Lecture 22: Ensemble Learning, Bagging and Boosting
Instructor: Prof. Ganesh Ramakrishnan
An ensemble is simply a collection of models that are all trained to perform the same task.

An ensemble can consist of many different versions of the same model, or many different types of models.

The final output for an ensemble of classifiers is typically obtained through a (weighted) average or vote of the predictions of the different models in the ensemble.

An ensemble of different models that all achieve similar generalization performance often outperforms any of the individual models.

**Question:** How is this possible?
Example: Netflix Prize (2009)

Winning team used Ensemble of 450+ different models.
Suppose we have an ensemble of binary classification functions $f_k(x)$ for $k = 1, \ldots, K$.

Suppose that on average they have the same expected error rate

$$\epsilon = E_{p(x,y)}[y \neq f_k(x)] < 0.5,$$

but that the errors they make are independent.

The intuition is that the majority of the $K$ classifiers in the ensemble will be correct on many examples where any individual classifier makes an error.

A simple majority vote can significantly improve classification performance by decreasing variance in this setting.

**Question:** How can we come up with such an ensemble?
Suppose we have an ensemble of binary classification functions $f_k(x)$ for $k = 1, \ldots, K$.

Suppose that on average they have the same expected error rate

$$\epsilon = E_{p(x,y)}[y \neq f_k(x)] < 0.5,$$

but that the errors they make are independent.

The intuition is that the majority of the $K$ classifiers in the ensemble will be correct on many examples where any individual classifier makes an error.

A simple majority vote can significantly improve classification performance by decreasing variance in this setting.

**Question:** How can we come up with such an ensemble?

1. Eg 1: Dropout in Deep NN - Performs Regularization
2. Eg 2: Bagging (motivated by Independent Datasets) - Balances Bias & Variance
3. Eg 3: Boosting - Balances Bias & Variance
Eg 1: Dropout in Deep NN & Ensemble Learning (Section 7.12)

- Masking binary variable $\mu^l_i \in \{0, 1\}$ sampled independently from each other
- Probability $\Pr(\mu^l_i = 1) = \beta$ is a hyperparameter $\beta$. Usually $\beta = 0.5$ for the hidden layers ($l > 1$) and $\beta = 0.8$ for the input ($l = 1$).
- The resulting $\sigma^l_j$ is multiplied by $\frac{1}{\beta}$ to ensure probabilistic semantics and then multiplied with its own mask $\mu^l_i$
- Equivalent to randomly selecting one of the sub-networks from the complete network and running forward propagation through it ⇒

$\mu^l_1 \sigma^l_1$  
$\mu^l_2 \sigma^l_2$  
$\mu^l_i \sigma^l_i$  
$\mu^l_{s_{l-1}} \sigma^l_{s_{l-1}}$

$w^l_{1j}$
$w^l_{2j}$
$w^l_{ij}$
$w^l_{s_{l-1}j}$

$\frac{1}{\beta} \sigma^l_j \mu^l_i$
Eg 1: Dropout in Deep NN & Ensemble Learning (Section 7.12)

- Masking binary variable $\mu^l_i \in \{0, 1\}$ sampled independently from each other.
- Probability $\text{Pr}(\mu^l_i = 1) = \beta$ is a hyperparameter $\beta$. Usually $\beta = 0.5$ for the hidden layers ($l > 1$) and $\beta = 0.8$ for the input ($l = 1$).
- The resulting $\sigma^l_j$ is multiplied by $\frac{1}{\beta}$ to ensure probabilistic semantics and then multiplied with its own mask $\mu^l_i$.
- Equivalent to randomly selecting one of the sub-networks from the complete network and running forward propagation through it $\Rightarrow$ Final network is Combination (Ensemble) of exponentially large number of similar neural networks.
- Dropout training $\Rightarrow$ Minimizing expected value of error wrt random variable $\mu$: $E_\mu[E(w, b)]$
Eg 1: Ensemble Learning through Dropouts

Note: This is an ensemble (with Arithmetic mean) of exponentially large number of neural networks. See Section 7.12 for other means such as Geometric mean.

(Source: "Deep Learning" by Yoshua Bengio Ian J. Goodfellow Aaron Courville)
Backpropagation with Dropouts

\[ \frac{\partial E}{\partial \sigma^l_j} = \begin{cases} \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial \sigma^l_{p+1}} \mu^l_{p+1} \sigma^l_{p+1} w^l_{jp} & \text{if } \mu^l_j = 1 \\ 0 & \text{otherwise} \end{cases} \]

\((l - 1)^{th}\) layer

Based on bookkeeping of weights that contributed in forward prop.
Eg 2: Independent Training Sets

- Suppose we collect multiple independent training sets $Tr_1, \ldots, Tr_K$ and use each of these training sets to train a different instance of the same classifier obtaining $K$ classification functions $f_1(x), \ldots, f_K(x)$.

- Classifiers trained in this way are guaranteed to make independent errors on test data.

- If the expected error of each classifier is less than 0.5, then the weighted majority vote is guaranteed to reduce the expected generalization error.

**Question:** What is the weakness of this approach?

To get “truly independent datasets” we might need to compromise on size of each dataset. Higher variance in size might become too small.
Eg 2: **Bagging**, i.e., (Bootstrap AGgregation)

- **Bootstrap AGgregation** or **Bagging** is an approximation to the previous method that takes a single training set $Tr$ and randomly sub-samples from it $K$ times (with replacement) to form $K$ training sets $Tr_1, ..., Tr_K$.

- Each of these training sets is used to train a different instance of the same classifier $\Rightarrow K$ classification functions $f_1(x), ..., f_K(x)$.

- The errors won't be totally independent because the data sets aren't independent, but the random re-sampling usually introduces enough diversity to decrease the variance and give improved performance.

 Ideally: Each base classifier $f_i(x)$ should have low bias ($Tr_i$ sampling might increase variance in any case)
Eg 2: Bagging and Random Forests

- Bagging: Particularly useful for high-variance, high-capacity models.
- Historically, closely associated with decision tree models (covered in class)
- Random forest classifier: popular extension of bagged trees
  - The random forests algorithm further decorrelates the learned trees by only considering a random sub-set of the available features when deciding which variable to split on at each node in the tree.
- Lab 8, Q2, Task 1 associates Bagging with Perceptron (as weak learner)
Eg 2: Bagging on Deep NNs

- Uniformly at random (with replacements), sample subsets $\mathcal{D}_s \subseteq \mathcal{D}$ of the training data, $\Phi_s \subseteq \Phi$ of the input layer features and construct (deep) model $N_s$ for each such random subset.
- Bagging Algorithm:
Eg 2: Bagging on Deep NNs

- Uniformly at random (with replacements), sample subsets $D_s \subseteq D$ of the training data, $\Phi_s \subseteq \Phi$ of the input layer features and construct (deep) model $N_s$ for each such random subset.

- **Bagging Algorithm:** For $s = 1$ to $B$ repeat:
  1. **Bagging:** Draw a bootstrap sample $D_s$ of size $n_s$ from the training data $D$ of size $n$
  2. Learn a random (deep) model $N_s$ on $D_s$

- **Output:** Ensemble of (deep) models $\{N_s\}_{1}^{B}$
Eg 2: Bagging applied to Query (Test) data

Ensemble of Models \( \{M_s\}_1^B \)

- Consider \( \Pr_n(c \mid x) \) for each model \( n \in N \) for each class \( c = [1..K] \)
- Decision for a new test point \( x \):
Eg 2: Bagging applied to Query (Test) data

Ensemble of Models \( \{M_s\}_1^B \)

- Consider \( \Pr_n(c \mid x) \) for each model \( n \in N \) for each class \( c = [1..K] \)

- Decision for a new test point \( x \): 
  \[
  \Pr(c \mid x) = \frac{1}{N} \sum_{n=1}^{N} \Pr_n(c \mid x)
  \]

- For \( m \) data points, with \( |N| = \sqrt{m} \), consistency results have been proved

\(^a\)Convergence of loss to Bayes Risk, \( i.e. \), expectation of loss under distribution of parameters \( \theta \)
Eg 2: Bagging – Balancing Bias and Variance

Ensemble of Models \( \{M_s\}_1^B \)

- Decision for a new test point \( x \): \( \Pr(c \mid x) = \frac{1}{N} \sum_{n=1}^{N} \Pr_n(c \mid x) \)
- Each single (deep) model, viewed as an estimator of the *ideal* model has high variance, with very less bias (assumptions)
- But since the models \( N_i \) and \( N_j \) are uncorrelated, when decision is averaged out across them, it tends to be very accurate.
Eg 3: Boosting

- Boosting is based on iteratively re-weighting the data set instead of randomly resampling it.

- **Main idea:**
  - Up-weight the importance of data cases that are missclassified by the classifiers currently in the ensemble
  - Next add a next classifier that will focus on data cases that are causing the errors.

- Can be shown to decrease error, assuming that base classifier can always achieve an error of less than 0.5 on any data sample

- AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier (covered in class)

- Lab 8, Q2, Task 2 associates Boosting with Perceptron (as a weak learner)
From **Bagging** to **Boosting**
Weak Models: From **Bagging** to **Boosting**

**Bagging:** Ensemble of **Independently Weakly Learnt** Models (Eg: Trees \( \{ T_s \}_1^B \)):

\[
\Pr(c | x) = \frac{1}{|B|} \sum_{t=1}^B \Pr_t(c | x)
\]

**Expected error** \( < \frac{1}{k} \)
Weak Models: From **Bagging** to **Boosting**

**Bagging:** Ensemble of **Independently Weakly Learnt** Models (Eg: Trees $\{T_s\}_1^B$):

$$\Pr(c | x) = \frac{1}{|B|} \sum_{t=1}^B \Pr_t(c | x)$$

**Boosting:** Wtd combinations of **Iteratively Weakly Learnt** Models (Eg: Trees $\{\alpha_t, T_t\}_1^B$):
Weak Models: From **Bagging** to **Boosting**

**Bagging**: Ensemble of **Independently Weakly Learnt** Models (Eg: Trees \( \{ T_s \}_{1}^{B} \)):

\[
Pr(c | x) = \frac{1}{|B|} \sum_{t=1}^{B} Pr_t(c | x)
\]

**Boosting**: Wtd combinations of **Iteratively Weakly Learnt** Models (Eg: Trees \( \{ \alpha_t, T_t \}_{1}^{B} \)):

\[
Pr(c | x) = \frac{1}{|B|} \sum_{t=1}^{B} \alpha_t Pr_t(c | x) \quad \text{where} \quad \alpha_t = \left( \frac{1}{2} \right) \ln \left( \frac{1 - err_t}{err_t} \right)
\]
Adaptive Boosting of Iteratively Learnt Weak Models

Error driven weighted linear combinations of models:
$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \text{err}_t}{\text{err}_t}\right)$$

$$z_{ti} = \text{fn} \left(\text{err}_t, \alpha_1, \ldots, \alpha_t\right)$$
Adaptive Boosting of Iteratively Learnt Weak Models

Error driven weighted linear combinations of models: \( \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \text{err}_t}{\text{err}_t} \right) \)

Reweighting of each data instance \( \mathbf{x}^{(i)} \) before learning the next model \( T_t \):
\[
\xi_i = \xi_i \exp \left( \alpha_t \delta \left( y^{(i)} \neq T_t \left( \mathbf{x}^{(i)} \right) \right) \right).
\]
Note that \( \text{err}_t = \frac{\sum_{i=1}^{m} \xi_i \delta \left( y^{(i)} \neq T_t \left( \mathbf{x}^{(i)} \right) \right)}{\sum_{i=1}^{m} \xi_i} \).
Initialize each instance weight $\xi_i = \frac{1}{m}$. For $t = 1$ to $B$ do:

1. Learn the $t^{th}$ model $T_t$ by weighing example $x^{(i)}$ by $\xi_i$

2. Compute the corresponding error on the training set $err_t = \frac{\sum_{i=1}^{m} \xi_i \delta(y^{(i)} \neq T_t(x^{(i)}))}{\sum_{i=1}^{m} \xi_i}$

3. Compute the error driven weighted linear factor for $T_t$:
   $\alpha_t = (1/2) \ln \left( (1 - err_t) / err_t \right)$

4. Reweigh each data instance $x^{(i)}$ before learning the next model:
   $\xi_i = \xi_i \exp \left( \alpha_t \delta \left( y^{(i)} \neq T_t(x^{(i)}) \right) \right)$. 

Lab: $T_t =$ Perceptron. Next class: $T_t =$ Decision Tree
Adaboost Algorithm: Motivation (Tutorial 10)

- Freund & Schapire, 1995: Converting a “weak” PAC\(^1\) learning algorithm that performs just slightly better than random guessing into one with arbitrarily high accuracy.
- Let \( C_t(x) = \sum_{j=1}^{t} \alpha_j T_j(x) \) be the boosted linear combination of classifiers until \( t^{th} \) iteration.
- Let the error to be minimized over \( \alpha_t \) be the sum of its exponential loss on each data point,

\[
E_t = \sum_{i=1}^{m} \delta (y^{(i)} \neq \text{sign} (C_t (x^{(i)}))) \leq \sum_{i=1}^{m} \exp (-y^{(i)} C_t (x^{(i)}))
\]

- Claim1: The error that is the sum of exponential loss on each data point is an upper bound on the simple sum of training errors on each data point
- Claim2: \( \alpha_t = (1/2) \ln \left( (1 - \text{err}_t)/\text{err}_t \right) \) actually minimizes this upper bound.
- Claim3: If each classifier is slightly better than random, that is if \( \text{err}_t < 1/K \), Adaboost achieves zero training error exponentially fast

\(^1\)http://web.cs.iastate.edu/~honavar/pac.pdf
AdaBoost Algorithm

1. **Input:** $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, Number of Iterations $T$
2. **Initialize:** $d_n^{(1)} = 1/N$ for all $n = 1, \ldots, N$
3. Do for $t = 1, \ldots, T$
   (a) Train classifier with respect to the weighted sample set $\{S, d^{(t)}\}$ and obtain hypothesis $h_t : x \mapsto \{-1, +1\}$, i.e. $h_t = \mathcal{L}(S, d^{(t)})$
   (b) Calculate the weighted training error $\varepsilon_t$ of $h_t$:

   $$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} I(y_n \neq h_t(x_n)) ,$$

   (c) Set:

   $$\alpha_t = \frac{1}{2} \log \frac{1 - \varepsilon_t}{\varepsilon_t}$$

   (d) Update weights:

   $$d_n^{(t+1)} = d_n^{(t)} \exp \{-\alpha_t y_n h_t(x_n)\} / Z_t ,$$

   where $Z_t$ is a normalization constant, such that $\sum_{n=1}^{N} d_n^{(t+1)} = 1$
4. **Break if** $\varepsilon_t = 0$ or $\varepsilon_t \geq \frac{1}{2}$ and set $T = t - 1$
5. **Output:** $f_T(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x)$
Example: AdaBoost

![Graph showing test classification error vs number of trees for different boosting depths and a random forest model. The graph illustrates how the error decreases as the number of trees increases.](image-url)