Model Ensembles

Makoto Yamada

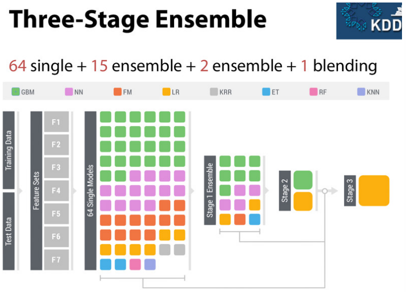
myamada@i.kyoto-u.ac.jp

Kyoto University

July/13/2020

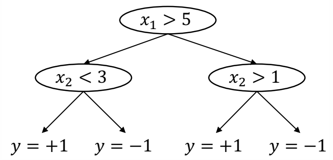
Model ensemble

- One model cannot fit all
- Combine different predictors to improve performance
- Commonly used technique in predictive modeling competitions (e.g., Kaggle)



Decision trees

Decision tree: Build a classifier by hierarchically dividing input space.



- Relatively fast (can scale with respect to the number of samples)
- High interpretability
- Instable (like other non-linear models)
- Sensitive to noise and hyper-parameters

Variants: regression trees, piecewise linear prediction, etc.

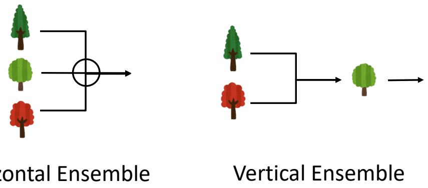
Two ways of ensemble

Horizontal ensemble (e.g., bagging, boosting)

- Construct a set of models in parallel
- Integrate their outputs to make final predictions

Vertical ensemble (e.g., Stacking, "deep" neural networks.)

- Make a cascade of models
- Outputs of ℓ-th level models are used as inputs of $\ell + 1$-th level models



Parallel ensemble methods

Bagging

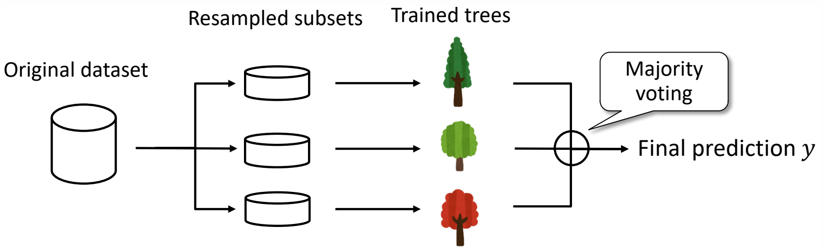
- Simple ensemble method based on data resampling
- Random forest is often "the first choice"
- Easily parallelizable

Boosting

- Adaptive version of bagging (weighted sum)
- AdaBoost
- Gradient boosted decision trees (GBDT)
- XGBoost is the regular winner in competitions (e.g., Kaggle)

Bagging

- Decision trees are sensitive to data change
- Bootstrapping: Train multiple classifiers using randomly resampled subsets of the original dataset
- Majority voting to aggregate predictions
- Stable, but lost interpretability

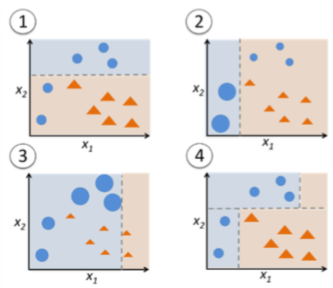


Random forest

- Randomly resample not only the data but also features
- Example: 5 samples $\{1, 2, 3, 4, 5\}$ and 4 features $\{A, B, C, D\}$
    - 1st tree trained with 2 features $\{A, B\}$ on subsamples $\{1, 3, 5\}$
    - 2nd tree trained with $\{A, B, C, D\}$ on subsamples $\{1, 2, 3\}$
    - 3rd tree trained with $\{A, C, D\}$ on subsamples $\{2, 3, 4, 5\}$
    - $\ldots$
- Off-the-shelf non-linear model: stable and high performance

Boosting

- In bagging, all models are independently trained using uniformly-random resamples
- In boosting, the next model is trained focusing on the "difficult" data that the current model cannot correctly classify



AdaBoost

We consider an additive model:

$$f_m(\boldsymbol{x}) = \sum_{k=1}^{m} \beta_k b(\boldsymbol{x}; \gamma_k),$$

where $\ell(y, f(\boldsymbol{x}))$ is a loss function and $b(\boldsymbol{x}; \gamma_m)$ are basis function.

Then, the optimization problem is given as

$$\min_{\{\beta_k, \gamma_k\}_{k=1}^{M}} \sum_{i=1}^{n} \ell\left(y_i, f_M(\boldsymbol{x}_i)\right)$$

If we consider to fit only one basis function, it is feasible:

$$\min_{\beta,\gamma} \sum_{i=1}^{n} \ell\left(y_i, \beta b(\boldsymbol{x}_i; \gamma)\right)$$

AdaBoost

The optimization problem:

$$\min_{\{\beta_m,\gamma_m\}_{m=1}^{M}} \sum_{i=1}^{n} \ell\left(y_i, f_M(\boldsymbol{x}_i)\right).$$

We can use the Forward stagewise Additive Modeling. We solve the optimization for a basis function one by one.

For squared-loss (regression):

$$\begin{aligned}\ell(y, f_m(\boldsymbol{x})) &= \left(y - \sum_{k=1}^{m} \beta_k b(\boldsymbol{x}; \gamma_k)\right)^2 \\ &= (\underbrace{y - f_{m-1}(\boldsymbol{x})}_{r_m} - \beta_m b(\boldsymbol{x}; \gamma_m))^2\end{aligned}$$

Estimating β_m and γ_m is easy if we know r_m

AdaBoost

Algorithm 1 Forward Stagewise Additive Modeling

```
1: $f_0(\boldsymbol{x})$
2: for $t = 1 \dots M$ do
3:     $(\beta_m, \gamma_m) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^{n} \ell(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \gamma))$
4:     $f_m(\boldsymbol{x}) = f_{m-1}(\boldsymbol{x}) + \beta_m b(\boldsymbol{x}; \gamma_m)$
5: end for
6: return $\{(\beta_m, \gamma_m)\}_{m=1}^{M}$
```

We fit model parameters one by one.

AdaBoost

For classification $y_i \in \{-1, 1\}$, we use the following loss function:

$$\begin{aligned} \ell(y, f_m(\boldsymbol{x})) &= \exp(-y f_m(\boldsymbol{x})) \\ &= \exp(-y(f_{m-1}(\boldsymbol{x}) + \beta_m b(\boldsymbol{x}; \gamma_m))) \\ &= \underbrace{\exp(-y f_{m-1}(\boldsymbol{x}))}_{w^{(m)}} \exp(-y \beta_m b(\boldsymbol{x}; \gamma_m))) \end{aligned}$$

Then, the objective function can be written as

$$J = \sum_{i=1}^{n} w_i^{(m)} \exp(-y_i \beta b(\boldsymbol{x}_i; \gamma)))$$

Note, $\beta_m \to \beta$ and $\gamma_m \to \gamma$.

AdaBoost

The objective function can be written as

$$\begin{aligned}
J &= \sum_{i=1}^{n} w_i^{(m)} \exp(-y_i \beta b(\boldsymbol{x}_i;\gamma))) \\
&= \sum_{y_i=b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-y_i \beta b(\boldsymbol{x}_i;\gamma)) + \sum_{y_i\neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-y_i \beta b(\boldsymbol{x}_i;\gamma)) \\
&= \sum_{y_i=b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) + \sum_{y_i\neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(\beta) \\
&= \sum_{y_i=b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) + \sum_{y_i\neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(\beta) \\
&+ \sum_{y_i\neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) - \sum_{y_i\neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta),
\end{aligned}$$

where $b(\boldsymbol{x}_i;\gamma) \in \{-1,1\}$.

AdaBoost

The objective function can be written as

$$\begin{aligned}
J = & \sum_{y_i = b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) + \sum_{y_i \neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(\beta) \\
& + \sum_{y_i \neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) - \sum_{y_i \neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} \exp(-\beta) \\
& = \sum_{i=1}^{n} w_i^{(m)} \exp(-\beta) + \sum_{y_i \neq b(\boldsymbol{x}_i;\gamma)} w_i^{(m)} (\exp(\beta) - \exp(-\beta)) \\
& = \exp(-\beta) \sum_{i=1}^{n} w_i^{(m)} + (\exp(\beta) - \exp(-\beta)) \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i;\gamma))
\end{aligned}$$

where $b(\boldsymbol{x}_i;\gamma) \in \{-1, 1\}$. Therefore, we can update γ_m as

$$\gamma_m = \operatorname*{argmin}_{\gamma} \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i;\gamma))$$

AdaBoost

Plugging the estimated γ_m, we have

$$J = \exp(-\beta) \sum_{i=1}^{n} w_i^{(m)} + (\exp(\beta) - \exp(-\beta)) \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i; \gamma_m))$$

Taking the derivative of J with respect to β and equate it to zero, we have

$$\begin{aligned}
\exp(-\beta) \sum_{i=1}^{n} w_i^{(m)} &= (\exp(\beta) + \exp(-\beta)) \sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i; \gamma_m)) \\
\exp(2\beta) &= \frac{\sum_{i=1}^{n} w_i^{(m)}}{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i; \gamma_m))} - 1 \\
\beta_m &= \frac{1}{2} \log \left(\frac{1 - \text{err}_m}{\text{err}_m} \right) \text{ with } \text{err}_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i; \gamma_m))}{\sum_{i=1}^{n} w_i^{(m)}}
\end{aligned}$$

AdaBoost

Updating $w_i^{(m)}$:

$$\begin{aligned}
f_m(\boldsymbol{x}_i) &= f_{m-1}(\boldsymbol{x}_i) + \beta_m b(\boldsymbol{x}_i;\gamma_m) \\
-y_i f_m(\boldsymbol{x}_i) &= -y_i f_{m-1}(\boldsymbol{x}_i) - y_i\beta_m b(\boldsymbol{x}_i;\gamma_m) \\
\exp(-y_i f_m(\boldsymbol{x}_i)) &= \exp(-y_i f_{m-1}(\boldsymbol{x}_i) - y_i\beta_m b(\boldsymbol{x}_i;\gamma_m)) \\
w_i^{(m+1)} &= w_i^{(m)} \exp(-y_i\beta_m b(\boldsymbol{x}_i;\gamma_m))
\end{aligned}$$

AdaBoost is

- $\gamma_m = \text{argmin}_\gamma \sum_{i=1}^n w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i;\gamma_m))$
- $\beta_m = \frac{1}{2}\log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$ with $\text{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} I(y_i \neq b(\boldsymbol{x}_i;\gamma_m))}{\sum_{i=1}^n w_i^{(m)}}$
- $w_i^{(m+1)} = w_i^{(m)} \exp(-y_i\beta_m b(\boldsymbol{x}_i;\gamma_m))$

XGBoost, LightGBM

- Implementation using gradient boosting + decision tree (Often appears in top ranked methods in competition)
- Training with constraints on the number of decision tree leaves and the total weights

Model stacking

Stacking: vertical ensemble method

- is similar to the multi-layer neural network (Neural Network is a stacked linear classification models)
- but can have heterogeneous components
- Outputs of ℓ-th level models are used as inputs to $\ell + 1$-th level models
    - Outputs of 0-th level models $\boldsymbol{y}_0$ is original feature vector $\boldsymbol{x}$
    - Outputs of ℓ-th level models $\boldsymbol{y}_\ell$
    - inputs to $\ell + 1$-th level models

$$\boldsymbol{x}_{\ell+1} = \begin{bmatrix} \boldsymbol{x}_\ell \\ \boldsymbol{y}_\ell \end{bmatrix}$$

Difficulty in model stacking

How can we train stacked models?

An easy solution:

- Train a classifier f using the training dataset L
- add the prediction values of f as a new feature
- $\ldots$

this seems to be working... but actually does NOT!

Remember the first principle: you cannot make a prediction for the data you used in the training! Easily overfitting to the data

The prediction value to the training data are biased because your model has been trained to reproduce the labels

How to stack?

Divide a given dataset into K non-overlapping sets

- Use $K-1$ of them for training a model
- Use the model to add a new feature to the remaining set
- Doing steps 1 and 2 for K hold out sets gives the new feature for the whole dataset

Train the level-2 predictor using the extended dataset

Finally, the level-1 predictor is (re-)trained using the original whole dataset

Summary

Summary of this class

- Horizontal ensemble (Bagging,Random forest, Boosting)
- Vertical ensemble (Stacking, "deep" neural networks)