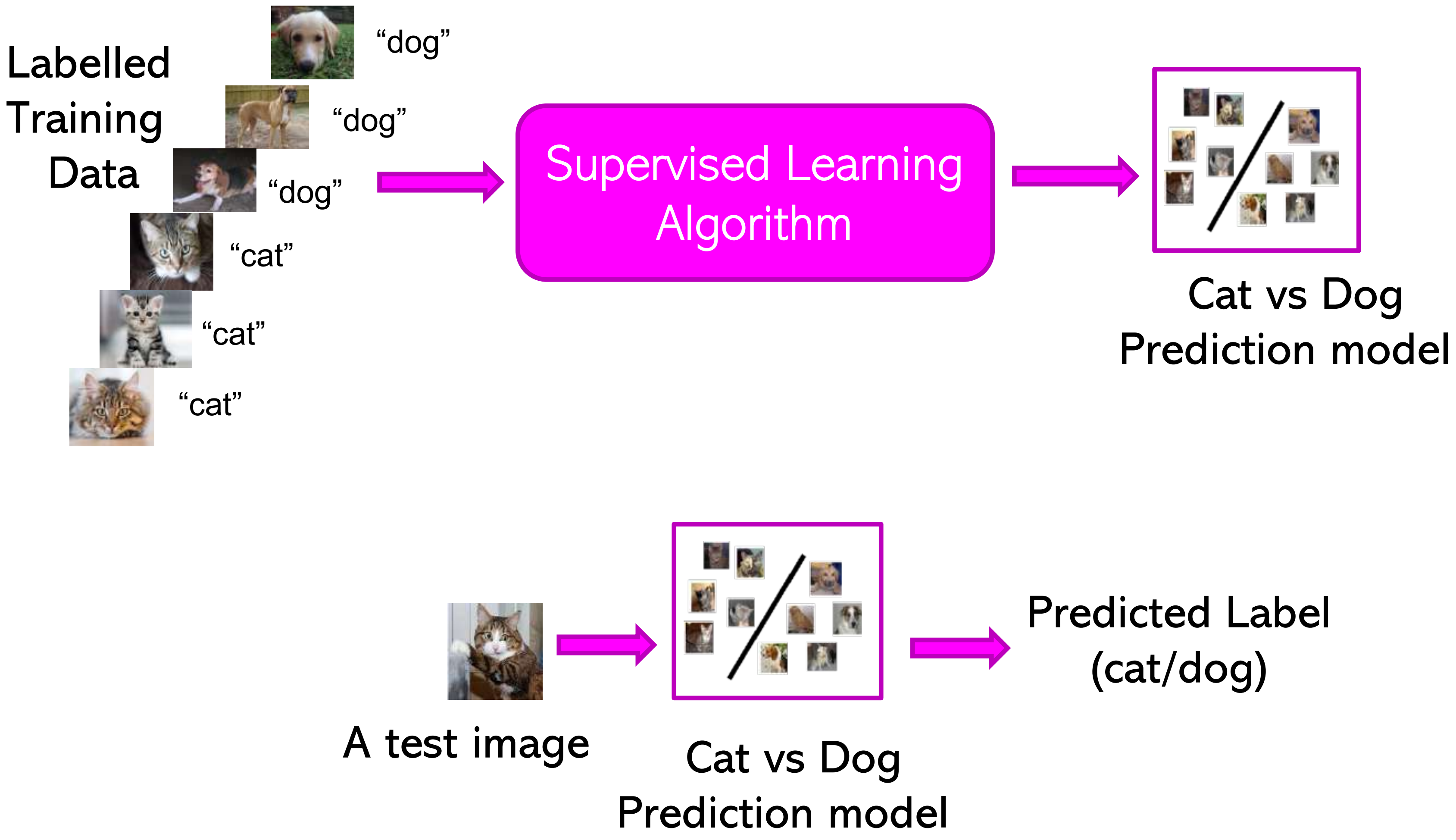
Classification

Dr. Abhijnan Chakraborty
Department of Computer Science & Engg.,
Indian Institute of Technology Kharagpur
https://cse.iitkgp.ac.in/~abhijnan

Supervised Learning


Labelled Training Data
"dog"
"dog"
"dog"
"cat"
"cat"
"cat"
Supervised Learning Algorithm
Cat vs Dog Prediction model
A test image
Cat vs Dog Prediction model
Predicted Label (cat/dog)

Types of Supervised Learning Problems

- Consider building an ML module for an e-mail client
- Some tasks that we may want this module to perform
  - Predicting whether an email of spam or normal: Binary Classification
  - Predicting which of the many folders the email should be sent to: Multi-class Classification
  - Predicting all the relevant tags for an email: Tagging or Multi-label Classification
  - Predicting what's the spam-score of an email: Regression
  - Predicting which email(s) should be shown at the top: Ranking
  - Predicting which emails are work/study-related emails: One-class Classification
- These predictive modeling tasks can be formulated as supervised learning problems

Classification: Definition

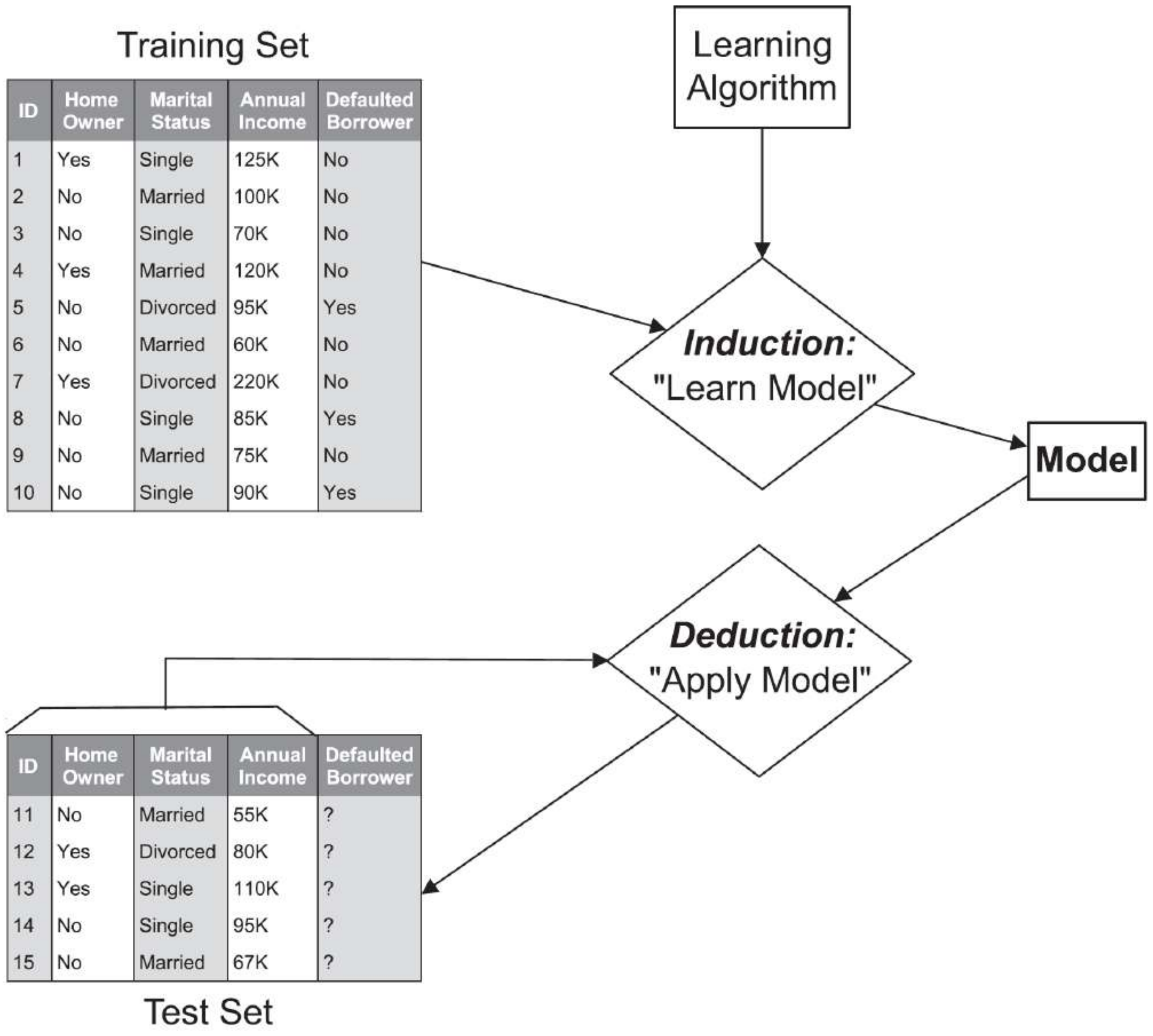
- Given a collection of records (training set)
  - Each record is characterized by a tuple (x,y), where x is the attribute set and y is the class label
    - x: attribute, predictor, independent variable, input
    - y: class, response, dependent variable, output

- Task:
  - Learn a model that maps each attribute set x into one of the predefined class labels y

Examples of Classification Task

Task	Attribute set, x	Class label, y
Categorizing email messages	Features extracted from email message header and content	spam or non-spam
Identifying tumor cells	Features extracted from x-rays or MRI scans	malignant or benign cells
Cataloging galaxies	Features extracted from telescope images	Elliptical, spiral, or irregular-shaped galaxies

General Approach for Building Classifiers



Training Set

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
11	No	Married	55K	?
12	Yes	Divorced	80K	?
13	Yes	Single	110K	?
14	No	Single	95K	?
15	No	Married	67K	?

Test Set

Classification Techniques

- Base Classifiers
  - Learning with Prototypes (LwP)
  - Nearest-neighbor (kNN)
  - Naïve Bayes (NB)
  - Support Vector Machines (SVM)
  - Decision Tree (DT)
  - Neural Networks, Deep Neural Nets

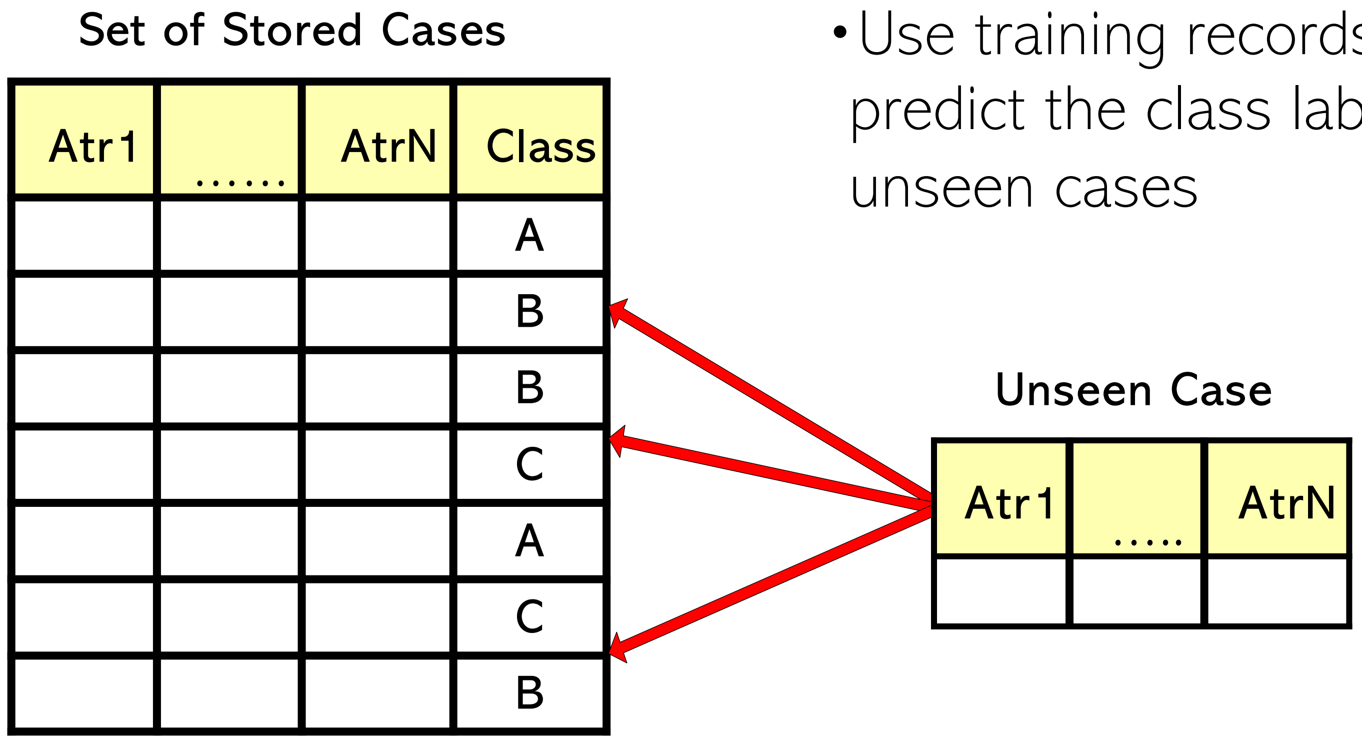
- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

Classification Techniques

- Base Classifiers
  - Learning with Prototypes (LwP)
  - Nearest-neighbor (kNN)
  - Naïve Bayes (NB)
  - Support Vector Machines (SVM)
  - Decision Tree (DT)
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

Instance Based Classifiers



- Store the training records
- Use training records to predict the class label of unseen cases

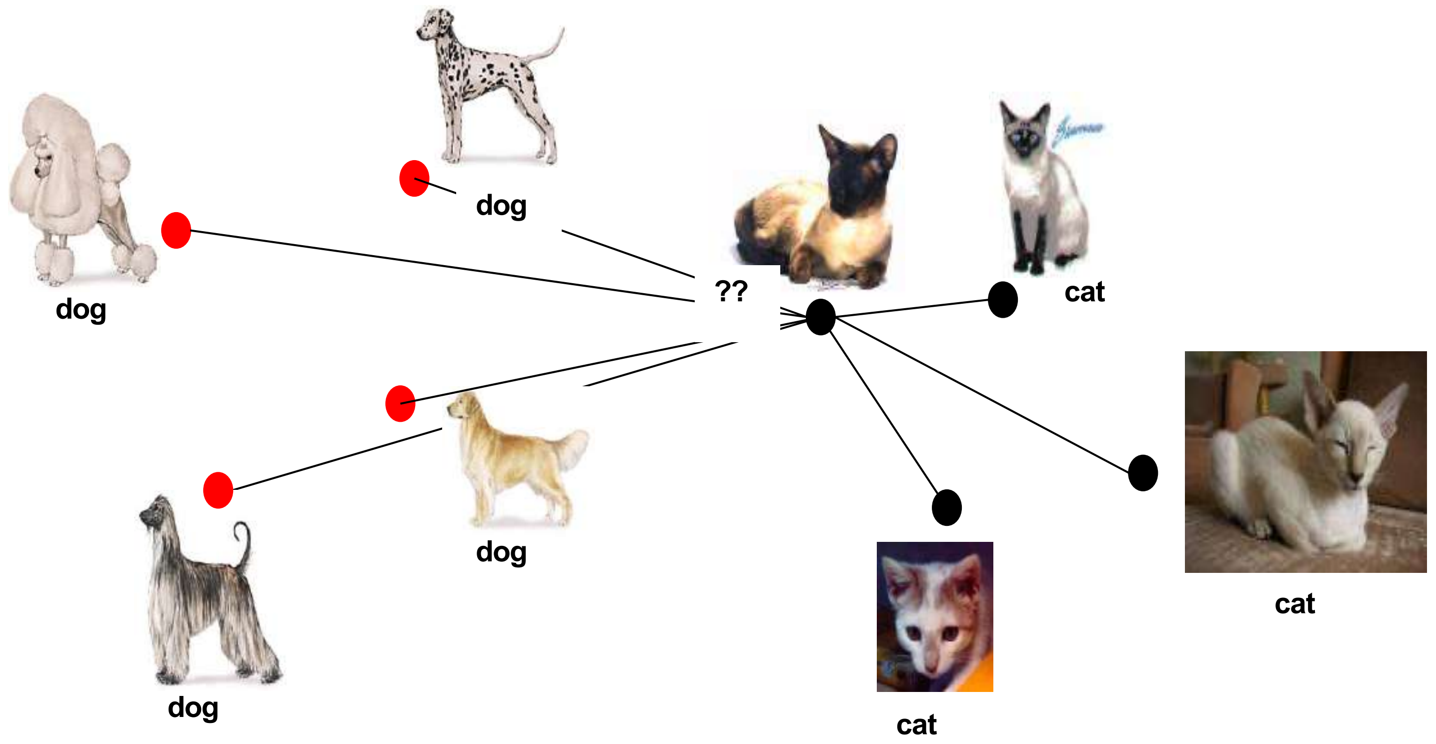
Instance Based Classifiers

- Rote-learner
  - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly

- Nearest neighbor
  - Uses k "closest" points (nearest neighbors) for performing classification

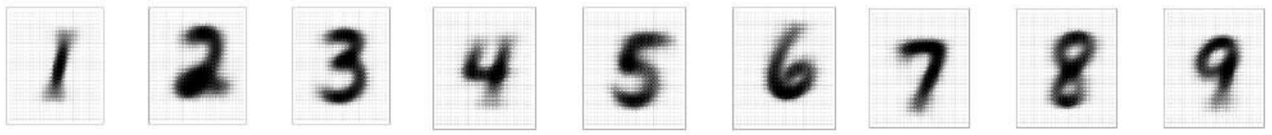
- Learning with Prototypes

Learning to Predict Categories



Learning with Prototypes (LwP)

- Basic idea: Represent each class by a "prototype" vector
- Class Prototype: The "mean" or "average" of inputs from that class

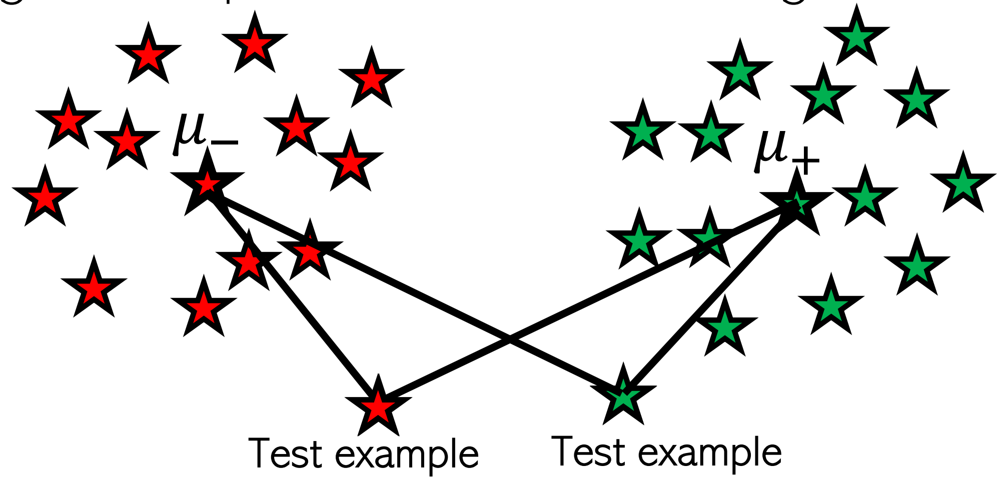


- Predict label of each test input based on its distances from the class prototypes
  - Predicted label will be the class closest to the test input
- How we compute distances may influence the accuracy of this model (Euclidean, Manhattan, Mahalanobis, ...)

Learning with Prototypes: An Illustration

- Suppose the task is binary classification (two classes assumed pos and neg)
- Training data: $\mathbf{N}$ labelled examples $\{(\mathbf{x_n}, \mathbf{y_n})\}_{\mathbf{n=1}}^{\mathbf{N}}$, $\mathbf{x_n} \in \mathbb{R}^{\mathbf{D}}$, $\mathbf{y_n} \in \{-1, +1\}$
  - Assume $\mathbf{N_+}$ example from positive class, $\mathbf{N_-}$ examples from negative class
  - Assume green is positive and red is negative

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n$$

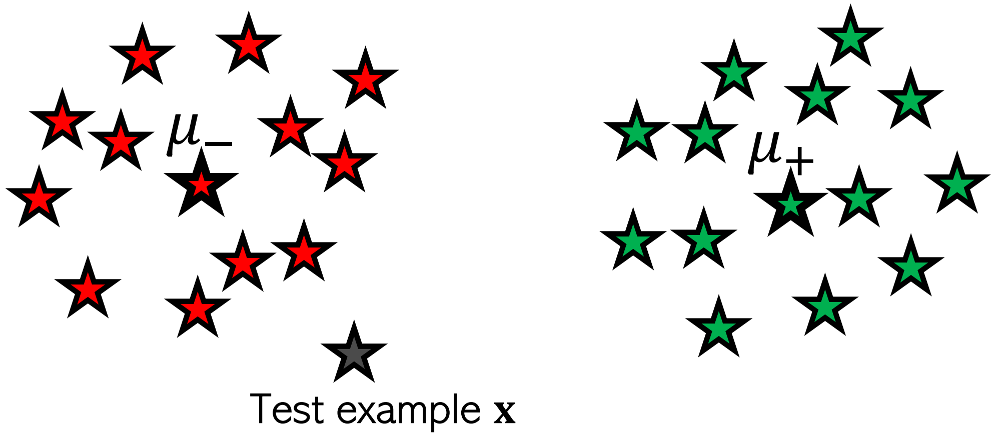


$$\mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$

LwP straightforwardly generalizes to more than 2 classes as well (multi-class classification) – K prototypes for K classes

LwP: The Prediction Rule

- What does the prediction rule for LwP look like?
- Assume we are using Euclidean distance

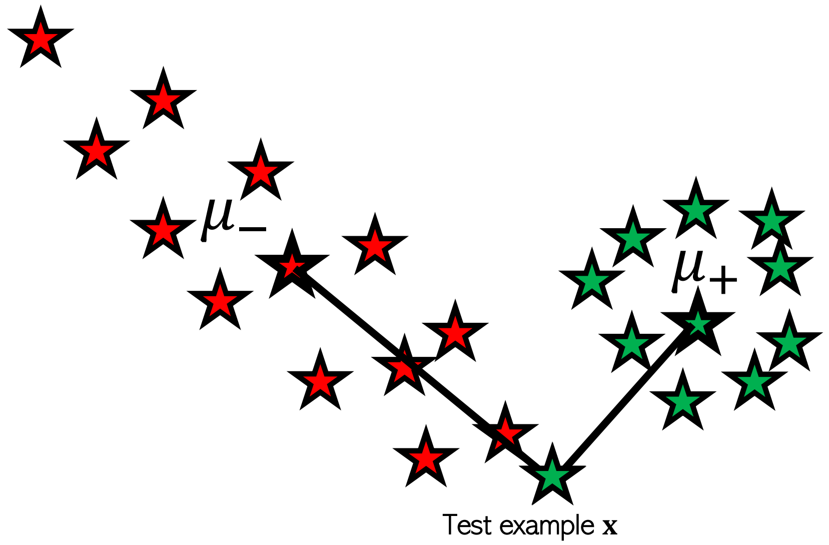


Prediction Rule: Predict label as +1 if

$f(\mathbf{x}) = ||\boldsymbol{\mu}_- - \mathbf{x}||^2 - ||\boldsymbol{\mu}_+ - \mathbf{x}||^2 > 0$ otherwise -1

LwP: Some Failure Cases

- Here is a case where LwP with Euclidean distance may not work well



- In general, if classes are not equisized and spherical, LwP with Euclidean distance will usually not work well
- Can use feature scaling or use Mahalanobis distance to handle such cases

Summary

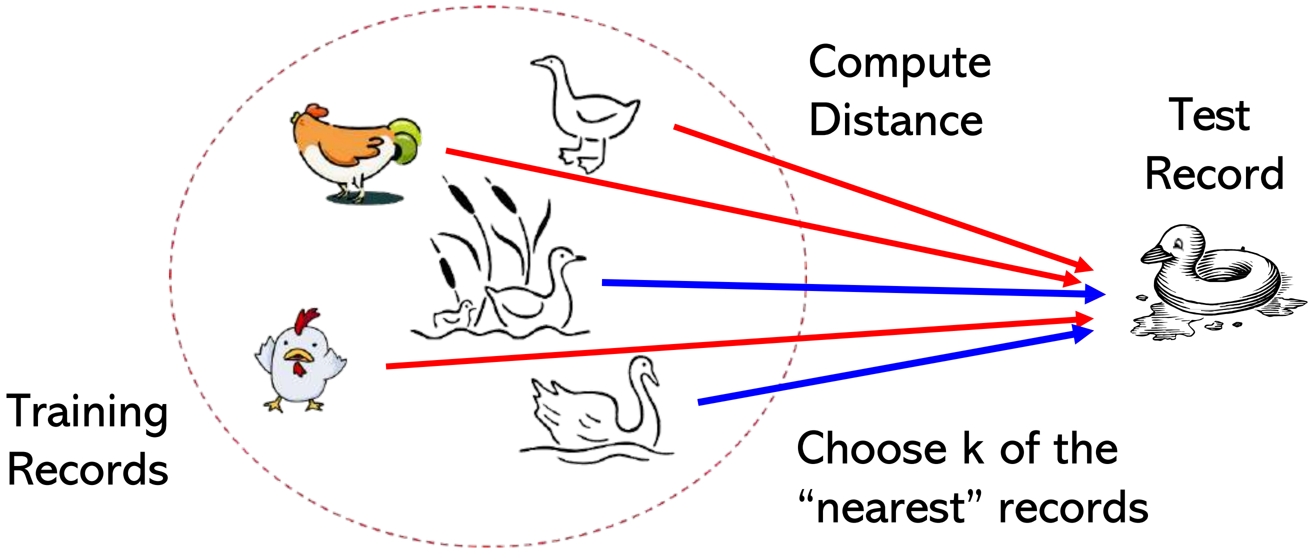
- Very simple, interpretable, and lightweight model
  - Just requires computing and storing class prototype vectors

- Works with any number of classes (thus for multi-class classification as well)

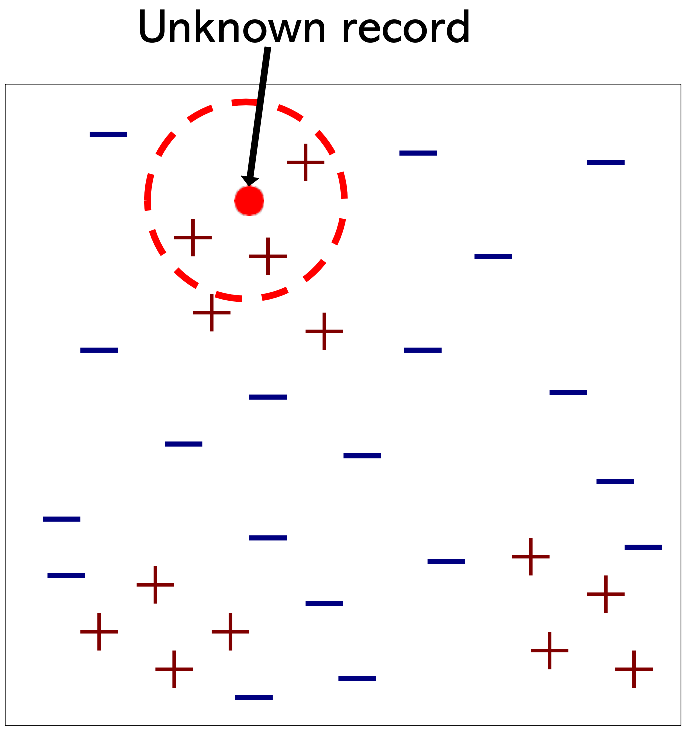
- Can be generalized in various ways to improve it further, e.g.,
  - Modeling each class by a probability distribution rather than just a prototype vector
  - Using distances other than the standard Euclidean distance (e.g., Mahalanobis)

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



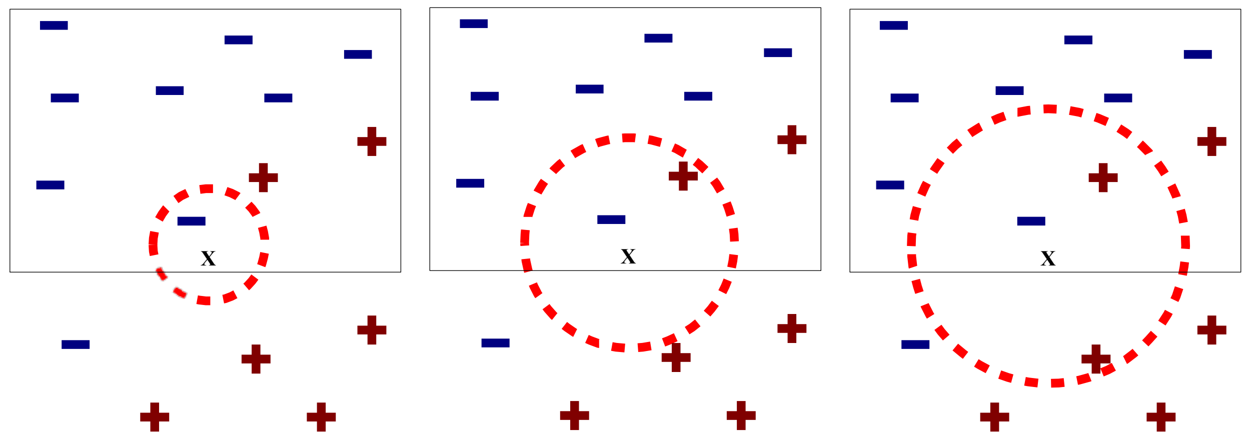
Nearest Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - Value of k, number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training records
  - Identify k nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record

Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance $$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$
  - Manhattan distance $$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$
  - q norm distance $$d(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^q \right)^{\frac{1}{q}}$$

Nearest Neighbor Classification

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors

$$y' = \underset{v}{\operatorname{argmax}} \sum_{x_i, y_i \in D_z} I(v = y_i)$$

where D_z is the set of k closest training examples to z.

- Weigh the vote according to distance

$$y' = \underset{v}{\operatorname{argmax}} \sum_{x_i, y_i} w_i \times I(v = y_i)$$

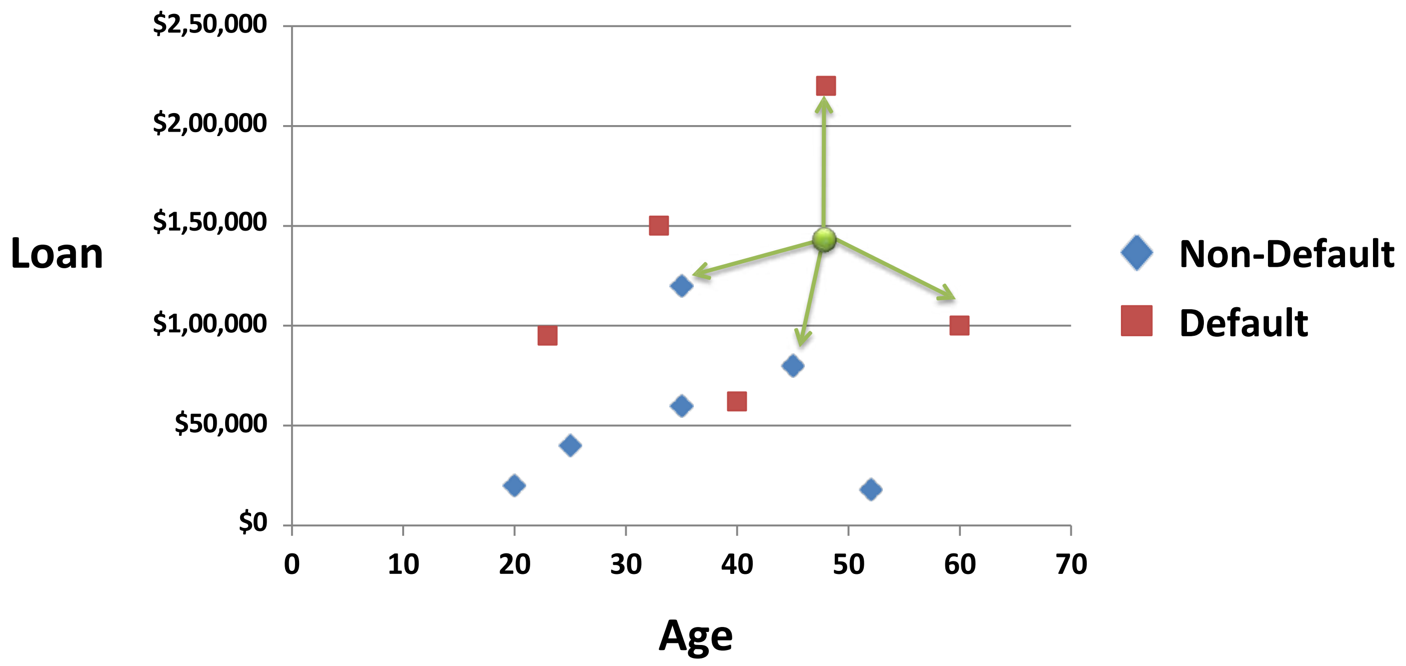
weight factor $w = 1/d^2$

The kNN Classification Algorithm

Let k be the number of nearest neighbors and D be the set of training examples.

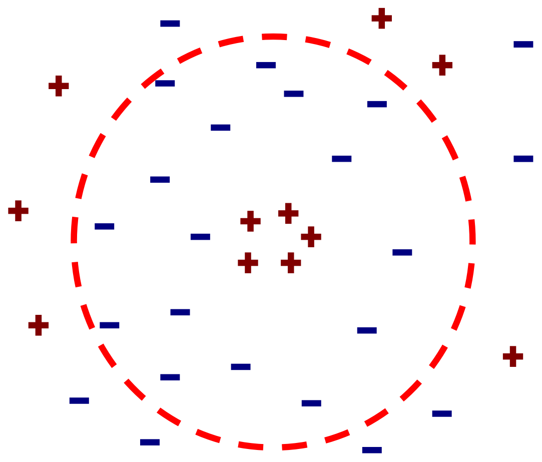
```
for each test example z = (x',y') do
    Compute d(x',x), the distance between z and every example, (x,y) ϵ D
    Select $D_z \subseteq D$, the set of k closest training examples to z
    y' = $\underset{v}{\operatorname{argmax}} \sum_{x_i, y_i \in D_z} I(v = y_i)$
end for
```

kNN Classification



Value of k

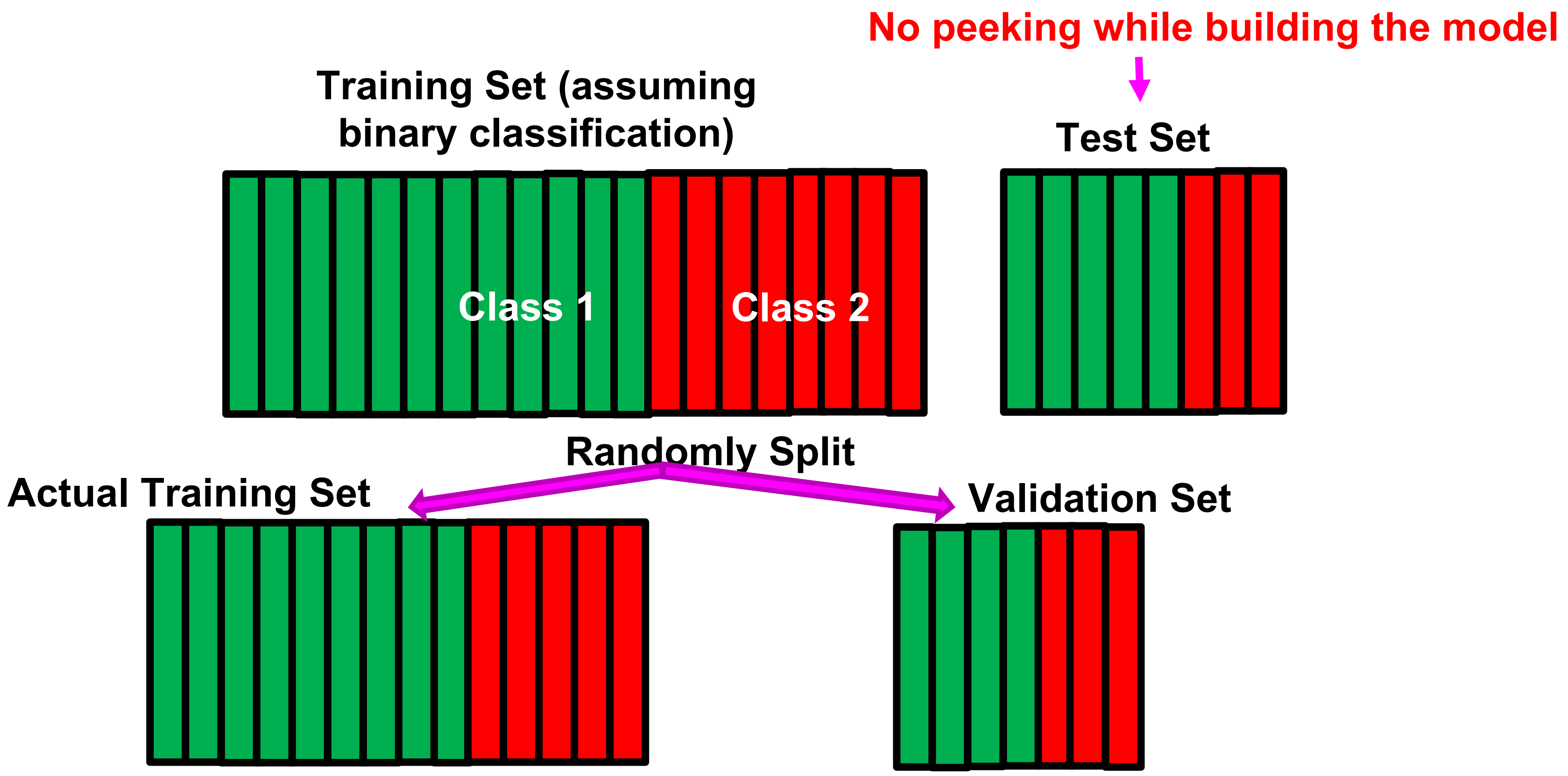
- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



Hyperparameter Selection

- Every ML model has some hyperparameters that need to be tuned, e.g.,
  - *k* in kNN
  - Choice of distance to use in LwP or nearest neighbors
- Would like to choose hyper parameter values that would give best performance on test data

Cross-Validation


No peeking while building the model
Training Set (assuming binary classification)
Class 1
Class 2
Test Set
Randomly Split
Actual Training Set
Validation Set

- Randomly split the original training data into actual training set and validation set.
- Using the actual training set, train several times, each time using a different value of the hyperparameter.
- Pick the hyperparameter value that gives best accuracy on the validation set

Cross-Validation

- ❑ What if the random split is unlucky (i.e., validation data is not like test data)?
- ❑ Try multiple splits and pick the hyperparameter value that gives the best average CV accuracy across all such splits
- ❑ If we use N splits, this is called N–fold cross validation
- ❑ Note: Not just hyperparameter selection; we can also use Cross Validation to pick the best ML model from a set of different ML models

Summary

- kNN classifiers are lazy learners
- It does not build models explicitly
  - Unlike eager learners such as decision tree
  - Classifying unknown records are relatively expensive

Sources of Error

Generalization

Training set (labels known)

Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Generalization

- Components of generalization error
  - Bias: how much the average model over all training sets differ from the true model?
  - Error due to inaccurate assumptions/simplifications made by the model
  - Variance: how much models estimated from different training sets differ from each other

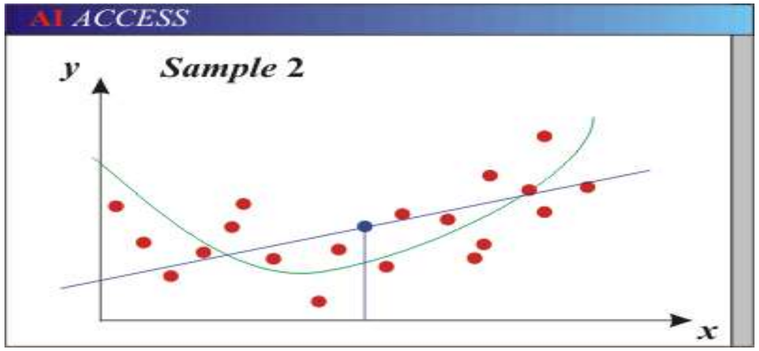
Generalization

- Underfitting: model is too "simple" to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error

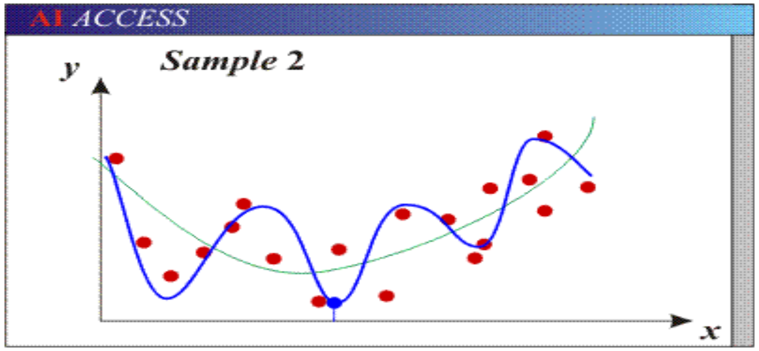
- Overfitting: model is too "complex" and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

No Free Lunch Theorem

Bias-Variance Tradeoff



- Models with too few parameters are inaccurate because of a large bias (not enough flexibility)



- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample)

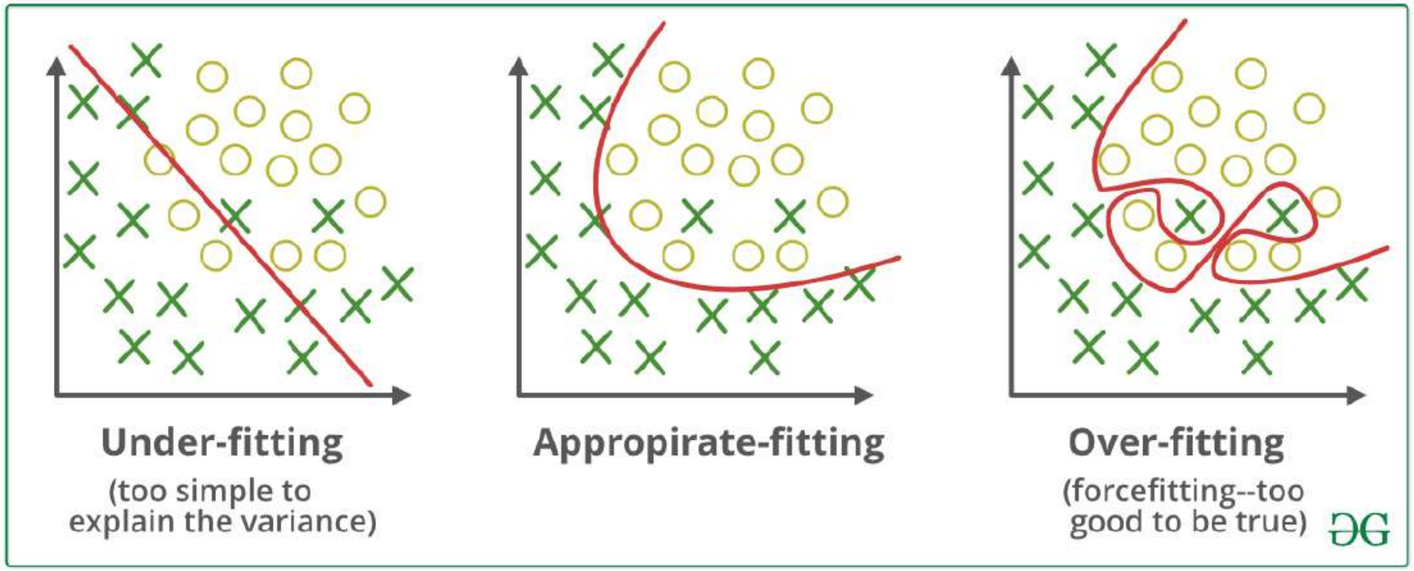
Bias-Variance Tradeoff

$$E(MSE) = noise^2 + bias^2 + variance$$

Unavoidable error $\rightarrow$ $noise^2$

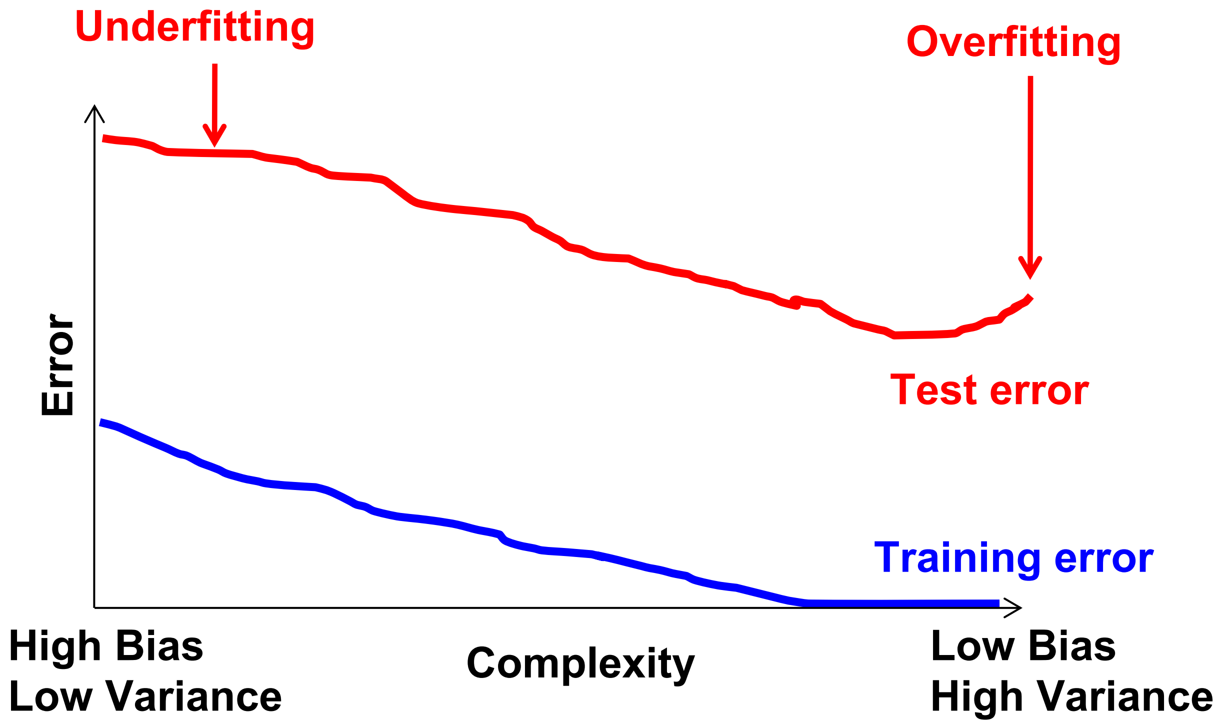
Error due to incorrect assumptions $\rightarrow$ $bias^2$

Error due to variance of training samples $\rightarrow$ $variance$

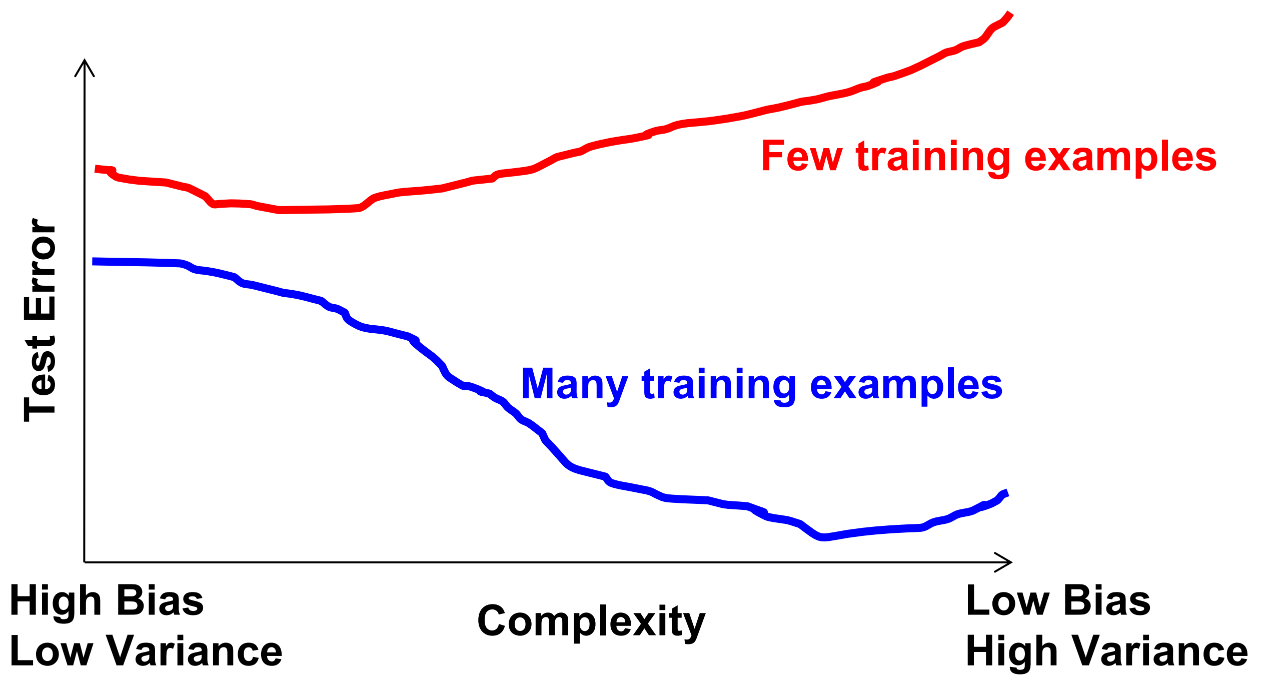

Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

See the following for further explanations on bias-variance:
https://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

Bias-Variance Tradeoff

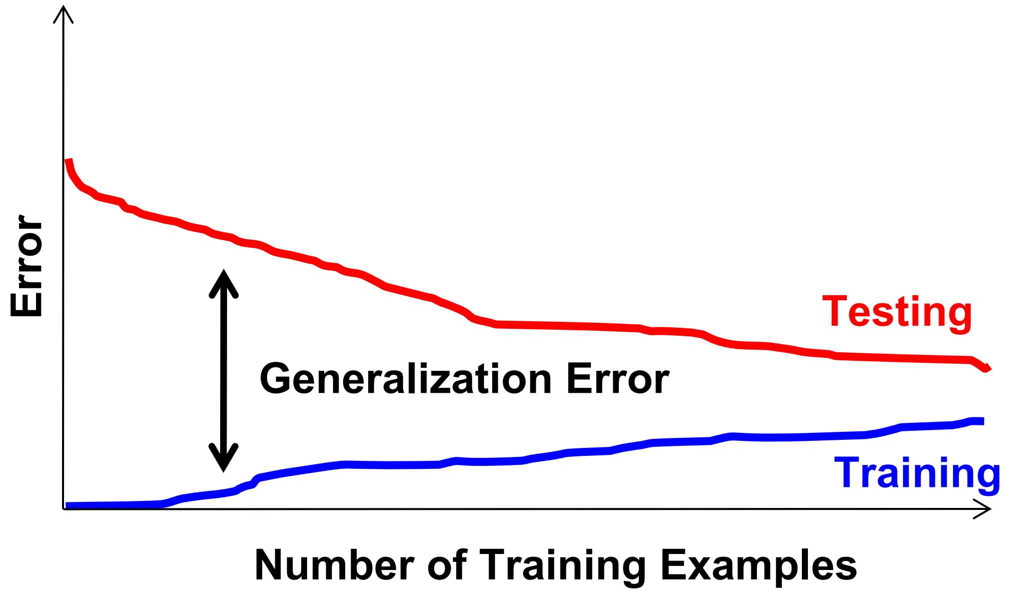


Bias-Variance Tradeoff



Effect of Training Size

Fixed prediction model



Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize

- Three kinds of error
  - **Inherent**: unavoidable
  - **Bias**: due to over-simplifications
  - **Variance**: due to inability to perfectly estimate parameters from limited data

How to Reduce Variance?

- Choose a simpler classifier
- Cross-validate the parameters
- Get more training data

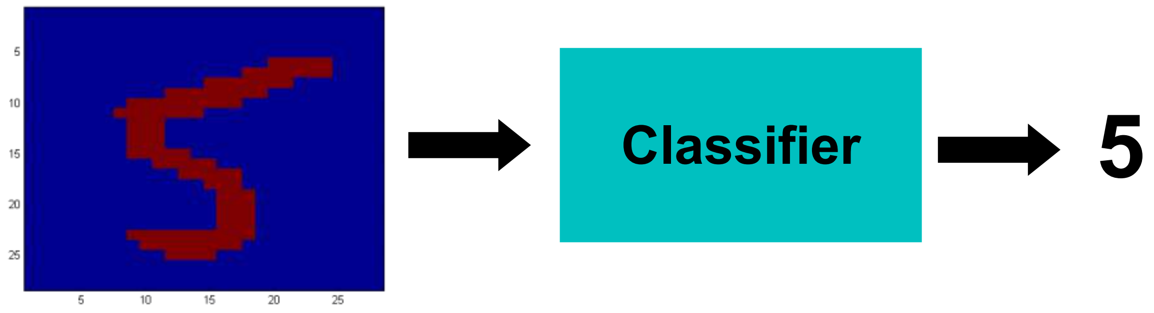
Classification Techniques

- Base Classifiers
  - Learning with Prototypes (LwP)
  - Nearest-neighbor (kNN)
  - Naïve Bayes (NB)
  - Support Vector Machines (SVM)
  - Decision Tree (DT)
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

An Example Application

- Digit Recognition



- Task: Given features $X_1, X_2, \ldots, X_n$; Predict a label Y
  - $X_1, \ldots, X_n \in \{0,1\}$ (Blue vs. Red pixels)
  - $Y \in \{5,6\}$ (predict whether a digit is a 5 or a 6)

The Bayes Classifier

- A good strategy is to predict:

$$\arg\max_{Y} P(Y|X_1, \ldots, X_n)$$

- For example: what is the probability that the image represents a 5 given its pixels?

- How do we compute that?

The Bayes Classifier

- Use Bayes Rule!

$$P(Y|X_1,\ldots,X_n) = \frac{\overbrace{P(X_1,\ldots,X_n|Y)}^{\text{Likelihood}}\overbrace{P(Y)}^{\text{Prior}}}{\underbrace{P(X_1,\ldots,X_n)}_{\text{Marginal Distribution}}}$$

- Why did this help?
  - We think that we might be able to specify how features are "generated" by the class label

The Bayes Classifier

- Let's expand this for our digit recognition task:

$$P(Y=5|X_1,\ldots,X_n) = \frac{P(X_1,\ldots,X_n|Y=5)P(Y=5)}{P(X_1,\ldots,X_n|Y=5)P(Y=5)+P(X_1,\ldots,X_n|Y=6)P(Y=6)}$$

$$P(Y=6|X_1,\ldots,X_n) = \frac{P(X_1,\ldots,X_n|Y=6)P(Y=6)}{P(X_1,\ldots,X_n|Y=5)P(Y=5)+P(X_1,\ldots,X_n|Y=6)P(Y=6)}$$

- To classify, we'll simply compute these two probabilities and predict based on which one is greater

Model Parameters

- For the Bayes classifier, we need to "learn" two functions, the likelihood and the prior
- The problem with explicitly modeling $P(X_1,\ldots,X_n|Y)$ is that there are usually way too many parameters:
  - High space and time complexity
  - We'll need tons of training data (which is usually not available)

The Naïve Bayes Model

- The *Naïve Bayes* Assumption: Assume that all features are independent given the class label Y

- Equationally speaking:

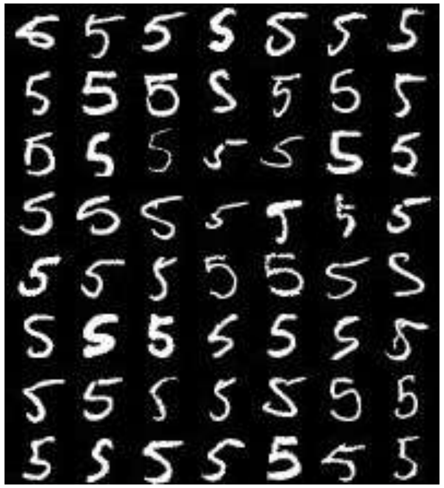
$$P(X_1, \ldots, X_n | Y) = \prod_{i=1}^{n} P(X_i | Y)$$

Why is this Useful?

- # of parameters for modeling $P(X_1,...,X_n|Y)$:
  - $2(2^n-1)$
- # of parameters for modeling $P(X_1|Y),...,P(X_n|Y)$
  - $2n$

Naïve Bayes Training

- Now that we've decided to use a Naïve Bayes classifier, we need to train it with some data:



MNIST Training Data

*MNIST dataset is a large database of handwritten digits

Naïve Bayes Training

- Training in Naïve Bayes is easy:
  - Estimate P(Y=v) as the fraction of records with Y=v

$$P(Y=v) = \frac{Count(Y=v)}{\#\ records}$$

  - Estimate $P(X_i=u|Y=v)$ as the fraction of records with Y=v for which $X_i=u$

$$P(X_i=u|Y=v) = \frac{Count(X_i=u \wedge Y=v)}{Count(Y=v)}$$

- This corresponds to **Maximum Likelihood Estimation** (MLE) of model parameters

Naïve Bayes Training

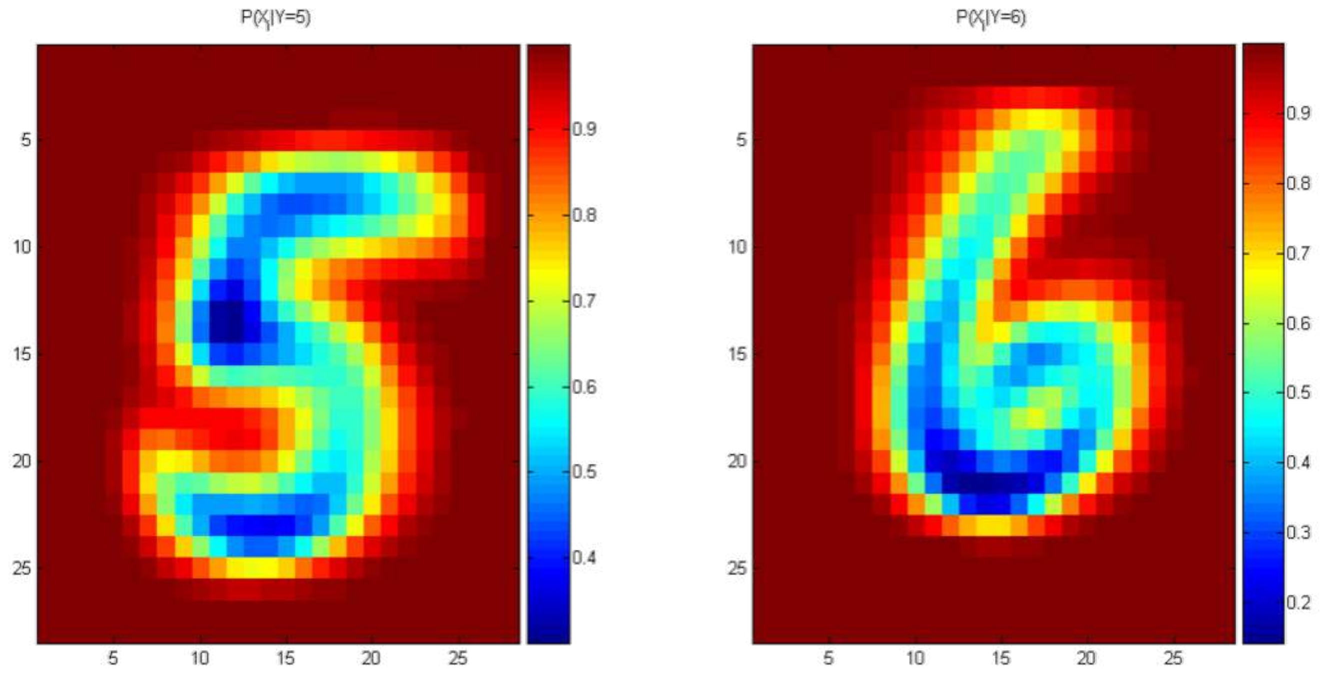
- In practice, some of these counts can be zero
- Fix this by adding "virtual" counts:

$$P(X_i = u|Y = v) = \frac{Count(X_i = u \wedge Y = v) + 1}{Count(Y = v) + 2}$$

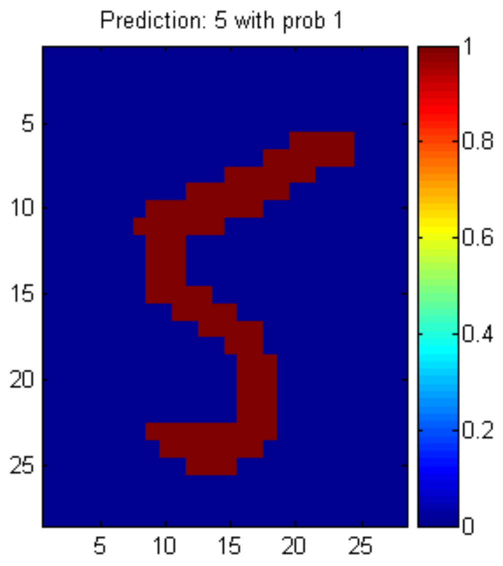
- This is called smoothing

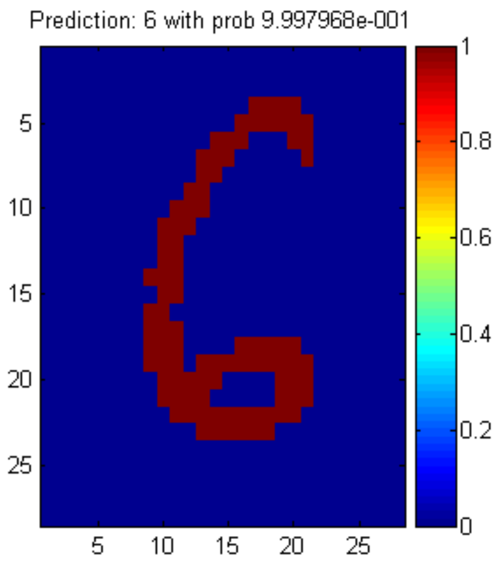
Naïve Bayes Training

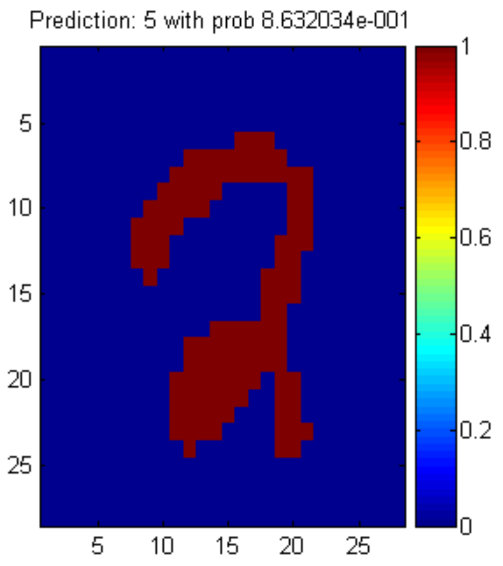
- For binary digits, training involves averaging all the trainings of digit five together and all the training of digit six together



Naïve Bayes Classification







Naïve Bayes Classifier: Another Example

The weather data, with counts and probabilities

outlook			temperature			humidity			windy			play	
	yes	no		yes	no		yes	no		yes	no	yes	no
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

A new day

outlook	temperature	humidity	windy	play
sunny	cool	high	true	?

Another Example …

- Likelihood of yes

$$= \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14} = 0.0053$$

- Likelihood of no

$$= \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} \times \frac{5}{14} = 0.0206$$

- Therefore, the prediction is No

Naïve Bayes Assumption

- Recall the Naïve Bayes assumption:
  - All features are independent given the class label Y
- Does this hold for the digit recognition problem?

Exclusive-OR Example

- Where conditional independence fails:
  - $Y = XOR(X_1, X_2)$

X_1	X_2	$P(Y=0 \mid X_1,X_2)$	$P(Y=1 \mid X_1,X_2)$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

Naïve Bayes Assumption

- The Naïve Bayes assumption is almost never true

- Still Naïve Bayes often performs surprisingly well even when its assumptions do not hold

Numerical Stability

- Naïve bayes algorithm needs to work with very small numbers
  - Imagine computing the probability of 2000 independent coin flips
  - MATLAB thinks that $(.5)^{2000} = 0$

Underflow Prevention

- Multiplying lots of probabilities ➔ floating-point underflow

- Recall: log(xy) = log(x) + log(y)

- Better to sum logs of probabilities rather than multiplying probabilities

$$P(X_1, \ldots, X_n | Y) = \prod_{i=1}^{n} P(X_i | Y)$$

$$\Big\downarrow$$

$$\log P(X_1, ..., X_n | Y) = \sum_{i=1}^{n} \log P(X_i | Y)$$

Underflow Prevention

- Recall

$$P(Y|X_1,\ldots,X_n) = \frac{P(X_1,\ldots,X_n|Y)P(Y)}{P(X_1,\ldots,X_n)}$$

- While predicting the outcome, compare the logarithms

$$\begin{aligned}
\log\left(P(Y|X_1,\ldots,X_n)\right) &= \log\left(\frac{P(X_1,\ldots,X_n|Y)\cdot P(Y)}{P(X_1,\ldots,X_n)}\right) \\
&= \text{constant} + \log\left(\prod_{i=1}^{n} P(X_i|Y)\right) + \log P(Y) \\
&= \text{constant} + \sum_{i=1}^{n} \log P(X_i|Y) + \log P(Y)
\end{aligned}$$

Underflow Prevention

- Class with highest final un-normalized log probability score is still the most probable.

$$\hat{Y} = \arg\max_{c_j \in C} [\sum_{i=1}^{n} \log P(X_i | c_j) + \log P(c_j)]$$

Summary

- We defined a Bayes classifier but saw that it's hard to compute $P(X_1,\ldots,X_n|Y)$
- We then used the Naïve Bayes assumption – that everything is independent given the class label Y
- We talked about using logarithms to avoid underflow
- Naïve Bayes is easy to implement and often works well
- Often a good first thing to try

Classification Techniques

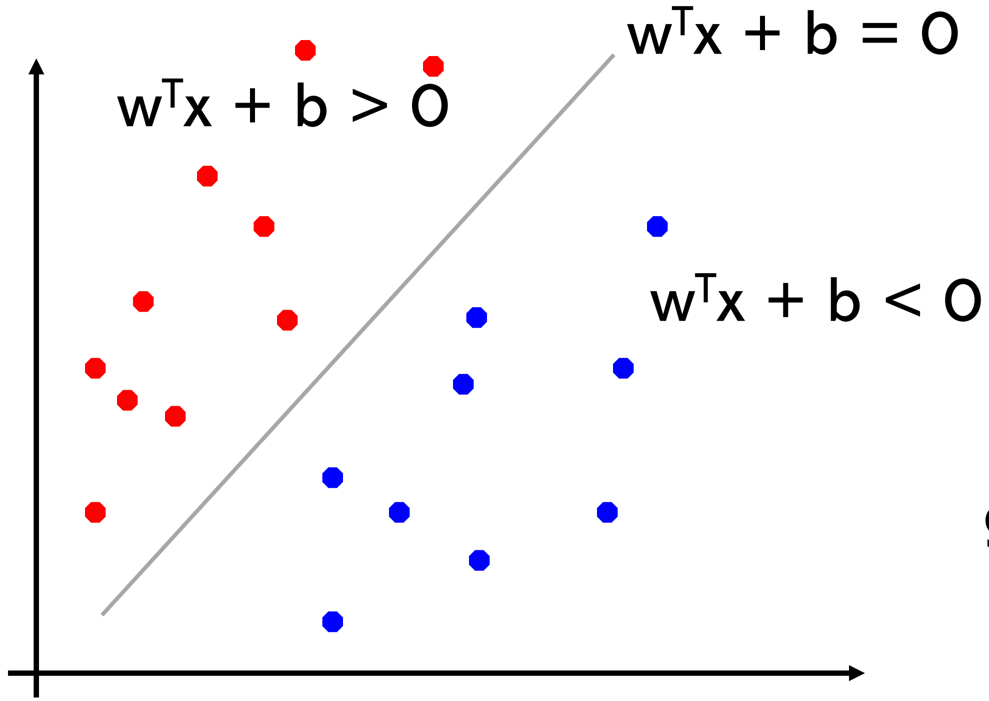
- Base Classifiers
  - Learning with Prototypes (LwP)
  - Nearest-neighbor (kNN)
  - Naïve Bayes (NB)
  - Support Vector Machines (SVM)
  - Decision Tree (DT)
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

Linear Classification

Linear Separators

- Data : $X = [x_1, \dots, x_m]$: a n x m matrix of data points in R^n
$y \in \{-1, 1\}^m$: m-vector of corresponding labels.
- Binary classification can be viewed as the task of separating classes in feature space:



$g(x) = \text{sign}(w^Tx + b)$

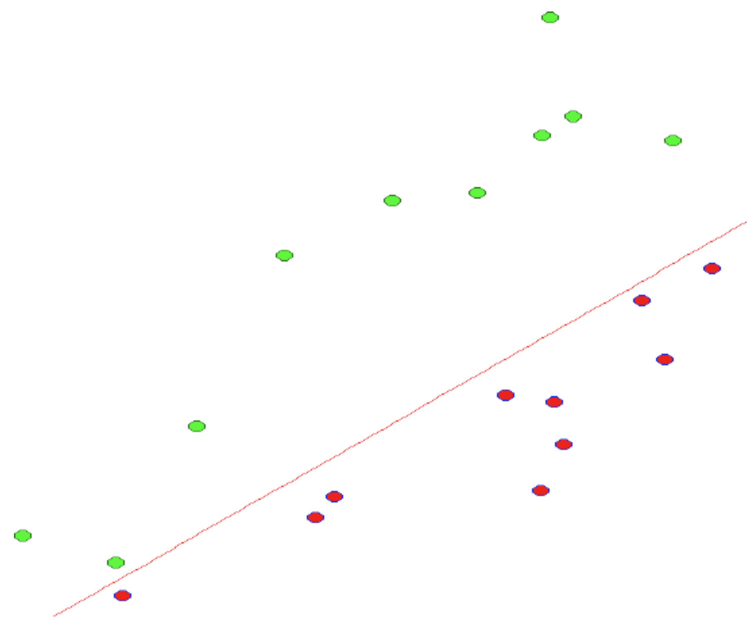
Linear Discriminant Function

- g(x) is a linear function: $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
- A hyper-plane in the feature space
- Main solution idea: formulate the task of finding w, b as a "loss function" minimization problem.

Separability Condition

$$y_i(w^T x_i + b) \geq 0, \quad i = 1, \ldots, m.$$

Ensures that negative (resp. positive) class is contained in half-space $w^T x + b \leq 0$ (resp. $w^T x + b \geq 0$).



0/1 Loss Function Minimization

When data is not strictly separable, we seek to minimize the number of errors

=> the number of indices i for which $y_i(w^T x_i + b) < 0$:

$$\min_{w,b} \sum_{i=1}^{m} L_{0/1}(y_i(w^T x_i + b))$$

where $L_{0/1}$ is the 0/1 loss function

$$L_{0/1}(z) = \begin{cases} 1 & \text{if } z < 0, \\ 0 & \text{otherwise.} \end{cases}$$

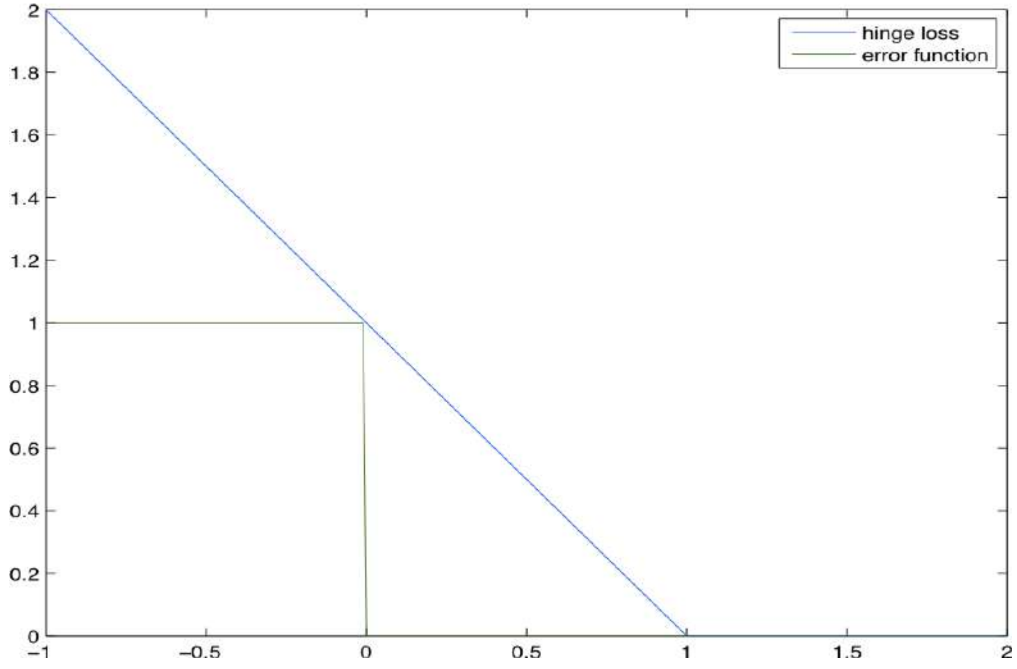
Above problem is very hard to solve exactly

From 0/1 Loss to Hinge Loss

- We approximate the 0/1 loss by the hinge loss:

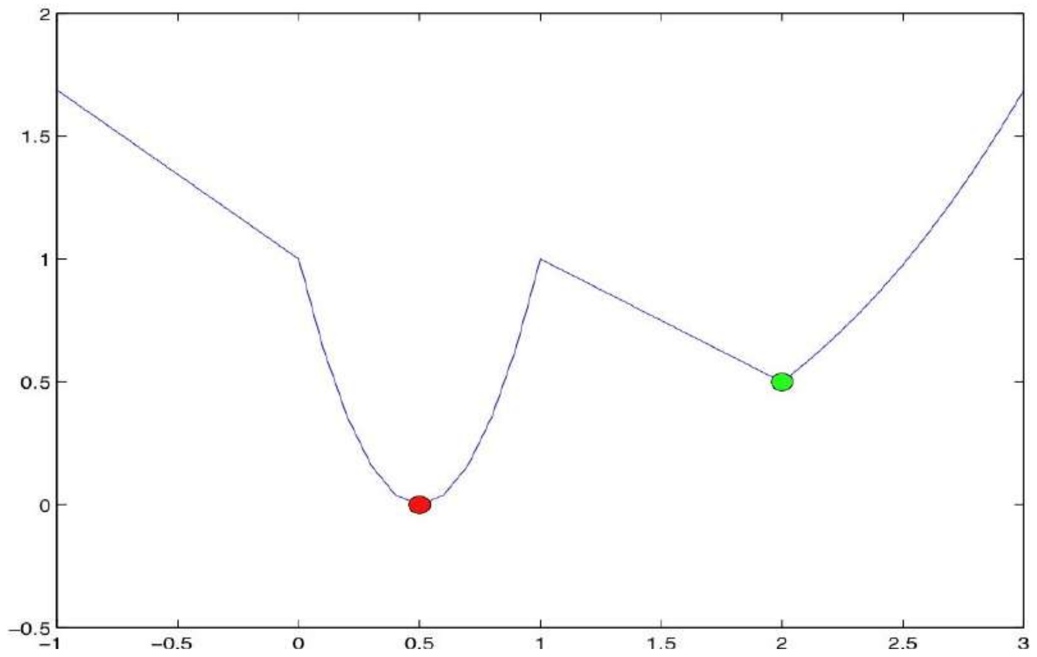
$$H(z) = \max(0, 1 - z)$$

- This function is convex (has a global optima)



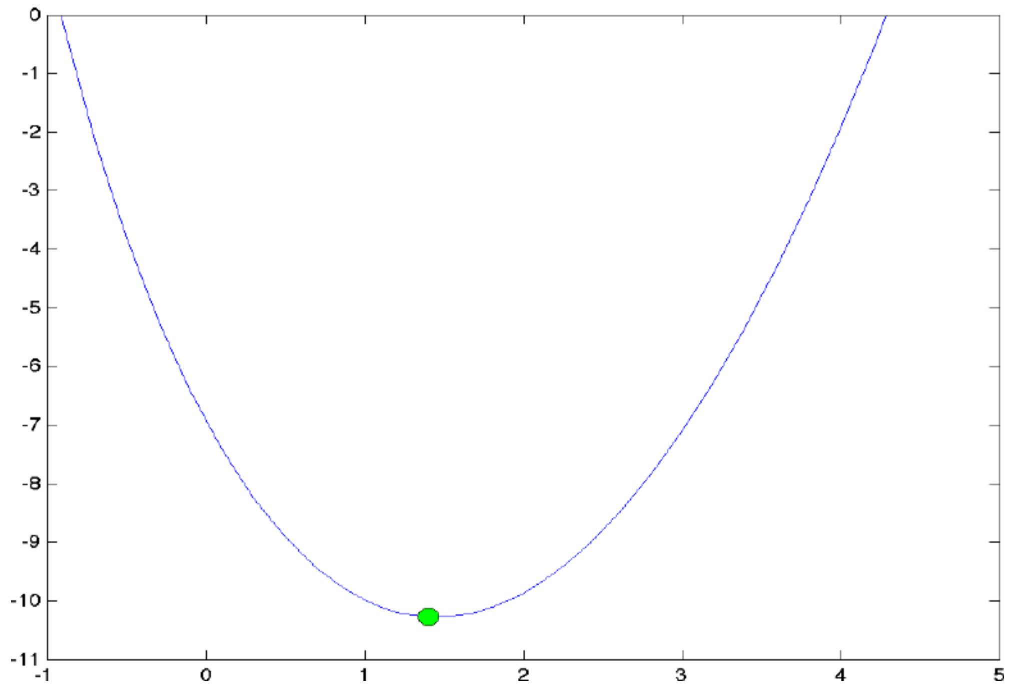
Non-Convex Minimization: Hard

- Optimizing non-convex functions (such as in 0/1 loss minimization) is usually very hard.
- Algorithms may be trapped in so-called “local” minima (in green), which do not correspond to the true minimal value of the objective function (in red).



Convex Minimization: Easy

- Convexity ensures that there are no local minima
- It is generally easy to minimize convex functions numerically via specialized algorithms
- The algorithms can be adapted to cases when the function is convex but not differentiable (such as the hinge loss)



Optimization Function

- Actual optimization problem turns out to be:

$$\min_{w,b} \sum_{i=1}^{m} \max(0, 1 - y_i(w^T x_i + b))$$

- Many commercial solvers available

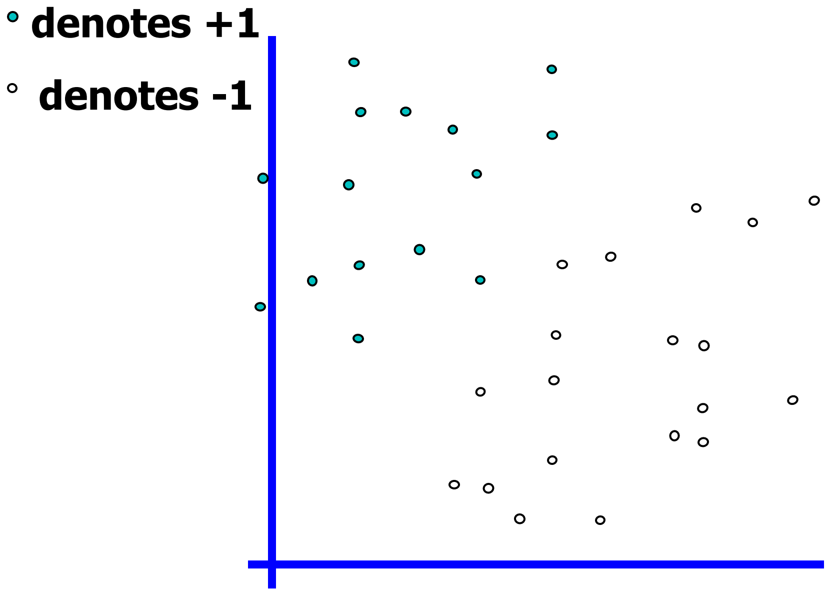
- For moderate-size convex problems, try free python toolbox CVXPY (https://www.cvxpy.org)

Support Vector Machine

Linear Classifiers

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$$f(x,w,b) = \text{sign}(w^T x + b)$$

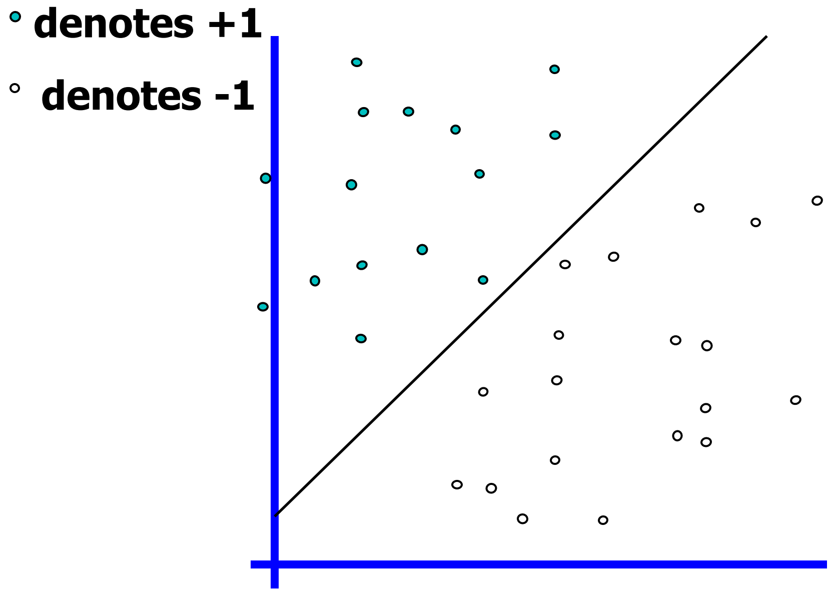


How would you classify this data?

Linear Classifiers

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$f(x,w,b) = \text{sign}(w^T x + b)$

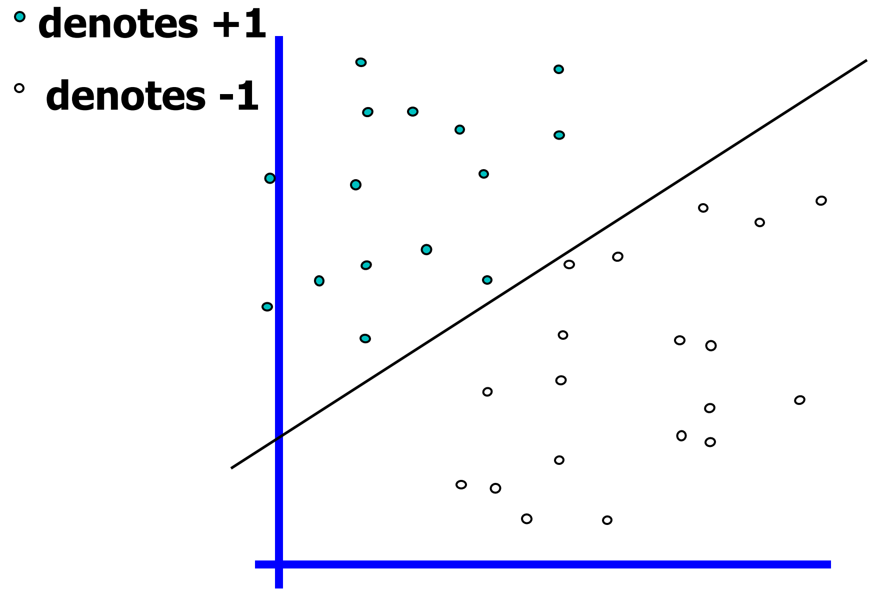


How would you classify this data?

Linear Classifiers

$$\begin{tikzcd} x \arrow[r] & \boxed{f} \arrow[r] & y^{est} \end{tikzcd}$$

$f(x,w,b) = \text{sign}(w^T x + b)$

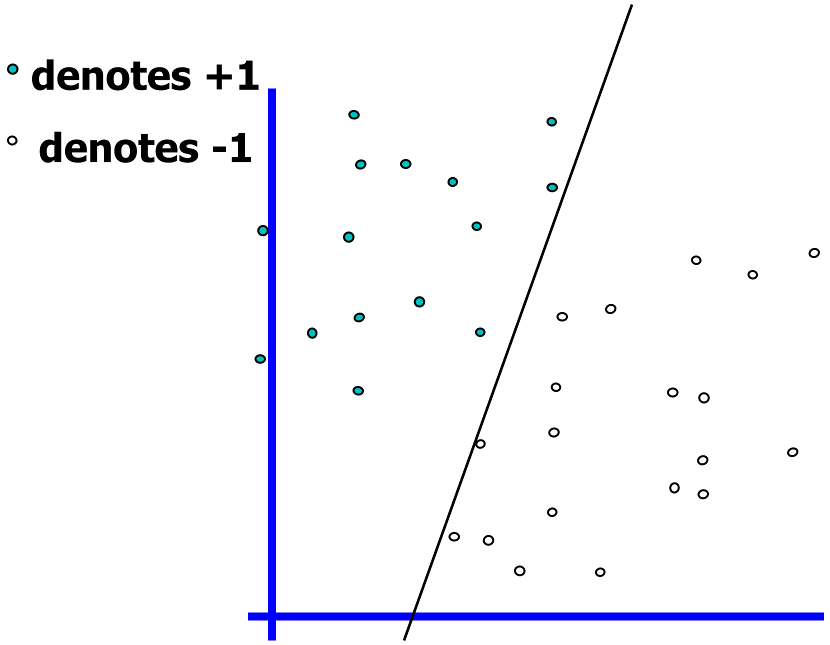


How would you classify this data?

Linear Classifiers

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$$f(x,w,b) = \text{sign}(w^T x + b)$$

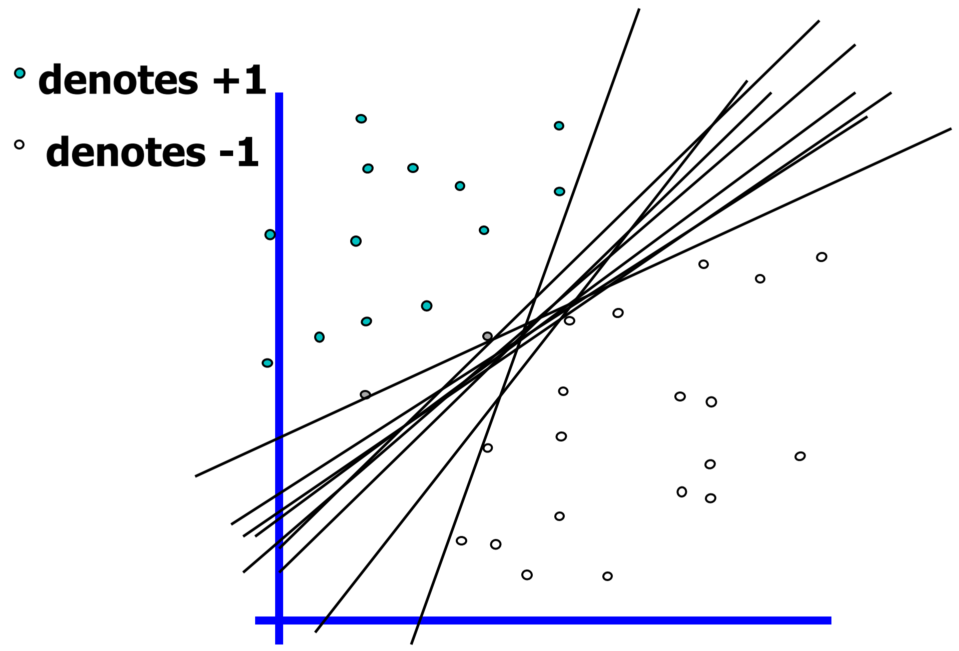


How would you classify this data?

Linear Classifiers

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$f(x,w,b) = \text{sign}(w^T x + b)$



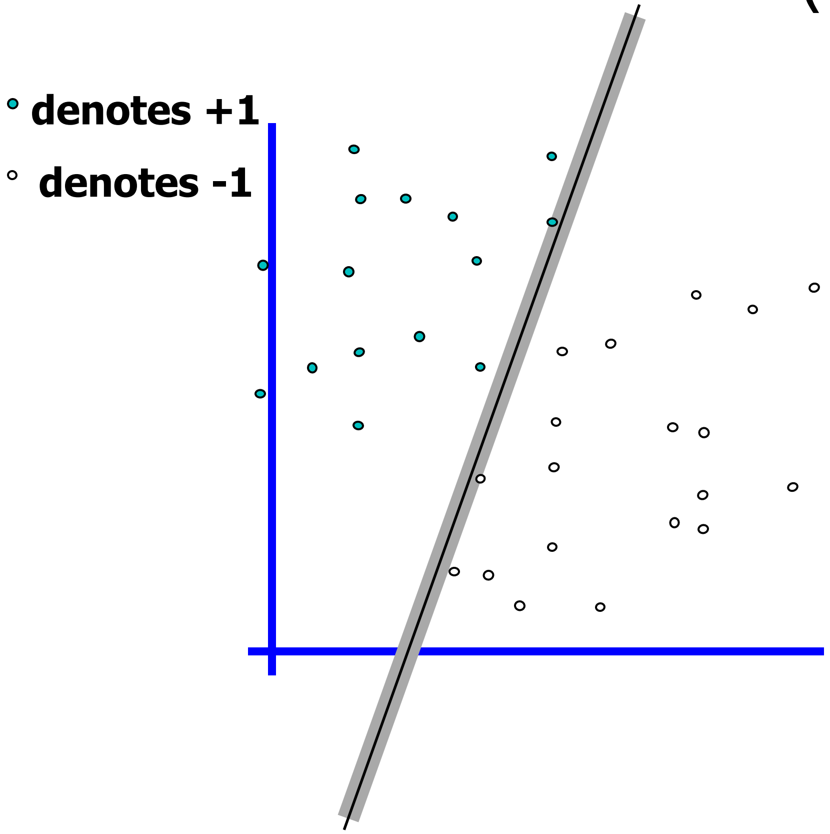
Any of these would be fine..

..but which is the best?

Classifier Margin

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$f(x,w,b) = \text{sign}(w^T x + b)$

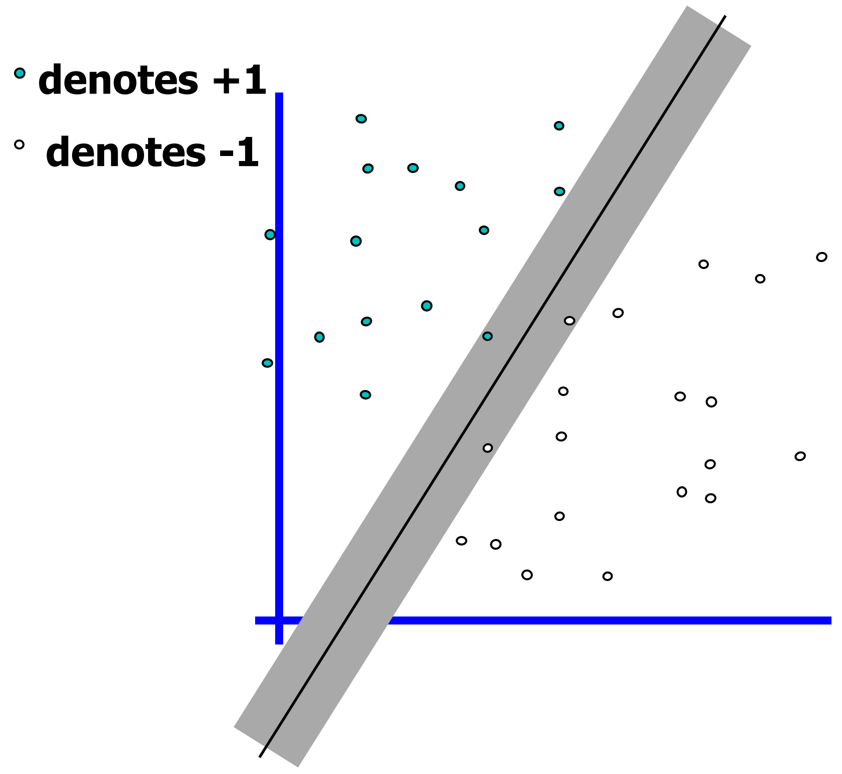


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin

$$x \longrightarrow \boxed{f} \longrightarrow y^{est}$$

$$f(x,w,b) = \text{sign}(w^T x + b)$$



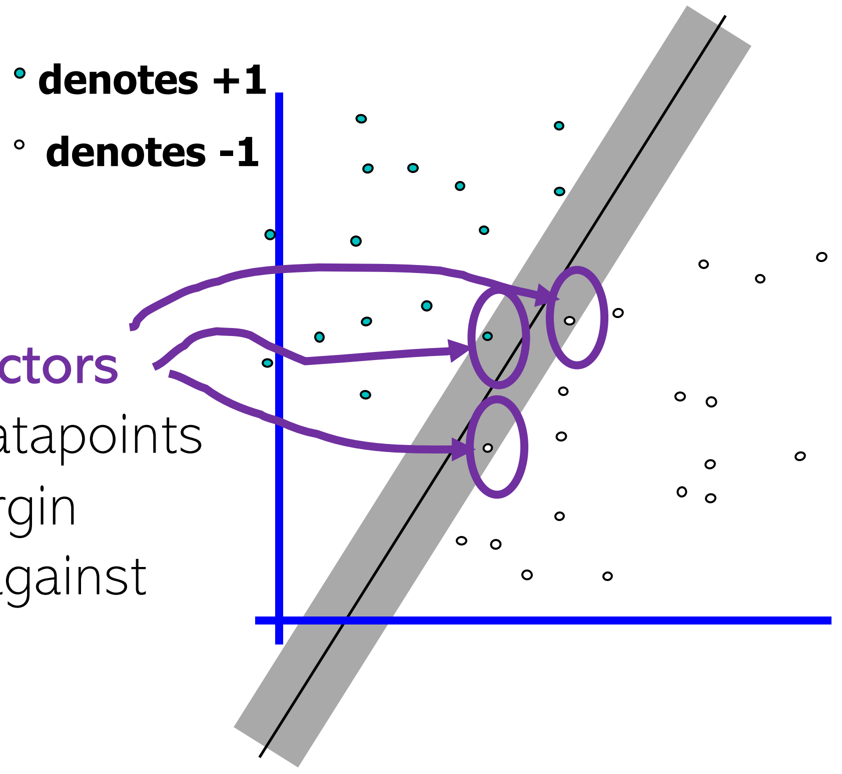
The maximum margin linear classifier is the linear classifier with the maximum margin.

This is the simplest kind of SVM (called Linear SVM)

Maximum Margin

$$\begin{tikzcd} x \arrow[r] & \boxed{f} \arrow[r] & y^{est} \end{tikzcd}$$

$$f(x,w,b) = \text{sign}(w^T x + b)$$

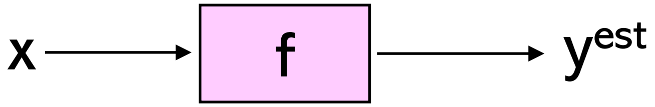


Support Vectors are those datapoints that the margin pushes up against

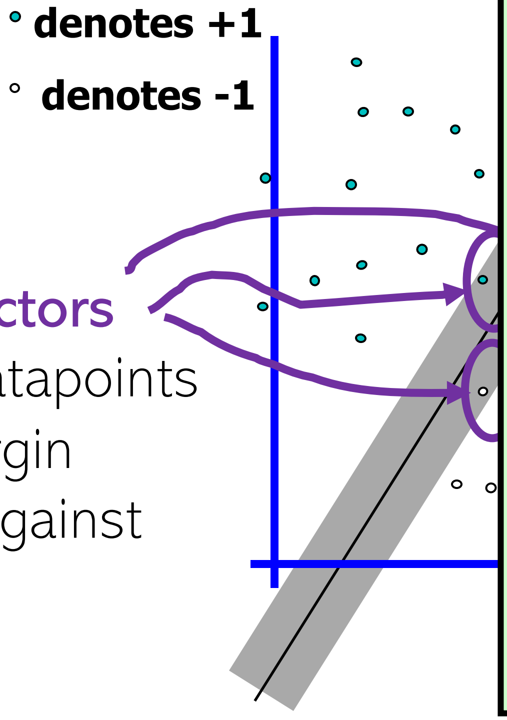
The maximum margin linear classifier is the linear classifier with the maximum margin.

This is the simplest kind of SVM (called Linear SVM)

Maximum Margin



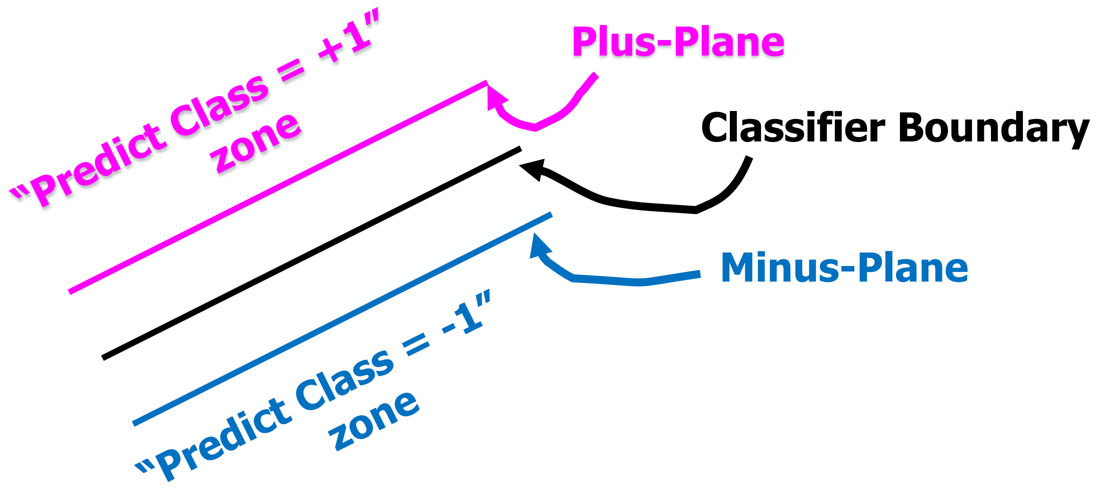
$f(x,w,b) = \text{sign}(w^T x + b)$



Support Vectors are those datapoints that the margin pushes up against

1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
3. The model is immune to removal of any non-support-vector datapoints.
4. Empirically it works very well.

Specifying a Line and Margin



- How do we represent this mathematically?

Some Background

- The length of a vector **w** is called its **norm**, which is written as ||**w**|| and calculated as $\sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$

- Note that ||**w**||^2 = w^{T}w
- Direction vector/unit vector of w is $\frac{w}{||w||}$

- Dot product of two vectors x and y is a scalar

$$x \cdot y = \sum_{i=1}^{n} x_i y_i = x^T y$$

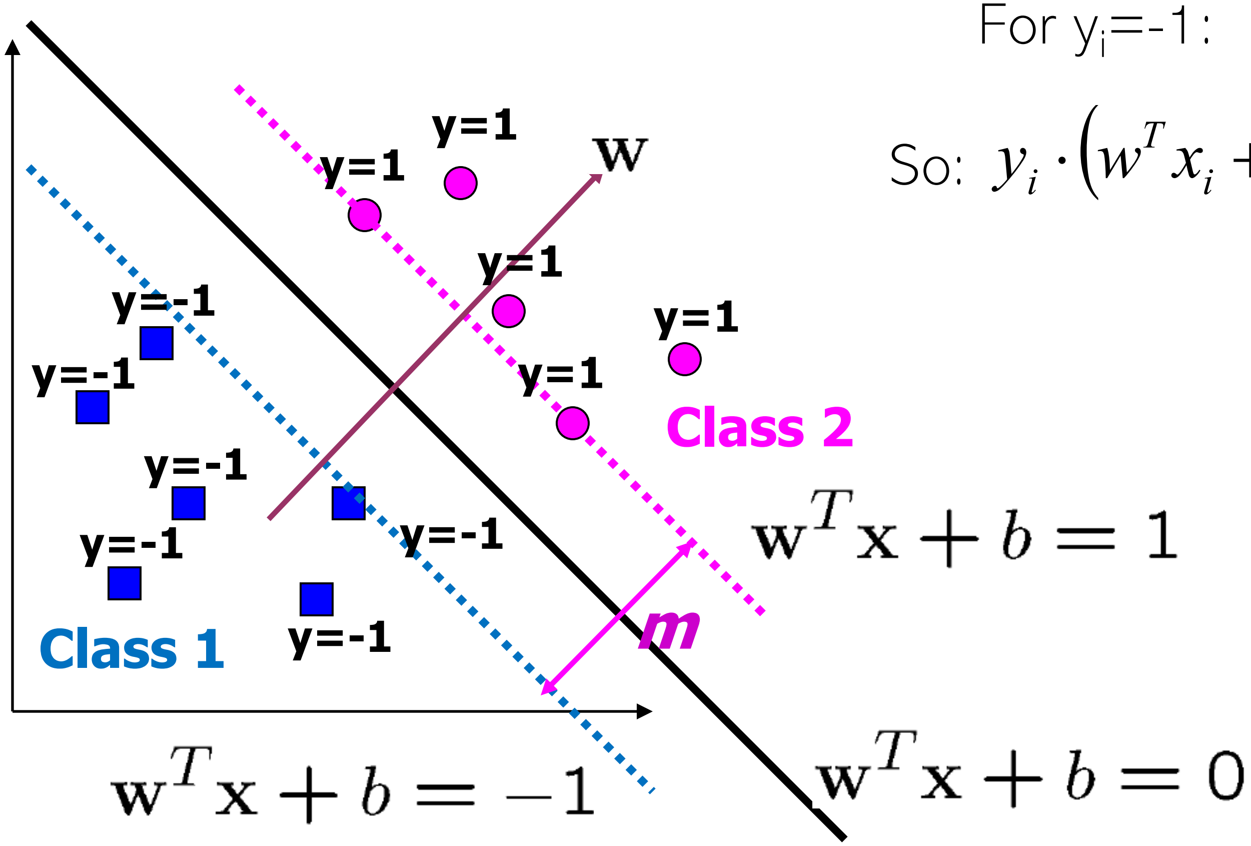
Finding the Decision Boundary

- $\{x_1, \ldots, x_n\}$ is our data/feature set and $y_i \in \{1,-1\}$ is the class label of x_i

For $y_i=1$: $w^T x_i + b \geq 1$

For $y_i=-1$: $w^T x_i + b \leq -1$

So: $y_i \cdot \left(w^T x_i + b\right) \geq 1, \forall (x_i, y_i)$

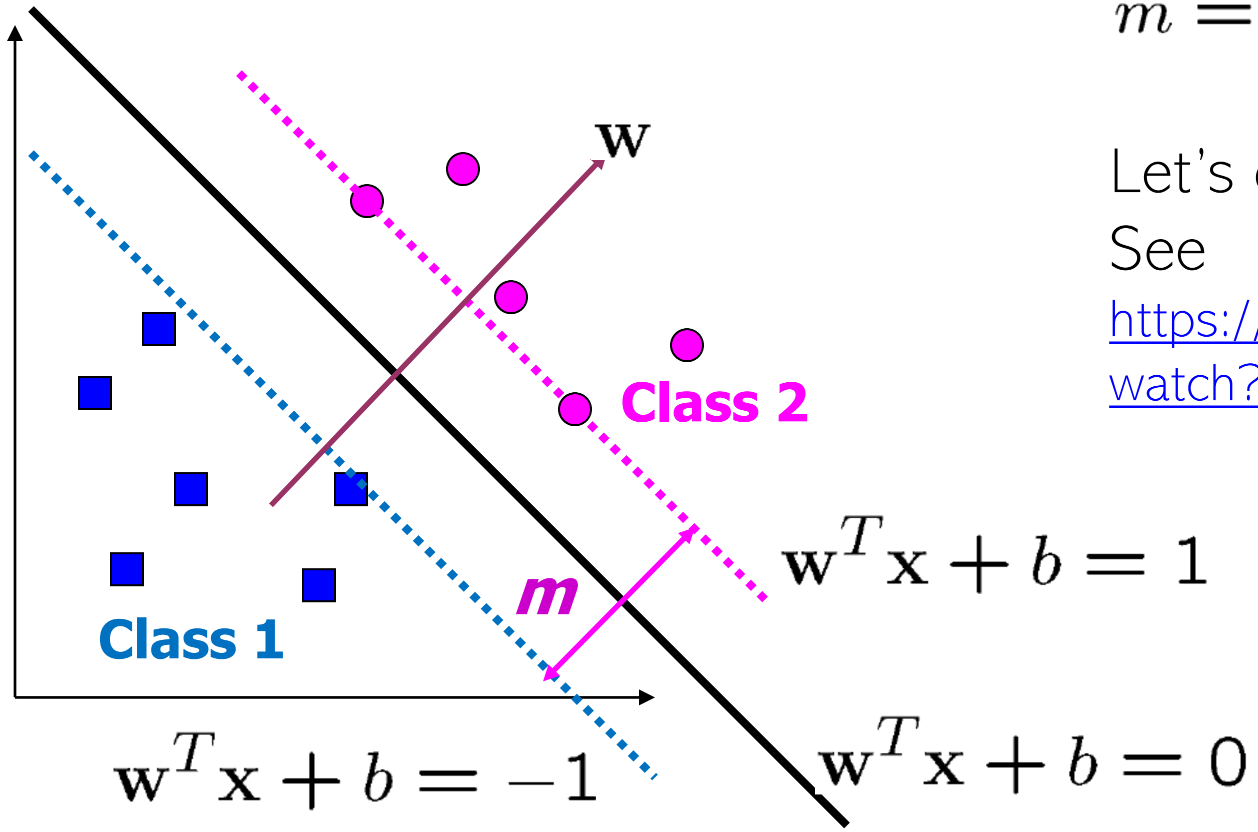

y=1
y=1
w
y=1
y=1
y=1
y=-1
y=-1
Class 2
y=-1
y=-1
y=-1
m
y=-1
Class 1
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}^T\mathbf{x} + b = 0$

m = Margin width

The vector W is perpendicular to the decision boundary/ hyperplane. Why?

Large Margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*



$\mathbf{w}$
Class 2
Class 1
m
$\mathbf{w}^T\mathbf{x}+b=1$
$\mathbf{w}^T\mathbf{x}+b=0$
$\mathbf{w}^T\mathbf{x}+b=-1$

$$m = \frac{2}{||\mathbf{w}||}$$

Let’s derive!

See https://www.youtube.com/watch?v=_PwhiWxHK8o

Finding the Decision Boundary

- Maximizing m is equivalent to minimizing 1/m which is further equivalent to

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

- The decision boundary should classify all points correctly $\Rightarrow$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- The decision boundary can be found by solving the following optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

- This is a constrained optimization problem. Solving it requires to use Lagrange multipliers.

Finding the Decision Boundary

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0 \qquad \text{for } i = 1, \ldots, n$$

The Lagrangian is

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right)$$

- $\alpha_i \geq 0$
- $||w||^2 = w^T w$

Gradient with respect to w and b

- Setting the gradient of $\mathcal{L}$ w.r.t. **w** and b to zero, we have

$$\mathcal{L} = \frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(w^T x_i + b\right)\right) =$$

$$= \frac{1}{2} \sum_{k=1}^{m} w^k w^k + \sum_{i=1}^{n} \alpha_i \left(1 - y_i \left(\sum_{k=1}^{m} w^k x_i^{\ k} + b\right)\right)$$

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w^k} = 0, \forall k & \mathbf{w} + \sum_{i=1}^{n} \alpha_i (-y_i) \mathbf{x}_i = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ \dfrac{\partial \mathcal{L}}{\partial b} = 0 & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases}$$

Note

- Partial derivative w.r.t. a component of a vector only affects the corresponding component of $\mathcal{L}$
- $w^T w = [w_1^{\ 2}, w_2^{\ 2}, \ldots, w_k^{\ 2}, \ldots, w_n^{\ 2}]$

The Dual Problem

- If we substitute $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ to $\mathcal{L}$, we have

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \left(1 - y_i\Big(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b\Big)\right) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b \sum_{i=1}^{n} \alpha_i y_i \\
&= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i
\end{aligned}$$

Since $\sum_{i=1}^{n} \alpha_i y_i = 0$

- This is a function of α_i only

The Dual Problem

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The original problem is known as the primal problem
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - Many approaches have been proposed (Loqo, cplex, etc)

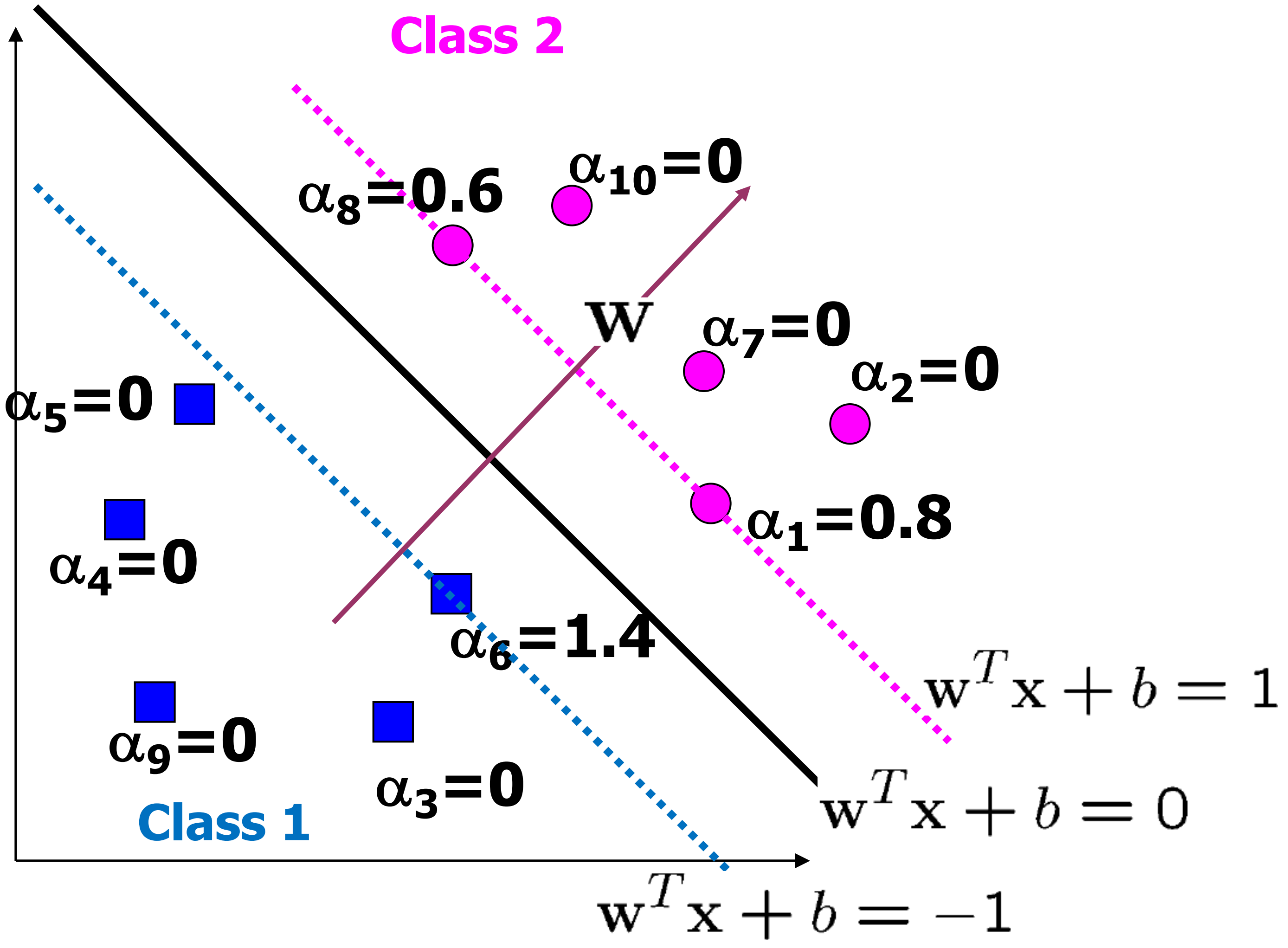
- w can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

Characteristics of the Solution

- Many of the α_i are zero
  - w is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression
- x_i with non-zero α_i are the support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j ($j=1, ..., s$) be the indices of the s support vectors.
  - We can write w as

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

A Geometrical Interpretation



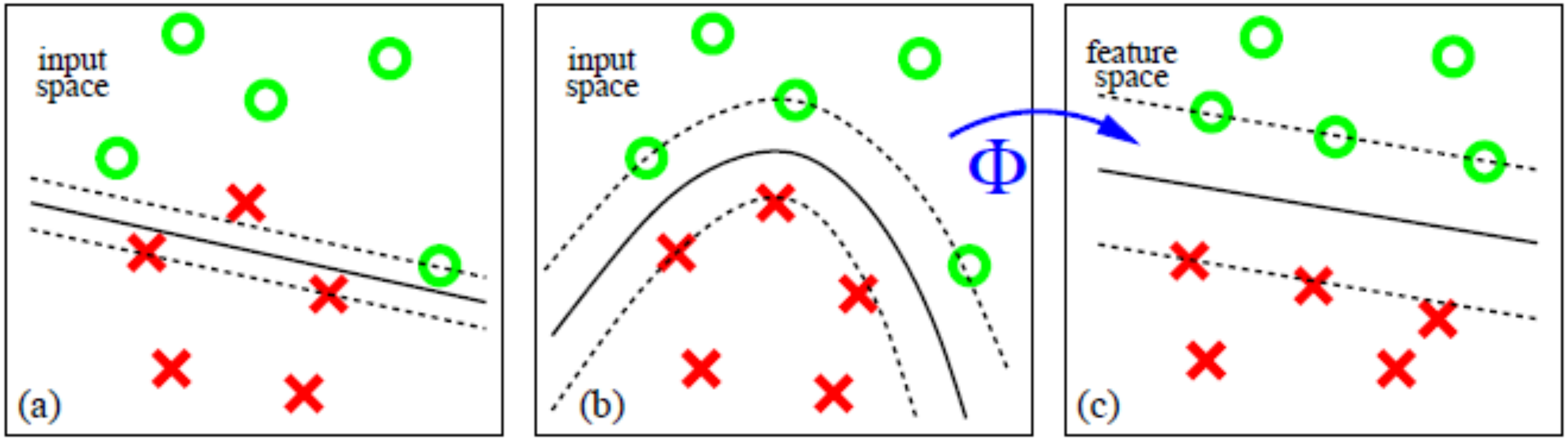
Characteristics of the Solution

- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise
  - Note: **w** need not be computed explicitly

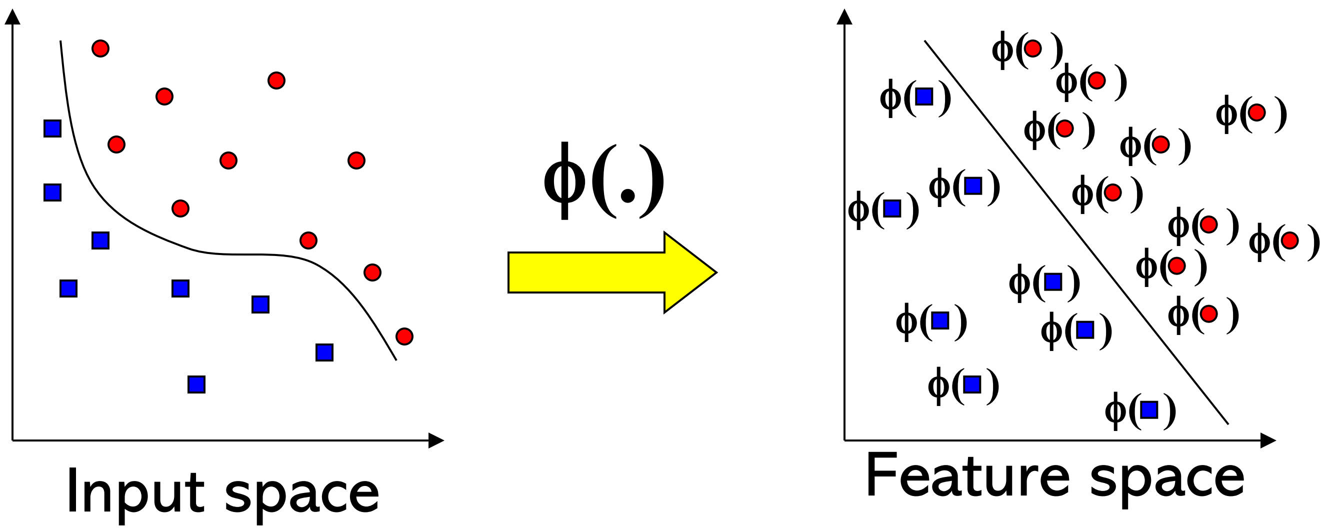
Extension to Non-linear Decision Boundary

- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space the point $\mathbf{x}_i$ are located
  - Feature space: the space of $\phi(\mathbf{x}_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Classification can become easier with a proper transformation

Find a feature space



Transforming the Data



Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

The Kernel Trick

- Recall the SVM optimization problem

$$\text{max.}\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i^T \mathbf{x}_j}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- The data points only appear as inner product
- If we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Define the kernel function K by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

An Example for ϕ(.) and K(.,.)

- Suppose ϕ(.) is given as follows

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\left\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right)\right\rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out ϕ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out ϕ(.) explicitly is known as the kernel trick

Kernels

- Given a mapping: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$

a kernel is represented as the inner product

- A kernel must satisfy the Mercer's condition:

$$K(\mathbf{x}, \mathbf{y}) \rightarrow \sum_i \varphi_i(\mathbf{x}) \varphi_i(\mathbf{y})$$

Modification due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original

$$\text{max.}\ \ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to} \qquad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

With kernel function

$$\text{max.}\ \ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \qquad \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

Modification due to Kernel Function

- For testing, the new data **z** is classified as class 1 if $f \geq 0$, and as class 2 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

With kernel function

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

More on Kernel Functions

- Since the training of SVM only requires the value of $K(\mathbf{x}_i, \mathbf{x}_j)$, there is no restriction of the form of $\mathbf{x}_i$ and $\mathbf{x}_j$
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is just a similarity measure comparing $\mathbf{x}_i$ and $\mathbf{x}_j$
- For a test object $\mathbf{z}$, the discriminant function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

$\mathcal{S}$: the set of support vectors

Examples of Kernel Functions

- Linear kernel

$$K(x, y) = x^T y$$

- Polynomial kernel up to degree *d*

$$K(x, y) = (x^T y + 1)^d$$

- Radial basis function kernel with width σ

$$K(x, y) = \exp(\frac{-||x - y||^2}{2\sigma^2})$$

Doing Multi-class Classification

- SVMs can only handle two-class outputs

- To extend to N class output, learn N SVMs
  - SVM 1 learns “Output==1” vs “Output != 1”
  - SVM 2 learns “Output==2” vs “Output != 2”
  - :
  - SVM N learns “Output==N” vs “Output != N”

- Also called One-vs-All (OvA) strategy

Summary

- Support Vector Machines work very well in practice
- Kernel functions enable SVMs to deal with linearly non separable cases
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner – cross-validation!

Classification Techniques

- Base Classifiers
  - Naïve Bayes
  - Nearest-neighbor
  - Support Vector Machines
  - Decision Tree
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

Classification: Definition

- Given a collection of records (training set)
  - Each record is characterized by a tuple (x,y), where x is the attribute set and y is the class label
    - x: attribute, predictor, independent variable, input
    - y: class, response, dependent variable, output

- Task:
  - Learn a model that maps each attribute set x into one of the predefined class labels y

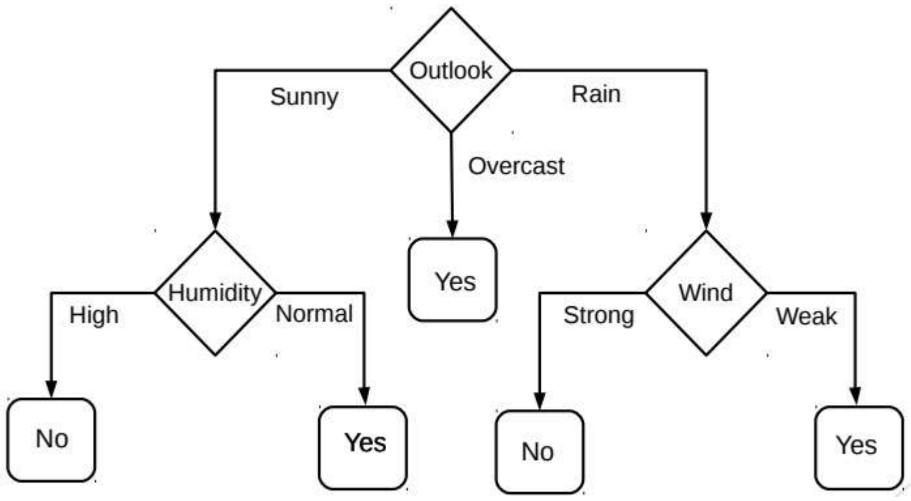
A Classification Problem

- Deciding whether to play Tennis on a particular day
- Each input day has 4 categorical features: Outlook, Temperature, Humidity and Wind
- A binary classification problem (play vs no-play)

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

Decision Trees

- A Decision Tree (DT) defines a hierarchy of rules to make a prediction

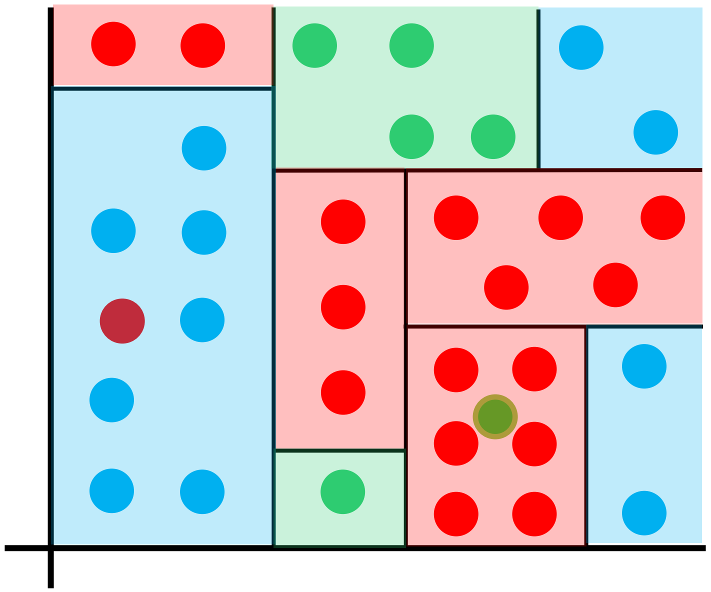


- Root and internal nodes test rules; leaf nodes make predictions
- Decision Tree Learning is about learning such a tree from labelled data

Learning Decision Trees with Supervision

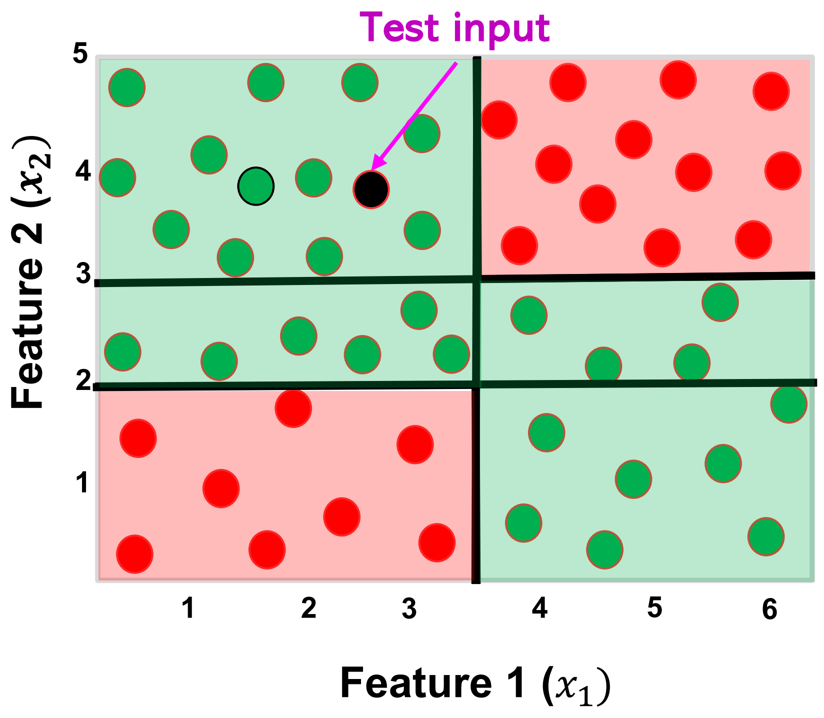
- The basic idea is simple
- Recursively partition the training data into homogeneous regions

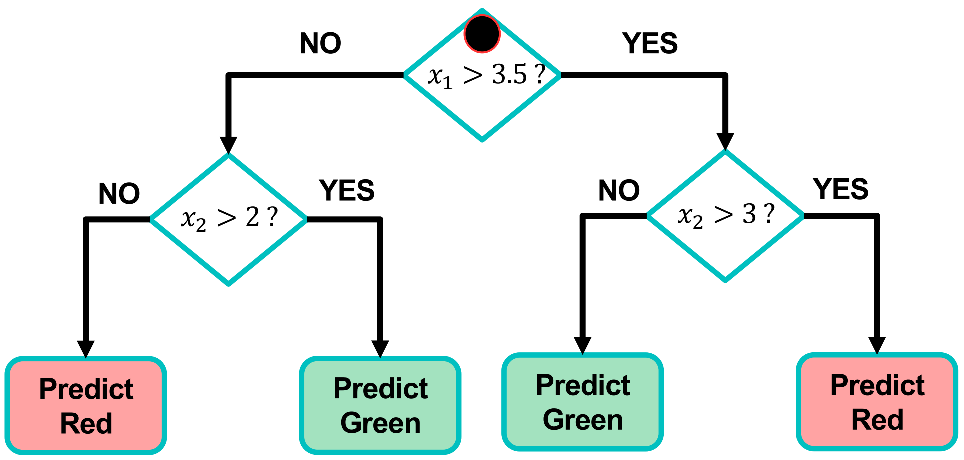
A homogeneous region will have all (or a majority of) training inputs with same labels



- Within each group, fit a simple supervised learner (e.g., predict the majority label)

Decision Trees for Classification





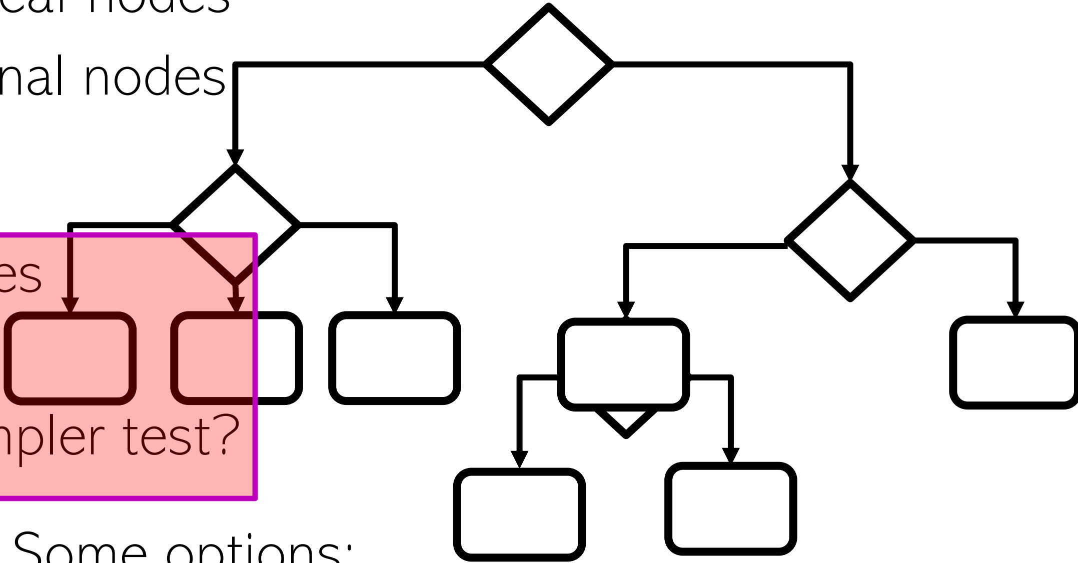
- Root node contains all training inputs
- Each leaf node receives a subset of training inputs

Decision Tree is very efficient at test time:

- To predict the label of a test point, kNN will compare distances from all training inputs
- DT predicts the label by doing just 2 feature-value comparisons!

Decision Trees: Some Considerations

- What should be the size/shape of the DT?
  - Number of internal and leaf nodes
  - Branching factor of internal nodes
  - Depth of the tree
- Split criterion at internal nodes
  - Use another classifier?
  - Or maybe by doing a simpler test?

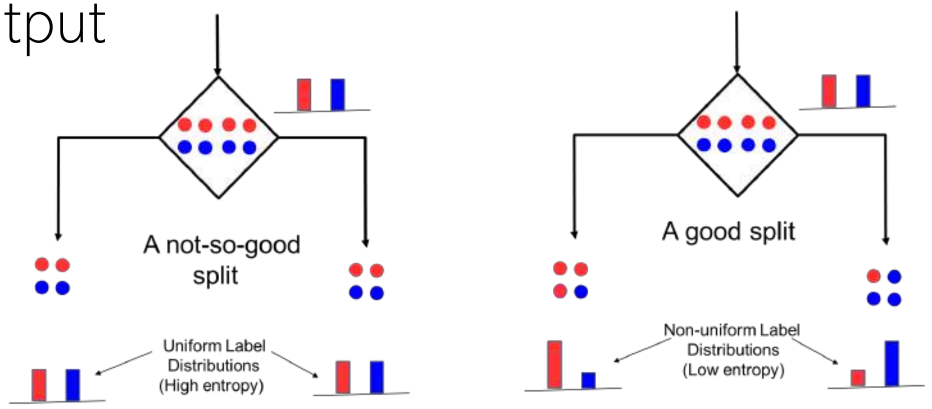


- What to do at the leaf node? Some options:
  - Make a constant prediction for each test input reaching there
  - Use a nearest neighbor based prediction using training inputs at that leaf node
  - Train and predict using some other sophisticated supervised learner on that node

Usually, cross-validation is used to decide size/shape

How to Split at Internal Nodes?

- Each internal node receives a subset of all the training inputs
- Regardless of the criterion, the split should result in as "pure" groups as possible
  - A pure group means that most of the inputs have the same label/output


A not-so-good split
Uniform Label Distributions (High entropy)
A good split
Non-uniform Label Distributions (Low entropy)

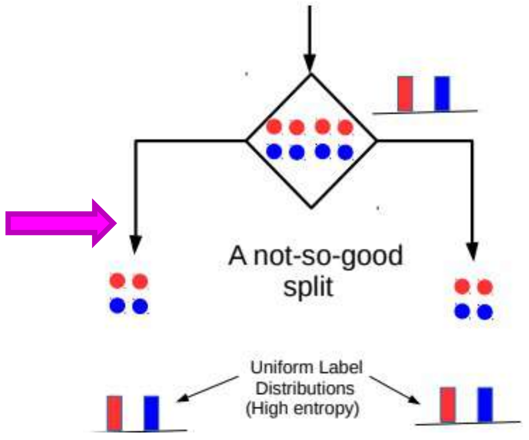
- For classification problems, entropy is a measure of purity
  - Low entropy ⇒ high purity (less uniform label distribution)
  - Splits that give the largest reduction (before split vs after split) in entropy are preferred
  - This reduction in entropy is called "information gain"

Entropy and Information Gain

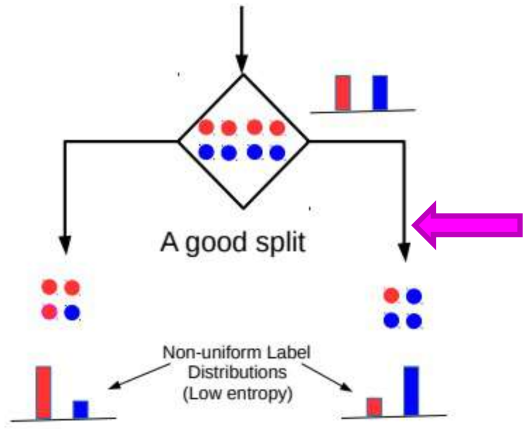
- Assume a set of labelled inputs $\boldsymbol{S}$ from C classes, p_c as fraction of class c inputs
- Entropy of the set $\boldsymbol{S}$ is defined as $\mathrm{H}(\boldsymbol{S}) = -\sum_{c \in C} p_c \log p_c$
- Uniform sets (all classes roughly equally present) have high entropy; skewed sets low
- Suppose a rule splits $\boldsymbol{S}$ into two smaller disjoint sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$
- Reduction in entropy after the split is called information gain

$$IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$$

This split has a low IG (in fact, zero IG)



VS

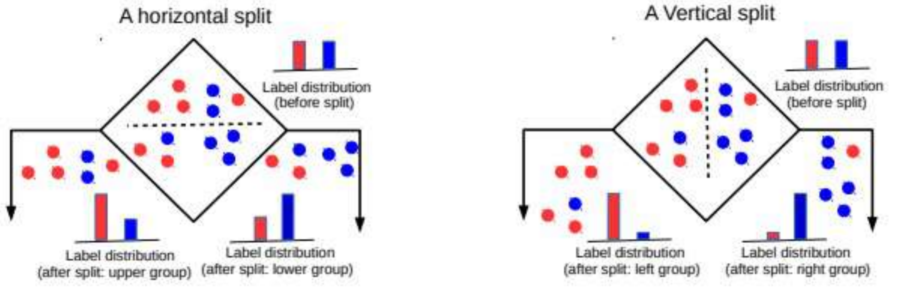


This split has higher IG

Techniques to Split at Internal Nodes

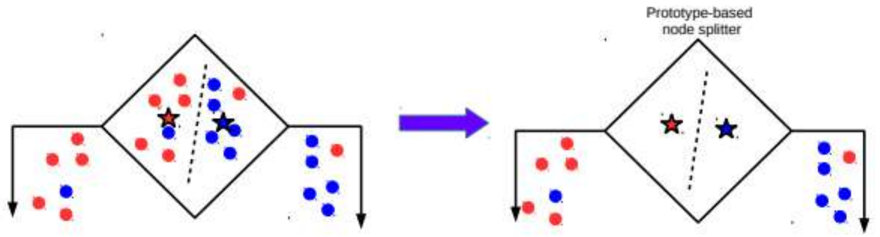
- Each internal node decides which outgoing branch an input should be sent to
- This decision/split can be done using various ways, e.g.,
  - Testing the value of a single feature at a time

With this approach, all features and all possible values of each feature need to be evaluated in selecting the feature to be tested at each internal node


A horizontal split
Label distribution (before split)
Label distribution (after split: upper group)
Label distribution (after split: lower group)
A Vertical split
Label distribution (before split)
Label distribution (after split: left group)
Label distribution (after split: right group)

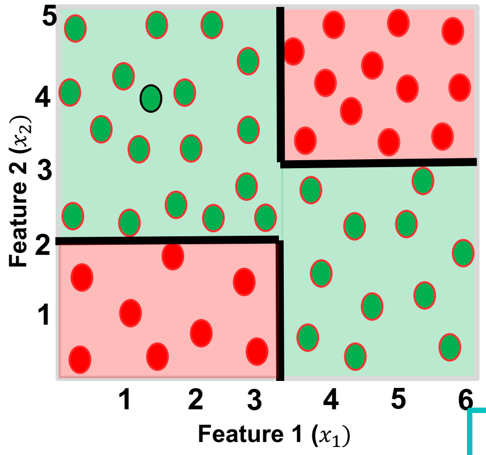
DT methods based on testing a single feature at each internal node are more popular (e.g., ID3, C4.5 algorithms)

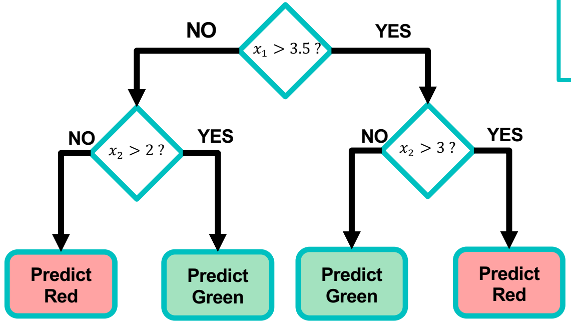
  - Learning a classifier



DT methods based on learning and using a separate classifier at each internal node are less common. But this approach can be very powerful and maybe used in advanced DT methods.

Constructing Decision Trees





Given some training data, what's the "optimal" DT?

How to decide which rules to test for and in what order?

How to assess informativeness of a rule?

In general, constructing DT is an intractable problem (NP-hard)

Often, we can use some "greedy" heuristics to construct a "good" DT

To do so, we use the training data to figure out which rules should be tested at each node

The same rules will be applied on the test inputs to route them along the tree until they reach some leaf node where the prediction is made

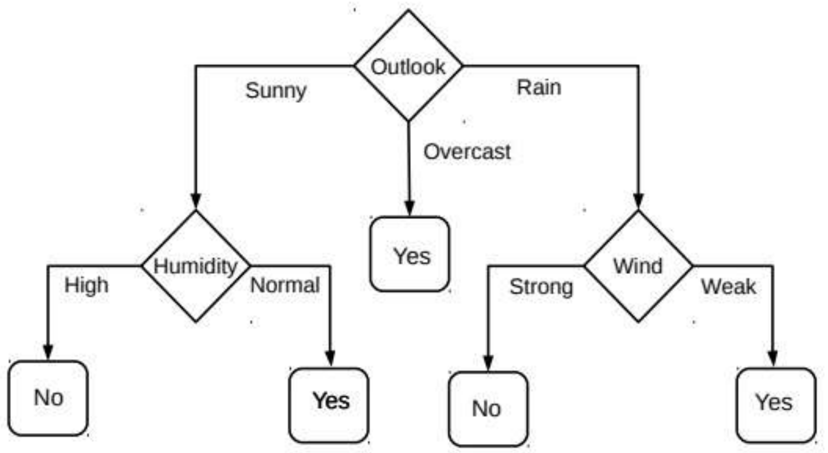
The rules are organized in the DT such that most informative rules are tested first

Informativeness of a rule is of related to the extent of the purity of the split arising due to that rule. More informative rules yield purer splits

Decision Tree Construction: An Example

- Let's consider the playing Tennis example again
- Assume each internal node will test one of the features

day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no



- Question: Why does it make more sense to test the feature "outlook" first?
- Answer: of all the 4 features, it's the most informative
  - It has the highest information gain as the root node

Entropy and Information Gain

- Let's use IG based criterion to construct a DT for the Tennis example
- At root node, let's compute IG of each of the 4 features
- Consider feature "wind". Root contains all examples $S = [9+,5-]$

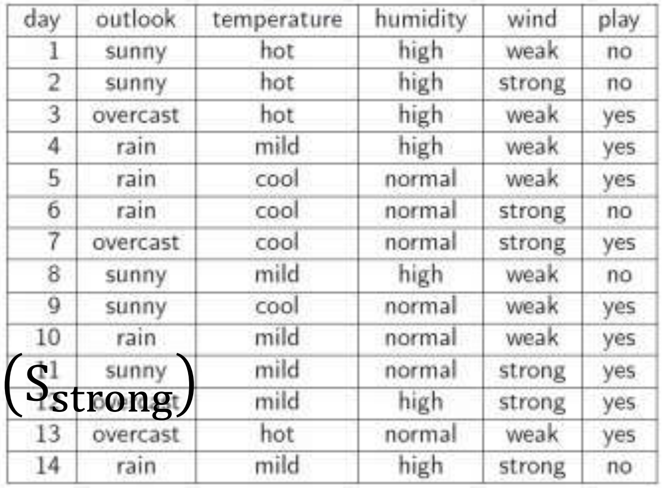
$H(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$
$= 0.94$

$S_{weak} = [6+, 2-] \Rightarrow H(S_{weak}) = 0.811$

$S_{strong} = [3+, 3-] \Rightarrow H(S_{strong}) = 1$

$$\mathrm{IG}(S, \mathrm{wind}) = \mathrm{H}(S) - \frac{|S_{\mathrm{weak}}|}{|S|}\mathrm{H}(S_{\mathrm{weak}}) - \frac{|S_{\mathrm{strong}}|}{|S|}\mathrm{H}(S_{\mathrm{strong}})$$

$= 0.94 - 8/14 * 0.811 - 6/14 * 1 = 0.048$

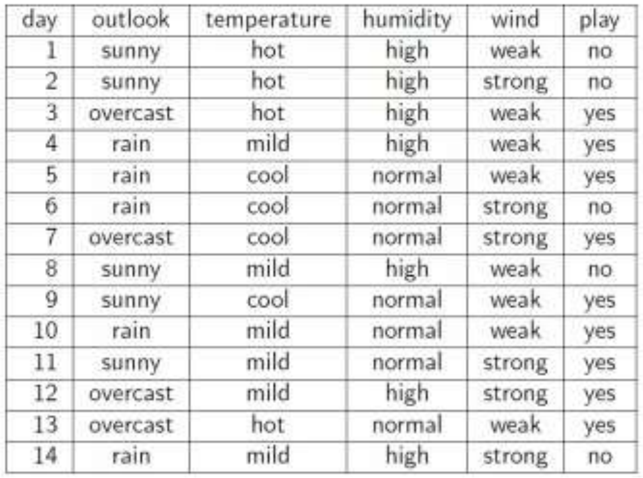


day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

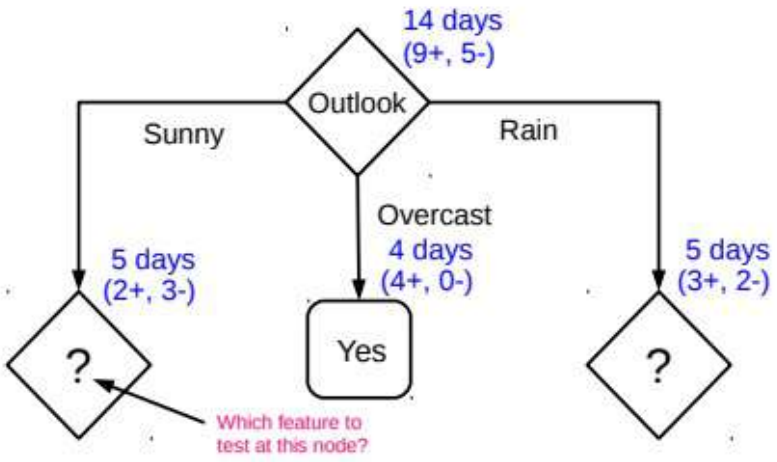
- Likewise, at root: IG(S, outlook)=0.246, IG(S, humidity)=0.151, IG(S,temp)=0.029
- Thus, we choose "outlook" feature to be tested at the root node

Growing the Tree

- How to grow the Decision Tree, i.e., what to do at the next level? Which feature to test next?
- Rule: Iterate - for each child node, select the feature with the highest Information Gain



day	outlook	temperature	humidity	wind	play
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no
7	overcast	cool	normal	strong	yes
8	sunny	mild	high	weak	no
9	sunny	cool	normal	weak	yes
10	rain	mild	normal	weak	yes
11	sunny	mild	normal	strong	yes
12	overcast	mild	high	strong	yes
13	overcast	hot	normal	weak	yes
14	rain	mild	high	strong	no

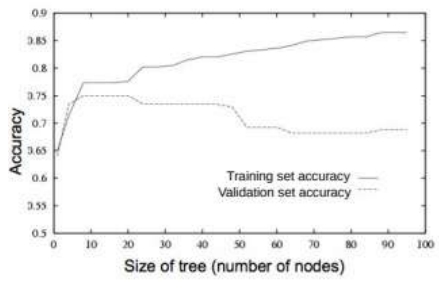


Growing the Tree

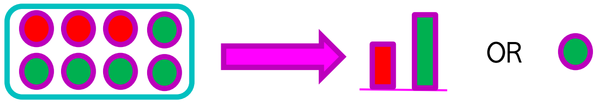
- Proceeding as before, for level 2, left node, we can verify that
    - IG(S,temp) = 0.570, IG(S, humidity) = 0.970, IG(S, wind) = 0.019
- Thus, humidity chosen as the feature to be tested at level 2, left node
- No need to expand the middle node (already "pure" - all "yes" training examples)
- Can also verify that wind has the largest IG for the right node
- If a feature has already been tested along a path earlier, we don't consider it again

When to Stop Growing the Tree?

- Stop expanding a node further (i.e., make it a leaf node) when
  - It consist of all training examples having the same label (the node becomes "pure")
  - We run out of features to test along the path to that node
  - The DT starts to overfit (can be checked by monitoring the validation set accuracy)



- Important: No need to obsess too much for purity
  - It is okay to have a leaf node that is not fully pure, e.g., this
  - At test inputs that reach an impure leaf, can predict probability of belonging to each class (for example, p(red) = 3/8, p(green) = 5/8), or simply predict the majority label



Avoiding Overfitting in DTs

- Desired: a DT that is not too big in size, yet fits the training data reasonably
- An example of a very simple DT is "decision-stump"
  - A decision-stump only tests the value of a single feature (or a simple rule)
  - Not very powerful in itself but often used in large ensembles of decision stumps
- Mainly two approaches to prune a complex Decision Tree
  - Prune while building the tree (stopping early)
  - Prune after building the tree (post-pruning)

Avoiding Overfitting in DTs

- Criteria for judging which nodes could potentially be pruned
    - Use a validation set (separate from the training set)
    - Prune each possible node that doesn't hurt the accuracy on the validation set
    - Greedily remove the node that improves the validation accuracy the most
    - Stop when the validation set accuracy starts worsening

Summary

Some key strengths:

- Simple and easy to interpret
- Easily handle different types of features (real, categorical, etc.)
- Very fast at test time
- Multiple DTs can be combined via ensemble methods: more powerful (e.g., Random Forests)

Some key weaknesses:

- Learning optimal DT is (NP-hard) intractable. Existing algos mostly greedy heuristics
- Can sometimes become very complex unless some pruning is applied

Classification Techniques

- Base Classifiers
  - Naïve Bayes
  - Nearest-neighbor
  - Support Vector Machines
  - Decision Tree
  - Neural Networks, Deep Neural Nets

- Ensemble Classifiers
  - Boosting, Bagging, Random Forests

Multiple Classifiers at Work

- Different Classifiers
  - Conduct classification on a same set of class labels
  - May use different input or have different parameters
  - May produce different output for a certain example

- Learning Different Classifiers
  - Use different training examples
  - Use different features

Multiple Classifiers at Work

- Performance
  - Each of the classifiers is not perfect
  - Complementary
  - Examples which are not correctly classified by one classifier may be correctly classified by the other classifiers

- Potential Improvements?
  - Utilize the complementary property

- Ensemble Learning
  - How to create classifiers with complementary performances
  - How to conduct voting

Strong and Weak Learners

- Strong Learner
  - Take labeled data for training
  - Produce a classifier which can be arbitrarily accurate
- Weak Learner
  - Take labeled data for training
  - Produce a classifier which is more accurate than random guessing

Idea Behind Ensembles

- Learners
  - Strong learners are very difficult to construct
  - Constructing weaker Learners is relatively easy

- Strategy
  - Derive strong learner from weak learner
  - Combine or boost weak classifiers to a strong learner

Ensemble Approaches

- Bagging
  - Bootstrap aggregating
- Boosting
- Random Forests
  - Bagging reborn

The Bagging Algorithm

Given data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

For $m = 1 : M$

- ☐ Obtain bootstrap sample D_m from the training data D
- ☐ Build a model $G_m(\mathbf{x})$ from bootstrap data D_m

The Bagging Model

- Regression

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} G_m(\mathbf{x})$$

- Classification:
  - Vote over classifier outputs $G_1(\mathbf{x}), \dots, G_M(\mathbf{x})$

Bagging Details

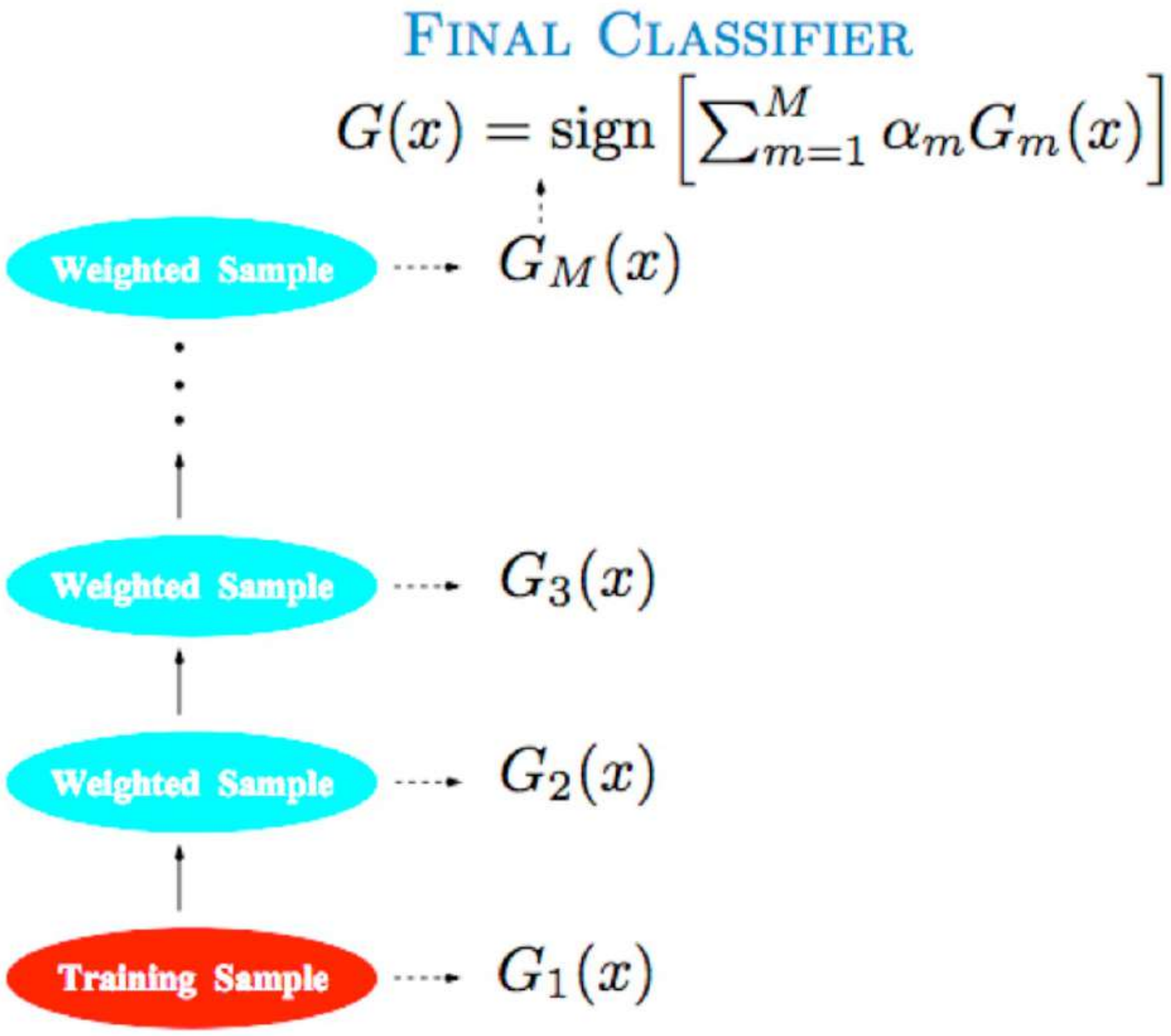
- Bootstrap sample of N instances is obtained by drawing N examples at random, with replacement.

- On average each bootstrap sample has 63% of instances
  - Encourages predictors to have uncorrelated errors

Boosting

- Construct weak classifiers using different data distribution
  - Start with uniform weighting
  - During each step of learning
    - Increase weights of the examples which are not correctly learned by the weak learner
    - Decrease weights of the examples which are correctly learned by the weak learner
- Idea
  - Focus on difficult examples which are not correctly classified in the previous steps

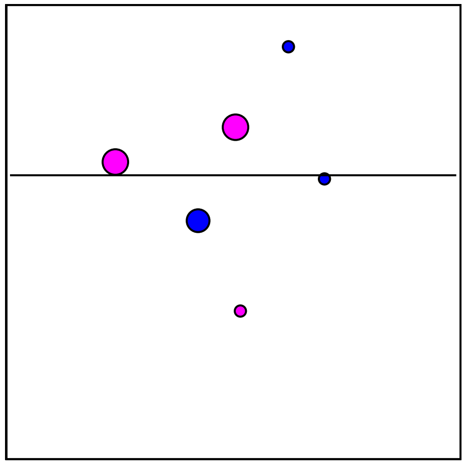
Boosting

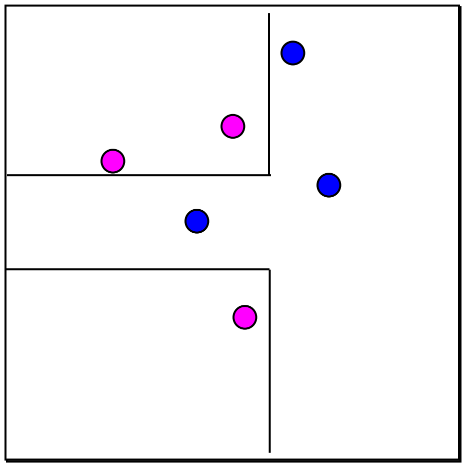

FINAL CLASSIFIER
$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$
Weighted Sample
$G_M(x)$
Weighted Sample
$G_3(x)$
Weighted Sample
$G_2(x)$
Training Sample
$G_1(x)$

Combine Weak Classifiers

- Weighted Voting
  - Construct strong classifier by weighted voting of the weak classifiers
- Idea
  - Better weak classifier gets a larger weight
  - Iteratively add weak classifiers
    - Increase accuracy of the combined classifier through minimization of a cost function

Example


Single Learner


Combined Classifier

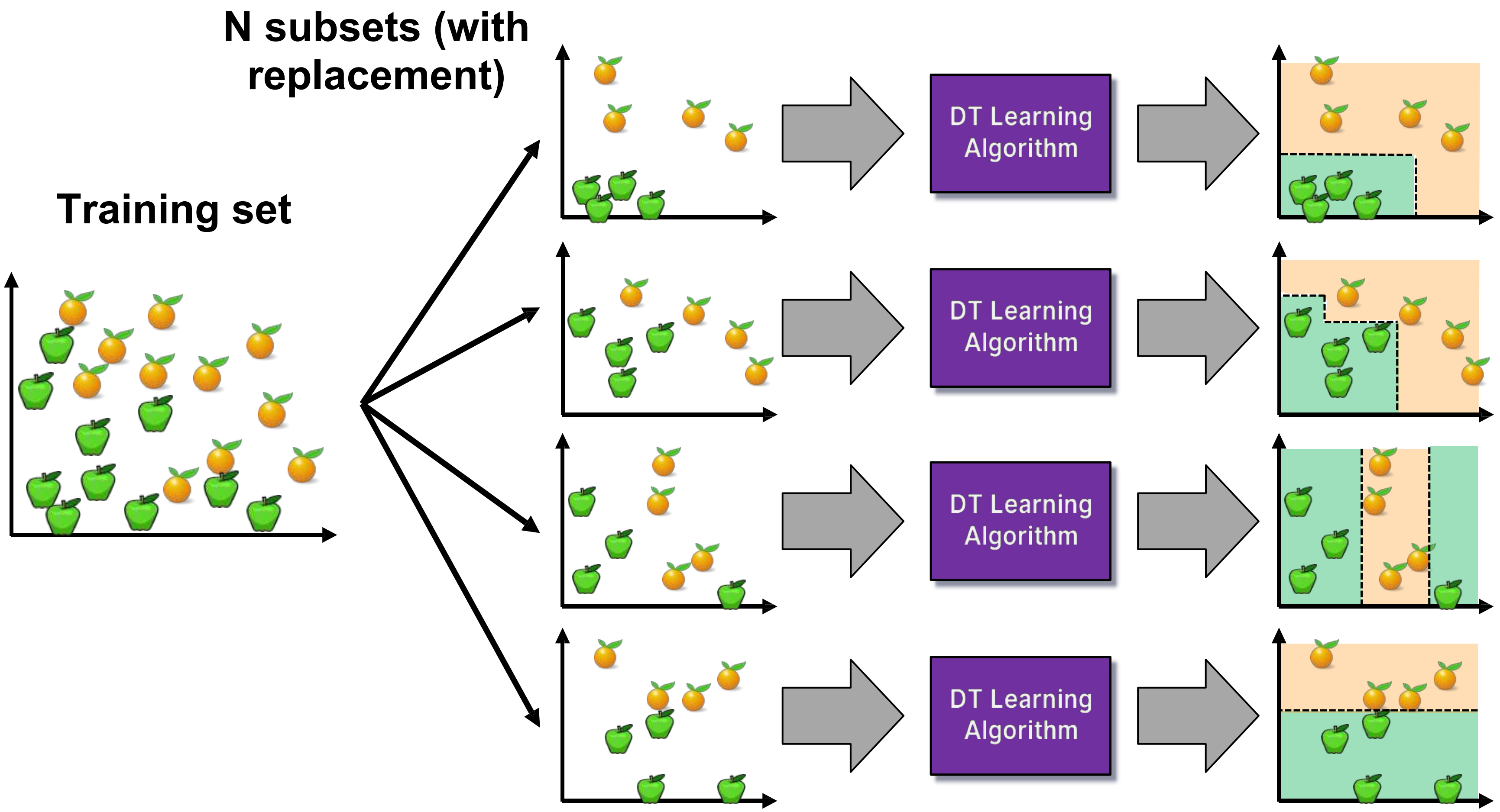
Random Forest

We have a single data set, so how do we obtain slightly different trees?

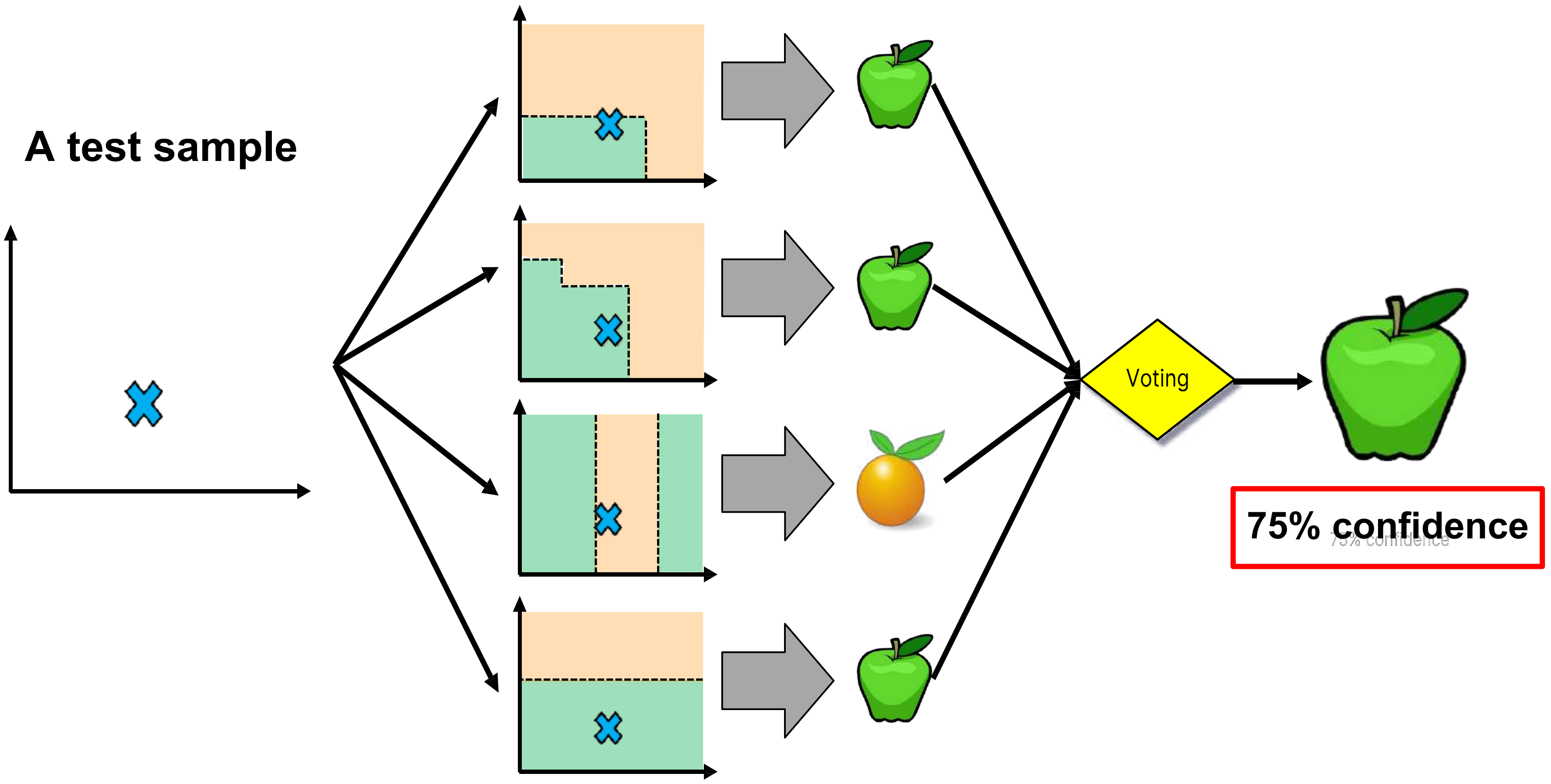
1. Bagging:
   - Take random subsets of data points from the training set to create N smaller data sets
   - Fit a decision tree on each subset

2. Random Subspace Method (aka Feature Bagging):
   - Fit N different decision trees by constraining each one to operate on a random subset of features

Bagging at Training Time



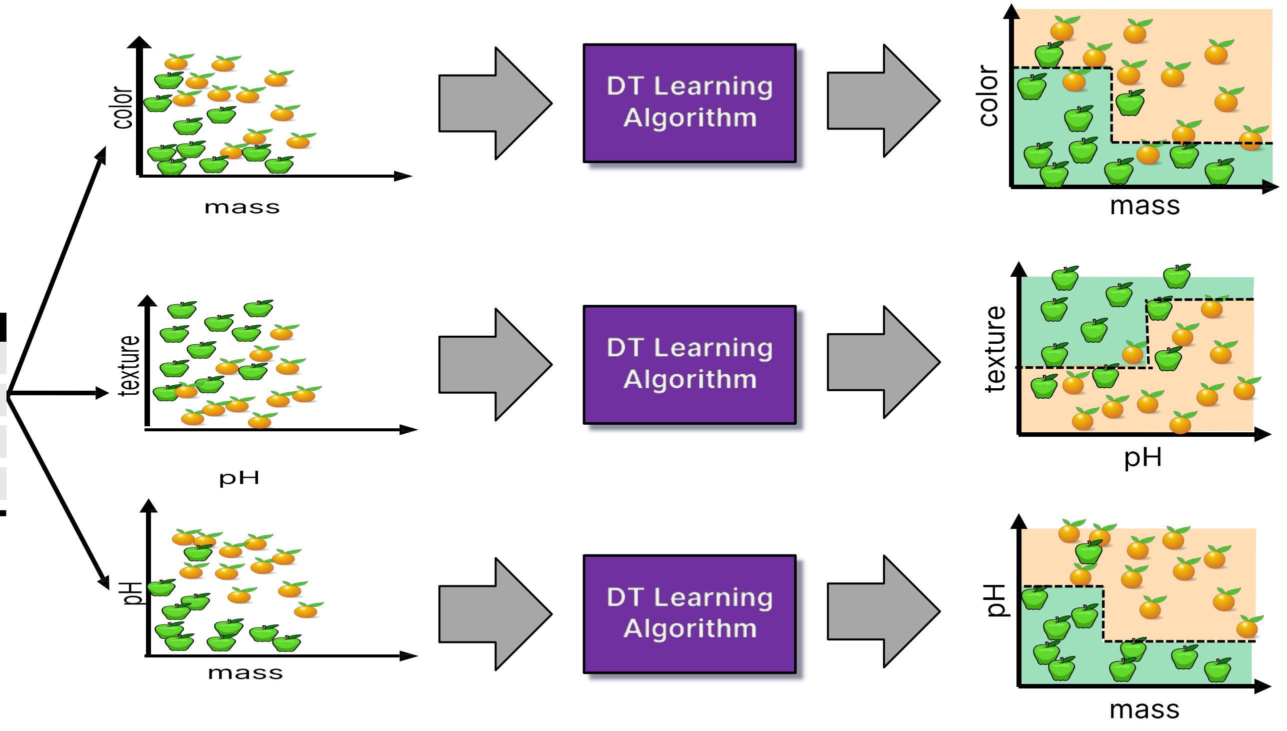
Bagging at Inference Time



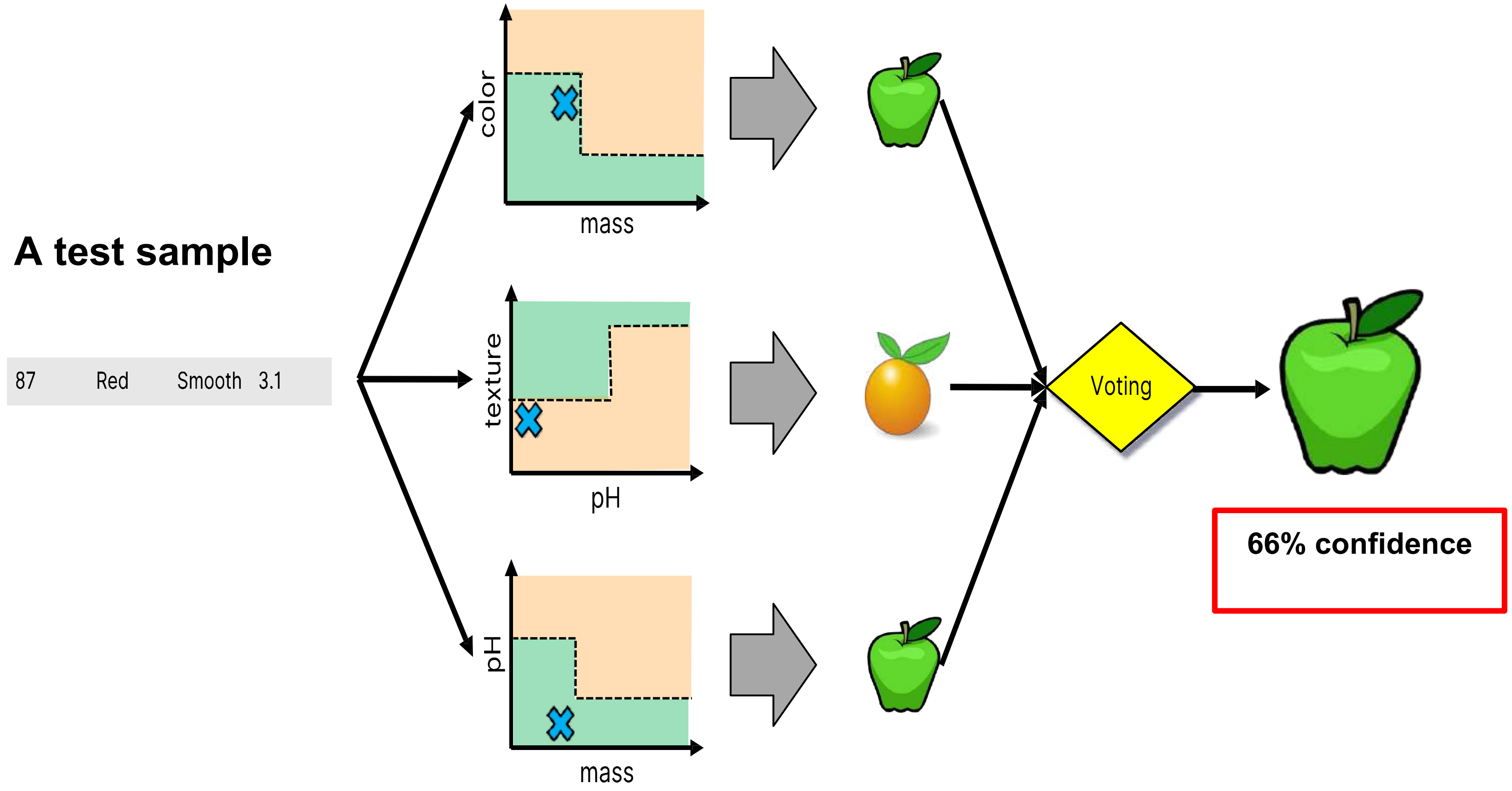
Random Subspace Method at Training Time

Training Data

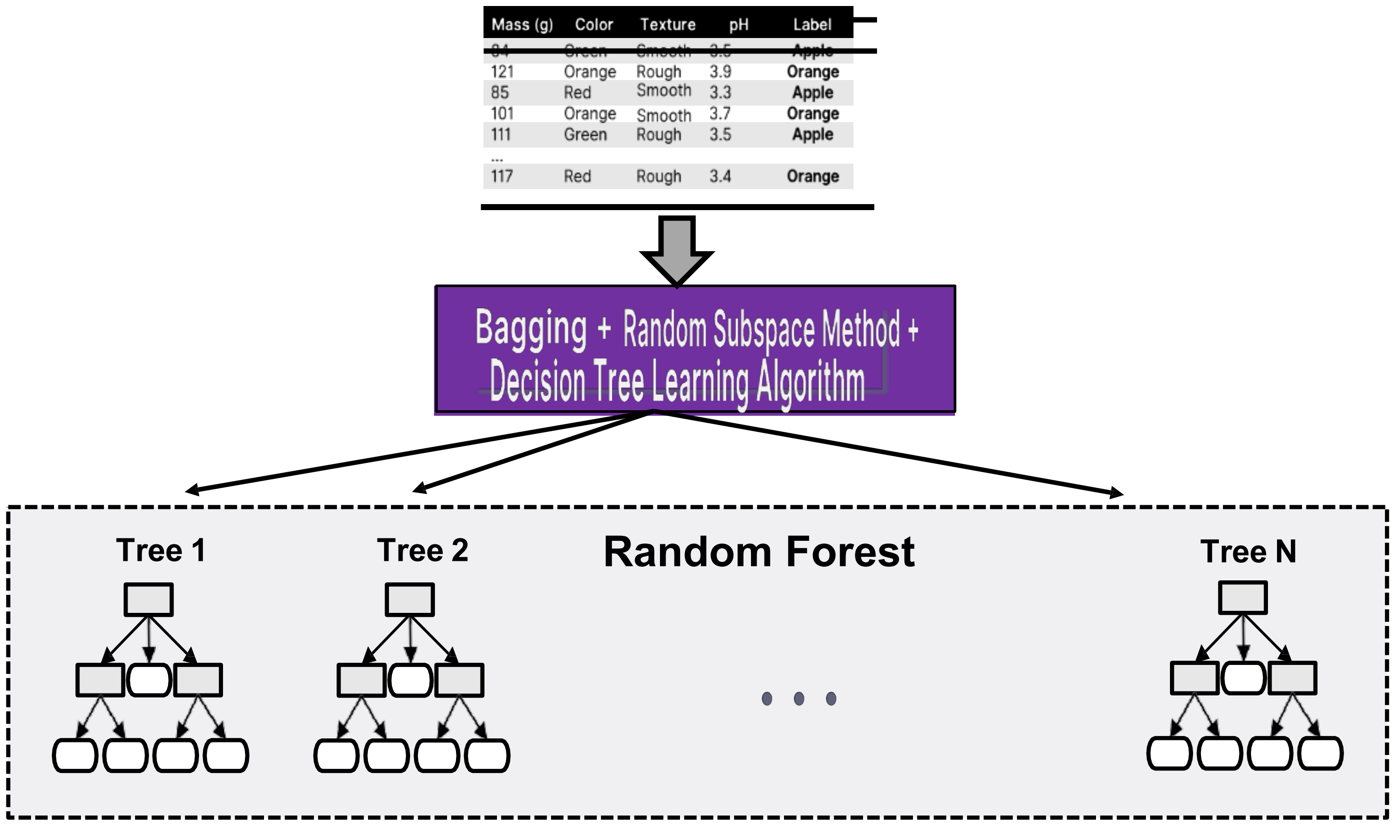
Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	**Apple**
121	Orange	Rough	3.9	**Orange**
85	Red	Smooth	3.3	**Apple**
101	Orange	Smooth	3.7	**Orange**
111	Green	Rough	3.5	**Apple**
...				
117	Red	Rough	3.4	**Orange**



Random Subspace Method at Inference Time



Random Forests



Mass (g)	Color	Texture	pH	Label
~~84~~	~~Green~~	~~Smooth~~	~~3.5~~	~~**Apple**~~
121	Orange	Rough	3.9	**Orange**
85	Red	Smooth	3.3	**Apple**
101	Orange	Smooth	3.7	**Orange**
111	Green	Rough	3.5	**Apple**
...				
117	Red	Rough	3.4	**Orange**

Bagging + Random Subspace Method + Decision Tree Learning Algorithm
Tree 1
Tree 2
Random Forest
Tree N

Summary

- Bagging trains multiple instances of the same model on different data subsets and combines their predictions
- Random Forest extends bagging by introducing additional randomness in feature selection for each tree, enhancing model diversity and reducing overfitting
- Boosting iteratively trains models by focusing on the errors of previous models, adjusting weights to correct misclassified instances and enhance overall accuracy
- Ensemble methods significantly enhance performance compared to single models like Decision Tree

Evaluation of Classification Models

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Evaluation…

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of misclassifying class j example as class i

Cost-Sensitive Measures

$$\text{Precision (p)} = \frac{a}{a+c}$$

$$\text{Recall (r)} = \frac{a}{a+b}$$

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

- Precision is biased towards **C(Yes|Yes) & C(Yes|No)**
- Recall is biased towards **C(Yes|Yes) & C(No|Yes)**
- F-measure is biased towards **all except C(No|No)**

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

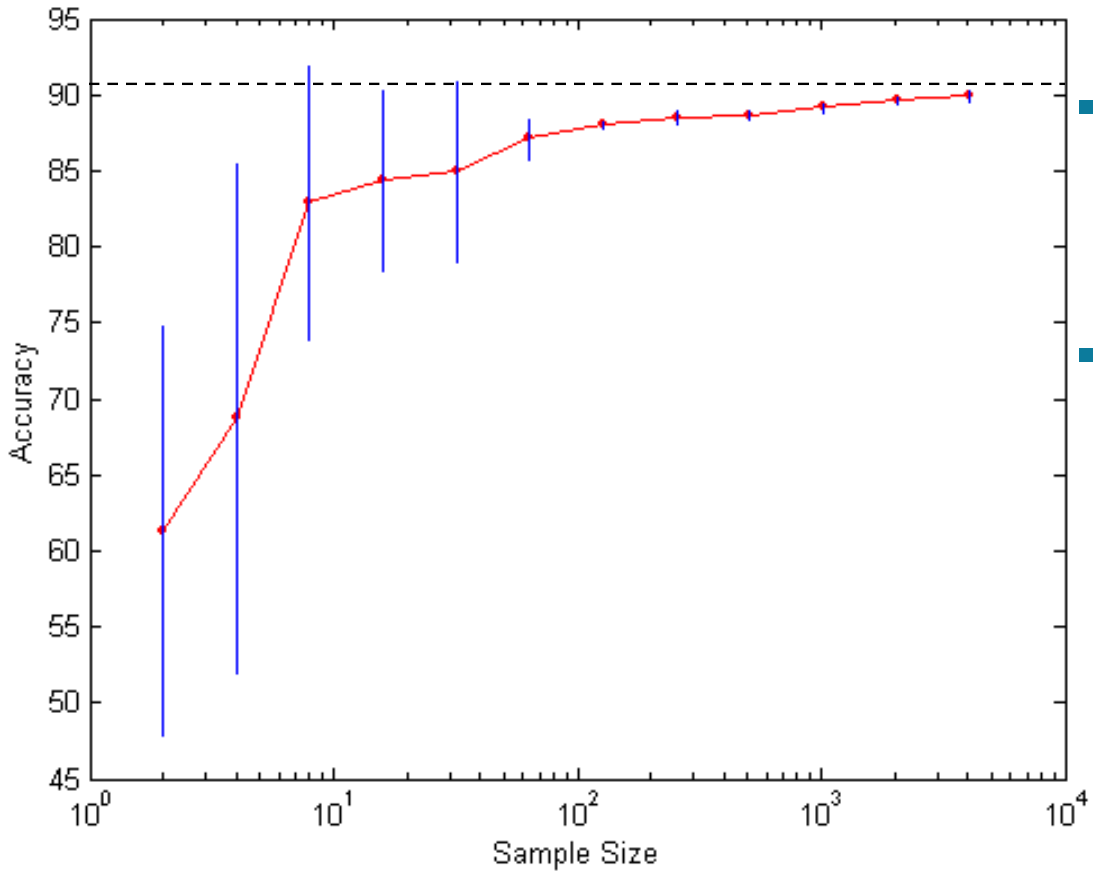
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?

- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires sampling of training data for creating learning curve

- Effect of small sample size:
  - Bias in the estimate
  - Variance of estimate

Methods of Estimation

- Holdout
  - Reserve 2/3 for training and 1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- Stratified sampling
  - oversampling vs undersampling
- Bootstrap
  - Sampling with replacement

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Thank You

Slides Courtesy
1. Introduction to Data Mining, 2nd Edition by Tan, Steinbach, Karpatne, Kumar
2. Prof. Nisheeth Srivastava, IIT Kanpur
3. Prof. Laurent El Ghaoui, UC Berkley
4. Prof. Andrew Moore, CMU
5. Prof. Pabitra Mitra, IIT Kharagpur