Recitation 13: Random forests and artificial neural networks

Seung-hun Lee

Columbia University
Introduction to Econometrics II Recitation

May 2nd, 2022

Random forest

Classical random forests: Classification and regression trees

- We have an IID data (y_i, x_i) for $i \in \{1, ..., n\}$
- Decision trees represent classification and regression problems using nonlinear basis functions (indicators of categories or cutoff)
  - Branch: Path from one node to other
  - Split: Process of dividing a sample based on a criterion specified at a node; creates two branches per non-terminal nodes
  - Depth: Number of stages with splits
  - Leaves: Output; or nodes on the last stage
  - Pruning: Reducing the size of the tree (steps or fewer splits)
- The goal is to find an algorithm that best predicts out-sample properties

CART uses recursive greedy algorithm

- Recursive in that there are repeated nodes with partitioning of two-way splits
  - Select a variable x_k and threshold/class that allows each split outcome to be as different as possible (between variance and information gain)
  - Between variance: Variation between each group mean to overall mean, R^2 of

$$y = \beta \mathbb{I}(x_k > t) + u$$

  - Information gain: For a potential split $S = S_1 \cup S_2$

$$IG = L(S) - \left[\frac{|S_1|}{|S|} L(S_1) + \frac{|S_2|}{|S|} L(S_2) \right], \quad L(C) = -\sum_{j=1}^{J} f_j \log f_j$$

    i.e. splits S_1 and S_2 that has lower combined entropies (leading to high IG)
- Greedy in that at each non-terminal node, the best test selected is only locally optimal
  - No look-forward
  - Due to NP-hardness of the optimal tree problem, this is cheaper

Finding the best CART is not easy

- Selection of hyperparameters (parameters whose value is used to control the learning process such as split criterion, stopping criteria)
  - Can lead to overfitting and bad out-sample predictions if improperly selected
- Pruning: Making the trees more compact by eliminating some splits
  - Cross-validation is used
- Greedy algorithm: Decision tree may not be stable and the most optimal possible, especially when the top node is incorrectly selected

Bootstrap AGGregatING, or Bagging

- Like ensemble method, it averages from multiple decision trees to enhance stability and accuracy of prediction, classification, and regression
- By averaging, bagging protects from overfitting the data and prevents instability
- The procedure is very similar to that of bootstrapping. They are as follows
    1. Bootstrap: Let's have a sufficiently large B resamples with replacement. In each resampling step, we fit a tree.
    2. Aggregation: We average the predictions of the B trees to predict y given x.
- Predictive capability: Are the predictions less dispersed? Then this is good prediction

Random Forest

- Averaging multiple deep decision trees, trained on different parts of the same training data set
- Unlike bagging, random forests choose splits from a smaller number of randomly selected covariates at each step of the tree construction (feature bagging)
  - With k available features, $\sqrt{k}$ is used in classification and $\max\{k/3, 5\}$ in regression
- Less correlation among trees compared to earlier methods (variety of top splits)
- Less overfitting and more stability
  - Variable with huge decline in split-criterion (out-of-bag-error from permutation over B trees) can be considered important
- Key problem is loss of interpretability (black-box procedure in selection of covariates)

Causal forest (Wagner, Athey 2018)

- Random forest to estimate treatment effect (under CIA)
- We first estimate the propensity score $\hat{p}_i$ and transform y_i into

$$\hat{y}_i = \frac{D_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)} y_i$$

  which is an IPW formula
- By taking the expected value of $\hat{y}_i$ we get

$$\widehat{TE}(x) = E[\hat{y}_i | x]$$

- We then do 'honest' inference (defined in the paper) by estimating on a subsample that is different from the training and test subsamples (double-sample trees)
- We can also do normal inference

Local linear forest (Friedberg, Tibshirani, Athey, Wagner 2020)

- Random forests are used to generate weights that can serve as a kernel for a local linear regression and estimates for the conditional mean function
- Think of random forests as adaptive weight generators

$$\hat{m}(x_0) = \frac{1}{B}\sum_{i=1}^{n}\sum_{b=1}^{B} y_i \frac{\mathbb{I}[X_i \in L_b(x_0)]}{|L_b(x_0)|} = \sum_{i=1}^{n} y_i \underbrace{\frac{1}{B}\sum_{b=1}^{B}\frac{\mathbb{I}[X_i \in L_b(x_0)]}{|L_b(x_0)|}}_{\alpha_i(x_0)}$$

- $L_b(x_0)$ represents the leaf at b'th tree, $|\cdot|$ indicates the size of the leaves.
- $\alpha_i(x_0)$ is the fraction of trees in which an observation appears in the same leaf as the target value of the covariate vector

Local linear forest (Friedberg, Tibshirani, Athey, Wagner 2020)

- Local linear forests solve the weighted least problem below with penalty term using the weights defined above

$$\begin{pmatrix} \hat{m}(x_0) \\ \hat{\theta}(x_0) \end{pmatrix} = \arg\min_{m,\theta} \left[\sum_{i=1}^{n} \alpha_i(x_0)(y_i - m(x_0) - (x - x_0)\theta(x_0))^2 + \lambda||\theta(x_0)||^2 \right]$$

- Performs better when smooth, strong signals are present (random forests don't do so well in this case)
- Also overcomes curse of dimensionality

Gradient-boosted trees

- Improve the performance of weak learners by boosting the weight of observations they misspecify
- Start with a loss function $\min \sum_{i=1}^{n} L(y_i, \hat{y}_i)$, say, $(y_i - \hat{y}_i)^2$
- $v(x, S)$: Average value of y in the leaf that contains x in a shallow tree with splits S
- Then we proceed in these steps
    1. Begin with an initial prediction: $\hat{y}_i^{(0)} = E[y_i]$
    2. For each iteration $b \in \{1, ..., B\}$, we choose split S_b in the tree of depth d to minimize

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i^{(b-1)} + v(x_i, S))$$

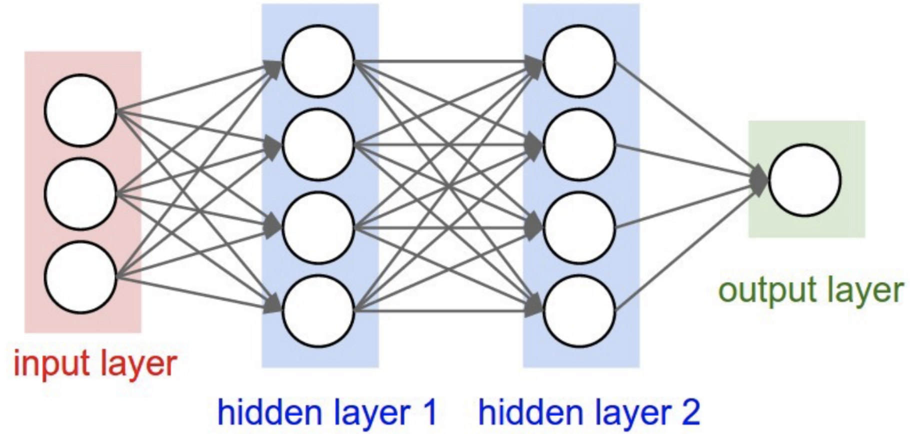
and update our prediction $\hat{y}_i^{(b)} = \hat{y}_i^{(b-1)} + \epsilon v(x_i, S_b)$. Note that ϵ is the learning rate and is not $\epsilon = 1$, which would mean updating the prediction by the exact value and risk overshooting

Other ways to do gradient-boosted trees

- We can also tune the model by including a complexity penalty in the loss function. So we minimize
$$\sum_{i=1}^{n} L(y_i, \hat{y}_i^{(b-1)} + v(x_i, S)) + C(S)$$
- XGBoost: Extreme gradient-boosting, increases the speed of the procedure using quadratic loss and complexity penalty functions
$$\sum_{i=1}^{n}(G_i v(x_i, S) + \frac{1}{2} H_i v(x_i, S)^2) + \gamma K + \frac{\lambda}{2} \sum_{k=1}^{K} v_k^2$$
where G_i and H_i are each the first and second derivatives of the $L(y_i.\hat{y}_i)$ w.r.t $\hat{y}_i$ at $(y_i, \hat{y}_i^{(b-1)})$, K is the number of leaves, and v_k is the predictor on leaf k
- Tuning and some hyperparameters still need to be determined

Artificial neural networks

Architecture of artificial neural networks



Architecture of artificial neural networks

- Input layers have covariates x_m
- Lines between layers have weights w
- Each direction represents signal $\sigma(\cdot)$, known as activation function
  - A sigmoid (logit)
  - Multinomial logit (softmax)
  - Rectified linear unit (ReLU, $\sigma(t) = \max\{0, t\}$)

Single layer perceptions: Setup and forward pass

- M neurons in input, L nodes in hidden layer
- Input neurons are denoted $x_1, \dots, x_M$ have weights $w_{1lm},\; m \in \{1, \dots, M\},\; l \in \{1, \dots, L\}$, and emit $\sigma(\sum_m w_{1lm} x_m)$ to a hidden layer l
- Hidden layer has weight for each node, w_{2l}, used to combine inputs from previous nodes to generate a prediction for the output node $\hat{y} = g\left(\sum_l w_{2l}\sigma(\sum_m w_{1lm} x_m)\right)$
- Epoch: One forward pass + backward pass
- Forward pass: Given loss function $L(y, \hat{y}(\theta))$ and an initial value $\theta = (w_1, w_2)$, compute

$$z_{il} = \sigma(w_{1l}' x_i)$$

and then the output can be produed as

$$\hat{y}_i(\theta) = w_2' z_i = \sum_l w_{2l}\sigma\left(\sum_m w_{1lm} x_{im}\right)$$

Single layer perceptions: Backward pass

- Compute the components of the loss functions, then take gradients with respect to elements in θ, and do approximate Newton iteration (e.g. Minibatch gradient descent)

$$\theta^{(s+1)} = \theta^{(s)} - \epsilon_s \frac{\partial L}{\partial \theta}(\theta^{(s)})$$

- ϵ_s is the learning rate which ideally should goes to 0
- Backpropagation: An automated differentiation and application of chain rule to identify gradients used to update the values of the weights

Deep learning: Multiple hidden layers

- Many (hyper)parameters, activation functions to choose
- We have p covariates, K outputs, D hidden layers, and M nodes per layer (p, K given)
- In total, we have $pM + (D-1)M^2 + kM$ parameters (or weights) to work with
- Optimal D and M? It is known that deep artificial neural networks are preferable to wider ones
  - Small D does not fit well, large D induces less overfitting problem especially if there are many small weights (also, we gain on expressiveness)
  - Incorporating penalty term $\lambda||\theta||_q^q$ helps too
- Dropout: Randomly eliminates some links and thins the neural networks, which is then retrained and tested/compared with all the eliminated units brought back in
  - Realistic alternative to cross-validation

Double machine learning debiasing techniques

- ATE under CIA

$$y_i = \mu(X_i, D_i) + \epsilon_i(D_i),\ D_i = \mathbb{I}(u_i < p(X_i)),\ \ (\epsilon(0), \epsilon(1)) \perp\!\!\!\perp u|X$$

- Use ML to estimate the $p(X_i)$ and $\mu(X_i, D_i)$ and obtain IPW ATE
- Debiasing? $p(X_i)$ and μ are nuisance parameters whose errors affect TE estimates
- Decouple the TE from the $p(X_i)$ and μ using Neyman orthogonalization (block-diagonalize the (Fisher) information matrix) - ML method involved
  - Split data to K folds, and leave one out
  - Estimate $\hat{p}_k = \Pr(D = 1|X)$ using LASSO or other ML methods
  - Take the Neyman-orthogonalized combination of the estimates of $p(X_i)$ and μ
  - If functions are smooth, estimates have standard asymptotics and we can do standard tests afterwards