

COMP 432 Machine Learning

Bagging and Boosting

Computer Science & Software Engineering
Concordia University, Fall 2024



Summary of the last episode....

What we have seen in the **last lecture**:

- Decision Trees
- Random Forest

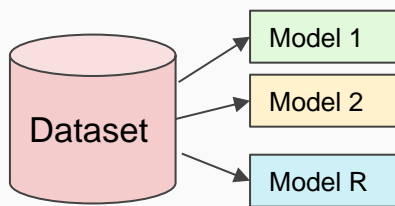
What we are going to **learn today**:

- Bagging
- Boosting
- k-nearest neighbor
- Naive Bayes

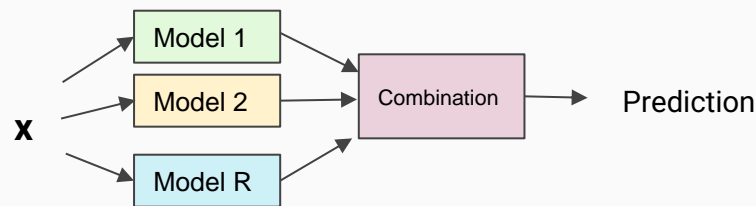
Bagging and Boosting

Bagging

- **Ensemble techniques:** all the techniques that create a new machine learning model by combining multiple models.
- We already have seen some examples of ensemble techniques: **bagging** and **dropout**.
- In **bagging** (Bootstrap Aggregating), we train $\{1, \dots, R\}$ models **independently**. At test/inference time, we **combine** their **predictions**.

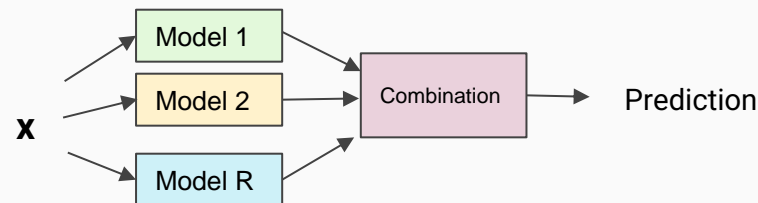
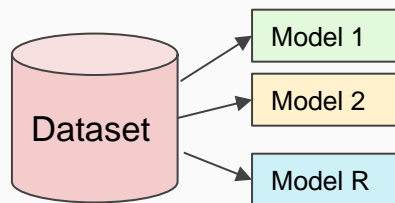


1) Train R models independently.



2) Combine their predictions at test/inference time.

Bagging

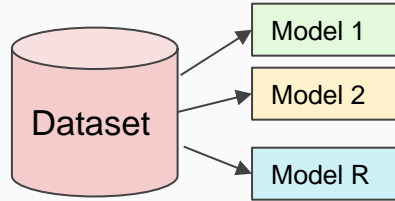


1) Train R models independently.

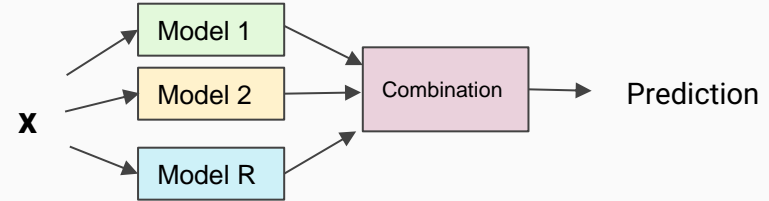
2) Combine their predictions at test/inference time.

- For **classification problems**, we can choose the class that takes the **majority of votes**.
- For **regression problems**, we can **average the predictions**.
- This is the technique used in **random forest**.
- It works well for many other machine learning models as well, including **neural networks**.

Bagging



1) Train R models independently.



2) Combine their predictions at test/inference time.

- Bagging is a powerful **regularization technique** that aims to reduce **generalization error**.
- The “trick” to make bagging work well is to make sure the models provide **uncorrelated errors**.

- *How can we achieve that?*



Bagging

- There are different ways to create a set of “**diverse**” models:
 1. Train the models with **different data**.
 2. Train the models with **bootstrapped datasets** (as done in random forest).
 3. Train models with different **hyperparameters**.
- If the models are “diverse”, we hope that they will provide errors that are as **uncorrelated** as possible.
- Unfortunately, this does not always hold in practice.

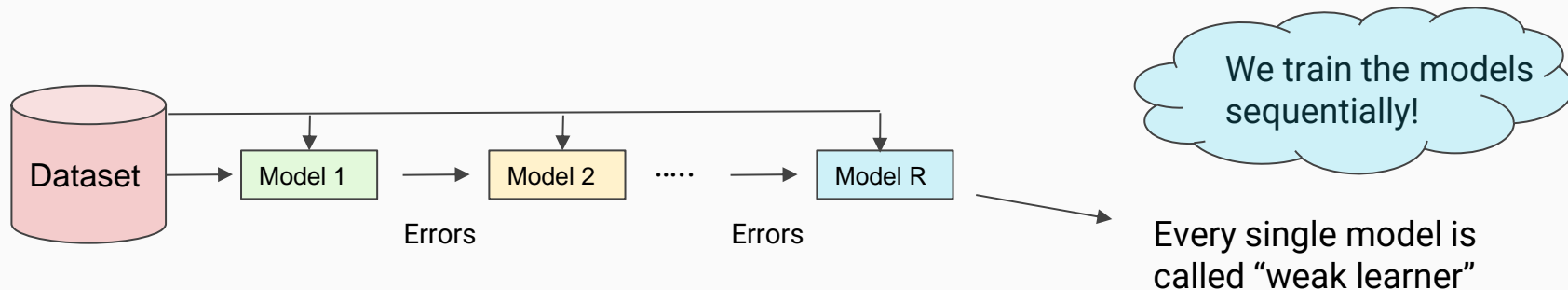


What about if we train the models to explicitly make uncorrelated errors?

Boosting

- The general idea of **boosting** is the following:

- We train the $\{1, \dots, R\}$ models **sequentially**.
- The r^{th} model is trained to fix the errors that models $\{1, \dots, r-1\}$ are currently making on the training set.



- This way we “boost” the performance and train models that provide “different” errors.
- At test/inference time, we can just combine all these models with a weight that depends on “how good” the model is.

AdaBoost

- The most popular boosting algorithm is called AdaBoost (Adaptive Boosting).
- With AdaBoost, we introduce a **weight** for each training sample:

$$\{w_1, \dots, w_N\}$$

The weight is **large** if the previous model has **performed poorly** on this data sample.

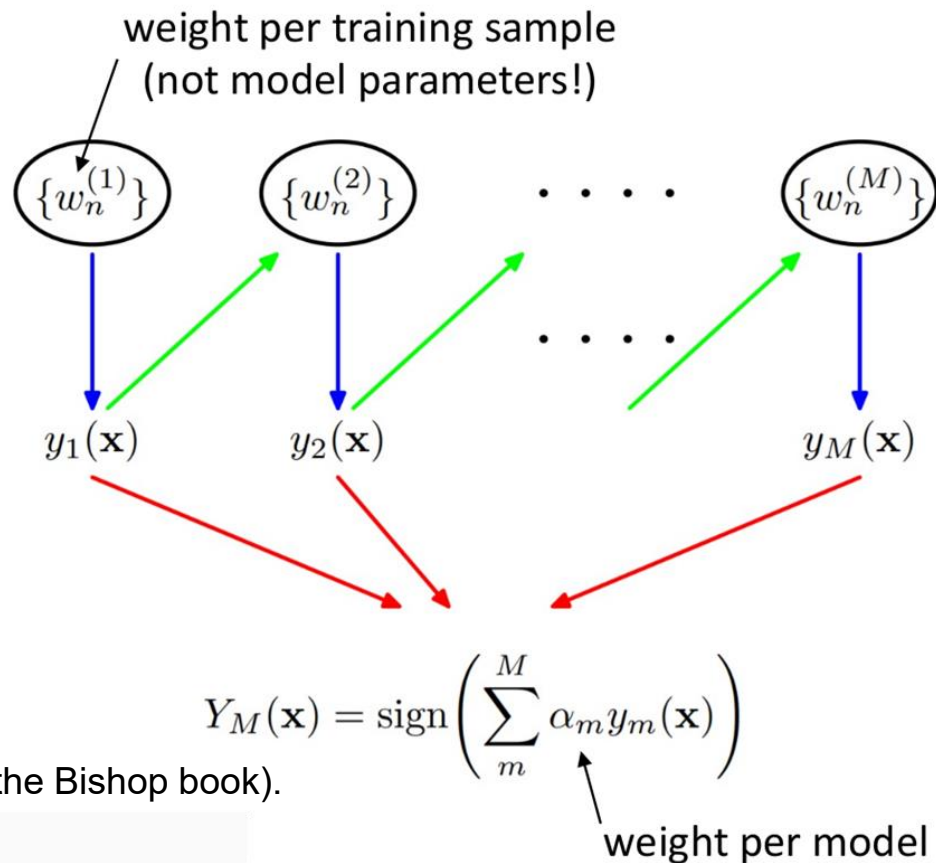
The weight is **small** if the previous model has **performed well** on this data sample.

- This way, the model at the r^{th} step knows which samples to “focus” on.

AdaBoost

Figure 14.1

Schematic illustration of the boosting framework. Each base classifier $y_m(\mathbf{x})$ is trained on a weighted form of the training set (blue arrows) in which the weights $w_n^{(m)}$ depend on the performance of the previous base classifier $y_{m-1}(\mathbf{x})$ (green arrows). Once all base classifiers have been trained, they are combined to give the final classifier $Y_M(\mathbf{x})$ (red arrows).



This depicts “boosting” a binary classifier (from the Bishop book).
Boosting can be generalized to multi-class too.

AdaBoost algorithm (binary)

Dataset

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\} \quad t_i \in \{-1, +1\}$$

Initialize $w_i = \frac{1}{N}$ for $i=1, \dots, N$

for $r= 1, \dots, R$:

Train y_r to minimize $\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$

Compute $\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$

Compute $w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$

Update $w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$

→ In the beginning, all samples are equally weighted.

→ We train R models in sequence.

→ This term computes the average error (weighted by the weights w).

$$[P] = \begin{cases} 1 & \text{if } P \text{ true} \\ 0 & \text{if } P \text{ false} \end{cases}$$

AdaBoost algorithm (binary)

Dataset

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\} \quad t_i \in \{-1, +1\}$$

Initialize $w_i = \frac{1}{N}$ for $i=1, \dots, N$

for $r= 1, \dots, R$:

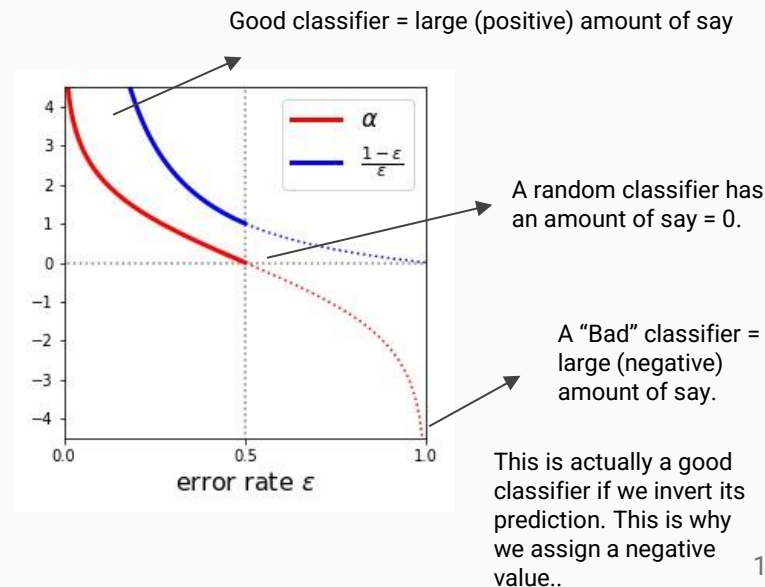
Train y_r to minimize $\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$

Compute $\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$

Compute $w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$

Update $w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$

- This term is called “amount of say” and tells how good is a classifier on average.



AdaBoost algorithm (binary)

Dataset

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\} \quad t_i \in \{-1, +1\}$$

Initialize $w_i = \frac{1}{N}$ for $i=1, \dots, N$

for $r= 1, \dots, R$:

Train y_r to minimize $\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$

Compute $\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$

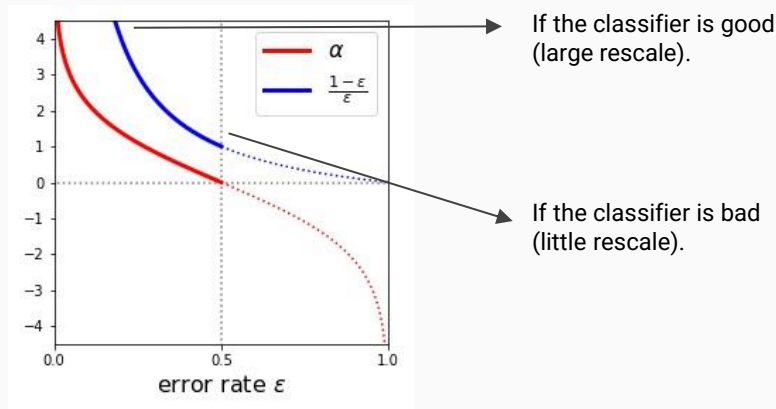
Compute $w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$

Update $w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$

For each training sample, we compute the new weights:

- The new weights are :

Scaled by $\frac{1 - \epsilon}{\epsilon}$ if the sample is misclassified.



Unchanged if the sample is classified correctly.

AdaBoost algorithm (binary)

Dataset

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\} \quad t_i \in \{-1, +1\}$$

Initialize $w_i = \frac{1}{N}$ for $i=1, \dots, N$


for $r= 1, \dots, R$:

Train y_r to minimize $\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$

Compute $\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$

Compute $w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$

Update $w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$

 We normalize the new weights such that their sum is 1.

AdaBoost algorithm (binary)

Dataset

$$D = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \dots, (\mathbf{x}_N, t_N)\} \quad t_i \in \{-1, +1\}$$

Initialize $w_i = \frac{1}{N}$ for $i=1, \dots, N$

for $r= 1, \dots, R$:


Train y_r to minimize $\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$

Compute $\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$

Compute $w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$

Update $w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$

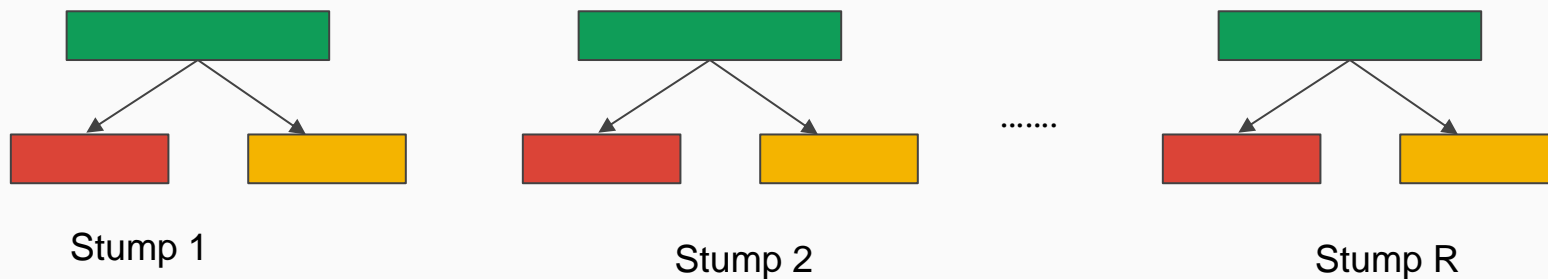
- We repeat this for all the R models.
- At test/inference time, we can perform a prediction in this way:

$$y(\mathbf{x}) = \text{sign}\left(\sum_{r=1}^R \alpha_r y_r(\mathbf{x})\right)$$


The output of each model is weighted by the amount of say.

AdaBoost algorithm (binary)

- AdaBoost can be applied to many machine learning algorithms (including neural networks).
- It is often applied to decision trees with $max_depth=1$ (one split) as the “weak learner” (base classifiers). This is very popular, and the default in sklearn.

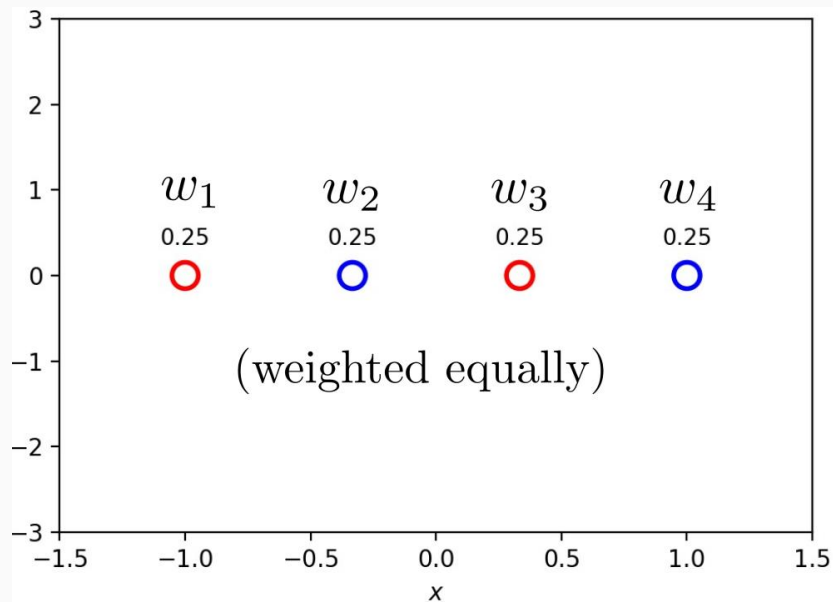


- These base classifiers are called a “**decision stump**”.

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$



Let's find the classifier that minimizes ϵ :

Model (stump): <i>Vote red if</i>	Data with Errors	ϵ
$x < -0.66$	{3}	0.25
$x < 0.0$	{2,3}	0.50
$x < 0.66$	{2}	0.25
$x > -0.66$	{1,2,4}	0.75
$x > 0.0$	{1,4}	0.50
$x > 0.66$	{1,3,4}	0.75

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right) \quad \alpha_1 = \ln\left(\frac{1 - 0.25}{0.25}\right) = 1.10$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$

Best model (min ϵ)

Model (stump)	Data with Errors	ϵ
$x < -0.66$	{3}	0.25

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$$

$$\alpha_1 = \ln\left(\frac{1 - 0.25}{0.25}\right) = 1.10$$

Update weights

Old Weight	New Weights	New Weights (Normalized)
$w_1=0.25$	0.25	0.17
$w_2=0.25$	0.25	0.17
$w_3=0.25$	$0.25 * 3 = 0.75$	0.50
$w_4=0.25$	0.25	0.17

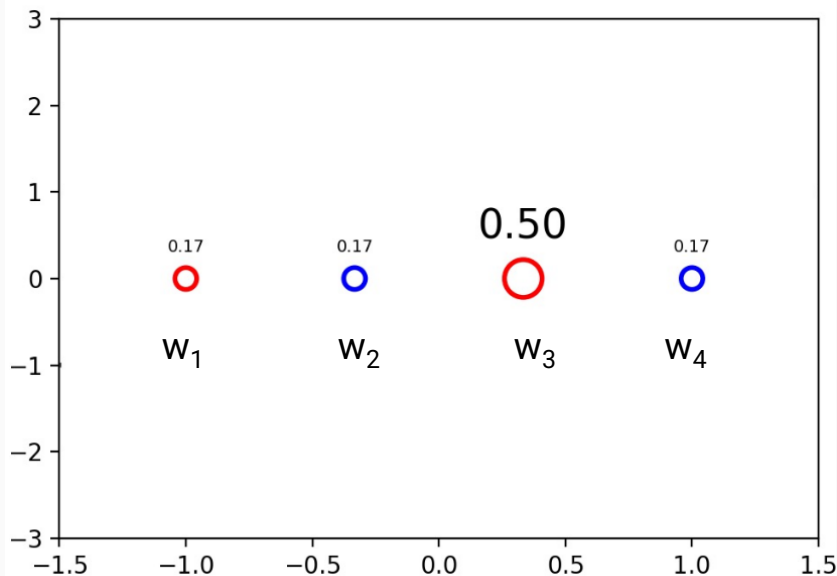
$$w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$$

$$w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$



Let's find another classifier that minimizes ϵ :

Model (stump) <i>Vote red if</i>	Data with Errors	ϵ
$x < -0.66$	{3}	0.50
$x < 0.0$	{2,3}	0.67
$x < 0.66$	{2}	0.17
$x > -0.66$	{1,2,4}	0.51
$x > 0.0$	{1,4}	0.34
$x > 0.66$	{1,3,4}	0.84

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right) \quad \alpha_r = \ln\left(\frac{1 - 0.17}{0.17}\right) = 1.59$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$

Best model (min ϵ)

Model (stump)	Data with Errors	ϵ
$x < 0.66$	{2}	0.17

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$$

$$\alpha_r = \ln\left(\frac{1 - 0.17}{0.17}\right) = 1.59$$

Update weights

Old Weight	New Weights	New Weights (Normalized)
w1=0.17	0.17	0.10
w2=0.17	0.83	0.50
w3=0.50	0.50	0.30
w4=0.17	0.17	0.10

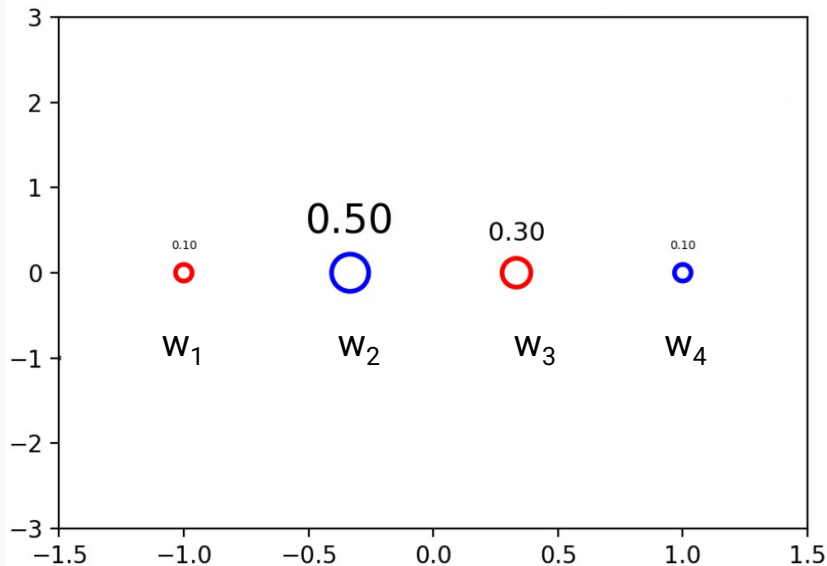
$$w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$$

$$w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$



Let find another classifier that minimizes ϵ :

Model (stump): <i>Vote red if</i>	Data with Errors	ϵ
$x < -0.66$	{3}	0.30
$x < 0.0$	{2,3}	0.80
$x < 0.66$	{2}	0.50
$x > -0.66$	{1,2,4}	0.70
$x > 0.0$	{1, 4}	0.20
$x > 0.66$	{1,3,4}	0.50

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right) \quad \alpha_r = \ln\left(\frac{1 - 0.20}{0.20}\right) = 1.39$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$

Best model (min ϵ)

Model (stump)	Data with Errors	ϵ
$x > 0.0$	$\{1, 4\}$	0.20

Update weights

Old Weight	New Weights	New Weights (Normalized)
$w_1=0.10$	0.40	0.25
$w_2=0.50$	0.50	0.31
$w_3=0.30$	0.30	0.19
$w_4=0.10$	0.40	0.25

$$\alpha_r = \ln\left(\frac{1 - \epsilon}{\epsilon}\right)$$

$$\alpha_r = \ln\left(\frac{1 - 0.20}{0.20}\right) = 1.39$$

$$w'_i = w_i e^{\alpha_r [y_r(\mathbf{x}_i) \neq t_i]}$$

$$w_i \leftarrow \frac{w'_i}{\sum_{i=1}^N w'_i}$$

AdaBoost (Example)

$$\mathcal{D} = \{(-1, -1), (-\frac{1}{3}, +1), (\frac{1}{3}, -1), (1, +1)\}$$

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i]$$

We trained the following models:

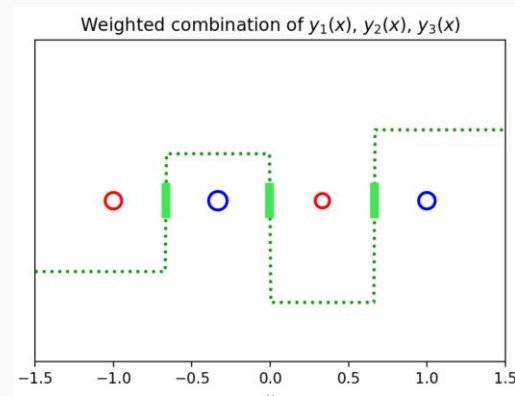
Prediction Model:

$$y(\mathbf{x}) = \text{sign}\left(1.10 \cdot y_1(\mathbf{x}) + 1.58 \cdot y_2(\mathbf{x}) + 1.38 \cdot y_3(\mathbf{x})\right)$$

Model (stump): <i>Vote red if</i>	α
$x < -0.66$	1.10
$x < 0.66$	1.59
$x > 0.0$	1.39

New inputs for testing

x	y_1	y_2	y_3	y
$x = -1.5$	-1	-1	+1	-1 (red)
$x = -0.1$	+1	-1	+1	+1 (blue)
$x=1.5$	+1	+1	-1	+1 (blue)




AdaBoost (Final Remarks)

- In some cases, you don't want (or you cannot) find the best system according to the loss ϵ :

$$\epsilon = \sum_{i=1}^N w_i [y_r(\mathbf{x}_i) \neq t_i] \longrightarrow$$

This is not a “soft” objective and it is not suitable for some models such as neural networks.

- One can reformulate the problem by using other metrics such as **weighted Gini index** for decision trees, or **weighted cross-entropy** for neural networks.
- When creating a weighted loss is not possible, one can sample a **bootstrapped dataset** using the weights as sampling probability.
- This way, we create a dataset that contains multiple times the misclassified data points (thus giving more “importance” to them implicitly).
- See  **StatQuest** for an example of boosting with a bootstrapped dataset.

Other Supervised Machine Learning Algorithms

Supervised Machine Learning

- We have already seen many **supervised machine learning algorithms**:

Linear models (linear regression, logistic regression, multiclass logistic regression)

Neural Networks (MLPs, CNNs, RNNs)

Support Vector Machines (linear and non-linear)

Decision Trees, Random Forest, Bagging,
Boosting.

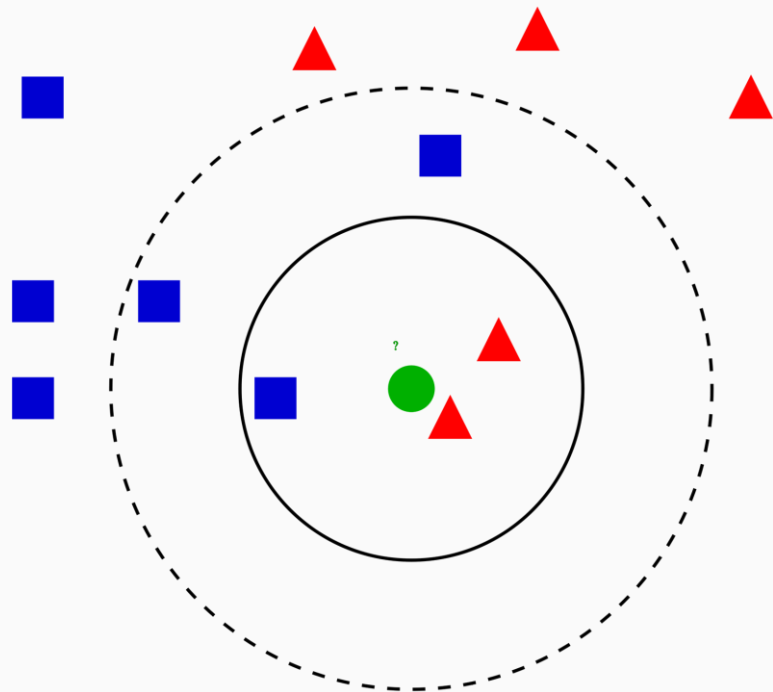
- There are many other algorithms proposed in the literature.
- We conclude the part on supervised learning with a couple of very simple algorithms:

K-nearest Neighbor

Naive Bayes

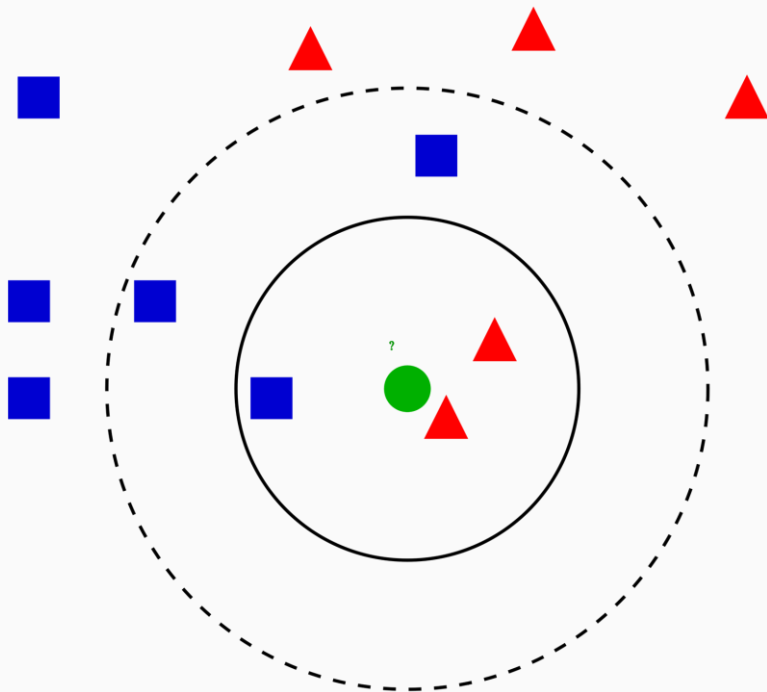
K-nearest Neighbor

K-nearest Neighbor



- This is a very simple algorithm that just **stores the training samples**.
- At test time, we just consider the K training samples that are closer to the current input \mathbf{x} .
- If we are solving a **classification problem**, the new sample is assigned to the class most common among its k nearest neighbors.
- If we are solving a **regression problem**, the new prediction is the average of the values of k nearest neighbors.

K-nearest Neighbor



- K is a **hyperparameter** of the model.
- Different measures of distance can be used (e.g, Euclidean, Cosine, Hamming, Manhattan, etc).
- A natural extension is **weighted nearest neighbor**, where each training sample influences the prediction using a weight that depends on the distance with the current input.

K-nearest Neighbor

Advantages



Simple algorithm with easy Implementation.



No training phase.



The classifier immediately adapts as we collect new training data.



Only one hyperparameter to tune.

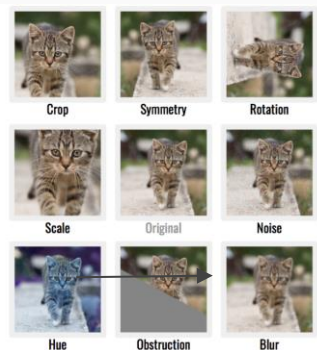
Disadvantages



Poor generalization.

If we have little training data → Overfitting

Even if we have a lot of training data is still hard to generalize (especially with high-dimensional data).



All these inputs are “semantically” the same but they are very far away in the input space (pixels).



Inference not efficient →

We need to loop over all the training data.



We need to store all the training data.

Naive Bayes

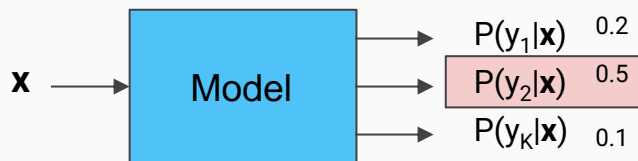
Naive Bayes

- From the probabilistic point of view, it is desirable to have a machine learning model that classifies our inputs based on the following rule:

$$k^* = \underset{k}{\operatorname{argmax}} P(y_k \mid x_1, \dots, x_D)$$

This term is called **posterior probability** and represents the probability that a certain input belongs to class k .

The posterior probabilities are the quantities returned by a classifier trained with neural network.



We estimate this probability for each class and we vote for the class with the highest posterior probability (maximum a posteriori decision rule).

Naive Bayes

- According to Bayes' Rule:

$$P(y_k \mid x_1, \dots, x_D) = \frac{P(y_k)P(x_1, \dots, x_D \mid y_k)}{P(x_1, \dots, x_D)}$$

Diagram illustrating the components of Bayes' Rule for Naive Bayes:

- Prior Probability (Class Probability):** $P(y_k)$
- Likelihood:** $P(x_1, \dots, x_D \mid y_k)$
- Posterior Probability:** $P(y_k \mid x_1, \dots, x_D)$
- Probability of observing a certain feature vector:** $P(x_1, \dots, x_D)$ (This term is the same for all the classes (does not depend on k).)

$$k^* = \operatorname{argmax}_k P(y_k \mid x_1, \dots, x_D) = \operatorname{argmax}_k P(y_k)P(x_1, \dots, x_D \mid y_k)$$

Naive Bayes

Prior Probability

Likelihood

$$k^* = \operatorname{argmax}_k P(y_k) P(x_1, \dots, x_D \mid y_k)$$

Using the **chain rule of probability**, we can write the likelihood in this way:

$$P(x_1, \dots, x_D \mid y_k) = P(x_1 \mid x_2, \dots, x_D, y_k) \cdot P(x_2 \mid x_3, \dots, x_D, y_k) \cdot \dots \cdot P(x_{D-1} \mid x_D, y_k) \cdot P(x_D \mid y_k)$$



This term is complex to model. However, if we assume that all the features are **independent** we have:


$$P(x_i \mid x_{i+1}, \dots, y_k) = P(x_i \mid y_k)$$

$$P(x_1, \dots, x_D \mid y_k) = \prod_{i=1}^D P(x_i \mid y_k)$$

→ We can simply model each feature independently.

Naive Bayes

Naive Bayes

$$k^* = \operatorname{argmax}_k P(y_k) \prod_{i=1}^D P(x_i | y_k)$$


This term can be simply estimated by counting:

$$P(y_k) \approx \frac{N_k}{N}$$

N_k → Training samples belong to class k .
 N → Total number of training samples.

This term can be estimated efficiently as well through a (1D) probability density estimation (that will be addressed in the last lecture).

If the features are discrete, we can estimate this term by counting as well (see the following example).

Naive Bayes (Example)

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

1. Estimate the **prior probabilities**:

$$P(\text{Yes}) = \frac{9}{14} = 0.64$$

$$P(\text{No}) = \frac{5}{14} = 0.36$$

2. Estimate the **likelihoods**:

$$P(\text{Sunny}|\text{Yes}) = 3/9$$

$$P(\text{Sunny}|\text{No}) = 2/5$$

$$P(\text{Overcast}|\text{Yes}) = 4/9$$

$$P(\text{Overcast}|\text{No}) = 0/5$$

$$P(\text{Rainy}|\text{Yes}) = 2/9$$

$$P(\text{Rainy}|\text{No}) = 3/5$$



Outlook



Repeat for all
the features

Naive Bayes (Example)

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

New data Sample to classify:

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

Frequency Table

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	$3/9$	$2/5$
	Overcast	$4/9$	$0/5$
	Rainy	$2/9$	$3/5$

		Play Golf	
		Yes	No
Humidity	High	$3/9$	$4/5$
	Normal	$6/9$	$1/5$

		Play Golf	
		Yes	No
Temp.	Hot	$2/9$	$2/5$
	Mild	$4/9$	$2/5$
	Cool	$3/9$	$1/5$

		Play Golf	
		Yes	No
Windy	False	$6/9$	$2/5$
	True	$3/9$	$3/5$

Naive Bayes (Example)

Frequency Table

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3



Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	3/9	2/5
	Overcast	4/9	0/5
	Rainy	2/9	3/5

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1



		Play Golf	
		Yes	No
Humidity	High	3/9	4/5
	Normal	6/9	1/5

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1



		Play Golf	
		Yes	No
Temp.	Hot	2/9	2/5
	Mild	4/9	2/5
	Cool	3/9	1/5

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3



		Play Golf	
		Yes	No
Windy	False	6/9	2/5
	True	3/9	3/5

New data sample to classify:

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

$$P(Yes | X) = P(Rainy | Yes) \times P(Cool | Yes) \times P(High | Yes) \times P(True | Yes) \times P(Yes)$$

$$P(Yes | X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529 \rightarrow 0.2 = \frac{0.00529}{0.02057 + 0.00529}$$

$$P(No | X) = P(Rainy | No) \times P(Cool | No) \times P(High | No) \times P(True | No) \times P(No)$$

$$P(No | X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057 \rightarrow 0.8 = \frac{0.02057}{0.02057 + 0.00529}$$

I don't play golf in that bad weather....



Naive Bayes

Advantages



Simple algorithm.



Fast training and inference.



Small amount of training data.

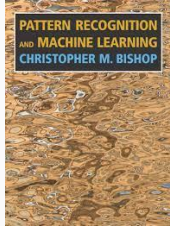
Disadvantages



The assumption of independent features does not hold in many practical problems.

This is a very **strong limitation!**

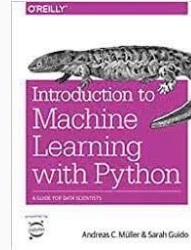
Additional Material



14.4 Tree-based models

14.2.0 Committees

14.3.0 Boosting



K-Nearest Neighbors pp 37-46

Naive Bayes pp 70-71



StatQuest with Josh Starmer

<https://youtu.be/efR1C6CvhmE>

Lab Session

- Please consider attending POD sessions for your project development.



Boosting

9th Lab Assignment