Topic:

"Story about decision trees based on lectures from one professor" (Part II/II)

Konstantin Burlachenko

-kburlachenko@nvidia.com

-bruziuz@stanford.edu

-burlachenkok@gmail.com

References

[1] First part of presentation
https://github.com/burlachenkok/presentations_bruziuz/tree/master/decision_trees/mpti

[2] Classification And Regression Trees, 1983 (Brieman, Friedman, Olshey, Stone) – CART
https://www.amazon.com/Classification-Regression-Wadsworth-Statistics-Probability/dp/0412048418

[3] Book: The Elements of Statistical Learning
(Friedman, Tibshirani, Hastie)
https://web.stanford.edu/~hastie/Papers/ESLII.pdf

[4] News: Bradley Efron has won the 80K USD Prize in Statistics for his 1970s work
https://www.nature.com/articles/d41586-018-07395-w?fbclid=IwAR3EbV3LbBQmls4KarxRzblvonn6HUxiC4IqRGaaXPY2XRsbOXtSPHq03pk

[5] [cvxbook] Convex Optimization (S.Boyd, L.Vandenberghe)
http://stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[6] My notes: What is cross-validation and some hints about it
https://sites.google.com/site/burlachenkok/articles/what-is-cross-validation-and-some-hints-about-it

[7] Example of Random Forest usage.
Real-Time Human Pose Recognition in Parts from a Single Depth Image, 2011
https://www.microsoft.com/en-us/research/publication/real-time-human-pose-recognition-in-parts-from-a-single-depth-image/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F145347%2Fbodypartrecognition.pdf

[8] Some way to interpretate black-box models
https://sites.google.com/site/burlachenkok/some-ways-to-intepretate-black-box-models

Plan

1. Reminder about Decision tree for Regression
2. Extremely short note about decision tree for classification
3. Problem with regression trees and what todo
4. Ensemble of trees. Why such thing matter.
5. Bagging
6. Random Forest
7. Bagging and Random Forest. Advantages
8. Slide about cross-validation (or x-validation)
9. Boosting
    1. Motivation
    2. Model structure and score for constructing model
    3. Arised huge linear regression problem and constraining it
    4. Massage constrained optimization problem into unconstrained
    5. Conclusion that find solution should be done in one dimensional path
    6. Assumption which will allow us to think about reasoable penalties
    7. Penalities
    8. Path seeking formulation

CART Decision Tree for regression. General

$$\{c_m, R_m\} = argmin_{\{c_m, R_m\}} \sum_{i=1}^{N} \left(y_i - \sum_{m=1}^{M} c_m \left(\prod_j I(x_i \epsilon S_{mj}) \right) \right)^2$$

$$R_i = S_{i1} x S_{i2} \dots x S_{im}$$

After several relaxations

- We come to this formulation for building piecewise constant function.
- Here in this formulation and more broadly in all Machine Learning people think that they know precisely (x_i, y_i), even it is not true (as have been mentioned in [1] part).
- People in ML do not use any *robust schemas* to handle uncertanty in (x_i, y_i)

Drama

Even in such formalization we can not solve it exactly and efficency - It is still hard combinatorial problem. Common heuristic is ***Recursive Partitioning***.

CART Decision Tree for regression. Split.

In each iteration of refinement we find split which maximize the following objective:

$$max.improve$$

$$improve = N\left(\frac{1}{N}\left(\sum_{i=1}^{N}\left[y_i - F_{c_{before\ split}}(x_i)\right]^2\right) - \frac{1}{N}\left(\sum_{i=1}^{N}\left[y_i - F_{c_{after\ split}}(x_i)\right]^2\right)\right)$$

(Implicit constraint $improve \geq 0$*)*

It will give us maximum decrease (via greedy step) for

Evaluated as: $\hat{R}N = N\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sum_{m=1}^{M} c_m\left(\prod_j I\left(x_i \epsilon S_{mj}\right)\right)\right)^2$

And if $improve = 0$ **it will just stops algorithm due to it's limitations**

CART Decision Tree for classification (easy upgrade of model structure)

- Schema can be upgrade for inference

$$F_c(x) = \sum_{m=1}^{M} c_m \left(\prod_j \; I\left(x_j \epsilon S_{mj}\right)\right)$$

Where:

1. c_m now is not scalar from $\mathbb{R}$ but element of finite set $C = \{class_1, class_2, \dots, class_K\}$
2. x is input vector for predictor and x_j is it's components
3. $R_m = S_{m1} \times S_{m2} \times \cdots \times S_{mn}$ Region is presented as cartesian product of simple sets

CART Decision Tree for classification (various problems in search strategy)

Assume we leave score criteria as it was for regression

$$\{c_m, R_m\} = argmin_{\{c_m, R_m\}} \sum_{i=1}^{N} \left(y_i - \sum_{m=1}^{M} c_m \left(\prod_j I(x_i \epsilon S_{mj}) \right) \right)^2$$

$$R_i = S_{i1} x S_{i2} \dots x S_{im}$$

Drama:

Greedy one look-ahead does not work at all in such circumstances with improvement .

To upgrade CART(Classification And Regression Trees) model to allow make classification various steps should be made.

CART Decision Tree for classification. (Only general picture how this model handle classification)

- Construct predictor which estimate (interval-scale) probability of each class for random variable $Y|X$ instead of estimating value of $Y|X$ directly.

  This trick allow to remove this strange categorical variables and come to interval scale

- We solve K-response regression problem with affine constraint that sum of all response is equal to 1 which allow to construct approximation of such probabilities

- Mathematical manipulation should be done

- Conclusion what is important during splitting for it is

  "Probability be in region" x "Diversity of the regions"

  Where Diversity is estimated by *Gini index of diversity*

- Gini index of diversity $G(p) = 1 - p^T p$ *on probability simplex*
  - *It archives maximum when* $p_i = \frac{1}{K}$
  - *It archives minimum when* p *probablity mas function is discrete delta*

CART Decision Tree for classification. (Only general picture how it is doing)

After various reformulation we can come to

- We can solve regression problem like we do in CART for regression, but we Find split which **maximize improve**:
  $$improve = P(R_m)Gini(R_m) - P(R_l)Gini(R_l) - P(R_r)Gini(R_r)$$

- Also because $P(R_m) = P(R_l) + P(R_r)$ we in fact can more easily to compute:
  $$improve = P(R_m)H(R_m) - P(R_l)H(R_l) - P(R_r)H(R_r)$$

- Quanity $\mathrm{H(p)} = -\mathrm{p}^{\mathrm{T}}\mathrm{p}$ sometimes call **"second order entropy"**

- Author of C.4.5 used not H(p) but use usual entropy ($-p_i \log(p_i)$) instead

- In leafs regions probability of each class is just computed as
  $$p_c = \frac{\#number\ of\ (x_i,y_i)\ where\ y_i=c}{\#total\ number\ of\ (x_i,y_i) from\ region}$$

To cover it in glorry detail - a separate presentation needed

Problem with decision trees for regression

Problem is that **trees have big variance** because:

1. Very rapid data fragmentation. During split's there are less and less data in daughter regions.
2. The possible error in upper part of decision tree propagates down to the terminal nodes.
3. The errors are not accumulated (or averaged), but in fact errors is cascaded/multiplied
4. Even minor change split in the root will dramatically change tree structure

What todo if we do not leave room of nice advantages of the decision trees?

- Live with the problem
- Fix up trees. Recent research is:
  1. **Bagging ,1996**
  2. **Boosting, 1996**
  3. **MARS (Multiple Adaptive Regression Splines) 1989.** (not so popular then 1,2 right now)

Ensemble of trees. Why such thing matter

First view of ensemble of trees

- Ensemble of tree can be viewed as linear combination of tees
- Trees is piecewise constant function with special form of regions
- Linear combination of piecewise constant function is piecewise constant function

But why does it matter

1. In ensemble of trees there are more pieces then in single tree
2. Pieces can overlap
3. Any function can be approximated by piecewise constant functions if there are:
    1. A lot of regions to split domain
    2. A lot of data to fit each constant value in each region

Bagging. History and goal

- Bagging was invented by Leo Brieman in 1996. Reference in the book **[3], ch 8.7**
- Goal is improve behaviour of unstable predictive schemas like:
  - Neural Nets
  - Decision Trees

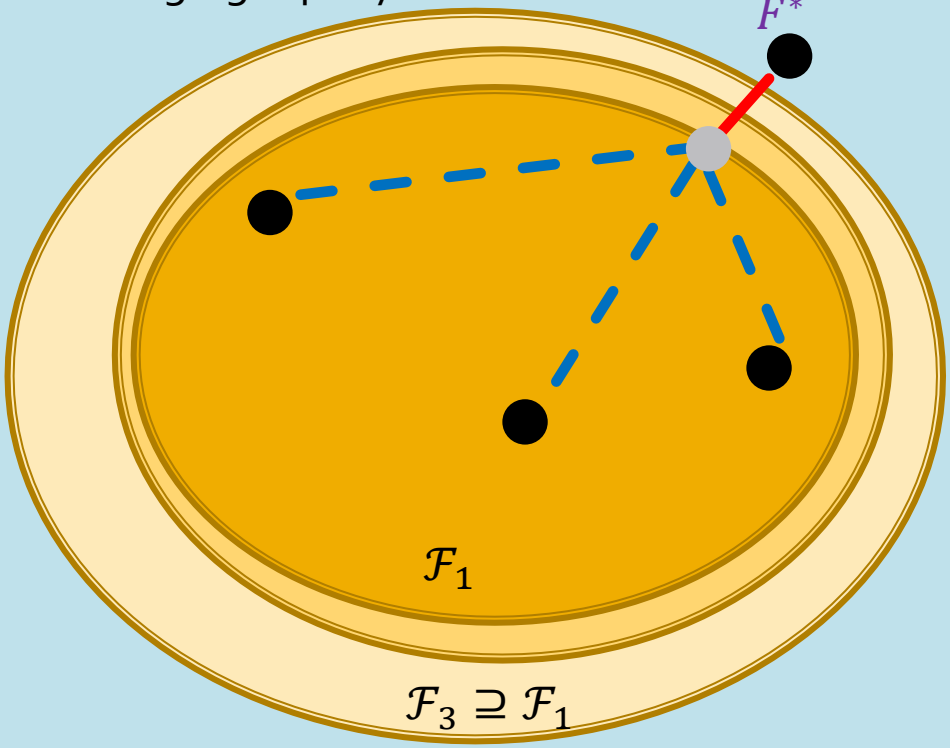
Unstable means the following.
We have optimization problem for empirical Loss minimization:

$$\hat{F} = argmin_{F(x)\in\mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^{N} L(y_i, F(x_i)) \right)$$

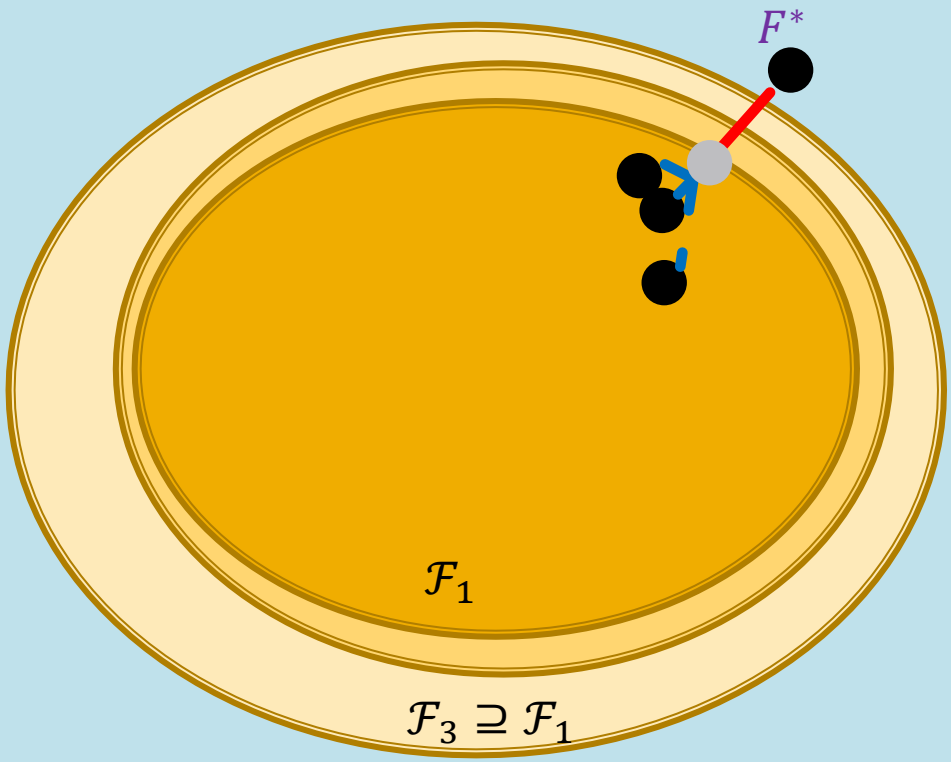
But if we will perturb "train set" (X, Y) then it will lead to dramatically change of $\hat{F}$.

Bagging. Illustration of unstable procedure behaviour when function class is big.

Distributions of solutions is
"very wild as changing rapidly" as we draw different "train set" from population



Bagging. Illustration of stable procedure behaviour when function class is big.



Bagging. Several quotes.

"The simplicity how Leo figure out how to fix it is awesome by it's simplicity" – J.H. Friedman

"Leo called it's bagging because it means bootstrap aggregation" – J.H.Friedman

Bagging algorithm Step 1/3

Step-1: Perturb original train data T in some way and get perturbed train set T_b

"In fact it doesn't matter a lot how to make change and there are many ways todo it" – J.Friedman.

One fancy technic is "bootstrap sample".

We have set T contains N observations. We sample randomly observation from this train data and each iteration we sample *observation* we back sample to train data.

Bootstrap technic was invented by *Brad Efford* for other things.

By the way **12NOV2018** he received 80K USD price International Prize in Statistics for his 1970s work [4]

Jerome H.Friedman:
"Another technic which Leo tried add random noise to Y. And it works pretty well too. In any case idea is make perturbation a bit, but not too much."

Bagging algorithm Step 2/3

Step-2:

Apply original procedure with pulling "best" function from our function family via search strategy:

$$\hat{F}_b = argmin_{F(x)\in\mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^{N} L(y_i, F(x_i)) \right), (x_i, y_i) \in T_b$$

Bagging algorithm Step 3/3

Step-3:

Repeat step-1, step-2 "*B*" times. Average behavior of "B" models and give final model as average of B models

$$\hat{F} = \frac{1}{B}\left(\sum_{b=1}^{B} \hat{F}_b\,(x)\right)$$

Bagging. Some observations

- Bagging doesn't do nothing with bias (almost always). If we consider function space which is closed at least to "sum/scaling".

⇒ So particulary for tree's "*this silly thing*" do nothing with bias error.

- Also what is averaging is **ouput** and **it's not the structure** of the trees.

- *"But why this thing can do something? How even it's possible! " – J.Friedman*

Bagging . Why this thing is working?

Deep answer lie in area of math optimization and one aspect of difference between **convex** and **non-convex** math optimization.

Bagging . About perturbed convex optimization problem

Let's talk a bit on convex optimization problem and it's perturbations

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq u_i \\ & h_i(x) = v_i \end{array}$$

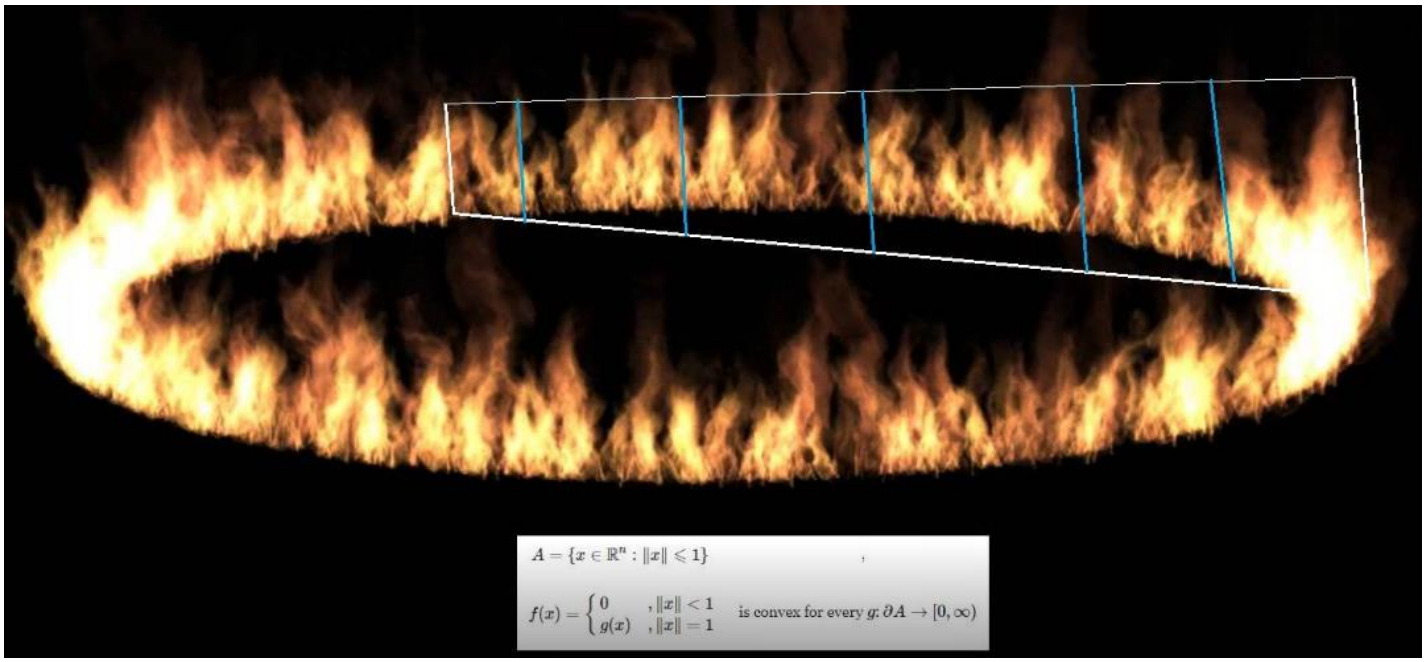
- **Optimal value $\boldsymbol{p}^*$** – is infimum of objective function attained in feasible set and intersected with the domain of optimization problem [cvxbook], p.127
- **Optimal value of the perturbed convex optimization problem** during perturbation of right hand side equality or inequality **is another convex function** of the parameters of this perturbations
- If optimal value of perturbed problem is differentiable then for the perturbed optimal value as a function of perturbations $\boldsymbol{p}^*(\boldsymbol{u}, \boldsymbol{v})$ we have: $\frac{\partial p^*(0,0)}{\partial u_i} = -\lambda_i^*$ and $\frac{\partial p^*(0,0)}{\partial v_i} = -v_i^*$

  => So in this case if perturbation are not big too much then optimal value roughly speaking is not changing too much

[cvxbook], 5.6. Perturbation Analysis, p.250,p.234-236

Bagging . About convex functions.

Convex function are continuous in relative interior of it's domain. So mostly they are continuous even some strange situations can happend on boundary:



https://www.facebook.com/photo.php?fbid=664179037304832&set=a.164584463930961&type=3&theater

Bagging. Will bagging work for problem which are reduced to convex optimization?

- **No**

"Bagging does not help for procedures which are themselves are convex optimization problem" – Jerome H. Friedman.

"Whole point of ensemble you combine different decision trees… ensemble methods in context of convex optimization are boring…" – S.Boyd, 58:40, https://www.youtube.com/watch?v=wqy-og_7SLs

Bagging. Ideas why bagging helps for approximation schemas reduced to non-convex optimization

1. Let's start. We have multiple local minimums in our objective.

2. If use local iterative strategy then obtained optimal point, depends on our start.
   (By the way one more fun moment with S.Boyd https://youtu.be/wqy-og_7SLs?t=2953, 49:12
   "*If you have non-convex problems and we will use solvers that you mentioned. Stephen: It depends, but the most accurate technical statements is the following.* ***Something happens***")
3. So change in starting point will lead to different solution.
4. We don't change starting point in this strategy. What we're doing – we perturb/shuffle a bit/change dataset. Which is the same as change start point if think about it.

We hope:

1. That there is global picture of convex function with several ripples.
2. Near global minimum there are several other local minimums, but they are clustered near each other
3. Sampling technic will more probably give us predictors with minimums which are near global minimums
4. Another thing that for convex function we have $f(E[w]) \leq E[f(w)]$.
   So for convex function averaging in model obtained by averaging parameters is better then average of the models

Empirically:

This technic from 1996 upgrade Decision Trees into extremely powerfull model.

Bagging. Nice decoupling.

- Via deep of decision trees – we control bias.
- Via bagging technic – we control variance .
- If to be completely honest we can bag anything
- **When bagging works**
  - Trees are not highly correlated
  - Each tree should be enough strong to predict something
- But Bagging as "average technic" works with improve quality in average, no for particular tree.
- *"It still can be a case then a single tree works better"* – J.Friedman

Random Forest as extreme case of Bagging

- Random Forest is most popular form of bagging
  - We create deepest Decision Trees without any pruning (cp=0)
  - The height of decision tree is only limited by the size of train samples
  - Wrap-around this procedure via bagging
- Nice thing about Random Forest – each tree can be compute completely in parallel.
- Each Decision Tree is building completely independent
- Example of usage. **Kinect** in human pose recognition use Random Forest [7]

26

Random Forest. Conclusion.

- In Random forest it's possible append extra variation of the trees via build tree based on random subset of variables.
- J.Friedman said that Leo didn't give any recommendation of the number of variables, but via consideration of J.Friedman selecting of variable **is just a strange thing**.
- For random forest it should be 2 meta-parameters:
  Size of the sample for bootstap.
  Smaller boostrap sample – more randomness in the tree.
  Bigger boostrap sample – less randomness in the tree.
  Fraction of selected variables.
- Unfortunately in standard implementations there is no necessary such thing have been implemented both.

Bagging and Random Forest. Advantages

- They share all nice properties of Decision Trees from [1] **exclude 1** interpretability
- Even If you have 1000 or 10 or **even 5** trees it's very hard to understand what is going on under the hood (or at least not so easy)
- But main concurent is *Neural Nets* – is not interpretable too (or at least it's not so easy)

If you're interesting in interpretability here a short note about several schemans to interpretate black-box models in AI/ML [8]

Model selection: slide about cross-validation

- Several times this word come into play I need to say at least something.
- Let's me describe the simplest way todo it.
  1. It's schema when you have opt.problem with variables **(a, λ)** and you split you data which you use to fit/extrapolate into two **sets** – train and **test**
  2. You solve opt.problem on a based on **train** data for some fixed λ
  3. You check how problem behaves on data called **test**
  4. Do if for several λ

ML community do not state that they want explicitly so decision which one λ to choose is up software engineer.
(usually smaller is better).

In terms of math optimization some variation os such strategy can be formalized as we have two sets Set_1 and Set_2 with data (we can call it **train** and **test,** but It's only a name) and we solve the following opt.problem:

$$min. \sup_{S \in \{Set_1, Set_2\}} \left(R_S\left(F(a,\lambda)\right)\right)$$

variables: a, λ (optimization community or when optimization based model arrised in ML/AI)
parameters: a, λ (machine learning community)

In such formulation it's identical to **robust worst case optimization**.
If λ is something wild enough then from algorithm above step $\#4$ can be repeated several times

Also here my note "*What is cross-validation and some hints about it*":
https://sites.google.com/site/burlachenkok/articles/what-is-cross-validation-and-some-hints-about-it

Boosting. References and motivation

"Motivation as I see it's right now is completely different how it was historically developed. It's not in our book [3], but I think it's just better way to understand from Optimization perspective" - J.H.Friedman, May2018.

- Reference [3], ch.10
- **Goal:** Maintain advantages of the trees while dramatically improve accuracy
- Motivation: build something like random forest
  But know similarity stops

Boosting. One more about ensembles methods.

- Tree at the end of the day is just *piecewise constant function* [1]. E.g. tree with M terminal nodes has specific form:

  $F(z) = T(z) = \sum_{m=1}^{M} c_m I(z \epsilon R_m)$, where R_i are disjoint are cover all function domain for $X \rightarrow Y$

- Any finite form of linear combination of piecewise constant function is just piecewise constant function

- **But In Ensemles methods (including boosting)**
  - There are a lot of pieces
  - Pieces of trees are overlap

Boosting: what is model structure ?

- Function class that we're going to consider is linear combination of all possible trees:

$$\mathcal{F} = \{g: g = \sum_{i=1}^{+\infty} a_i f_i \, , such\ that\ a_i \in R, f_i \in \ \mathcal{F}_{tree}\}$$

- More specifically we will consider some **huge,** but finite subsets from it only set of Regression Trees that can be built on any possible subset from train set

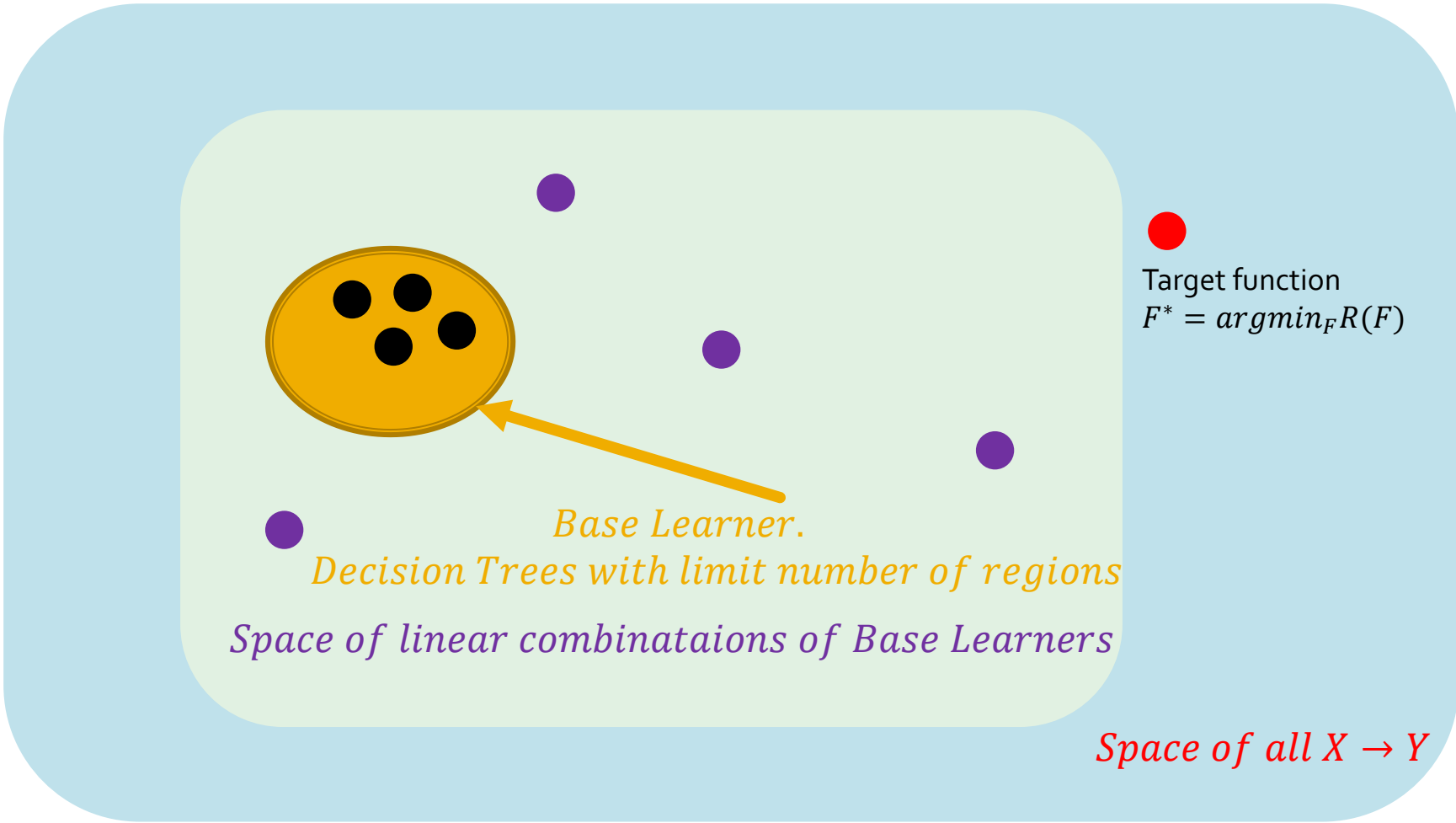
- Important note. For interval scale variable there is in fact infinite number of place where to insert split point. But as we discuss in [1] we allow to insert split point only in some specific place (like point between neighboors) for interval-scale variable.

- So let's consider huge linear combination of the trees.

$$\mathcal{F} = \{g: g = \sum_{i=1}^{J} a_i f_i \, , such\ that\ a_i \in R, f_i \in \ \mathcal{F}_{tree}\}$$

J is huge, but finite.

Boosting: illustration of model structure



33

Boosting. Why Boosting is mentioning in context of trees?

"You heard on the street people say Gradient Boosted Decision Trees. But why word boosting in context of trees?
Trees is mostly used because boosting improve quality of decision trees a lot, even other base learner can be used too" – J.Friedman

Boosting. Score on population

- $\mathcal{F} = \{g: g = \sum_{j=1}^{J} a_j f_j \, , such\ that\ a_j \in R, f_j \in \mathcal{F}_{tree}\}$
- $a^* = argmin\ E_{x,y}(L(y, \sum_{j=1}^{J} a_j f_j\ (x)))$
- If modify this problem with addition constraint that $a_j = \{0,1\}\ and\ a_j = 1$ only precisely for one j then it will move us to original regression problem for the tree
- So there is no loose of generality. Usual decision trees is elements of our space $\mathcal{F}$

Boosting. Score on data

- $a^* = argmin \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, \sum_{j=1}^{J} a_j f_j\left(x_i\right)\right)$

36

Boosting. We enormously increase function space and we bring problems

If consider decision trees with fixed number of terminal nodes as base learner. Then boosting increase hugely increase function space and it's good, but….

1. We bring problem that number of parameters is just huge and data is not enough to perform good fit. We have problem with variance.

2. But we will not have enough data to fit all this parameters, even remotely

3. What we're going do
   Linear regression with huge amount of parameters (variables in terms of optimization).

   Even in *a (not in x)* predictor it's just a linear models. But linear models can be huge too

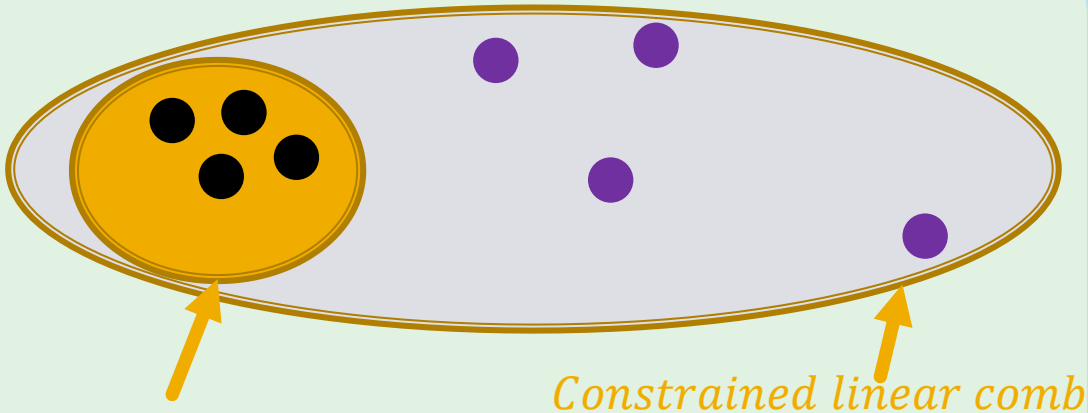
Boosting. About problems in search strategy

- *"This problem is not unique for Decision Trees or Machine Learning. With similar problem linear regression with huge amount of variables people faced in a lot of other areas" – J.Friedman*

Boosting. Huge linear regression.

- We observed some problems with solve emp. risk minimization $\boldsymbol{a^*_{orig} = argmin_a \widehat{R(a)}}$ [orig problem]
- Instead of it we will solve with tunable parameter t $\boldsymbol{a = argmin_a \widehat{R(a)}}$ **s.t.** $\mathbf{P}(\boldsymbol{a}) \leq \boldsymbol{t}$ [modified problem]
  - Here $\mathbf{P}(\boldsymbol{a})$ is constraint function which reduce size of feasible solutions and has several properties
  - $\mathbf{P}(\boldsymbol{a}) \geq \mathbf{0}$ just as a design choice
  - $\mathbf{P}(\boldsymbol{a}) = \mathbf{0}$ <=> $\boldsymbol{a} = \mathbf{0}$ just as a design choice.
  - So If $\boldsymbol{t} = \mathbf{0}$ => $\mathbf{P}(\boldsymbol{a}) \leq \mathbf{0}$ => $\boldsymbol{a} = \mathbf{0}$. This is a case with maximum constraint.
  - So If $\boldsymbol{t} \geq \mathbf{P}(\boldsymbol{a^*_{orig}})$ => constrain does not affect int opt. problem [orig]. Ad in this case $\boldsymbol{a} = \boldsymbol{a^*_{orig}}$. This is a case with no constraints at all.

Boosting. Huge constrained linear regression illustration.



Target function
$F^* = argmin_F R(F)$

Base Learner

Constrained linear comb. of base learners

Space of linear combinataions of Base Learners

Space of all $X \to Y$

40

Boosting. Why constraining has sense

- As far as restrict function space in correct way we will decrease variance because we will consider some functions

- Penalty constraint will restrict class of functions

- And as a reminder from [1,.p17]

"Reason of variance. It is about uncertanty which function is the best in case of limited amount of data"

Boosting. Reformulating our constrained problem as unconstrained

- Non-convex optimization problem:

$$\boldsymbol{a = argmin_a \widehat{R(a)}} \text{ s.t. } \mathbf{P}(\boldsymbol{a}) \leq \boldsymbol{t}$$

Is equivalent to

$$\boldsymbol{a = argmin_a \widehat{R(a)}} \text{ s.t. } \mathbf{P}(\boldsymbol{a}) - \boldsymbol{t} \leq \mathbf{0}$$ [opt.orig.problem]

And in fact via use **Exact Penalty Method** it can be reformulated as

$$\boldsymbol{a = argmin_a \widehat{R(a)} + \lambda \cdot max(0, P(a) - t)}$$

$$\boldsymbol{\lambda > 0}$$

In fact for $\boldsymbol{\lambda > \lambda_{critical}}$ solution for this problem coincident exactly with solution for [opt.orig.problem].

(One reference on it: S.Boyd ,EE364B/Lecture 11, 2008, 38:35, https://youtu.be/upMWYV7S1Yo?t=2313)

Boosting. Some intuition about reformulating constrained problem as unconstrained

Probably it can be show in this way:

1. $p^*_{orig} = min_a \widehat{R(a)}$
2. $\mathrm{p}^*_{\mathrm{modif}} = min_a \widehat{R(a)}$ s.t. $\mathrm{P}(a) \leq t$ (Here we have smaller feasible set then in 1)
3. $p^*_{orig} = min_a \widehat{R(a)} + 0 \cdot I(\mathrm{P}(a) \leq t)$ $(I(expr) = \begin{cases} 0, if\ expr\ is\ true \\ +\infty, if\ expr\ is\ not\ true \end{cases})$
4. $p^*_{\mathrm{modif}} = min_a \widehat{R(a)} + 1 \cdot I(\mathrm{P}(a) \leq t)$
5. Assume p^*_{orig} and $\mathrm{p}^*_{\mathrm{modif}}$ are finite
6. $p^*_{orig} \leq min_a \widehat{R(a)} + \lambda \cdot max(0, \mathrm{P}(a) - t) \leq p^*_{\mathrm{modif}}$
7. As we increasing λ cost per violation as more and more, so quanity in the middle can not decrease as far as λ is increasing

Boosting. Reformulating constrained problem as unconstrained

$$a = argmin_a \widehat{R(a)} + \lambda \cdot max(0, \mathrm{P}(a) - t)$$
$$\lambda > \lambda_{critical}$$

- If $\mathrm{P}(a) \geq t$ then

$$a^* = argmin_a \widehat{R(a)} + \lambda \cdot (\mathrm{P}(a) - t) = argmin_a \widehat{R(a)} + \lambda \cdot P(a)$$

- If $\mathrm{P}(a) < t$ then

$$a^* = argmin_a \widehat{R(a)} + \lambda \cdot 0 = argmin_a \widehat{R(a)} + 0 \cdot P(a)$$

So in both cases for any t optimal solution belong to optimal set of this problem:

$$a^* = argmin\left(R(a) + \lambda\big(P(a)\big)\right)$$
$$\lambda \geq 0$$

Boosting. Conclusion where find solution

So we come to conclusion that we should find a solution in this
One dimensional path in high dimensional space of solutions:

$a^*(\lambda) = argmin\left(R(a) + \lambda\big(P(a)\big)\right)$

$\lambda \geq 0$

Or let's say in such words. Solution (point from optimal set) for this optimization problem for any t lie on path mentioned above:

$$\boldsymbol{a = argmin_a \widehat{R(a)} = argmin\, E_{x,y}\left(L\left(y, \sum_{j=1}^{J} a_j f_j\,(x)\right)\right)}$$

$$\textbf{s.t. } \boldsymbol{P(a) \leq t}$$

Here:

1. $\lambda \in \mathbb{R} \cup \{+\infty\}$ is go from $+\infty$ to **o** (parameter)
2. $P(a) \geq 0$ is penalty function
3. And also we have $P(a) = 0$ <=> $a = \mathbf{0}$
4. $R(a)$ is empirical risk
5. a is an optimization variable

Couple observations

1. When $\lambda = 0$ => $a^*(0) = argmin\big(R(a)\big)$
2. When $\lambda = +\infty$ => $a^*(0) = \{a: P(a) = 0\} = \mathbf{0}$

Boosting. Why we need a path?

In fact **we don't need a path**.

What **we need only one point** in path defined as
$a^*(\lambda) = argmin\left(\widehat{R(a)} + \lambda\big(P(a)\big)\right)$
But we don't know λ.

What is really matter if just back to the start of out journey:
$a = argmin(E_{x,y}\left(L\big(y, F(x;a)\big)\right)$

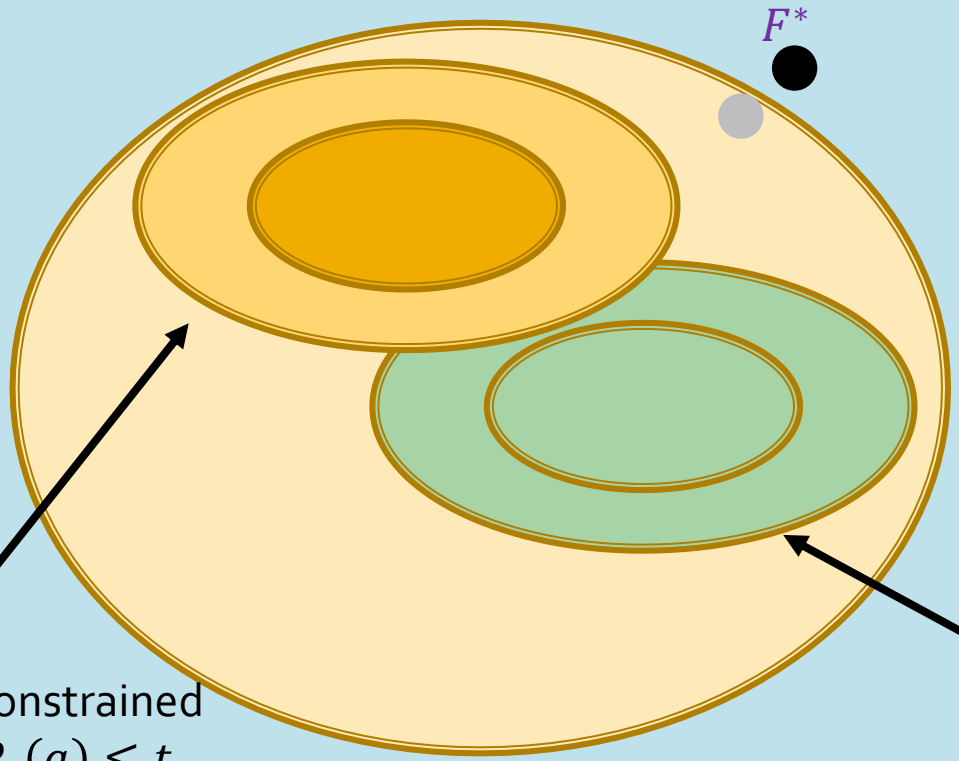
J.Friedman:
"There are a lot of criteria how people create rules to select which λ should be used. They are rather complicated and also depends on form of R, P.

*But in fact we can use cross-validation here. It's usual schema for model selection. Train in one **train-set**. And then test how we good in **test-set**."*

Also we can try to pick which penalty P use also via cross-validation.

Boosting. Another illustration of constraints

The big problem: we do not know what penalty to use!
Only one way: incorporate prior knowledge.....



Family of constrained Problems $P_2(a) \leq t$

Family of constrained Problems $P_1(a) \leq t$

47

Boosting. Assumptions for cold start

Big assumption #1:

- if we find $\tilde{\tilde{a}} \approx a^*$ => sparse is the same and is small (sparse a^* contains a lot of zeros) where a^* is target combination

Big assumption #2:

- Our target function can be approximated via sparse linear combination of base learners

So:

- When we will be near optimal configuration of weights the sparcity **pattern** will be the same
- But we don't know what is sparsity pattern
- Dense solution is definitely is not the thing that we're looking for

Boosting. Why our assumptions are good?

More deep reason that If our assumption is not true – the we will still loose.

Real phenomen as linear combination of our base learners	Predictor as linear comb. of base learners	Status of solution
Dense	**Dense**	We loose. We can not handle arbitrarily infinite series of coefficients
Sparse	**Dense**	We loose. We can not handle arbitrarily infinite series of coefficients. We should try to understand and find real good base learner, but nobody knows what are they.
Sparse	Sparse	We win. We can handle it.
Dense	Sparse	We loose. We're building poor approximation.

Boosting. Why our assumptions are good?

- I heard (from Stephen Boyd lectures near ~2017) that Modern Signal Processing (after 90-th) assume that signal in the nature are sparse in some specific basis. The goal to find this good basis.
- So such strategy has interconnection with signal processing.
- Also basis pursuit strategy from statistics and math optimization is very similar concept.
- We as humans have no instrument to handle infinite series, excluding series with specific repeating pattern or series given by some recurrent formulas.
- So sparse is something that we can handle and trees is *rather general weak learner* so it's all seems be a good idea.

Boosting. Penalties

- Penalites which produce sparse solution a
- We need to choose a penalty which produce sparse solution
- But we don't know sparse pattern
- We handle it in approximate way:
  - We can try to select from several function family
  - Each such function famity should have a parameter which regulates sparsity

Boosting. Penalties

$$a^*(\lambda) = argmin\left(\widehat{R(a)} + \lambda\left(P_\gamma(a)\right)\right)$$

- γ bridges dense to sparse
- Form of sparsity is encoded in form of P
- Now we roughly do parameter selection (λ, γ) via cross-validation.

Boosting. Possible penalties

- **Power family for penalty (PF)** $P_\gamma(a) = \sum|a_j|^\gamma$, $\gamma\epsilon[0,2]$. When $\gamma \geq 1$ – penalty is convex function
  - $\gamma = 2$ Ridge-regression
  - $\gamma = 1$ Lasso-regression
  - $\gamma = 0$ all subset selection ($a^0 = 0 \Leftrightarrow\ a = 0$)

- **Elastic Net (Zou and Hastie, 2005)**
  $P_\beta(a) = \sum(\beta - 1)a_j^2\frac{1}{2} + (2 - \beta)|a_j|$, $\beta\epsilon[1,2]$
  - $\beta = 1$ is Lasso
  - $\beta = 2$ is Ridge

- **Generalized Elastic Net (EN) (**J.Friedman mentioned that there is the following generalization)
  $P_\beta(a) = \sum \log\left((1 - \beta)|a_j| + \beta\right)$, $\beta\epsilon[0,1)$ **non-convex**
  - $\beta = 0$ is subset selection
  - $\beta \to 1(from\ the\ left)$ is Lasso (because ln(1+x)~x)

  p.s. I also saw a fun moment with S.Boyd:

  (EE364b, L15 https://youtu.be/DsXzUU691ts?t=2280 "Stephen, what are you doing?")

Boosting. Penalties compare

Point **o** is in very important point because it's place where the sensivity of forcing a_i to zero is encoded.

- For PF convex members (**except Lasso**) they have **o** derivative in a=o
- For PF non-convex members they have infinite derivative in a=o
- For EN exclude extreme cases $\beta = 0$ or $\beta = 2$ members have finite derivative in a=o

And it's all very important to force something a=o

- Infinite derivative - we can not handle it with our (usual) computation model in computer
- "o" derivative is useless. At least because: *when some a_i will go slightly out of zero it will not penalized*

Boosting. Strategy so far.

General strategy so far:

For each (β, λ)

$0 \leq \lambda \leq +\infty$

$\beta \epsilon [0,2]$ (Let's assume we use Generalized Elastic Net)

$$a^* = argmin\left(R(a) + \lambda\left(P_\beta(a)\right)\right)$$

- Select best (β, λ) via cross-validation
- Also if we will have enough compute power try several families of functions like PF, EN, Generalized EN

List of problems:

1. P_β can be non-convex
2. There is just huge a number of opt.problems if solve it one by one.
3. **We need really (algorithmically) fast enough algorithm**
4. Even discretized β and λ it is still be a big number of optimization problem to solve

Boosting. Solve approximately. Path seeking.

Big idea is instead solving sequence a problem look into one dimensional path which $a^*(\lambda)$ traced

$$a^*(\lambda) = argmin\left(\widehat{R(a)} + \lambda\left(P_\beta(a)\right)\right)$$

(for fixed β and form of P)

Step-1:
We always know end points of path:
At $\boldsymbol{\lambda = +\infty}$ $\boldsymbol{a = 0}$
And at $\boldsymbol{\lambda = 0}$ $\boldsymbol{a^*(\lambda) = argmin\left(R(a)\right)}$

Step-2: increase lambda a little bit and it will lead to another solution a^* which near the previous

Boosting. Path seeking formulation.

First of all we introduce extra quantities:

- ν – is path length
- $d\nu$ – is small increment of the path length
- $\nu(t)$ – is path length at iterate **t**
- $d(\nu)$ – is direction to move when we reach point in path
- $a^*(\lambda)$ with length ν from the start.

Step-1:
$\nu(0) = 0, a(0) = 0$ – at the beginning of the path (which correspond to $\lambda = +\infty$)

Step-2:
$a(\nu + d\nu) = a(\nu) + d(\nu)d\nu$

Step-3:
$\nu = \nu + d\nu$.

Increase ν correspond to decrease λ .
When λ is decreasing $R(a)$ in problem formulation $a^*(\lambda) = argmin\left(R(a) + \lambda\left(P_\beta(a)\right)\right)$ has more chance to be decreased. So $R(a)$ by itself has more opportunity to be minimized.

So we come to conclusion that $\boldsymbol{d(\nu)}$ **is direction of non-increasing value of** $\boldsymbol{R(a)}$

Step-4:
Stop when $R(a(\nu)) = p^* = min. R(a)$

Boosting. Path seeking methods before J.Friedman GPS method

Path-seeking methods are differ how find $d(v),\ dv$

There are some of them created in history:

- **Partially Least Square (PLS)**

Loss should be $L(y, \hat{y}) = (y - \hat{y})^2$
Penalty should be in form of Ridge.
(J.Friedman mentioned that it is widely used in Analytical Chemistry)

- **Least Angle Regression (LAR)**

Loss should be $L(y, \hat{y}) = (y - \hat{y})^2$
Penalty should be in form of Lasso
(Created by Bradley Efron from Stanford)
(J.Friedman mentioned that there is a trick which can upgrade this algorithm from "approximate" follow the path to exact)

- **Forward stepwise - Forward stepwise**

Loss should be $L(y, \hat{y}) = (y - \hat{y})^2$

General Path Seeking (GPS) method.

- It's fast path seeking algorithm.
- This algorithm approximates exact path seeking.
- Algorithm designed by Jerome H.Friedman which works for:
  - Any convex loss $L(y, F)$ which should be convex in (y, F)
    (not of parameters of *F* just convex in *y,F.* F can be arbitarily)
  - And convex or not $P(a)$ such that $\frac{\partial P}{\partial |a_j|} \geq 0$.
    I.e. $P(a)$ should be non-decreasing in $|a_j|$ $\forall j$

J.Friedman: "*It works for any functions (convex or not) from PF, and It works for any functions (convex or not) from generalized EN*"

p.s. I did not look into prove. So let me only describe algorithm, based on my Lecture notes.

Boosting. GPS need terminology

- $a(\nu)$ – vector of parameters as a function of length of the path we traced so far
- ν – link along a path
- $d\nu$ – small increment path length
- $g(\nu)$ – value negative gradient vector of the empirical risk in point $\widehat{a(\nu)}$.
  $g_j = -\left[\frac{\partial R}{\partial a_j}\right]|_{a=\widehat{a(\nu)}}$
- p – vector of partial derivatives of the penalty P w.r.t. to absolute values of all coefficients in point $\widehat{a(\nu)}$
- $p_j = \left[\frac{\partial P}{\partial |a_j|}\right]|_{a=\widehat{a(\nu)}}$
- $\lambda_j = \frac{g_j(\nu)}{p_j(\nu)}$ component-wise ratios.

GPS algorithm as fundament for Boosting

This algorithm emulates path obtained of solving the following optimization problem:

$a^*(\lambda) = argmin\Big(R(a) + \lambda\big(P_\beta(a)\big)\Big)\ \lambda \in [+\infty, 0]$

Algorithm from Jerome Friedman:

1. $\nu = 0$ and $\hat{a}(0) = 0$
2. **Start loop**
3. Compute $\lambda_j(\nu) = \frac{g_j(\nu)}{p_j(\nu)}$
4. Compute set S such that $S = \{j|\lambda_j(\nu)a_j(\nu) < 0\}$ i.e. $\lambda_j(\nu)\ and\ a_j(\nu)$ has different signs.
5. If S is empty then $j^* = argmax_j\big(|\lambda_j(\nu)|\big)$
6. If S is not empty then $j^* = argmax_{j\epsilon S}\big(|\lambda_j(\nu)|\big)$
7. For $j = j^*$ --- $\widehat{a_{j^*}}(\nu + d\nu) = \widehat{a_{j^*}}(\nu) + d\nu \cdot sign(\lambda_{j^*}(\nu))$
8. For $j \neq j^*$ --- $\widehat{a_j}(\nu + d\nu) = \widehat{a_j}(\nu)$
9. $\nu = \nu + d\nu$
10. Until $\lambda(\nu) = 0$ i.e. all elements of all λ is zero. In fact we're interesting in case of $g = 0$

Note:

"In fact if use steps #4, #6 and half of step #5 then it will more closely approximate a path. But in practice really it can be discarded." – J.Friedman

"For case when penalty is convex optimization problems path is continuous, for non-convex penalties path have discontinuities. And algorithm don't provide such paths." – J.Friedman

Boosting. Apply GPS to formalize boosting algorithm

- $\{f(x;b)\}_{b\epsilon\mathcal{B}}$ is a class of **base(or weak) learners**. In our case it's all possible decision trees build with CART regression (with fixed strategy to select split point)

- We're going to construct predictor in form

$$\mathrm{F}(x) = \sum_{j=1}^{J} a_j f_j(x;b_j)$$

- We're going to find coefficient via algorithm via which we solve:
$a^*(\lambda) = argmin\left(\widehat{R(a)} + \lambda\left(P_\beta(a)\right)\right)$

- There are so enormous components number of a so we idea use GPS algorithm

Boosting. Compute negative gradient vector of the empirical risk

$g_j = -\left[\frac{\partial \hat{R}}{\partial a_j}\right]|_{a=\widehat{a(v)}}$ we can use chain rule

$g_j = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial(L(y_i,F_i)}{\partial F_i}\frac{\partial F_i}{\partial a_j} = \frac{1}{N}\sum_{i=1}^{N} -\frac{\partial(L(y_i,F_i)}{\partial F_i} f(x_i;b_j)$ [*]

But:

$l_i = -\frac{\partial(L(y_i,F_i)}{\partial F_i}$ - it's a property of the loss only. E.g. $L = \frac{(y-F)^2}{2}$ => $l = (y - F)$

This quanity has different names:

- Pseudo-response - should be called, because it specifically what the tree try to predict as it will be seen in future
- Generalized Residual - the term that people is really use.

Now via this quanity equation [*] can be written as:

$g_j = \frac{1}{N}\sum_{i=1}^{N} l_i \cdot f(x_i;b_j)$

Boosting. Extra couple sets

We need to introduce couple sets:

- Active set of parameters
  $A(\nu) = \{j : a_j(\nu) \neq 0\}$

- (Active) Set of Base learners
  $H(\nu) = \{f(x; b_j) : a_j(\nu) \neq 0\}$

Boosting GPS algorithm.

Name as J.Friedman attached is Boosting GPS algorithm, which nobody implemented currently.

Step#1
$\nu = 0$ and $A(\nu) = \{\}$ and $H(\nu) = \{\quad\}$

Step #2
Start loop to choose which one update

Step #3
Compute $\lambda_j(\nu) = \frac{g_j(\nu)}{p_j(\nu)}$ and do it for active variables i.e. $j \in A$

Step #4
Compute set S such that $S = \{j | \lambda_j(\nu) a_j(\nu) < 0\}$ i.e. $\lambda_j(\nu)\ and\ a_j(\nu)$ has different signs and do it for active variables i.e. $j \in A$

Boosting GPS algorithm.

#Step 5

If S is empty then

$$j^* = argmax_{j \epsilon A(v)}\left(|\lambda_j(v)|\right)$$

which happens more often then case-6

Magic step-1:

Assume we know $\lambda_k(v) = \frac{g_k(v)}{p_k(0)}\, k \notin A(v)$

Magic step-2:

$$k^* = argmax_{k \notin A(v)}\left(|\lambda_k(v)|\right)$$

Boosting GPS algorithm.

#Step 5.5

If $|\lambda_{j^*}(\nu)| \geq |\lambda_{k^*}(\nu)|$
{
// Do not include weak learner k^* into model
$\widehat{a_{j^*}}(\nu + d\nu) = \widehat{a_{j^*}}(\nu) + d\nu \cdot sign(\lambda_{j^*}(\nu))$
$\widehat{a_j}(\nu + d\nu) = \widehat{a_j}(\nu)$ (and $j \neq j^*, j \in A$)
$\nu = \nu + d\nu$
}
else
{
// Include weak learner into the model
$A(\nu + d\nu) = A(\nu) \cup \{k^*\}$
$H(\nu + d\nu) = H(\nu) \cup \{f(x; b_{k^*})\}$
For $j \neq k^*$ --- $\widehat{a_j}(\nu + d\nu) = \widehat{a_j}(\nu)$ ($j \in A$)
For $j = k^*$ --- $\widehat{a_{k^*}}(\nu + d\nu) = \widehat{a_{k^*}}(\nu) + d\nu \cdot sign(\lambda_{k^*}(\nu)) = 0 + d\nu \cdot sign(\lambda_{k^*}(\nu))$
}

Boosting GPS algorithm.

Step #6

If S is not empty then

$j^* = argmax_{j \epsilon S}\left(|\lambda_j(\nu)|\right)$

$\widehat{a_{j^*}}(\nu + d\nu) = \widehat{a_{j^*}}(\nu) + d\nu \cdot sign(\lambda_{j^*}(\nu))$

$\widehat{a_j}(\nu + d\nu) = \widehat{a_j}(\nu)$ (For $j \neq j^*$)

Increment ν: $\nu = \nu + d\nu$

Boosting GPS algorithm.

Step# 7

End of the loop. Goto step#2.

Leve the cycle when $g_j(v) = 0$for all active variables i.e. in fact we're interesting in case when $g = 0$ and we're in local minimum

Boosting GPS algorithm.

People ignore in implementations active and non-active set.
And it saves some time.
Argument of people who such thing:
We really don't know what we're finding

Magic step-1/step-2 description:

$g(b;\nu) = \frac{1}{N}\sum_{i=1}^{N} l_i(\nu) \cdot f(x_i;b)$
$p_k(0)$ is the same so really what we want find based learner parameters such that.

So $\lambda_k(\nu) = \frac{g_k(\nu)}{p_k(0)}$ is maximized <=> $g_k(\nu)$ is maximized <=>
So $b^*(\nu) = argmax_b(|g(b;\nu)|)$

Boosting GPS algorithm. Handle magic steps via equivalency of two optimization problems

Assume we have several r.v. $\boldsymbol{x_j}$ and one **y**

Also:

If $E[x_j] = 0$ and $D[x_j] = const$

Then

$$D[x_j] = E[x_j^2] - E[x_j]^2 = E[x_j^2] - 0 = E[x_j^2]$$

$$j^* = argmax_j(|corr(x_j, y)|) = \frac{|cov(x, y)|}{\sqrt{D[x]}\sqrt{D[y]}} = \cdots$$

$$= argmax_j|E[x_j y]|$$

Boosting GPS algorithm. Handle magic steps via equivalency of two optimization problems

$$j^* = argmin_j \left(\min_{\rho} E\left[y - \rho x_j\right]^2 \right) = argmin_j \left(\min_{\rho} \left(E[y^2] - 2\rho E\left[y x_j\right] + \rho^2 E\left[x_j^2\right] \right) \right)$$

$E[y^2]$ can removed it's just additive constant and does not affect into j^*.

For fixed j inner minimization problem is unconstrained convex optimization problem with quadratic objective

$$\left(\rho^2 E\left[x_j^2\right] - 2\rho E\left[y x_j\right] \right)'_{\rho} = 0 \; \rho = \frac{E\left[y x_j\right]}{E\left[x_j^2\right]}$$

$$j^* = argmin_j \left(\min_{\rho} E\left[y - \rho x_j\right]^2 \right) = \cdots argmin_j \left(-\frac{E\left[y x_j\right]^2}{E\left[x_j^2\right]} \right) = \cdots$$

$$= argmax \left(E\left[y x_j\right]^2 \right) = argmax_j \left| E\left[x_j y\right] \right|$$

Boosting GPS algorithm. Handle magic steps via equivalency of two optimization problems

- $E[x_j] = 0$ and $D[x_j] = const$
- x_j, y are random variables

So I shortly without derivations mentioned that solution for two optimization problems:

- $j^* = argmin_j \left(\min_{\rho} E[y - \rho x_j]^2 \right)$ and
- $j^* = argmax_j |E[x_j y]|$

Is the same!

So last result is used to move from problem:

$$b^*(v) = argmax_b(|g(b;v)|) = argmax_b\left(\left|\frac{1}{N}\sum_{i=1}^{N} l_i \cdot f(x_i;b)\right|\right)$$

$$b^*(v) = argmin_{b,\rho}\left(\left|\frac{1}{N}\sum_{i=1}^{N} (l_i(v) - \rho \cdot f(x_i;b))^2\right|\right)$$

We come to task where we need:

- Predict generalized residual but we need
- Loss is measured by quadratic loss (not original loss)
- ρ^* is useless because decision trees have already constant value per region. But formally it's in optimization problem. For trees it can **obserbed** into $f(x_i;b)$

Lasso Gradient boosting (implemented in XgBoost)

- $P(a) = |a|_1 \Rightarrow \frac{\partial P}{\partial |a_j|} = 1$
- Assume that set S is always empty
- Step size selection
  $dv = eps \cdot argmin_\eta \sum_{i=1}^{N} L(y_i, F_i(v) + \eta \cdot f(x_i; b^*(v)))$
- $eps \approx 0.1$ is "linkage" factor, also known as "learning factor". Idea is not making full-step. It's also called learning rate
- All this lead to more simplified version of algorithm

Boosting. Lasso Gradient boosting (implemented in XgBoost)

- $k = 0$ and $\{F_i = 0\}$. Where F_i is current prediction for i-th observation. And k is number of base learners in the model.
- Start loop to choose which one update
- Compute generalized residuals (or pseudo-response) $l_i(v) = -\frac{\partial L(y_i, F_i)}{\partial F_i}$
- Find best learner to predict $b^*(v), \rho^*(v) = argminx_{b,\rho} \left(\left| \frac{1}{N} \sum_{i=1}^{N} (l_i(v) - \rho \cdot f(x_i; b))^2 \right| \right)$
- Compute step size $dv = eps \cdot argmin_\eta \sum_{i=1}^{N} L(y_i, F_i(v) + \eta \cdot f(x_i; b^*(v)))$ and eps should be as small as possible
- $k = k + 1$; $b_k = b^*$; $\widehat{a_k} = dv$
- $F_i = F_i + dv \cdot f(x_i; b_k)$ for all i=1,N
- Stop when we received enough trees

Tricks:

- Decide when to stop append trees. How many trees do we need - can be done via cross-validation.

- $dv = eps \cdot argmin_{\eta} \sum_{i=1}^{N} L(y_i, F_i(v) + \eta \cdot f(x_i; b^*(v)))$

Can be considered not only as one scalar but really η can be inserted into the tree due to structure of $\boldsymbol{f}$
Instead $\eta f\left(x; b^*(v)\right) f = \sum_{m=1}^{M} c_m I(x \epsilon R_m)$

Merry Christmas and a happy new year!