P1 → sketched
P2 → ~~approp~~ original.
run.sh
→ In P1 & P2 both. $\sum |y - \hat{y}|$

$(X^T X)^{-1} \sigma^2 = var(\hat{w})$

ssh all name @ 10.192.13.11

Model : $y = w_0 + w_1 x + 6$ → data derived from this model (St line with added noise).

Q1 How close $\hat{w}_1$ & $\hat{w}_2$ are to true values w_0 & w_1?

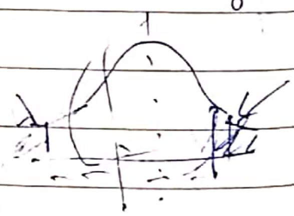
$$\hat{w}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x}$$

Use variance formulae to compute confidence intervals :

$$var(\hat{w}_1) = \frac{\sigma^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

0.05



$$var(\hat{w}_0) = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \right]$$

↗ Use var formulas to derive how good are estimates $\hat{w}_0$ & $\hat{w}_1$

- With approx 0.95 prob, the interval

$$\left[\hat{w}_1 - 2\sqrt{var(\hat{w}_1)},\ \hat{w}_1 + 2\sqrt{var(\hat{w}_1)} \right]$$

If we change 0.95, this changes (interval len)

The smaller the variance, the more confident we are that w_1 is in this interval.

$y_i \rightarrow$ uncorrelated
var of $y_i \rightarrow$ constant $= \sigma^2$
$x_i \rightarrow$ fixed (non random)

$\sum x_i^T x_i$

Page No.
Date / /20

Approximation here it assumes normality.
Hastie & Tibshirani chapter 3.

Std error → Std deviation.
If std error is small, our estimate is good.

→ Use similar procedure to obtain confidence set for the parameter vector w [Hastie Tibshirani chap 3]

Q) How to check if there exists a relationship between Y and X?

Ans → Use hypothesis testing.

Credit Data set.

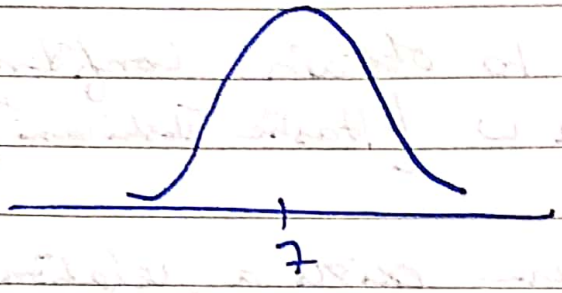
- # get summary of data set (min, 1st Qu, median, etc)
plot density also.
remove outliers (by plotting density & seeing some abnormality)
Bar chart also helps
histogram
Scatter plot of different variables
(looking at correlation plots, we get a good idea, what are the variables that are related)

correlation plots

lm(rating, lit), lm → 0.99
lm of output y with each individual input var

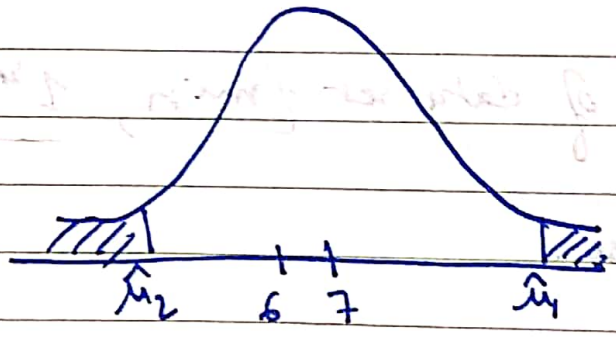
There is other way called hypothesis testing (rather than drawing plots).

our hypothesis was that a batch has avg cgpa of any class is 7.



Now a new batch comes with avg cgpa was 6. (it rejected our hypothesis)

$\alpha = 0.05$ → level of significance.
$\alpha = 0.01$



shaded = 0.05 if $\alpha = 0.005$

If avg is in the shaded region, then our hypothesis cannot be rejected.

hypothesis assumed to be true → null hypothesis.

We need good amt of evidence to prove that null hypothesis is false.

Page No.
Date / /20

- Null hypothesis, H_0 :- There is no relationship between X and y.

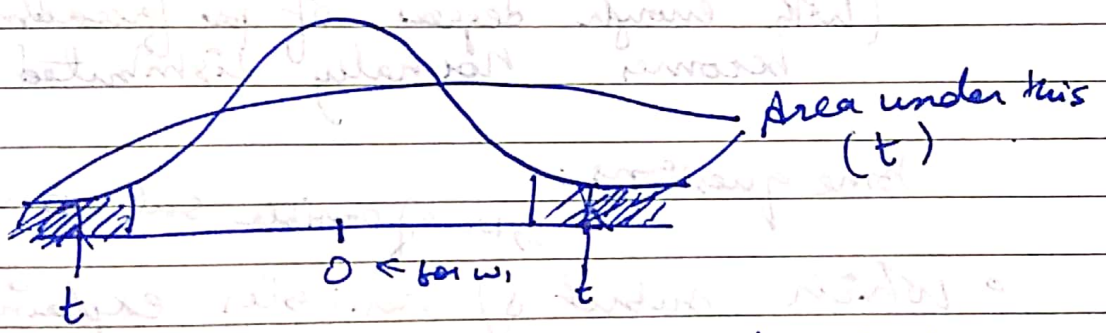
$$H_0 = w_1 = 0$$

So, $y = w_0 + \cancel{\hat{w}_1 x}$ (should vanish)

- Alternate hypothesis, H_a: There is some relationship between X and y

$$H_a : w_1 \neq 0$$

Did we get non zero w_1 by chance or is there a (really) relationship between y & X.



To test null hypothesis, we need to check if $\hat{w}_1$ is sufficiently far from 0.

(decided by level of significance.)

(Everything true because of normality assumption)

Use t-statistic: $t = \dfrac{\hat{w}_1 - 0}{\sqrt{\text{var}(\hat{w}_1)}}$ ← what we are assuming w_1 to be.

Either reject the null hypothesis or do not reject.

Compute p-value using for large sample using Normal distribution. p-value : $Pr(|w_1| \geq |t|)$

If p-value is less than the level of significance, then we can reject the null hypothesis. (then $w_1 \neq 0$, then there exist a relationship between X & Y)

A small p-value $\Rightarrow$ reject null hypothesis & $\Rightarrow$ association between predictor & response exists.

Here it is a two sided test.
Here we assumed w_1 is normally distributed. (With enough degrees of freedom, t becomes normally distributed).

Some questions:

- Which subset of variables explains the response? → No of possible subsets very very large
- How good is the designed model? → we can use least squares error for this.

$w_0 + w_1 x(1) + w_2 x(2) \cdots + w_d x(d)$

Null hyp → all $w_1 \ldots w_d$ are 0.
Alternate " → atleast one of them is non zero.

If only w_0 = non zero
→ w_0 → mean of target.

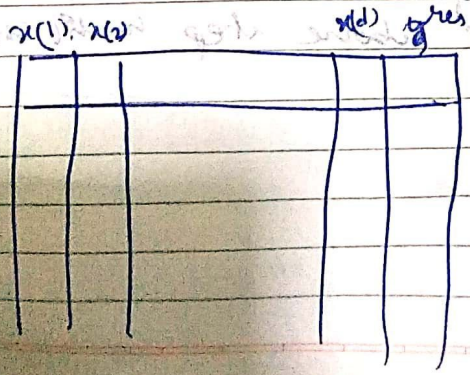
Some practical approaches:

- Forward stepwise (greedy algo) → gives suboptimal solution (not best optimal solution) Taking all subsets not practical.
  → Start with only the intercept term in the model.
  → Add the feature that improves the fit the most.
  → Repeat until the above until some stopping condition satisfied.

  keep checking if new feature reduces the least squared error.
  If it changes error significantly, we keep it. → when LSE does not improve.
  when we get a flat line, we stop.
  Every time we train the model.

- Backward stepwise
  → Start with complete model.
  → ~~Start~~ Remove the feature having the least impact on the model performance
  → Repeate the above step until some stopping condition is satisfied.

- Stagewise.



1st → which var correlated with y.
2nd → " " " with residual.

once $h_1(x)$ is designed, it will be fixed

$$y - h_1(x)$$
$$\| $$
$$y - h_1(x) + h_2(x)$$

since we are retraining, a variable previously selected may drop out (Not in stagewise)

→ In forward stepwise → everytime we add the feature, parameter change. & we find correlation ~~with~~ between y & variables.

In forward stagewise →

- Forward stagewise :
  - → Start with the only intercept term in the model
  - → Identify the variable most correlated with the residual and find its linear regression coeff of the residual
  - → Repeat the above step until stopping condition satisfied

Although computationally expensive gives good result.

Assume: All variables have 0 mean. (All the cols have zero means)

① First model (using "intercept only"): $h_1(x) = \bar{y}$, $t = 1$

② while stopping condition is not satisfied

③ ~~while~~ → Find the variable most correlated with $y - h_{[t]}(x)$, say x_{k_t}

all integers upto t.

$h_{[t]} \hookrightarrow h_1(x) + h_2(x) + \ldots + h_t(x)$

④ → Design a regressor using the data

$$\{(x_1(k_t), y_1 - h_{[t]}(x_1)), (x_2(k_t), y_2 - h_{[t]}(x_2)), \ldots, (x_n(k_t), y_n - h_{[t]}(x_n))\}$$

to get

$$h_{[t+1]}(x) = h_{[t]}(x) + w_{k_t} x(k_t)$$

← verified that it gives very good result.

→ $t = t+1$

Endwhile

We introduce that variable that reduces the residual. ↳ Stopping condition is met.

or keeps coming down till

$res_1 = y - h_1(x)$

$res_2 = y - h_1(x) - h_2(x)$

$\vdots$

$res_t = y - h_{[t]}(x)$

If at any step, error

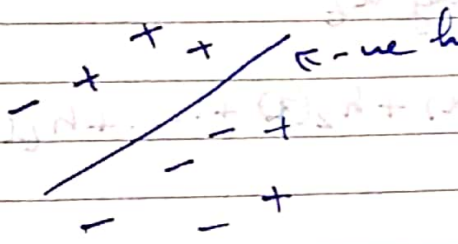
where a variable correlated to residual that reduces it.

one dim optimization problem (easy to solve)

Mean of each col is assumed to be zero, so that finding correlation between x & y residual is easy.

Other ways are ridge regression,
lasso ",
etc, that automatically selects the feature.

Decision Trees & Ensemble Methods

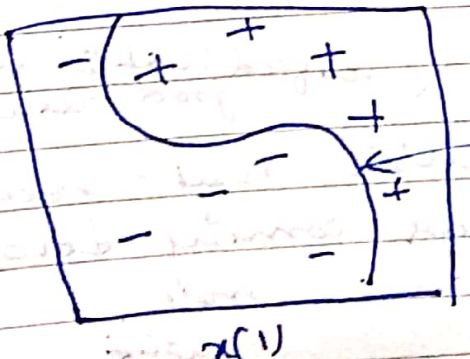


Decision fn: Linear

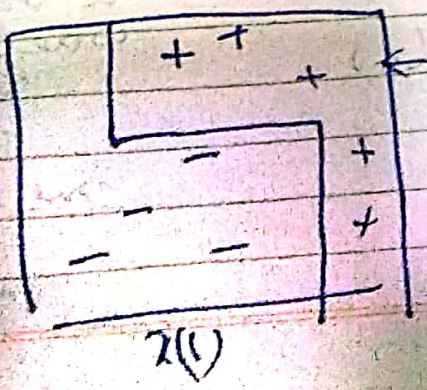
Here least squares can also be used. y will be +1 or -1.

If an example plugged in the eqn, if it is +ve, it is in +ve class, if -ve, in -ve class.

This is not interpretable (so decision trees used) mimics our way of thinking. It infers some easily inferible rules)



Non linear fn approximated by piecewise fns.



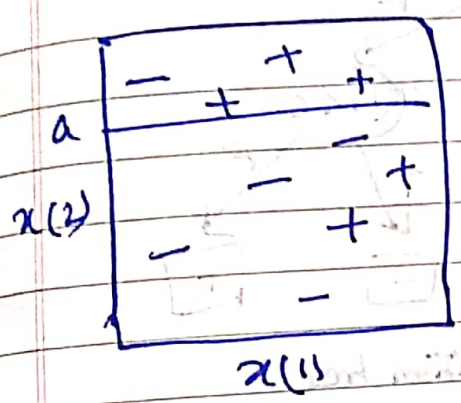
If blood pressure > something then abnormal

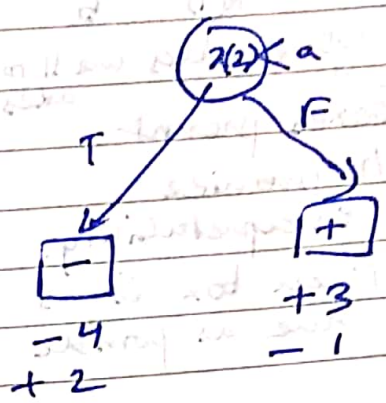
e.g. If blood sugar x(1) is > something it is abnormal.

Page No.
Date / /20

Decision tree other application - bank loan sanction.
↳ handy for interpretability.

Decision stump → height 1 decision tree



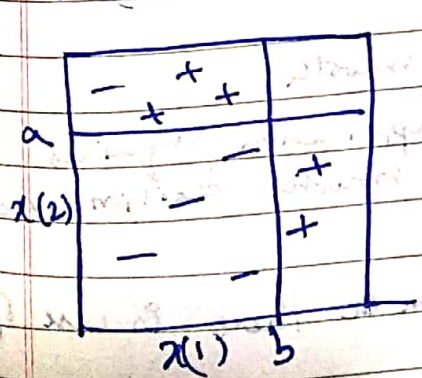


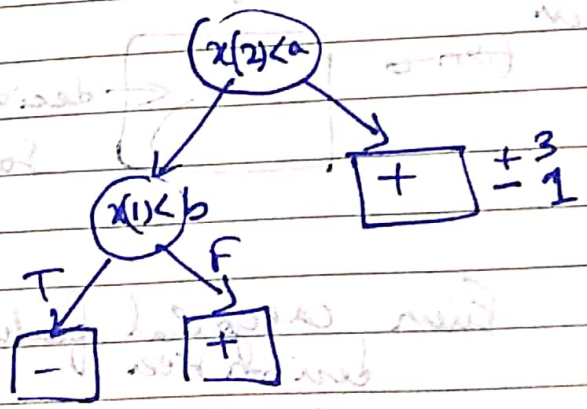
Decision stump

we can split along the horizontal side as well.

here we have made some mistakes (2 +ve misclassified, 1 -ve misclassified)

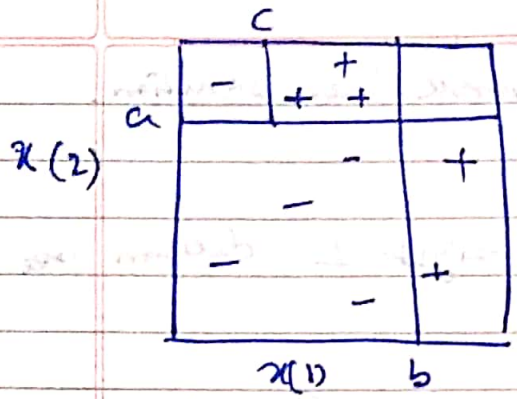
Most imp thing is to get the parameter a.
Use any appropriate feature & split the input space. Split is such that it is parallel to the coordinate axis.





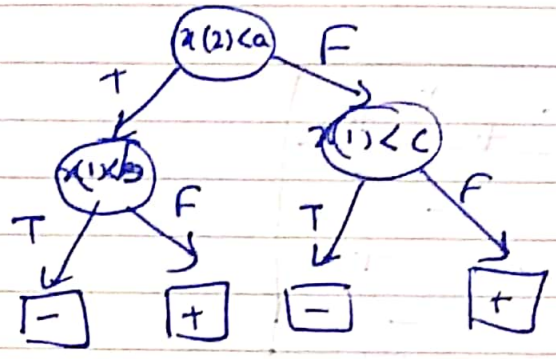
Decision tree

Leaf nodes → pure nodes, (no example misclassified)
There is a class associated with each leaf node.



We divide input space into multiple regions such that each region is as pure as possible

⇒ Boxes present whose edges are ll to axes to provide interpretability. Each box is as pure as possible

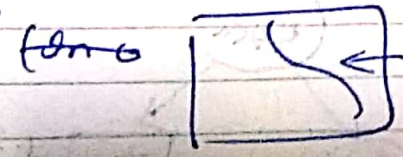


Decision tree

- Features used at nodes should be imp features
  (decision trees – a way to select imp features)
  way of doing classification
  can be used for multiclass classification.
  Depends on application whether 2 or more branches should be there.

→ Decision surface is non smooth here.

deterministic process



← decision for smooth.
Some application requires smooth decision surfaces.

Even categorical feature can be there in case of decision tree

We can stop at any level in decision tree.
Possibility to prune some parts of trees
(then some features will become irrelevant)
By pruning, we reduce the height of tree.

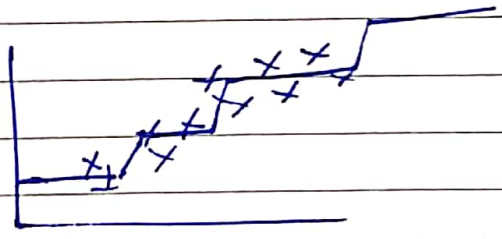
Sometimes tree is grown from max possible depth & then start pruning ⊖ until tree feels fit.

Categorical variables - take specific values.

Previously we could work with only real or Integer value data.
Here with categorical.
Normalization not reqd before decision tree is created.

Decision tree can be used for both classification & regression problem.
↳ Divide Input region into multiple regions, & treat each region as having constant fn.
(Discontinuity may be there) At that part of region things may not be continuous.
we could have used non linear fn that could have been smoother.

↑ Disadv of using decision tree.



Decision tree

A decision tree is a hierarchical structure, with each node splitting the data space into disjoint partitions based on feature value

- Input space X is partitioned into disjoint regions, $\{R_1, R_2, \ldots, R_m\}$ e.g. (each region belongs to class)
- Boundaries of regions parallel to coordinate axes in classical decision trees

Choose a feature that will reduce the impurity. (entropy as low as possible)

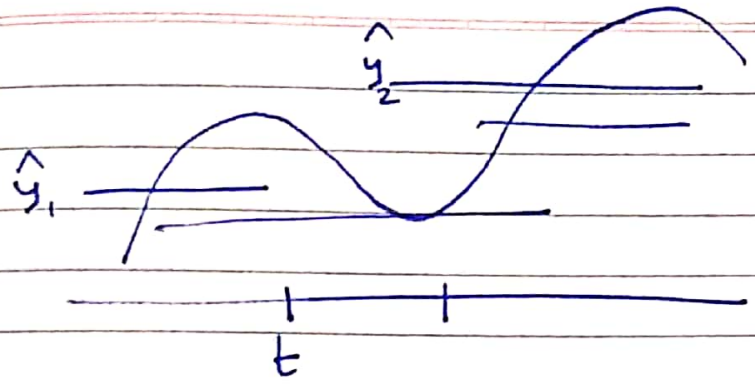
Initially S had some entropy.

We want to find a feature that gives minimizes the max expected reduction in entropy

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

expected entropy based on feature A.

expected reduction.



Among all possible t's we find one that gives least value of obj fn.
This problem is hard, so we divide into regions and then find the mean to find t^*.
We divide into regions by clustering.

In case of least squares error, we keep on reducing error.

But in case of decision tree, there is ~~no~~ back tracking.

Early stopping possible or post-pruning. (to avoid overfitting)

If in case of validation set performs well at any stage, we can stop.
↙ i.e. we reached a flat state

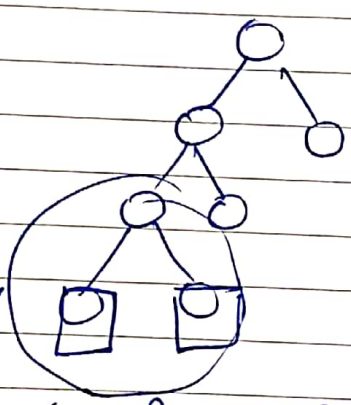
Validation set used to know when to stop training

Train	Val	Test

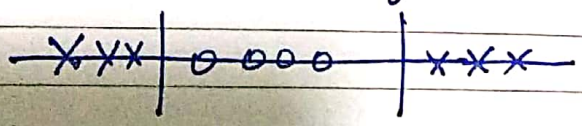
Early stopping → As we grow the tree, we test the performance on validation set

Performance on validation set is crucial.

Post pruning →



→ Continuous valued features can be handled by converting them to discrete-valued features.

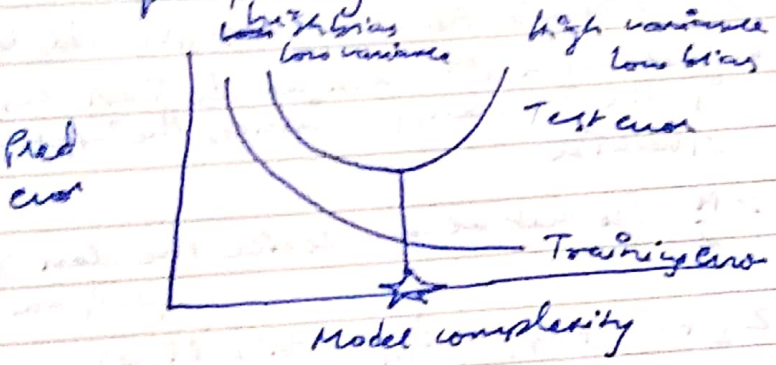


→ Possible to handle training set examples with missing feature values.
→ One can be use mean
→ Other can be put them with some probability. probability of them being present based on some previous experience

Ch-9 Hastie Tibshirani

→ Use of alternate measures for finding the "best" feature to split on:
- → Gini Index: $2p_{\oplus}p_{\ominus}$
- → Mis classification error: $1 - \max(p_{\oplus}, p_{\ominus})$
- → Squared error (for regression): $\sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - \hat{y}_m)^2$

Banana dataset (2 dim)
Decision tree classifier (overfitting)
- Do post pruning



Ensemble Methods

- Make an informed decision after consulting experts having diverse domain knowledge
  - → Admission to university
  - → Making investment in a company's stocks.

- "large groups of people are smarter than an elite few"
  [The wi

- When are the crowds wise?
  - → Diversity of opinion
  - → Independence (unbiased)
  - → Decentralization (every person has access to some private info not accessible to others)
  - → Aggregation

- Are there ideas useful in m/c learning?
  → Combining the decisions of a diverse set of learners
  → Committee m/cs – Boosting, mixture of experts.

- $D = \{(x_i, y_i)\}_{i=1}^{n}$, $y_i \in \{-1, +1\}$
- M classifiers trained independently on different views of D.
- $p = Pr$ (a classifier correctly classifies an example).
- Assumption: For every classifier, the same p holds and $p > 0.5$ (otherwise random classifier will be better)
- Z_i : RV associated with i^{th} classifier defined as
  $Z_i = \begin{cases} 1 & \text{if the } i\text{th classifier classifies the input correctly} \\ 0 & \text{otherwise.} \end{cases}$

~~Also~~ $Z > \frac{M}{2}$ (so that we can decide the class)

- $S_M = Z_1 + Z_2 + \dots + Z_M$ ($S_M \sim Bin(M, p)$)
  ↑ No of trails (n)
  ↳ prob of getting class +1

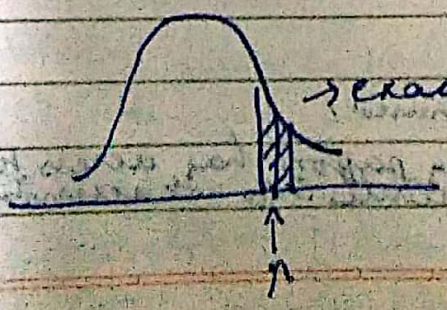
- Prediction: Feed input x to all M classifiers and take a simple majority vote (need at least $\lceil \frac{M+1}{2} \rceil$ votes for correct prediction)
  (all Z_s are iids)

check $Pr(S_M = n) = \binom{M}{n} p^n (1-p)^{(M-n)}$

$$= \phi\left(\frac{n + \frac{1}{2} - Mp}{\sqrt{Mp(1-p)}}\right) - \phi\left(\frac{n - \frac{1}{2} - Mp}{\sqrt{Mp(1-p)}}\right)$$

↑ Area under the curve (assuming normal distribution.

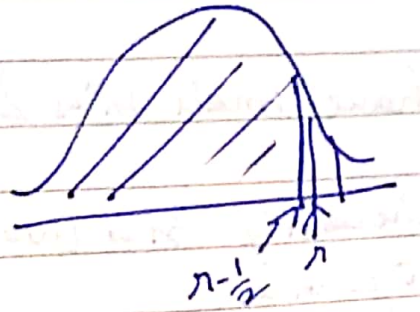
→ exact value at a pt = 0
so we take area under the slab around n.



- Probability of getting at least r votes :

$$P_r(S_M \geq r) \approx 1 - \phi\left(\frac{r - \frac{1}{2} - Mp}{\sqrt{Mp(1-p)}}\right)$$

From normal tables we can get these values.



Q Find P_r(correct classification of x using majority vote).

Some examples :

- Let $M = 15$, $p = 0.6$ We want at least 8 classifiers giving correct

$$P_r(S_M \geq 8) \approx 1 - \phi\left(\frac{6 - \frac{1}{2} - 9}{\sqrt{(15)\cdot(.24)}}\right) = 0.785$$

- Let $M \xrightarrow{\text{around}} 51$, $p = 0.6$

$$P_r(S_M \geq 26) \approx 1 - \phi\left(\frac{26 - \frac{1}{2} - 30.6}{\sqrt{(51)\cdot(.24)}}\right)$$

Each one doing just better than random of n classifiers.

Remarks :

- As M increases, $P_r\left(S_M \geq \left\lceil \frac{M+1}{2} \right\rceil\right)$ increases slightly (though rate of increase decreases slowly).
- $p > 0.5 + \delta$ where δ is small & +ve could be even 0.51 also (just has to be better than random classifier)
- Weak classifiers : Perform slightly better than random classifier
  - ↳ e.g. decision stump (decision tree of height 1) (just better than random classifier)

- Possible to combine weak classifiers so as to boost their performance.
  decision tree
  Ensemble Methods
- Sufficiently Deep trees have small bias & high variance.
- Reduce variance w/o increasing bias (too much).
  Decision stump → high bias
  Decision tree to max depth → high variance.

Idea: Averaging reduces variance

Let $Z_1, Z_2, \dots Z_M$ be i.i.d RVs such that

$$Var(Z_i) = \sigma^2 \quad \forall i$$

Define $Z = \frac{1}{M}\sum_{j=1}^{M} Z_j$. Then

$$Var(Z) = \frac{\sigma^2}{M}$$

$Var\left(\frac{1}{M}(Z_1 + Z_2 \dots Z_M)\right)$

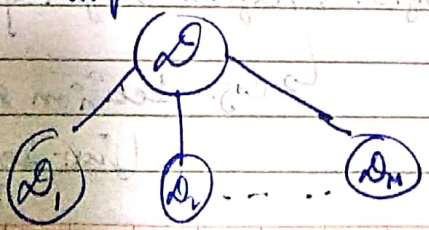
$= \frac{1}{M^2}(Var Z_1 + \dots Var Z_M)$

$= \frac{M\sigma^2}{M^2}$

- Idea: Train multiple models & average their outputs
  Give one train data, only one decision tree possible
  How to train decision trees (decision trees).

Q) Given one training set D, how do we train multiple models?

Soln: Generate M bootstrap samples from D and train a different component of a base model using each bootstrap sample.



bootstrap sampling
bootstrap sampling means randomly with replacement

Page No.
Date / /20

- Bootstrap sample:
  → Draw a sample of size $|D|$ with replacement from D
  → e.g. $D = \{x_1, x_2, x_3, x_4, x_5\}$
  - Bootstrap sample 1: x_3, x_2, x_3, x_1, x_1; $D_1 = \{x_1, x_2, x_3\}$.
    (every sample may not have all examples)
  - Bootstrap sample 2: x_1, x_5, x_2, x_4, x_4; $D_2 = \{x_1, x_2, x_4, x_5\}$
  - Bootstrap sample 3: x_3, x_4, x_4, x_3, x_4; $D_3 = \{x_3, x_4\}$.
  - and so on....

we can generate as many bootstrap samples as we want.

→ Remarks:
- Each model trained on less data

$\left(1 - \frac{1}{n}\right)^n$ → prob of an example not being picked

as $n \to \infty$, $\left(1 - \frac{1}{n}\right)^n \to \frac{1}{e} \approx 33\%$.

Every bootstrap sample will be picked with 65% prob.

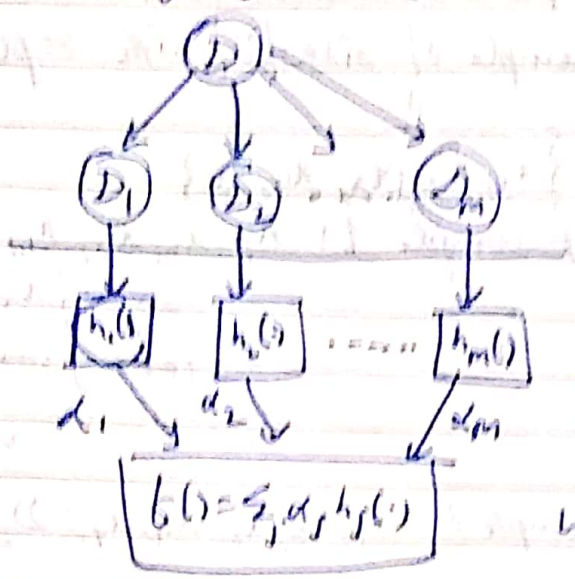
- Every data sample is seen by at least one of the models if M is reasonably large.

→ Use weighted average model for prediction.

Bootstrap Aggregation - Bagging

decision trees unstable models

Bagging → only unstable models (high variance) to create bias



ensemble of models

weighted average model (used for prediction)

Works well if base models are unstable (e.g. decision trees)

- Training Data: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$
- Single model learning: Design a $h: \mathcal{X} \to \mathcal{Y}$ using $\mathcal{D}$
- Ensemble learning: Design an ensemble $\{h_1(\cdot), h_2(\cdot), \ldots, h_m(\cdot)\}$ using $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$ respectively where

  $h_j: \mathcal{X} \to \mathcal{Y}$, $j = 1, \ldots, M$

- Inference for test input x
  - ↳ Binary classification

$$\hat{y} = \text{sign}\left(\sum_j \alpha_j h_j(x)\right)$$

→ Regression $\hat{y} = \sum_j \alpha_j h_j(x)$

trees unstable
model.
models (high variance)
works here

emble of models

ghted average
Model
used for prediction,

$\in \mathcal{Y}$, $y_i \in \mathcal{Y}$

using $\mathcal{D}$

$\{h_1(\cdot), h_2(\cdot),$
$\ldots, h_M(\cdot)\}$
where

Bagging

Algorithm

Input: Data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, M: No of models to be built

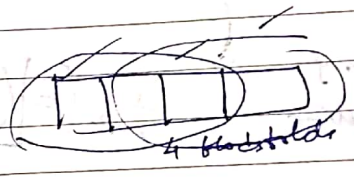
(1) for $j = 1, \ldots M$
  (a) Take a bootstrap sample $\mathcal{D}_j$ of size n from $\mathcal{D}$
  (b) Build a model $h_j(\cdot)$ using $\mathcal{D}_j$.
  end for

Output: A set of M models

$$H = \{h_1(\cdot), h_2(\cdot) \ldots, h_M(\cdot)\}$$

Inference: For an unseen example x, predicted label $\hat{y}$ is calculated as

- Regression: $\hat{y} = \frac{1}{M} \sum_{j=1}^{M} h_j(x)$
- Classification: $\hat{y} = \arg\max_i |C_i|$
  (where C_i



Bagging

- Bagging with k-fold cross-validation
  → train a model on (k-1) folds with early stop on the remaining fold.
  → k models available at the end of k-fold cross-validation

Consider the regression data set derived from the true fn $f(x)$ and noise ϵ.

- Suppose we have M bootstrap data sets & train M models, $y_m(x)$
- $y_m(x) = f(x) + \epsilon_m$
- Average error made by M models acting individually, is

If we have M independent models, avg expected error

$$E_{avg} = \frac{1}{M}\sum_{m=1}^{M} E_x\left[(y_m(x) - f(x))^2\right]$$

$$= \frac{1}{M}\sum_{m=1}^{M} E_x\left[\epsilon_m(x)^2\right] = \frac{1}{M}\left(E[\epsilon_1^2] + \dots E[\epsilon_m^2]\right)$$

- The output of the bagged model is

$$y_{bag} = \frac{1}{M}\sum_{m=1}^{M} y_m(x)$$

- The expected error from the bagged model is

$$E_{bag} = E_x\left[(y_{bag} - f(x))^2\right]$$

$$= \frac{1}{M^2} E_x\left[\left(\sum_{m=1}^{M} \epsilon_m(x)\right)^2\right]$$

Average expectation

$$= \frac{1}{M} E_{avg} \quad \text{(for uncorrelated errors)}$$

$$E[\epsilon_i(x)\,\epsilon_j(x)] = 0 \quad \text{if } i \neq j$$

with bagging reduction

$$\frac{1}{M^2}\left(E\left[(\epsilon_1 + \epsilon_2 + \dots \epsilon_m)^2\right]\right)$$

↓

cross terms will also be there. (If errors are uncorrelated, all the cross terms will vanish.)

No guarantee that we get uncorrelated examples in $D_1, D_2 \dots D_m$ because bootstrap samples are not totally unrelated.

- Models are uncorrelated and so variance reduction is smaller than $\frac{1}{M}$ (difficult to completely ~~Independent~~ uncorrelated training sets)
- Not suitable for stable learning methods (as they typically have low variance) (A model is unstable if small changes in the data can cause changes in the predicted values)
- Has little effect on bias.

Useful technique for large data sets & unstable models.

Bootstrap samples are actually correlated

Page No.
Date / /20

- Models are correlated ⇒ variance reduction is smaller.

ρ : correlation between every pair of models.
σ^2 : variance of each model.

$$\text{Var}\left(\frac{1}{M}\sum_{m=1}^{M} Z_m\right) = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} \text{cov}(Z_i, Z_j)$$

$$= \frac{1}{M^2}\sum_{i=1}^{M}\left(\sum_{j=1, j\neq i}^{M} \text{cov}(Z_i, Z_j) + \text{var}(Z_i)\right)$$

$$= \frac{1}{M^2}\sum_{i=1}^{M}\left((M-1)\rho\sigma^2 + \sigma^2\right)$$

$$= \frac{M(M-1)\rho\sigma^2 + M\sigma^2}{M^2}$$

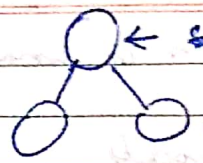
$$= \rho\sigma^2 + \frac{(1-\rho)\sigma^2}{M}$$

decreases if $\rho \downarrow$ $\rightarrow 0$ as $M \uparrow$

- If we used model averaging for variance reduction, we want
  - → Large no of ~~mod~~ models
  - → Low correlation between them.

Random forest
↳ At every node look at a random subset of features from which we choose the best feature.
If we'll make sure that models are uncorrelated to each other.

- Idea : Use decision tree with different set of features (To reduce correlation between models.



← ~~Set~~ take a set a n features & take best feature

← At every node, take a random feature subset & among them find the best

Approach: When learning a split, find the best feature on a reduced random feature subset.

- RandomForest
  - → Bootstrapping
  - → Random selection of features
  - → Ensemble of decision trees

Algorithm:

All user defined

Input: d-dimensional data $D = \{(x_i, y_i)\}_{i=1}^{n}$, M: Number of models to be built, m: No of features used at each node, mtd: minimum tree depth

(1) for $j = 1, \ldots, M$ ($|D_j| < |D|$)

(a) Take a bootstrap sample D_j of size n from D

(b) Grow a tree T_j using D_j by recursively repeating the following ~~sets~~ steps at each node of the tree, until the min tree depth mtd is reached.

(i) Draw a random subset Q of size m from $\{1, 2, \ldots d\}$

~~(Then for~~

(ii) Pick the "best" variable from Q (based on information gain)

(iii) Split the node into appropriate no of children nodes

end for.

(Helps in generated better uncorrelated trees than the simple bagging model)

Output: A random forest of M Decision Trees

$RF = \{T_1(\cdot), T_2(\cdot), \ldots, T_M(\cdot)\}$

Combine
Every tree
Disadv
Here Info
if Info

On sing
axes.
Here
But

To d

→ keep
per

→ Ca

Page No.
Date / /20

Combine RF just as in bagging.
Every tree generated, we use different set of features.

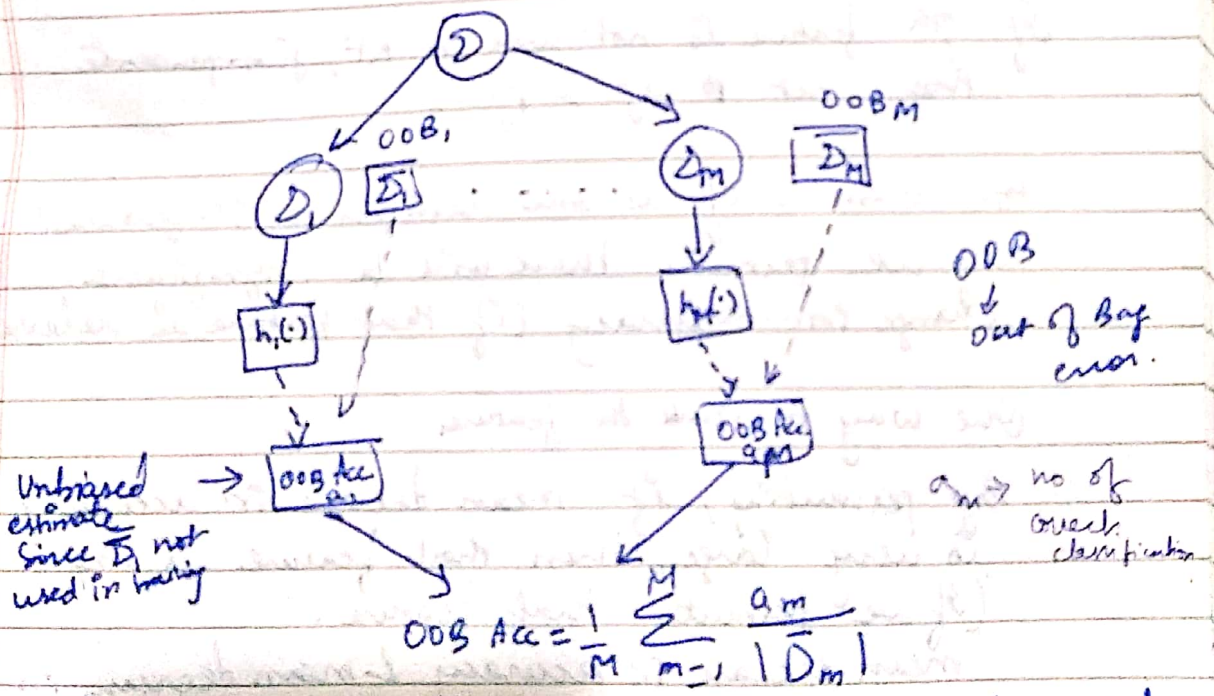
Disadv
Here Interpretability part of traditional tree is lost. But if Interpretability is not needed this can be used.

In single decision tree all the boundaries are ∥ to axes.
Here need not be ∥. So interpretability part lost.
But it is still a good model.

To decide how many trees in Random Forest?

→ keep a Validation set. keep adding a model until performance deteriorates

→ Can also work with only training set.



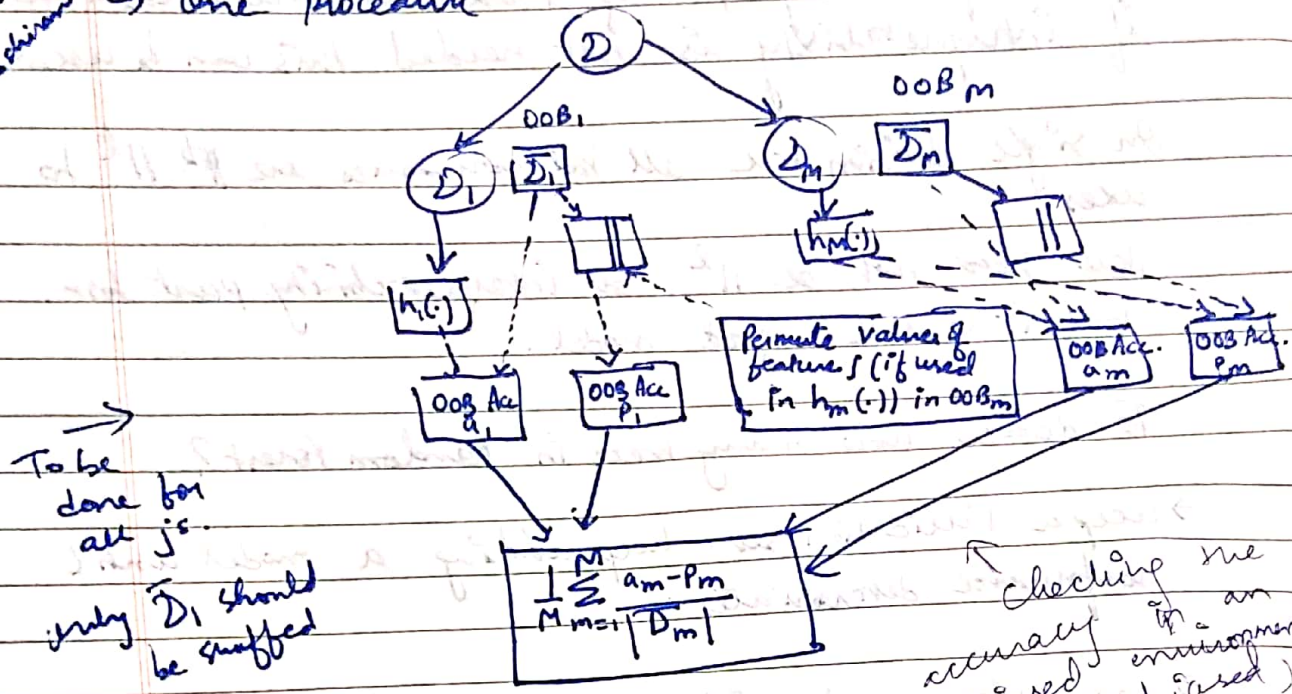
Whenever we add a new model, check OOB Acc when OOB Acc stops increasing, stop

2nd assignment: Build a random forest regression on same dataset.
no of trees, features to use, min tree depth, etc tune them

Random forest can also be used for feature selection.

How to find whether features is good or not?

Basic Technique → One procedure



If jth feature is not used in RF, just ~~permute~~ then set a_j & $p_j = 0$.

How sensitive is our classifier to jth feature?
If we permute there will be significant change in accuracy (if that feature is relevant).

One way to rank the features.
By permuting, if mean decrease in accuracy is very large, then that feature is imp
(If we permute that feature.
mean decrease in accuracy & mean decrease in gini index.

feature selection → Lasso, PCA, decision tree, random forest

Q Why RF is used over decision tree for feature selection?

Page No.
Date / /20

Many times ppl use RF to select features & then give it to linear model.

- Bagging reduce variance by averaging
- Is it possible to average models & reduce bias?

Solution: Boosting

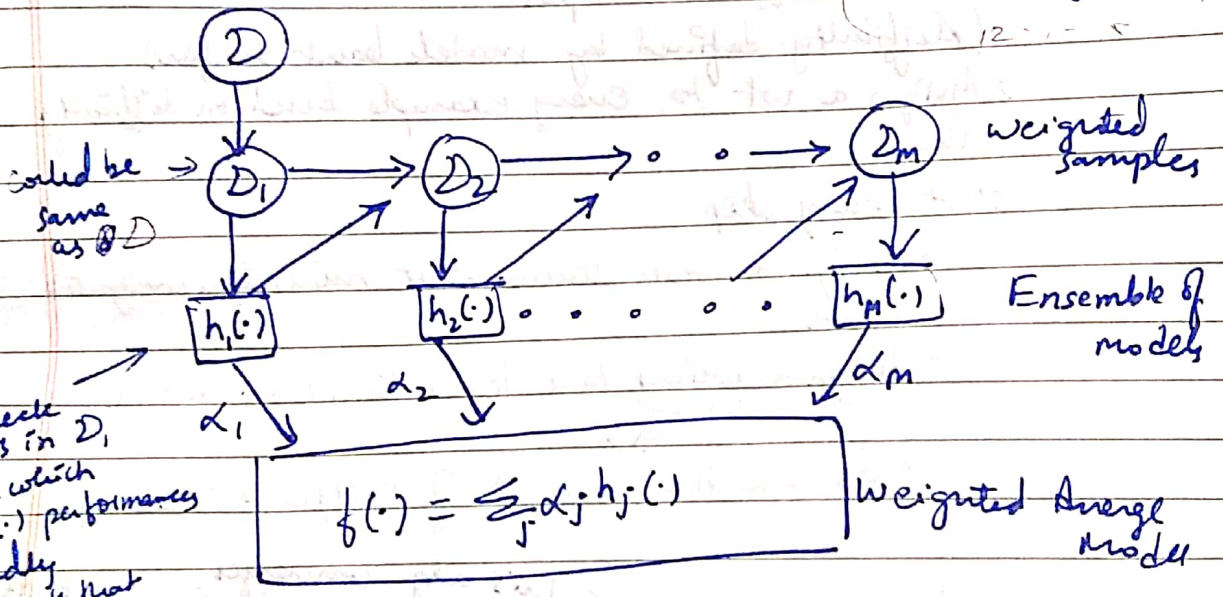
Average weak models & then boost the performance.

high biased

Boosting → similar to bagging.

Boosting

datasets not independent anymore



check pts in D_1 on which $h_1(\cdot)$ performance badly (i.e. pts that are difficult to classify)

Then

Stage wise approach.
once α_1, h_1 decided, they become fixed.

we can assign wt to every example.
Difficult to classify, wt ↑,
easy " " , wt ↓

Initially we don't know which is difficult & which not. So assign equal wts. Increase the wts incorrectly classified by $h_1(\cdot)$

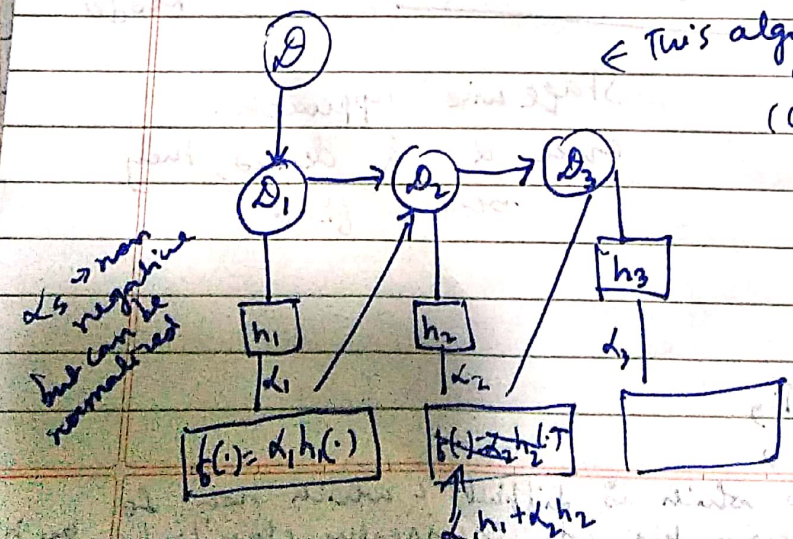
We should have a classifier, that minimizes the weighted misclassification error.

- Weak Learner: Performance on any training set is slightly better than random prediction
  e.g. Stump.

By → Boosting

- Designing a strong learner from many weak learners.
- Weak learners relatively easy to design than strong learners.
- How to design weak learners?
  → Focus on "difficult" examples.
  (Difficulty defined by models learnt so far)
  → Assign a wt to every example based on difficulty level
  → At every step
    - Design a weak learner that minimizes weighted error.
    - Assign a weight (α) to a learner based on weighted error.
    - Increase the weights of "difficult" examples.

← This algo converges, as no of models go to infinity (can be shown)



All hj's are weak model.

Schapire & Freund
↳ Introduction to Boosting (paper) read

- decrease the wts of "easy" examples

: boosting: construct a strong learner using weighted combination of weak learners

: Final model: weighted prediction of weak models

As no of examples ↑, training set error asymptotically goes to 0.

Aim: Design a classifier using a linear combination of weak classifiers, $h_j(\cdot)$

$$f(x) = \sum_{j=1}^{M} \alpha_j h_j(x) \quad \text{where } h_j(x) \in \{-1, +1\}.$$

? How do we determine α's & h's given D & M? (M: no of models to be used)

$$h_{[j-1]}(x) = \sum_{k=1}^{j-1} \alpha_k h_k(x)$$

Given we have already designed $j-1$ classifiers, how to find α_j & h_j.

$x_1, \ldots, x_n$
$w_1^j \quad w_n^j$
$w_1^2 \quad \ldots \quad w_n^2$

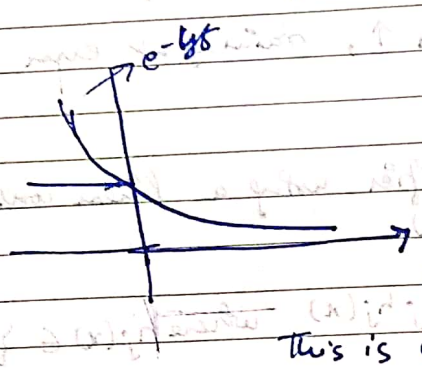
superscript - iteration no
wts determined using combined model.

$$h_{[j]}(x) = \underbrace{h_{[j-1]}(x)}_{\text{determined \& fixed after } (j-1) \text{ iterations}} + \alpha_j h_j(x)$$

Now how to get D_j, h_j & α_j.

h_2 is designed using D_2.
But $(\alpha_1 h_1 + \alpha_2 h_2)$ used to test on D_2.

Determine α_j & $h_j(\cdot)$ by optimizing misclassification loss or its surrogate fn.



An example is correctly classified if $yf > 0$.

0-1 loss: loss = 1 misclassified, loss = 0 for classification.

This is discrete loss. Not continuous so difficult to optimize.

So surrogate loss fn used for a loss.

we can use e^{-yf}

For example, using exponential loss fn,

$$l(y, h_{[j]}(x)) = e^{-y h_{[j]}(x)}$$

$$(\alpha_j, h_j) = \underset{\alpha, h}{\operatorname{argmin}} \sum_{i=1}^{n} e^{-y_i (h_{[j-1]}(x_i) + \alpha h(x_i))}$$

$$= \underset{\alpha, h}{\operatorname{argmin}} \sum_{i=1}^{n} e^{-y_i h_{[j-1]}(x_i)} \cdot e^{-y_i \alpha h(x_i)}$$

w_i^{j-1} ↑ positive quantity

where w_i^{j-1} is the wt assigned with i^{th} example after $(j-1)$ iterations.

keep one part fixed & find do optimization wrt the other part.

Good idea to optimize α & $h(\cdot)$ independently while keeping the other fixed.

Let us first fix $\alpha > 0$ & solve the following problem:

$$h_j^* = \arg\min_h \sum_{i=1}^{n} w_i^{j-1} e^{-y_i \alpha h(x_i)}$$

$$= \arg\min_h \sum_{i: y_i = h(x_i)} w_i^{j-1} e^{-\alpha} + \sum_{i: y_i \neq h(x_i)} w_i^{j-1} e^{\alpha}$$

$$= \arg\min_h \underbrace{\sum_{i=1}^{n} w_i^{j-1} e^{-\alpha}}_{\text{independent of } h} + \sum_{i: y_i \neq h(x_i)} (e^{\alpha} - e^{-\alpha}) w_i^{j-1}$$

$$= \arg\min_h \sum_{i=1}^{n} \underbrace{(e^{\alpha} - e^{-\alpha})}_{\text{positive \& independent of } h} \underbrace{w_i^{j-1} I(y_i \neq h(x_i))}$$

$$= \arg\min_h \sum_{i=1}^{n} \underbrace{w_i^{j-1} I(h_i \neq h(x_i))}_{\text{weighted misclassification error}}$$

design a classifier (h) that minimizes wted misclassification error.

- find $h_j(\cdot)$ which minimizes wted misclassification error.

Design decision stump such that to reduce weighted misclassification error.

Since every classifier is a weak classifier, they all are likely to make non zero error

- Weighted misclassification error of $h_j(\cdot)$:

$$\epsilon_j = \sum_{i=1}^{n} w_i^{j-1} I(y_i \neq h_j(x_i))$$

↑ error incurred by h_j on D_j
Calculated once we design the classifier.

Now, keep $h_j(\cdot)$ fixed & determine α_j.

$$\alpha_j = \arg\min_{\alpha} \sum_{i=1}^{n} w_i^{j-1} e^{-\alpha} + \sum_{i: y_i \neq h_j(x_i)} (e^{\alpha} - e^{-\alpha}) w_i^{j-1}$$

w_i^{j-1}: weights are normalized. Sum = 1.
misclassified examples.

$$= \arg\min_{\alpha} e^{-\alpha} + (e^{\alpha} - e^{-\alpha})\epsilon_j \quad \left(\text{letting } \sum_i w_i^{j-1} = 1\right)$$

$$= \frac{1}{2} \ln \frac{1-\epsilon_j}{\epsilon_j}$$ → diff with respect to

α_j decided based on weighted error incurred by h_j classifier

if $\epsilon \uparrow$, $\alpha \downarrow$

What if $\epsilon_j > \frac{1}{2}$? → h_j is worse than random classifier.

So we want $0 < \epsilon_j < \frac{1}{2}$

So, $\alpha_j > 0$

↳ just has to be a true quantity.

Sum of α_j's need not be 1.

because α's are decided by ϵ's

Input:
Initialize:

(1)

w_i^j : j^{th} iteration

$w_i^j = e^{-h} e^{-y_i h_j(x_i)}$

weights of i^{th} example in j^{th} iteration

$= w_i^{j-1} e^{-y_i \alpha_j h_j(x_i)}$

of $y_i = h_j(x_i)$

[∵ if = both +1 or both -1]

then it will be $w_i^{j-1} e^{-\alpha_j}$

↑ weights pulled down of misclassified example

$$= \begin{cases} w_i^{j-1} e^{-\alpha_j} & \text{if } y_i \neq h_j(x_i) \quad (\downarrow \text{wts if correct predicting}) \\ w_i^{j-1} e^{\alpha_j} & \text{if } y_i \neq h_j(x_i) \quad (\uparrow \text{wts if misclassified}) \end{cases}$$

Adaboost algorithm for classification

Input: Data $D = \{(x_i, y_i)\}_{i=1}^n$, M: Max no of iterations

Initialize: $w_i^0 = \frac{1}{n} \; \forall i$

(1) for $j = 1, \ldots, M$

(a) $h_j = \arg\min_h \sum_{i=1}^n w_i^{j-1} I(y_i \neq h(x_i))$

(b) $\epsilon_j = \sum_{i=1}^n w_i^{j-1} I(y_i \neq h_j(x_i))$

(c) $\alpha_j = \frac{1}{2} \ln \frac{1-\epsilon_j}{\epsilon_j}$ (assuming $0 < \epsilon_j < \frac{1}{2}$)

(d) Update weights of examples:

$$w_i^j = \begin{cases} w_i^{j-1} e^{-\alpha_j} & \text{if } y_i = h_j(x_i) \; \forall i \\ w_i^{j-1} e^{\alpha_j} & \text{if } y_i \neq h_j(x_i) \; \forall i \end{cases}$$

Paper: Short Introduction to boosting
Shapire &

Classifier should minimize the weighted misclassification error.

In case of bagging $h_j(x) \rightarrow$ decision tree
& $\alpha_t = \frac{1}{n}$

Pa
- Will this procedure converge?
  - A correct example can be misclassified after few iterations
  - does the error get minimized?

$$\text{Average error} = \frac{1}{n} \sum I(y_i \neq f(x_i)) \leq \sum e^{-y f(x)}$$

(mis)

exp error upper bound of average error.

zero error

$y \cdot f(x) \leq 0$

$y, f(x)$ has same sign

Algo also used to solve regression problems.

Adaboost

Learning weak classifier does not take much time, but even with 10 weak classifier we get reasonably good result.

For regression problem:

- Learn $h_1(x)$ using $\{(x_i, y_i)\}_{i=1}^{n}$

$$y_i = h_1(x_i) + \text{error}_{1i} \quad \forall i$$

- Learn $h_2(x)$ using $\{(x_i, \text{err}_{1i})\}_{i=1}^{n}$

$$\text{err}_{1i} = h_2(x_i) + \text{err}_{2i} \quad \forall i$$

We 1st find the upper bound of error & then show that upper bound converges to 0 as $t \to \infty$ (no. of iterations)

- Learn $h_3(x)$ using $\{(x_i, err_{2i})\}_{i=1}^{n}$

$$err_{2i} = h_3(x_i) + err_{3i} \; \forall i$$

Thus, $y_i = h_1(x_i) + h_2(x_i) + h_3(x_i) + err_{3i} \; \forall i$

- Learning wts ~~using~~ along with fns will result in

$$y_i = \alpha_1 h_1(x_i) + \alpha_2 h_2(x_i) + \alpha_3 h_3(x_i) + err_{4i} \; \forall i$$

We will try to find out $\alpha_1, \alpha_2, \ldots \alpha_3$ so that ~~it~~ it is as close as possible to y_i.

Proof of convergence of AdaBoost

- Initially, $w_i^0 = \frac{1}{n} \; \forall i$
- Normalized wts after T rounds:

$$w_i^T = \frac{w_i^{T-1} e^{-y_i \alpha_T h_T(x_i)}}{Z_T}$$

where $Z_T = \sum_{j=1}^{n} w_j^{T-1} e^{-y_j \alpha_T h_T(x_j)}$

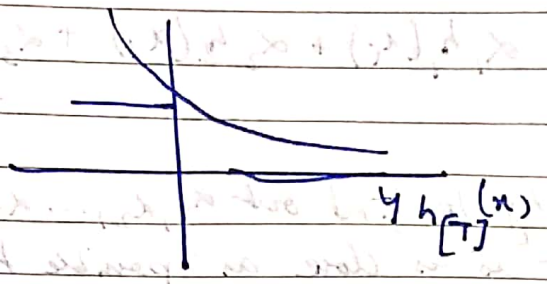
so that $\sum_{i=1}^{n} w_i^T = 1$. Equivalently,

$$w_i^T = \frac{1}{n} \cdot \frac{e^{-y_i h_{[T]}(x_i)}}{\prod_{t=1}^{T} Z_t} \leftarrow \text{because of normalization}$$

$$\frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \text{sign}\left(h_{[T]}(x_i)\right)\right)$$

↑ average 0-1 error incurred by after T iterations

$$\frac{1}{n}\sum_i e^{-y_i h_{[T]}(x_i)}$$



- Training error due to $h_{[T]}(x)$

$$Err = \frac{1}{n}\sum_i I\left(y_i \neq \text{sign}\left(h_{[T]}(x_i)\right)\right)$$

$$\leq \frac{1}{n}\sum_i e^{-y_i h_{[T]}(x_i)}$$

$$= \sum_i w_i^{T} \prod_{t=1}^{T} Z_t$$

($w_i^{T} = 1$)

$$= \prod_{t=1}^{T} Z_t$$

Now, $Z_t = \sum_i w_i^{(t-1)} e^{-y_i \alpha_t h_t(x_i)}$

$$= \sum_{i: y_i = h_t(x_i)} w_i^{(t-1)} e^{-\alpha_t} + \sum_{i: y_i \neq h_t(x_i)} w_i^{(t-1)} e^{\alpha_t}$$

$= (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$

($1-\epsilon_t$: correct classification rate)

$\therefore Err \leq \prod_{t=1}^{T} Z_t$

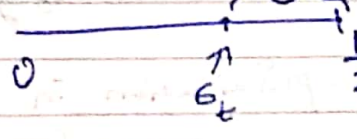
$= \prod_{t=1}^{T} (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$

$= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \quad (\because \alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t})$

ϵ_t → different for every iteration, so we remove it

$= \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \quad (\text{letting } \epsilon_t = \frac{1}{2} - \gamma_t)$

γ_t → how far it is close to random



$0 < \epsilon_t < \frac{1}{2}$

Let $\gamma = \min(\gamma_1, \gamma_2, \ldots \gamma_T)$. Note $\gamma > 0$

↑ independent of T

γ always >0 [$\because \epsilon_t$ always $< \frac{1}{2}$]

$\therefore$ Clearly, $\gamma^2 < \frac{1}{4}$ $(\because \gamma_t = \frac{1}{2} - \epsilon_t)$

$\therefore Err \leq (1-4\gamma^2)^{T/2}$

Our algo ensures that $0 < \epsilon_t < \frac{1}{2}$

$= e^{\ln(1-4\gamma^2)^{T/2}}$

$= e^{\frac{T}{2}\ln(1-4\gamma^2)}$ $\quad \ln(1+x) \leq x$ if $x > -1$

$= e^{\frac{T}{2}(-4\gamma^2)}$

$= e^{-2T\gamma^2}$ ← error asymptotically goes towards zero in adaboost.

Thus $E_m \to 0$ as $T \to \infty$.

We stop adding models when validation error starts to deteriorating.

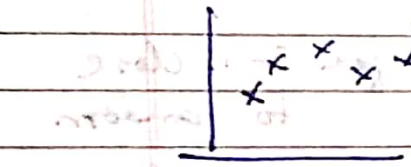
How to design a classifier that classifies weighted misclassification error?

~~Consider the~~

$$\min_F \bar{E} \triangleq \sum_i l(y_i, F(x_i))$$

F^0

$$F^1 = F^0 - \eta_0 \frac{\partial \bar{E}}{\partial F}\Big|_{F=F^0}$$



optimization in non parametric space (or fn space)

Consider the problem of loss for optimization in fn space.

$$\min_F l(F) = \sum_{i=1}^{n} l(y_i, F(x_i))$$

where $F = (F(x_1), F(x_2) \ldots F(x_n))^T$

- gradient descent in fn space (assuming sufficient smoothness of $l(\cdot)$)
  - → Initialize with F_1
  - → $F_2 = F_1 - \eta_1 \frac{\partial l}{\partial F}\Big|_{F=F_1}$
  - → $F_3 = F_2 - \eta_2 \frac{\partial l}{\partial F}\Big|_{F=2}$

$$\Rightarrow F_t = F_{t-1} - \eta_{t-1} \frac{\partial l}{\partial F}\Big|_{F=f_{t-1}}$$

- Given $\frac{\partial l}{\partial F}\Big|_{F=f_{t-1}}$, it is easy to determine η_{t-1}.

$$\eta_{t-1} = \underset{\eta \geq 0}{\operatorname{argmin}} \sum_{i=1}^{n} l\left(y_i, F_{t-1}(x_i) - \eta \frac{\partial l}{\partial F(x_i)}\Big|_{F=f_{t-1}}\right)$$

which can be used to update F_{t-1} as

$$F_t = f_{t-1} - \eta_{t-1} \frac{\partial l}{\partial F}\Big|_{F=F_{t-1}}$$

or $F_t = \sum_{j=1}^{t-1} \eta_j \cdot \left(-\frac{\partial l}{\partial F}\Big|_{F=f_j}\right)$ (letting $F_1(x_i) = 0 \ \forall i$)

- Compare with the Adaboost model after $(t-1)$ iterations

both similar

$$h_{[t-1]}(x) = \sum_{j=1}^{t-1} \alpha_j h_j(x)$$

~~α_j similar to η_j~~

Can we align our classifier ~~who~~ to -ve gradient direction of l wrt F.

- $h_j(\cdot)$'s in boosting are analogous to negative gradients in function space.

Paper: Greed for Approximation: A gradient Boosting Machine. J. H. Friedman

↑ for regression, but can be extended to classification problems.

↕

- Gradient Boosting Machine originally designed for regression problems.

- At t-th Iteration, Induce a model whose predictions are aligned with the -ve gradient direction of the loss fn (design a model based on pseudo response)

$$h_t = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} l\left(\underbrace{\frac{-\partial l}{\partial F(x_i)}\Big|_{F=F_{t-1}}}_{\text{pseudo response}}, h(x_i)\right)$$

Our t'th model will be designed such that for each x_i it outputs pseudo response.

and determine η_{t-1} using

$$\eta_{t-1} = \underset{\eta}{\operatorname{arg\,min}} \sum_{i=1}^{n} l\left(y_i, F_{t-1}(x_i) + \eta h_t(x_i)\right)$$

$\eta \leftarrow \eta > 0$ always

Helps to design model at tth iteration in better way.

$$g = \begin{pmatrix} g_1 \\ \vdots \\ g_n \end{pmatrix} \qquad g_i = -\frac{\partial l}{\partial F(x_i)}\Big|_{F=F_{t-1}}$$

same ↑

$$\min \sum_{i=1}^{n} \left(-g(x_i) - h(x_i)\right)^2$$

↓ try to make h = neg gradient

$$F_t(x) = F_{t-1}(x) + \eta_{t-1} h_t(x)$$

No class on 27/02
Test #2 06/03 Friday 9:30 - 11:00
9:30 - 10:30 Am
Assignment 2 : due on 06/03 11:59 pm
extra class : 08/03

Page No.
Date / /20

Review :

$$f(x_i) = w^T x = \sum_{j=1}^{d} w_j x(j) \qquad \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \Omega(w)$$

data dependent term

$f(x_i; w)$ can be overfit because of data dependency

Now

$$\hat{y}_i = \sum_{j=1}^{M} \alpha_j h_j(x_i) + \sum_{j=1}^{M} \Omega(h_j)$$

Model is in form of h_j & we want to control its complexity. Term associated with every classifier we have designed

$$\hat{y}_i^1 = 0 \qquad \hat{y}_i^2 = \hat{y}_i^1 + h_1(x_i)$$

$$\hat{y}_i^3 = \hat{y}_i^2 + h_2(x_i)$$

suppose we have added $t-1$ models & we want to add t^{th} model :

$$\sum_{j=1}^{t-1} \alpha_j h_j(x_i) + \sum_{j=1}^{t-1} \Omega(h_j)$$

$$\sum_{i=1}^{r} l(y_i, \hat{y}_i^{t-1}) + \lambda \sum_{j=1} \Omega h(j)$$

Now h's itself are parameters.

$$h_t = \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{t-1} + h(x_i)\right) + \lambda \sum_{j=1}^{t-1} \Omega h(j) + \lambda \Omega(h)$$

① ② ③ (② is constant)

Adding new model such that this obj fn is minimized we use stagewise strategy. (All other models fixed ⇐ minimizing ① & ③.

→ may not have a form like least squares loss.

So we get quadratic approximation of this.

Taylor's series approximation.

$$\Psi(x+h) \approx \Psi(x) + h\Psi'(x) + \frac{h^2}{2!}\Psi''(x)$$

Why quadratic?

→ Closed form soln of quadratic possible for quadratic (may not be possible with other fns).

F's here are $\hat{y}_i$.

We are dropping η_t term for simplicity.

- Additive training (models added in stagewise manner keeping previous constant)

Stagewise (keeping all previous models fixed)

η_t can be easily determined by simple optimization problem.

$\hat{y}_i^1 = 0$ → Initially no knowledge of tree fn.

$\hat{y}_i^2 = F_2(x_i) = \hat{y}_i^1 + \eta_1 h_2(x_i)$

$\vdots$

$\hat{y}_i^t = F_t(x_i) = \hat{y}_i^{(t-1)} + \eta_{t+1} h_t(x_i)$.

where $\hat{y}_i^t$: Output of model on i^{th} example at t^{th} round.

Page No.
Date / 120

- How do we decide which $h_t(\cdot)$ to add?
  could be a decision tree (may not be decision tree always)

- If we use least squares error fn, $l(y,F) = (y-F)^2/2$, then one iteration of the algo is :-
  (neg of grad of this fn)

  → Pseudo response : $\tilde{y}_i = y_i - F_{t-1}(x_i) = y_i - \hat{y}_i^{(t-1)}$ (residual error from previous round)

  → $h_t = \text{argmin}_h \sum_{i=1}^{n} (\tilde{y}_i - h(x_i))^2$

  → $\eta_{t-1} = \underset{\eta>0}{\text{argmin}} \sum_{i=1}^{n} (\tilde{y}_i - \eta h_t(x_i))^2$

  → $F_t(x) = \underbrace{F_{t-1}(x)}_{\text{fixed}} + \eta_{t-1} h_t(x)$ (Friedman's paper Gradient boost).

We use gradient to decide what next model will be. New model should be aligned with neg gradient of obj fn. ~~Here it is neg gradient~~

Gradientboost Algo :-

Input : data $D = \{(x_i, y_i)\}_{i=1}^{n}$, M : max no of iteration.
Initialize : $F_0 = \text{argmin}_\rho \sum_{i=1}^{n} l(y_i, \rho)$

(1) for $t = 1, \ldots, M$ ← On Friedman's paper.

(a) $\hat{y}_i = -\left[\frac{\partial l}{\partial F(x_i)}\right]_{F=F_{t-1}}$ $\quad \forall\ i = 1, \ldots, n$

(b) $h_t = \text{argmin}_h \sum_{i=1}^{n} (\hat{y}_i - h(x_i))^2$

(c) $\eta_{t-1} = \text{argmin}_{\eta>0} \sum_{i=1}^{n} l$

Regression tree: partitions input space with axis parallel rectangle

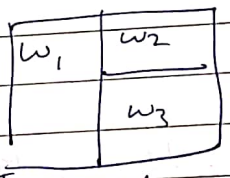
Regression Tree: → will give no output as a model

- Assume each base learner is a regression tree having J leaf nodes
- Partitions input space $\mathcal{X}$ into J disjoint regions.

$$h(x; \{w_j, R_j\}_{j=1}^{J}) = \sum_{j=1}^{J} w_j \, I(x \in R_j) = w_{r(x)}$$

where w_j is the leaf score value associated with region R_j and $r : \mathcal{X} \to \{1, 2, \dots J\}$

Every leaf node corresponds to given constant output.



→ 3 leaf nodes.

deep regression tree can be thought of a fn.

Every pt lies in one of the region

$$f(x) = \sum_{j=1}^{J} w_j \, I(x \in R_j)$$

$$= w_{r(x)}$$

r : a fn

$$r : \mathcal{X} \to \{1, 2, \dots J\}$$

maps a fn to one of leaf nodes.

- Parameters a
  - → No of leaf
  - → Quantities R_j's.
  - → Tree depth
  - → norm of

So far for the

- Assume
  - to den
  - $w_j t$
- R_{jt} (iter
- w_j
  - one
  - in

Page No.
Date / /20

- Parameters associated with regression tree

→ No of leaf nodes (J) in the tree
→ Quantities that define the boundaries of the regions R_j's.
→ Tree depth
→ norm of the leaf ~~score~~ Scores $\|w\|^2$

So far we have ignored regularization.
For the time being we assume we will use

- Assume we have designed $(t-1)$ regression trees & want to design a new tree at t-th iteration.

w_{jt} → weight associated at j^{th} region at t^{th} instant.

$$F_t(x) = F_{t-1}(x) + \eta h_t(x; \{w_{j,t}, R_{jt}\}_{j=1}^{J})$$

- R_{jt}: Regions at terminal nodes of the tree at t-th iteration constructed to predict the pseudo-response (Line 1(a) of GB algo)

- $w_{jt} = \text{avg}_{x_i \in R_{jt}} \hat{y}_i$

once we identify the R_{jt}, we take all examples in that region & take their average to find w_{jt}.

- Obj fn minimized after t-th round:

$$E^{(t)} = \underbrace{\sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)})}_{\text{training loss}} + \Omega(h_t)$$ Structural loss

But, $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + h_t(x_i)$ - (seperately optimize h)

So, the obj fn at t-th iteration is,

$$\min_{h(\cdot)} \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + h(x_i)) + \Omega(h)$$

↑ Optimization prob at tth iteration (h dropped, we will bring it later)

~~loss optimized at end of $(t-1)^{th}$ iteration~~

Consider $E^{(t)} = \sum_{i=1}^{n} l(y_i, F_{t-1}(x_i) + h(x_i)) + \Omega(h)$.

↑ loss fn to be optimized at t^{th} iteration

- Second order Taylor series approximation:

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2} f''(x).$$

- Define $p_i = \left.\frac{\partial l}{\partial F}\right|_{F=F_{t-1}(x_i)}$, $q_i = \left.\frac{\partial^2 l}{\partial F^2}\right|_{F=F_{t-1}(x_i)}$

Instead of F we can write $\hat{y}$ here

- Therefore,

$$E^{(t)} \approx \sum_{i=1}^{n} \left(\underbrace{l(y_i, F_{t-1}(x_i))}_{\text{constant}} + h(x_i) p_i + \frac{h(x_i)^2}{2} q_i \right) + \Omega(h)$$

$l(y_i, F_{t-1}(x_i) + h(x_i))$ → step in Taylor's expansion

$\approx l(y_i, F_{t-1}(x_i)) + h(x_i) \frac{\partial l}{\partial F}\Big|_{F=F_{t-1}}$

- ~~Using this approx, the~~ $+ \frac{h^2(x_i)}{2} \frac{\partial^2 l}{\partial F}\Big|_{F=F_{t-1}}$
- Using this approx, the opt prob is :

$$h_t = \arg\min_h \sum_{i=1}^{n} \left(p_i h(x_i) + \frac{1}{2} q_i h(x_i)^2 \right) + \Omega(h)$$

we are not worrying abt this now.

- Can define complexity of a regression tree as (different ways to control complexity of tree).

$$\Omega(h) = \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} w_j^2$$

$\gamma, \lambda \rightarrow$ controls contribution of each term

- Write the approx obj fn in terms of w & J
- Recall, $h(x) = \sum_{j=1}^{J} w_j I(x \in R_j) = w_{R(x)}$
- Obj fn :

$$E^{(t)} \approx \sum_{i=1}^{n} p_i h(x_i) + \frac{1}{2} q_i h(x_i)^2 + \Omega(h)$$

$$= \sum_{i=1}^{n} p_i w_{R(x_i)} + \frac{1}{2} q_i w_{R(x_i)}^2 + \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} w_j^2$$

- Define $I_j = \{ i : R(x_i) = j \}$

$$\therefore E^{(t)} \approx \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} p_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} q_i + \lambda \right) w_j^2 \right] + \gamma J$$

sum of J independent quadratic fns (can be optimized independently)

- Obj fn:

$$\tilde{E}^{(t)} \approx \sum_{j=1}^{J} \left[\left(\sum_{i \in I_j} p_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} q_i + \lambda \right) w_j^2 \right] + \gamma J$$

- Define $G_j = \sum_{i \in I_j} p_i$ & $H_j = \sum_{i \in I_j} q_i$

$$\therefore \tilde{E}^{(t)} \approx \sum_{j=1}^{J} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma J$$

- Given a tree with a fixed structure $\pi(x)$ (and also fixed J), the optimal parameters (for leafs) & optimal obj fn are

$$w_j^*(\pi) = \frac{-G_j}{H_j + \lambda}, \quad \tilde{E}^{(t)*}(\pi) = -\frac{1}{2} \sum_{j=1}^{J} \frac{G_j^2}{H_j + \lambda} + \gamma J$$

$T_1, T_2 \ldots T_m$ ← m trees from which we have to choose (their structure fixed)

For each tree find the optimal obj fn value associated with the tree.

$\tilde{E}^{t*}(1) \quad \tilde{E}^{t*}(2) \quad \ldots \quad \tilde{E}^{t*}(m)$

If we fix a tree, we have a fixed structure $\pi(x)$ & fixed obj fn value.

Choose the one which has min obj fn value.

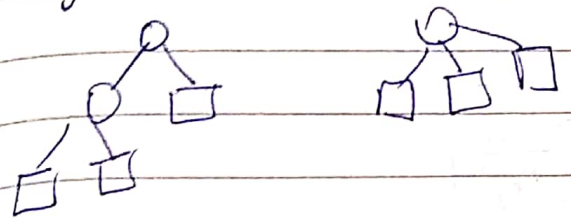
There could be infinitely many tree structures possible for a given data set.

So heuristic used (Greedy approach) to choose from infinitely many tree structures possible.

- $\tilde{E}^{(t)*}\pi$ measures "goodness" of a tree structure (smaller the value, better the structure).

Page No.
Date / /20

Say tree with 3 leaf nodes : Structures possible.



- How to select the best regression tree, given a set T of tree structures at t-th iteration?
- for every tree in $T \in T$
  → Compute G_j, H_j, w_j^* $\forall j$ and $E_T^{(t)*}$
- Best regression tree :

$$T^* = \underset{T \in T}{\operatorname{argmin}} E_{T_t}^{(t)*} \quad \underset{T \in T}{\operatorname{argmin}} E_T^{(t)*}$$

and use the corresponding leaf score values w_j^*.

- Unfortunately, getting T is hard! (similar to feature selection (maximize gain))
- Solution : Grow the tree greedily. (In the paper Tianqi Chen & Carlos Guestrin) xgboost.

O O	O O O
g_1, h_1 g_4, h_4	g_2, h_2, g_5, h_5, g_3, h_3
$G_L = g_1 + g_4$	$G_R = g_2 + g_3 + g_5$