ADABOOST & ENSEMBLE LEARNING

Shingchern D. You

Numerical illustration of voting

- Given the following 1-D example (not linearly separable)

X =	0	1	2	3	4	5	6	7	8	9
d =	1	1	1	-1	-1	-1	1	1	1	-1

- 1st classifier

X =	0	1	2	3	4	5	6	7	8	9
h1 =	1	1	1	-1	-1	-1	-1	-1	-1	-1

Numerical illustration of voting

- 2nd classifier

X =	0	1	2	3	4	5	6	7	8	9
h2 =	1	1	1	1	1	1	1	1	1	-1

- 3rd classifier

X =	0	1	2	3	4	5	6	7	8	9
h3 =	-1	-1	-1	-1	-1	-1	1	1	1	1

Numerical illustration of voting

- Perform majority vote

X=	0	1	2	3	4	5	6	7	8	9
h1 =	1	1	1	-1	-1	-1	-1	-1	-1	-1
h2 =	1	1	1	1	1	1	1	1	1	-1
h3 =	-1	-1	-1	-1	-1	-1	1	1	1	1
H=	1	1	1	-1	-1	-1	1	1	1	-1

- All samples are correctly classified

Numerical illustration of voting

- Although each classifier is a linear weak classifier (i.e., low accuracy), combined classifier is a strong **nonlinear** classifier
- Explain why (**where do we introduce nonlinearity**?)
- AdaBoost follows the same idea, but with weighted sum instead of voting

AdaBoost algorithm

- Symbol definition
  - Samples $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in R^p$
  - Desired output $d_1, \dots, d_n \in \{-1, +1\}$
  - Initial weights $w_{1,1}, \dots, w_{n,1}$ set to $\frac{1}{n}$ (note: 2nd index is classifier index)
  - Weak classifiers $h_t \colon \boldsymbol{x}_k \rightarrow \{-1, +1\}$

AdaBoost algorithm

- For t = 1 … T
  - Find and save weak classifier $h_t(\boldsymbol{x})$ minimize
    $$\epsilon_t = \frac{1}{n}\sum_{k=1}^{n} w_{k,t}\ell(h_t(\boldsymbol{x}_k) \neq d_k)$$
    (Note: ϵ_t sometimes could be very small)
  - Update $\alpha_t \leftarrow \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - Update weights: $w_{k,t+1} \leftarrow w_{k,t}\exp(-\alpha_t h_t(\boldsymbol{x}_k) d_k)$
    $$w_{k,t+1} \leftarrow w_{k,t+1} / \sum_{k=1}^{n} w_{k,t+1}$$
  - For k = 1 …n : $H(\boldsymbol{x}_k) = \mathrm{sign}((\sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k))$
  - Stop condition: (1) No error on classifying training data

Or (2) Upper limit of iterations reached

Next classifier

- What is this part doing
  - Update $\alpha_t \leftarrow \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  - Update weights: $w_{k,t+1} \leftarrow w_{k,t}\exp(-\alpha_t h_t(\boldsymbol{x}_k) d_k)$
- To make classifier $h_t(\boldsymbol{x})$ has an error rate of 0.5 when classifying all training samples
- $h_{t+1}(\boldsymbol{x})$ can not receive any help from $h_t(\boldsymbol{x})$

AdaBoost algorithm

- Two classifiers in use
  - Use current weak classifier $h_t(\boldsymbol{x}_k)$ to update weights

$$w_{k,t+1} \leftarrow w_{k,t} \exp(-\alpha_t h_t(\boldsymbol{x}_k) d_k)$$

  - Use combined strong classifier $H(\boldsymbol{x}_k)$ to check error samples (but **cannot** be used for weights updating)

$$H(\boldsymbol{x}_k) = \text{sign}\left(\sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k)\right) == d_k?$$

AdaBoost algorithm

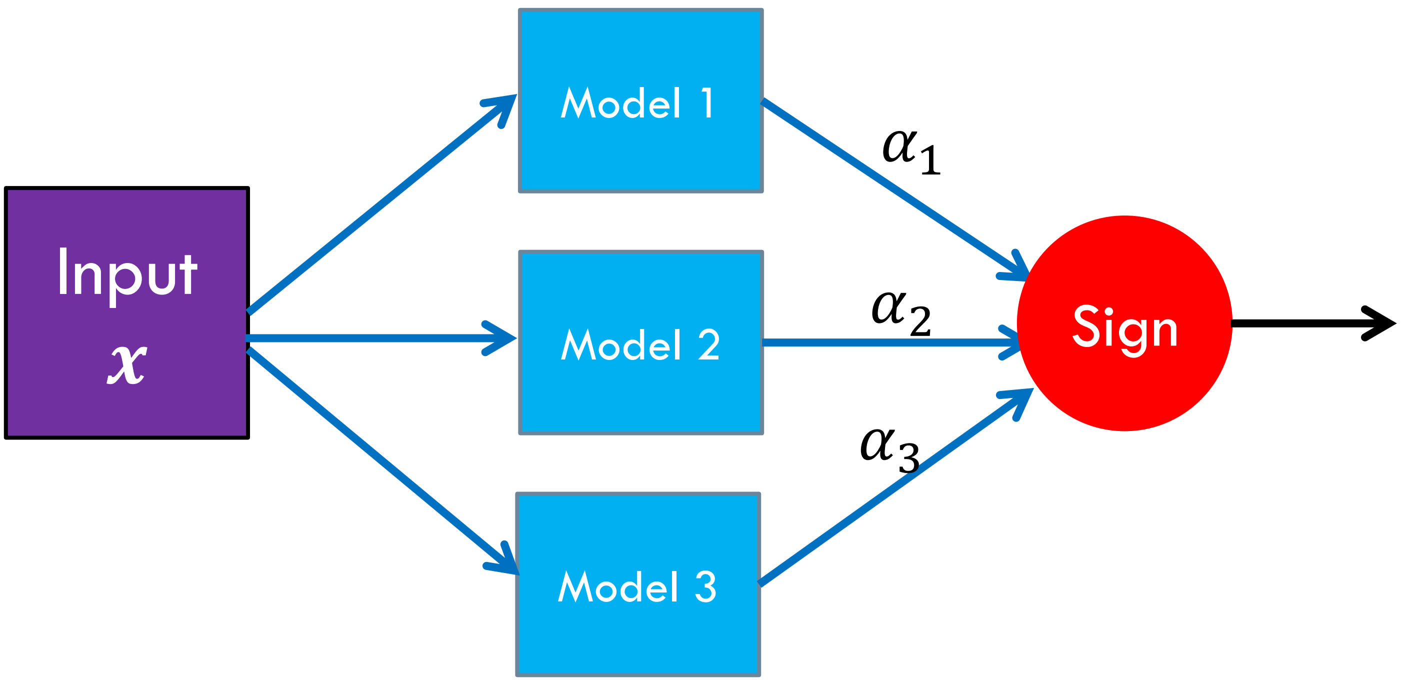
- For classification after training, use

$$H(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)\right)$$

- There are several variations on AdaBoost
- The version given here is from **Machine Learning in Action (a good book for engineers)**
- You can compare this algorithm with the one in textbook (original AdaBoost.M1)

Adaboost as weighted voting

- Meaning of $H(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)\right)$
- Recall each $h_t(\boldsymbol{x})$ has binary answer (± 1)



AdaBoost

- AdaBoost has solid theories behind it, to be briefly explained later
- Some key points in algorithm
  - Which weak classifier to use (should not be too strong)
  - How to perform **optimal decision** for weighted error in each weak classifier
  - Numerical issues (could be bad)
  - Very sensitive to noise and outliers in training set

Weak classifier

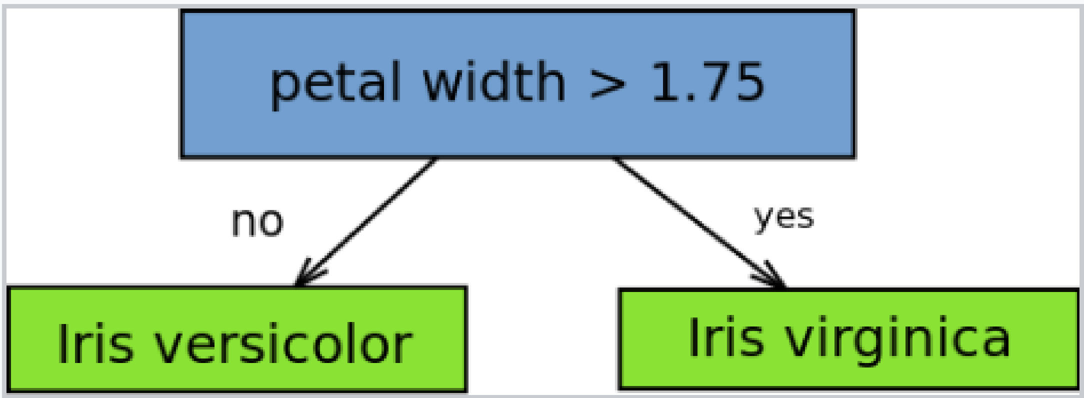
- In the algorithm, we need to search over all possible combinations of parameters to find optimal weighted error

$$\epsilon_t = \sum_{k=1}^{n} w_{k,t} \ell(h_t(\boldsymbol{x}_k) \neq y_k)$$

- Not easy with many classifiers (such as SVM)
- One widely used classifier is **decision stump**: making a decision on one feature only

Weak classifier

- Decision stump example (from wiki)



An example of a decision stump that discriminates between two of three classes of Iris flower data set: *Iris versicolor* and *Iris virginica*. The petal width is in centimetres. This particular stump achieves 94%

Weak classifier

- To find ϵ_t, we need to check all possible threshold values for all features
- Consider the following example with five training samples:
  - P1 = (1, 2.1), C = +1
  - P2 = (2, 1.1), C = +1
  - P3 = (1.3 1) ,C = -1
  - P4 = (1, 1), C = -1
  - P5 = (2, 1), C = +1

Weak classifier

- For 1st feature, we need to check (for example)

Threshold = {0.9, 1.1, 1.4, 2.1} (other values OK, too)

- For 2nd feature, we need to check

Threshold = { 0.9, 1.05, 1.2, 2.2}

- We also need to know if $h(\boldsymbol{x}_k) > 0$ means C=1 or C=-1
- Finally, pick the threshold with lowest ϵ_t

Weak classifier

- For example, we set thr = 1.4 in 1st feature: if 1st feature > thr, C =1, else C = -1
- We have only one error in 1st iteration (t = 1)
- Therefore, $\epsilon_1 = 0.2$, $\alpha_1 = 0.6931$,
$$\boldsymbol{w}_{\cdot,1} = [0.5, 0.125, 0.125, 0.125, 0.125]^T$$
- We can do more steps with same approach

XOR experiment

- Use 100 samples in XOR as training samples:
- If (feature 1) * (feature 2) > 0

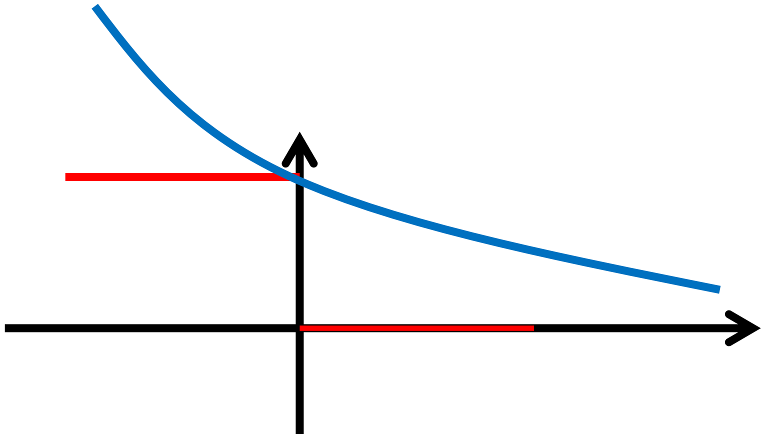
  then C = 1, else C = -1
- Feature 1 and 2 are random numbers
- No error in training set at around 400 iterations (i.e., 400 weak classifiers)

Adaboost theory

- Want to explain why combining many weak classifiers can make a strong classifier with training error $\rightarrow 0$
- The explanation follows https://www.youtube.com/watch?v=tH9FH1DH5n0
  - It does not consider re-normalization for simplicity
  - Can also consider re-normalization with a bit more complicated math, cf. https://www.cs.princeton.edu/courses/archive/fall07/cos402/readings/boosting.pdf

Adaboost theory

- Preliminary
  - Let $u(x) = \begin{cases} 0, x \geq 0 \\ 1, x < 0 \end{cases}$
  - It is easy to see that $u(x) \leq e^{-x}$
  - Red: $u(x)$, blue: e^{-x}



Training error

- $H(\boldsymbol{x}_k) = \mathrm{sign}((\sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k))$ is the used classifier
- Error at time t $e_t = \frac{1}{n}\sum_{k=1}^{n} \ell(H(\boldsymbol{x}_k) \neq d_k)$
- Because $d_1, \dots, d_n \in \{-1, +1\}$, we have

$$\ell(H(\boldsymbol{x}_k) \neq d_k) = u(H(\boldsymbol{x}_k)d_k)$$

- Let $g_t(\boldsymbol{x}_k) = \sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k)$

Training error

- $u(H(\boldsymbol{x}_k)d_k)$ can further be simplified as

$$u(H(\boldsymbol{x}_k)d_k) = u\left(\left(\sum\nolimits_{z=1t} \alpha_z h_z(\boldsymbol{x}_k)\right) d_k\right)$$
$$= u(g_t(\boldsymbol{x}_k)d_k)$$

- Therefore, $e_t = \frac{1}{n}\sum_{k=1}^{n} u(g_t(\boldsymbol{x}_k)d_k)$
- Recall $u(x) \leq e^{-x}$, thus

$$e_t \leq \frac{1}{n}\sum_{k=1}^{n} \exp(-g_t(\boldsymbol{x}_k)d_k)$$

Training error

- Consider error-weight update in one sample

$$w_{k,t+1} = w_{k,t} \exp(-\alpha_t h_t(\boldsymbol{x}_k) d_k)$$

  - Initial condition: $w_{k,1} = \frac{1}{n}$
  - Thus, $w_{k,2} = \frac{1}{n}\exp(-\alpha_1 h_1(\boldsymbol{x}_k) d_k)$
  - $w_{k,3} = \frac{1}{n}\exp(-\alpha_1 h_1(\boldsymbol{x}_k) d_k)\exp(-\alpha_2 h_2(\boldsymbol{x}_k) d_k)$

Training error

- Expanding it with $\prod$ notation, we have
- $w_{k,t+1} = \frac{1}{n}\prod_{z=1}^{t} \exp(-\alpha_z h_z(\boldsymbol{x}_k) d_k) = \frac{1}{n}\exp(-d_k \sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k)\ )$
- Recall $g_t(\boldsymbol{x}_k) = \sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k)$
- We have $w_{k,t+1} = \frac{1}{n}\exp(-g_t(\boldsymbol{x}_k) d_k)$

Relation between error & weights

- Summing over all k, we have
- $w_{all,t+1} = \frac{1}{n}\sum_{k=1}^{n} \exp(-g_t(\boldsymbol{x}_k)d_k)$
- But, $e_t \leq \frac{1}{n}\sum_{k=1}^{n} \exp(-g_t(\boldsymbol{x}_k)d_k)$
- We have $e_t \leq w_{all,t+1}$
- Therefore, all we have to do is to show $w_{all,t+1} \to 0$ if $t \to \infty$

Weights decay

- $w_{k,t+1} \leftarrow w_{k,t} \exp(-\alpha_t h_t(\boldsymbol{x}_k) d_k)$ with $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

Actually means

$$w_{k,t+1} = \begin{cases} w_{k,t} \times \sqrt{\dfrac{1-\epsilon_t}{\epsilon_t}} & \boldsymbol{x}_k \text{ wrong class} \\ w_{k,t} \times \sqrt{\dfrac{\epsilon_t}{1-\epsilon_t}} & \boldsymbol{x}_k \text{ correct class} \end{cases}$$

Weights decay

- On the average, we have
- $w_{k,t+1} = \epsilon_t \times w_{k,t} \times \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ (wrong classification)

  $+(1-\epsilon_t)w_{k,t} \times \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}$ (correct classification)

Therefore, $w_{k,t+1} = w_{k,t} \times 2 \times \sqrt{\epsilon_t(1-\epsilon_t)}$

- If $\epsilon_t < 0.5$ (assumption of weak classifier), then $2 \times \sqrt{\epsilon_t(1-\epsilon_t)} < 1$

Weights decay

- Therefore, $w_{k,t+1} = \gamma_t \times w_{k,t}$, where$\gamma_t < 1$
- As $w_{k,t+1} = \gamma_t \times \gamma_{t-1} \times \gamma_{t-2} \times \cdots \times \frac{1}{n}$

  This term approaches to zero if $n \to \infty$

Using AdaBoost

- **Keep in mind: AdaBoost is very sensitive to noise (i.e., training samples with wrong labeling)**
- Need to use weak classifiers for best performance
- Theories show that AdaBoost also widens the "margin" as SVM does

Concept of gradient boosting

General algorithm

- Initial function $H_0(\boldsymbol{x}_k) = 0 \ \forall k$
- For i = 1 .. T
  - Find a function $h_t(\boldsymbol{x}_k)$ and w_t to improve $H_{t-1}(\boldsymbol{x_k})$ for all samples k = 1.. N, based on loss function and gradient descent
  - $H_t(\boldsymbol{x}_k) = H_{t-1}(\boldsymbol{x}_k) + \beta_{k,t}\, h_t(\boldsymbol{x}_k)$
- Final classifier output $\mathrm{sign}(H_T(\boldsymbol{x}_k))$

Concept of gradient boosting

- From the above algorithm we know

$$H_t(\boldsymbol{x}_k) = H_{t-1}(\boldsymbol{x}_k) + \beta_{k,t}\, h_t(\boldsymbol{x}_k)$$
$$= \sum_{z=1}^{t} \beta_{k,z} h_z(\boldsymbol{x}_k)$$

- Thus, the key of the algorithm is to find
  - $\beta_{k,t}$
  - $h_t(\boldsymbol{x}_k)$
- To do so, we treat the problem as an optimization problem

Gradient descent

- Define a loss (objective) function

$$\mathcal{L}(H_t) = \sum_{k=1}^{n} \exp(-d_k\, H_t(x_k))$$

- Want to update

$$H_t(\boldsymbol{x}) = H_{t-1}(\boldsymbol{x}) - \eta \nabla \mathcal{L}(H_{t-1})$$

  via gradient descent

- Ignore η (step size) at this moment

What is $\beta_{t,k}$

- $\nabla \mathcal{L}(H_{t-1}) = \frac{\partial}{\partial H_{t-1}(x_k)} \mathcal{L}(H_{t-1}) = -\sum_{k=1}^{n} \exp(-d_k H_{t-1}(\boldsymbol{x}_k)) d_k$
- Therefore, (gradient update in one sample)

$$H_t(\boldsymbol{x_k}) = H_{t-1}(\boldsymbol{x_k}) + \exp(-d_k H_{t-1}(\boldsymbol{x_k})) d_k$$

- In the algorithm, $H_t(\boldsymbol{x_k}) = H_{t-1}(\boldsymbol{x_k}) + \beta_{t,k} h_t(\boldsymbol{x_k})$
- We may reasonably assume

$$\beta_{t,k} = \exp(-d_k H_{t-1}(x_k))$$

d_k is related to $h_t(\boldsymbol{x_k})$

What is $\beta_{t,k}$

- Therefore, $\nabla\mathcal{L}(H_{t-1}) = -\sum_{k=1}^{n} \beta_{t,k}\, d_k$ with $\beta_{t,k} = \exp(-d_k H_{t-1}(x_k)) = \exp\left(-d_k \sum_{z=1}^{t-1} \beta_{k,z} h_z(\boldsymbol{x}_k)\right)$
- From our previous derivation, we have (in Adaboost)
- $w_{k,t+1} = \frac{1}{n}\exp(-d_k \sum_{z=1}^{t} \alpha_z h_z(\boldsymbol{x}_k)\,)$
- Therefore, we know (in Adaboost)

  $\beta_{t,k} = w_{k,t}$ is a function of α_t (need to find it later) subject to a constant $({}^{1}/_{n})$

How to determine $h_t(\boldsymbol{x}_k)$

- In real gradient descent, we have the freedom to use true gradient in iteration
- But in the present case, we want to use a weak classifier (such as a decision stump) as an estimate of gradient
- Therefore, we want to match the gradient direction as much as possible between d_k and $h_t(\boldsymbol{x}_k)$ for all samples

How to determine $h_t(\boldsymbol{x}_k)$

- What can we do?
- Pick the weak classifier which minimizes

$$\beta_{t,k}\ell(h_t(\boldsymbol{x}_k) \neq d_k)$$

- Usually finding such a weak classifier requires a search

How to determine α_t

- $\beta_{t,k}$ is a function of α_t and $\beta_{t,k}$ is the step size of the gradient (similar to the role of η in previous equation)
- Want to find the optimal value of α_t
- It can be found by taking derivatives (detailed omitted)
- With computation, we have $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ is the optimal step size (same as in Adaboost)

Other boosting methods

- Other than AdaBoost, we also have gradient tree boosting methods
  - XGBoost
  - LightGBM (Light Gradient Boosting Machine)
  - CatBoost

Introduction to XGBoost

- Brief introduction to XGBoost (eXtreme Gradient Boosting)
- Similar concept as gradient boosting mentioned previously
- Use a different objective function
- Ref: https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Introduction to XGBoost

- Objective Function: to minimize (Training Loss + Regularization)

$$J(\theta) = \mathcal{L}(\theta) + \Omega(\theta)$$

- $\mathcal{L}(\theta)$ could be
  - MSE $\sum_k (y_k - d_k)^2$
  - Other (such as logistic loss)

Regularization

- $\Omega(\theta)$ is a regularization term, defined as

$$\Omega(\theta) = \gamma T + \frac{1}{2}\lambda \sum_{k=1}^{K} \|\boldsymbol{w}_k\|^2$$

  - K is number of trees
  - T is the number of leaves in each tree (remember all trees have the same structure, such as number of leaves)
  - $\boldsymbol{w}_k \in \mathbb{R}^T$ is the vector of scores (weights) on leaves for each tree

Regularization

- We can follow the concept of gradient descent (use up to 2nd derivative) to minimize objective function
- Details see reference (original paper) https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf

Ensemble learning

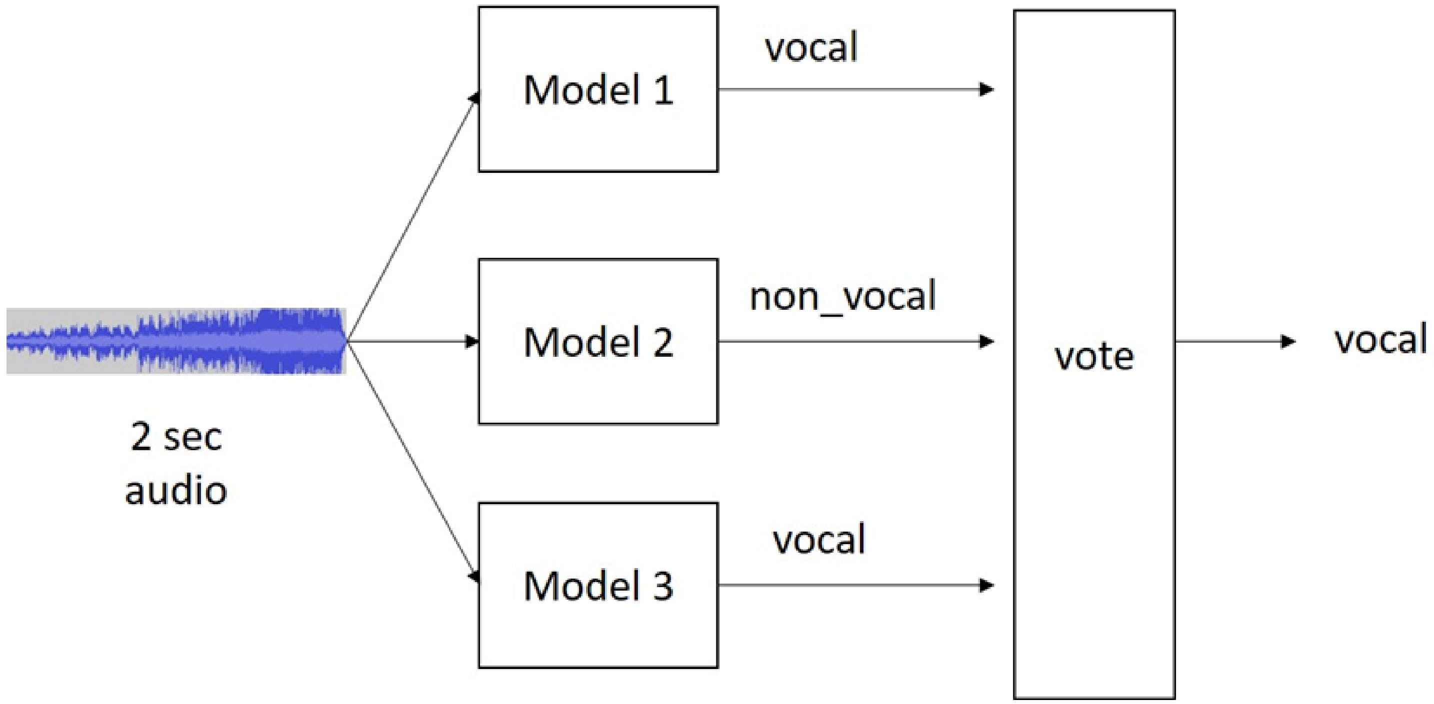
- Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) – from http://www.scholarpedia.org/article/Ensemble_learning

Ensemble learning

- Boosting
  - Well known: AdaBoost (mentioned before)
  - Mainly for weak classifiers
- Bagging
  - Well known: Random forest (mentioned before)
  - Mainly for classifiers easy to overfit
- **Stacking**
  - Voting
  - Post classifier
  - Fusion

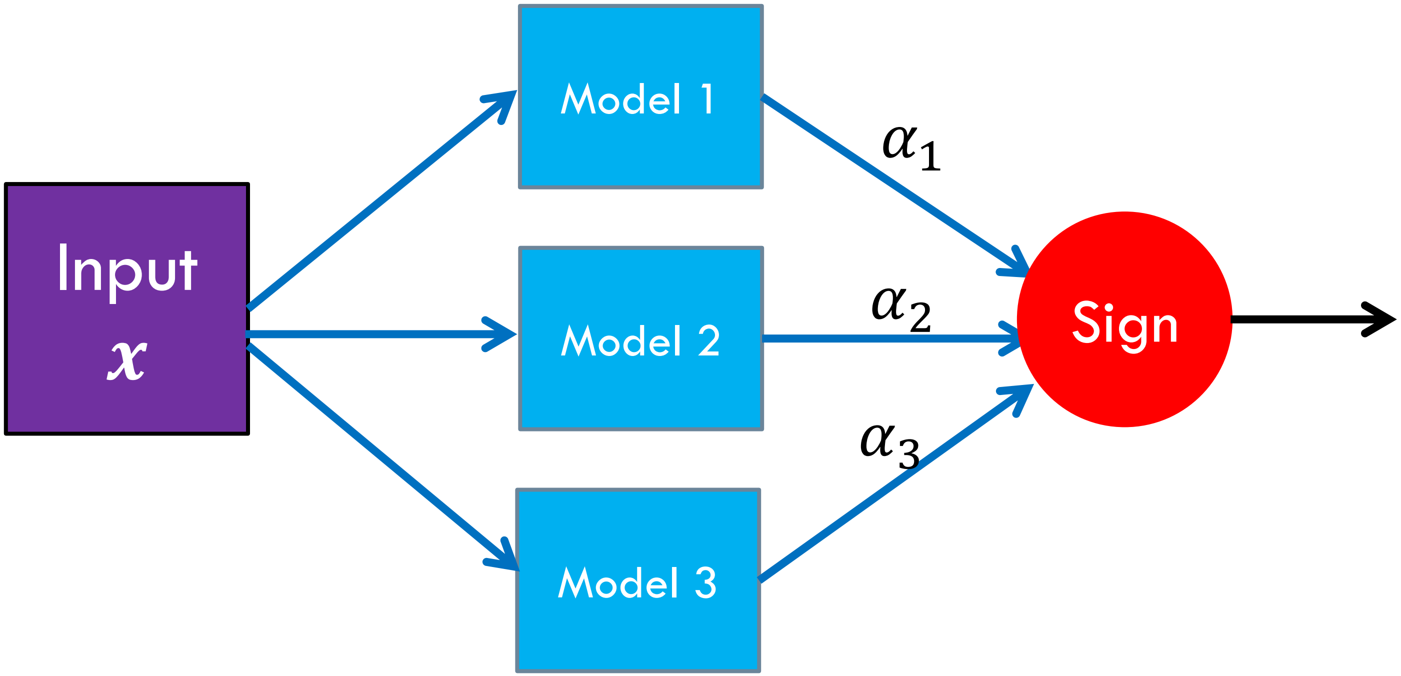
Voting

- Do a majority vote (equal weights for decision of each classifier)
  - Simple, yet powerful



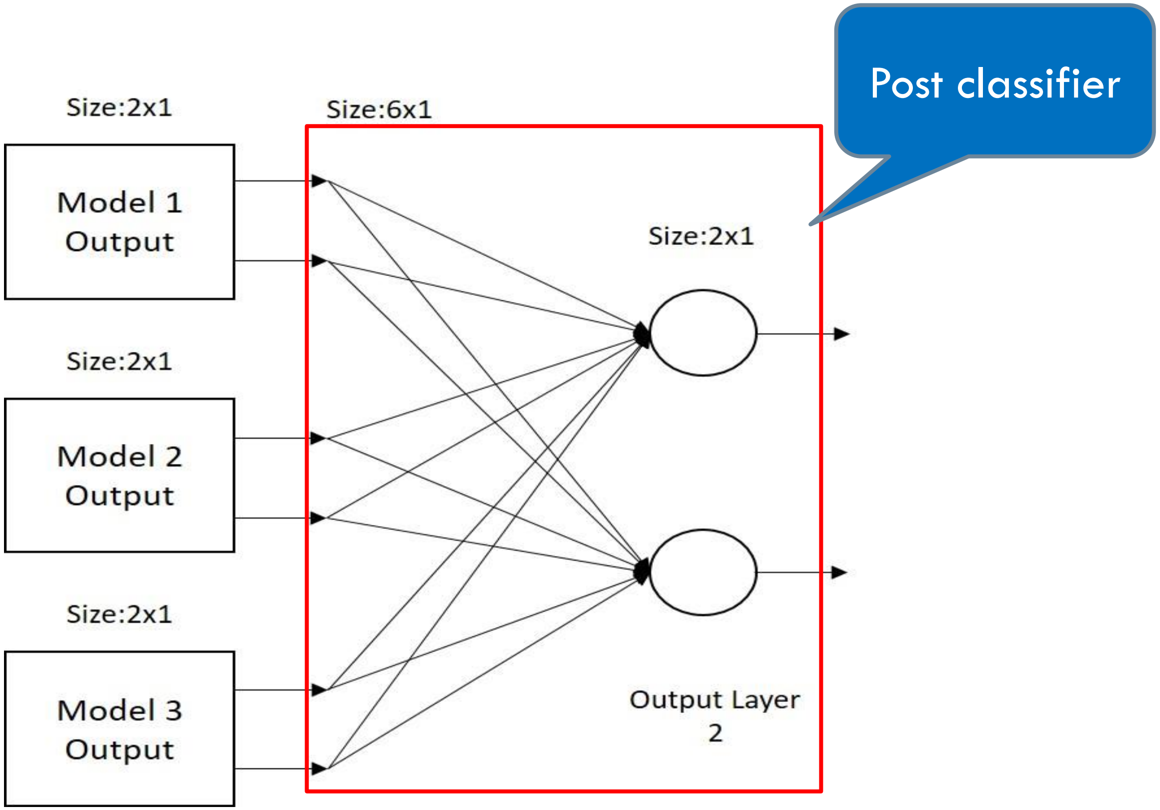
Adaboost as weighted voting

- Meaning of $H(\boldsymbol{x}) = \text{sign}\left(\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)\right)$
- Recall each $h_t(\boldsymbol{x})$ has binary answer (± 1)



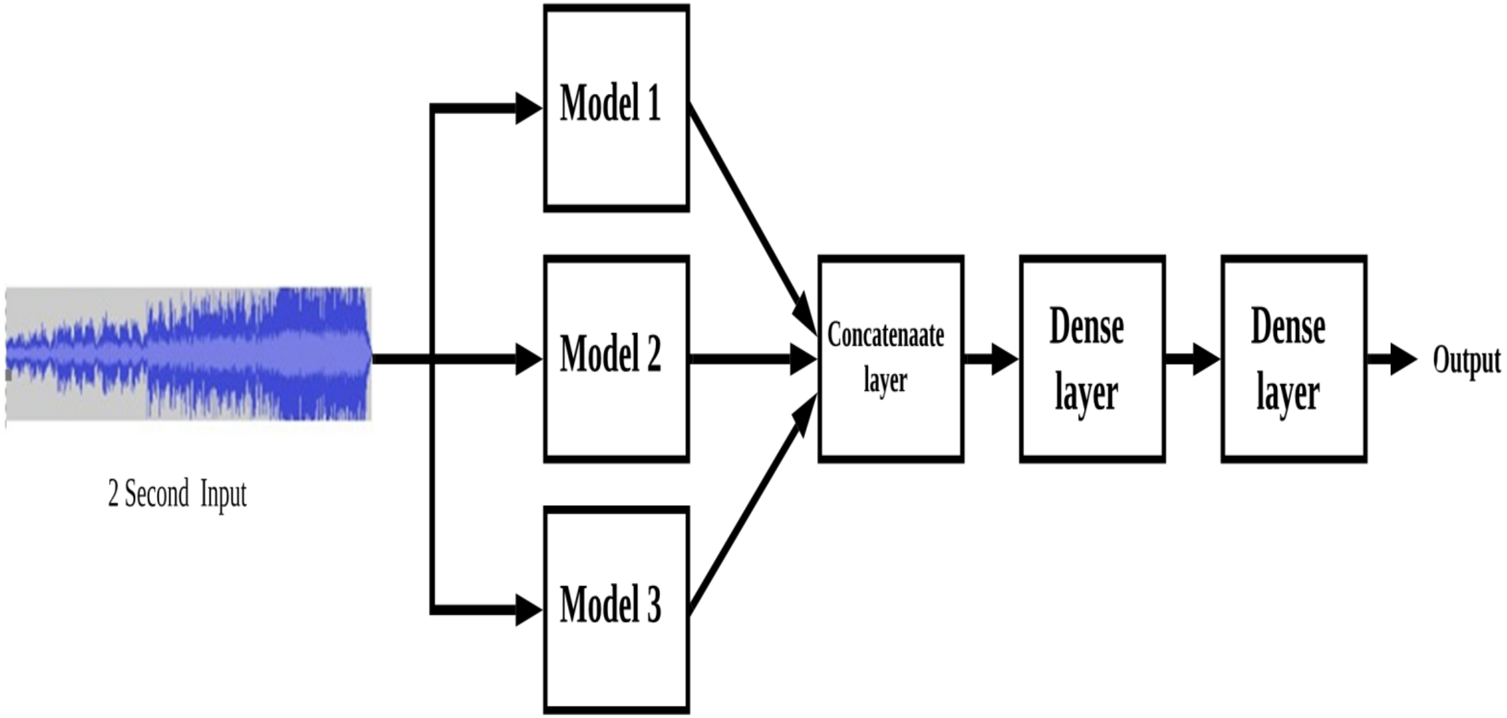
Post classifier

- Use post classifier to classify model outputs
  - Need to split training dataset to train post classifier



Fusion

- Merge together to build a larger network



Comparison

- Based on our experiments
  - Voting is actually better
  - Post classifier is not too much useful if the base classifier is a neural network (output values too close to one or zero)
  - Fusion seems to have too many parameters (complexity too high)
- In some cases, even 3 classifiers improve accuracy by one or two percents

Regression

- Ensemble learning can also be used for regression problem
  - Replace voting with average
  - Similar to what random forest does for regression