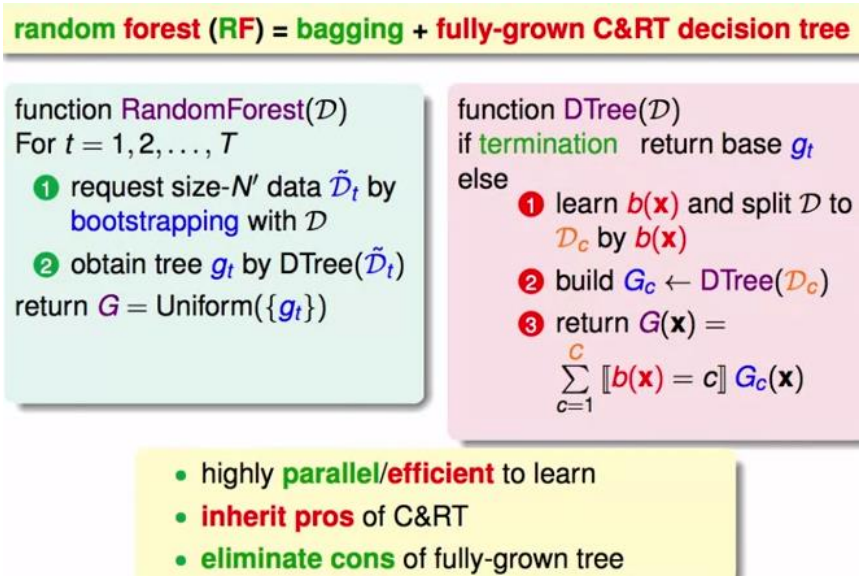
Random forest (bagging + fully-grown decision tree)

A basic random forest:



Randomly resample features. The dimension of selected features should be much less than original dimension. Since the features in each tree are different, the variance would be very large. Besides, it makes model robust and prevents overfitting. 每棵树的 features 都不同，新选出来的特征维度要远小于原来的维度。这样 variance 会更大。不同的 feature 也降低了 overfitting。

In each new branch, rf resample features without replacement.

Bagging improves prediction accuracy at the cost of interpretability. bagged tree 不如单棵树来的好解释。

Should Bagging use unpruned tree? I think it depends on oob error.

Section 15.3.4 of Elements of Statistical Learning (Hastie et al 2009) (PDF is freely available) discusses this. In short, depending on your point of view, random forest *can* overfit the data, but not because of `ntree`.

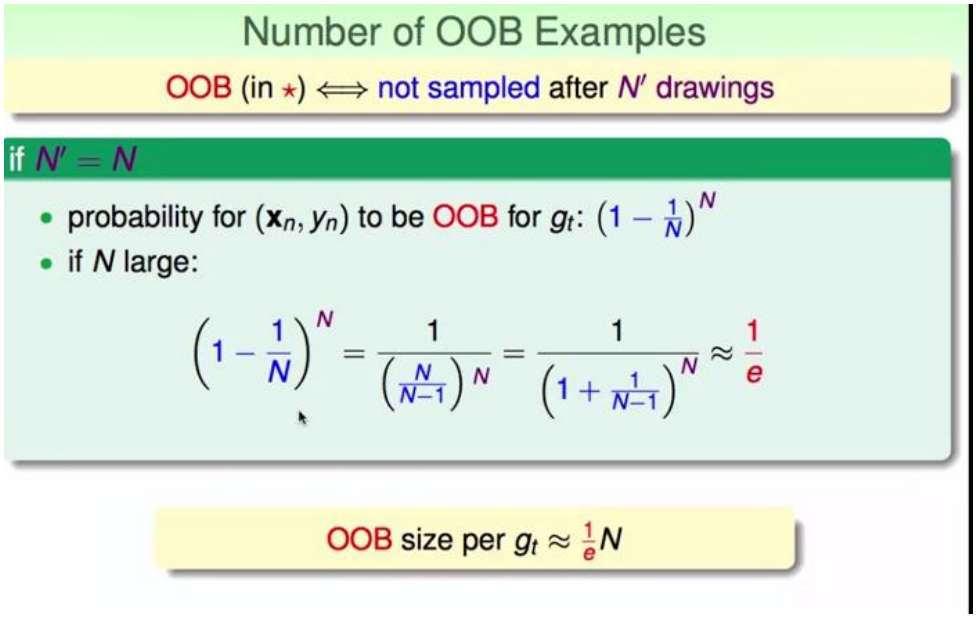
Hastie et al (2009, page 596) states "it is certainly true that increasing $\mathcal{B}$ [the number of trees] does not cause the random forest sequence to overfit". However, they also state that "the average of fully grown trees can result in too rich a model, and incur unnecessary variance" (*op cit.*).

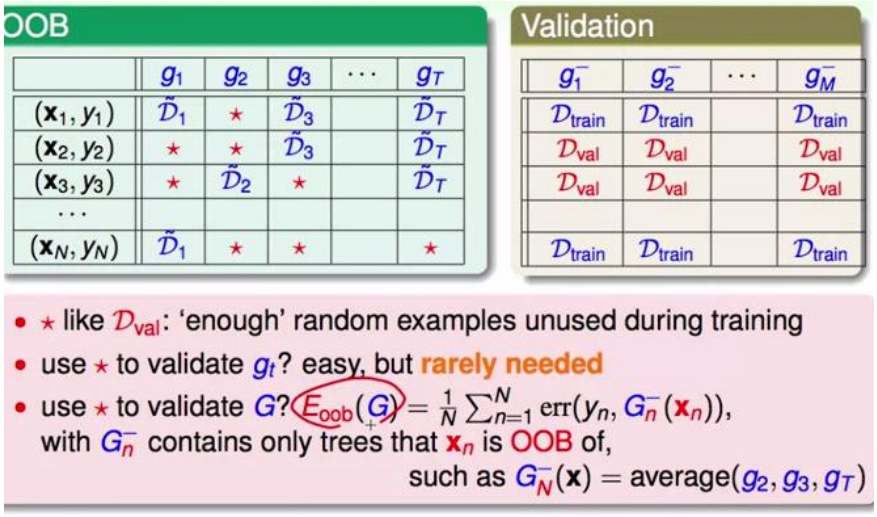
In short there may be some overfitting due to the use of fully grown trees (the "average" that they talk about in the second statement), which may be showing up as you add more trees to the forest.

Hastie et al (2009) suggest that this "overfitting" does not often cost much in terms of prediction error, and if you don't tune that parameter then tuning is simplified.

If you want to assure yourself, you could try tuning over `mtry` plus `nodesize` and/or `maxnodes` the latter two which controls the depth of trees fitted.

Out of bag error:





For each data, rf uses models which do not use this data to do validation.
对每条数据都用所有未使用到这条数据的 g 来检验，所以 random forest 不用另外的 validation 了。但是他最后举例也还是用的 validation error，这要看下？

Why rf leads to large margin from a mathematical view.
Noise corrected by voting
The more trees, the better

Bagged tree vs rf: 对于 bagged tree 和 RF 来说区别就是 RF 每次 branch 都选择 feature 的 subset 例如根号下 N。 这样做可以有效地避免每棵树都长得很相似，这样做也是为了增大 variance。
The biggest difference between bagged tree and random forest is that RF resamples the features in each branch.

- Build a random forest
  - At each split:
  - The algorithm is not allowed to consider a majority of the available predictors.
  - What is the reasoning?
  - Suppose one very strong predictor along with many moderately strong predictors.
- In the collection of bagged trees, most or all of the trees will use this strong predictor in the top split.
- All of the bagged trees will look very similar.
- Predictions from the bagged trees highly correlated.
- Averaging highly correlated trees doesn't lead to as large of a reduction in variance as averaging many uncorrelated trees.
- This means that bagging will NOT lead to a substantial reduction in variance over a single tree for this example.

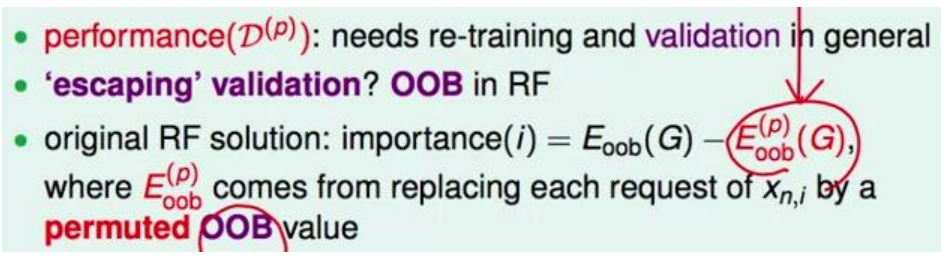
- How do random forests overcome this problem?
- Force each split to consider only a subset of the predictors.
- On average, $(p-m)/p$ of the splits won't even consider a strong predictor
- Means other predictors have more of a chance!
- This process de-correlates the trees.

Two feature selection methods from random forest:

1. impurity decrease of decision tree.
2. performance decrease。

Feature selection using random forest. In sklearn feature_importances_ method uses the first way to score features.

Permutation test: 用原来的资料的训练结果和第 N 个维度洗牌（不添加各种 gaussian noise 为了不改变 distribution）之后的资料的训练结果对比，就可以看到第 N 个维度有多重要。



为了避免针对洗牌后的数据再训练一次，可以对于后面的 performance。我还用以前的 G 来做 validation。但是我把这条数据的 xn,i 替换成 out of bag 里面的随机一个值。

To prevent retraining the model, we can calculate the second performance by using original model with different input data. We can randomly choose a data from out of bag to replace the original data.

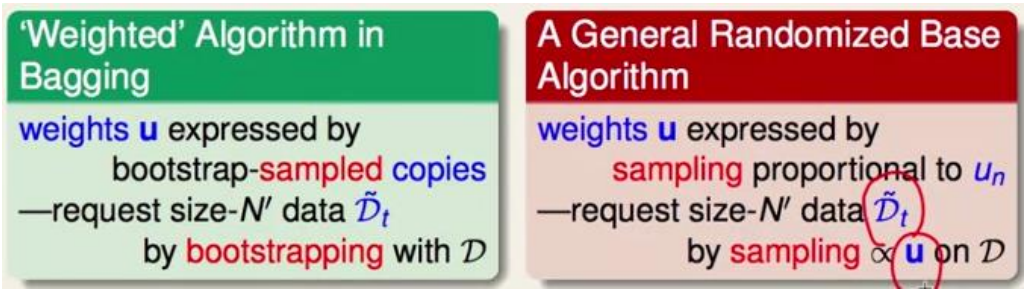
Gradient Boosting Decision Tree

Difference between GBDT and adaptive boost decision tree!

$$L(y, f(x)) = \exp(-y\, f(x)). \qquad (10.8)$$

Mainly used in information retrieval (search engine)!
How to set weight

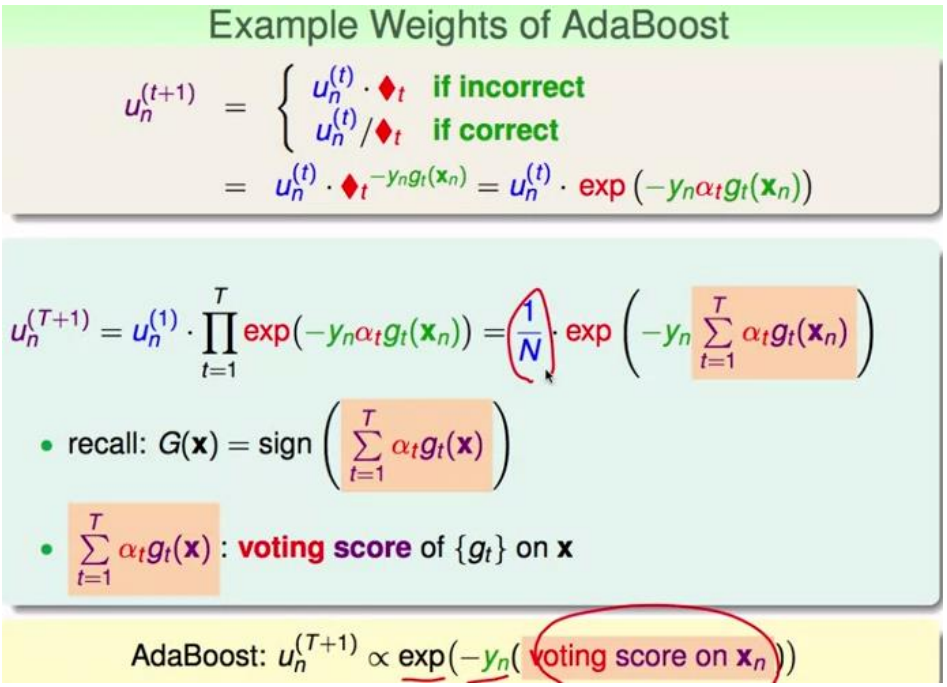
权重代表资料有几份。根据权重 sample，因为 decision tree 很复杂如果是把权重放进 DT 方法里面就很不好办，但是如果是用 stump 就很简单，就不用做 sample 的工作了，可以把权重放进 stump 里面。

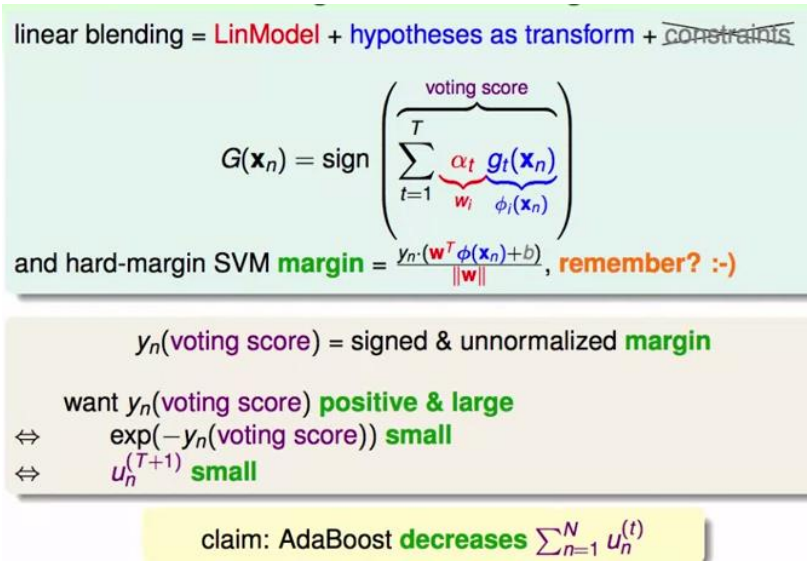


Use stump or not？ Depends on oob error?

Compare adaboost and svm. The margin
Sum of all weights should decrease gradually in Adaboost. 的每一条数据的权重之和会随着迭代次数不断减小。
感觉就是把求 u 和 alpha 和 G 联系起来

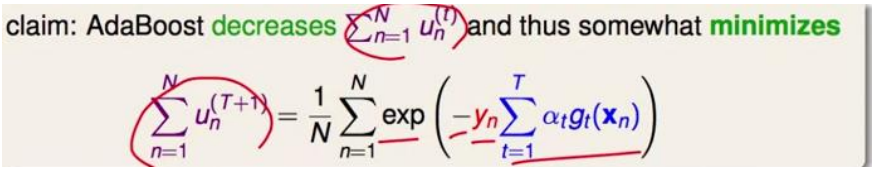
Prove adaboost from gradient view. Prove from a mathematical view.



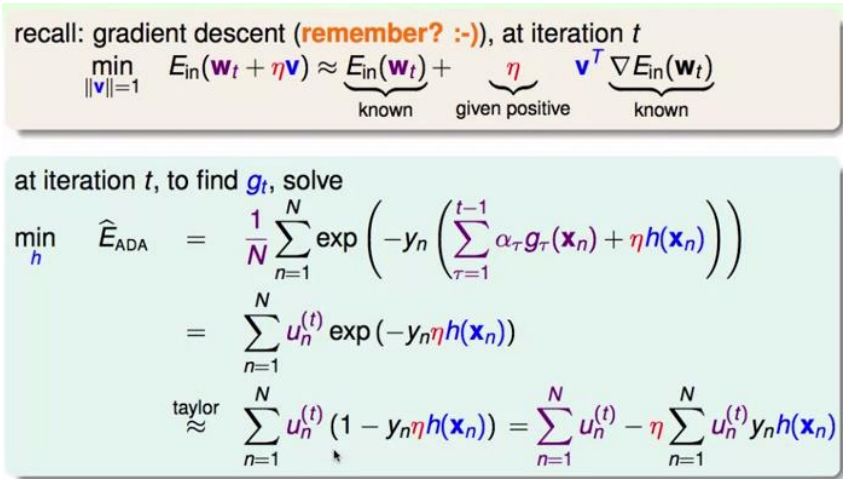


由于把 voting score 也看成是一种 margin distance，所以希望他越大越好，所以希望 u 越小越好。所以如果 u 每次迭代越来越小就说明了 adaboost 模型 makes sense。

We can treat voting score as a kind of margin distance, so we need large voting score. By contrast, we need small u.



Get u by using gradient descent.



为了最小化第二步，要使得后面一项-bla 越小越好

所以从数学上验证了是让 Ein 变小。

finding good h (function direction) $\Leftrightarrow$ minimize $\sum_{n=1}^{N} u_n^{(t)} (-y_n h(\mathbf{x}_n))$

for binary classification, where y_n and $h(\mathbf{x}_n)$ both $\in \{-1, +1\}$:

$$\begin{aligned}\sum_{n=1}^{N} u_n^{(t)} (-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^{N} u_n^{(t)} \begin{cases} -1 & \text{if } y_n = h(\mathbf{x}_n) \\ +1 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^{N} u_n^{(t)} + \sum_{n=1}^{N} u_n^{(t)} \begin{cases} 0 & \text{if } y_n = h(\mathbf{x}_n) \\ 2 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^{N} u_n^{(t)} + 2E_{\text{in}}^{\mathbf{u}^{(t)}}(h)\end{aligned}$$

—who minimizes $E_{\text{in}}^{\mathbf{u}^{(t)}}(h)$? $\mathcal{A}$ in AdaBoost! :-)

$\mathcal{A}$: good g_t = h for 'gradient descent'

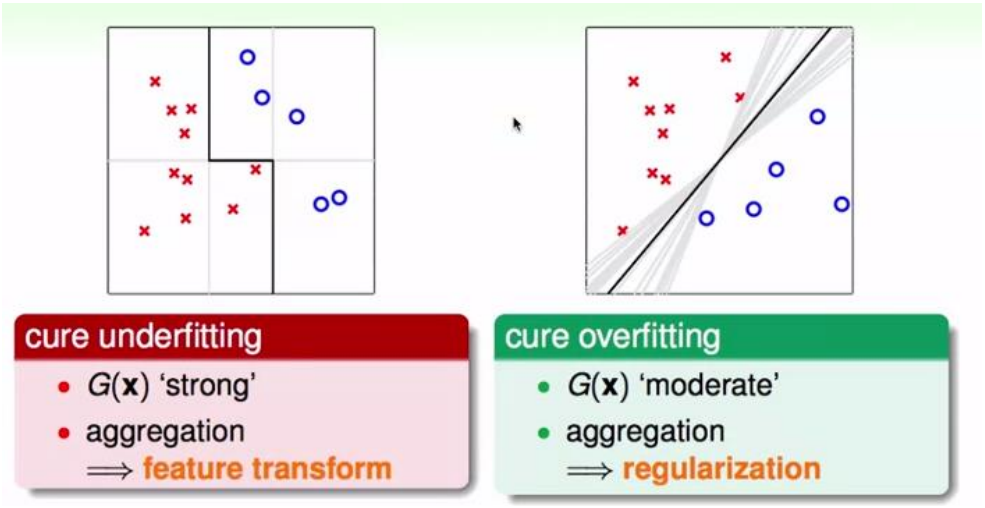
从数学上证明 alpha 应该是那么大，alpha 是在最佳化 u，每一步都走最大的尽可能让 u 减小。We prove alpha should equal to this value. Alpha tries to optimize u.

AdaBoost finds g_t by approximately $\min_h \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n))$

after finding g_t, how about $\min_\eta \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n))$

- optimal η_t somewhat 'greedily faster' than fixed (small) η
  —called steepest descent for optimization
- two cases inside summation:
  - $y_n = g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(-\eta)$ (correct)
  - $y_n \neq g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(+\eta)$ (incorrect)
- $\widehat{E}_{\text{ADA}} = \left(\sum_{n=1}^{N} u_n^{(t)}\right) \cdot \left((1 - \epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta)\right)$

by solving $\frac{\partial \widehat{E}_{\text{ADA}}}{\partial \eta} = 0$, steepest $\eta_t = \ln \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \alpha_t$, remember? :-)



cure underfitting
- $G(\mathbf{x})$ 'strong'
- aggregation
  $\Longrightarrow$ feature transform

cure overfitting
- $G(\mathbf{x})$ 'moderate'
- aggregation
  $\Longrightarrow$ regularization

Aggregation model is like regression after feature transformation by using hypotheses.

In sklearn

We now have introduced a number of hyperparameters — as usual in machine learning it is quite tedious to optimize them. Especially, since they interact with each other (`learning_rate` and `n_estimators` , `learning_rate` and `subsample` , `max_depth` and `max_features`).

We usually follow this recipe to tune the hyperparameters for a gradient boosting model:

1. Choose `loss` based on your problem at hand (ie. target metric)
2. Pick `n_estimators` as large as (computationally) possible (e.g. 3000).
3. Tune `max_depth` , `learning_rate` , `min_samples_leaf` , and `max_features` via grid search.
4. Increase `n_estimators` even more and tune `learning_rate` again holding the other parameters fixed.

Higher tree complexity，less learning rate，get to minimum error very fast。

Three ways to reduce overfitting: (control parameters in gbdt)

1. Control tree structure.
2. Shrinkage (add learning rate to alpha*hypothesis)
3. Stochastic gradient boosting. (Subsampling the training set before growing each tree and subsampling the features before finding the best split node.)

Gradient boost for regression:

$$\min_{\eta} \min_{h} \frac{1}{N}\sum_{n=1}^{N} \text{err}\Big(\underbrace{\sum_{\tau=1}^{t-1}\alpha_\tau g_\tau(\mathbf{x}_n)}_{s_n} + \eta h(\mathbf{x}_n), y_n\Big) \quad \text{with } \text{err}(s,y) = (s-y)^2$$

Add a new g to decrease error. It is like the idea of gradient descent. The new g is the direction. We take a small step towards this direction to decrease error.
加上一个新的 g，使得 error 变得更小，好像 gradient descent 的思想，这个新的 g 就是一个方向，原来的整体朝着这个新的 g 方向走一步，使得 error 变小。

Gradient Boosted Decision Tree (GBDT)

$s_1 = s_2 = \ldots = s_N = 0$
for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\{(\mathbf{x}_n, y_n - s_n)\})$ where $\mathcal{A}$ is a (squared-error) regression algorithm
   —**how about sampled and pruned C&RT?**
2. compute $\alpha_t = \text{OneVarLinearRegression}(\{(g_t(\mathbf{x}_n), y_n - s_n)\})$
3. update $s_n \leftarrow s_n + \alpha_t g_t(\mathbf{x}_n)$

return $G(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$

最开始先用 x 解一个最基本的 regression。之后每一次用 residual 当 y。用 x 来 regression y，然后再用得到的小 g 当 x 来 regression residual。感觉是用两次 regression 不断叠加结果，不断接近最终结果，使得 error 越来越小。
While(1){
To get g, we use a base learner (decision tree) to solve a regression problem (x, residual).
To get alpha, we solve the second regression problem (g, residual)
}

Reference

Feature selection methods from rf
http://blog.datadive.net/selecting-good-features-part-iii-random-forests/
rf vs bagged tree
http://www.stat.cmu.edu/~rsteorts/slides/slides_lecture15.pdf
gbdt
http://www.datarobot.com/blog/gradient-boosted-regression-trees/