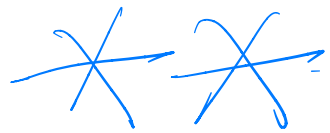
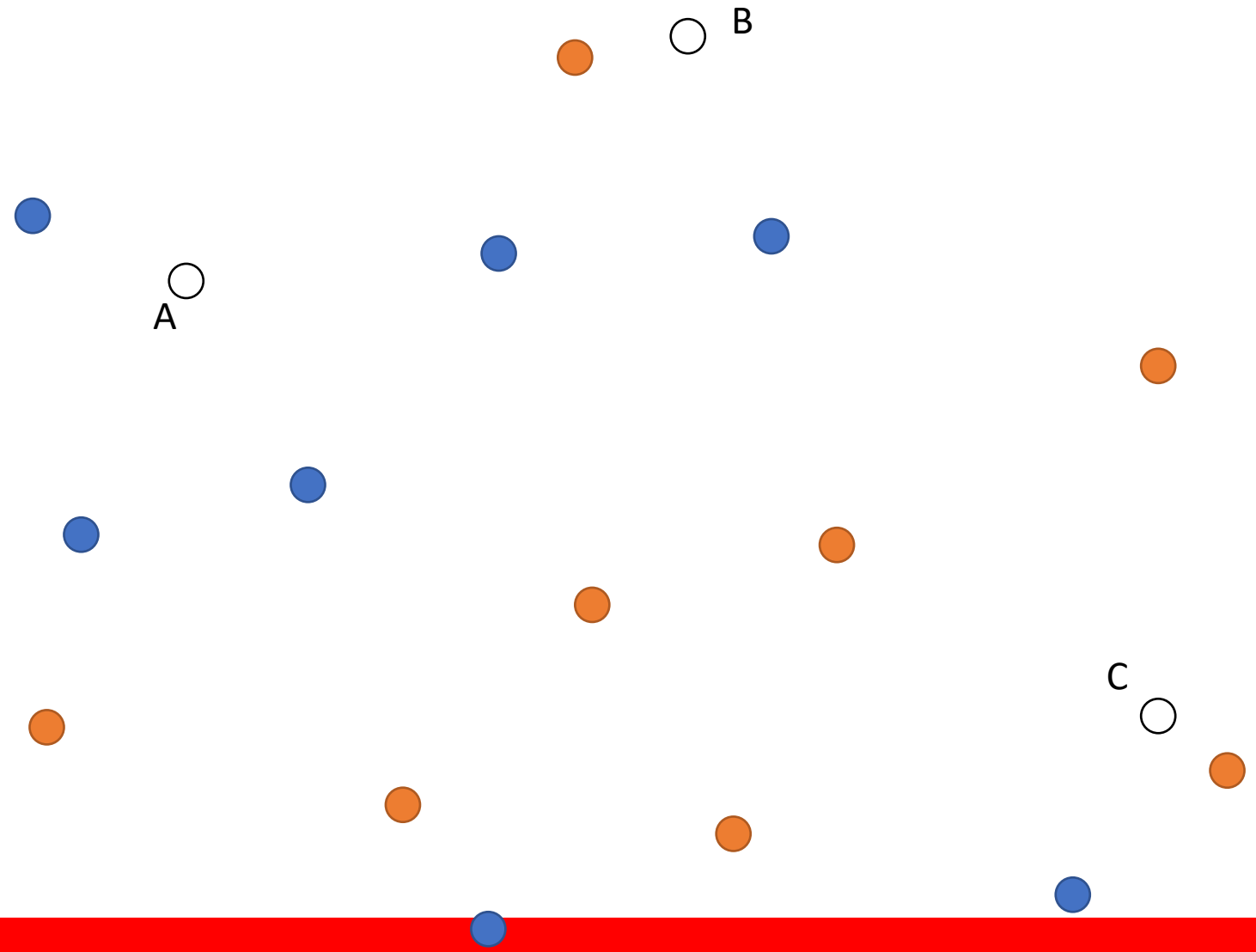
Nearest Neighbor Classification



How would you color the blank circles?



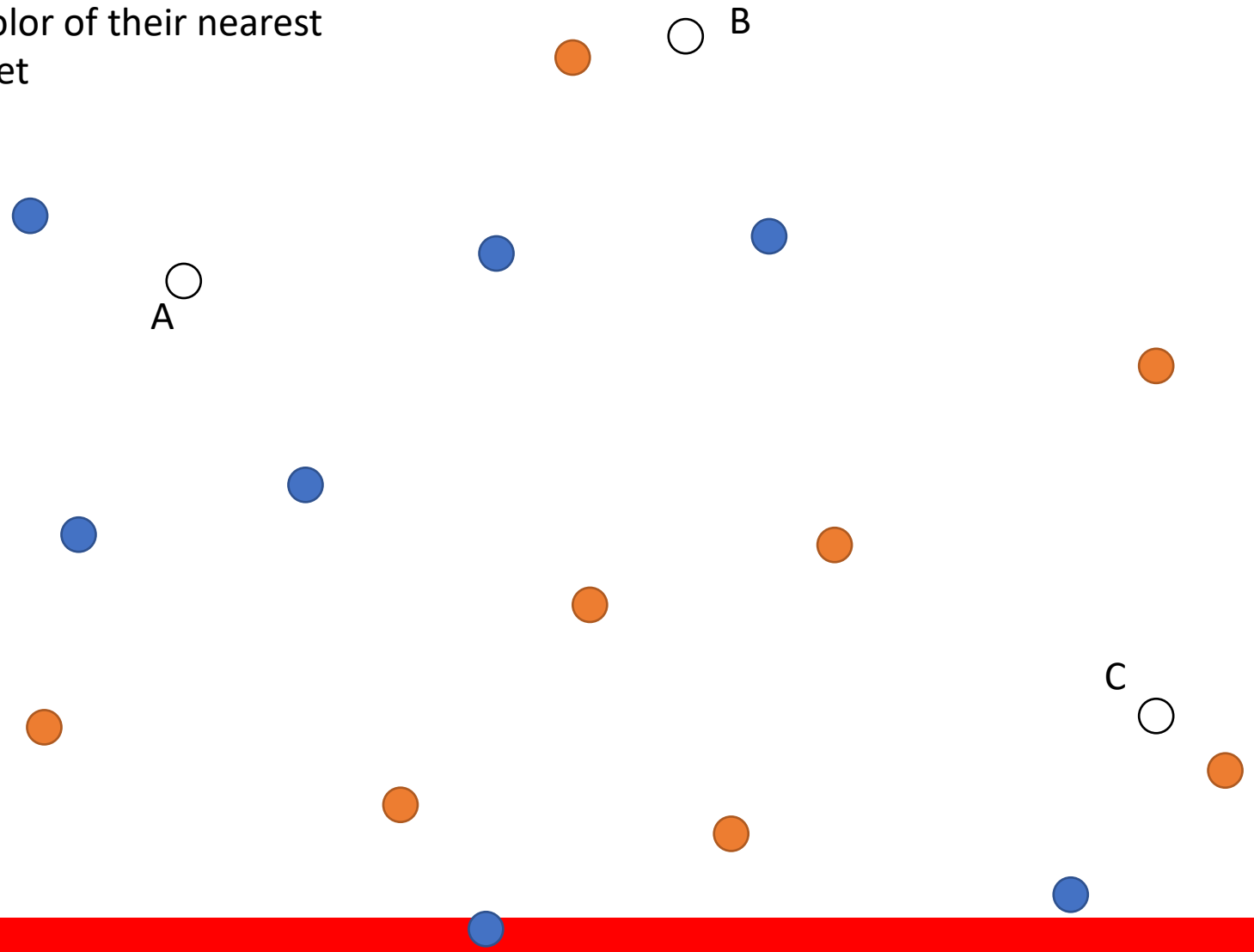
How would you color the blank circles?

If we based it on the color of their nearest neighbors, we would get

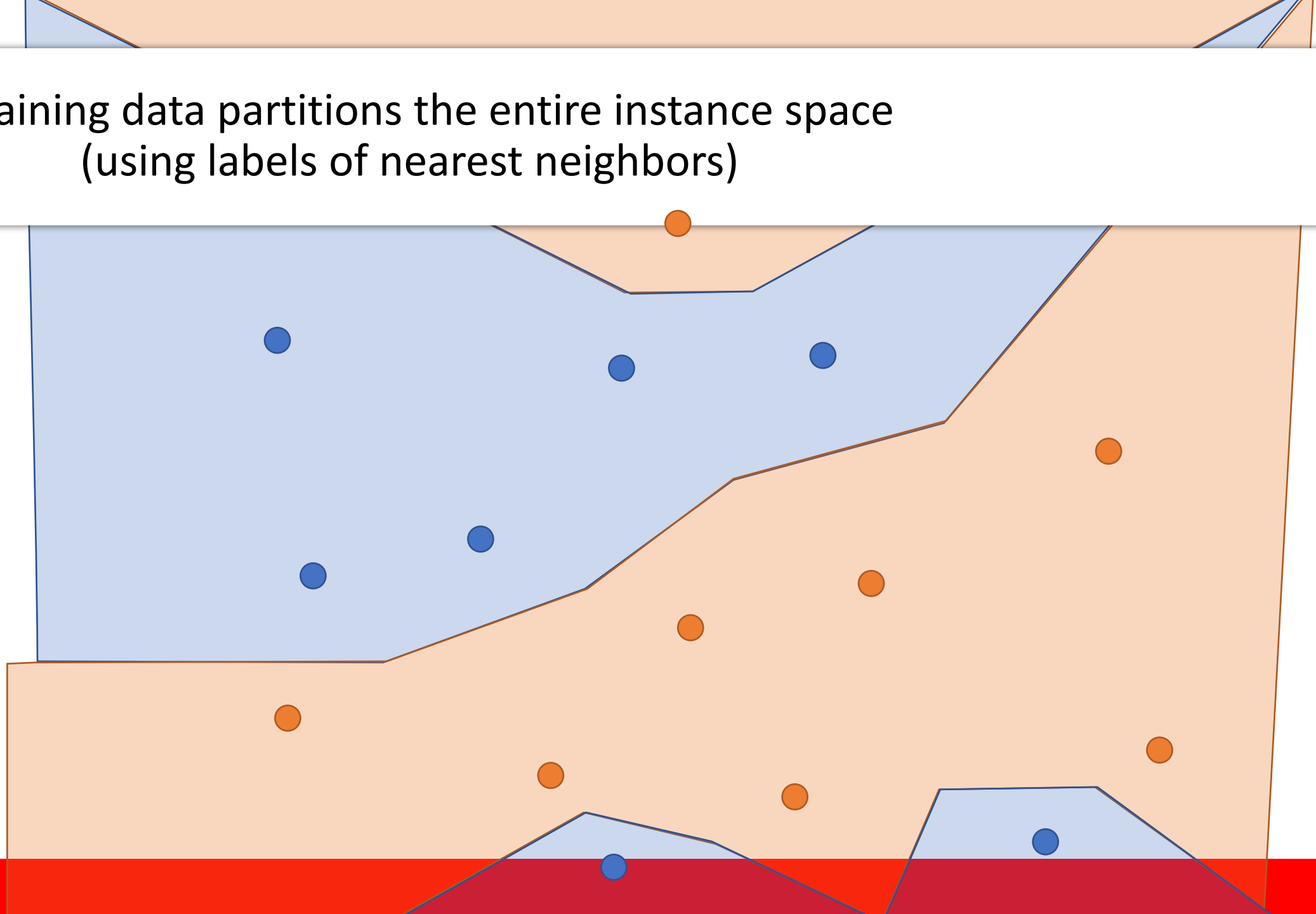
A: Blue

B: Orange

C: Orange



Training data partitions the entire instance space
(using labels of nearest neighbors)



Nearest Neighbors: The basic version

- Training examples are vectors $\mathbf{x}_i$ associated with a label y_i
- Learning: Just store all the training examples
- Prediction for a new example $\mathbf{x}$
  - Find the training example $\mathbf{x}_i$ that is *closest* to $\mathbf{x}$
  - Predict the label of $\mathbf{x}$ to the label y_i associated with $\mathbf{x}_i$

K-Nearest Neighbors

- Training examples are vectors $\mathbf{x}_i$ associated with a label y_i

- Learning: Just store all the training examples

- Prediction for a new example $\mathbf{x}$
  - Find the k *closest* training examples to $\mathbf{x}$
  - Construct the label of $\mathbf{x}$ using these k points. *How?*
  - For classification: ?

K-Nearest Neighbors

- Training examples are vectors $\mathbf{x}_i$ associated with a label y_i
- Learning: Just store all the training examples
- Prediction for a new example $\mathbf{x}$
  - Find the k *closest* training examples to $\mathbf{x}$
  - Construct the label of $\mathbf{x}$ using these k points. *How?*
  - For classification: Every neighbor votes on the label. Predict the most frequent label among the neighbors.
  - For regression: ?

K-Nearest Neighbors

- Training examples are vectors $\mathbf{x}_i$ associated with a label y_i

- Learning: Just store all the training examples

- Prediction for a new example $\mathbf{x}$
  - Find the k *closest* training examples to $\mathbf{x}$
  - Construct the label of $\mathbf{x}$ using these k points. *How?*
  - For classification: Every neighbor votes on the label. Predict the most frequent label among the neighbors.
  - For regression: Predict the mean value

Distance between instances

- In general, a good place to inject knowledge about the domain
- Behavior of this approach can depend on this
- How do we measure distances between instances?

Distance between instances

Numeric features, represented as n dimensional vectors

- Euclidean distance

$$||\mathbf{x}_1 - \mathbf{x}_2||_2 = \sqrt{\sum_{i=1}^{n} (\mathbf{x}_{1,i} - \mathbf{x}_{2,i})^2}$$

- Manhattan distance

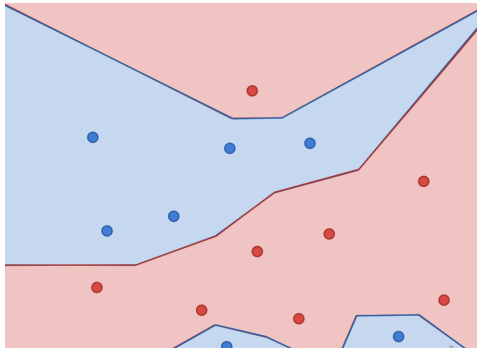
$$||\mathbf{x}_1 - \mathbf{x}_2||_1 = \sum_{i=1}^{n} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|$$

- L_p-norm
  - Euclidean = L_2
  - Manhattan = L_1
  - Exercise: What is L_∞?

$$||\mathbf{x}_1 - \mathbf{x}_2||_p = \left(\sum_{i=1}^{n} |\mathbf{x}_{1,i} - \mathbf{x}_{2,i}|^p \right)^{\frac{1}{p}}$$

Advantages

- Training is *very fast*
  - Just adding labeled instances to a list
  - More complex indexing methods can be used, which slow down learning slightly to make prediction faster
- Can learn very complex functions

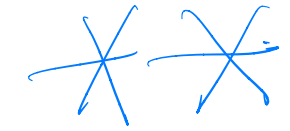


- We always have the training data
  - For other learning algorithms, after training, we don't store the data anymore. What if we want to do something with it later…

Disadvantages

- Needs a lot of storage
  - Is this really a problem now?

- Prediction can be slow!
  - Naïvely: $O(dN)$ for N training examples in d dimensions
  - More data will make it slower
  - Compare to other classifiers, where prediction is very fast

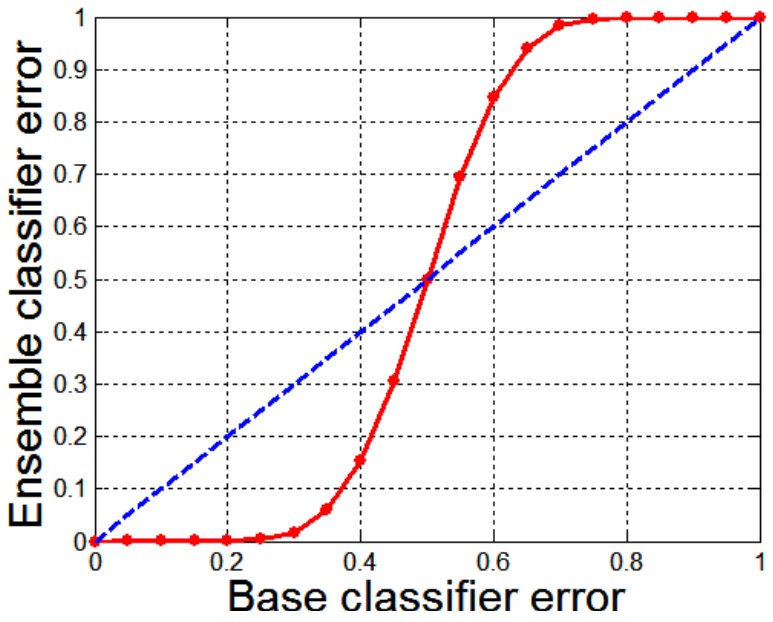
Ensemble Classifiers 

Ensemble Classifiers

- Given a training set D = { (x_i, y_i) } (i=1,2,…,N)
  - For binary classification, assume $y_i \in$ {-1, +1}
- Construct an ensemble of classification models $f_1(x), f_2(x), \ldots, f_k(x)$ from the training set
- Predict the class of an example x by combining the predictions made by the classifier ensemble, i.e., $f(x) = g[\ \Sigma_j\ \alpha_j f_j(x)\ ]$
  - where g is a function to combine the individual predictions (e.g., sign function)

Why Ensemble Methods work?

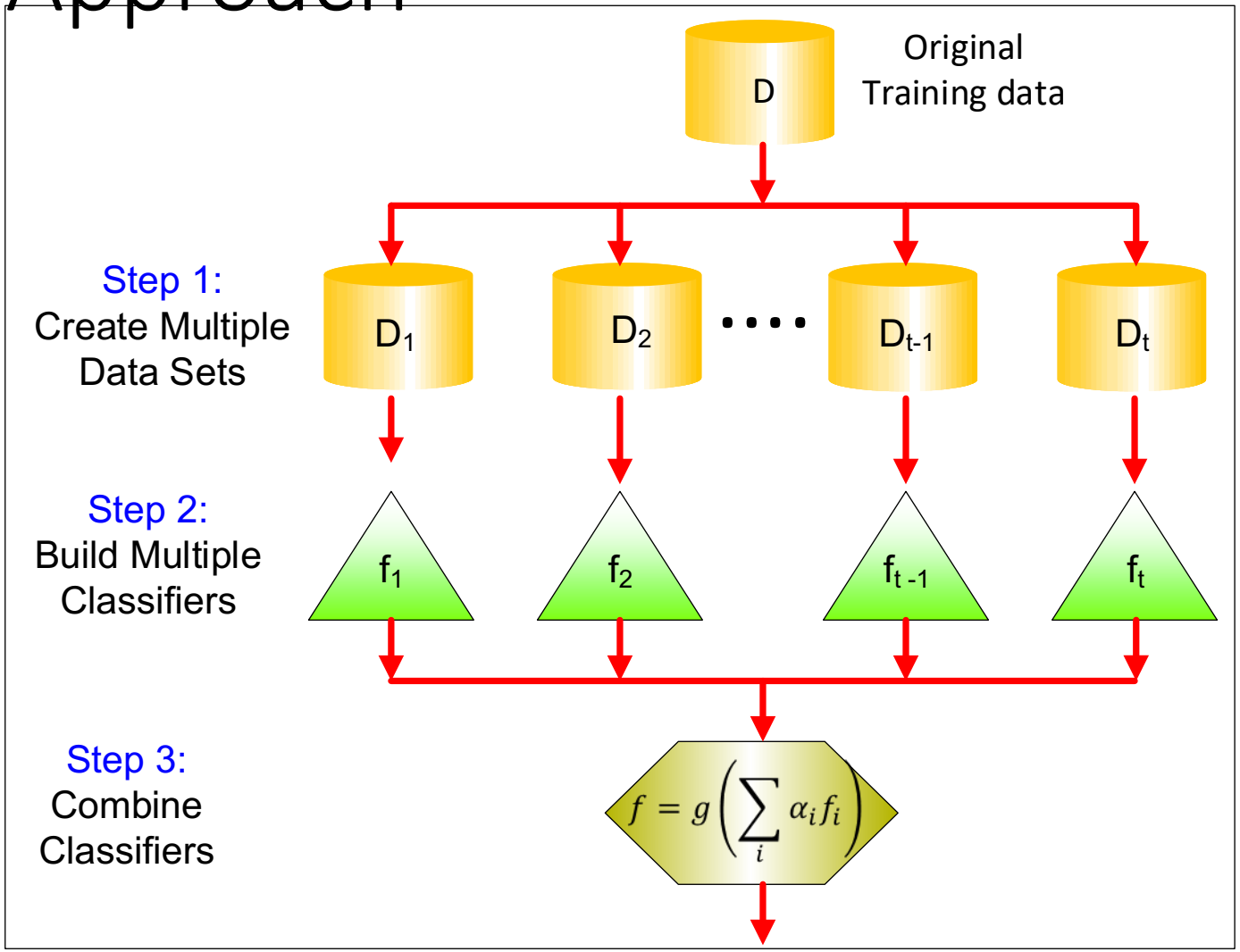
- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume the errors made by the classifiers are independent
  - Probability that the ensemb classifier makes a wrong prediction:



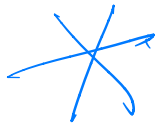
X: number of classifiers that made a wrong prediction

$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06 \quad < 0.35$$

General Approach



Bagging



Bagging

- Use bootstrap sampling (i.e., sampling with replacement) to create the multiple training sets
- Train a classifier on each bootstrap sample
  - Each sample has a probability $1 - (1 - 1/n)^n$ of being selected

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	$\rightarrow f_1(x)$
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2	$\rightarrow f_2(x)$
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7	$\rightarrow f_3(x)$

- Prediction step (for Binary Classification):

$$f(x) = \text{sign}[\ \Sigma_j f_j(x)\ ] = \begin{cases} 1 & \text{if } \Sigma_j f_j(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

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: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

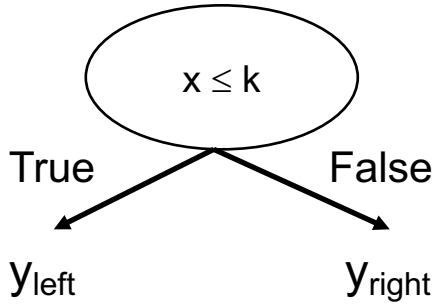
Bagging Example

- 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

- Base classifier is a decision stump (a decision tree with a single node)
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging Example

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

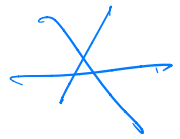
Bagging Example

- Suppose the test set is the same as training set
- Use majority vote to determine class of ensemble classifier: y_i = sign[$f(x_i)$] = sign[$\Sigma_j f_j(x_i)$]

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
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Boosting



Boosting

- Similar to bagging:
  - Ensemble is created by resampling the training data
- Resampling procedure is different from bagging
  - In bagging, each example has equal probability of being selected to form the training set (at every bagging round)
    - In boosting, each training example is assigned a weight w_i (initially, all examples have equal weights; but the weights are changed after each boosting round)
    - Boosting adaptively changes the distribution of training data by focusing more on examples that are hard to classify
  - Final classifier: $f^*(x) = \Sigma_j \alpha_j f_j(x)$
    - In boosting, α_j measures the "importance" of model $f_j(x)$
    - In bagging, $\alpha_j = 1$ for all j's

Boosting

- After each boosting round
  - Examples that are wrongly classified will have their weights increased
  - Examples 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 so that it is more likely to be chosen again in subsequent rounds

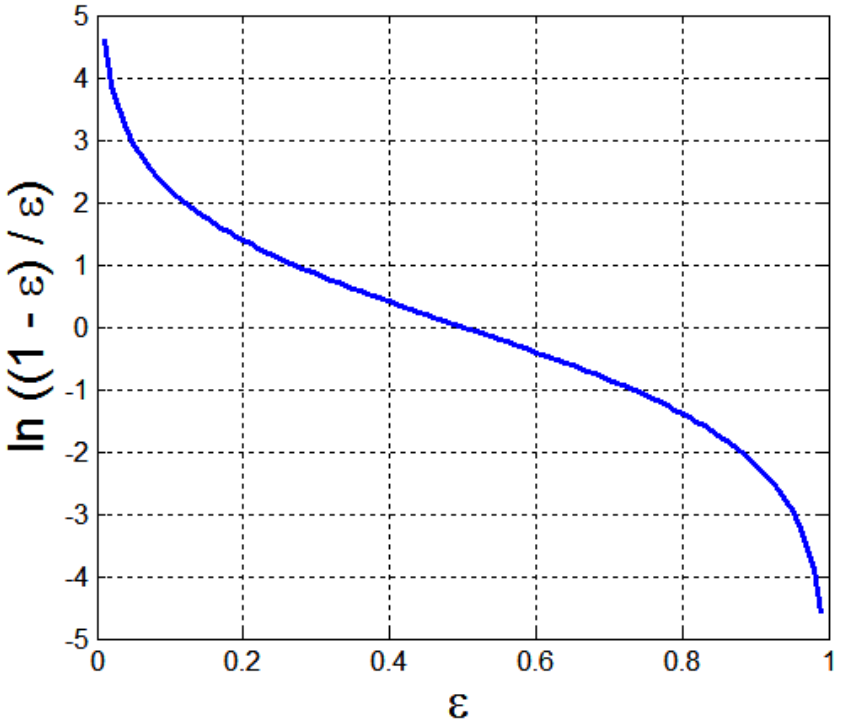
AdaBoost

- Let f_1, f_2, …, f_T be the base models
- Each training example has a weight w_j
- Error rate for model f_t:

$$\varepsilon_t = \sum_{j=1}^{N} w_j \delta\left(f_t(x_j) \neq y_j\right)$$

- Importance of model f_t

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



AdaBoost Algorithm

1. Assign uniform weights to all the data points
2. Repeat until maximum iteration is reached
   - Create a training set D_t by sampling the data points according to their weights
   - Train a new model f_t based on D_t
   - Apply f_t to the training set and calculate its error rate e_t and importance factor α_t
   - Update the weights for all data points
     - Increase the weight of misclassified data points
     - Decrease the weight of correctly classified data points

Weight Update in AdaBoost

- Weight update:

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

$$= \frac{w_i^{(j)}}{Z_j} \exp^{-\alpha_j y_i f_j(x_i)}$$

$$\text{where } Z_j \text{ is the normalization factor}$$

- 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

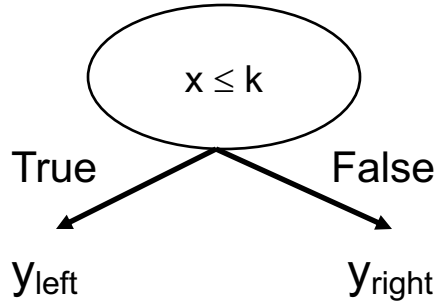
AdaBoost Example

- 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: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



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

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

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

- 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

AdaBoost Example

- 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

- Classification

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

Summary of Ensemble Methods

- Very powerful
  - Generally outperforms most of the single classifier methods
- More expensive to train
  - Bagging is easier to parallelize
  - Boosting is harder due to serial dependencies between models (f_t depends on f_{t-1})
- Reduce bias and variance of classifiers

Bias and Variance of a Classifier

- Bias is high if classifier is too simple or makes a strong assumption
  - Example: linear classifiers
- Variance is high if model is sensitive to the choice of the training set
  - Example: nearest-neighbor classifiers

Practical Guide for Users

- What to do if your classification results are poor?
  1. Check for model overfitting/underfitting
     - If overfit:
       - Do more careful model selection to reduce model complexity
       - Eliminate noisy/ correlated features
       - Increase training set size by collecting more labeled examples
     - If underfit:
       - Add more discriminative features
       - Use more flexible models (e.g., nonlinear instead of linear models)

Practical Guide for Users

- What to do if your classification results are poor?
  2. Check data quality and algorithm execution
     - Are there a lot of mislabeled examples?
     - Are there lots of missing values?
     - Are the features correlated/irrelevant/noisy?
     - Has the learning algorithm converged?
     - Is the optimization procedure used suitable for the data? Is the error function appropriate?
     - Don’t just rely on the default parameters of algorithm

Practical Guide for Users

- What to do if your classification results are poor?
  3. Check for model bias and variance
     - High bias: train error is high; test error ≈ train error
     - High variance: train error varies significantly when you apply cross-validation or repeated holdout
     - In both cases, ensemble methods generally help