Traditional discriminative approaches

Nearest Neighbor

uses the nearest sample in the training set to determine the lable of the sample we want to classify/regress.

The k-nearest neighbor uses exactly the k closest point and averages their lables

$$\hat{y}_i = \frac{1}{K} \sum_{k \in N_k(x_i)} y_i$$

$N_k(x_i)$ is the neighborhood of x_i defined using a distance metric.

This algorithm performs a Voronoi tessellation of the training data. Therefore none of the training data are misclassified.

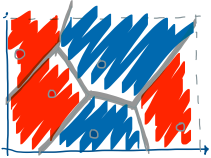
Wait... is this always the case? No, only for $K=1$. The error increases with K on the training set.

what about the number of parameters? worst case scenario

N/K

$\downarrow$

# samples on the training set



$K=1$

Any subregion of the decision boundary depends on a handful of points and their position. Therefore, it is wiggly and unstable $\longrightarrow$ high variance / low bias

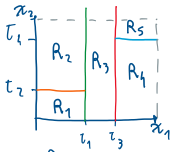
Tree based methods.

They partition the feature space into a set of rectangles and then fit a simple model (a constant) in each one.

Let's consider a simple regression Y given two features x_1 x_2 $\in [0,1]$.

We create a partition with a simple assignment:

$\{x_1=\underline{t_1}, x_2=\underline{t_2}, x_1=\underline{t_3}, x_2=\underline{t_4}\}$



$R_1 = x_1 \le t_1 \,\&\, x_2 \le t_2$

$R_2 = x_1 \le t_1 \,\&\, x_2 > t_2$

$R_3 = x_1 > t_1 \,\&\, x_1 \le t_3$

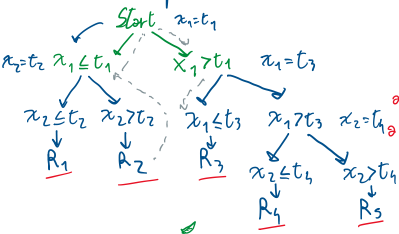
$R_4 = x_1 > t_3 \,\&\, x_2 \le t_4$

$R_5 = x_1 > t_3 \,\&\, x_2 > t_4$

$$y_i = \sum_{n=1}^{R_N} c_n I\{(x_1,x_2)\in R_N\}$$

↘ constant associated to eac region

This Procedure can be represented as a tree



The regions are the leaves of the tree and have associated a constant value

How do we grow our trees?

The algorithm needs to automatically decide the splitting variable and the splitting point and also the topology of the tree itself.

We will use a greedy algorithm.

Let's define two half planes $R_1(J,s)=\{x|x_J<s\}$

$R_2(J,s)=\{x|x_J\ge s\}$

$$\min_{J,s}\left[\min_{c_1}\sum_{x_i\in R_1(J,s)}(y_i-c_i)^2+\sum_{x_i\in R_2(J,s)}(y_i-c_2)^2\right]$$

$$\hat{c}_1=\frac{1}{M_1}\sum_{x_i\in R_1}^{M_1} y_i \qquad \hat{c}_2=\frac{1}{M_2}\sum_{x_i\in R_2}^{M_2} y_i$$

Therefore we scan through all the possible combinations (J,s).

Once we partition the data we repeat the process on each partition.

The size of the tree is a very important parameter.

A large tree might overfit the training data.

A small tree might not capture relevant pattern.

The solution is growing a big tree and then performing this pruning strategy.

T_0 is the original tree. $|T_0|$ is the cardinality of terminal node/Regions/Leaves

$N_m \# \{x_i \in R_m\}$ $\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$ $L(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$

↓ mse per region

We define the cost complexity

$$\underline{C_\alpha(T) = \sum_{m=1}^{|T|} N_m L(T) + \alpha|T|}$$

$\alpha \geq$ hyperparameter that governs tree size and goodness of fit.

What about classification?

$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$ → proportion of class k observations in Region m

$k(m) = \operatorname{argmax}_k \hat{p}_{mk}$ → Optimize crossentropy $-\sum_i^K \hat{p}_{mk} \log \hat{p}_{mk}$

Trees suffers of high variance, a small change in data can result in completely different trees

Boosting

Idea: combine the output of many weak classifiers to produce a powerful committee

This is different from bootstrapping and then bagging

Bootstrapping: a statistical resampling technique where multiple versions of the dataset are created by sampling with replacement.

Bagging: is an ensemble technique that leverages bootstrapping to improve performance and reduce overfitting.

1. Bootstrap the original dataset
2. Train a decision tree independently on each bootstrap
3. Aggregate the prediction using majority voting for classification or averaging the predictions for regression.

At step 2 each bootstrap tree will typically have different topology and regions.

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

Random Forests

Is a modification of the bagging strategy that builds de-correlated trees.

Facts: each tree is ~~independently generated~~ and identically distributed.
the expectation on the average of the bags is the same of the individual trees. Meaning, the only performance gain comes from the variance reduction.

An average of B iid random variables has variance $\frac{1}{B}\sigma^2$
if the variables are id (identically distributed) the variance of the average is

$\underline{\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2}$ where ρ is a positive pairwise correlation
$B \to \infty$ the second term disappears

Therefore ρ the amount of correlation among the trees limits the benefits.
Our goal is to limit this correlation. How we achieve it?
With a growing strategy.

For $b=1$ to B: (bootstrap of the original dataset)
1) Draw a bootstrap sample
2) Grow one random forest tree by repeating these steps until the minimum node size is reached
   i select $\underline{m<d}$ variables at random
   ii Pick the best split among those m variables
   iii Create two children nodes.

Now the output is the ensamble of trees $\{T_b\}_1^B$
To make a prediction at a new point:
Regression: $\hat{y}(x) = \frac{1}{B}\sum^B T_b(x)$
Classification: $\hat{y}(x) =$ majority voting $\{\hat{y}_b\}_1^B$

Hyperparameters: the inventor suggests $m = \lfloor\sqrt{d}\rfloor$ for classification ↘ $\min: 1$
$m = d/3$ for regression ↓ $\min: 5$

Back to Boosting

Idea: Instead of training multiple models independently, train them sequentially, each model focusing on the errors of its predecessor.

In short: 1) train a simple decision tree
2) Analyze its errors identifying the missclassified samples
3) Train a new tree giving more weight to those samples
4) Repeat
5) At the end combine all trees → using weighs

This time we improve both bias and variance.
ADABoost

1 $w_i = 1/N$ N number of observation

2 for $m = 1$ to M: number of models

  i fit a classifier $f_m(x)$ using weights w_i

  ii compute the error : $e_m = \frac{\sum^N w_i I(y_i \neq f_m(x_i))}{\sum^N w_i}$

  iii compute $\alpha_m = \log((1-e_m)/e_m)$ Low error higher α

  iv update the weights $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot I(y_i \neq f_m(x_i)))$

3 $f(x) = \text{sign}\left(\sum^T \alpha_m f_m(x)\right)$

What about Boosted Trees?

Let's use regression. Remember the decision rule :

$$x \in R_j \rightarrow f(x) = \gamma_j$$ constant value for each region

We can express a single tree as: $T_\theta(x) = \sum^J \gamma_j I(x \in R_j)$

$\theta = \{R_j, \gamma_j\}_1^J$. The best parameters are found minimizing the empirical risk as usual

$$\theta^* = \arg\min_\theta \sum_1^J \sum_{x_j \in R_j} L(y_i, \gamma_j)$$

Finding γ_j is trivial: $\hat{\gamma}_j = \frac{1}{N_j} \sum_i^{N_j} y_i$

Finding R_j is more difficult and requires greedy strategies

For any boosted model we have the following optimization problem

$$\theta_m^* = \arg\min_{\theta_m^*} \sum^N L(y_i, \underline{f_{m-1}(x) + T_{\theta m}(x)}) \quad \text{where } f_{m-1}(x) = \sum_{i=1}^{M-1} T_{\theta_i}(x)$$

given the Region R_{jm} found greedyly

$$\gamma_{jm}^* = \arg\min_{\gamma_{jm}} \sum_{x \in R_m} L(y_i, \overset{\overset{\gamma_{jm-1}}{\uparrow}}{f_{m-1}(x)} + \gamma_{jm}).$$

We are fitting a tree on the residuals of the current model. $(y_i - (f_{m-1} + \gamma_{jm}))^2$

Trivial cases

$L =$ squared error γ_{jm} is the mean of the residuals in R_{jm}

$L =$ exponential loss γ_{jm} is weighted log-odds of class proportions in R_{jm} (Adaboost)

Gradient Boosting

In each step we find a new tree $T_{\theta m}(x)$ that best reduces the loss given the current model.

We solve a local optimization for the current step without thinking about future trees. Isn't it similar to something we know very well?

Boosting is very similar to gradient descent with few caveats.

Gradient Descent

Given $L(y, f(x))$ $-\nabla_\theta L$ is the direction that most rapidly reduces L we use it to update the weights.

Gradient Boosting

$-\nabla_\theta L$ is the target of the next tree. $T_{\theta m}(x)$ is fitted to minimize the squared error between its prediction and the gradient

$$\theta_m = \arg\min_\theta \sum_i^N \left(-\nabla_\theta^m(x_i) - T_\theta(x_i)\right)^2$$

after we fit this tree we update the model $f_m(x) = f_{m-1}(x) + T_{\theta m}(x)$

In regression $-\nabla_\theta = y_i - f_{m-1}(x)$

In classification $-\nabla_\theta^k = I(y_i = k) - p_k(x)$ → predicted probability for class k

↓ k separated trees are built at each iteration

What about XGBoost?

XGBoost introduces a regularization term hyperparam

$$J = \sum_i^N L(y_i, \hat{y}_i) + \sum_{m=1}^M \Omega(T_m) \rightarrow \gamma |T| + \frac{\lambda}{2} \sum^{|T|} w_j^2$$

It modifies also the prediction update introducing η a learning rate that works as a shrinkage factor to reduce the contribution of newer trees

$$f_m(x_i) = f_{m-1}(x_i) + \eta T_m(x_i)$$

Moreover it uses both first order and second order derivatives

$$g_i = \nabla_\theta^{m} = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{(m-1)}(x_i)} \qquad h_i = \nabla_\theta''^{m} = \frac{\partial^2 L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)^2}$$

g_i and h_i drive the splitting criterion

$$\text{Gain} = \frac{1}{2}\left[\frac{\left(\sum^{R_\ell} g_i\right)^2}{\sum^{R_\ell} h_i + \lambda} + \frac{\left(\sum^{R_r} g_i\right)^2}{\sum^{R_r} h_i + \lambda} - \frac{\left(\sum^{R} g_i\right)^2}{\sum^{R} h_i + \lambda}\right] - \gamma$$

γ → Minimum gain required to make a new split

w_j^2 =

R_ℓ left child Region

R_r Right child Region

R current Region

XGBoost prunes also branches with low gain to avoid overfitting

The model is updated: $f_m(x) = f_{m-1}(x) + \eta T_m(x)$