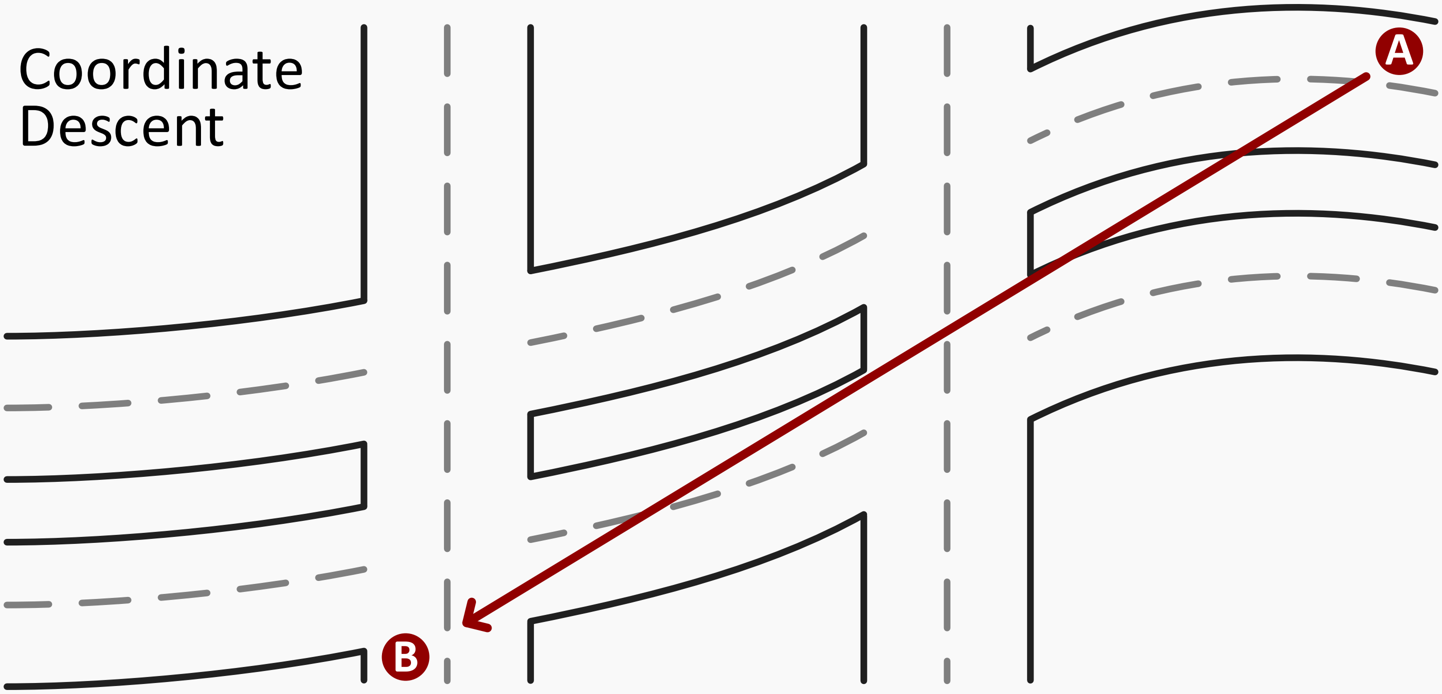
coordinate ⇓ descent

Coordinate Descent has synonymous logic to **gradient descent** focusing on **exponential loss** applied through combining weak classifiers into a powerful learning algorithm (**boosting**).

In the context of **AdaBoost**, the above illustrations are oriented towards general **binary classifiers**.

Understanding the origins of the formula and coordinate weighting functions can be illustrated using **coordinate descent** geometrically as shown below:



Assuming a starting point at **A**, the natural intuition of arriving at point **B** is to travel in a straight line. This, however, is not possible in the illustration above considering the available paths restrict the directions and distances point **A** can travel at a time. **Coordinate Descent**, in turn, has the objective of minimizing the function to determine the best path to the final point **B** from starting point **A**.

Coordinate Descent Pseudocode

For $t = 1{:}T$

Choose direction h_t

Choose the distance to travel in direction α_t

End

Output the final position: $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x_i)$

Therefore, each **Weak Classifier** (**decision boundary**) $h_1, h_2, \ldots, h_t$ can be thought of as the **direction traveled** above.

In abstract terms, h_t represents the chosen directions and α_t represents how far traveled in the determined direction for each iteration of the **AdaBoost** algorithm.

AdaBoost Pseudocode

Assign observation i the weight of $d_{1i} = \frac{1}{n}$ (equal weights)

For $t = 1{:}T$

Train weak learning algorithm using data weighted by $d_{1i} = \frac{1}{n}$ ...

...producing weak classifier h_t.

Choose coefficient α_t.

Update weights:

$y_i h_t(x_i) = 1$ if correct, -1 if incorrect

$$d_{t+1,i} = \frac{d_{t,i}\exp\left(-\alpha_t\left(y_i h_t(x_i)\right)\right)}{Z_t}$$ with Z_t serving as a normalization factor

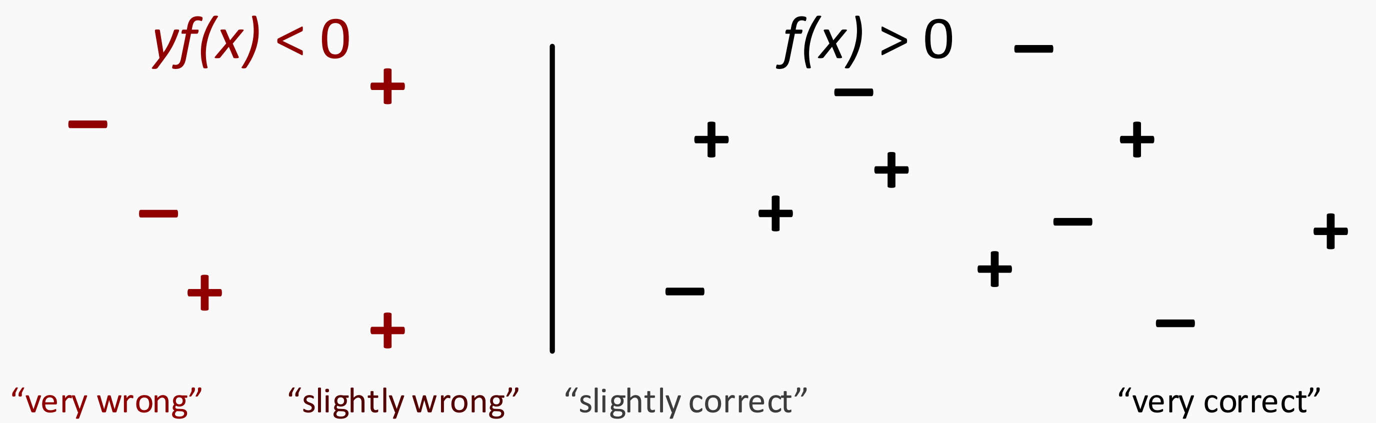
End

Output the final classifier: $H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x_i)\right)$

The above representation of **AdaBoost** can be explained logically in steps as follows:

- The **Weak Learning Algorithm** is essentially the most promising chosen direction of travel
- The assignment of **coefficient $\boldsymbol{\alpha_t}$** tells the algorithm know how far to travel in that direction
- The **weight update** essentially aids in selecting the next direction of travel in the next iteration
- The **final classifier** determines the total amount of distance travels in all directions

The final component to **Coordinate Descent** is the formula used to determine the hill to travel upon:

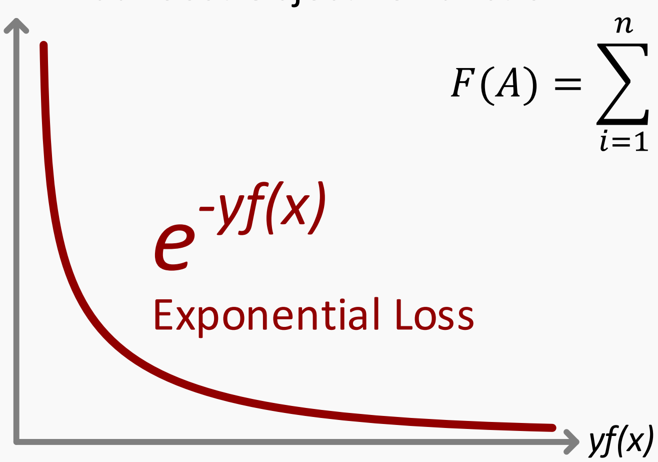


Returning to the geometrics representation of **classification** above, the left frame represents **misclassified points** and the right frame represents **correctly classified** points.

The objective is to choose **loss functions** that heavily penalize any and all **misclassified points** seen the left frame. This is achieved through the **Exponential Loss Function** as illustrated in the example to the right:

AdaBoost Objective Function:

$$F(A) = \sum_{i=1}^{n} e^{-yf(x)}$$



AdaBoost Objective Function:

$$F(A) = \sum_{i=1}^{n} e^{-yf(x)}$$

The direction of travel chosen will represent the **steepest descent**:

$$j_t = \underset{j}{\operatorname{argmax}} \left| -\left(\frac{dF(A + \alpha l_j)}{d\alpha} \right) \right|_{a=0} = \underset{j}{\operatorname{argmax}} \sum_{\substack{i \text{ that are misclassified} \\ y_i h_j(x_i) = -1}} d_{t,i}$$

The direction is that of the **lowest weighted error**.

In the circumstance that the optimal direction is not chosen, the algorithm as flexible and will correct.

The above is the concept of applying coefficient a_t as a weighting function to **AdaBoost**:

$$0 = \left| \frac{dF(A + \alpha l_j)}{d\alpha} \right|_{\alpha_t} \rightarrow \alpha_t = \frac{1}{2} \ln \left(\frac{1 - Error_t}{Error_t} \right)$$

Lastly, the weights will be updated to ensure the steepest descents are computed in each reiteration.

The algorithm essentially will continue travelling along the descent through reiterations until the benefit is **substantially decayed**. Therefore, a **1 dimensional optimization problem** is being solved; where the minimized function along that dimension is chosen.

The Difference Between Boosted Decision Trees and Decision Forests

Decision Forests

- Computing many trees from different subsets of data and features
- Average the results (**bagging**)

Boosted Decision Trees

- Reweight the data to generate different trees
- The combination of multiple weights minimizes the training error (**coordinate descent performed on exponential lost**)

The differences coverage as the result of each is essentially a multitude of **overfitted, heuristic decision trees** that are combined in some determined way. The philosophy behind **decision forests** is to **average** everything to **reduce variance**. The approach behind **boosting** tries to **minimize bias** and make the model **more accurate**, but ends up **reducing variance** anyway because the trees that it generates tend to be diverse from continuous reweighting.