data.

- The WEKA example with Calenouria and Dorenouria I showed you used the REPTree classifier with 21 nodes.

- The classic decision tree for the same data had 65 nodes.

- Performance was similar for our test set.

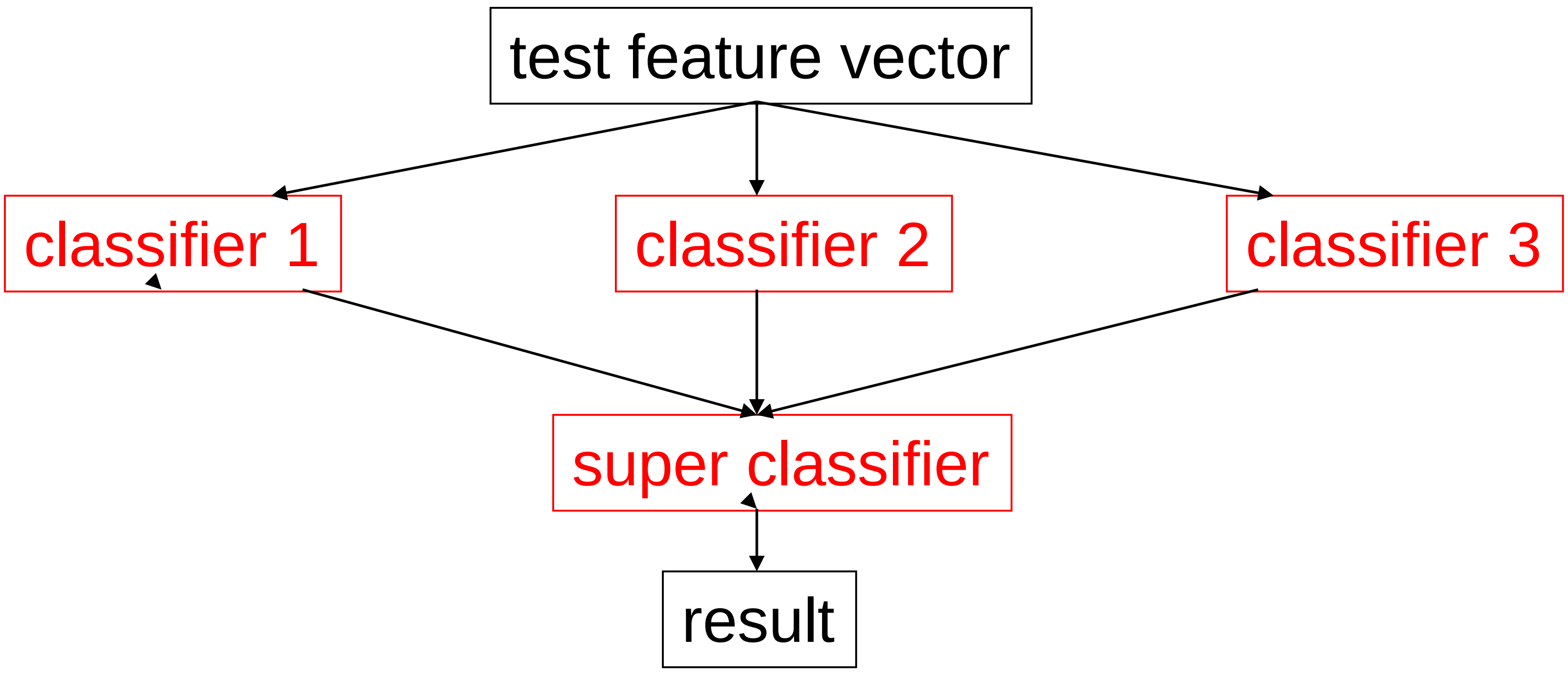
- Performance increased using a random forest of 10 trees, each constructed with 7 random features.

Decision Trees: Summary

- Representation=decision trees
- Bias=preference for small decision trees
- Search algorithm=none
- Heuristic function=information gain or information content or others
- Overfitting and pruning
- Advantage is simplicity and easy conversion to rules.

Ensembles

- An ensemble is a set of classifiers whose combined results give the final decision.



MODEL* ENSEMBLES

- Basic Idea
  - Instead of learning one model
  - Learn several and combine them

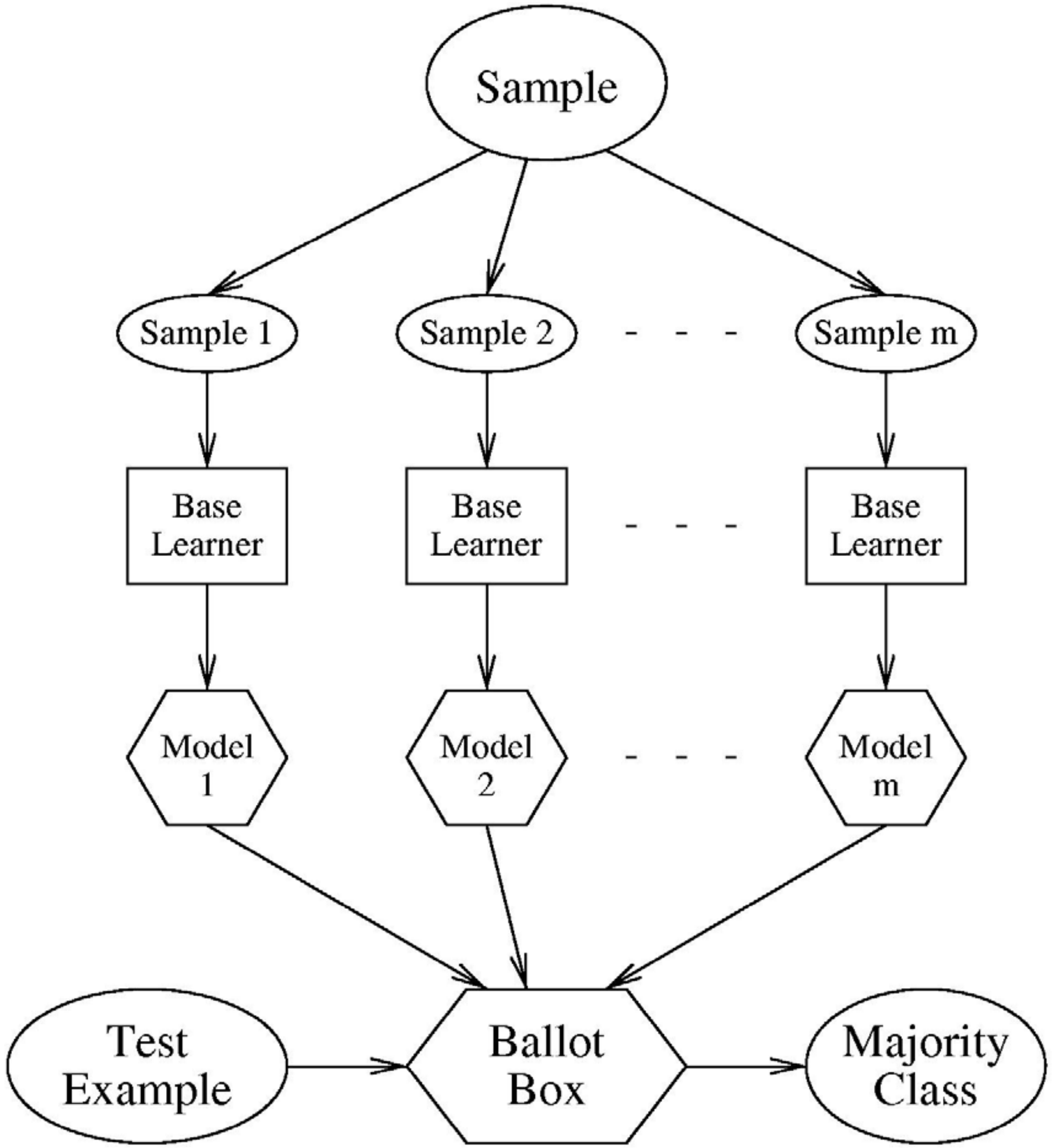
- Often this improves accuracy by a lot

- Many Methods
  - Bagging
  - Boosting
  - Stacking

*A model is the learned decision rule. It can be as simple as a hyperplane in n-space (ie. a line in 2D or plane in 3D) or in the form of a decision tree or other modern classifier.

Bagging

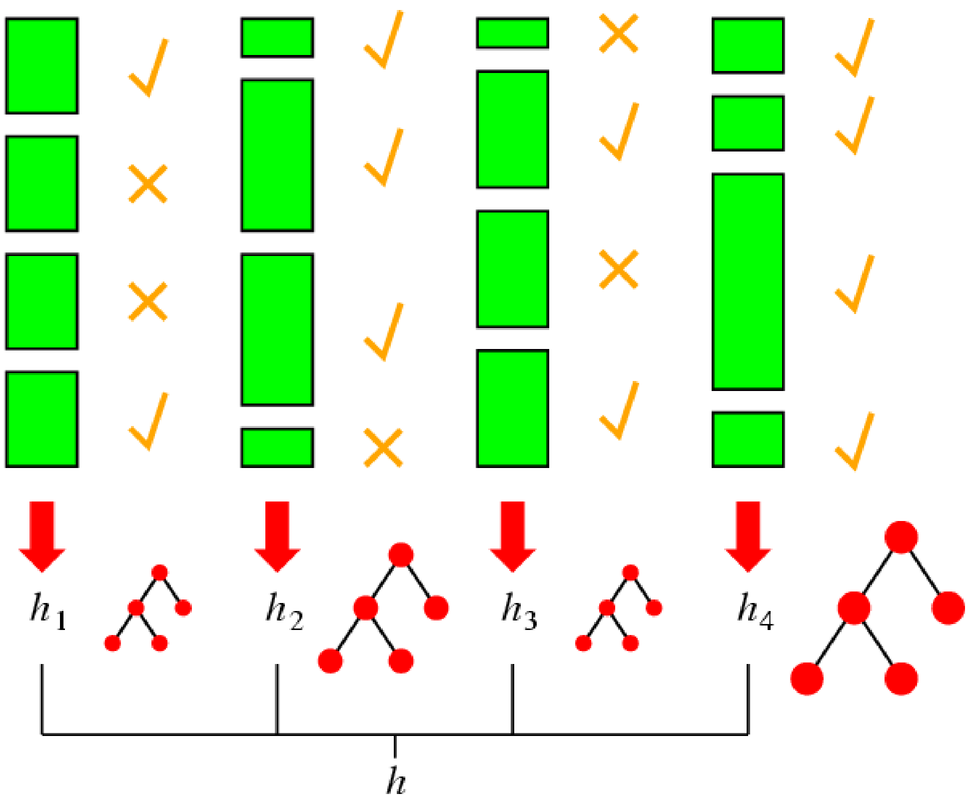
- Generate bootstrap replicates of the training set by sampling with replacement
- Learn one model on each replicate
- Combine by uniform voting


Sample
Sample 1
Sample 2
Sample m
Base Learner
Base Learner
Base Learner
Model 1
Model 2
Model m
Test Example
Ballot Box
Majority Class

Boosting

- Maintain a vector of weights for samples
- Initialize with uniform weights
- Loop
  - Apply learner to weighted samples
  - Increase weights of misclassified ones
- Combine models by weighted voting

Idea of Boosting



ADABoost

- ADABoost boosts the accuracy of the original learning algorithm.

- If the original learning algorithm does slightly better than 50% accuracy, ADABoost with a large enough number of classifiers is guaranteed to classify the training data perfectly.

ADABoost Weight Updating (from Fig 18.34 text)

/* First find the sum of the weights of the misclassified samples */

for j = 1 to N do /* go through training samples */

if h[m](x_j) <> y_j then error <- error + w_j

/* Now use the ratio of error to 1-error to change the weights of the correctly classified samples */

for j = 1 to N do

if h[m](x_j) = y_j then w[j] <- w[j] * error/(1-error)

Example

- Start with 4 samples of equal weight .25.
- Suppose 1 is misclassified. So error = .25.
- The ratio comes out .25/.75 = .33
- The correctly classified samples get weight of .25*.33 = .0825

.2500
.0825
.0825
.0825

What's wrong? What should we do?

We want them to add up to 1, not .4975.

Answer: To normalize, divide each one by their sum (.4975).

Sample Application: Insect Recognition

Using circular regions of interest selected by an interest operator, train a classifier to recognize the different classes of insects.

Boosting Comparison

- **ADTree classifier only** **(alternating decision tree)**

- Correctly Classified Instances 268 70.1571 %
- Incorrectly Classified Instances 114 29.8429 %
- Mean absolute error 0.3855
- Relative absolute error 77.2229 %

Classified as ->	Hesperperla	Doroneuria
Real Hesperperlas	167	28
Real Doroneuria	51	136

Boosting Comparison

<u>AdaboostM1 with ADTree classifier</u>

- Correctly Classified Instances 303 79.3194 %
- Incorrectly Classified Instances 79 20.6806 %
- Mean absolute error 0.2277
- Relative absolute error 45.6144 %

Classified as ->	Hesperperla	Doroneuria
Real Hesperperlas	167	28
Real Doroneuria	51	136

Boosting Comparison

- **<u>RepTree classifier only</u> (reduced error pruning)**

- Correctly Classified Instances 294 75.3846 %
- Incorrectly Classified Instances 96 24.6154 %
- Mean absolute error 0.3012
- Relative absolute error 60.606 %

Classified as ->	Hesperperla	Doroneuria
Real Hesperperlas	169	41
Real Doroneuria	55	125

Boosting Comparison

AdaboostM1 with RepTree classifier

- Correctly Classified Instances 324 83.0769 %
- Incorrectly Classified Instances 66 16.9231 %
- Mean absolute error 0.1978
- Relative absolute error 39.7848 %

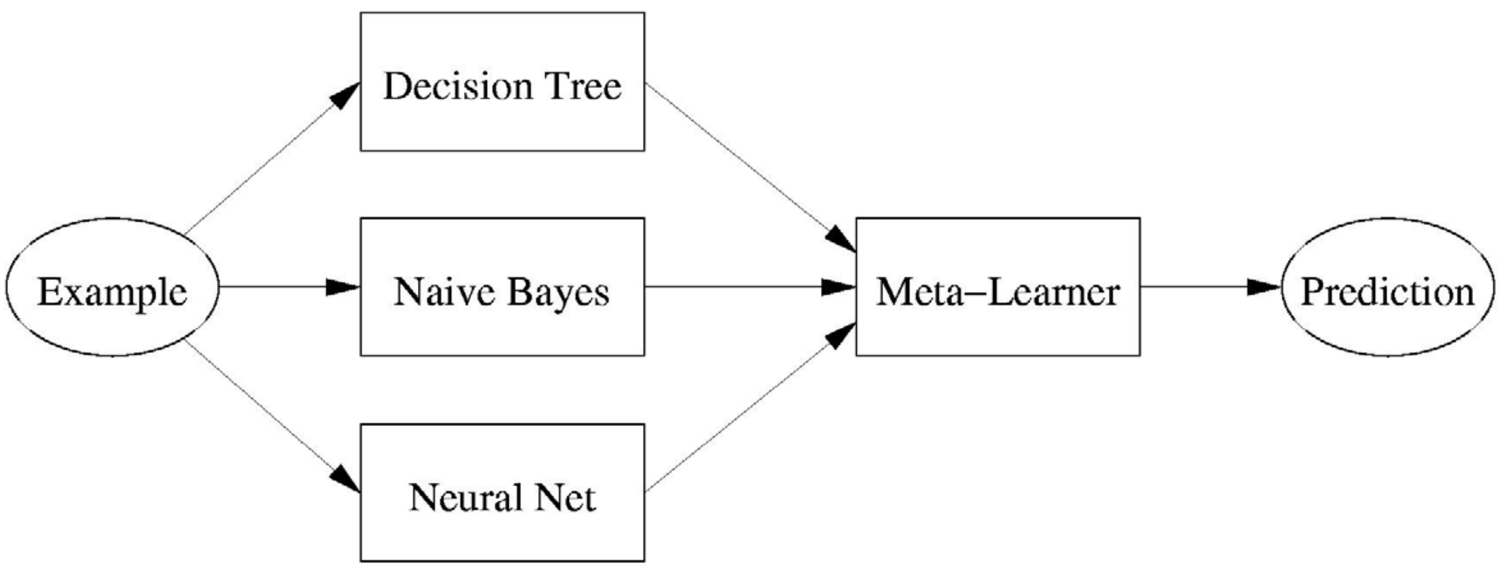
Classified as ->	Hesperperla	Doroneuria
Real Hesperperlas	180	30
Real Doroneuria	36	144

References

- **AdaboostM1**: Yoav Freund and Robert E. Schapire (1996). "Experiments with a new boosting algorithm". Proc International Conference on Machine Learning, pages 148-156, Morgan Kaufmann, San Francisco.
- **ADTree**: Freund, Y., Mason, L.: "The alternating decision tree learning algorithm". Proceeding of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, (1999) 124-133.

Stacking

- Apply multiple base learners
  (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation:
  Meta-L. inputs = Predictions on left-out examples



Random Forests

- Tree bagging creates decision trees using the bagging technique. The whole set of such trees (each trained on a random sample) is called a decision forest. The final prediction takes the average (or majority vote).

- Random forests differ in that they use a modified tree learning algorithm to reduce variance.

Random Forest Algorithm

- At each split point in constructing the tree, select a random sample of attributes.
- Then compute which of *those* gives the highest information gain.
- If there are n attributes, the default choice is to randomly pick sqrt(n) attributes for classification problems.
- Furthermore, can use randomness to select the split point *values*. This leads to Extremely Randomized Trees. (ExtraTrees)

Back to Stone Flies

Random forest of 10 trees, each constructed while considering 7 random features.
Out of bag error: 0.2487. Time taken to build model: 0.14 seconds

Correctly Classified Instances	292	76.4398 % (81.4 with AdaBoost)
Incorrectly Classified Instances	90	23.5602 %
Kappa statistic	0.5272	
Mean absolute error	0.344	
Root mean squared error	0.4069	
Relative absolute error	68.9062 %	
Root relative squared error	81.2679 %	
Total Number of Instances	382	

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.69	0.164	0.801	0.69	0.741	0.848	cal
	0.836	0.31	0.738	0.836	0.784	0.848	dor
WAvg.	0.764	0.239	0.769	0.764	0.763	0.848	

```
  a   b   <-- classified as
129  58 |   a = cal
 32 163 |   b = dor
```

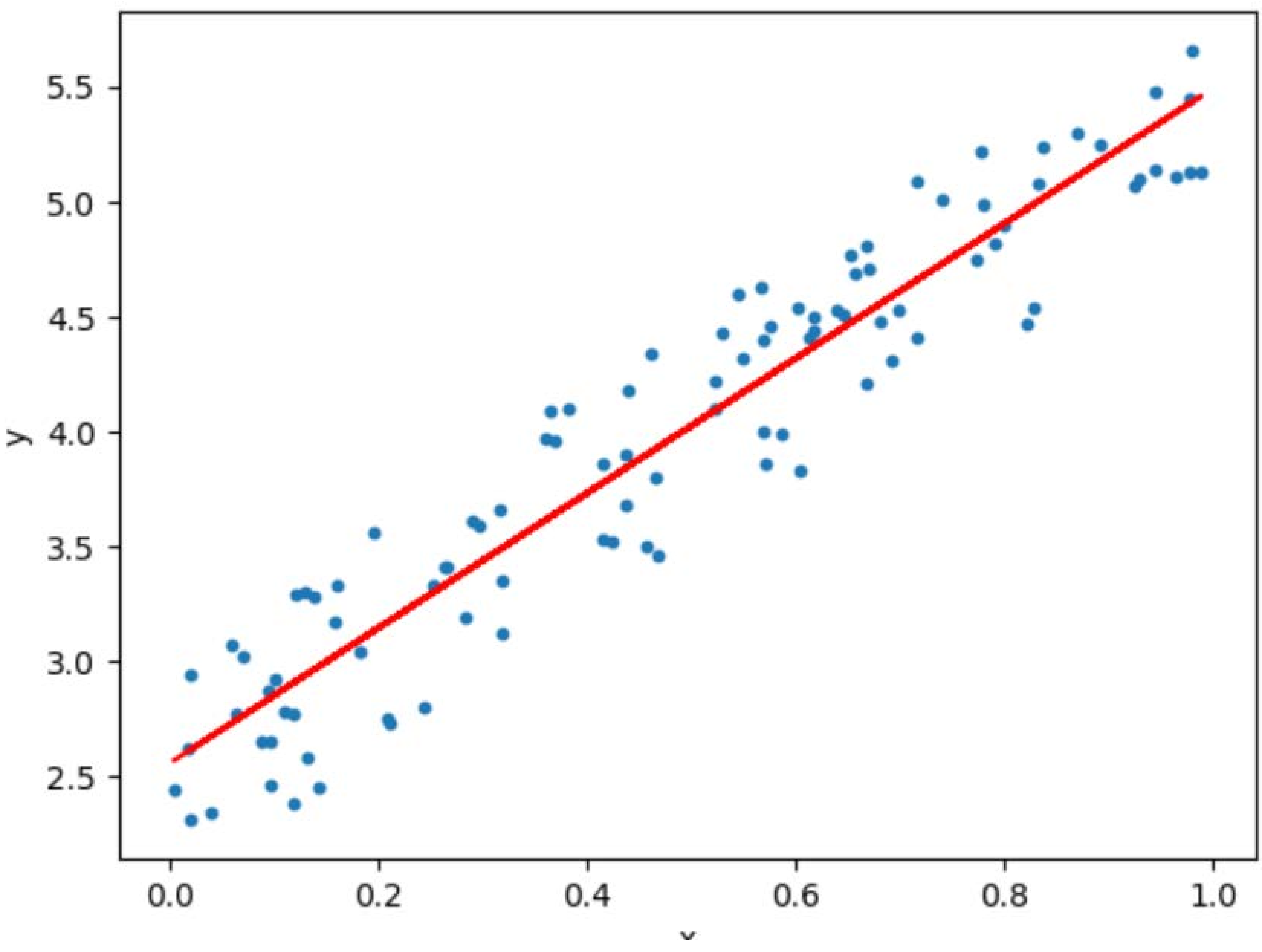
More Terminology in Learning

- Loss: We've been talking about errors, but modern machine learning theory talks about *loss*.
- We minimize a loss function, rather than maximizing a utility function.
- The loss function L(x,y,y') is defined as the amount of utility lost by predicting h(x) = y' when the correct answer is f(x)=y.

Regression

- One common kind of learning is linear regression.
- Simple case: univariate linear regression
- $y = w_1 x + w_0$
- **w** is the vector (w_1, w_0)
- The linear function with those weights is

  $h_{\mathbf{w}} = w_1 x + w_0$
- Loss($h_{\mathbf{w}}$) = $\Sigma\ (y_j - (w_1 x_j + w_0))^2$ for j=1 to n
- Summed over all training examples.



5.5
5.0
4.5
4.0
3.5
3.0
2.5
y
0.0
0.2
0.4
0.6
0.8
1.0

Solution

- We want to find $w^* = \text{argmin}_{\mathbf{w}} \text{Loss}(h_{\mathbf{w}})$
- This is computed by taking partial derivatives wrt to w_0 and w_1 and setting them to zero.
- The more general solution for the more general case, and for neural nets, is called gradient descent.
- We will look at it wrt neural nets.