# Announcements

- Homework 3 due in **one week**
- Quiz 3 due **tomorrow**

# Project

- Everyone should be on a project team now!
  - **Email me ASAP if not!**

- Project Milestone 1 due in 2 weeks (10/18) at 8pm
  - 1 page plan (details shortly)
  - Should be easy, but please don’t procrastinate!

# Project

- **Project Instructions**
    - https://docs.google.com/document/d/1q_iR-EH28eqwq2oCi9uSpBnWRKP8XFrN/

- **Project Milestone 1**
    - https://docs.google.com/document/d/1R8SL6gcl0GlqmeB_p8fv_jVX7b62mlZ5/

# Goal

- Build experience experimenting with machine learning algorithms on real-world datasets
  - Lots of insights that you don’t get from lectures, or even homework!
  - Build intuition for relative importance of different design decisions
  - Learn to start simple and increment from there

# Datasets

- **Computer vision**
  - CIFAR-10 dataset
  - 10-class classification dataset (cat, dog, deer, car, truck, etc.)
  - https://www.cs.toronto.edu/~kriz/cifar.html

- **Natural language processing (NLP)**
  - IMDB reviews dataset
  - Sentiment prediction dataset (binary classification)
  - https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

# Datasets

- Strongly encourage subsetting data while prototyping algorithms
  - Typically, a thousand examples is plenty to train on for prototyping
- You should scale up the dataset when performing final training/evaluation runs
  - However, if you have limited compute, you are free to subset the dataset even for final training/evaluation runs to a reasonable extent
  - E.g., you should probably have at least a few thousand training examples)

# Compute

- You are strongly encouraged to use (relatively) small architectures
- Thus, you should be able to use Google Colab for all evaluations
- You may also consider signing up for Amazon SageMaker Studio Lab
  - https://studiolab.sagemaker.aws

# Implementation

- **For each dataset, you must implement:**
  - One traditional pipeline (e.g., logistic/softmax regression, etc.)
  - One deep learning pipeline (e.g., CNNs, RNNs, transformers, etc.)

- Four pipelines total

# Traditional Pipeline

- **For NLP:**
  - You should use feature engineering in the traditional pipeline
  - You are allowed (but not required) to use pretrained word embeddings
  - You may want to try using PCA or LASSO regularization for feature selection
  - You should experiment with different sets of features

- **For computer vision:**
  - You should try softmax regression
  - You should make sure to standardize your features as described in class

# Deep Learning Pipeline

- For at least one of the two datasets, you should build your own architecture from scratching in PyTorch
  - **MLPs do not count!**

- If using a preexisting architecture, you should compare training from scratch vs. finetuning a pretrained model

- For architectures that you build yourself, you should compare varying hyperparameters including the dimension of intermediate layers and the number of intermediate layers

# Keep It Simple!

- For the architecture(s) you implement yourself, keep it simple!
- Even very simple architectures such as a single convolutional or LSTM layer can already be very effective

# Evaluation

- You are expected to perform two evaluations:
  - **Standard evaluation:** Evaluate performance on test set, including different hyperparameter choices
  - **Robustness evaluation:** Evaluate performance on dataset shifts

# Test Set Performance

- Report the test set performance of your approach

- Hyperparameter variations
  - **Traditional pipeline:** At least one hyperparameter of your learning algorithm
  - **Deep learning pipeline:** Hyperparameters described on a previous slide

# Dataset Shifts

- For each dataset, you should try some kind of shift to the inputs, ideally finding one that breaks your model
  - **Computer vision:** Change contrast or brightness of images, rotate images, etc.
  - **NLP:** Train on short reviews and test on long reviews, swap out words with their synonyms, etc.

# Grading

- Most of your grade is on completing all the tasks described above

- A part is on comprehensive exploration of design choices
  - **Milestone 2:** 2/10 points
  - **Milestone 3:** 3/15 points
  - **Tentative breakdown**

- **Examples:**
  - Trying interesting features or feature selection techniques
  - Comparing interesting variations of neural network architectures
  - Devise interesting choices of dataset shift

# Project Milestone 1 (1 Page)

- **Part 1:** Implementation
  - Provide plans for feature engineering
  - Brainstorm neural network architectures (optional)

- **Part 2:** Evaluation
  - Describe hyperparameters you intend to vary
  - Brainstorm dataset shifts

- Describe how you are going to split work among your group
  - Everyone should contribute to each part!

# Lecture 10: Learning Ensembles

CIS 4190/5190

Fall 2023

# Recap: Ensemble Design Decisions

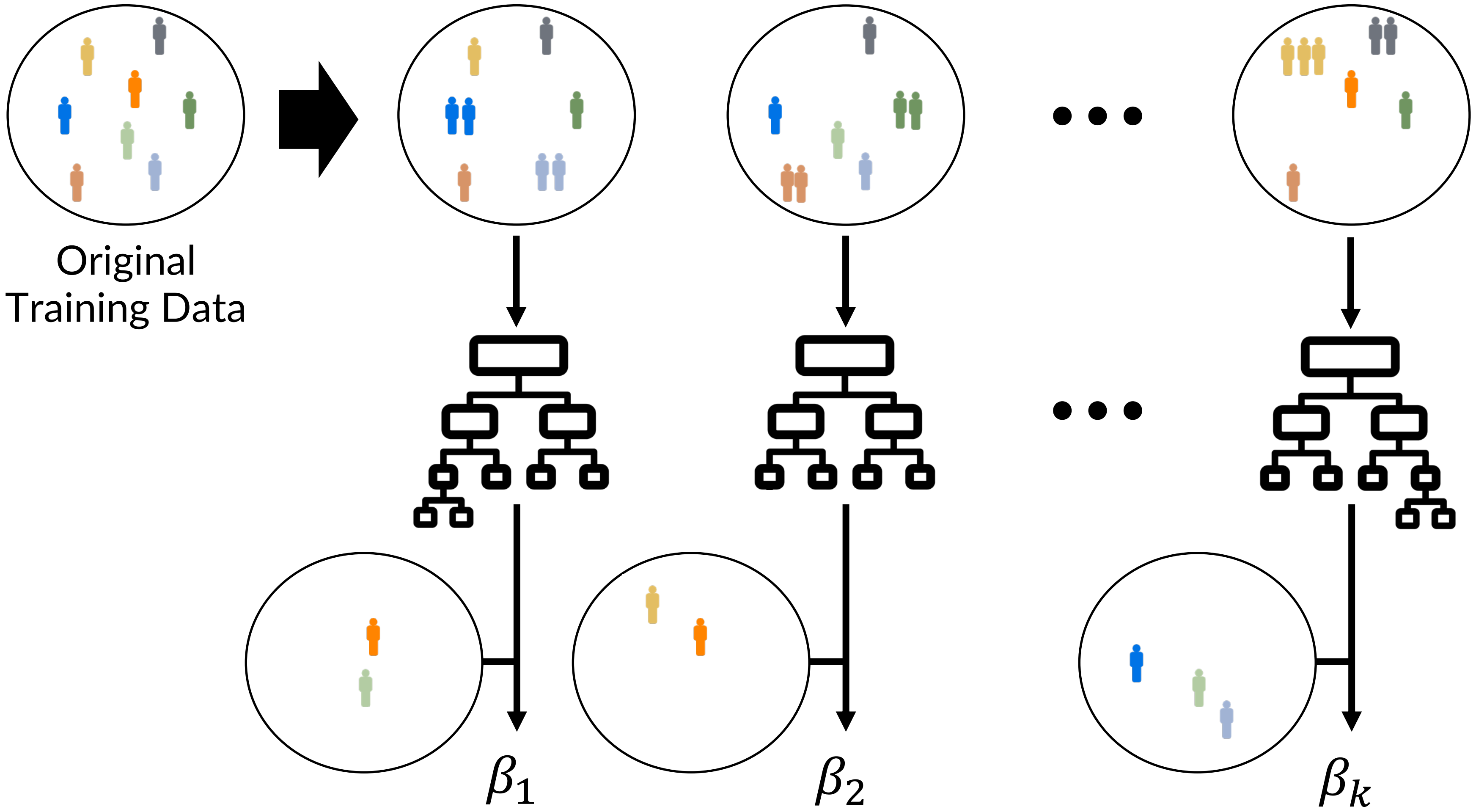
- How to learn the base models?
  - Bagging (randomize dataset)
  - Boosting (weighted dataset)

- How to combine the learned base models?
  - Averaging (regression) or majority vote (classification)

# Recap: Bagging

- **Step 1:** Create bootstrap replicates of the original training dataset
- **Step 2:** Train a classifier for each replicate
- **Step 3 (Optional):** Use held-out validation set to weight models
  - Can just use average predictions

# Recap: Bagging



# Recap: Random Forests

- Ensemble of decision trees using bagging
  - Typically use simple average (over probabilities for classification)

- **Intuition:**
  - Large decision trees are good nonlinear models, but high variance
  - Random forests average over many decision trees to reduce variance without increasing bias

# Recap: Random Forests

- **Tweak 1:** Randomize features in learning algorithm instead of bagging
  - At DT node splitting step, subsample $\approx \sqrt{d}$ features
  - Allows each tree to use all features, but not at every node
  - **Aside:** If a few features are highly predictive, then they will be selected in many trees, causing the base models to be highly correlated

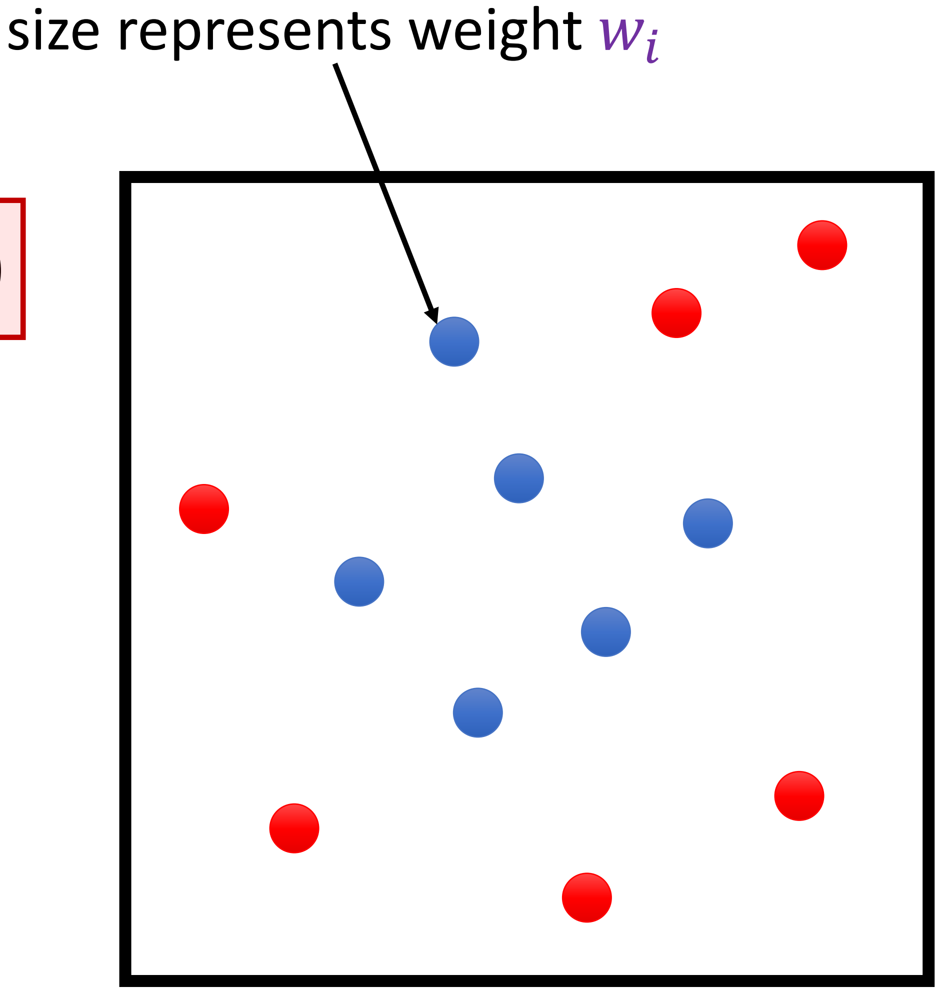
- **Tweak 2:** Train **unpruned** decision trees
  - Ensures base models have higher capacity
  - **Intuition:** Skipping pruning increases variance

# Recap: AdaBoost

- **Input**
  - Training dataset $Z$
  - Learning algorithm $\mathrm{Train}(Z, w)$ that can handle weights $w$
  - Hyperparameter $T$ indicating number of models to train
- **Output**
  - Ensemble of models $F(x) = \sum_{t=1}^{T} \beta_t \cdot f_t(x)$

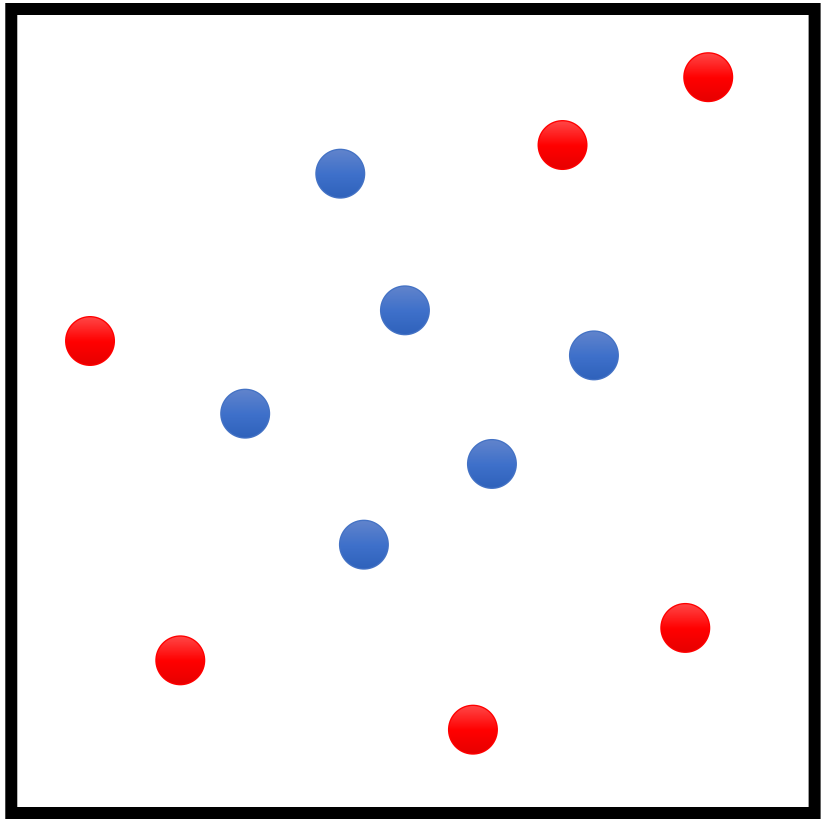
# AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



# AdaBoost

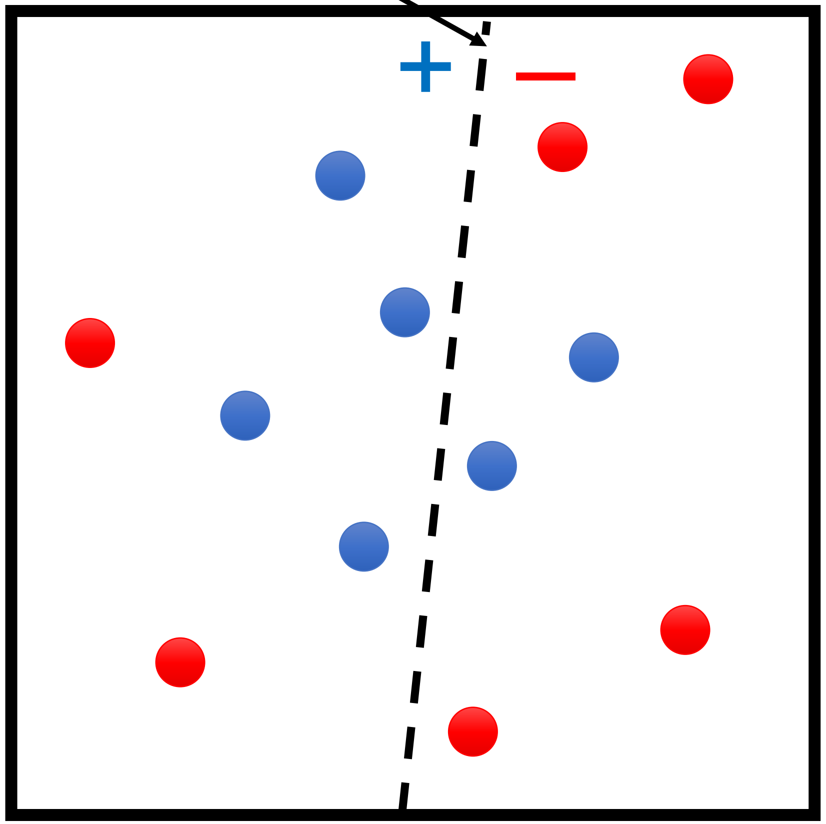
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



# AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

focus on linear classifiers $f_t$

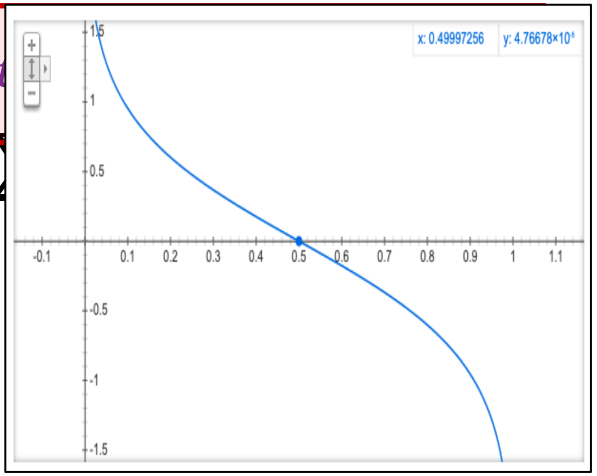


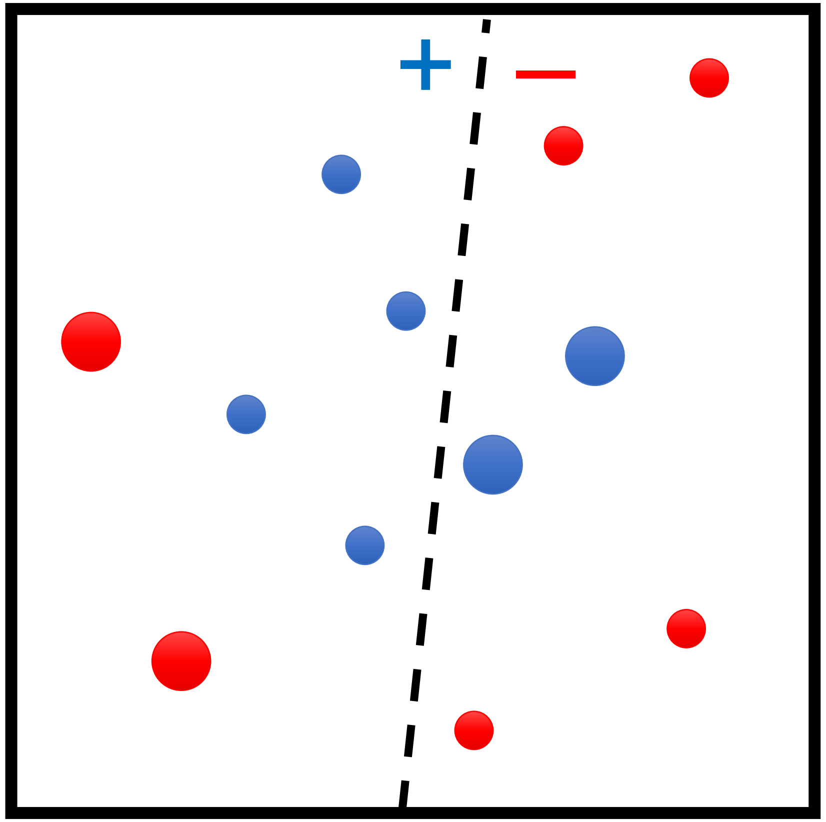

$t = 1$

# AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t}$
7. **return** $F(x) = \text{sign}($

$\beta_t$ becomes larger as
$\epsilon_t$ becomes smaller



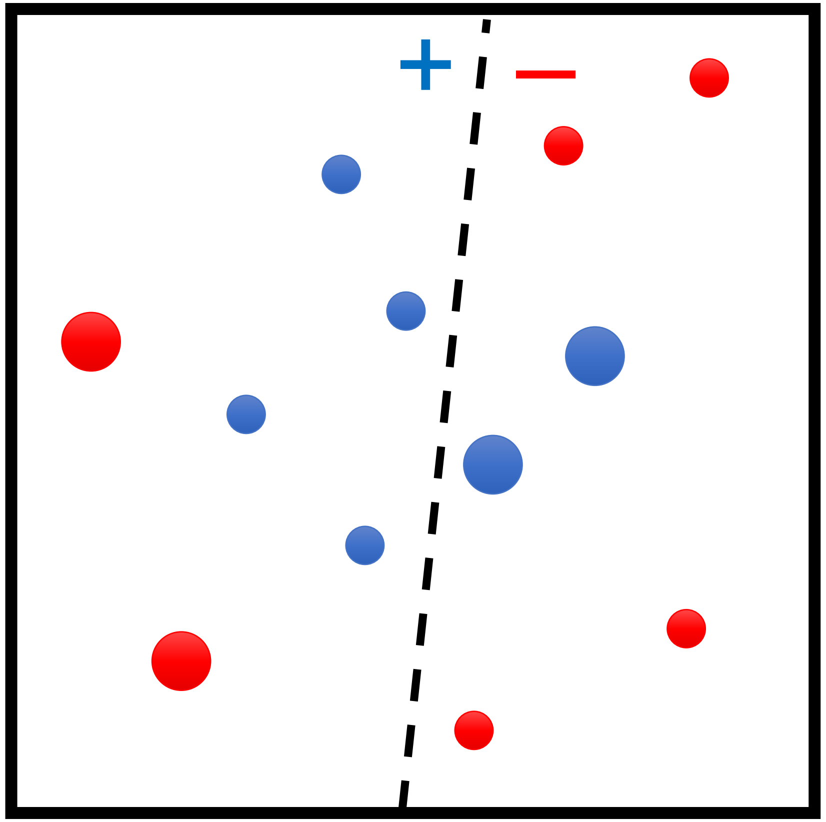




$t = 1$

# AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$

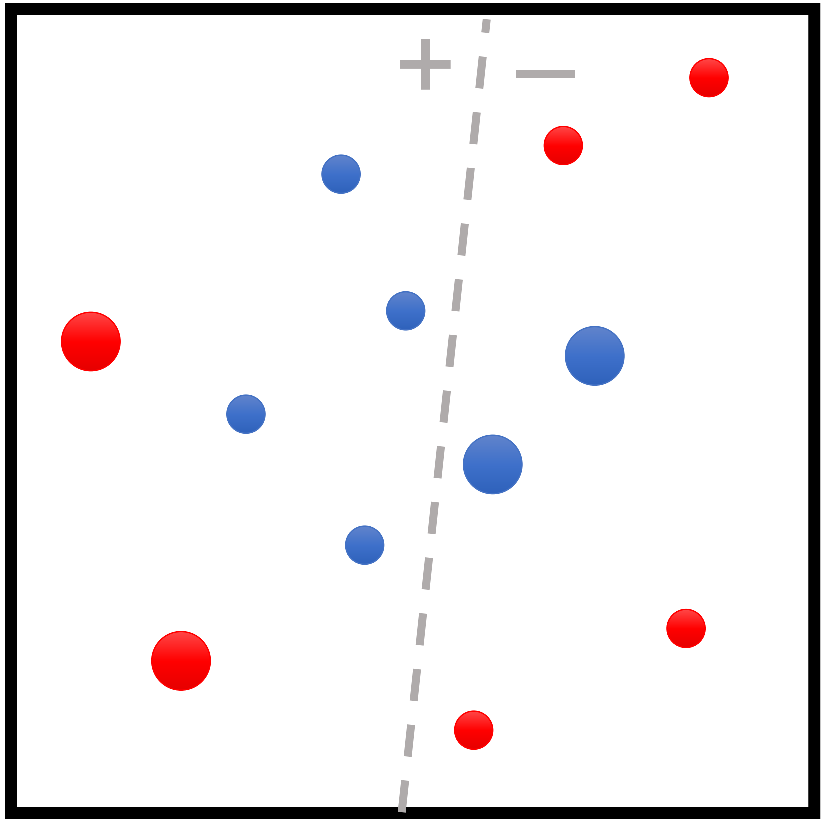
Use convention $y_i \in \{-1, +1\}$
If correct ($y_i = f_t(x_i)$) then multiply by $e^{-\beta_t}$
If incorrect ($y_i \neq f_t(x_i)$) then multiply by $e^{\beta_t}$




$t = 1$

# AdaBoost

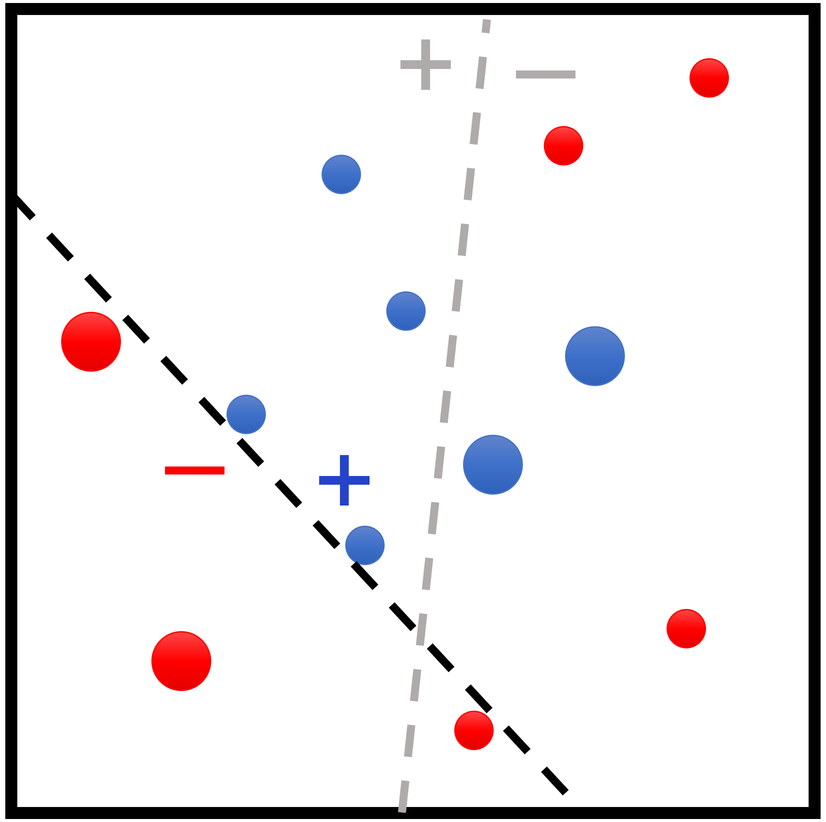
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 1$

# AdaBoost

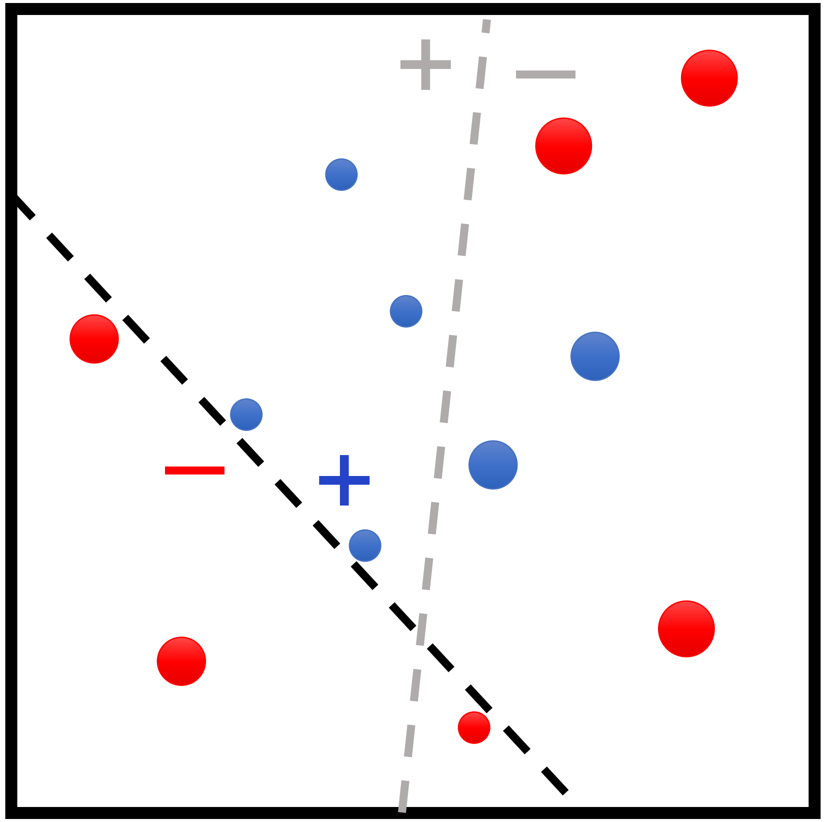
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$




$t = 2$

# AdaBoost

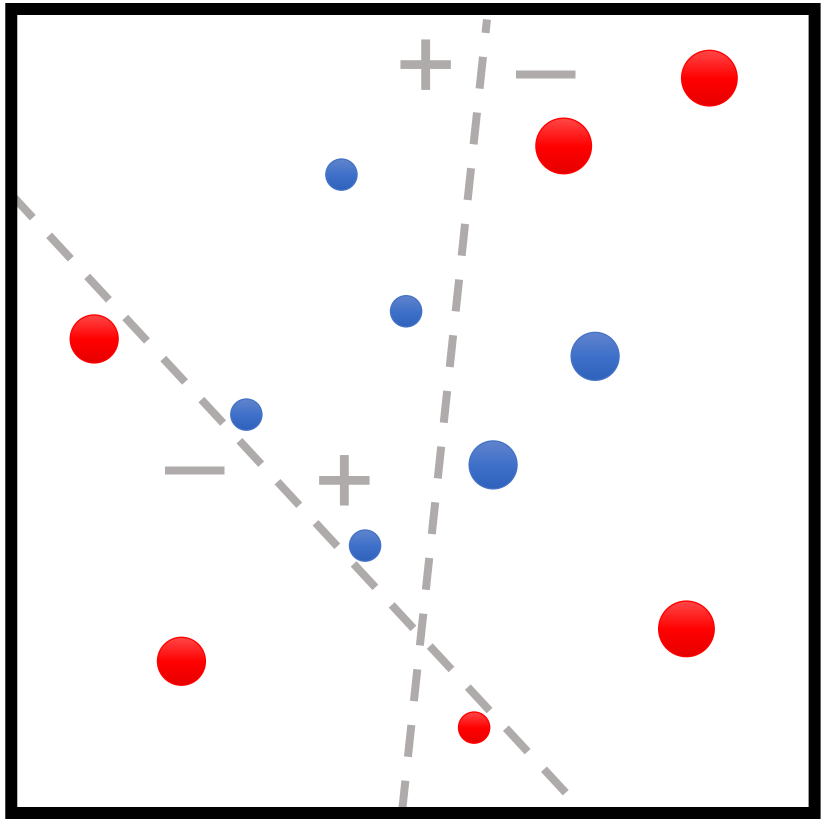
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$




$t = 2$

# AdaBoost

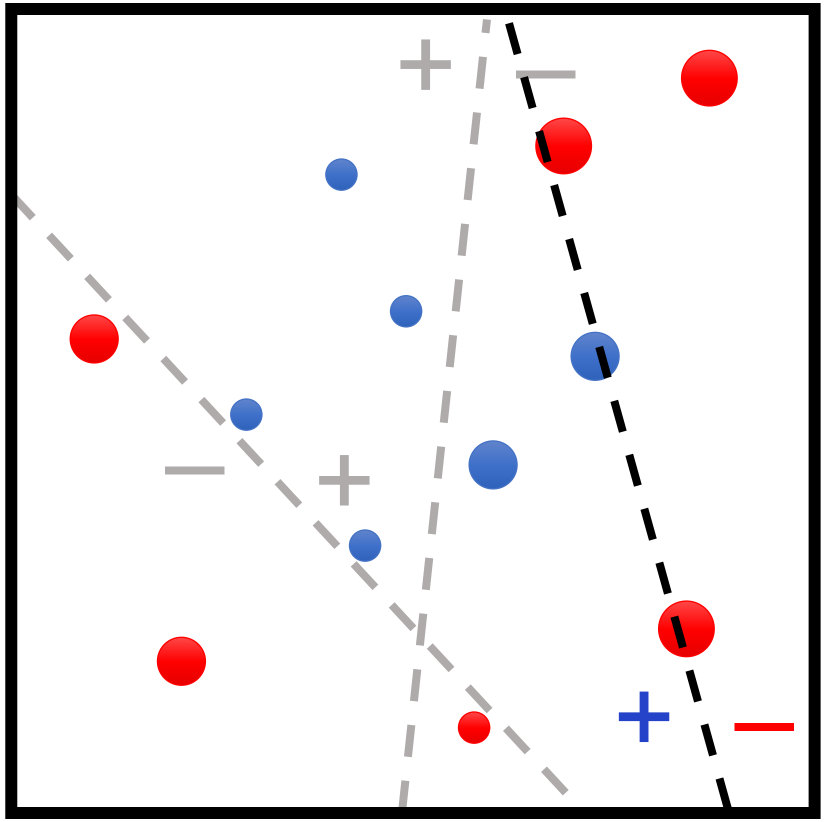
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$




$t = 2$

# AdaBoost

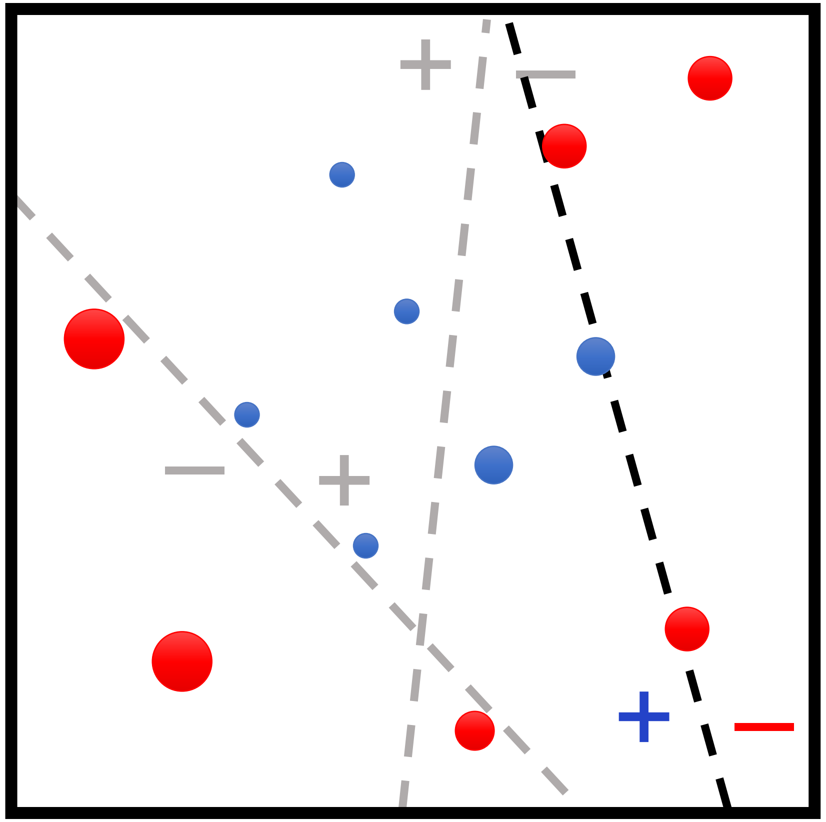
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$




$t = 3$

# AdaBoost

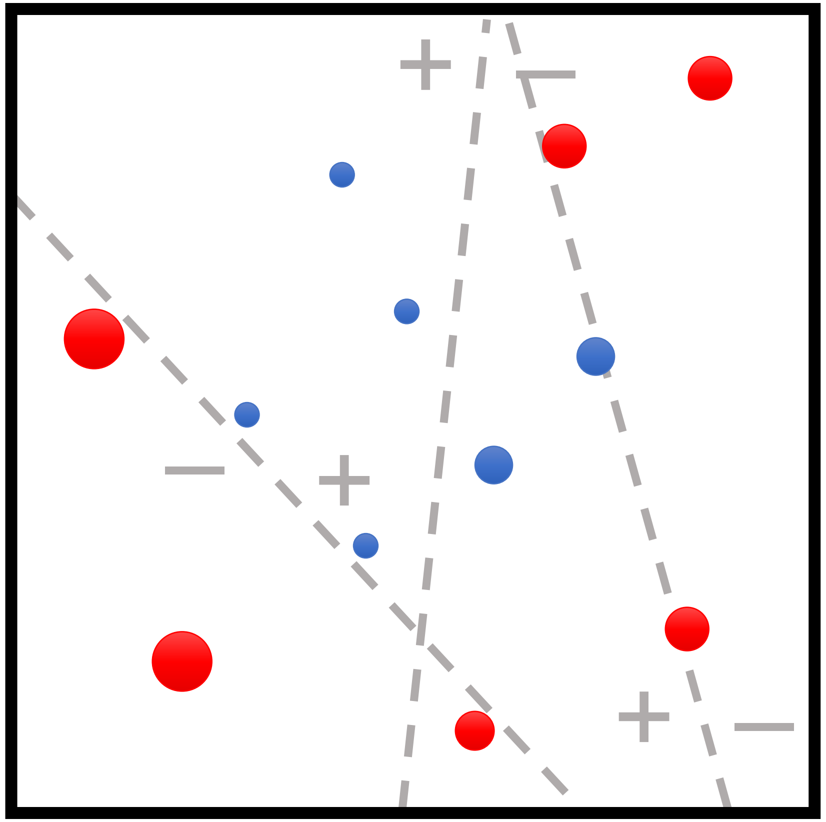
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 3$

# AdaBoost

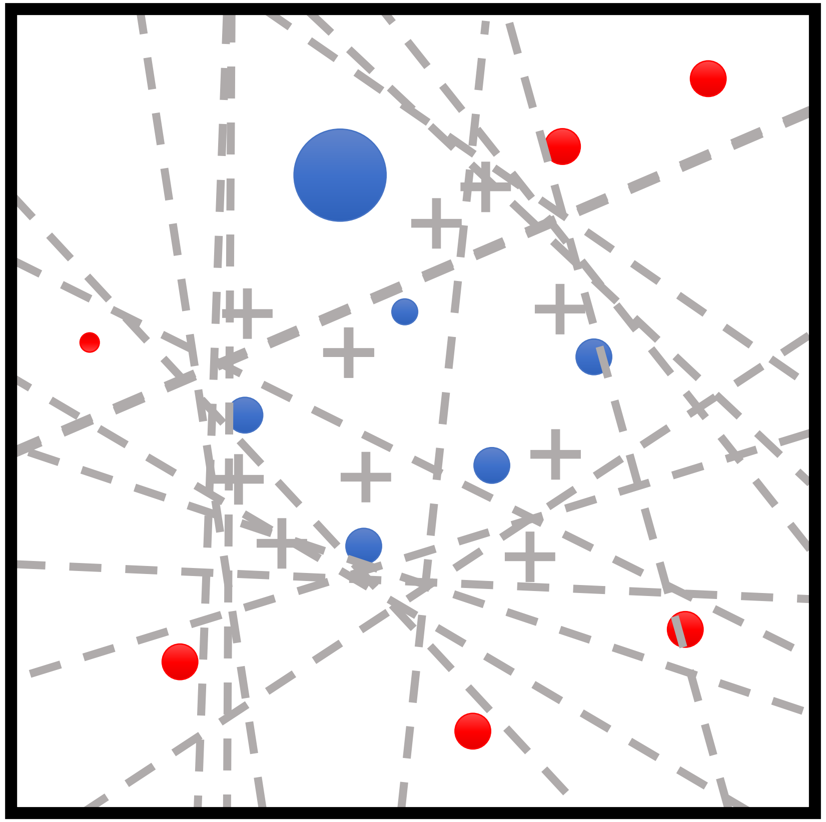
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$




$t = 3$

# AdaBoost

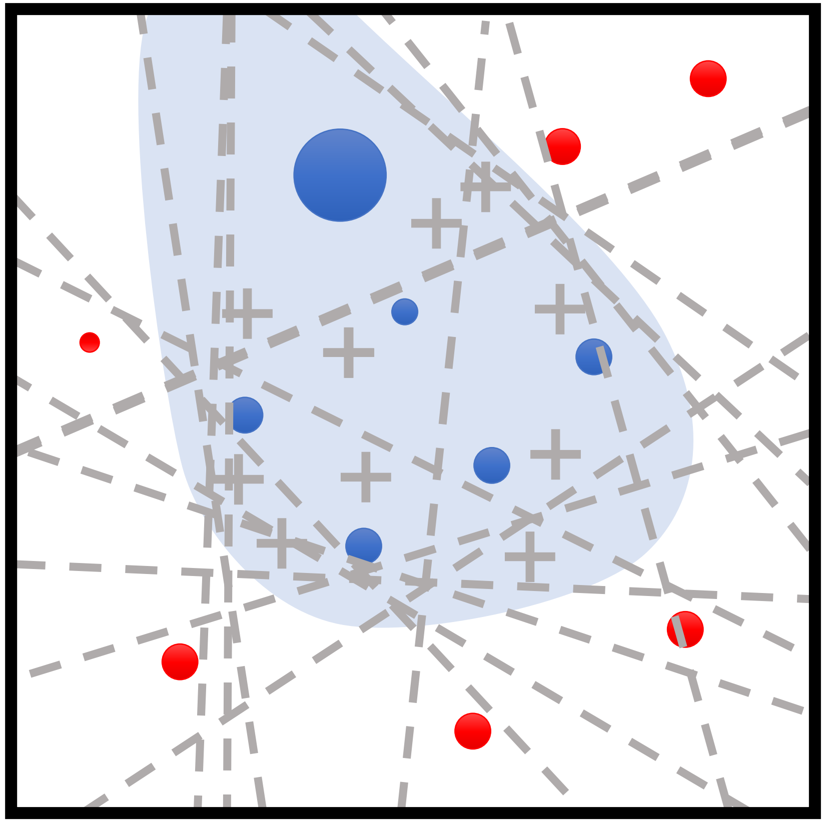
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = T$

# AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for $(x_i, y_i)$)
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all $i$)
7. **return** $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$



# AdaBoost Summary

- **Strengths:**
  - Fast and simple to implement
  - No hyperparameters (except for $T$)
  - Very few assumptions on base models
- **Weaknesses:**
  - Can be susceptible to noise/outliers when there is insufficient data
  - No way to parallelize
  - Small gains over complex base models
  - **Specific to classification!**

# Boosting as Gradient Descent

- Both algorithms: new model = old model + update

- **Gradient Descent:**

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta L(\theta_t; Z)$$

- **Boosting:**

$$F_{t+1}(x) = F_t(x) + \beta_{t+1} \cdot f_{t+1}(x)$$

- Here, $F_t(x) = \sum_{i=1}^{t} \beta_i \cdot f_i(x)$

# Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all $t$, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i)$$

# Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all $t$, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i) \approx y_i$$

- Rewriting this equation, we have

$$f_{t+1}(x_i) = F_{t+1}(x_i) - F_t(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{"residuals", i.e., error of the current model}}$$

# Boosting as Gradient Descent

- In other words, at each step, boosting is training the next model $f_{t+1}$ to approximate the residual:

$$f_{t+1}(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{“residuals”, i.e., error of the current model}}$$

- **Idea:** Train $f_{t+1}$ directly to predict residuals $y_i - F_t(x_i)$

- **This strategy works for regression as well!**

# Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train $f_{t+1}$ using dataset

$$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model $F_T$

# Boosting as Gradient Descent

- Consider losses of the form

$$L(F;Z)=\frac{1}{n}\sum_{i=1}^{n}\tilde{L}(F(x_i);y_i)$$

- In other words, sum of individual label-level losses $\tilde{L}(\hat{y};y)$ of a prediction $\hat{y}=F(x)$ if the ground truth label is $y$
- For example, $\tilde{L}(\hat{y};y)=\frac{1}{2}(\hat{y}-y)^2$ yields the MSE loss

# Boosting as Gradient Descent

- Residuals are the gradient of the squared error $\tilde{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$:

$$-\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i) = y_i - F_t(x_i) = \mathrm{residual_i}$$

- For general $\tilde{L}$, instead of $\{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$ we can train $f_{t+1}$ on

$$Z_{t+1} = \left\{\left(x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i)\right)\right\}_{i=1}^{n}$$

# Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \dots, T\}$:
  - **Step 1:** Train $f_{t+1}$ using dataset

$$Z_{t+1} = \left\{\left(x_i, y_i - F_t(x_i)\right)\right\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model $F_T$

# Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train $f_{t+1}$ using dataset

$$Z_{t+1} = \left\{ \left( x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}} (F_t(x_i); y_i) \right) \right\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model $F_T$

# Boosting as Gradient Descent

- Casts ensemble learning in the **loss minimization framework**
  - **Model family:** Sum of base models $F_T(x) = \sum_{t=1}^{T} f_t(x)$
  - **Loss:** Any differentiable loss expressed as

$$L(F; Z) = \sum_{i=1}^{n} \tilde{L}(F(x_i), y_i)$$

- Gradient boosting is a general paradigm for training ensembles with specialized losses (e.g., most NLL losses)

# Gradient Boosting in Practice

- Gradient boosting with depth-limited decision trees (e.g., depth 3) is one of the most powerful off-the-shelf classifiers available
  - **Caveat:** Inherits decision tree hyperparameters

- XGBoost is a very efficient implementation suitable for production use
  - A popular library for gradient boosted decision trees
  - Optimized for computational efficiency of training and testing
  - Used in many competition winning entries, across many domains
  - https://xgboost.readthedocs.io