Boosting

Breaking down the expected text error:

$$\underbrace{\mathbb{E}_{\mathcal{D}\sim P^n,(\boldsymbol{x},y)\sim P}\left[\left(f_{\mathcal{D}}(\boldsymbol{x})-y\right)^2\right]}_{\text{Expected Test Error}} =$$

$$\underbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{\mathcal{D}}}\left[\left(f_{\mathcal{D}}(\boldsymbol{x})-\bar{f}(\boldsymbol{x})\right)^2\right]}_{\text{Variance}} +$$

$$\underbrace{\mathbb{E}_{\boldsymbol{x},y}\left[\left(\bar{y}(\boldsymbol{x})-y\right)^2)\right]}_{\text{Noise}} +$$

$$\underbrace{\mathbb{E}_{\boldsymbol{x}}\left[\left(\bar{f}(\boldsymbol{x})-\bar{y}(\boldsymbol{x})\right)^2\right]}_{\text{Bias}}$$

Combining weak learners

- **Weak learner**
  - A regressor or classifier which is just barely better than random guessing
- Famous question: Michael Kearns (1988) - Can weak learners be combined to create a strong learner with low bias?
- Famous answer: Robert Schapire (1990) - Yes.

Boosting

- Start with some weak learners $f_i(\boldsymbol{x})$
  - For instance, CART trees with very limited depth (1-2)
- The ensemble classifier will be $F(\boldsymbol{x}) = F_m(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i f_i(\boldsymbol{x})$
- **Inference:** evaluate all classifiers and return the weighted sum
- **Training:** build the classifiers f_i and weights α_i iteratively

Boosting training

- We will create $F_1(\boldsymbol{x}), F_2(\boldsymbol{x}), \ldots$
- Very similar to gradient descent but
  - instead of modifying the parameters
  - we add a new function to the ensemble
- Our loss will be:

$$\mathcal{L}(F) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(F(\boldsymbol{x_i}), y_i\right)$$

What are we optimizing when adding a new function?

- Let us say that we are at iteration t, with F_t being our current ensemble classifier
- What are we searching for in the new function (weak learner) to add for the ensemble?
- The one that minimizes the loss the most!

$$f_{t+1} = argmin_{f,\alpha}\, \mathcal{L}(F_t + \alpha f)$$

- Then, our new classifier will be:

$$F_{t+1} = F_t + \alpha f_{t+1}$$

Gradient descent in functional space

- The problem with this is the argmin!
  - Before, we were calculating minimums in an n-dimensional space of numbers $\mathbb{R}^n$
  - This happens in the space of *functions*!
  - the math is a bit more advanced...
- But maybe we can do some approximations
  - Assume that the loss function $\mathcal{L}$ is linear in the neighborhood (i.e. for small α)
  - So we can just work with the first two terms of the Taylor approximation
  - This is gradient descent, a problem we had seen before
  - But now in functional space.

Generic boosting (a.k.a Anyboost)

Input: ℓ, α, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{A}$
$H_0 = 0$
for *t=0:T-1* **do**
  $\forall I : r_i = \frac{\partial \ell((H_t(\mathbf{x}_1), y_1), \ldots, (H_t(\mathbf{x}_n), y_n))}{\partial H(\mathbf{x}_i)}$
  $h_{t+1} = \mathbb{A}(\{(\mathbf{x}_1, r_1), \ldots, (\mathbf{x}_n, r_n)\}) = \mathrm{argmin}_{h \in \mathbb{H}} \sum_{i=1}^{n} r_i h(\mathbf{x}_i)$
  if $\sum_{i=1}^{n} r_i h_{t+1}(\mathbf{x}_i) < 0$ **then**
    $H_{t+1} = H_t + \alpha_{t+1} h_{t+1}$
  else
    return (H_t)
  end
end
return H_T

Algorithm 1: AnyBoost in Pseudo-Code

Specific implementations

- There are many variants and implementations
- Classification and regression
- Various kinds of small learners
- Various ways to approximate the minimum
- Various ways to choose the new learner $\mathbb{A}$
    - It doesn't have to be perfect!
    - As long as we make progress, i.e. decrease the loss

Gradient boosted regression tree (GBRT)

- **Used for**: classification $y_i \in \{+1, -1\}$ or regression (single or multidimensional) $y_i \in \mathbb{R}^k$
- **Weak learners**: $f \in \mathbb{F}$ are regressors, typically fixed-depth (eg. depth = 4) regression trees
- **Step size**: α fixed to a small constant (hyper parameter)
- **Loss function**: can be any differentiable, convex loss that decomposes over the samples

AdaBoost

- **Used for**: classification $y_i (but had been extended to regression)
- **Weak learners**: binary classifiers
  - Typically **decision stumps**: very shallow decision trees
- **Loss function**: Exponential loss

$$\mathcal{L}(F) = \sum_{i=1}^{n} e^{-y_i F(\boldsymbol{x}_i)}$$

- **Step size**: in the settings of AdaBoost we can find the optimal step size α in closed form!
  - Then, you need to update all the weights and re-normalize

AdaBoost general picture

- Name comes from Adaptive Boosting
- It is adaptive in the sense that new weak learners added are tweeked to classify correctly instances misclassified by previous learners
- The fact that α can be computed in closed form, makes AdaBoost converge extremely fast!
  - Training loss decreases exponentially!
  - It reaches zero training error in $O(log(n))$ time
  - In practive it often makes sense to continue boosting after no classification mistakes are done...

AdaBoost general picture

- AdaBoost can turn any weak learner that can classify slightly better than 0.5 into a strong learner
- AdaBoost with decision trees as weak learners is in competition to be one of the best out of the box algorithms