Ensemble methods

Fraida Fund

Contents

Math prerequisites for this lecture: You should know about:

- Variance of a random variable
- Independence of random variables
- Variance of sum of random variables

Ensemble methods

Recap: decision trees

- Let trees grow deep - low bias, high variance
- Don’t let trees get deep: low variance, high bias

Ensemble methods - the idea

Combine multiple **weak learners** - having either high bias or high variance - to create an **ensemble** with better prediction

Ensemble methods - types (1)

- Combine multiple learners with high **variance** in a way that reduces their variance
- Combine multiple learners with high **bias** in a way that reduces their bias

Ensemble methods - types (2)

- **Parallel**: build base estimators *independently* and then average their predictions. Combined estimator is usually better than any single base estimator because its *variance* is reduced.
- **Sequential**: (boosting) build base estimators *sequentially* and each one tries to reduce the *bias* of the combined estimator.

Bagging

Bagging - background

- Designed for, and most often applied to, decision trees
- Name comes from **bootstrap aggregation**

Bootstrapping

- Basic idea: Sampling **with replacement**
- Each “bootstrap training set” is *same size* as full training set, and is created by sampling with replacement
- Some samples will appear more than once, some samples not at all

Bootstrap aggregation

- Create multiple versions $1, \dots, B$ of training set with bootstrap
- Independently train a model on each bootstrap training set: calculate $\hat{f}_1(x) \dots, \hat{f}_B(x)$
- Combine output of models by voting (classification) or averaging (regression):

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)$$

Bagging trees

- Construct B trees using B bootstrapped training sets
- Let the trees grow deep, no pruning
- Each individual tree has low bias, high variance
- Average the prediction of the trees to reduce variance (if independent!)

Variance reduction rule

$$\begin{aligned}
\text{Var}(\bar{X}) &= \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) && (1)\\
&= \frac{1}{n^2}\,\text{Var}\left(\sum_{i=1}^{n} X_i\right) && (2)\\
&= \frac{1}{n^2}\left(\sum_{i=1}^{n}\text{Var}(X_i) + 2\sum_{i<j}\text{Cov}(X_i, X_j)\right) && (3)\\
&= \frac{1}{n^2}\cdot n\text{Var}(X_i) \quad \text{(if } X_i \text{ i.i.d.)} && (4)\\
&= \frac{1}{n}\,\text{Var}(X_i). && (5)
\end{aligned}$$

where:

1. uses definition of the sample mean: $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$.
2. uses the scaling rule: $\text{Var}(aY) = a^2\,\text{Var}(Y)$ with $a = \frac{1}{n}$.
3. uses variance of sum (+ symmetry of covariance):

$$\text{Var}(\sum_i X_i) = \sum_i \text{Var}(X_i) + \sum_{\substack{j=1\\ j\neq i}} \text{Cov}(X_i, X_j) = \sum_i \text{Var}(X_i) + 2\sum_{i<j}\text{Cov}(X_i, X_j)$$

4. because independence $\Rightarrow$ $\text{Cov}(X_i, X_j) = 0$ for $i \neq j$, and iid means $\text{Var}(X_i)$ is same for all i.

Variance reduction rule (general)

For n i.i.d. random variables, the variance of their mean decreases with n:

	General RV
The average	$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$
Variance with independence	$\text{Var}(\bar{X}) = \frac{1}{n}\text{Var}(X_i)$

Variance reduction rule (bagged trees)

For B **independent** bagged trees, variance decreases with B:

	Bagged Trees
The average	$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x)$
Variance with independence	$\text{Var}(\hat{f}_{bag}(x)) = \frac{1}{B}\text{Var}(\hat{f}_b(x))$

...but, they are not really independent!

Correlated trees

Problem: trees produced by bagging are highly correlated, and:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n}\left[1 + (n-1)\rho\right]$$

where $\rho = \text{Corr}(X_i, X_j) = \frac{\text{Cov}(X_i, X_j)}{\sqrt{\text{Var}(X_i)\,\text{Var}(X_j)}}$

- Imagine there is one feature that is strong predictor, several moderate predictors
- Most/all trees will split on this feature
- Averaging correlated quantities does not reduce variance as much.

Random forests

Grow many decorrelated trees:

- **Bootstrap**: grow each tree with bootstrap resampled data set.
- **Split-variable randomization**: Force each split to consider *only* a subset of m of the p predictors.

Typically $m = \frac{p}{3}$ but this should be considered a tuning parameter.

Bagged trees illustration

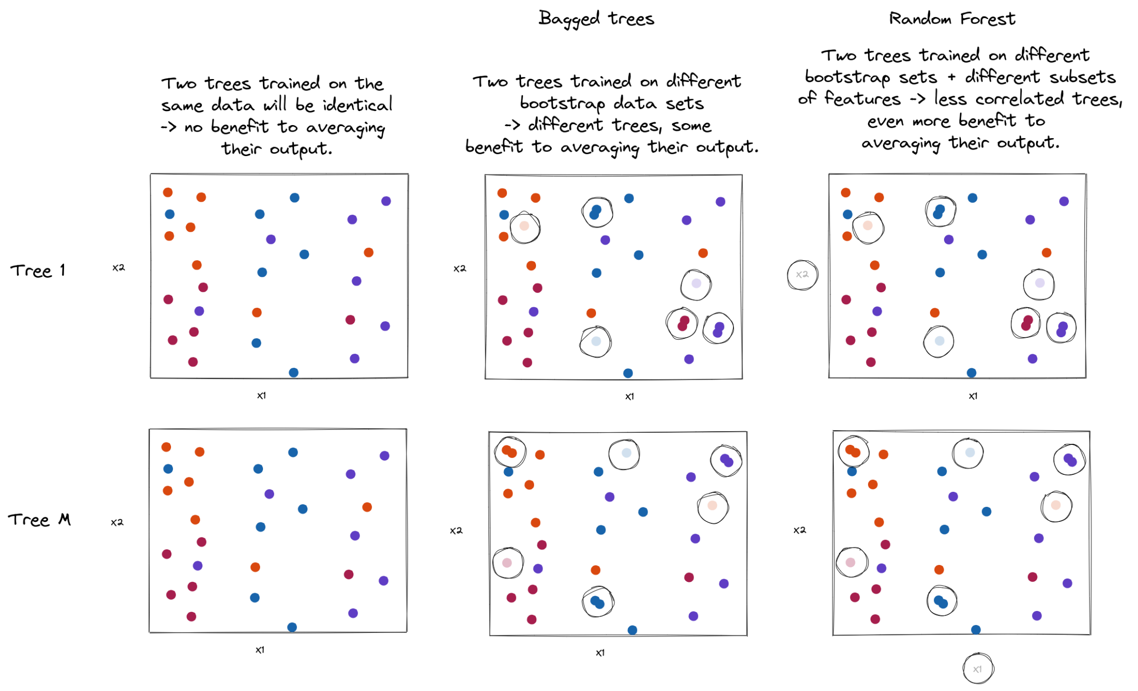
Bagged trees
Random Forest
Two trees trained on the same data will be identical -> no benefit to averaging their output.
Two trees trained on different bootstrap data sets -> different trees, some benefit to averaging their output.
Two trees trained on different bootstrap sets + different subsets of features -> less correlated trees, even more benefit to averaging their output.
Tree 1
Tree M
x2
x1

Figure 1: Identical data, bootstrapped data, and bootstrapped data with split variable randomization.

A note on computation

- Bagged trees and random forests can be fitted in parallel on many cores!
- Each tree is built independently of the others

Boosting

Boosting - training

Iteratively build a succession of models:

- Train a weak model. Typically a very shallow tree.
- In training set for bth model, focus on errors made by $b-1$th model.
- Use (weighted) model output
- Reduces bias *and* variance!

AdaBoost (Adaptive Boosting)

Adjust *weights* so that each successive model focuses on more "difficult" samples.

Consider a binary classification problem, where $y_i \in \{-1, +1\} \forall i$.

AdaBoost algorithm

1. Let $w_i = \frac{1}{N}$ for all i in training set.
2. For $m = 1, \dots, M$, repeat:

AdaBoost algorithm (inner loop)

- Fit weak learner $\hat{f}^m$, on training data with sample weights w_i.
- Compute weighted error err_m:

$$err_m = \frac{\sum_{i=1}^N w_i 1(y_i \neq \hat{f}^m(x_i))}{\sum_{i=1}^N w_i}$$

- Compute coefficient $\alpha_m = \log \left(\frac{1-err_m}{err_m} \right)$
- Update weights: $w_i \leftarrow w_i e^{\alpha_m 1(y_i \neq \hat{f}^m(x_i))}$ (for misclassified samples, scale weight by e^{α_m})

AdaBoost algorithm (final step)

3. Output final ensemble model:

$$f_M(x) = \sum_{m=1}^M \alpha_m \hat{f}^m(x), \quad \hat{y}(x) = \text{sign}[f_M(x)]$$

Gradient Boosting

- General goal of boosting: find the model at each stage that minimizes loss function on ensemble (computationally difficult!)
- AdaBoost interpretation (discovered years later): Gradient descent algorithm that minimizes exponential loss function.
- Gradient boosting: works for any differentiable loss function. At each stage, find the local gradient of loss function, and take steps in direction of steepest descent.

Gradient boosting can be viewed as *functional gradient descent*. We have a differentiable loss function

$$L(y, f(x)),$$

(exponential loss function, in the case of AdaBoost), and an additive model:

$$f_M(x) = \sum_{m=1}^{M} \alpha_m \hat{f}^m(x),$$

where each weak learner $\hat{f}^m(x)$ acts like a basis function.

Each iteration performs an update:

$$f_m(x) = f_{m-1}(x) + \alpha_m \hat{f}^m(x).$$

where at each step:

- $\hat{f}^m(x)$ - the weak learner - is chosen to align with the direction of the negative gradient of the loss,
- α_m - the coefficient - determines the step size in that direction.

Summary of (selected) ensemble methods

- Can use a single estimator that has poor performance
- Combining the output of multiple estimators into a single prediction: better predictive accuracy, less interpretability
- Also more expensive to fit