F-DS 701 Data Science Bootcamp

Machine Learning

Ensemble Learning

What’s Ensemble Learning

- Ask a question to many people, and aggregate their answers
- ***Wisdom of the crowd***
- Aggregate the predictions of a group of classifiers or regressors
    - Often better than the best individual predictor
    - Often use ensemble near the end of a project
    - For example, winning solutions in Kaggle or Netflix Price often involve ensemble methods
    - Bagging, Boosting, Stacking
- As independent as possible
    - To get diverse classifiers, using different algorithms or different dataset

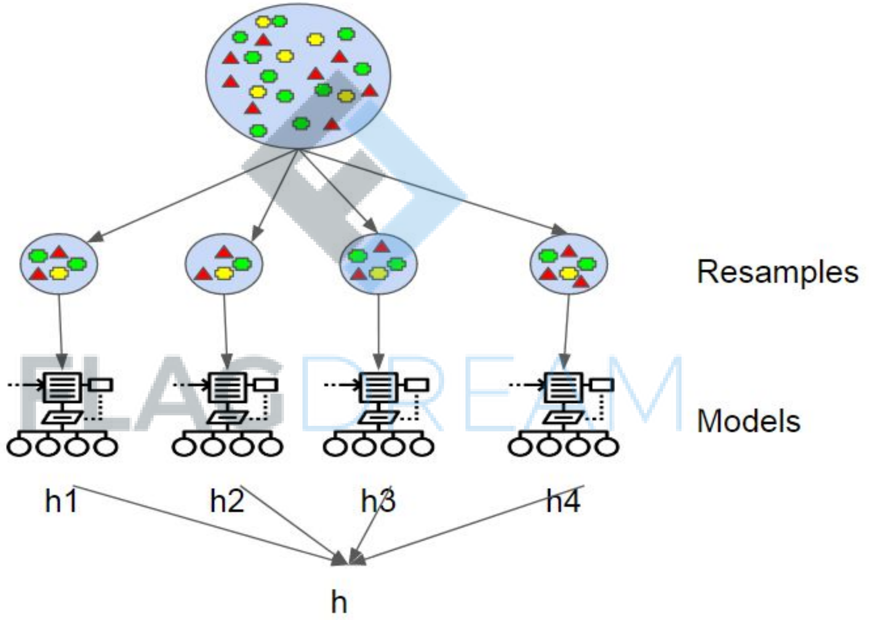
Bagging

Bootstrap Aggregating

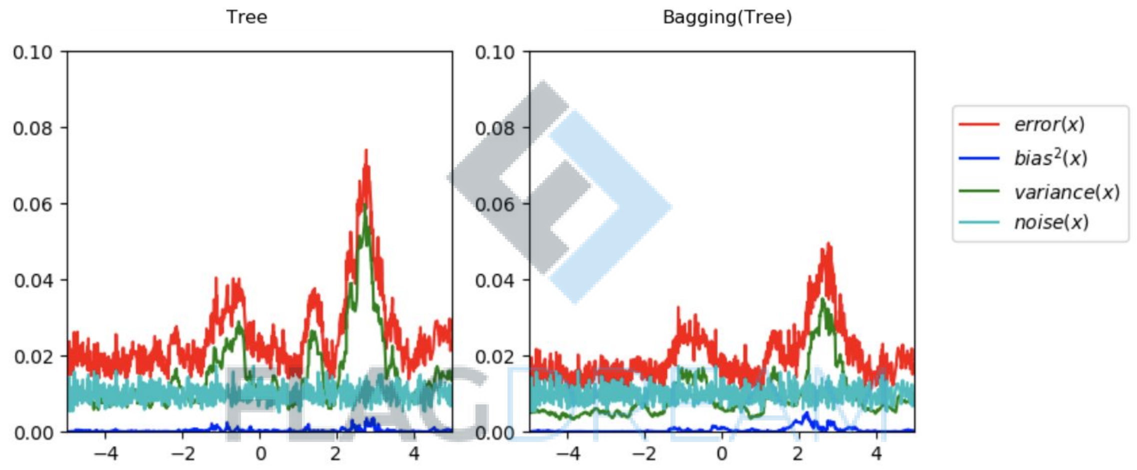
1. Create k bootstrap samples D1,D2,...,Dk
2. Train distinct estimator on each Di
3. Classify new instance by majority vote or averaging

- Same algorithm for every estimator
  - Logistic Regression, Decision Tree, etc.
- Parallel training through different CPU cores

Bootstrap Aggregating



Overfitting Problem



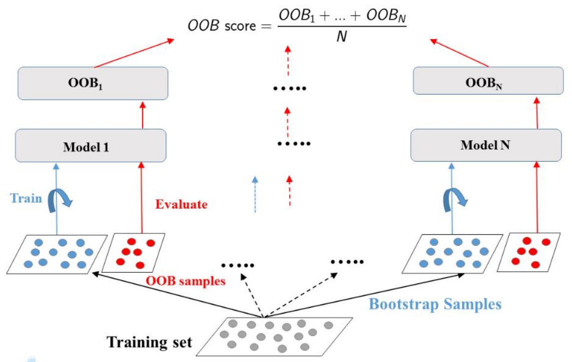
In Decision Tree, complex tree has low bias but high variance, due to the greedy approach

Bagging can reduce variance and at same time NOT affect bias much

Interview

Out-of-Bag Evaluation

- In bootstrapping step, some instances may be sampled several times, while others may not be sampled at all
- Only **<u>about 63%</u>** of the training set are sampled
- Averaging out the oob evaluations of each estimator
- Not exactly identical to Cross Validation



http://cican17.com/machine-learning-with-tree-based-models-ii/

Interview

Bagging is not for every algorithm

- Bagging is good for methods with high variance
    - Decision trees with deep trees
    - K nearest neighbor with small value of K

- Not good for stable methods
    - Linear/Logistic regression
    - Decision trees with shallow trees
    - K nearest neighbor with large value of K

Sampling Features

- High correlation among bagged estimators, if all features are used for training
- In sklearn, *BaggingClassifier* supports sampling the features
- In bias and variance tradeoff, sampling features trades more bias for a lower variance
- **Random Patches** method: sampling both instances and features
- **Random Subspaces** method: sampling features but keep all training data

Random Forest

Work

Random Forest

- An ensemble method of **Decision Tree**
- Instead of using *BaggingClassifier* + *DecisionTreeClassifier*, you can use *RandomForestClassifier*
- *RandomForestRegressor* for regression problem
- For each split, it searches for the best feature among a random subset of features
  - For classification, **√total # of features**
  - For regression, **total # of features / 3**
- Trees are more independent. Trade bias for lower variance
- Generally, yield an overall better model

Random Forest Algorithm

1. For $b = 1, \ldots, B$:
   a. Draw a bootstrap sample of size N from training data
   b. Grow a tree T_b to the bootstrapped data, by recursively repeating the following steps, until minimum node size is reached
      i. Select m variables at random from p variables
      ii. Pick the best variable and split point among m
      iii. Split into two children nodes
2. Output the ensemble of trees $\{T_b\}_{b=1 \to B}$

To make a prediction at a new point x:

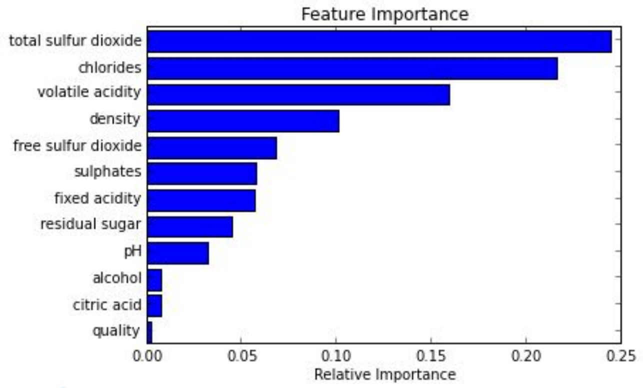
- Regression: $\frac{1}{B} \sum_{b=1}^{B} T_b(x)$
- Classification: $majority\ vote\ \{T_b(x)\}_{b=1,\ldots,B}$

Extremely Randomized Trees

- Based on Random Forest, build a more random model
- Instead of searching for the best threshold to split for each feature, using **random threshold**
- Trade more bias for lower variance
- Much faster to train than regular random forest
- *ExtraTreesClassifier* vs *RandomForestClassifier*
    - Hard to tell which is better
    - Try both. Compare using cross validation

Feature Importance

- Random Forest makes it easy to calculate importance of each feature
- In scikit-learn, feature importance is measured by looking at **how much the tree nodes** that use this feature **reduce impurity** on **<u>weighted average</u>** across all tree in the forest
- The node's weight is related to the number of training samples



Advantages of Random Forest

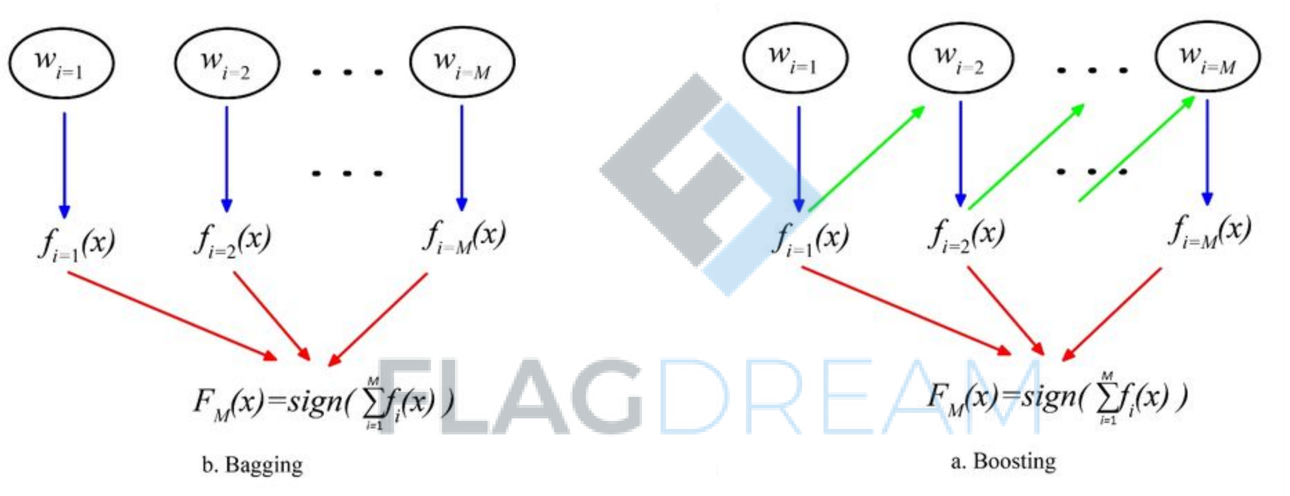
- One of **top performing** algorithms without much tuning (easier to tune than boosting)
- **Runs efficiently** on large data sets: intrinsically easy to parallel.
- **No feature transformation** needed for nonlinear or non-monotonic features
- Automatically addresses **variable interactions**
- Able to handle **thousands of input variables** without variable deletion: handles correlation and collinearity among features well.
- Gives **feature importance**
- **OOB** error
- It has an effective method for estimating **missing data** and maintains accuracy when a large proportion of the data are missing.

Disadvantages of Random Forest

- Relative long time to train
- Complex models (large files) to save for later use
- May still have remnant overfitting effect left
- Performance often is often beaten by Boosting
- Hard to interpret the results as compared with individual trees

Boosting

Bagging vs Boosting

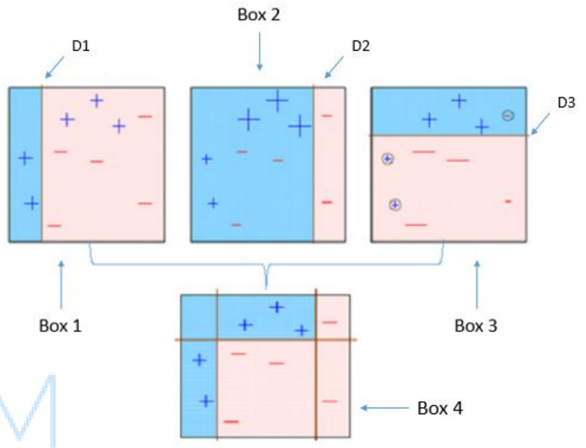


Boosting is to train estimators **sequentially**, each trying to correct its predecessor.
Combine several **weak learners** into a strong one.

https://www.researchgate.net/figure/Boosting-and-Bagging_fig1_304040874

AdaBoost

- Adaptive Boosting
- New estimators will put more emphasis on where previous estimator gets wrong
- For the instances being classified wrong, their **<u>weights</u>** get boosted
- For each weak learner fitting step, the corresponding parameter will be updated
- The final adaboost model is the weighted sum of all weak learners



AdaBoost Algorithm

N is the sample size, M is the number of estimators, $y_i \in \{-1, +1\}$

1. Initialize the data weights $w_i = \frac{1}{N}$, $i = 1, 2, ..., N$
2. For $m = 1, ..., M$:
    a. Fit a classifier $G_m(x)$ to training data with current weights w_i
    b. Compute error $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$, where $I(y_i \neq G_m(x_i))$ is the indicator function
    c. Compute $\alpha_m = \frac{1}{2}\log(\frac{1-err_m}{err_m})$, which is the corresponding weight for $G_m(x)$
    d. Update $w_i = w_i \exp[-\alpha_m y_i G_m(x_i)]$, $i = 1, ..., N$
3. The output is $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$

Advantages of AdaBoost

- Little parameters to tune
- Fast, simple and easy to program
- It comes with theoretical guarantee
- Only need to find weak classifiers that are better than random
  - For example, shallow decision trees with level 2~5

Disadvantages of AdaBoost

- The actual performance depends on the dataset and base learners
- Especially susceptible to noise
- If the number of outliers is large, the emphasis placed on those hard examples will hurt the performance

Additive Model

Boosting & Additive Model

- Boosting is a way of fitting an additive expansion in a set of elementary "basis" function
    - $f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$
- Estimate the model by minimizing loss function averaged over training data
    - $\min_{\{\alpha_m, G_m\}_{m=1,\ldots,M}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \alpha_m G_m(x_i))$

Boosting & Additive Model

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) , \quad \min_{\{\alpha_m, G_m\}_{m=1,\dots,M}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \alpha_m G_m(x_i))$$

- $(\alpha_1, G_1) = \arg \min_{\alpha_1, G_1} \sum_{i=1}^{N} L(y_i, \alpha_1 G_1(x_i)) \Longrightarrow f_1(x) = \alpha_1 G_1(x)$
- $(\alpha_2, G_2) = \arg \min_{\alpha_2, G_2} \sum_{i=1}^{N} L(y_i, f_1(x_i) + \alpha_2 G_2(x_i)) \Longrightarrow f_2(x) = \sum_{m=1}^{2} \alpha_m G_m(x)$

……

- $(\alpha_M, G_M) = \arg \min_{\alpha_M, G_M} \sum_{i=1}^{N} L(y_i, f_{M-1}(x_i) + \alpha_M G_M(x_i)) \Longrightarrow f_M(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$

Forward Stagewise

Sequentially add new basis functions, without adjusting parameters of those that have already been added.

1. Initialize $f_0(x) = 0$
2. Loop $m = 1, \ldots, M$:
   a. Compute $(\alpha_m, G_m) = \arg\min_{\alpha_m, G_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \alpha_m G_m(x_i))$
   b. Set $f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$

AdaBoost and Additive Model

AdaBoost = Forward Stagewise Additive Modeling

+ **Exponential Loss Function**

- Exponential Loss: $L(y, f(x)) = \exp(-y\,f(x))$
- At m step:
  - $(\alpha_m, G_m) = \underset{\alpha_m, G_m}{\arg\min} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \alpha G(x_i))$
  - $= \underset{\alpha_m, G_m}{\arg\min} \sum_{i=1}^{N} \exp(-y_i f_{m-1}(x_i) - \alpha y_i G(x_i))$
  - $= \underset{\alpha_m, G_m}{\arg\min} \sum_{i=1}^{N} \exp(-y_i f_{m-1}(x_i)) \times \exp(-\alpha y_i G(x_i))$
- Then $w_i = \exp(-y_i f_{m-1}(x_i))$ ← **weights**

AdaBoost and Additive Model

- $\sum_{i=1}^{N} w_i \exp(-\alpha y_i G(x_i)) = e^{\alpha} \sum_{y_i \neq G(x_i)} w_i + e^{-\alpha} \sum_{y_i = G(x_i)} w_i$
- $= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{N} w_i I(y_i \neq G(x_i)) + e^{-\alpha} \sum_{i=1}^{N} w_i$
- If fix α, $G_m(x) = \arg\min_{G} \sum_{i=1}^{N} w_i I(y_i \neq G(x_i))$
  - Fit G_m to data to minimize weighted training error

classifier

- If fix G, take the derivative and set to 0. We have
  - $\alpha_m = \frac{1}{2} \log \frac{1 - err_m}{err_m}$

alpha

  - $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i}$

error

  - Exactly the same as AdaBoost

Work + Interview

Gradient Boosting

Another Additive Model

- Forward Stagewise Additive Model
- Instead of tweaking instance weights at each iteration, it tries to fit a new estimator to ***residual error*** made by previous estimator
- Using Decision Tree as base learner, it is called Gradient Boosting Decision Tree (GBDT)
    - Can solve both classification and regression problem
- Usually it uses gradients of loss function to estimate the residual error

An Example: Boosting Tree

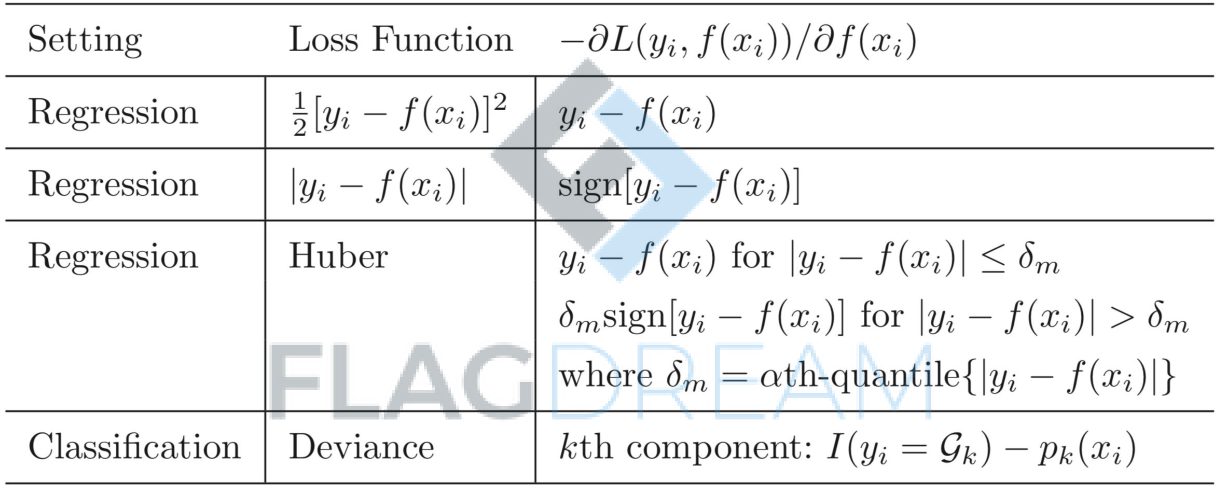
- Squared Loss: $L(y, f(x)) = (y - f(x))^2$
- At m step:
  - $(\alpha_m, G_m) = \arg\min_{\alpha_m, G_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \alpha G(x_i))$
  - $= \arg\min_{\alpha_m, G_m} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \alpha G(x_i))^2$
  - $= \arg\min_{\alpha_m, G_m} \sum_{i=1}^{N} (r_i - \alpha G(x_i))^2$, where $r_i = y_i - f_{m-1}(x_i)$
- r_i is the residual error of the previous step
- Just fit an estimator to the residual error

Algorithm

Input: training dataset $\{x_i, y_i\}_{i=1,...,N}$, a differentiable loss function $L(y, f(x))$, number of estimators M

```
1. Initialize model with a constant value: $f_0(x) = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha)$
2. Loop $m = 1, ..., M$:
    a. Compute pseudo-residuals: $r_i = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f(x)=f_{m-1}(x)}, \; i = 1, ..., N$
    b. Fit a base learner $G_m(x)$ to pseudo-residuals
    c. Compute $\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \alpha G_m(x_i))$
    d. Update model: $f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$
3. Output final model: $f_M(x)$
```

Loss Function



Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\|y_i - f(x_i)\|$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\|y_i - f(x_i)\| \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\|y_i - f(x_i)\| > \delta_m$ where $\delta_m = \alpha\text{th-quantile}\{\|y_i - f(x_i)\|\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

https://web.stanford.edu/~hastie/Papers/ESLII.pdf

Work + Interview

Summary

FLAGDREAM

	Logistic Regression	Neural Networks	SVM	Decision Tree	Random Forest	Gradient Boosted Trees	KNN
Natural handling of mixed data types (numeric and categorical)	⬇	⬇	⬇	⬆	⬆	⬆	⬇
Natural handling of nonlinear features	⬇	⬆	⬆	⬆	⬆	⬆	⬆
Natural handling of non-monotonic features	⬇	⬇	⬇	⬆	⬆	⬆	⬇
Robust to outliers	⬇	⬇	⬆	⬆	⬆	⬆	⬇
Insusceptible to noise in data (overfitting)	🙂	🙂	🙂	⬇	🙂	⬇	⬇
Computational scalability (large N)	⬆	🙂	⬇	⬆	⬆	⬆	⬇
Can output feature importance	⬆	⬇	⬇	⬆	⬆	⬆	⬇
Handling of missing values	⬇	⬇	⬇	⬆	⬆	⬆	⬆
Interpretability	⬆	⬇	⬇	🙂	⬇	⬇	⬇
Predictive power (1-5)	3	4	4	3	4.5	5	3
Ways to overcome overfitting	Regularization (L1, L2, elastic net)	Regularization (L2), drop out	simpler functions	Pruning, earlier stopping	Ensemble, average to reduce variance	Ensemble, boost weak learners to reduce bias	Choose larger K

Work + Interview

Python Coding

- Random Forest, Gradient Boosting, XGBoost
- Voting Classifier
- Stacking Ensemble