Data Mining
Classification: Alternative Techniques

Lecture Notes for Chapter 5

Introduction to Data Mining

by

Tan, Steinbach, Kumar

These slides have been modified for the CS 4232/ 5232 course

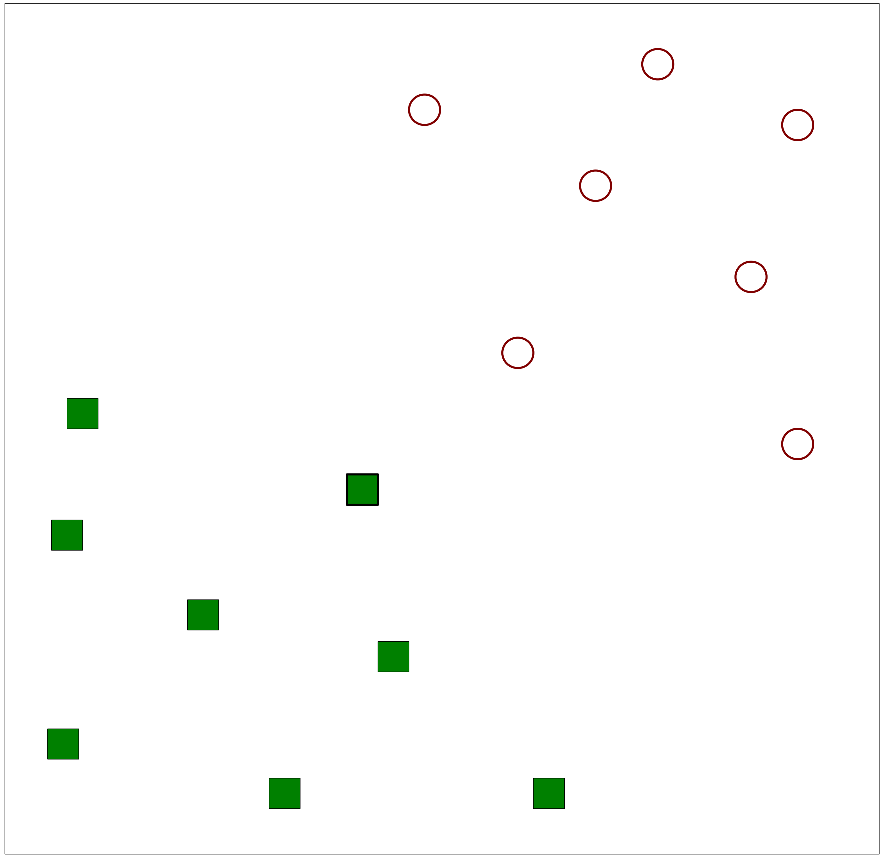
Part IIc

Support Vector Machines, Ensemble Methods, Other Classification Issues

Classifiers

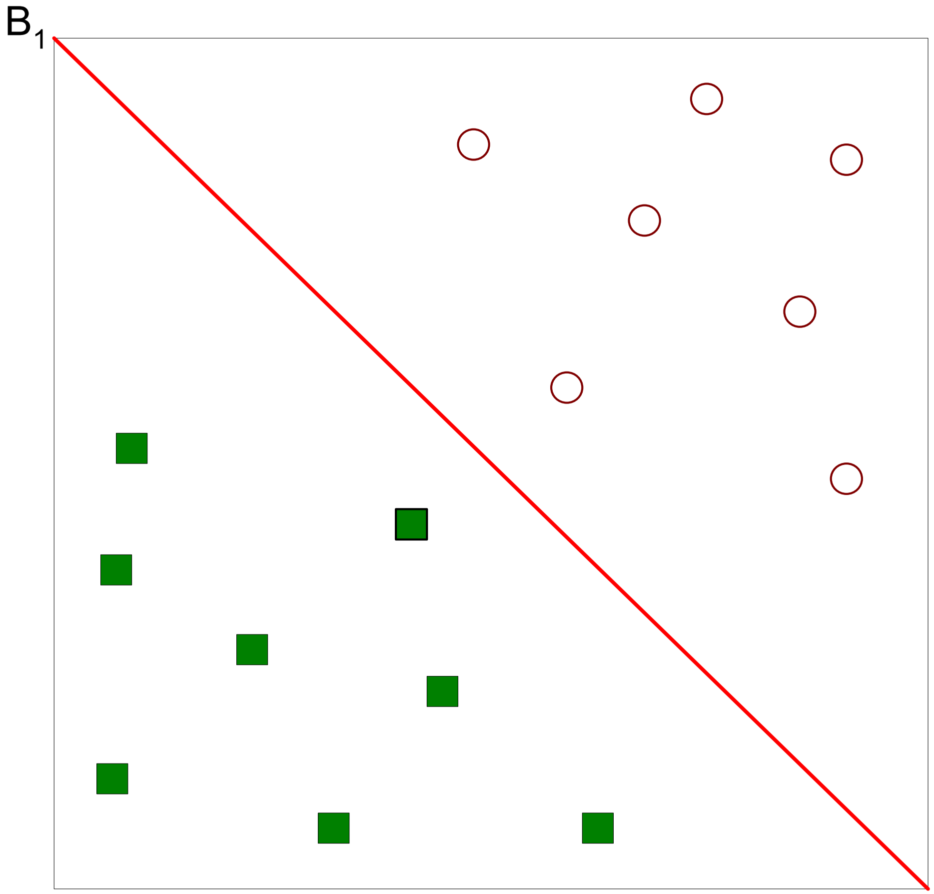
Support Vector Machines

Support Vector Classifier



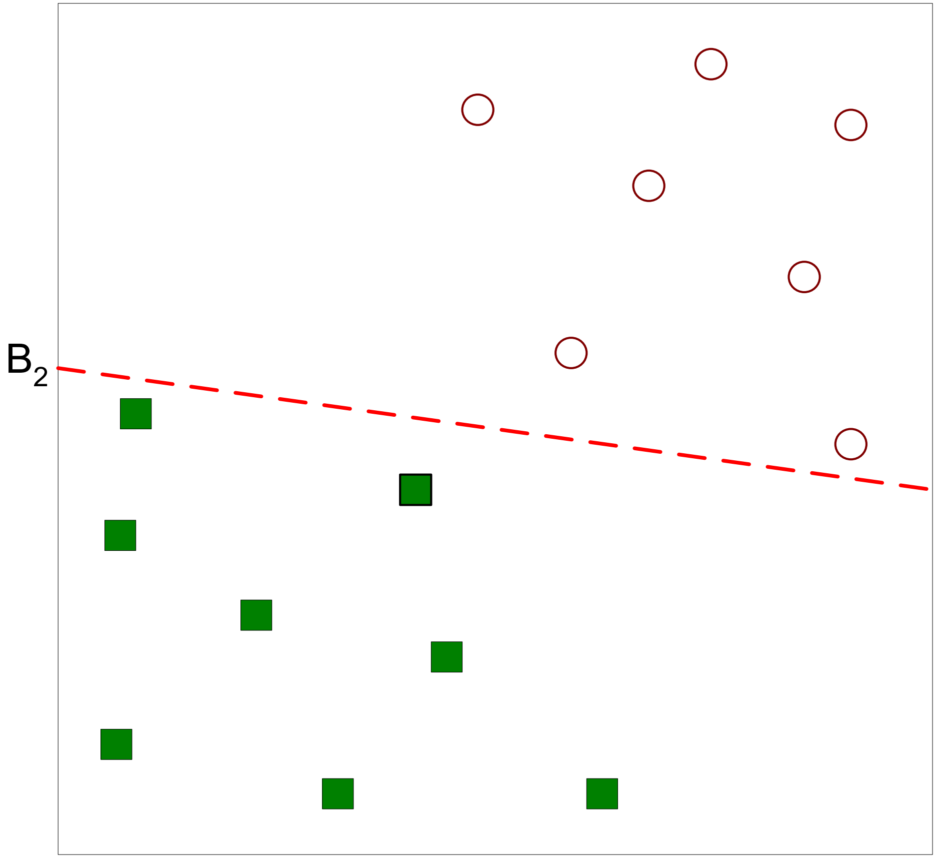
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Classifier



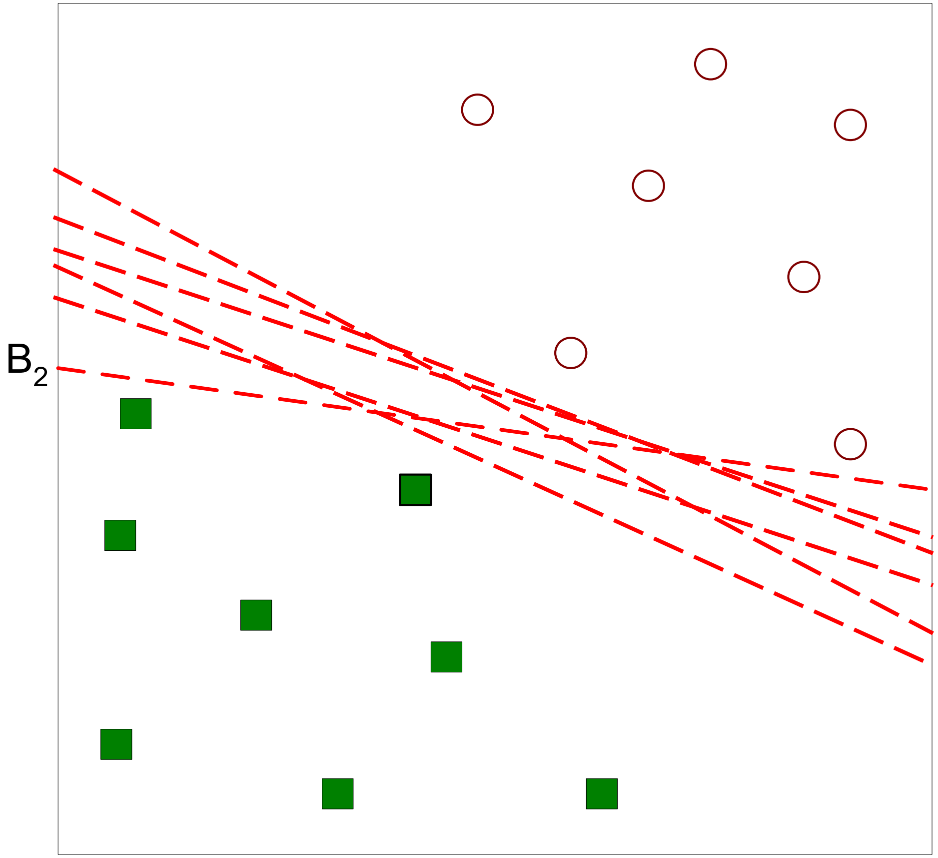
- One Possible Solution

Support Vector Classifier



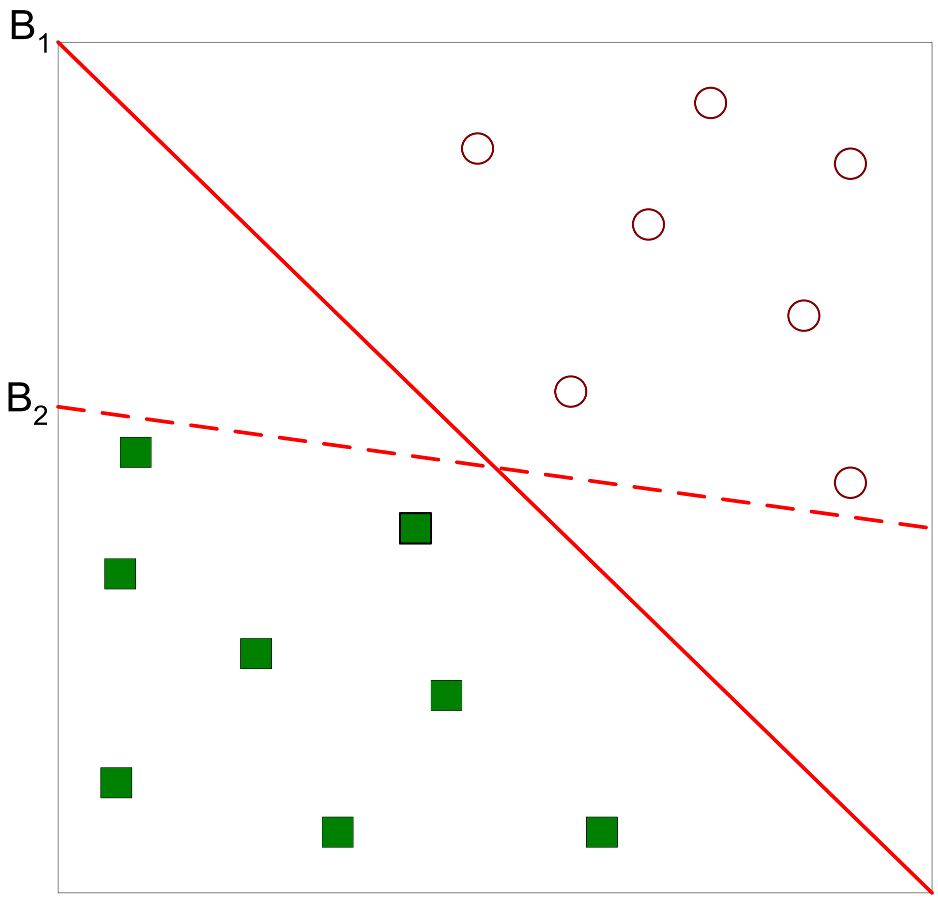
- Another possible solution

Support Vector Classifier



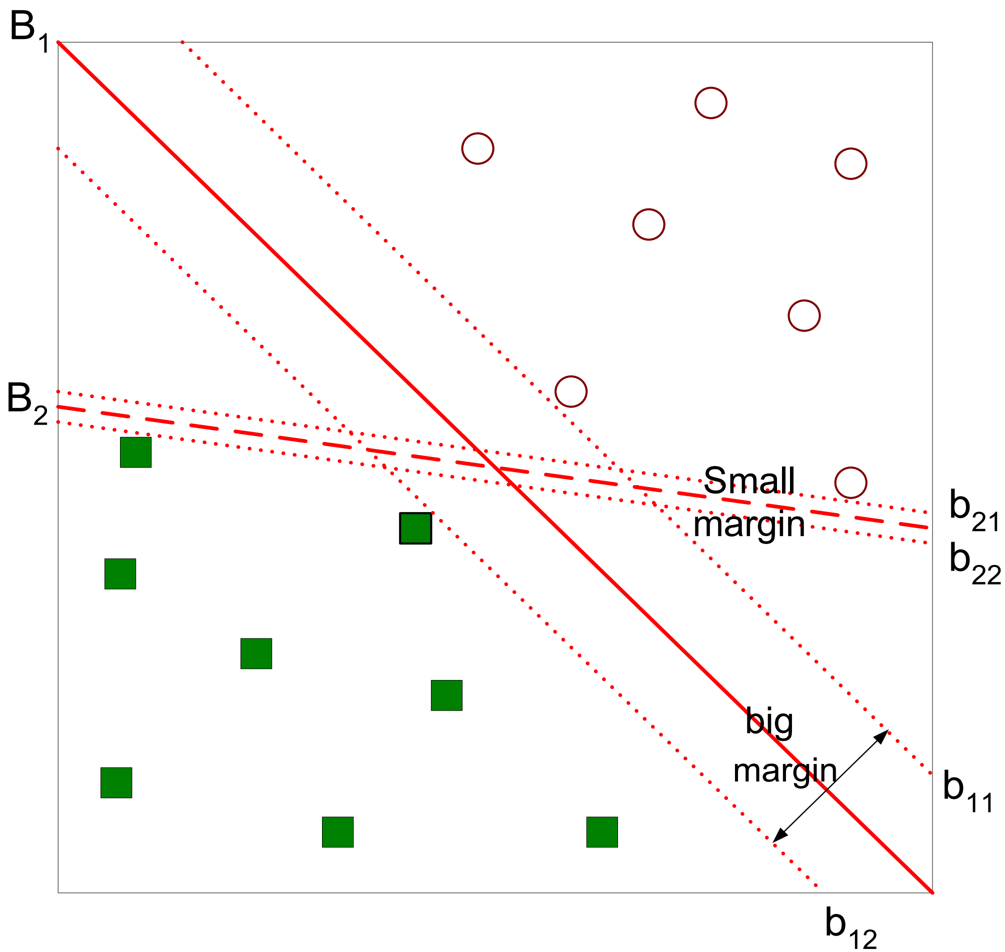
- Other possible solutions

Support Vector Classifier



- Which one is better? B1 or B2?
- How do you define better?

Support Vector Classifier



- The best hyperplanes are those that maximize the margin.
- Therefore, B1 is better than B2

Support Vector Classifier

- We want to maximize: $\text{Margin} = \dfrac{2}{\|\vec{w}\|^2}$ M
  - Which is equivalent to minimizing: $L(w) = \dfrac{\|\vec{w}\|^2}{2}$ $\frac{1}{n}$
  - But subjected to the following constraints:

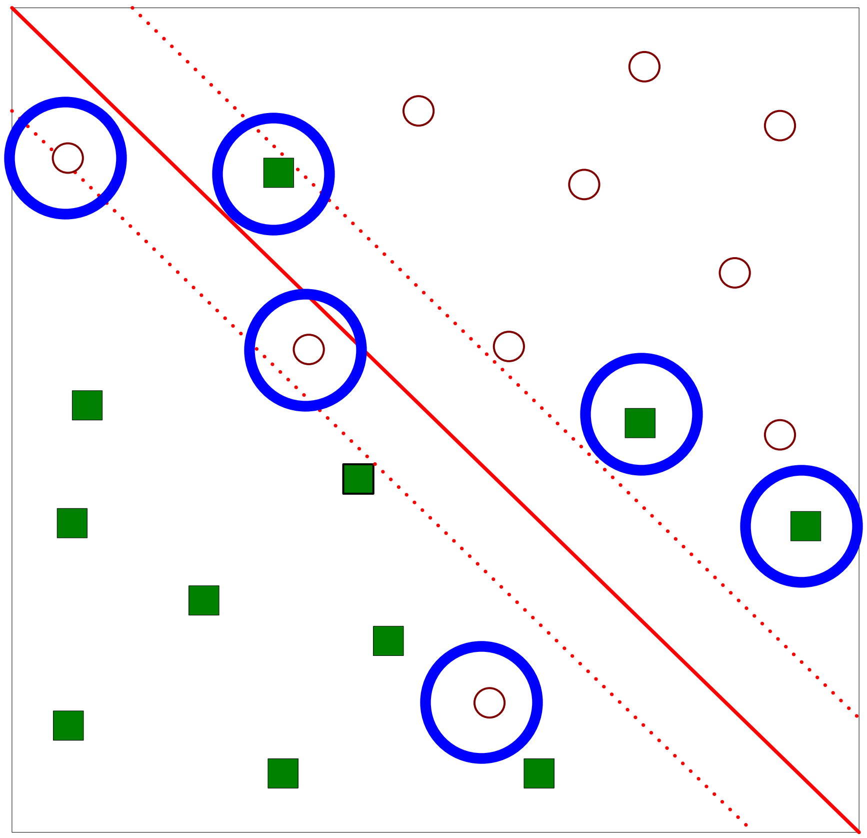
$$y_i = \begin{cases} 1 & \text{if } w \cdot x_i + b \geq 1 \\ -1 & \text{if } w \cdot x_i + b < 1 \end{cases}$$

These constraints are saying: correctly classify all the examples

- This is a constrained optimization problem
  - Numerical approaches to solve it (e.g., quadratic programming)

Support Vector Classifier

- What if the problem is not linearly separable?



Support Vector Classifier

- What if the problem is not linearly separable?
  - Introduce slack variables $\xi_i \geq 0$
    - Need to minimize:

$$L(w) = \frac{\|w\|^2}{2} + C\sum_{i=1}^{n} \xi_i$$

    - Subject to:

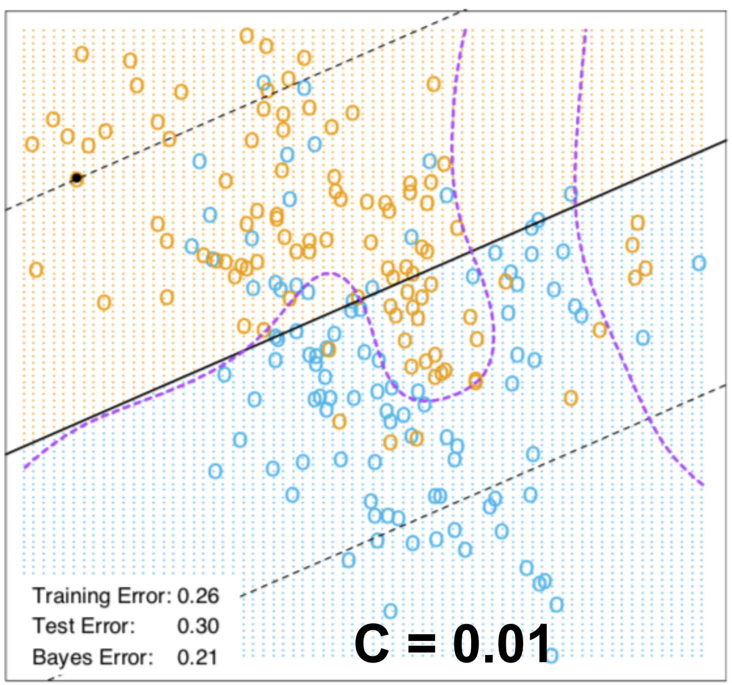
$$y_i = \begin{cases} 1 & \text{if } w \cdot x_i + b \geq 1 - \xi_i \\ -1 & \text{if } w \cdot x_i + b < 1 - \xi_i \end{cases}$$

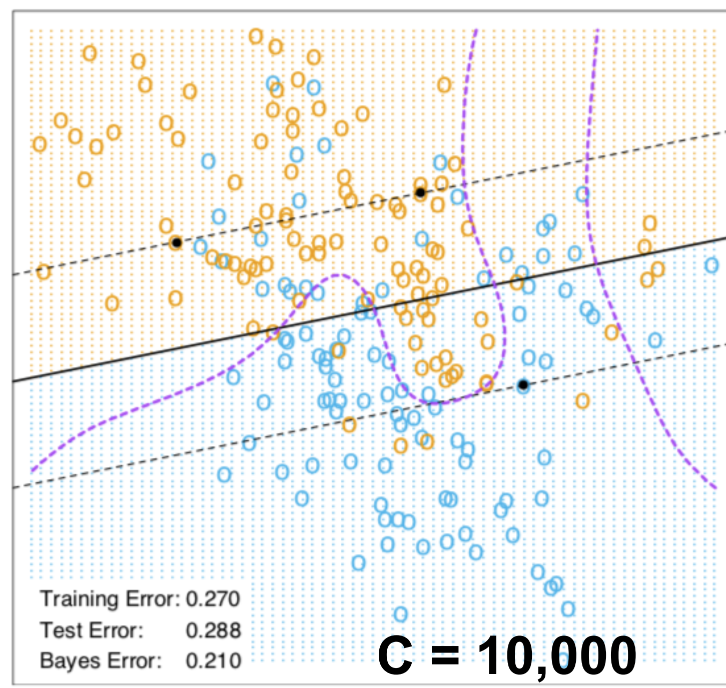
$$\xi_i \geq 0$$

Support Vector Classifier

- Non-linearly separable problem:
  - Minimize $$L(w) = \frac{\|w\|^2}{2} + C\sum_{i=1}^{n}\xi_i$$
  - Subject to:

$$y_i = \begin{cases} 1 & \text{if } w \cdot x_i + b \geq 1 - \xi_i \\ -1 & \text{if } w \cdot x_i + b < 1 - \xi_i \end{cases} \qquad \xi_i \geq 0$$



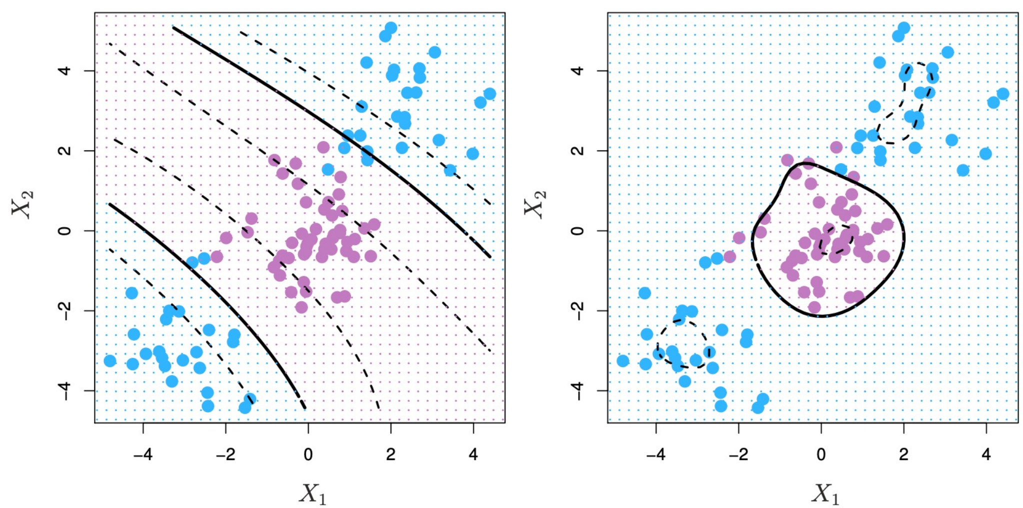


Figures from Hastie, Tibschirani, Friedman. Elements of Statistical Learning. 2nd Edition, 2009

Support Vector Machines

- When also making use of the kernel trick, we call them Support Vector Machines (SVMs)
- Kernels: Kernel Trick

$$
\begin{aligned}
d\text{th-Degree polynomial: } & K(x, x') = (1 + \langle x, x' \rangle)^d, \\
\text{Radial basis: } & K(x, x') = \exp(-\gamma \|x - x'\|^2), \\
\text{Neural network: } & K(x, x') = \tanh(\kappa_1 \langle x, x' \rangle + \kappa_2).
\end{aligned}
$$



Figures from Hastie, Tibschirani, Friedman. Elements of Statistical Learning. 2nd Edition, 2009

Characteristics of SVMs

- Since the learning problem is formulated as a convex optimization problem, efficient algorithms are available to find the global minima of the objective function (many of the other methods use greedy approaches and find locally optimal solutions)
- Overfitting is addressed by maximizing the margin of the decision boundary, but the user still needs to provide the type of kernel function and cost function
- Difficult to handle missing values
- Robust to noise
- High computational complexity for building the model

Advantages and Disadvantages of SVMs

- **Advantages:**
  - SVMs scale linearly in terms of the dimensionality
  - SVMs are memory efficient: to classify a new point, they need to store only the support vectors in memory.
  - SVMs are kernelized
- **Disadvantages:**
  - By default, SVMs do not output probabilities estimates: they only output the classification

Computational Complexity of SVMs

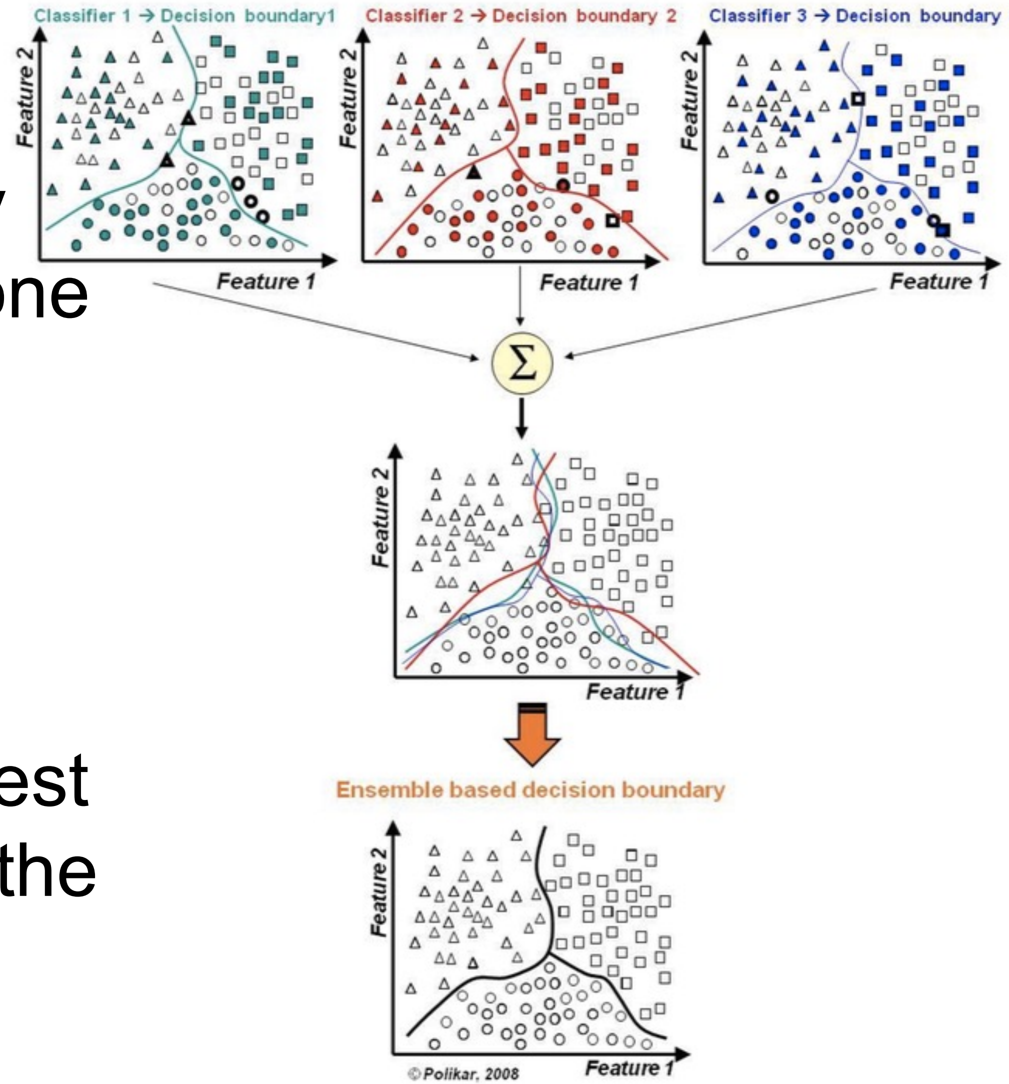
- If using the kernel trick, between n^2p and n^3p where n is the number of samples and p is the number of predictors.

Ensemble Methods

Ensemble Methods

Ensemble Methods

- Aim to improve classification accuracy produced from using one single classifier
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers



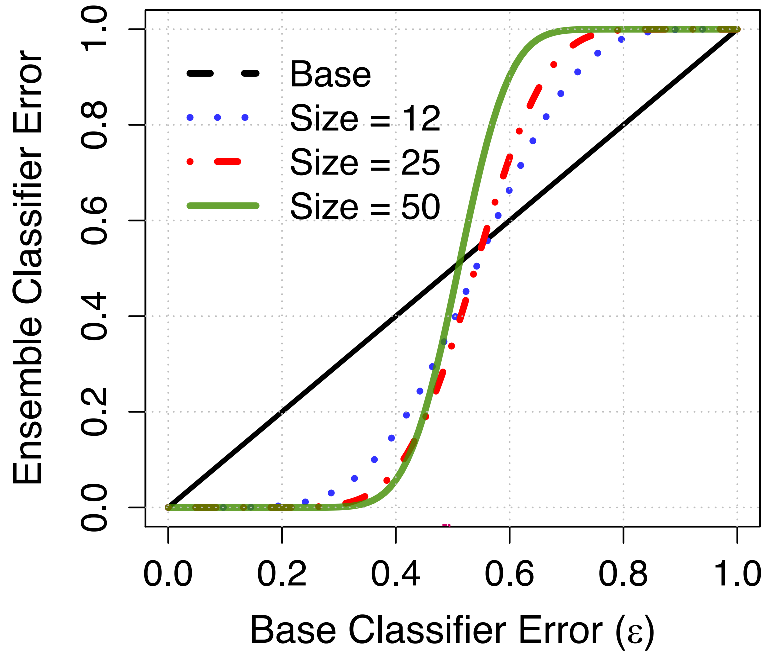
Ensemble Methods

- Construct a set of classifiers from the training data

- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

Why do Ensemble Methods work?

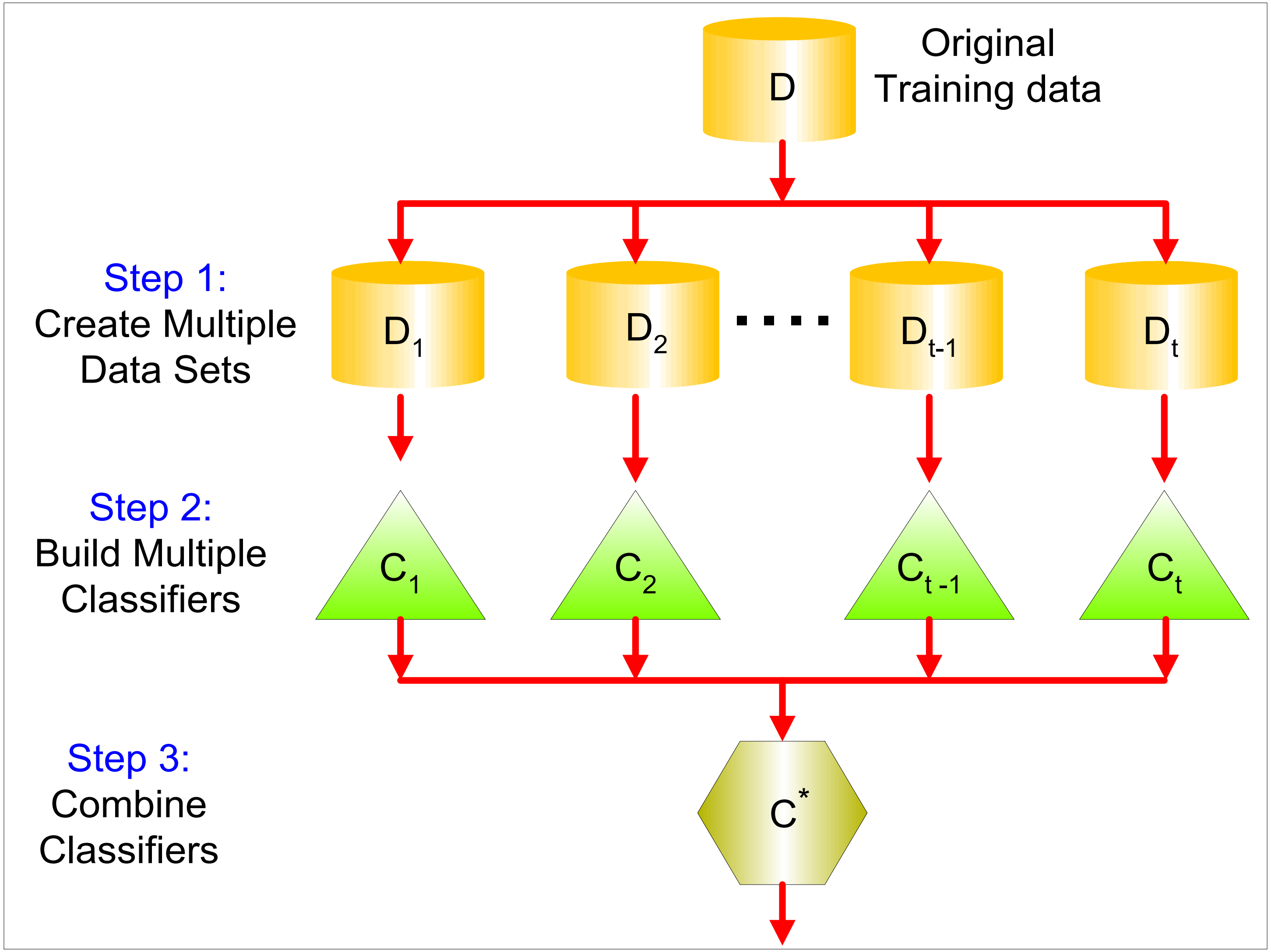
- Suppose there are Size = 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume the **errors** made by the classifiers **are independent**
  - The ensemble makes a wrong prediction only if **more than half** of the base classifiers predict incorrectly
  - Probability that the ensemble classifier makes a wrong prediction:
  - The ensemble classifier performs **worse** than the base classifiers when $\varepsilon > 0.5$



$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

In the figure above, the black line represents the identity function, while the blue, red, and green curves are the ensemble classification errors for an ensemble of 12, 25, and 50 classifiers, respectively

General Idea



Types of Ensemble Methods

- Manipulate data distribution/training set
  - **Multiple** training sets are created by **resampling** the original data
  - A classifier is built from each training set using a particular learning algorithm
  - Examples: Bagging and Boosting
- Manipulate input features
  - A **subset** of input features is chosen to form each training set
  - Either chosen randomly or based on the recommendation of domain experts
  - Works well with data that contain highly redundant features
  - Example: Random Forests

Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
  - Bagging
  - Boosting

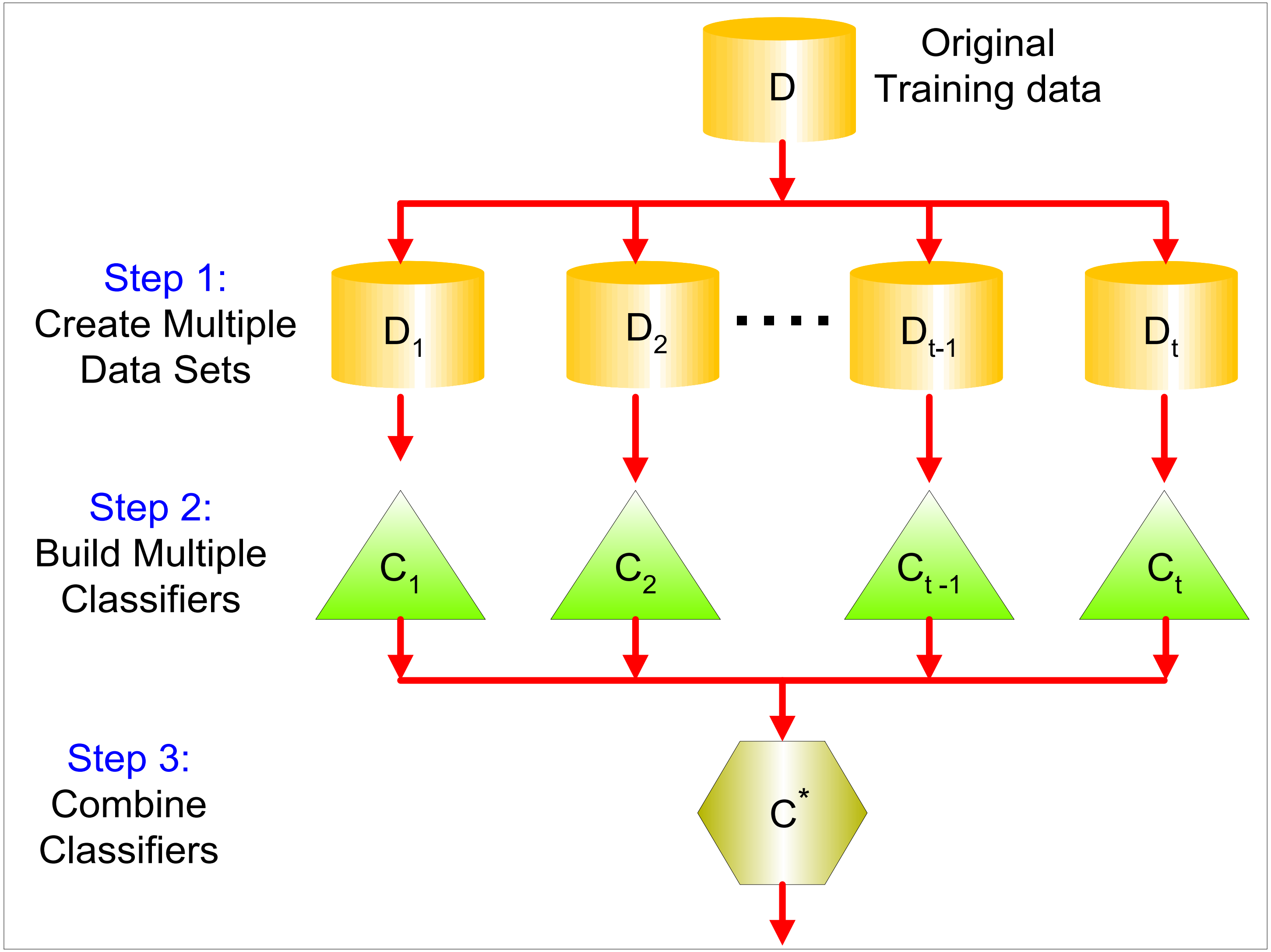
Bagging

- Sampling with replacement
  - Each bootstrap sample has the **same size as the original data**
  - Some instances **may appear several times**
  - Others may be omitted

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each **bootstrap** sample
- Each sample has probability $1-(1 - 1/n)^n$ of being selected, so the sample contains around 63% of the original training data

General Idea



Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

n is the size of the dataset

- Train *k* classifiers
- A test instance is assigned to the class that receives the highest number of votes

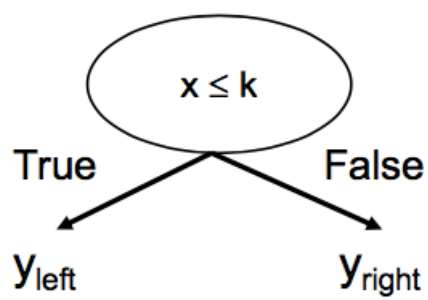
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

Bagging Example

$$Entropy(t) = -\sum_j p(j\,|\,t)\log_2 p(j\,|\,t)$$

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 → y = 1
x > 0.05 → y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging

- It improves error **by reducing the variance** of the base classifiers
- Its performance depends on the stability of the base classifier
  - If a base classifier is unstable, bagging helps to reduce errors associated with random fluctuations in the training data.
  - If a base classifier is **stable**, bagging **might not be able to improve error**; it might degrade the classifier's performance due to its use of 37% less training data.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on **previously misclassified** records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, **weights may change at the end** of boosting round

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased; therefore, it is more likely to be chosen again in subsequent rounds

Example: AdaBoost

$\delta(p) = \begin{cases} 1 & \text{if } p \text{ is true} \\ 0 & \text{otherwise} \end{cases}$

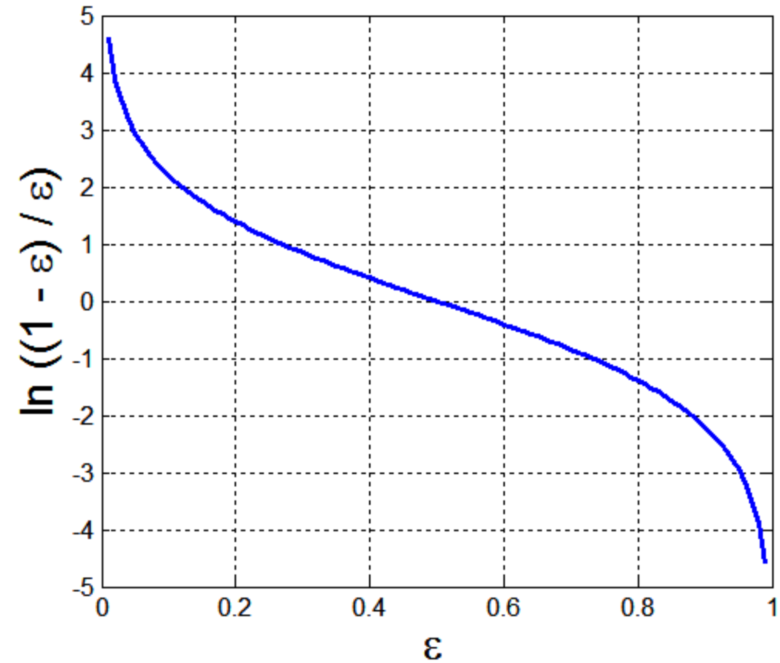
- Base classifiers: $C_1, C_2, \ldots, C_i, \ldots, C_T$

- Error rate of the i-th classifier:

$$\epsilon_i = \frac{1}{n} \sum_{j=1}^{n} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of the i-th classifier:

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1 - \epsilon_i}{\epsilon_i}\right)$$



Classifiers with smaller (weighted) errors should be more important (have larger alpha)

Example: AdaBoost

- Weight update:

w_i is the weight of the i-th example (row j is an index over classifiers)

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

- If any intermediate rounds produce an error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated
- Classification:

$$C^*(x) = arg\, max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(x) = y)$$

AdaBoost Algorithm

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \frac{1}{n}\left[\sum_j w_j\, \delta\big(C_i(x_j) \neq y_j\big)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\big(C_j(\mathbf{x}) = y\big)$.

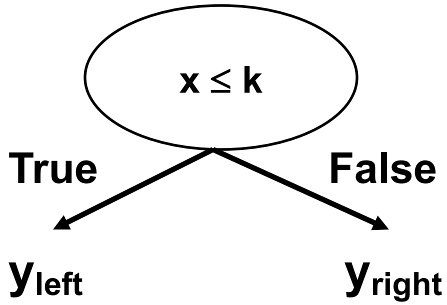
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \le k$ versus $x > k$
  - Split point k is chosen based on entropy



AdaBoost Example

- Boosting Round 1:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

x <= 0.75 → -1
x > 0.75 → 1
Error rate(whole dataset) = 0.3

Weights:

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

All weights equal!

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738

The errors above are not weighted errors.

AdaBoost Example

- Boosting Round 2:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 → -1
x > 0.05 → 1
Error rate(whole dataset) = 0.4

Weights:

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Examples that are hard to train have higher weights

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784

AdaBoost Example

- Boosting Round 3:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

Weights:

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

x <= 0.25 → 1
x > 0.25 → -1
Error rate(whole dataset) = 0.4

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

x <= 0.75 ➔ -1
x > 0.75 ➔ 1
Error rate(whole dataset) = 0.3

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ -1
x > 0.05 ➔ 1
Error rate(whole dataset) = 0.4

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

x <= 0.25 ➔ 1
x > 0.25 ➔ -1
Error rate(whole dataset) = 0.4

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

The errors above are not weighted errors.

AdaBoost Example

- Using the ensemble for classification:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

X= 0.1 gets a sum of 5.16 because we combine the predictions of all three rounds. For example, the prediction for x = 0.1 are -1, 1, and 1. Therefore, sign(sum) = sign(-1 * 1.738 + 1 * 2.7784 + 1 * 4.1195) = sign(5.16) = 1

Random Forests

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

From T. Hastie, R. Tibschirani, Friedman. The Elements of Statistical Learning, Springer 2009

Classification

The Class Imbalance Problem

The Class Imbalance Problem

- Lots of classification problems where the classes are skewed (more records from one class than another)
  - Credit card fraud
  - Intrusion detection
  - Defective products in manufacturing assembly line

Challenges

- Evaluation measures such as **accuracy is not well-suited for an imbalanced class**
  - For example, if 1% of credit card transactions are fraudulent
  - Predict every transaction legitimate => 99% accuracy
    - Fail to detect fraudulent activities

- Detecting the rare class is like finding needle in a haystack
  - Susceptible to **noise** in the training data

Metrics for Performance Evaluation

- 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:

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

Problem with Accuracy

- Consider a 2-class problem:
  – Number of Class 0 examples = 1990
  – Number of Class 1 examples = 10
- If a model predicts everything to be class 0, accuracy is 9990/10000 = 99.9%
  – This is misleading because the model does not detect any class 1 example
  – Detecting the rare class is usually more interesting (e.g., frauds, intrusions, defects, etc.)

Alternative Measures

	PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class=No

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

- Precision: fraction of the records predicted as positive, that are truly positive
- Recall: fraction of the true positive records correctly predicted as positive.
- F-measure: The harmonic mean between precision and recall

$$H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{n}}$$

$$\text{precision}(p) = \frac{a}{a+c} = \frac{TP}{TP+FP}$$

$$\text{recall}(r) = \frac{a}{a+b} = \frac{TP}{TP+FN}$$

$$\text{f-measure}(F) = \frac{2}{\frac{1}{r} + \frac{1}{p}} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

Alternative Measures

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

- True positive rate (TPR): fraction of the true positive records correctly predicted as positive.
- False positive rate (FPR): fraction of the true negative records incorrectly predicted as positive.

$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \text{recall}$$

$$FPR = \frac{FP}{TN + FP} = \frac{c}{c+d}$$

Alternative Measures

	PREDICTED CLASS	

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

$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \text{recall}$$

$$FPR = \frac{FP}{TN + FP} = \frac{c}{c+d}$$

$$H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \cdots + \frac{1}{a_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{n}}$$

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

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

$$\text{f-measure}(F) = \frac{2}{\frac{1}{r} + \frac{1}{p}} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

Alternative Measures

	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	10	0
	Class=No	10	980

$$\text{precision}(p) = \frac{10}{10+10} = 0.5$$

$$\text{recall}(r) = \frac{10}{10+0} = 1$$

$$\text{f-measure}(F) = \frac{2 \cdot 1 \cdot 0.5}{1+0.5} = 0.62$$

$$\text{accuracy} = \frac{990}{1000} = 0.99$$

Measures of Classification Performance

	PREDICTED CLASS		
ACTUAL CLASS		Yes	No
	Yes	TP	FN
	No	FP	TN

α is the probability that we reject the null hypothesis when it is true. This is a Type I error or a false positive (FP).

β is the probability that we accept the null hypothesis when it is false. This is a Type II error or a false negative (FN).

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$ErrorRate = 1 - accuracy$$

$$Precision = Positive\ Predictive\ Value = \frac{TP}{TP + FP}$$

$$Recall = Sensitivity = TP\ Rate = \frac{TP}{TP + FN}$$

$$Specificity = TN\ Rate = \frac{TN}{TN + FP}$$

$$FP\ Rate = \alpha = \frac{FP}{TN + FP} = 1 - specificity$$

$$FN\ Rate = \beta = \frac{FN}{FN + TP} = 1 - sensitivity$$

$$Power = sensitivity = 1 - \beta$$

ROC (Receiver Operating Characteristic)

- A graphical approach for displaying trade-off between detection rate and false alarm rate
- Developed in 1950s for signal detection theory to analyze noisy signals
- ROC curve plots TPR against FPR
  - Performance of a model represented as a point in an ROC curve
  - Changing the threshold parameter of classifier changes the location of the point

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

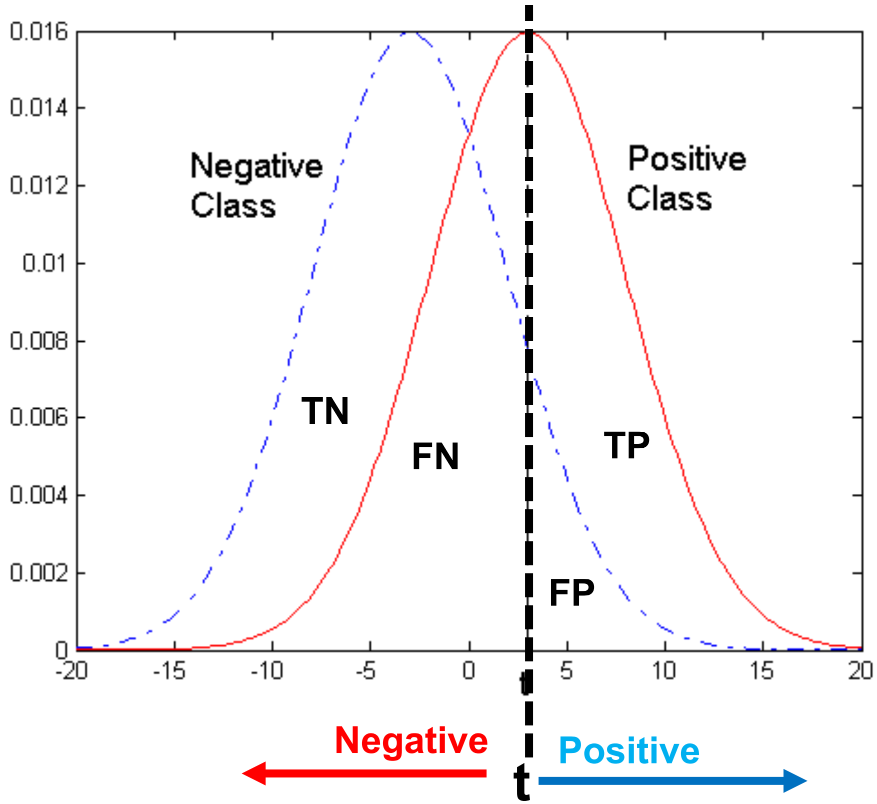
$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \text{recall}$$

$$FPR = \frac{FP}{TN + FP} = \frac{c}{c+d}$$

ROC Curve

$$TPR = \frac{TP}{TP+FN} = \frac{a}{a+b} = \text{recall} \qquad FPR = \frac{FP}{TN+FP} = \frac{c}{c+d}$$

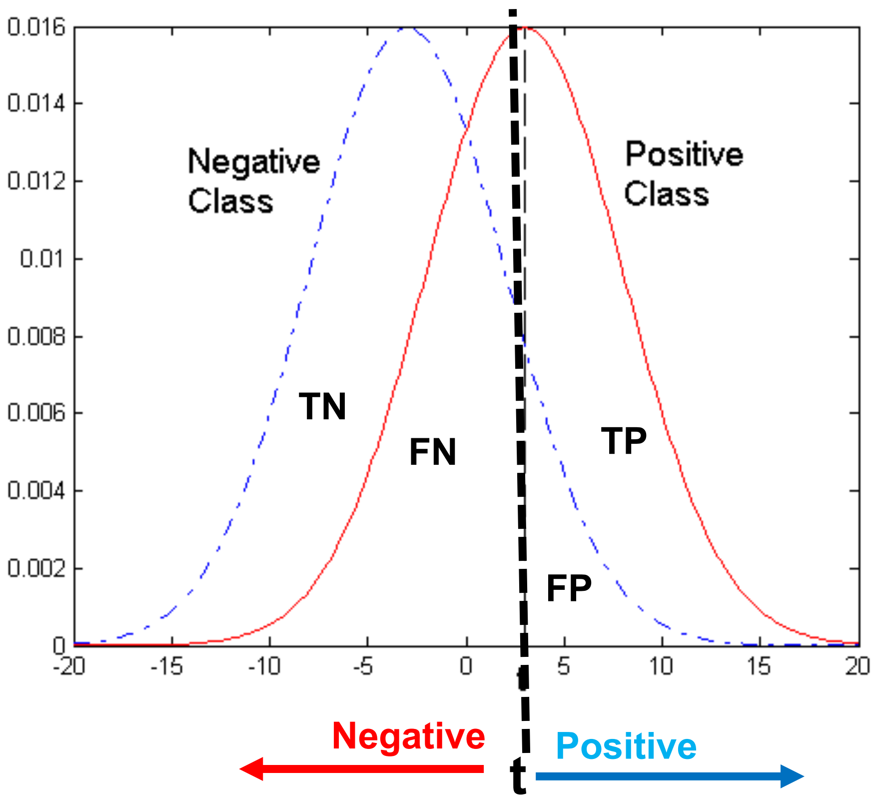
- 1-dimensional data set containing 2 classes (positive and negative)
- Any point located at x > t is classified as positive

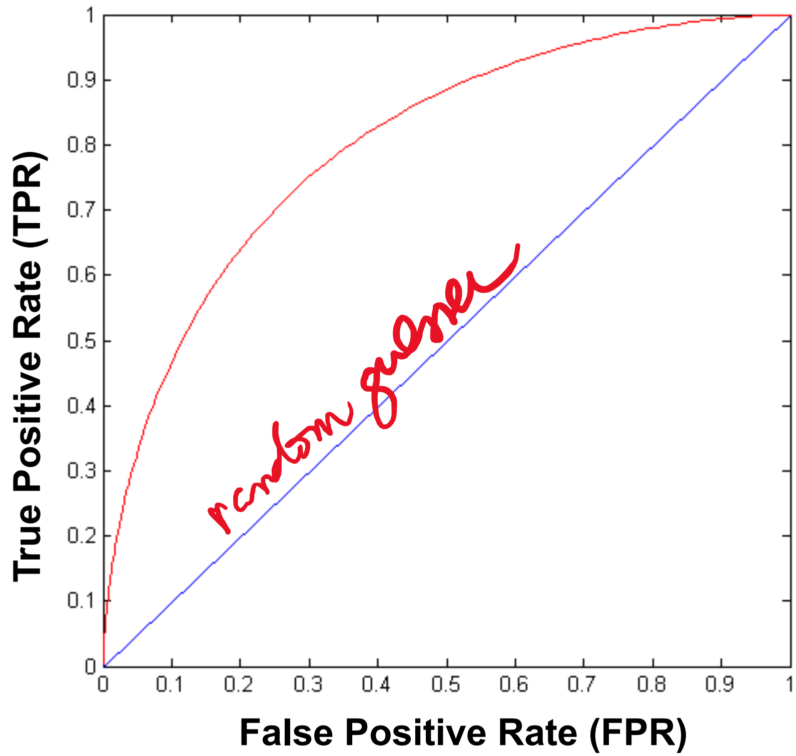


ROC Curve

$$TPR = \frac{TP}{TP+FN} = \frac{a}{a+b} = \text{recall} \qquad FPR = \frac{FP}{TN+FP} = \frac{c}{c+d}$$

- **1-dimensional data set** containing 2 classes (positive and negative)
- Any point located at x > t is classified as positive



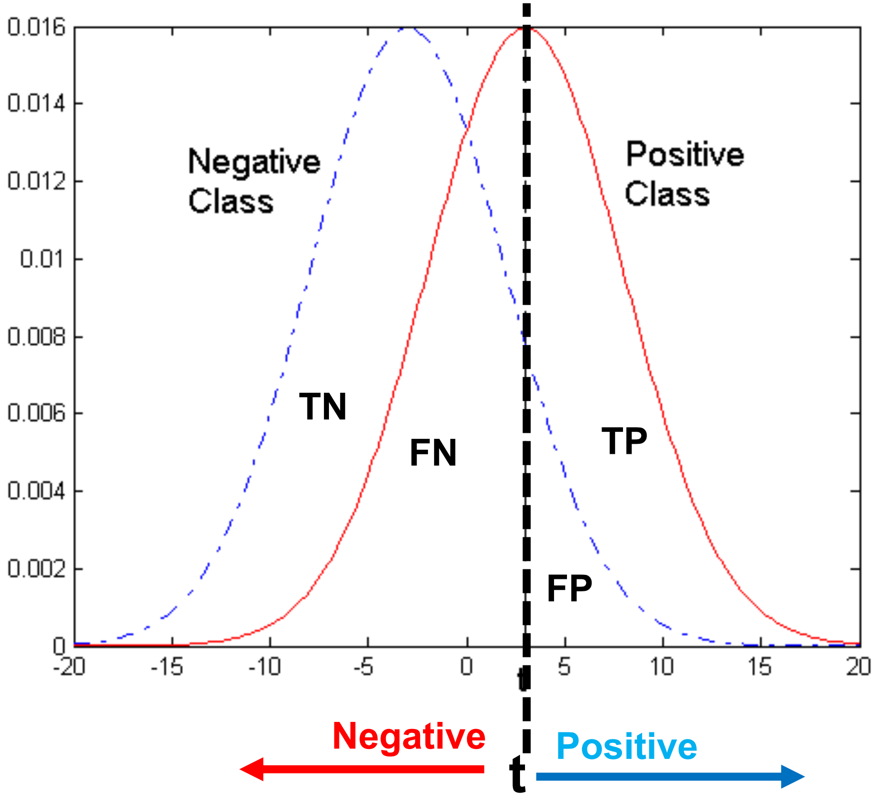


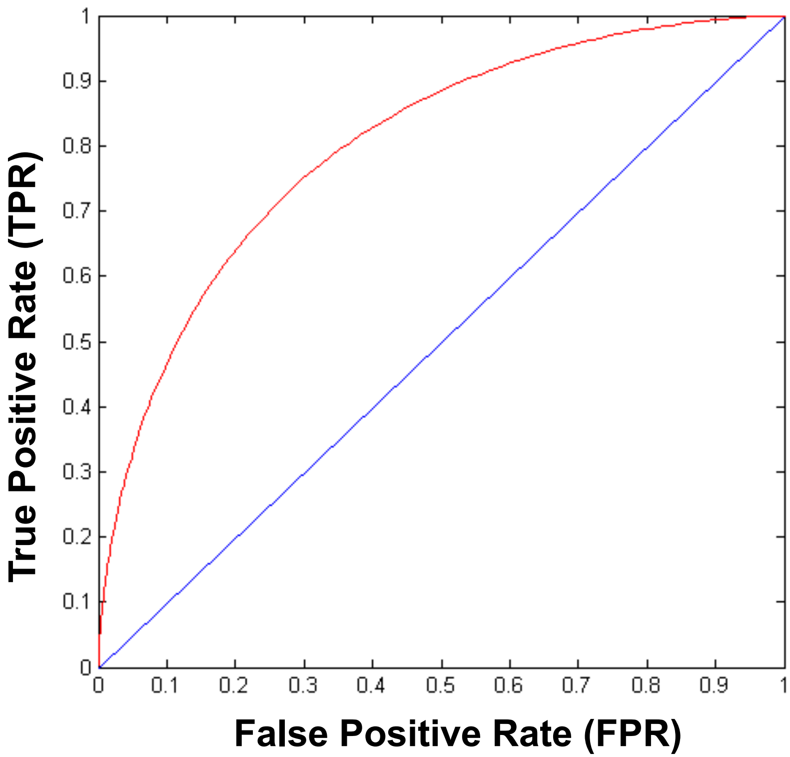
As you move the border t, we get different models, each with different TPR and FPR pair of values. We can plot TPR vs. FPR for different values of t.

ROC Curve

$$TPR = \frac{TP}{TP+FN} = \frac{a}{a+b} = \text{recall} \qquad FPR = \frac{FP}{TN+FP} = \frac{c}{c+d}$$

- 1-dimensional data set containing 2 classes (positive and negative)
- any points located at x > t is classified as positive





At threshold t:

TP=0.5, FN=0.5, FP=0.12, FN=0.88

ROC Curve

(TPR,FPR)= $TPR = \frac{TP}{TP+FN} = \frac{a}{a+b} = \text{recall} \quad FPR = \frac{FP}{TN+FP} = \frac{c}{c+d}$

- (0,0): Therefore, TP = 0 and FP=0, so I declared everything is negative
- (1,1): Therefore, FN = 0, TN = 0, so I declared everything is positive
- (1,0): ideal (FN = 0, and FP = 0)

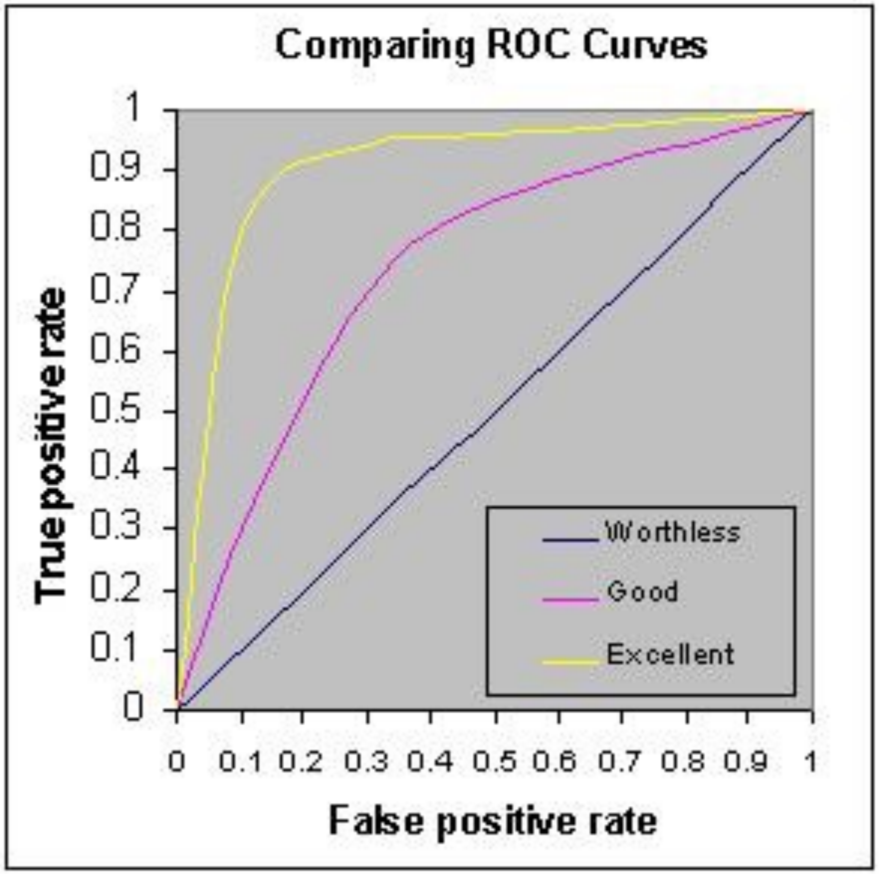
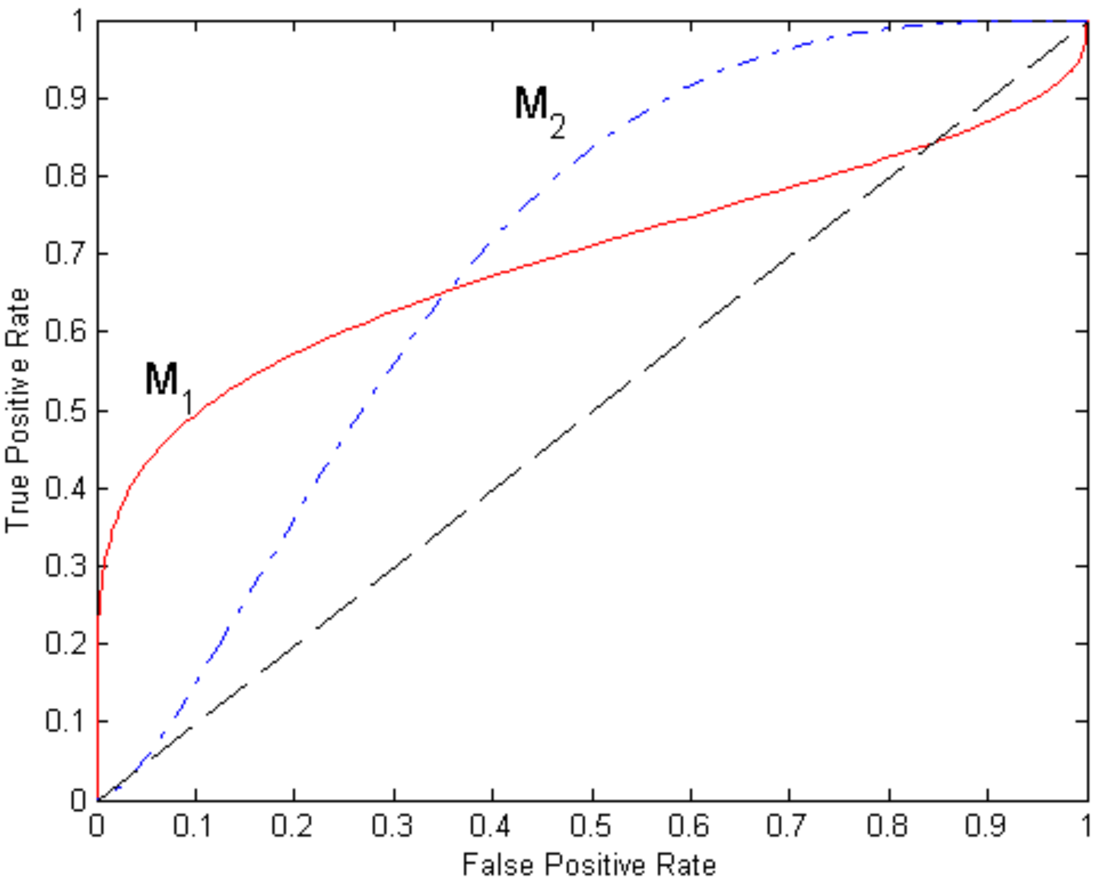


Figure from http://gim.unmc.edu/dxtests/roc3.htm

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class

Using ROC for Model Comparison



- No model consistently outperforms the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5

$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \text{recall} \qquad FPR = \frac{FP}{TN + FP} = \frac{c}{c+d}$$

How to Construct an ROC curve

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

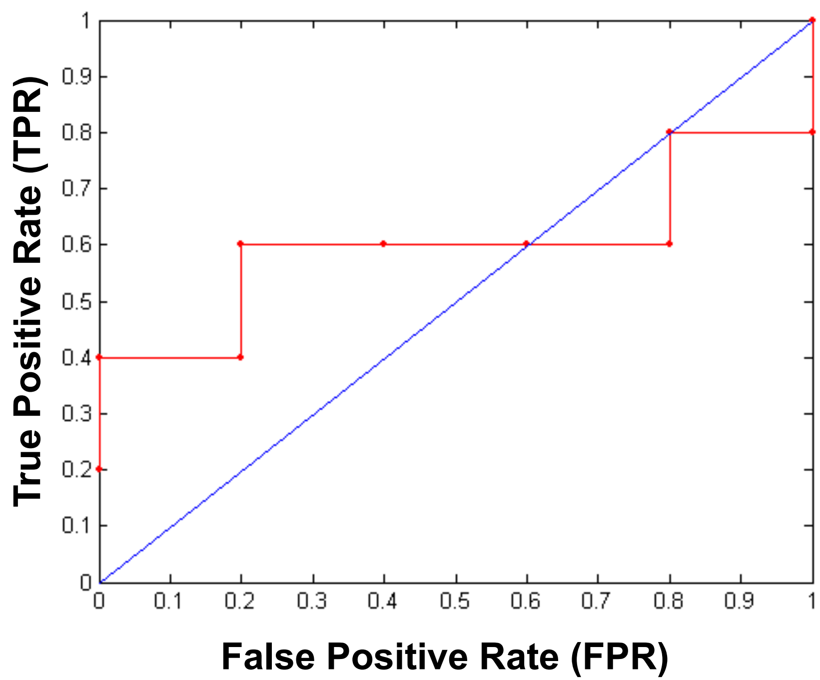
This is like in the example of the Gaussian we saw before, where we "moved t". P(+|A) here plays the role of t in that other example.

- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/(TP+FN)
- FP rate, FPR = FP/(FP + TN)

How to construct an ROC curve

Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=	0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
TP	5	4	4	3	3	3	3	2	2	1	0
FP	5	5	4	4	3	2	1	1	0	0	0
TN	0	0	1	1	2	3	4	4	5	5	5
FN	0	1	1	2	2	2	2	3	3	4	5
TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

ROC Curve:



$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \text{recall}$$

$$FPR = \frac{FP}{TN + FP} = \frac{c}{c+d}$$

Handling the Class Imbalance Problem

- Cost-sensitive classification
  - Misclassifying a rare class as belonging to a majority class is **more expensive** than misclassifying a majority instance as belonging to a rare class

- Sampling-based approaches

Cost Matrix

	PREDICTED CLASS	
ACTUAL CLASS	Class=Yes	Class=No
Class=Yes	f(Yes, Yes)	f(Yes,No)
Class=No	f(No, Yes)	f(No, No)

Cost Matrix	PREDICTED CLASS	
ACTUAL CLASS C(i, j)	Class=Yes	Class=No
Class=Yes	C(Yes, Yes)	C(Yes, No)
Class=No	C(No, Yes)	C(No, No)

f(i,j): number of examples belonging to class i that are classified to be in class j

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

$$\text{Cost} = \sum C(i,j) \times f(i,j)$$

Computing Cost of Classification: Example

This cost function penalizes not finding an outlier (class +) more severely than false alarms.

Cost Matrix	PREDICTED CLASS		
	C(i,j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

You say there is no rain, when it is raining

+ rare class
- majority class

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

Which model is better?

Cost Sensitive Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i,j)	+	-
	+	C(+,+)	C(+,-)
	-	C(-,+)	C(-.-)

- Example: Bayesian classifer
  - Given a test record x:
    - Compute p(i|x) for each class *i*
    - Decision rule: classify node as class k if

$$k = \arg\max_{i} p(i \mid x)$$

  - For 2-class, classify x as + if p(+|x) > p(-|x)
    - This decision rule implicitly assumes that C(+|+) = C(-|-) = 0 and C(+|-) = C(-|+)

Cost Sensitive Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i,j)	+	-
	+	C(+,+)	C(+,-)
	-	C(-,+)	C(-.-)

- General decision rule:
  - Classify test record x as class k if

$$k = \arg\min_j \sum_i p(i \mid x) \times C(i, j)$$

- 2-class:
  - Cost(+) = p(+|x) C(+,+) + p(-|x) C(-,+)
  - Cost(-) = p(+|x) C(+,-) + p(-|x) C(-,-)
  - Decision rule: classify x as + if Cost(+) < Cost(-)
    - if C(+,+) = C(-,-) = 0:

$$p(+ \mid x) > \frac{C(-,+)}{C(-,+) + C(+,-)}$$

Using ROC for Model Comparison

- Cost-sensitive classification
  - Misclassifying rare class as majority class is more expensive than misclassifying majority as rare class
- Sampling-based approaches

Sampling-based Approaches

- Modify the distribution of training data so that the rare class is well-represented in the training set
  - For example, 100 positive examples and 1000 negative examples
  - Under-sample the majority class
    - Random or focused subsampling
  - Oversample the rare class
    - Replicate existing positive examples
    - Generate new positive examples in the neighborhood of the existing positive examples

Summary of Topic 4

- Instance-Based Classifiers
  - Nearest-Neighbor Classifiers
- Linear Classifier
  - Logistic Regression Classifier
- Bayesian Classifiers
  - Naïve Bayes Classifier
- Neural Networks
  - Perceptron
  - Neural Networks and Deep Learning
- SVMs
- Ensemble Methods
  - Bagging
  - Boosting
  - Random Forests
- Class Imbalance Problem
  - Alternative Performance Evaluation Measures
  - ROC
  - Cost Matrix
  - Cost-Sensitive Classification
  - Sampling-based Approaches