Ensemble Methods

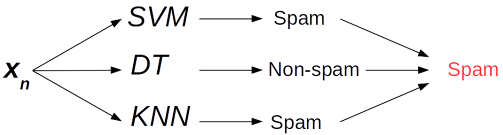
Piyush Rai

Introduction to Machine Learning (CS771A)

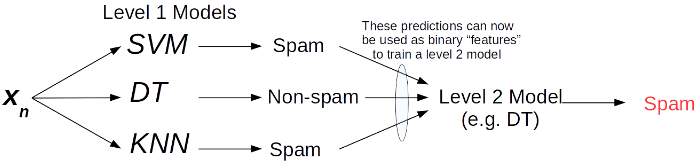
November 8, 2018

Some Simple Ensembles

- Voting or Averaging of predictions of multiple pre-trained models



- "Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data



.. are also like Ensembles..

- Mixture of Experts: Many "local" models are combined

$$p(y_*|\boldsymbol{x}_*) = \sum_{k=1}^{K} p(z_* = k)p(y_*|\boldsymbol{x}_*, z = k)$$

- Deep Learning: Outputs of several hidden units are combined

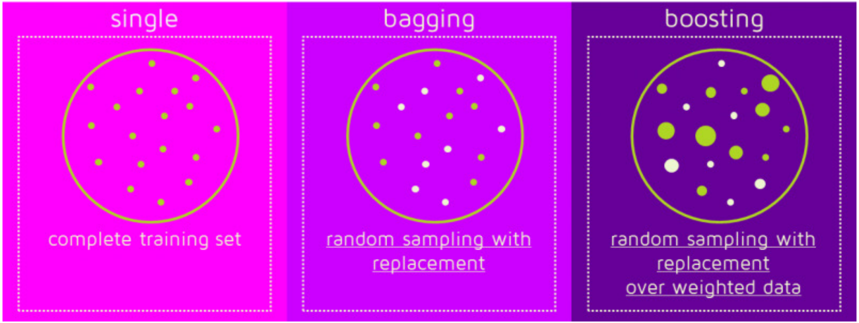
$$y_* = \sum_{k=1}^{K} v_k h_k(x_*)$$

- Bayesian Learning: Posterior weighted averaging of predictions by all possible parameter values

$$p(y_*|\boldsymbol{x}_*, \mathbf{X}, \boldsymbol{y}) = \int p(y_*|\boldsymbol{w}, \boldsymbol{x}_*)p(\boldsymbol{w}|\boldsymbol{y}, \mathbf{X})d\boldsymbol{w}$$

Ensembles: Another Approach

- Train same model multiple times on different data sets, and "combine" these "different" models
- Bagging and Boosting are two popular approaches for doing this
- How do we get multiple training sets (in practice, we only have one training set)?



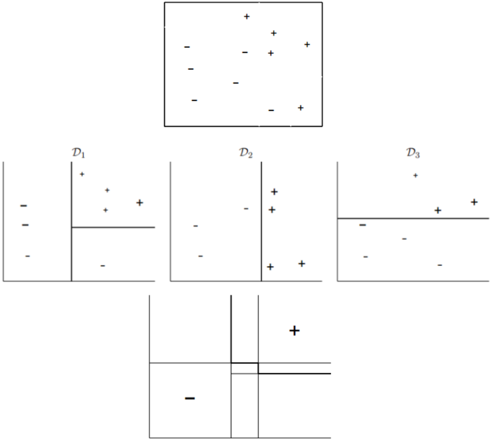
- Note: Bagging trains independent models; boosting trains them sequentially (we'll see soon)

Bagging

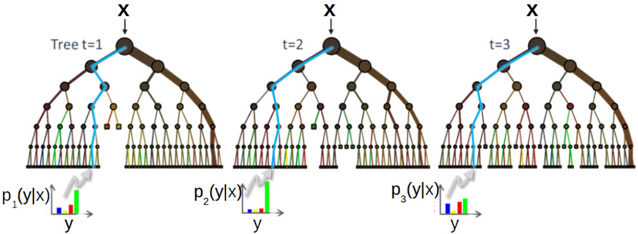
- Bagging stands for Bootstrap Aggregation
- Takes original data set $\mathcal{D}$ with N training examples
- Creates M copies $\{\tilde{\mathcal{D}}_m\}_{m=1}^M$
  - Each $\tilde{\mathcal{D}}_m$ is generated from $\mathcal{D}$ by sampling with replacement
  - Each data set $\tilde{\mathcal{D}}_m$ will have N examples
  - These data sets are reasonably different from each other. Reason: Only about 63% unique examples from $\mathcal{D}$ appear in each $\tilde{\mathcal{D}}_m$
- Train models $h_1, \ldots, h_M$ using $\tilde{\mathcal{D}}_1, \ldots, \tilde{\mathcal{D}}_M$, respectively
- Use an averaged model $h = \frac{1}{M}\sum_{m=1}^M h_m$ as the final model
- Bagging is especially useful for models with high variance and noisy data
- High variance models = models whose prediction accuracies varies a lot across different data sets

Bagging: illustration

Top: Original data, Middle: 3 models (from some model class) learned using three data sets chosen via bootstrapping, Bottom: averaged model



A Decision Tree Ensemble: Random Forests



- An bagging based ensemble of decision tree (DT) classifiers
- Also uses bagging on features (each DT will use a random set of features)
  - Example: Given a total of D features, each DT uses $\sqrt{D}$ randomly chosen features
  - Randomly chosen features make the different trees uncorrelated
- All DTs usually have the same depth
- Prediction for a test example votes on/averages predictions from all the DTs

Boosting

- Another ensemble based approach
- The basic idea is as follows
  - Take a weak learning algorithm
    - Only requirement: Should be only slightly better than random
  - Turn it into an awesome one by making it focus on difficult cases
- Most boosting algoithms follow these steps:
  1. Train a weak model on some training data
  2. Compute the error of the model on each training example
  3. Give higher importance to examples on which the model made mistakes
  4. Re-train the model using "importance weighted" training examples
  5. Go back to step 2
- Note: Unlike bagging, boosting is a sequential algorithm (models learned in a sequence)

The AdaBoost Algorithm (Freund and Schapire, 1995)

- Given: Training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ with $y_n \in \{-1, +1\}, \forall n$
- Initialize importance weight of each example $(\boldsymbol{x}_n, y_n)$: $D_1(n) = 1/N, \forall n$
- For round $t = 1 : T$
  - Learn a weak $h_t(\boldsymbol{x}) \to \{-1, +1\}$ using training data weighted as per D_t
  - Compute the weighted fraction of errors of h_t on this training data

$$\epsilon_t = \sum_{n=1}^{N} D_t(n) \mathbb{1}[h_t(\boldsymbol{x}_n) \neq y_n]$$

  - Set "importance" of h_t: $\alpha_t = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$(gets larger as ϵ_t gets smaller)
  - Update the weight of each example

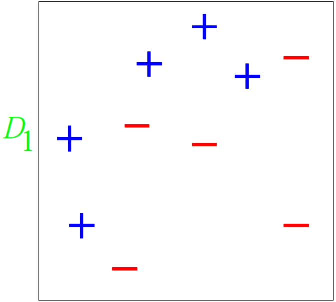
$$\begin{aligned} D_{t+1}(n) &\propto \begin{cases} D_t(n) \times \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) = y_n \quad \text{(correct prediction: decrease weight)} \\ D_t(n) \times \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) \neq y_n \quad \text{(incorrect prediction: increase weight)} \end{cases} \\ &= D_t(n)\exp(-\alpha_t y_n h_t(\boldsymbol{x}_n)) \end{aligned}$$

  - Normalize D_{t+1} so that it sums to 1: $D_{t+1}(n) = \frac{D_{t+1}(n)}{\sum_{m=1}^{N} D_{t+1}(m)}$
- Output the "boosted" final hypothesis $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

AdaBoost: A Toy Example

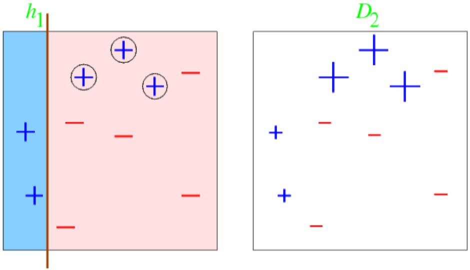
Consider binary classification with 10 training examples

Initial weight distribution D_1 is uniform (each point has equal weight $= 1/10$)



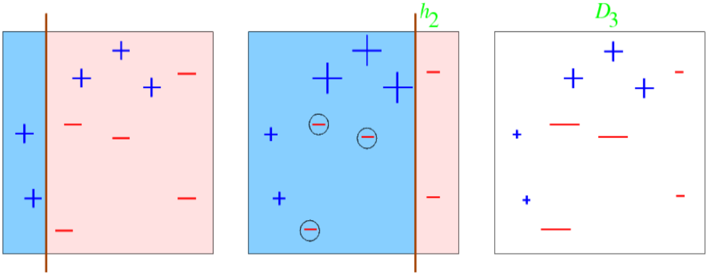
Let's assume each of our weak classifers is a very simple **axis-parallel linear classifier**

After Round 1



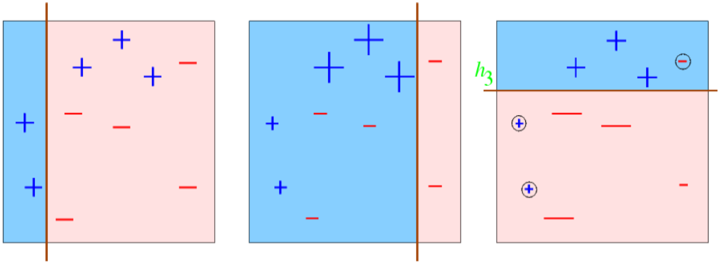
- Error rate of h_1: $\epsilon_1 = 0.3$; weight of h_1: $\alpha_1 = \frac{1}{2}\ln((1-\epsilon_1)/\epsilon_1) = 0.42$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_1)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_1)$)

After Round 2



- Error rate of h_2: $\epsilon_2 = 0.21$; weight of h_2: $\alpha_2 = \frac{1}{2}\ln((1-\epsilon_2)/\epsilon_2) = 0.65$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

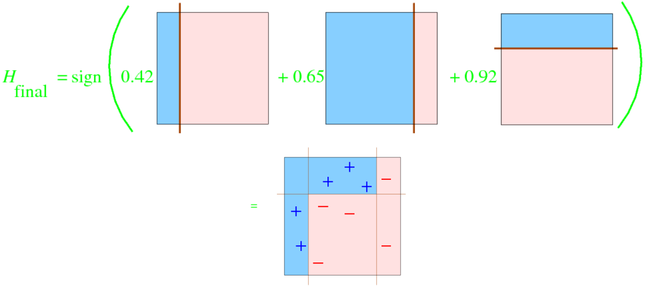
After Round 3



- Error rate of h_3: $\epsilon_3 = 0.14$; weight of h_3: $\alpha_3 = \frac{1}{2}\ln((1-\epsilon_3)/\epsilon_3) = 0.92$
- Suppose we decide to stop after round 3
- Our ensemble now consists of 3 classifiers: h_1, h_2, h_3

The Final Classifier

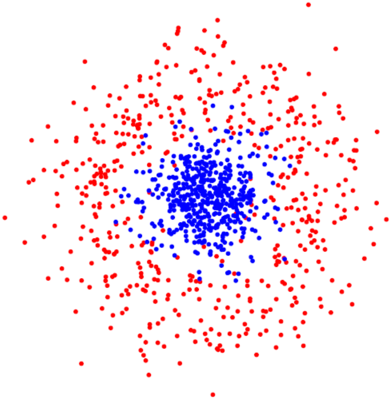
- Final classifier is a weighted linear combination of all the classifiers
- Classifier h_i gets a weight α_i



- Multiple weak, linear classifiers **combined** to give a strong, nonlinear classifier

Another Example

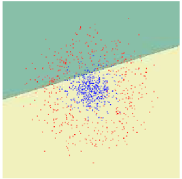
- Given: A nonlinearly separable dataset
- We want to use Perceptron (linear classifier) on this data

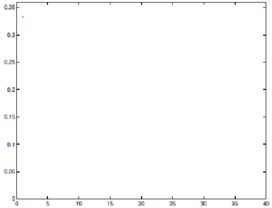


AdaBoost: Round 1

- After round 1, our ensemble has 1 linear classifier (Perceptron)
- Bottom figure: X axis is number of rounds, Y axis is training error

$t = 1$

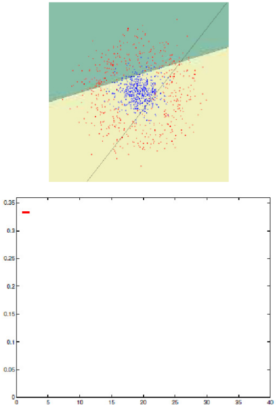




AdaBoost: Round 2

- After round 2, our ensemble has 2 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

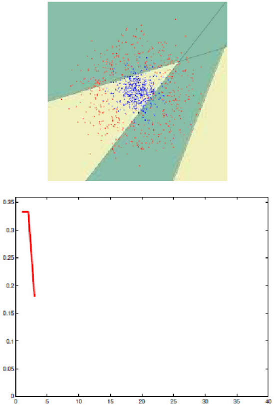
$t = 2$



AdaBoost: Round 3

- After round 3, our ensemble has 3 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

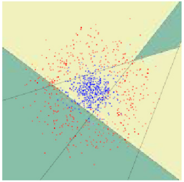
$t = 3$

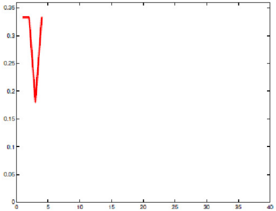


AdaBoost: Round 4

- After round 4, our ensemble has 4 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

$t = 4$

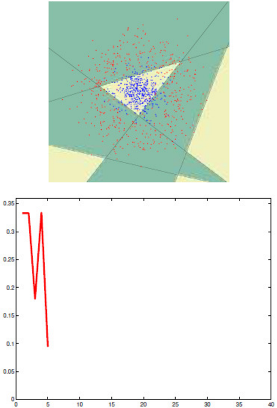




AdaBoost: Round 5

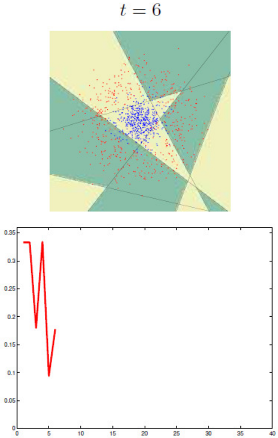
- After round 5, our ensemble has 5 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

$t = 5$



AdaBoost: Round 6

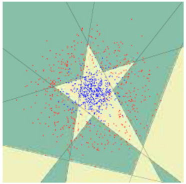
- After round 6, our ensemble has 6 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

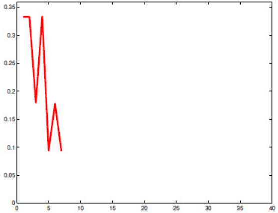


AdaBoost: Round 7

- After round 7, our ensemble has 7 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

$t = 7$

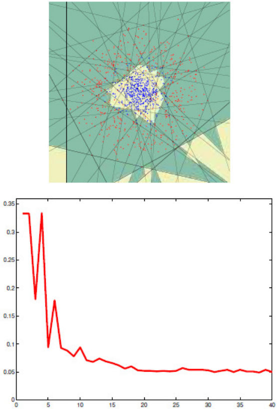




AdaBoost: Round 40

- After round 40, our ensemble has 40 linear classifiers (Perceptrons)
- Bottom figure: X axis is number of rounds, Y axis is training error

$t = 40$



AdaBoost: The Loss Function

- AdaBoost can be shown to be optimizing the following loss function

$$\mathcal{L} = \sum_{n=1}^{N} \exp\{-y_n H(\boldsymbol{x}_n)\}$$

where $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$, given weak base classifiers $h_1, \ldots, h_T$

Gradient Boosting Machine (GBM)

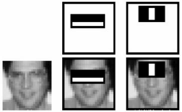
- Consider learning a function $F(x)$ by minimizing a squared loss $\frac{1}{2}(y - F(x))^2$
- Gradient boosting is a sequential way to construct such an $F(x)$
- For simplicity, assume we start with $F_0(x) = \frac{1}{N}\sum_{n=1}^{N} y_n$
- Given an existing model $F_m(x)$, let's assume the following improvement to it

$$F_{m+1}(x) = F_m(x) + h(x)$$

- Assume $F_m(x) + h(x) = y$. Find $h(x)$ by learning a model from x to the "residual" $y - F_m(x)$
- Called gradient boosting because the residual is the negative gradient of the loss w.r.t. $F(x)$
- Extensions for classification and ranking problems as well
- A very fast, parallel implementation of GBM is XGBoost (eXtreme Gradient Boosing)

Summary

- Ensemble methods are highly effective at boosting performance of simple learners
- Often achieve state-of-the-art results on many problems
  - AdaBoost was used in one of the first real-time face-detectors (Viola and Jones, 2001)



  - Netflix Challenge was won by an ensemble method (based on matrix factorization)
  - Many Kaggle competition have been won by Gradient Boosting methods such as XGBoost
  - Even outperform deep learning models on many problems
- Help reduces bias or/and variance of machine learning models
  - High bias: very simple models have high bias. Boosting can reduce it
  - High variance: very complex models have high variance. Bagging can reduce it