# CSCE 478/878 Lecture 8: Combining Classifiers: Weighted Majority, Boosting, and Bagging

Stephen D. Scott
(Adapted from Tom Mitchell's slides and Rob Schapire)



## Outline

- Combining classifiers to improve performance
- Combining arbitrary classifiers: Weighted Majority algorithm
- Combining while learning:
  - Boosting
  - Bagging



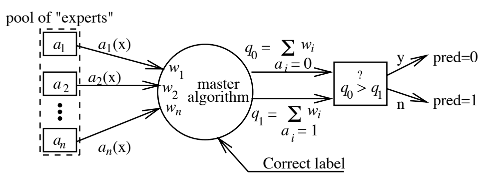
## Combining Classifiers

- Sometimes a single classifier (e.g. neural network, decision tree) won't perform well, but a weighted combination of them will
- Each classifier (or expert) in the pool has its own weight
- When asked to predict the label for a new example, each expert makes its own prediction, and then the master algorithm combines them using the weights for its own prediction (i.e. the "official" one)
- If the classifiers themselves cannot learn (e.g. heuristics) then the best we can do is to learn a good set of weights
- If we are using a learning algorithm (e.g. ANN, dec. tree), then we can rerun the algorithm on different subsamples of the training set and set the classifiers' weights during training



## Weighted Majority Algorithm (WM)
[Sec. 7.5.4]

$A$ = pool of fixed "experts"






## Weighted Majority Algorithm (WM)
(cont'd)

$a_i$ is $i$th prediction algorithm in pool $A$ of algorithms; each algorithm is arbitrary function from $X$ to $\{0, 1\}$

$w_i$ is weight the master alg associates with $a_i$

$\beta \in [0, 1)$ is parameter

- $\forall\, i$ set $w_i \leftarrow 1$
- For each training example (or trial) $\langle x, c(x) \rangle$
  - Set $q_0 \leftarrow q_1 \leftarrow 0$
  - For each algorithm $a_i$
    - If $a_i(x) = 0$ then $q_0 \leftarrow q_0 + w_i$
      else $q_1 \leftarrow q_1 + w_i$
  - If $q_1 > q_0$ then predict 1 for $c(x)$, else predict 0 (case for $q_1 = q_0$ is arbitrary)
  - For each $a_i \in A$
    - If $a_i(x) \neq c(x)$ then $w_i \leftarrow \beta\, w_i$

Setting $\beta = 0$ yields Halving Algorithm over $A$



## Weighted Majority
Mistake Bound (On-Line Model)

- Let $a_{opt} \in A$ be expert that makes fewest mistakes on arbitrary sequence $S$ of exs; let $k$ = number of mistakes $a_{opt}$ makes
- Let $\beta = 1/2$ and $W_t = \sum_{i=1}^{n} w_{i,t}$ = sum of wts before trial $t$ ($W_1 = n$)
- On trial $t$ such that WM makes a mistake, the total weight reduced is
$$W_t^{mis} = \sum_{a_i(x_t) \neq c(x_t)} w_i \geq W_t/2$$
so
$$W_{t+1} = \left(W_t - W_t^{mis}\right) + W_t^{mis}/2 = W_t - W_t^{mis}/2 \leq 3W_t/4$$
- After seeing all of $S$, $w_{opt,|S|+1} = (1/2)^k$ and $W_{|S|+1} \leq n(3/4)^M$ where $M$ = total number of mistakes, yielding
$$\left(\frac{1}{2}\right)^k \leq n\left(\frac{3}{4}\right)^M,$$
so
$$\boxed{M \leq \frac{k + \log_2 n}{-\log_2(3/4)} \leq 2.4\,(k + \log_2 n)}$$

## Weighted Majority

Mistake Bound (cont'd)

- Thus for any arbitrary sequence of examples, WM guaranteed to not perform much worse than best expert in pool plus log of number of experts
- Other results:
  - Bounds hold for general values of $\beta \in [0, 1)$
  - Better bounds hold for more sophisticated algorithms, but only better by a constant factor (worst-case lower bound: $\Omega(k + \log n)$)
  - Get bounds for real-valued labels and predictions
  - Can track shifting concept, i.e. where best expert can suddenly change in $S$; key: don't let any weight get too low relative to other weights, i.e. don't overcommit



## Bagging Classifiers

[Breiman, ML Journal, '96]

Bagging = Bootstrap aggregating

Bootstrap sampling: given a set $D$ containing $m$ training examples:

- Create $D_i$ by drawing $m$ examples uniformly at random with replacement from $D$
- Expect $D_i$ to omit $\approx 37\%$ of examples from $D$

Bagging:

- Create $k$ bootstrap samples $D_1, \ldots, D_k$
- Train a classifier on each $D_i$
- Classify new instance $x \in X$ by majority vote of learned classifiers (equal weights)

Result: An ensemble of classifiers



## Bagging Experiment

[Breiman, ML Journal, '96]

Given sample $S$ of labeled data, Breiman did the following 100 times and reported avg:

1. Divide $S$ randomly into test set $T$ (10%) and training set $D$ (90%)
2. Learn decision tree from $D$ and let $e_S$ be its error rate on $T$
3. Do 50 times: Create bootstrap set $D_i$ and learn decision tree (so ensemble size = 50). Then let $e_B$ be the error of a majority vote of the trees on $T$

Results

| Data Set | $\bar{e}_S$ | $\bar{e}_B$ | Decrease |
|---|---|---|---|
| waveform | 29.0 | 19.4 | 33% |
| heart | 10.0 | 5.3 | 47% |
| breast cancer | 6.0 | 4.2 | 30% |
| ionosphere | 11.2 | 8.6 | 23% |
| diabetes | 23.4 | 18.8 | 20% |
| glass | 32.0 | 24.9 | 27% |
| soybean | 14.5 | 10.6 | 27% |



## Bagging Experiment

(cont'd)

Same experiment, but using a nearest neighbor classifier (Chapt. 8), where prediction of new example $x$'s label is that of $x$'s nearest neighbor in training set, where distance is e.g. Euclidean distance

Results

| Data Set | $\bar{e}_S$ | $\bar{e}_B$ | Decrease |
|---|---|---|---|
| waveform | 26.1 | 26.1 | 0% |
| heart | 6.3 | 6.3 | 0% |
| breast cancer | 4.9 | 4.9 | 0% |
| ionosphere | 35.7 | 35.7 | 0% |
| diabetes | 16.4 | 16.4 | 0% |
| glass | 16.4 | 16.4 | 0% |

What happened?



## When Does Bagging Help?

When learner is unstable, i.e. if small change in training set causes large change in hypothesis produced

- Decision trees, neural networks
- Not nearest neighbor

Experimentally, bagging can help substantially for unstable learners; can somewhat degrade results for stable learners



## Boosting Classifiers

[Freund & Schapire, ICML '96; many more]

Similar to bagging, but don't always sample uniformly; instead adjust resampling distribution over $D$ to focus attention on previously misclassified examples

Final classifier weights learned classifiers, but not uniform; instead weight of classifier $h_t$ depends on its performance on data it was trained on

Repeat for $t = 1, \ldots, T$:

1. Run learning algorithm on examples randomly drawn from training set $D$ according to distribution $\mathcal{D}_t$ ($\mathcal{D}_1 =$ uniform)
2. Output of learner is hypothesis $h_t : X \to \{-1, +1\}$
3. Compute $error_{\mathcal{D}_t}(h_t)$ = error of $h_t$ on examples drawn according to $\mathcal{D}_t$ (can compute exactly)
4. Create $\mathcal{D}_{t+1}$ from $\mathcal{D}_t$ by increasing weight of examples that $h_t$ mispredicts

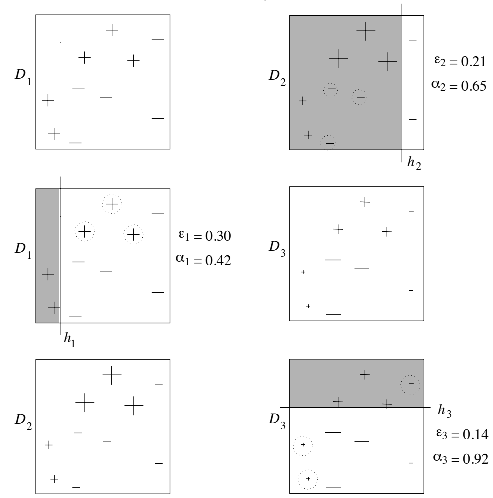
Final classifier is weighted combination of $h_1, \ldots, h_T$, where $h_t$'s weight depends on its error w.r.t. $\mathcal{D}_t$

## Boosting
(cont'd)

- <u>Preliminaries</u>: $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)\}$, $y_i \in \{-1, +1\}$, $\mathcal{D}_t(i)$ = weight of $(\vec{x}_i, y_i)$ under $\mathcal{D}_t$
- <u>Initialization</u>: $\mathcal{D}_1(i) = 1/m$
  (in general, require $\forall t$, $\sum_{i=1}^{m} \mathcal{D}_t(i) = 1$)
- <u>Error Computation</u>: $\epsilon_t = \Pr_{\mathcal{D}_t}[h_t(\vec{x}_i) \neq y_i]$
  (easy to do since we know $\mathcal{D}_t$)
- <u>If $\epsilon_t > 1/2$ then halt; else</u>:
- <u>Weighting Factor</u>: $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  (grows as $\epsilon_t$ decreases)
- <u>Update</u>: $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i) \exp\left(-\alpha_t\, y_i\, h_t(\vec{x}_i)\right)}{\underbrace{Z_t}_{\text{normalization factor}}}$
  (increase weight of mispredicted examples, decrease wt of correctly predicted exs)
- <u>Final Hypothesis</u>: $H(\vec{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t\, h_t(\vec{x})\right)$

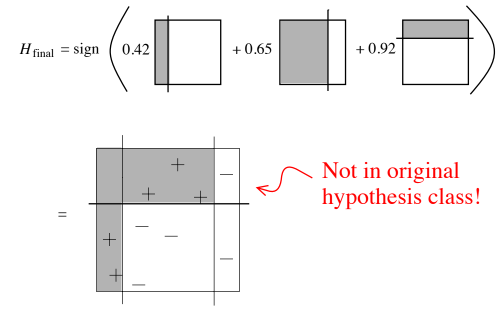


## Boosting
Example






## Boosting
Example (cont'd)




Other advantages to ensembles (boost/bag):

- Helps with problem of choosing one of several consistent hypotheses
- Compensates for imperfect search algorithms (e.g. it is hard to find smallest decision tree or a consistent ANN)



## Boosting
Miscellany

- If each $\epsilon_t < 1/2 - \gamma_t$, error of $H(\cdot)$ on $D$ drops exponentially in $\sum_{t=1}^{T} \gamma_t$
- Can also bound generalization error of $H(\cdot)$ <u>independent of $T$</u>
- Also successful empirically on neural network and decision tree learners
  - Empirically, generalization sometimes improves if training continues after $H(\cdot)$'s error on $D$ drops to 0 [cf. generalization error's independence of T]
  - Contrary to intuition: would expect overfitting
  - Related to increasing the combined classifier's <u>margin</u> (confidence in prediction)
- Can apply to labels that are multi-valued using e.g. <u>error-correcting output codes</u>