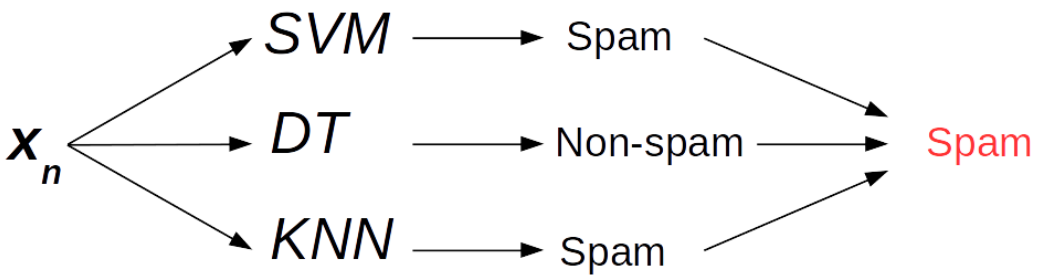
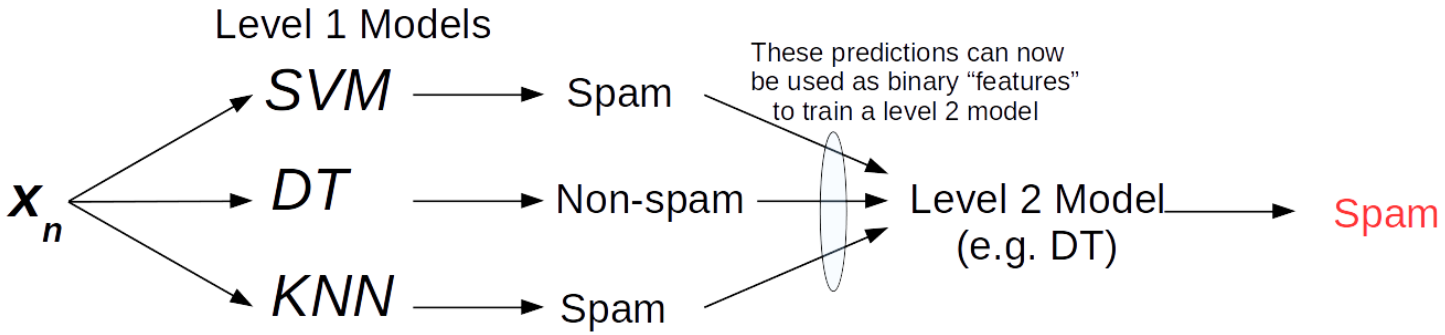
Some Simple Ensembles

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

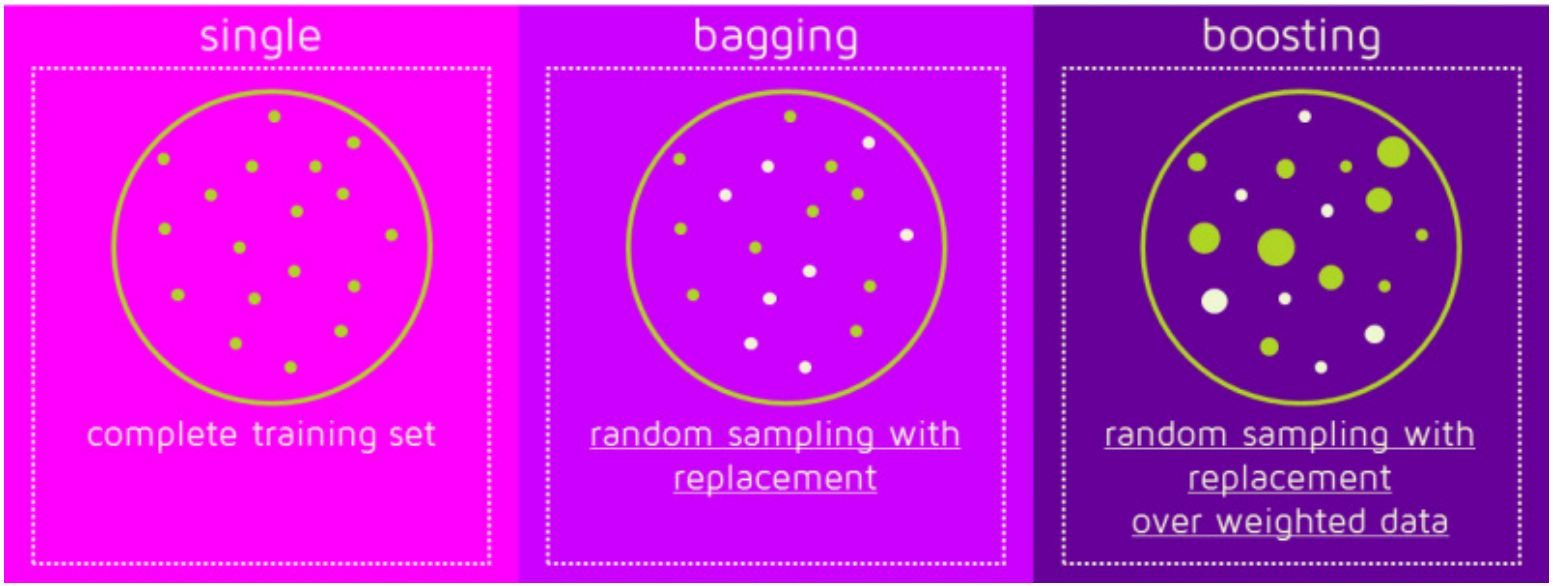


- "Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data



Ensembles: Another Approach

- Instead of training different models on same data, train same model multiple times on different data sets, and "combine" these "different" models
- We can use some simple/weak model as the base model
- How do we get multiple training data sets (in practice, we only have one data set at training time)? One option is to use the ideas illustrated below:



(note: green points are the selected ones)

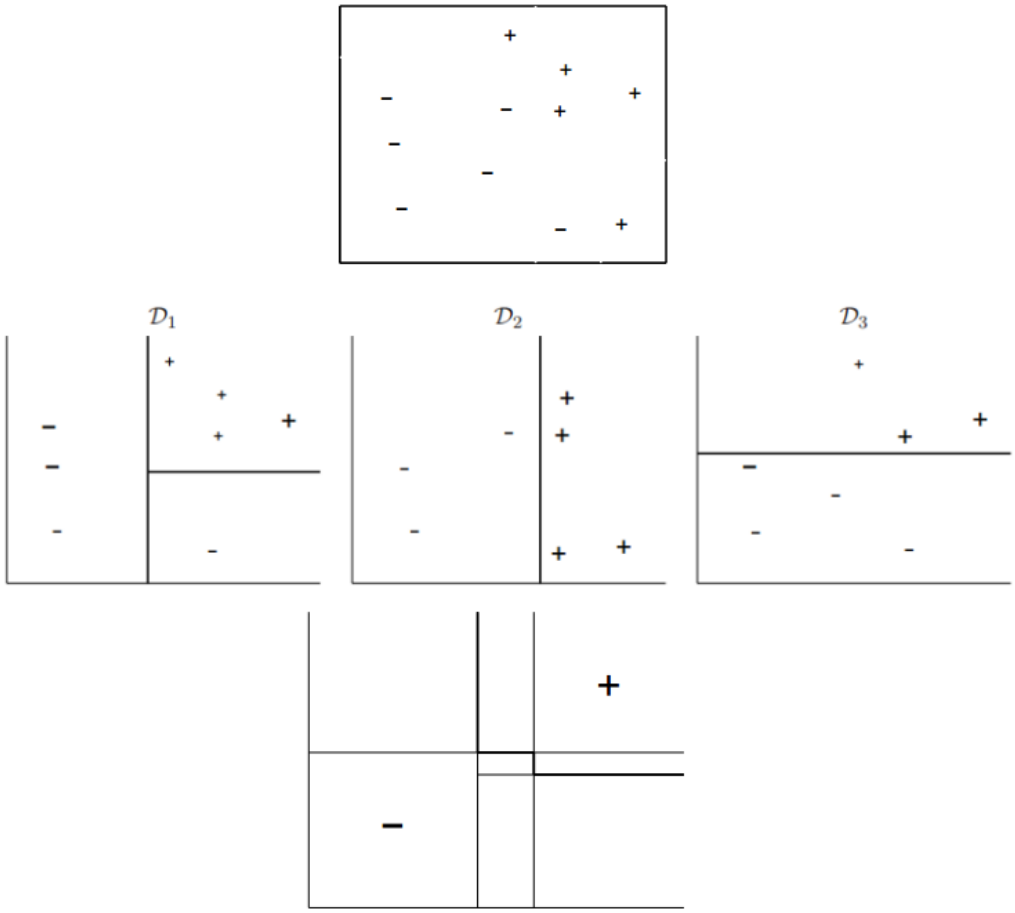
Figure courtesy: http://quantdare.com/

Bagging

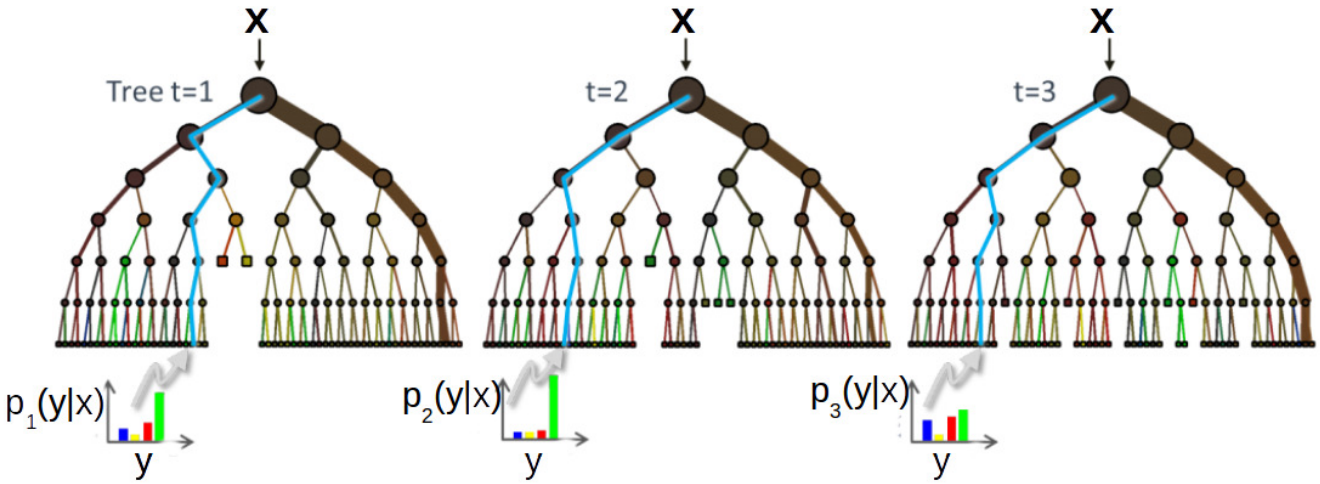
- Bagging stands for Bootstrap Aggregation
- Takes original data set D with N training examples
- Creates M copies $\{\tilde{D}_m\}_{m=1}^{M}$
  - Each $\tilde{D}_m$ is generated from D by sampling with replacement
  - Each data set $\tilde{D}_m$ has the same number of examples as in data set D (though not all are unique)
  - These data sets are reasonably different from each other (since only about 63% of the original/unique examples appear in any of these data sets)
- Train models $h_1, \ldots, h_M$ using $\tilde{D}_1, \ldots, \tilde{D}_M$, respectively
- Use an averaged model $h = \frac{1}{M} \sum_{m=1}^{M} h_m$ as the final model
- Useful for models with high variance and noisy data

Bagging: Illustration

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



Random Forests



- An ensemble of decision tree (DT) classifiers
- Uses bagging on features (each DT will use a random set of features)
  - 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
- Each DT will split the training data differently at the leaves
- Prediction for a test example votes on/averages predictions from all the DTs

Figure courtesy: "Decision Forests" by Criminisi *et al*

Boosting

- The basic idea
  - Take a weak learning algorithm
    - Only requirement: Should be 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: Most models naturally allow learning from importance-weighted examples (or can be easily modified to do that)

The AdaBoost Algorithm

- Given: Training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ with $y_n \in \{-1, +1\}$, $\forall n$
- Initialize weight of each example $(\boldsymbol{x}_n, y_n)$: $D_1(n) = 1/N$, $\forall n$
- For round $t = 1 : T$
  - Learn a weak $h_t(\boldsymbol{x}) \rightarrow \{-1, +1\}$ 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

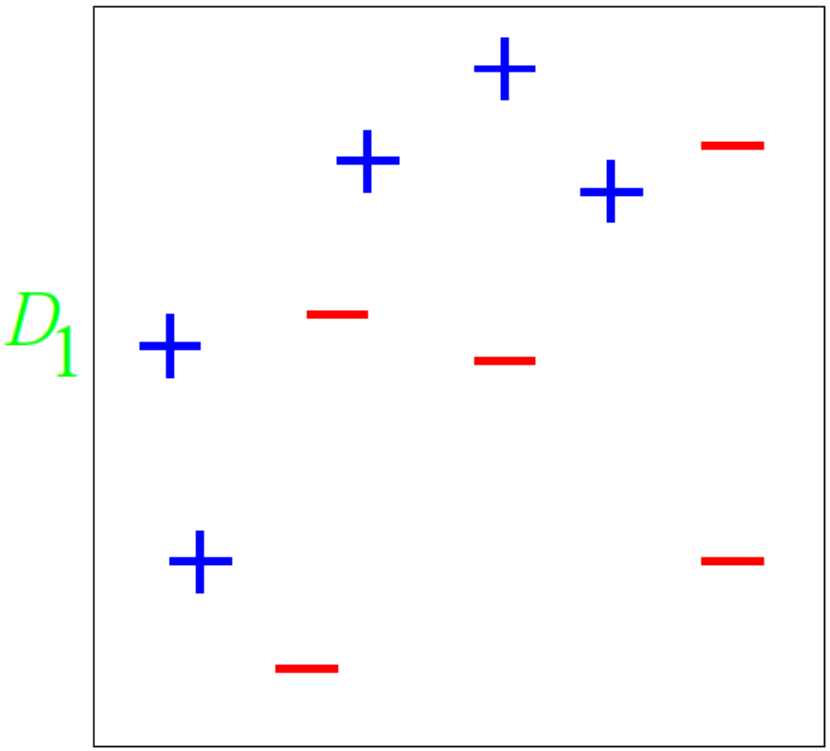
$$\begin{aligned} D_{t+1}(n) &\propto \begin{cases} D_t(n) \times \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) = y_n \quad \text{(correct prediction: decrease weight)} \\ D_t(n) \times \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_n) \neq y_n \quad \text{(incorrect prediction: increase weight)}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

  - 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: Example

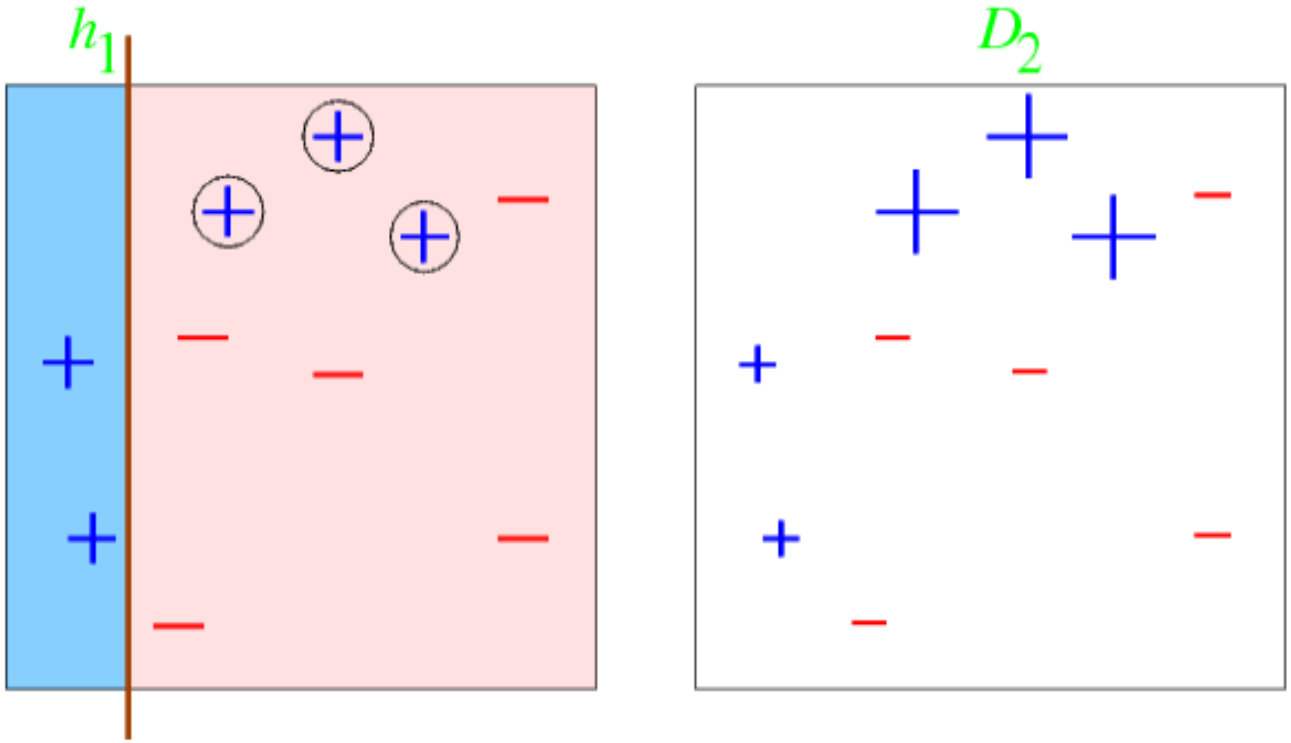
Consider binary classification with 10 training examples

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



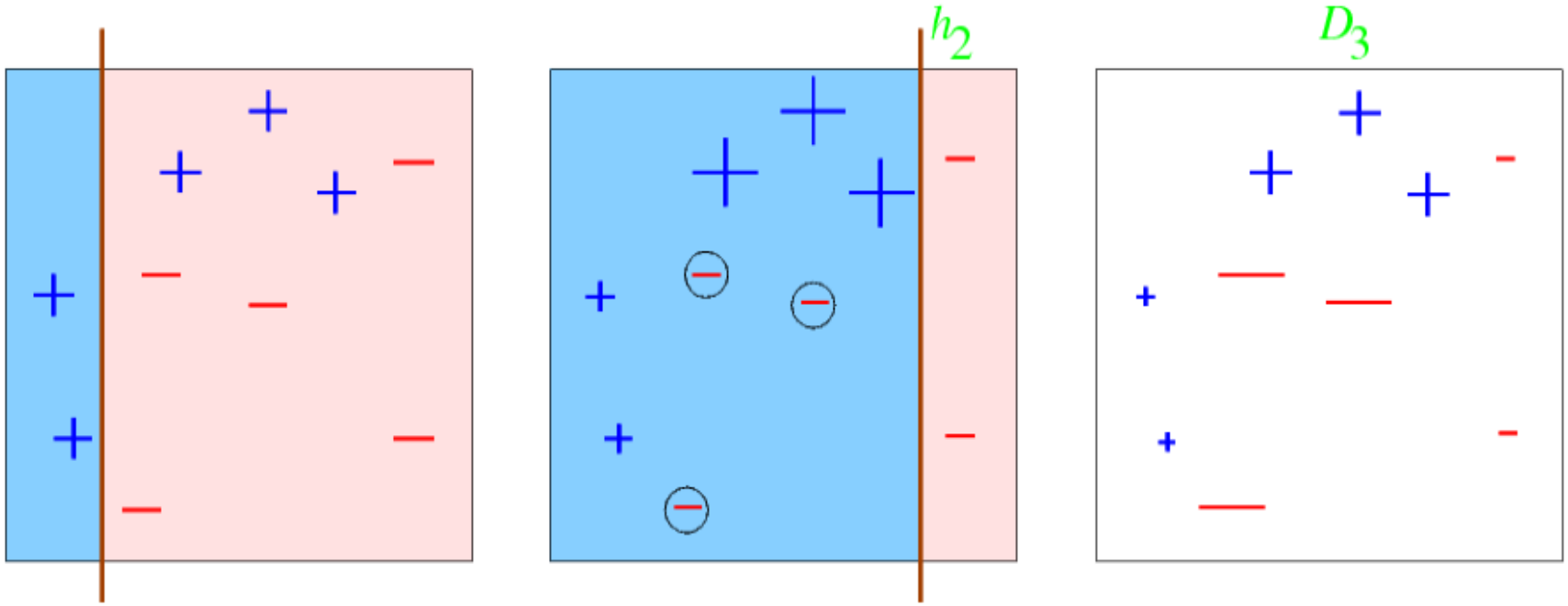
Each of our weak classifers will be an **axis-parallel linear classifier**

After Round 1



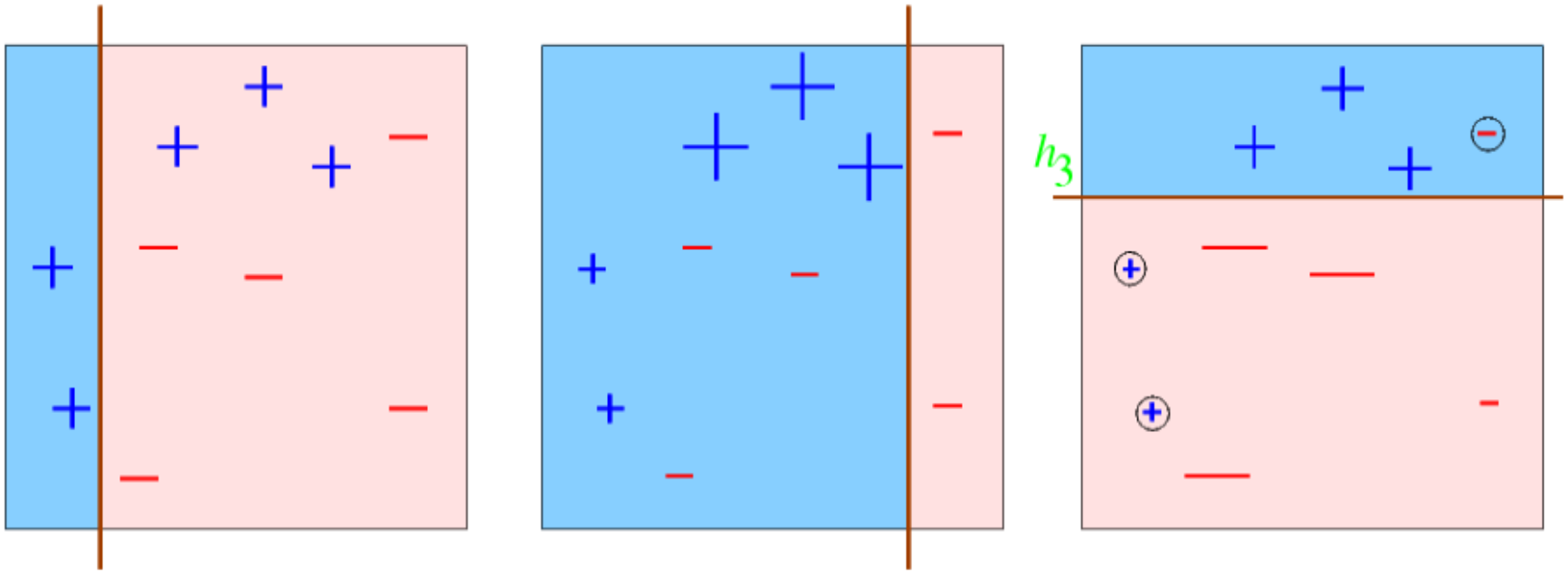
- Error rate of h_1: $\epsilon_1 = 0.3$; weight of h_1: $\alpha_1 = \frac{1}{2}\ln((1-\epsilon_1)/\epsilon_1) = 0.42$
- Each misclassified point upweighted (weight multiplied by $\exp(\alpha_2)$)
- Each correctly classified point downweighted (weight multiplied by $\exp(-\alpha_2)$)

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 $\exp(-\alpha_2)$)

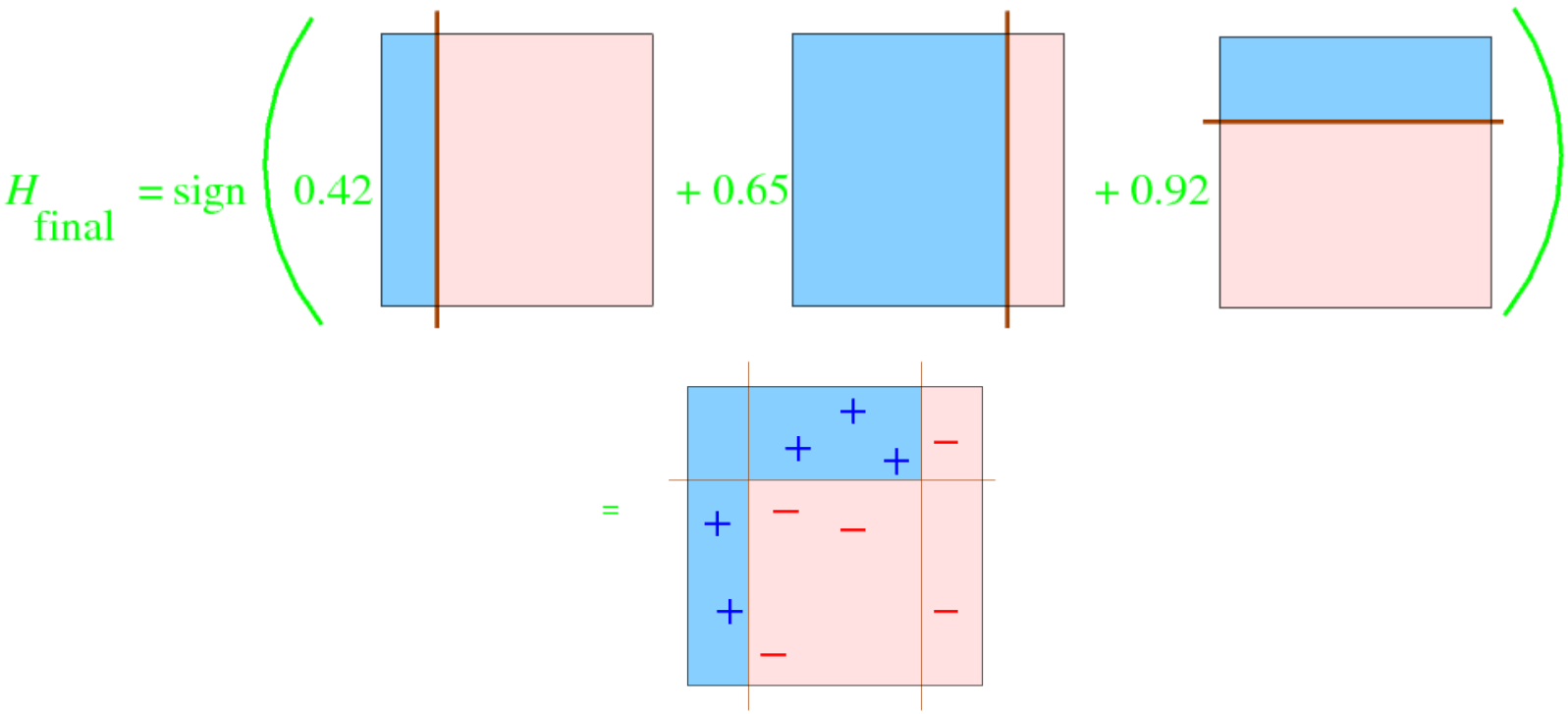
After Round 3



- 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

Final Classifier

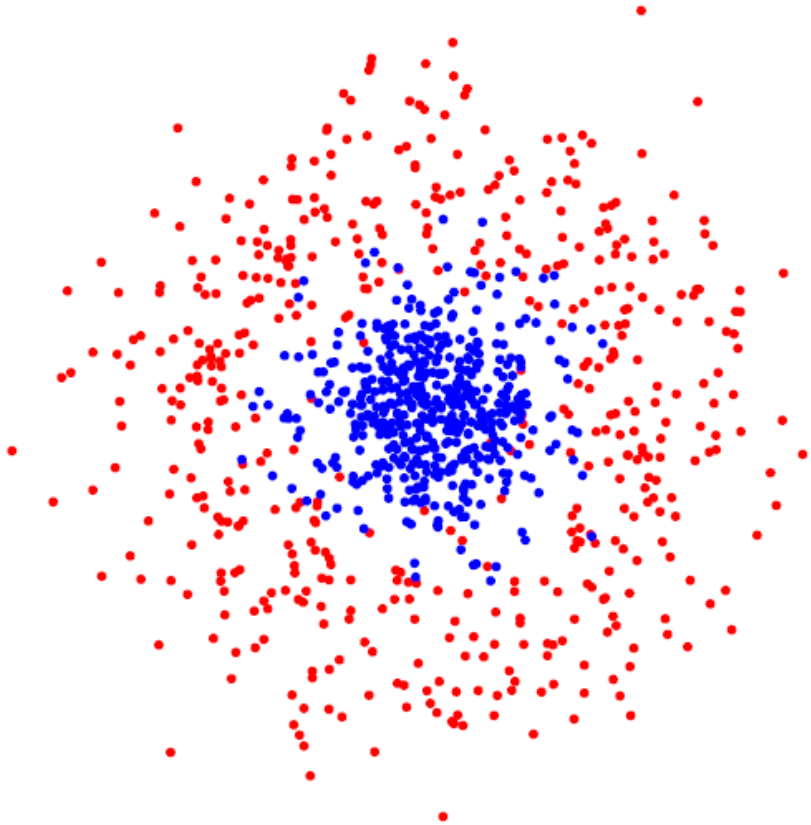
- Final classifier is a weighted linear combination of all the classifiers
- Classifier h_i gets a weight α_i



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

Another Example

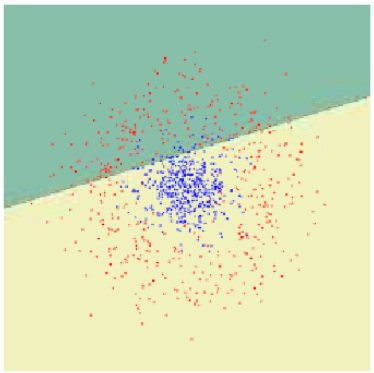
- Given: A nonlinearly separable dataset
- We want to use Perceptron (linear classifier) on this data

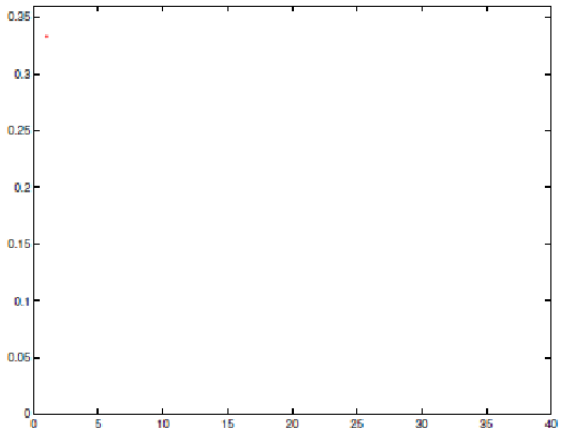


AdaBoost: Round 1

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

$t = 1$

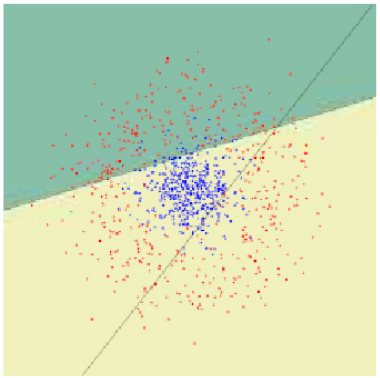


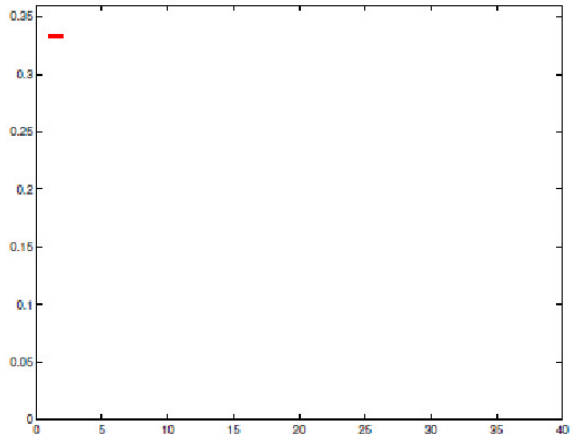


AdaBoost: Round 2

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

$t = 2$

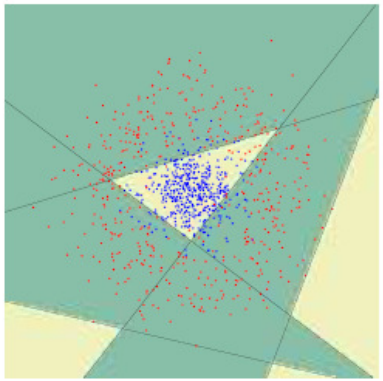


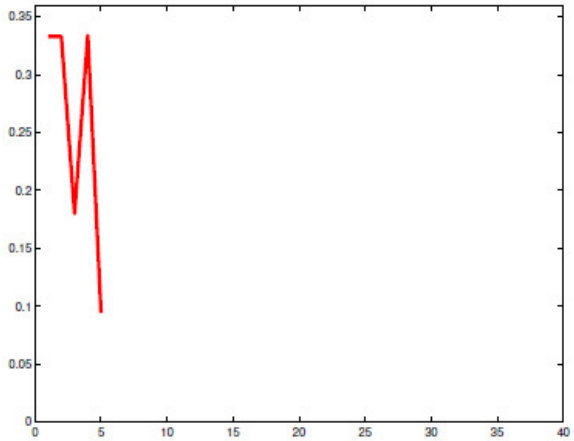


AdaBoost: Round 5

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

$t = 5$

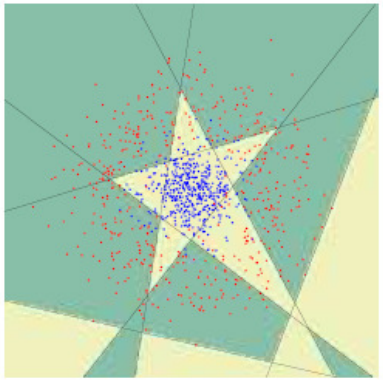


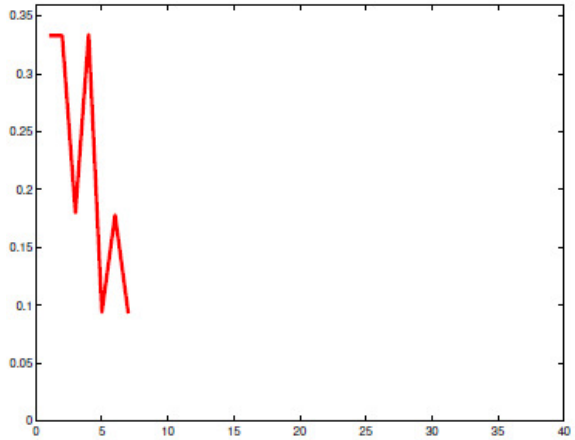


AdaBoost: Round 7

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

$t = 7$

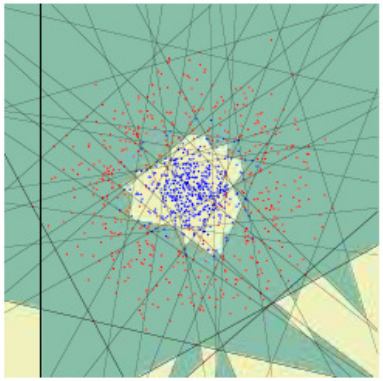


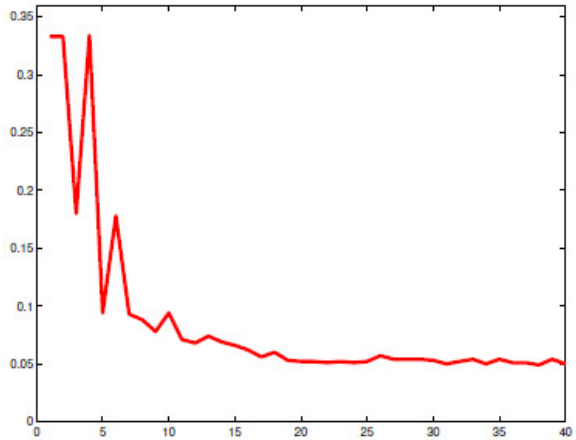


AdaBoost: Round 40

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

$t = 40$





Boosted Decision Stumps = Linear Classifier

- A decision stump (DS) is a tree with a single node (testing the value of a single feature, say the d-th feature)
- Suppose each example $\boldsymbol{x}$ has D binary features $\{x_d\}_{d=1}^D$, with $x_d \in \{0, 1\}$ and the label y is also binary, i.e., $y \in \{-1, +1\}$
- The DS (assuming it tests the d-th feature) will predict the label as as

$$h(\boldsymbol{x}) = s_d(2x_d - 1) \qquad \text{where } s_d \in \{-1, +1\}$$

- Suppose we have T such decision stumps $h_1, \ldots, h_T$, testing feature number $i_1, \ldots, i_T$, respectively, i.e., $h_t(\boldsymbol{x}) = s_{i_t}(2x_{i_t} - 1)$
- The boosted hypothesis $H(\boldsymbol{x}) = \text{sgn}(\sum_{t=1}^T \alpha_t h_t(\boldsymbol{x}))$ can be written as

$$H(\boldsymbol{x}) = \text{sgn}(\sum_{t=1}^T \alpha_{i_t} s_{i_t}(2x_{i_t} - 1)) = \text{sgn}(\sum_{t=1}^T 2\alpha_{i_t} s_{i_t} x_{i_t} - \sum_{t=1}^T \alpha_{i_t} s_{i_t}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

where $w_d = \sum_{t:i_t=d} 2\alpha_t s_t$ and $b = -\sum_t \alpha_t s_t$

Boosting: Some Comments

- For AdaBoost, for each model's error $\epsilon_t = 1/2 - \gamma_t$, the training error consistently gets better with rounds

$$\text{train-error}(H_{final}) \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2)$$

- Boosting algorithms can be shown to be minimizing a loss function
  - E.g., AdaBoost has been shown to be minimizing an exponential loss

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

  where $H(\boldsymbol{x}) = \text{sign}(\sum_{t=1}^{T}\alpha_t h_t(\boldsymbol{x}))$, given weak base classifiers $h_1, \ldots, h_T$. The minimizer α_t's turn out to be what we saw in AdaBoost description!

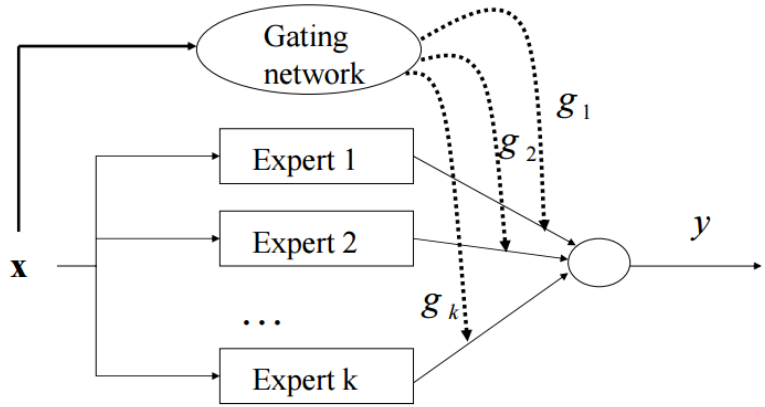
- Boosting in general can perform badly if some examples are outliers

Bagging vs Boosting

- No clear winner; usually depends on the data
- Bagging is computationally more efficient than boosting (note that bagging can train the M models in parallel, boosting can't)
- Both reduce variance (and overfitting) by combining different models
  - The resulting model has higher stability as compared to the individual ones
- Bagging usually can't reduce the bias, boosting can (note that in boosting, the training error steadily decreases)
- Bagging usually performs better than boosting if we don't have a high bias and only want to reduce variance (i.e., if we are overfitting)

Some Other Ensemble Methods

- Mixture of Experts: Another very powerful ensemble learning approach
  - Consist of a set of "experts" (each expert is a prediction model)
  - Each of these experts may be good in specific regions of the input space



  - Given an example $\boldsymbol{x}$, a "gating network" $g(\boldsymbol{x}) = [g_1(\boldsymbol{x})\ g_2(\boldsymbol{x})\ \ldots\ g_K(\boldsymbol{x})]$ gives the weight of each of the experts for making the prediction for $\boldsymbol{x}$
  - Mixture of experts models can be formulated as a probabilistic model and parameters (experts, gating network params, etc.) can be learned using EM

Summary

- Complex learning problems can often be solved by effectively combining multiple weak model via ensemble learning methods
- Looked at some representative ensemble learning methods
  - Simple ones: Voting/averaging or stacking
  - Bagging: Random Forests
  - Boosting: Adaboost
  - Mixture of Experts
- These models help reduce bias, variance, or both, leading to better test accuracies (and can often also be significantly faster than complex models)