Machine Learning:: Trees and Ensemble Methods

Dozenten: Prof. Dr. M. O. Franz, Prof. Dr. O. Dürr

Remark: Much of the material have been developed together with Beate Sick (ZHAW/Uni Zurich).

Content

- Decision Trees
  - Decision Trees are the basis for recent ML developments
- Bagging and Random Forest
  - Random Forest
- Boosting
  - Ada-Boost (Algorithm)
  - Gradient Boosting



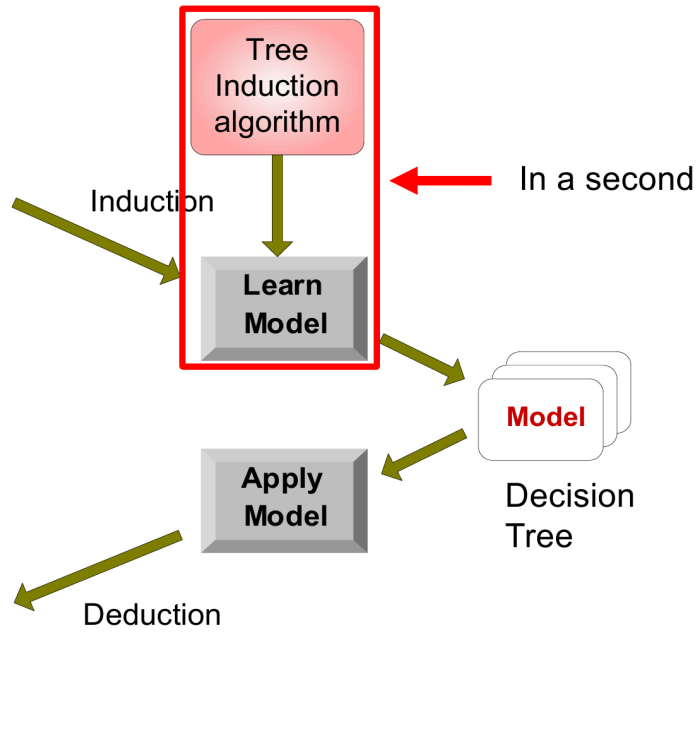
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

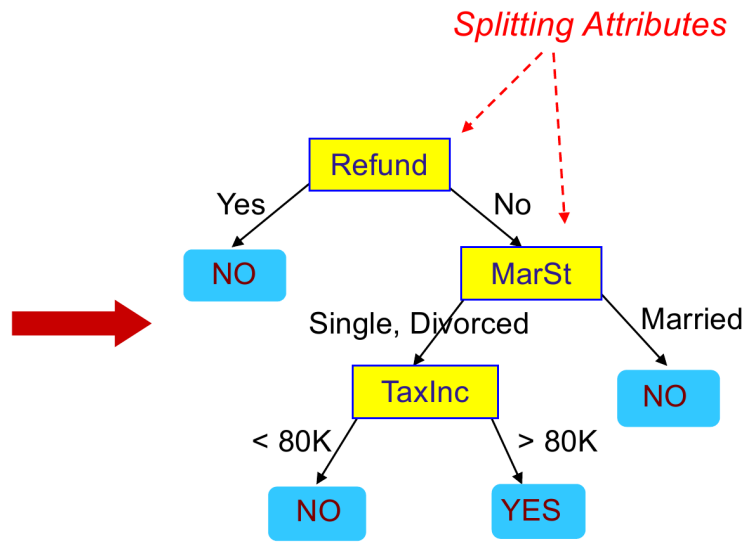


Source: Tan, Steinbach, Kumar

Example of a Decision Tree for classification

Tid	Refund	Marital Status	Taxable Income	Cheat
		categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

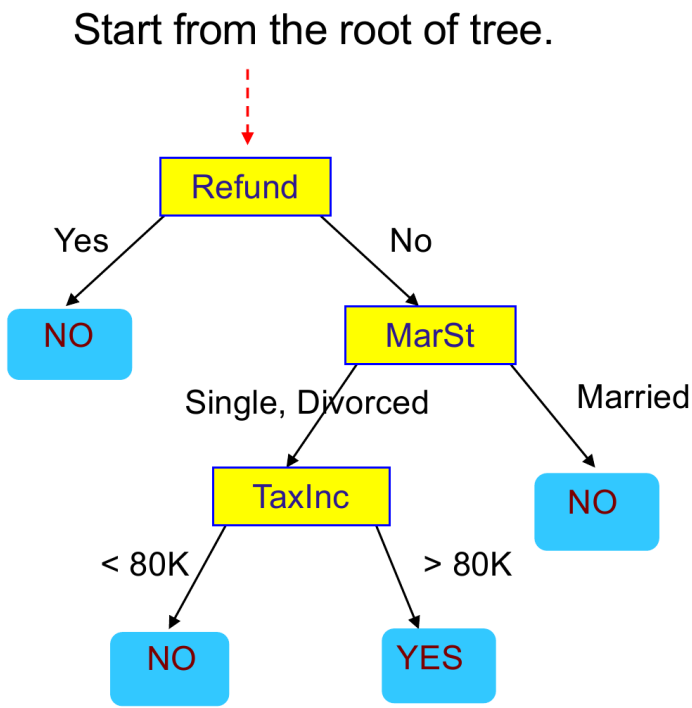
Training Data



Model: Decision Tree

Source: Tan, Steinbach, Kumar

Apply Model to Test Data



Test Data

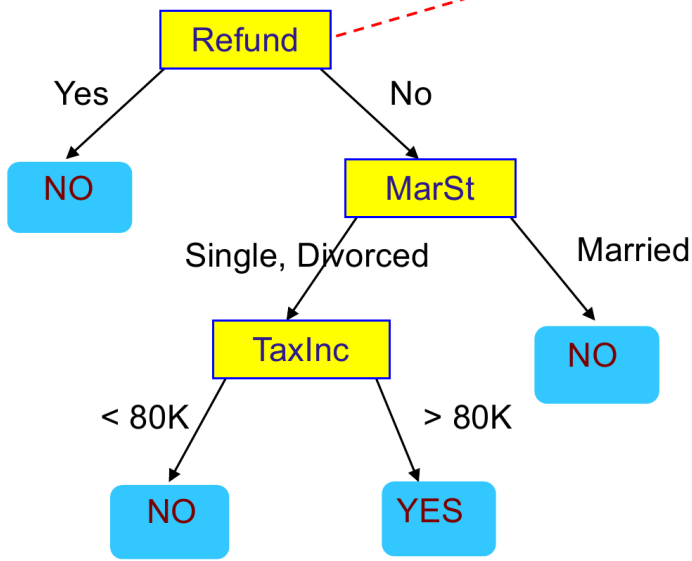
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Source: Tan, Steinbach, Kumar

Apply Model to Test Data

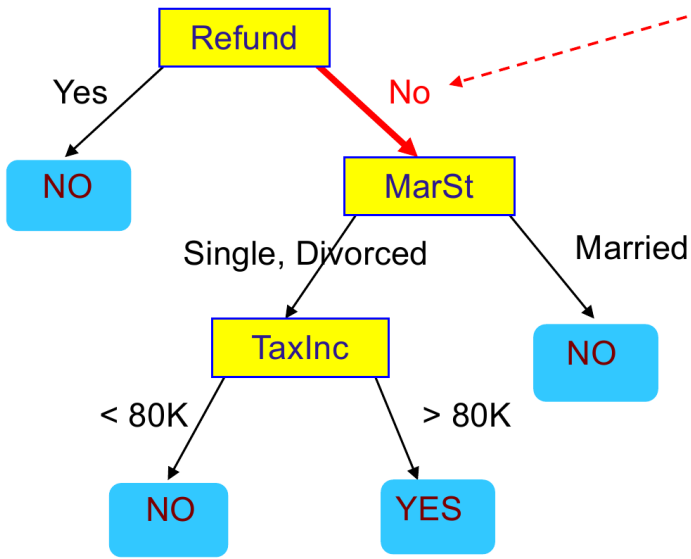
Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Source: Tan, Steinbach, Kumar

Apply Model to Test Data

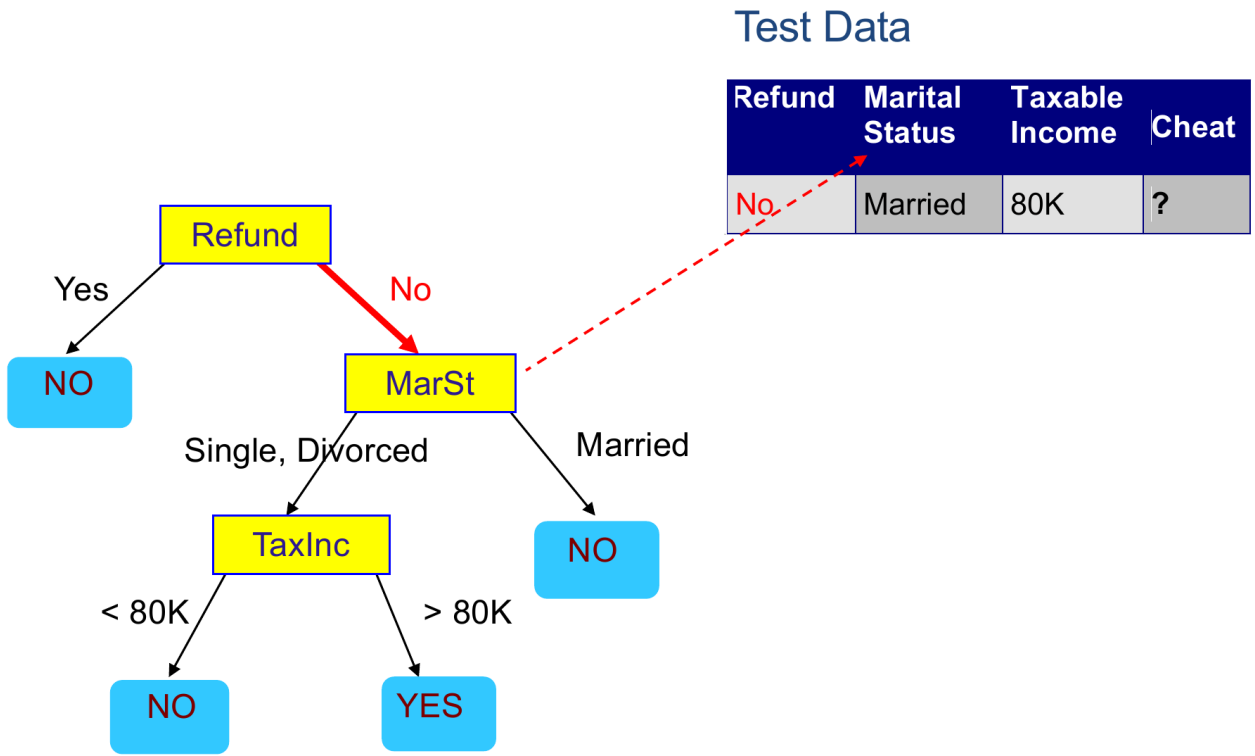


Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

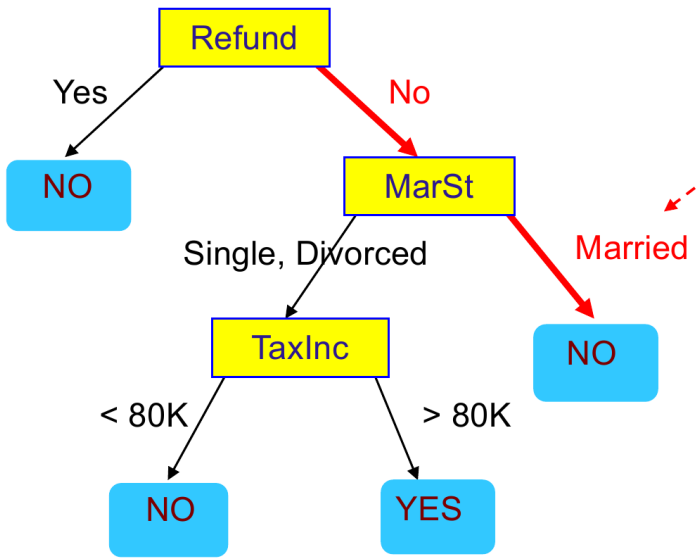
Source: Tan, Steinbach, Kumar

Apply Model to Test Data



Source: Tan, Steinbach, Kumar

Apply Model to Test Data

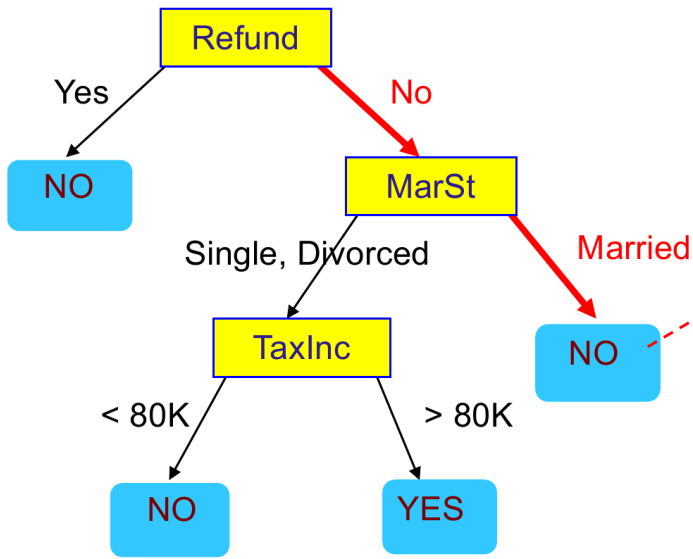


Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Source: Tan, Steinbach, Kumar

Apply Model to Test Data



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

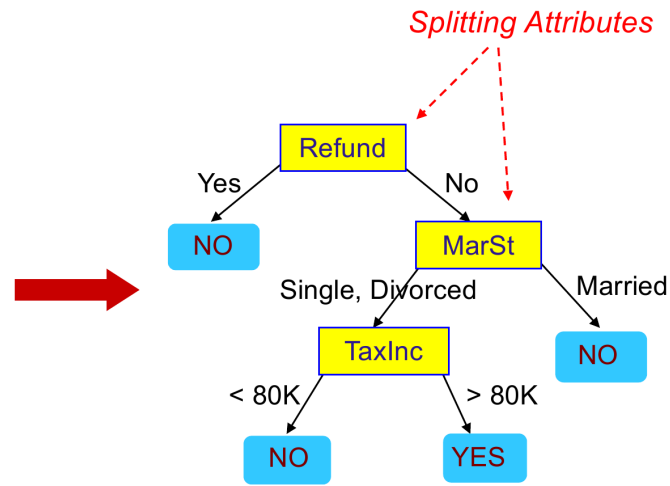
Assign Cheat to “No”

Source: Tan, Steinbach, Kumar

Ambiguity of learned tree:
Example of a Decision Tree for classification

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

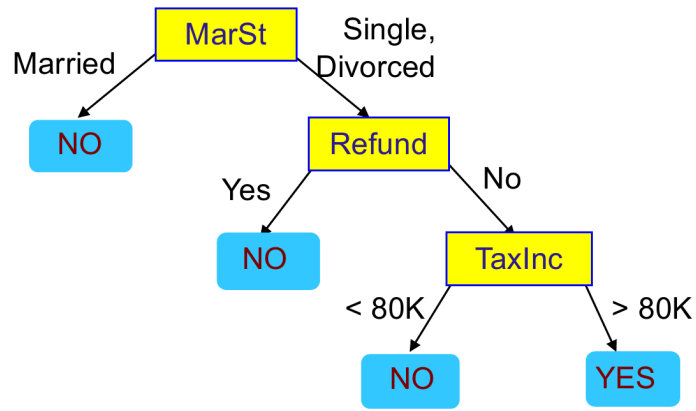


Model: Decision Tree

Source: Tan, Steinbach, Kumar

Ambiguity of learned tree:
Example of a Decision Tree for classification

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

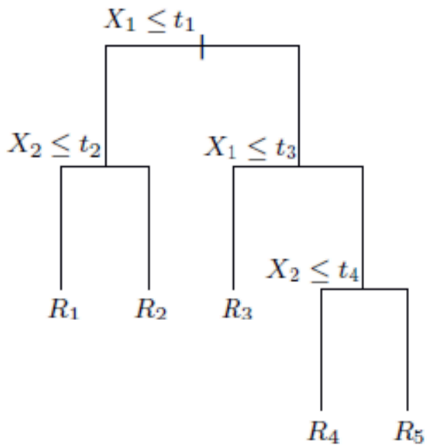


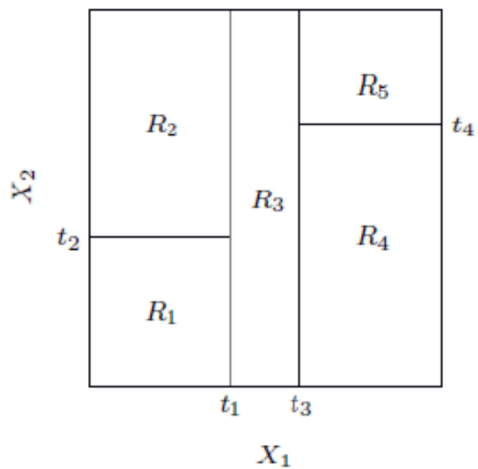
There could be more than one tree that fits the same data!

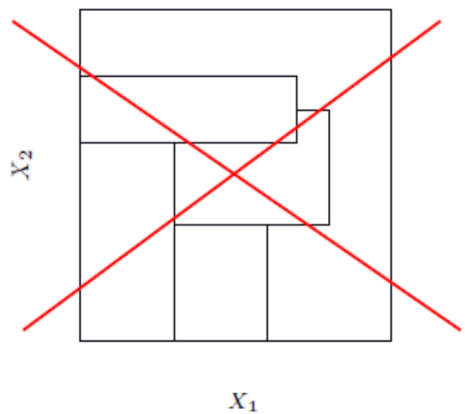
Source: Tan, Steinbach, Kumar

Trees are also called recursive partitioning

- There many approaches
  - C 4.5 / C5 Algorithms, SPRINT
- Here top down splitting until no further split possible (or other criteria)







Only straight lines are possible

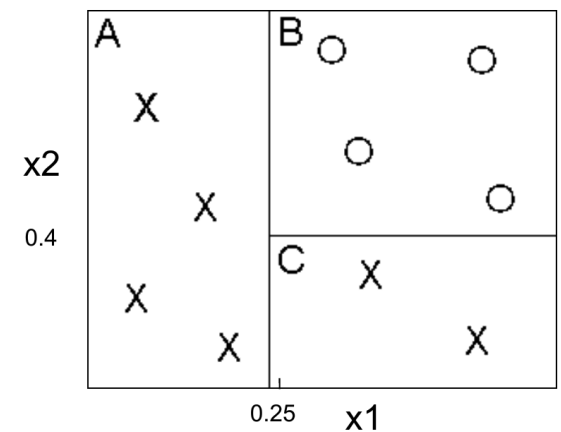
How to train a classification tree: your turn

- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score - Node impurity
(in a sec)

- Draw 2 splits to separate the data.
- Draw the corresponding tree



How to train a classification tree: [solution]

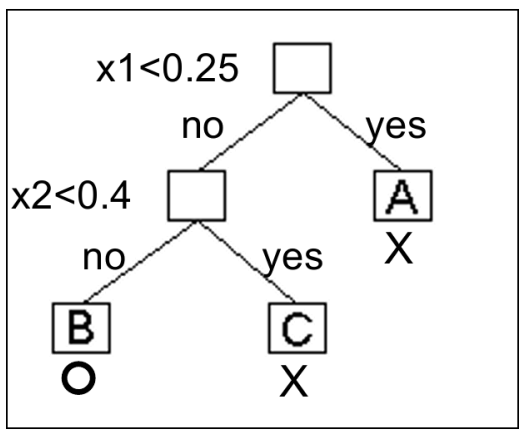
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

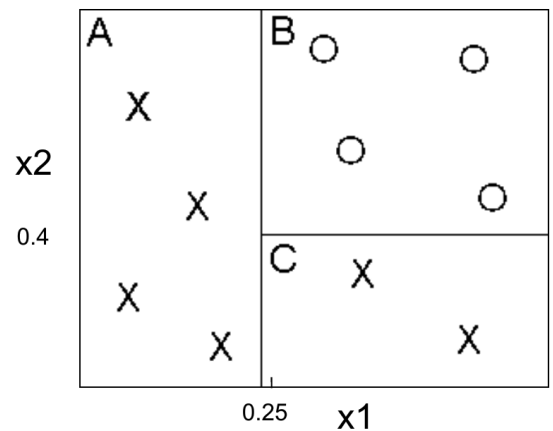
for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r

Score - Node impurity (in a sec)

Choose (x_j, s_j) giving maximum improvement to fit

Replace R with R^l; add R^r



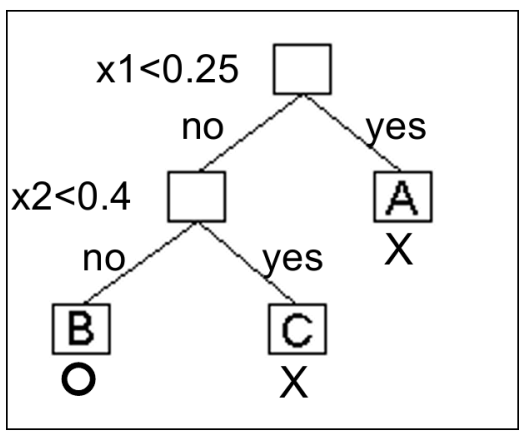


Putting it all together

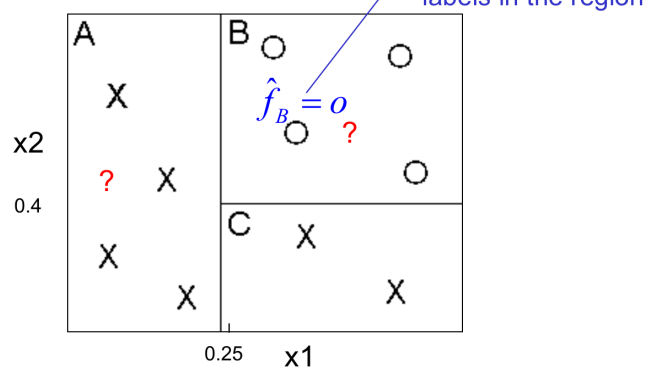
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r

Score - Node impurity (in a sec)

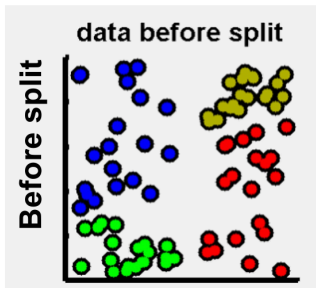


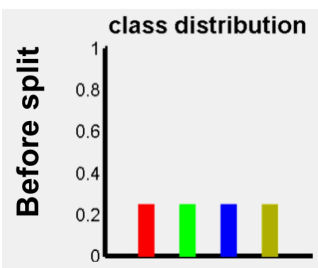
Model class label is given by the majority of all observation labels in the region

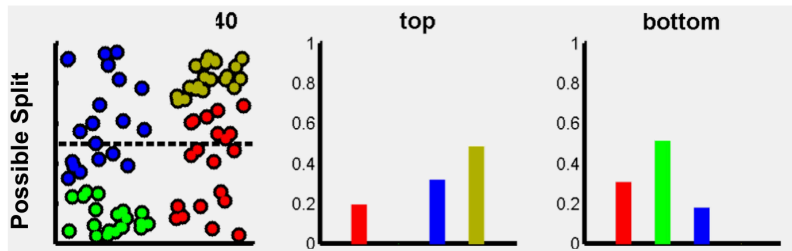


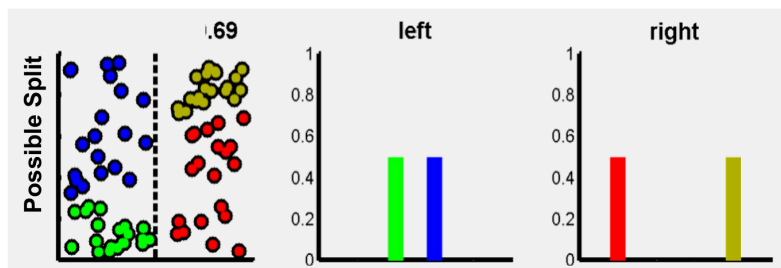
Construction of a classification tree: Minimize the impurity in each node

We have n_c=4 different classes: red, green, blue, olive









The most common impurity measures

p_i is the relative frequency of class $i = 1 \dots C$

Gini Impurity* (default rpart) $\text{Gini} = 1 - \sum_i^C p_i^2$

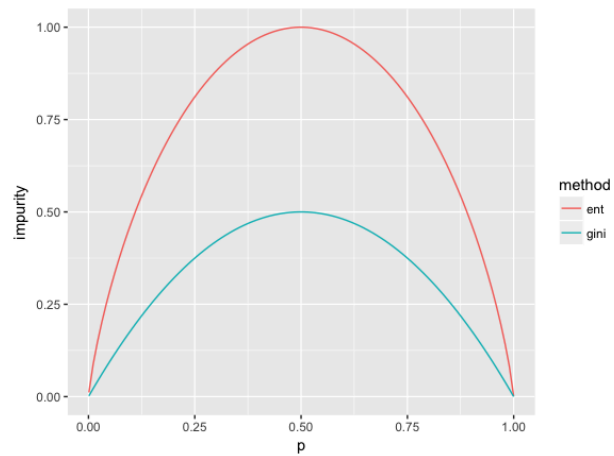
Entropy: $\text{Entropy} = -\sum_i^C p_i \log_2(p_i)$

For C=2 classes

p probability for the one class

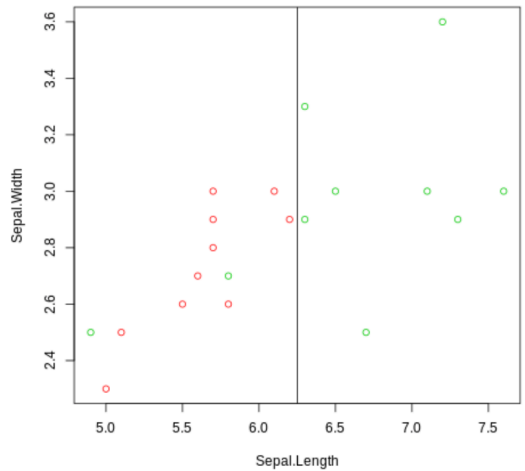
$1 - p$ for the other

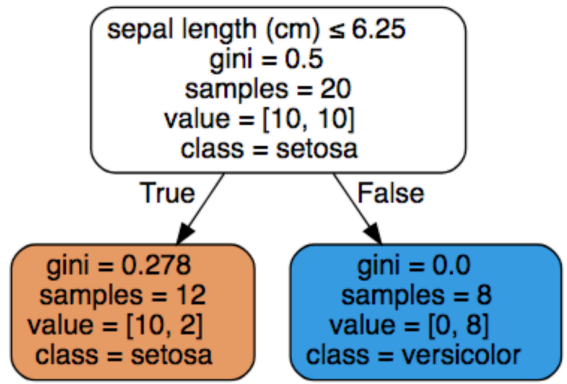
Gini = $1 - p^2 - (1-p)^2$



*Not to be confused with Gini coefficient

Example: One split by hand






```
giniroot = 0.5 # p = 0.5
giniright = 0  # p = 0.0
p = 2.0/12
gineleft = 1 - p**2 - (1-p)**2
gineleft
```

Out[19]:

```
0.2777777777777777
```

$$\frac{N_t}{N} \cdot (\text{giniroot} - \frac{N_{tr}}{N_t} \cdot \text{giniright} - \frac{N_{tL}}{N_t} \cdot \text{gineleft})$$

```
impurity_decrease =  1 * (giniroot - 12/20 * gineleft - 8/20.0 * giniright)
impurity_decrease
```

```
0.3333333333333337
```

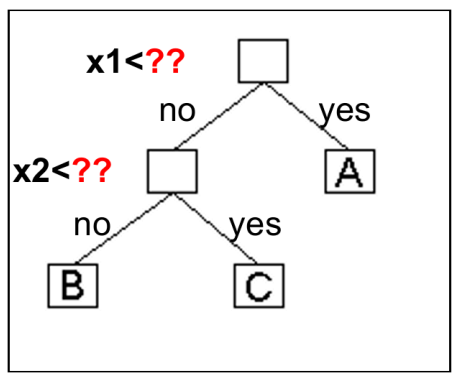
https://github.com/oduerr/ml-playground/blob/master/python/d_tree/Decision_tree.ipynb

Decision Trees for Regression

How to find the tree structure of a regression tree?

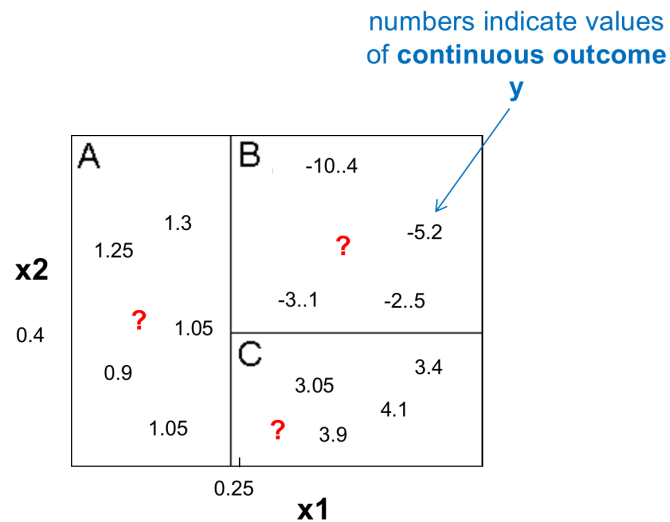
- Starting with a single region -- i.e., all given data
- At the m-th iteration:

for each region R
  for each attribute x_j in R
    for each possible split s_j of x_j
      record change in score when we partition R into R^l and R^r
Choose (x_j, s_j) giving maximum improvement to fit
Replace R with R^l; add R^r



Score:

- We assume to report a fixed number c for the region.
- Calculate MSE (mean squared error)



What to report in the regions?

Best value in region

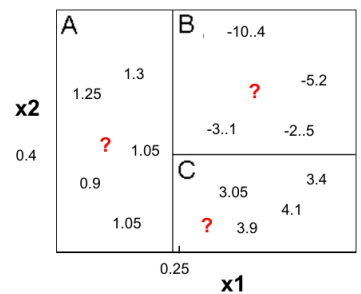
- Say we have the regions
  - What single number to report in a certain region?
- If loss is MSE and we have values y_i then mean reduces the loss
- Proof (optimal value c for MSE is mean)

$$\text{loss}(c) = \sum_{i=1}^{N} (y_i - c)^2$$

$$\frac{\partial \text{loss}(c)}{\partial c} = 0$$

$$\sum_{i=1}^{N} (-2) \cdot (y_i - c) = 0$$

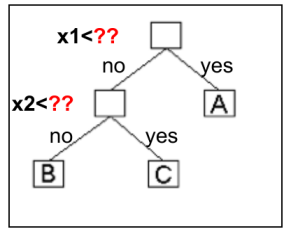
$$N \cdot c = \sum_{i=1}^{N} y_i \rightarrow c = \frac{1}{N} \sum_{i=1}^{N} y_i = \text{mean}(y_i)$$



Given a region, we now best single value for that region (it's the mean when MSE)

Best Boundary for MSE

for each attribute x_j *in* R
  for each possible split s_j *of* x_j



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

c_1 and c_2 are the mean values (see last slide)

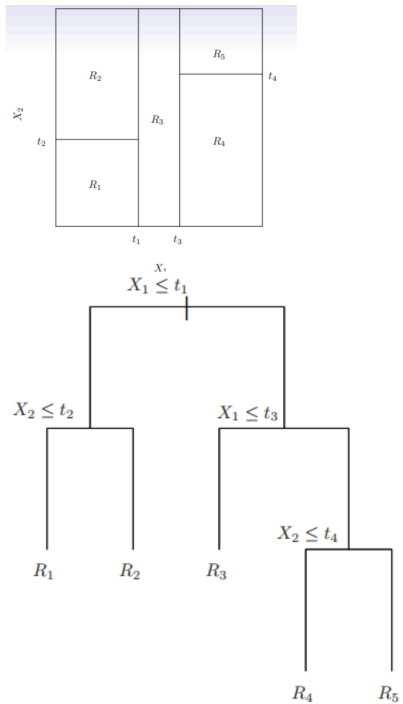
Regression Trees based on MSE:

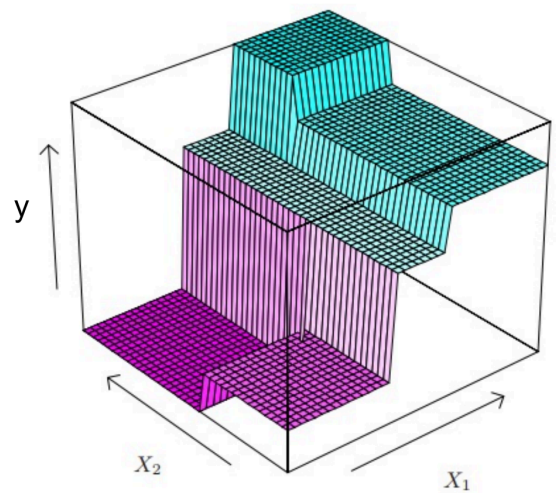
- Best Value for region → mean
- Best Split → mean squared error like

Other possible losses:

- Mean Absolute Error
- For Poisson (or other distributed) data use e.g. glmtree (in R)

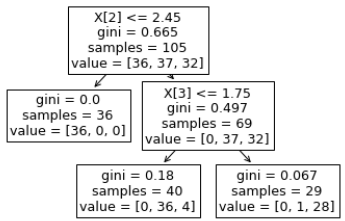
Trees do a recursive partitioning of the predictor space



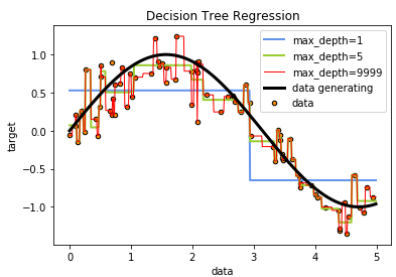


Demo

Classification



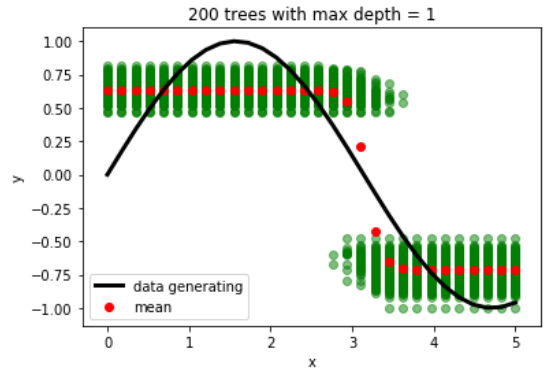
1-D regression (single simulation)

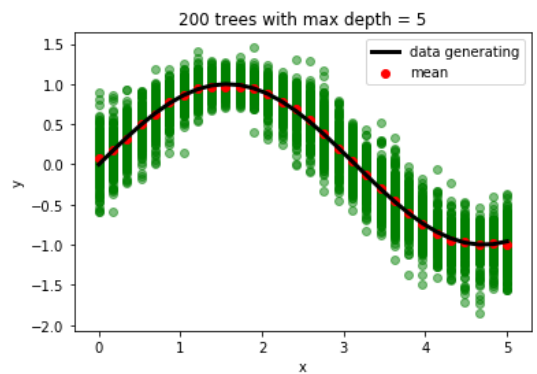


- DT with low max depth have high bias
- DT with large max depth have low bias but high variance

https://github.com/ioskn/mldl_htwg/blob/master/trees_and_rf/Decission_Tree.ipynb

1-D Regression (several simulations)





Pause bis 14:50

Pros and Cons of Decision Trees

Pros

- Easy to interpret.
- Can handle mixed discrete and continuous inputs.
  - Not available in scikit-learn, but available in R.
- Insensitive to monotone transformations of the inputs.
- Perform automatic variable selection.
- Robust to outliers.
- Scale to large data sets.
- Support multiclass and regression tasks.

Cons

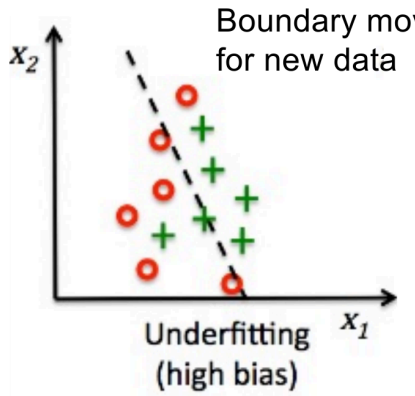
- Poor performance in some cases.
- Deep trees can be unstable.
  - High variance, which can be fixed.

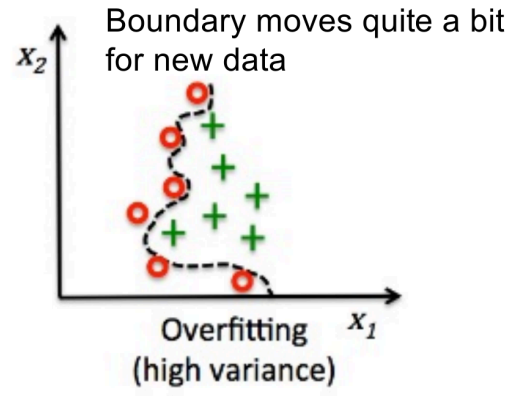
Pause bis 18:15

Ensemble Methods: Joining Forces



Recap: Bias Variance (Classification)

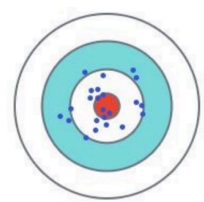




One blue dot is one classifier



One blue dot is one classifier



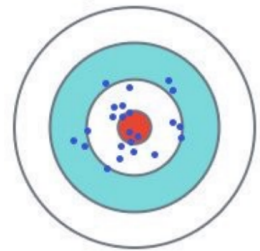
The concept of Bias and Variance in general

One blue dot is one classifier / regressor



An underfitting model

- is not flexible enough
- quite many errors on train data and **systematic test error (high bias)**
- **will not vary much if new train data is sampled from population (low variance)**

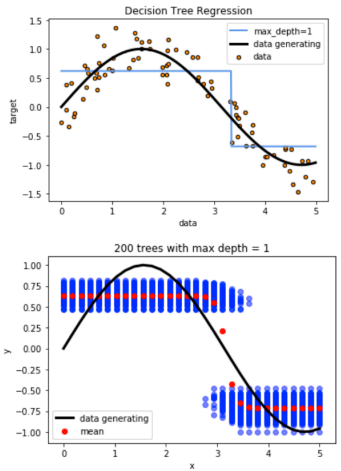


An overfitting model

- is too flexible for data structure
- few errors on train set large but non-systematic test errors **(low bias)**
- will vary a lot if fitted to new train data **(high variance)**

Recap: Bias Variance (Regression)

Stumps

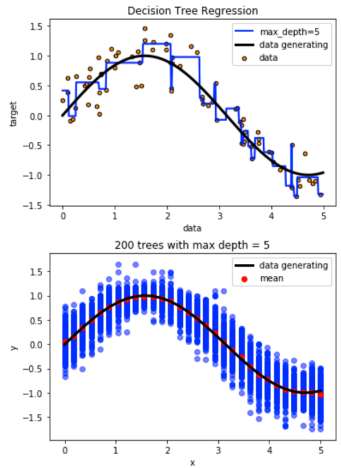


One blue dot is one tree

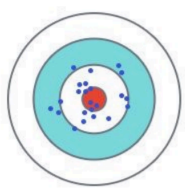


High Bias
Underfitting

Trees of depth 5



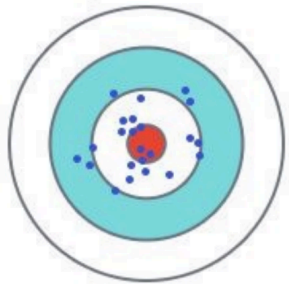
One blue dot is one tree



High Variance
Overfitting

Use ensemble methods to fight under and overfitting

High Variance



High Bias



fight the deficits of the single model by

Bagging

Adaptive boosting

improve ensemble approach further by

Random Forest
in case of tree models

Gradient boosting

Overview of Ensemble Methods

Many instances of the same classifier (**often trees!**)

- Bagging
  - Bagging (bootstrapping & aggregating)
    - Create “new” data using bootstrap
    - Train classifiers on new data
    - Average over all predictions (e.g. take majority vote)
  - Bagged Trees
    - Use bagging with decision trees
  - Random Forest
    - Bagged trees with special trick

- Boosting
  - (Ada) Boosting
    - An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.
  - Gradient Boosting
    - Fits trees to residuals determined by the gradient of the loss

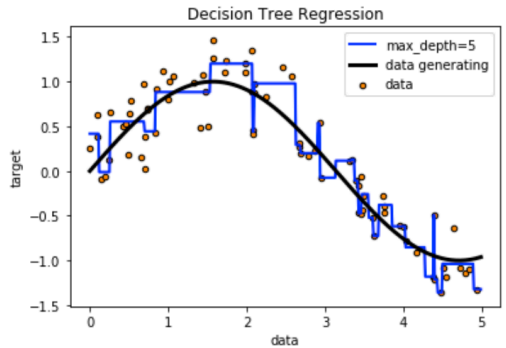
Combining classifiers (different classifiers)

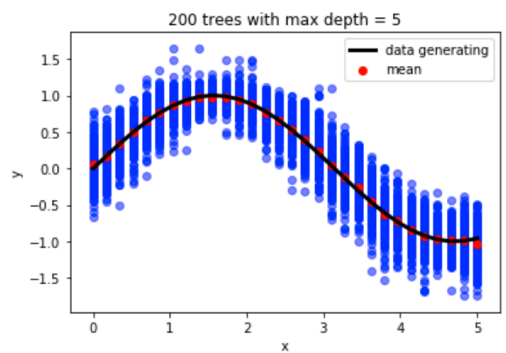
- Stacking classifiers
  - Use output of classifiers as input for a new classifier

Bagging
&
Random Forest

Recap: Bias Variance (Regression)

Trees of depth 5





Recipe for a perfect Regressor:

1. Draw a samples from the data generating process (DGP) and fit a deep tree to it.
2. Repeat 1 often and take the average of all trees.

Q: Is this procedure possible in practice?

Sorry Dave, we don’t have access to DGP

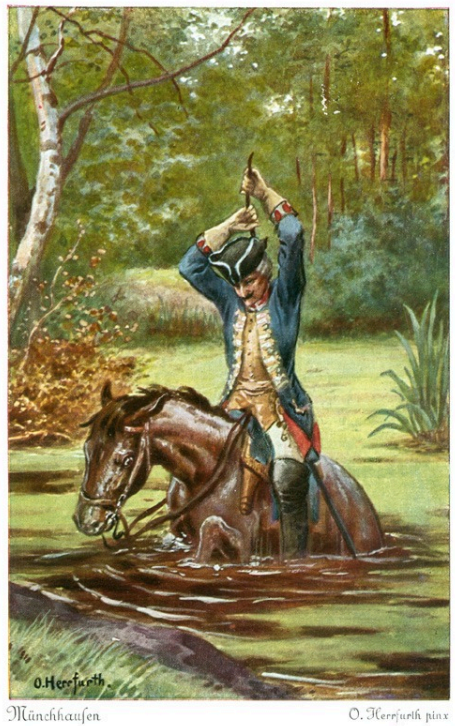
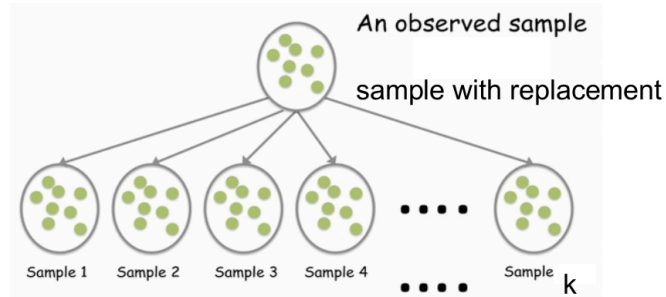


Image Credit: Wikipedia, https://github.com/founddrama/

- Create a Bootstrap Sample



- BTW Bootstrapping is widely used trick in stats to create confidence intervals by creating “virtual” samples.

Bagging as ensemble of parallel fitted models

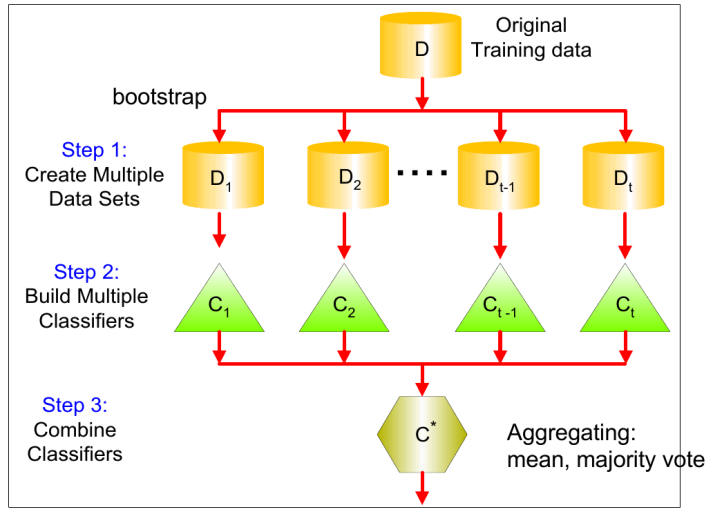
Each classifier tends to overfit its version of training data.

Bagging: bootstrapping and averaging

1) Fit flexible models on different bootstrap samples of train data

2) Minimize Bias by using flexible models and allow for overfitting

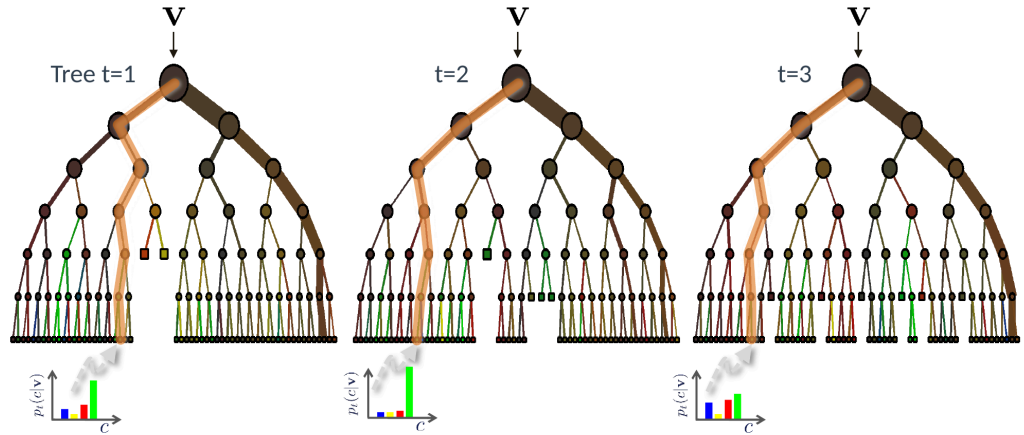
3) Reduce variance by averaging over many models



Remarks: highly non-linear estimators like trees benefit the most by bagging

If model does not overfit the data bagging does not help nor hurt.

How to classify a new observation v with boosted trees?

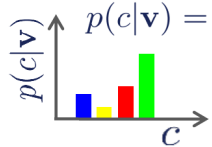


The to derive the ensemble result:

a) Majority vote:
Each tree has a “winner-class”
Take the class which was most often the winner

b) Average probabilities:

$$p(c|\mathbf{v}) = \frac{1}{T}\sum_{t}^{T} p_t(c|\mathbf{v})$$



Animation taken from microsoft

Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- Assume classifiers are independent (that's the most questionable assumption)
- Take **majority vote**
- Majority voter is wrong if 13 or more are wrong
- **Number of wrong predictors???**

Why does it work [Solution]

- Suppose there are 25 base classifiers
- Each classifier has error rate, ε = 0.35
- Assume classifiers are independent (that's the most questionable assumption)
- Take **majority vote**
- Majority voter is wrong if 13 or more are wrong
- Number of wrong predictors X ~ Bin(size=25, p_0=0.35)

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

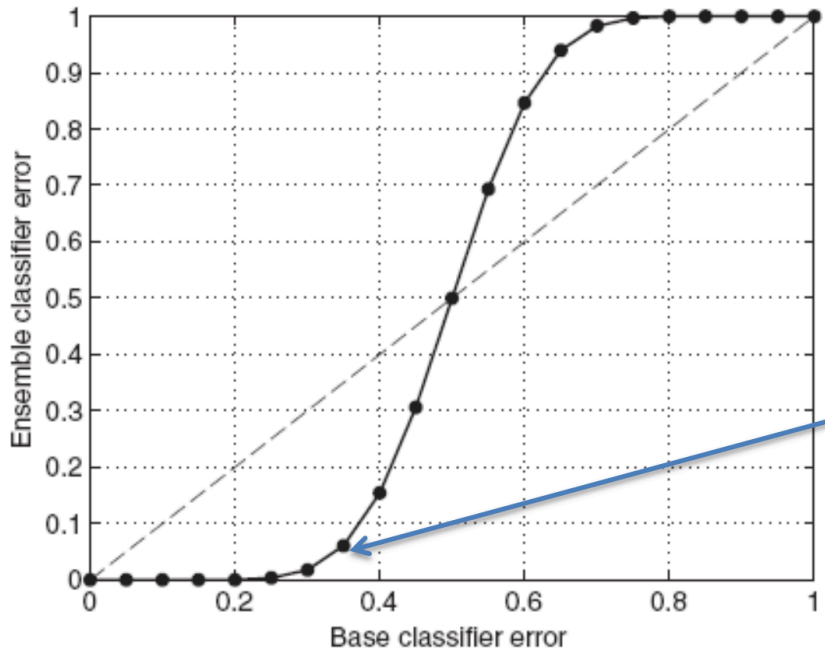
```
1 - scipy.stats.binom.cdf(12, 25, 0.35)
```

```
0.06044491356702042
```

> 1 - pbinom(12, size=25, prob = 0.35) or in python
[1] 0.06044491

Why does it work?

- 25 Base Classifiers

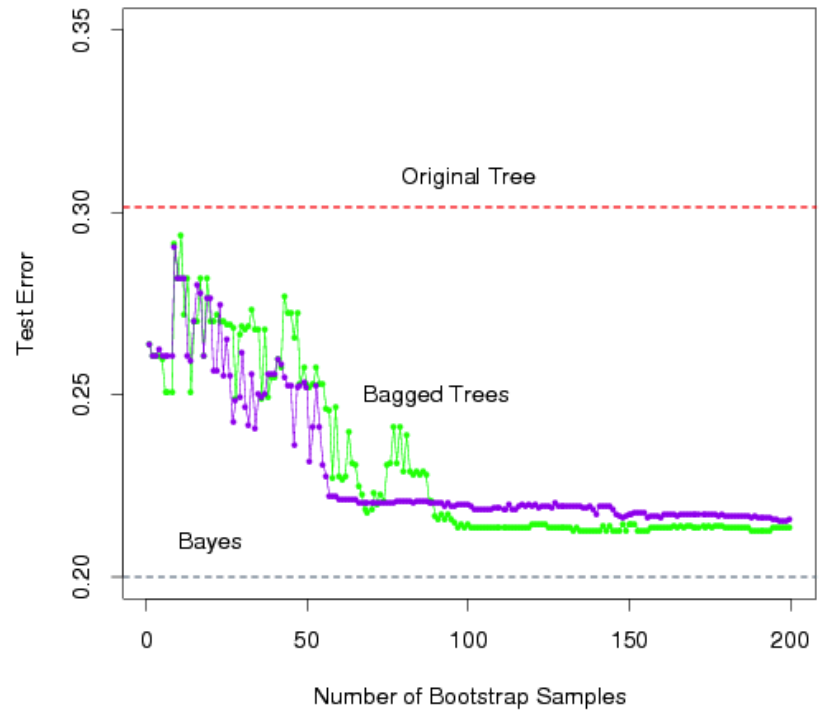


See last slide
1 - pbinom(12, size=25, prob = 0.35)

Ensembles are only better than one classifier, if each classifier is better than random guessing! Assuming independence between the classifiers.

A Comparison of Error Rates

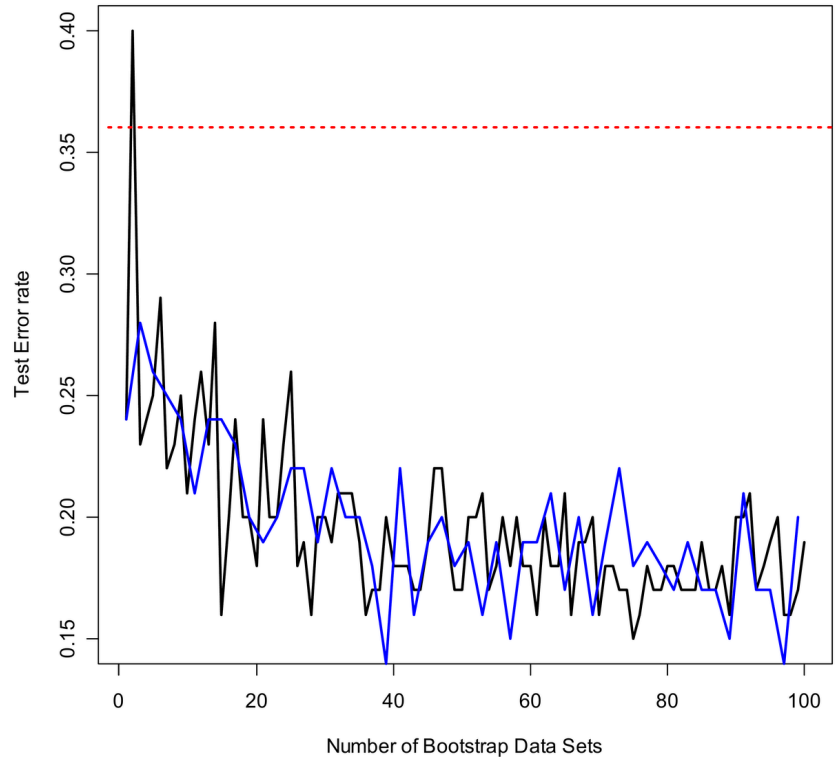
- Here the green line represents a simple majority vote approach
- The purple line corresponds to averaging the probability estimates.
- Both do far better than a single tree (dashed red) and get close to the Bayes error rate (dashed grey).



Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

Example 2: Car Seat Data

- The red line represents the test error rate using a single tree.
- The black line corresponds to the bagging error rate using majority vote while the blue line averages the probabilities.



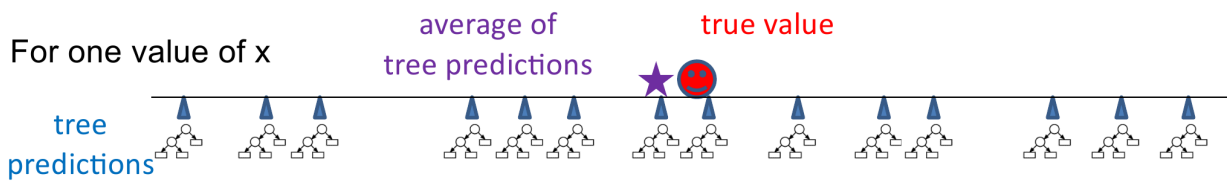
Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

Aggregate of regression trees?

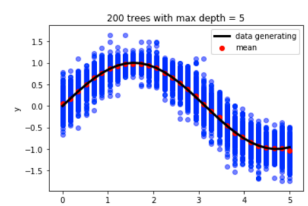
- Aggregation by taking the mean
- Suppose there are n=25 deep regression trees
  - All trees are deep
    - -> the trees have no or only small bias
    - -> the trees have high variance

Handwaving:

- Unbiased ➔ True value is reached on average (expectation)
- Central Limit Theorem ➔ variance is reduced by factor n (std by $\sqrt{n} = \sqrt{25} = 5$)

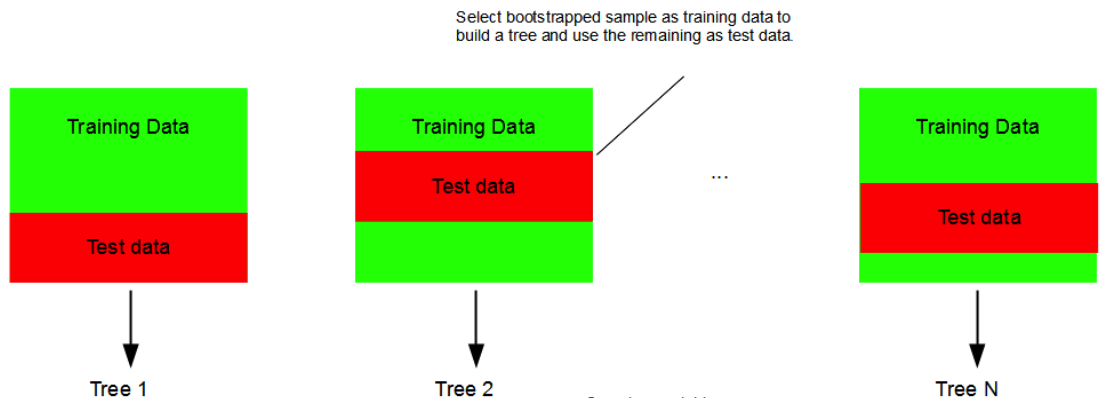


1-D Example
30 values of X



Out-of-Bag Estimation: “Cross-validation on the fly”

- Since bootstrapping involves random selection of subsets of observations to build a training data set, then the remaining non-selected part could be the testing data.
- On average, each bagged tree makes use of around 2/3 of the observations, so we end up having 1/3 of the observations used for testing



Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

The importance of independence

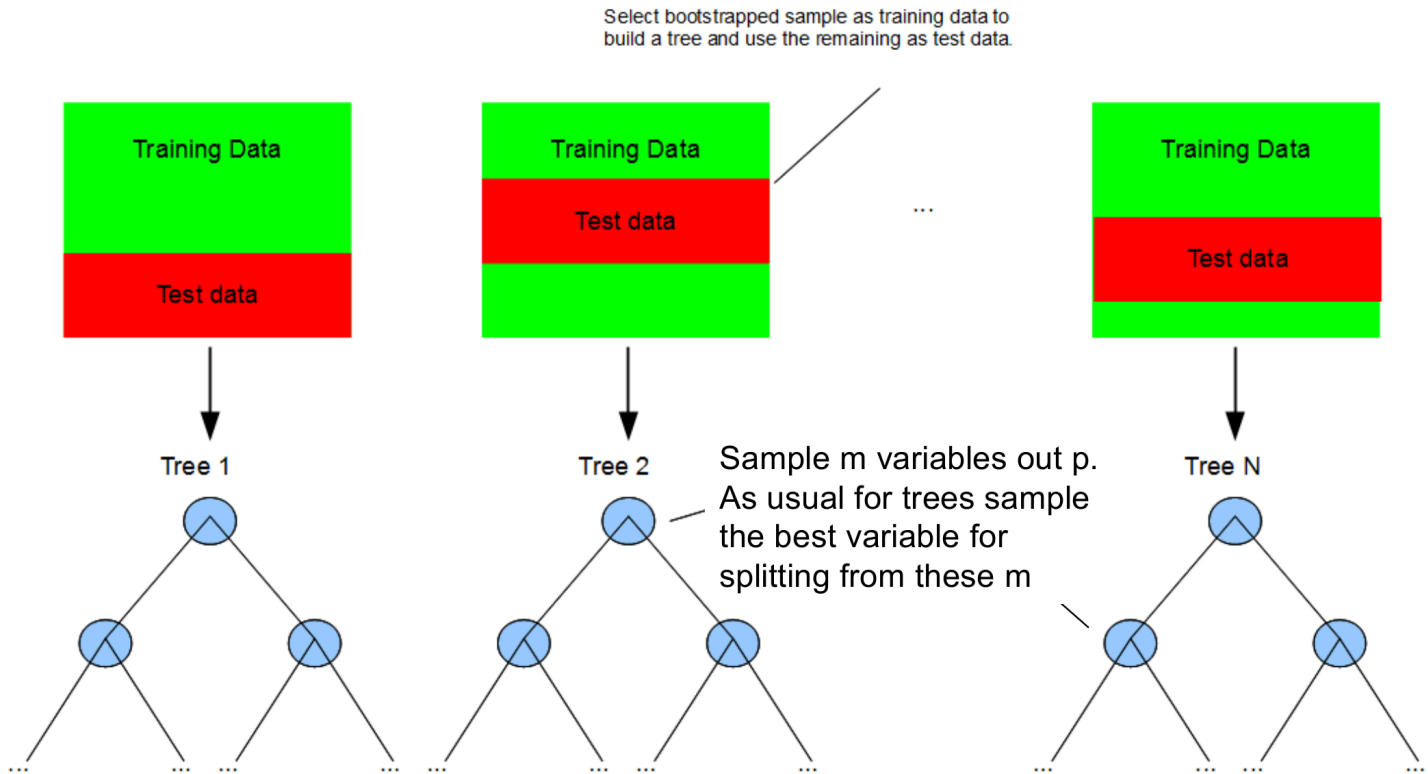
- It’s a quite strong assumption, that all classifiers are independent.
- Since the same features are used in each bag, the resulting classifiers are quite dependent.
  - **We don’t really get new (uncorrelated) classifiers or trees**
- **De-correlate by any means! ➔ Random Forest**
  - We are willing that the individual trees have worse performance
  - Our main goal is to decorrelate, we are quite brutal…

Random Forest

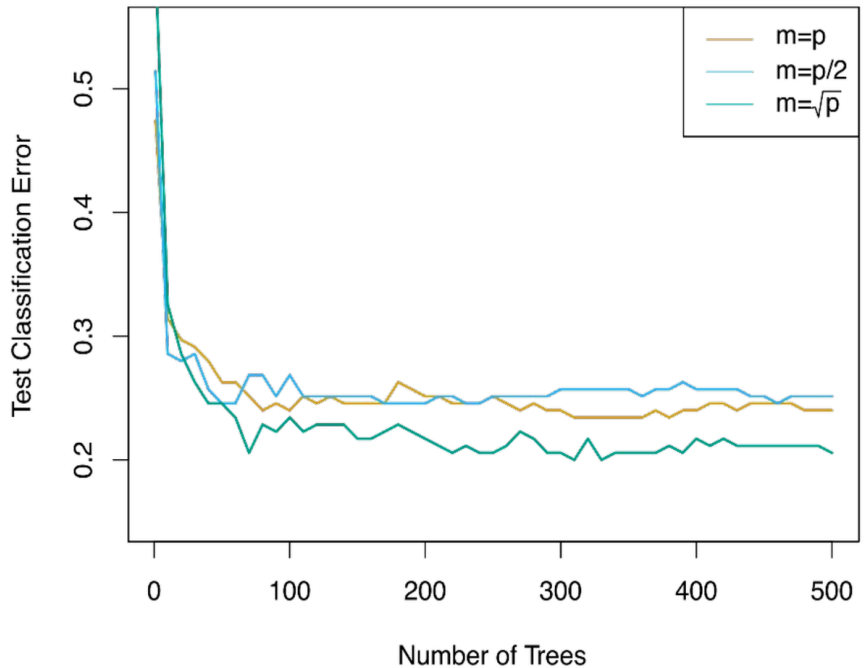
Random Forests

- Very efficient statistical learning method
- It builds on the idea of bagging, but it provides an improvement because it de-correlates the trees
  - Different trees are independent
- How does it work?
  - Build a number of decision trees on bootstrapped training sample, but when building these trees, each time a split in a tree is considered, **a random sample of *m* features** is chosen as split candidates from the full set of *p* features (usually $m \approx \sqrt{p}$ for classification)

Random Forest: Learning algorithm



Random Forest with different values of “m”



Random forests with
m = p is bagging.

Figure from ISLR

Random Forest in sklearn

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, oob_score=True)
rf.fit(X,y)
rf.oob_score_


# max_features='auto’#max_features corresponds to m
```

Variable Importance Measure

- Bagging (and RF) typically improves the accuracy over prediction using a single tree, but it is now hard to interpret the model!
- We have hundreds of trees, and it is no longer clear which variables are most important to the procedure
- Thus bagging improves prediction accuracy at the expense of interpretability
- But, we can still get an overall summary of the importance of each predictor using relative influence plots

Slide taken from Al Sharif: http://www.alsharif.info/#!iom530/c21o7

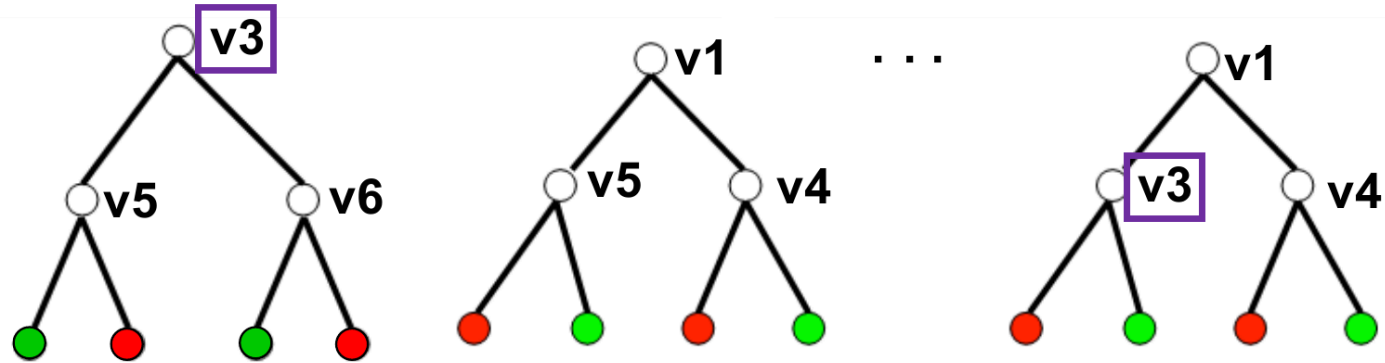
Variable Importance 1: performance-loss by permutation

Determine importance-2 for v4:

1) obtain standard OOB-performance
2) values of v4 are randomly permuted in OOB samples and OOB-performance is again computed. If variable is not important nothing happens. Hence:
3) Use decrease of performance as importance measure of v4

v1	v2	v3	v4	v5	v6	v7
0	1	1	2	0	1	0
0	2	2	1	2	0	1
1	0	0	1	1	2	0
1	0	0	1	1	0	2
0	2	1	0	2	0	1

Variable Importance 2: Score improvement at each Split



At each split where the variable, e.g. v3, is used the improvement of the score (Decrease of Giniindex) is measured. The average over all these v3-involving splits in all trees is the “MeanDecreaseGini” measure of v3.

Known Pros of Random Forest

Pros

- It is very easy to handle since based on Trees
  - No transformation needed
  - Can handle categorical features (and a mixture of categorical and numerical features)
  - Can handle missing values
- Random Forest does not tend to overfit.
- CV incorporated (Out of Bag)
  - Observations must be independent (Otherwise same observations in ”test and Train”
- Can handle lots of noise variable even with quite few relevant variables (6 out of 100 already works fine)
  - No variable selection needed (in general)
- Importance of variables can be quantified -> variable importance
- It can compute proximities between observations -> clustering data [not shown here]

Cons

- Quite accurate classifier, but usually not the best.
- **Ideal baseline**
- Hard to interpret

Let‘s praise the Random Forest

RF is not in all situations the best method - however it often just works!

The Random Forest™ is my shepherd; I shall not want.
He makes me watch the mean squared error decrease rapidly.
He leads me beside classification problems.
He restores my soul.
He leads me in paths of the power of ensembles
for his name’s sake.

Even though I walk through the valley of the curse of dimensionality,
I will fear no overfitting,
for you are with me;
your bootstrap and your randomness,
they comfort me.

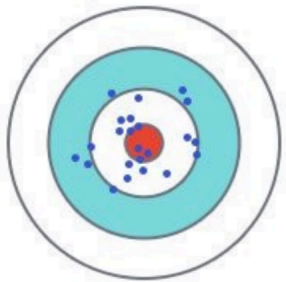
You prepare a prediction before me
in the presence of complex interactions;
you anoint me data scientist;
my wallet overflows.
Surely goodness of fit and money shall follow me
all the days of my life,
and I shall use Random Forests™
forever.

Christoph Molnarhttp://www.statsblogs.com/2014/02/06/random-forest-almighty/

Boosting 🚀

Use ensemble methods to fight under and overfitting

High Variance



fight the deficits of the single model by

Bagging

improve ensemble approach further by

Random Forest
in case of tree models

High Bias



Adaptive boosting

Gradient boosting
(fights under- and overfitting)

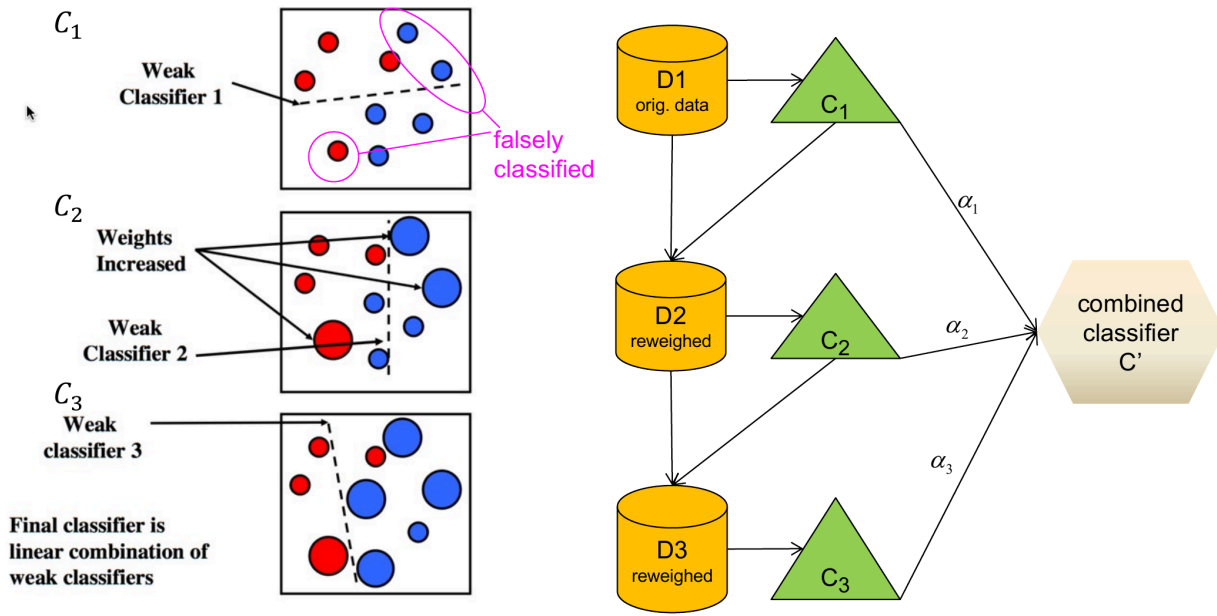
Historical View on boosting methods

There is nothing more practical than a good theory (Kurt Lewin)

- Adaptive Boosting (adaBoost) *Algorithm*
  - Freund & Schapire, 1990-1997
  - An *algorithmic* method for *two class {-1,1}* classification.
- Gradient Boosting (GB) Breiman (1999) Friedman/Hastie/Tibshirani (2000)
  - Boosting can be seen as a functional gradient descent of probabilistic models
  - Different likelihoods (Gaussian, Bernoulli, Poisson, Categorical)
- New powerful implementations/extensions (starting ~2014)
  - Extreme Gradient Boosting (xgboost - a particular implementation of GB)
    - Tianqi Chen (2014 code, 2016 published arXiv://1603.02754)
    - Often used in Kaggle Competitions as part of the winning solution
- Still active research, e.g.
  - "Transformation Boosting Machines" (T. Hothorn, 2019)
  - T. Duan, A. Avati, D.Y. Ding, S. Basu, A.Y. Ng, A. Schuler, "NGBoost: Natural Gradient Boosting for Probabilistic Prediction", 37th International Conference on Machine Learning, 2020.

Adaptive Boosting: Ingredient 1 upweighting

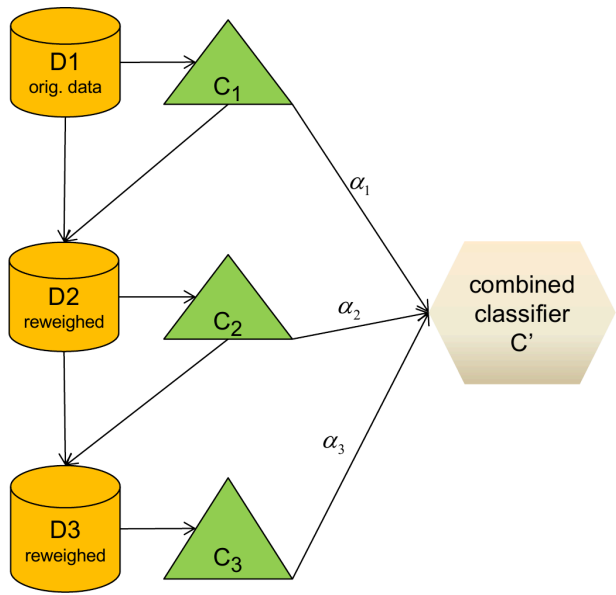
We use in each step a simple (underfitting) model to fit the current version of the data.
After each step misclassified observations get up-weighted.



left fig credits: http://vinsol.com/blog/2016/06/28/computer-vision-face-detection/

Adaptive Boosting: Ingredient 2 Weighting of individual classifiers

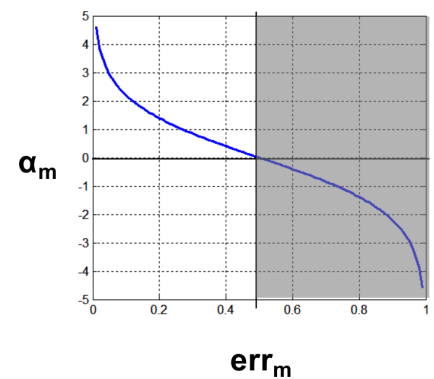
The final classifier C' is a weighted average of all sequential models C_m, where the model-weights α_m are given by the misclassification rate err_m of the model C_m



$$C'(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m \cdot C_m(x)\right]$$

$$\alpha_m = \log((1 - err_m)/err_m)$$

C_m error small ➔large weight α_m



Details of Ada Boost Algorithm

Algorithm:

1) Set $y_i \in \{-1,+1\}$ and start with identical weights $w_i = 1/n$

2) Repeat for $m = 1,2,...,M$:

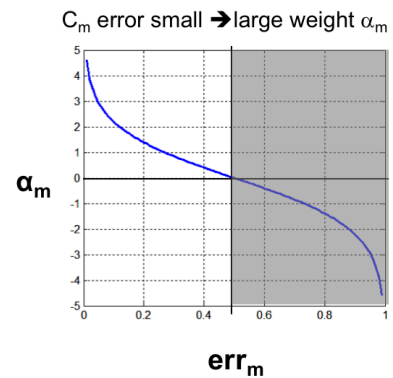
   a) Fit the classifier $f_m(x) \in \{-1,+1\}$ using weights w_i

   b) Compute the weighted error $err_m = \sum_i w_i \cdot I[y_i \neq f_m(x_i)]$

   c) Compute the aggregation weight $\alpha_m = \log((1-err_m)/err_m)$

   d) Set $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot I[y_i \neq f_m(x_i)])$; normalize to $\sum_i w_i = 1$

3) Output $F_M(x) = sign\sum_{m=1}^{M} \alpha_m f_m(x)$



Remark: There is no compelling reason for this algorithm, just as in linear regression, where minimizing the sum of squared deviations is reasonable but not required by a deeper idea. Many of the design decisions here are sensible, but none of them are driven by a deeper underlying idea.

Later, however, one can show that the algorithm corresponds to certain likelihoods when interpreted probabilistically.

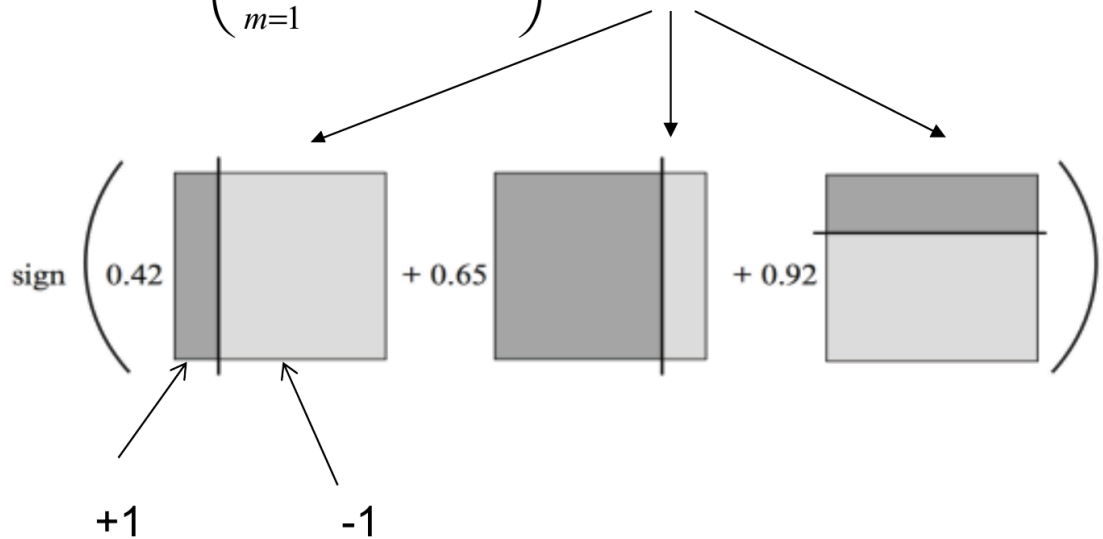
Ada Boost in simple words

- Fit an additive model $\sum_{m=1}^{M} \alpha_m \cdot C_m$ (ensemble) in a forward stage-wise manner.
- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners, from previous steps.
- In Adaboost,“shortcomings” are identified by high-weight data points.
- Weak learns often trees, sometimes only grown to depth 1 (stumps)

Example of such a boosted tree with M=3 stages…

Example (You Turn)

$$F_3(x) = \text{sign}\left(\sum_{m=1}^{3} \alpha_m f_m(x) \right)$$ ← 3 Stubs

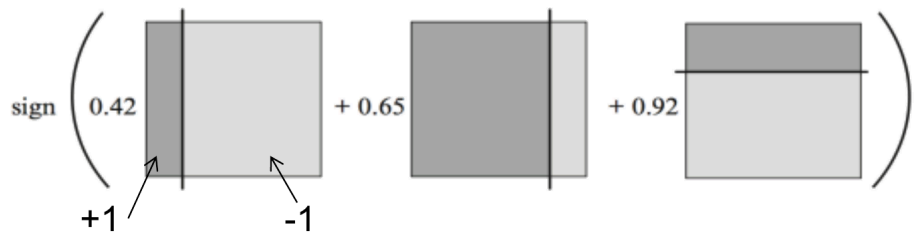


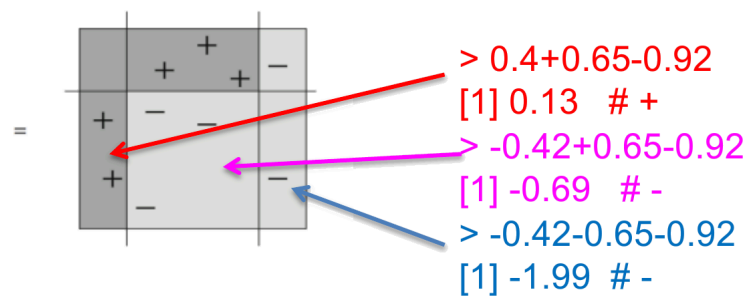
Ende der 1. Doppel Stunde WS 2025

Example (You Turn)

Lösung

$$F_3(x) = \text{sign}\left(\sum_{m=1}^{3} \alpha_m f_m(x) \right)$$





Performance of Boosting / Diagnostic Setting

Look at a specific case

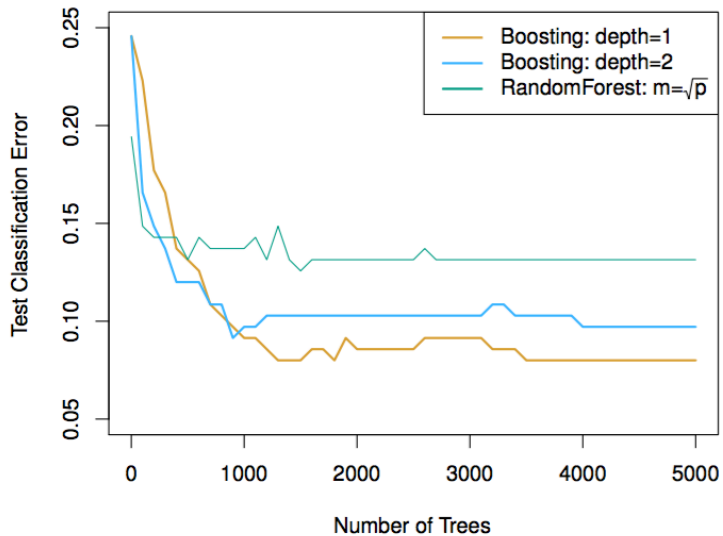


FIGURE 8.11 (ISLR) Results from performing boosting and random forests on the 15-class gene expression data set in order to predict cancer versus normal

Boosting most frequently used with trees (not necessary).
Trees are typically only grown to a certain depth – often 1 or 2.

Gradient Boosting

Content

- Start with Regression and MSE
  - Boosting is fitting a stage-wise additive model to the residuals
  - Overfitting: early stopping and learning rate
- Generalization
  - Generalization of residuals (negative gradients of a loss function)
- Different Loss Functions
  - Adopt MSE Boosting to
  - 2 Class Classification, Multi-Class
  - Non-MSE regressions
  - Generally all probabilistic models

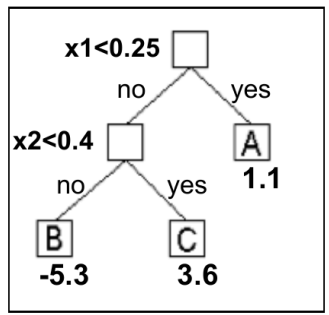
We follow the line of thought of “The elements of statistical learning” (ESL) chapter 10
From Hastie/Tibshirani the inventors of gradient boosting machines

Boosting fits an additive model

- Let's look at the decision function of Adaboost

$$F_M(x) = \text{sign} \underbrace{\sum_{m=1}^{M} \alpha_m f_m(x)}_{f(x)}$$

$f_m(x)$ for trees



- Can be seen as the sign of an additive model of basis fcts $f_m = b(x, \gamma_m)$

$$f(x) = \sum_{m=1}^{M} \underbrace{\beta_m b(x; \gamma_m)}_{f_m(x) \text{ as a function of } \gamma_m}$$

- Where β_m are expansion coefficients of (simple) basis functions $b(x)$ Functions ("base learners")
  - $b(x; \gamma_m) = \gamma_1 x + \gamma_2 x^2 + \cdots + \gamma_m x^m$ Taylor
  - Wavelets γ_m location and scale shifts of a mother wavelet
    - Example Viola&Jones Face Detection
  - …

Forward stagewise additive modeling

- Naïve: Minimizing a loss L (MSE) on the training data $\{y_i, x_i\}_{i=1,\ldots,N}$

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\Big(y_i, \underbrace{\sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)}_{f_M(x)}\Big)$$

← Too complex to fit all at once.

- Approximation: step by step (stagewise)

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute

$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, \overbrace{f_{m-1}(x_i)}^{\text{old model}} + \underbrace{\beta b(x_i;\gamma)}_{\text{“delta model”, weak learner (} \beta,\gamma \text{ are optimized)}}).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \underbrace{\beta_m b(x;\gamma_m)}_{\text{“delta model”}}$.

For MSE: fitting the delta model is fitting to residuals

We use MSE as a loss $L = \sum(y_i - f_m(x_i)^2)$

$$
\begin{aligned}
(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L\Big(y_i, \overbrace{f_{m-1}(x_i)}^{\text{old model}} + \beta b(x_i;\gamma)\Big) \\
(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} \Big(\underbrace{y_i - f_{m-1}(x_i)}_{\text{Residual } r_i} - \beta b(x_i;\gamma)\Big)^2 \\
(\beta_m, \gamma_m) &= \arg\min_{\beta,\gamma} \sum_{i=1}^{N} \Big(r_i - \beta b(x_i;\gamma)\Big)^2 \qquad (1)
\end{aligned}
$$

This corresponds to fitting the "delta" model / weak learner to the residuals r_i between the observed values and the current model.

Example: Least Squares Regression

$$L(y, f(x)) = (y - f(x))^2$$

- After $m-1$ steps, suppose we have the model $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x; \gamma_j)$. ← sum of m-1 trees*
- At the mth step we solve

$$\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2$$

← Fitting a new tree m

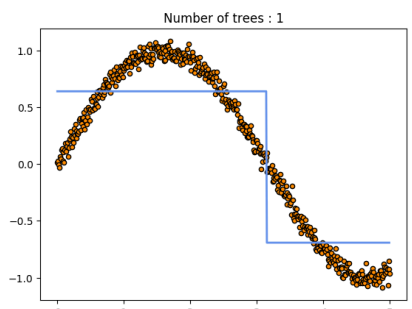
residual!

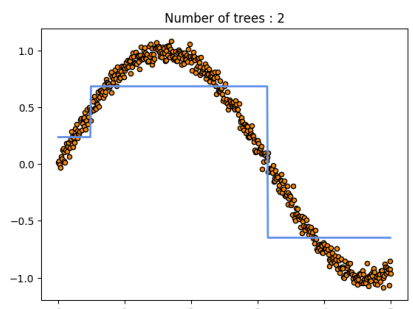
"delta model" will be fitted to the residual

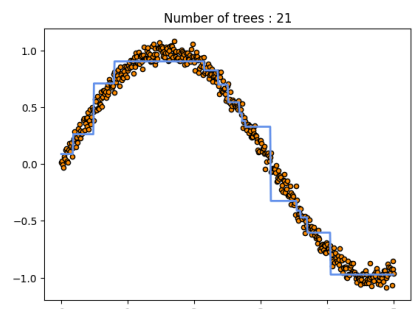
First step ($f_o(x) = 0$), we fit tree to data minimizing $(y_i - \beta b(x_i; \gamma))^2$
Then:
Instead of fitting a tree to the data ($f_o(x) = 0$), we fit a tree to the residuals $(y_i - f_{m-1}(x_i))^2$ of the previous trees.

Demo-Time: Least Square Regression

```python
fx = np.zeros(num_data) #See Step 1 in Algo 10.2
for m in range(9):
    tree = DecisionTreeRegressor(max_depth=1)
    res = y - fx #Calculation of the rediduals
    tree.fit(X, res) #Fitting to the residuals (Steps 2a)
    fx = fx + tree.predict(X) #The new predictions (Step 2b)
    plot_current_f(X,y,fx,m+1)
    plt.show()
```



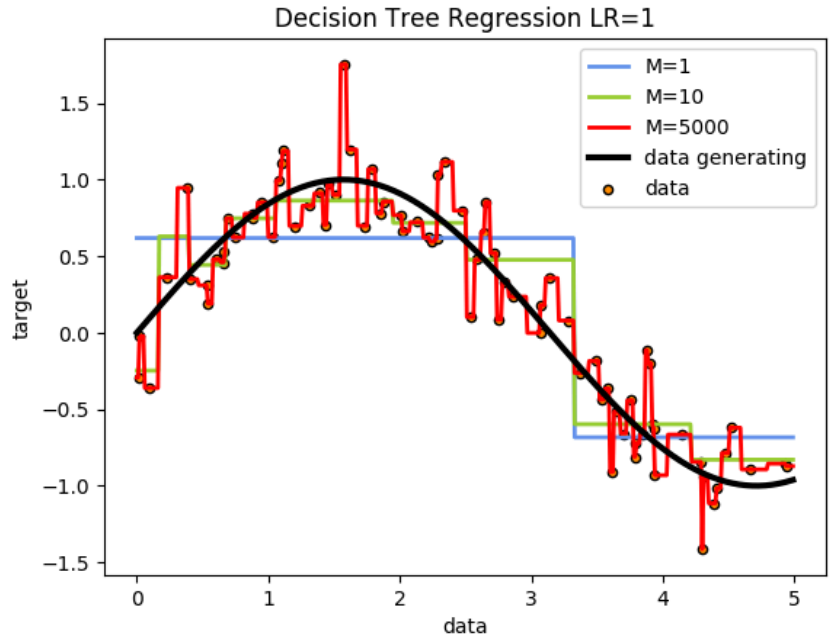




https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Pause bis 18:15

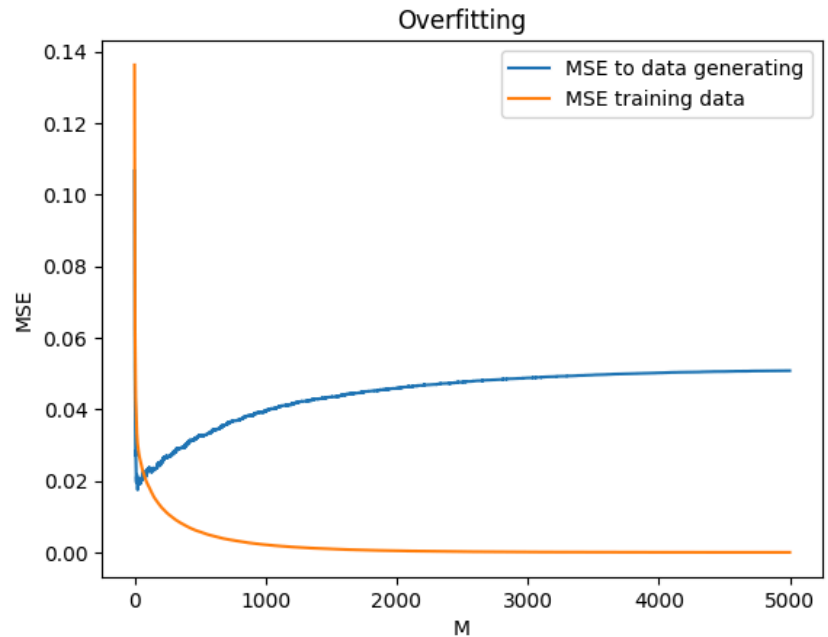
More examples



Quite greedy! Shows overfitting if too many trees are used.

https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Overfitting: Early Stopping



Use validation set to find the optimal number of trees M (*"early stopping"*)

https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Learning rate

- Scale the contribution of the new tree, for regularization

```
fx = np.zeros(num_data) #See Step 1 in Algo 10.2
for m in range(1000):
    tree = DecisionTreeRegressor(max_depth=1)
    res = y - fx #Calculation of the rediduals
    tree.fit(X, res) #Fitting to the residuals (Steps 2a)
    fx = fx + LR * tree.predict(X) #The new predictions (Step 2b) with LR
```

↑ LR a new hyper parameter a.k.a. `learning_rate`, η, ν

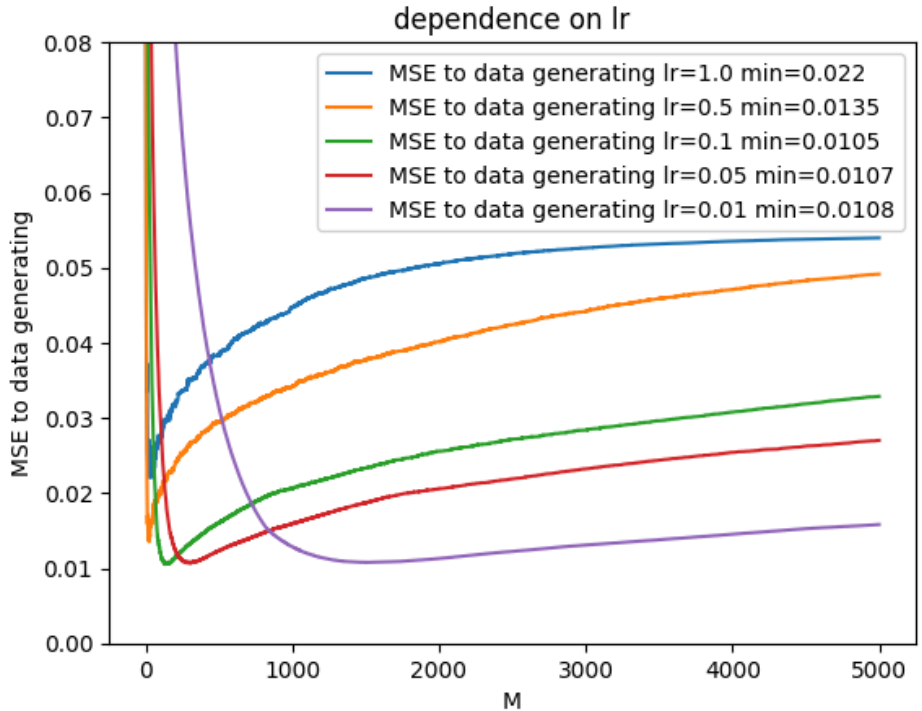
sklearn implementation

↓

```
from sklearn.ensemble import GradientBoostingRegressor
est = GradientBoostingRegressor(n_estimators=1000, learning_rate=LR,
                                max_depth=1, random_state=0, loss='ls')
```

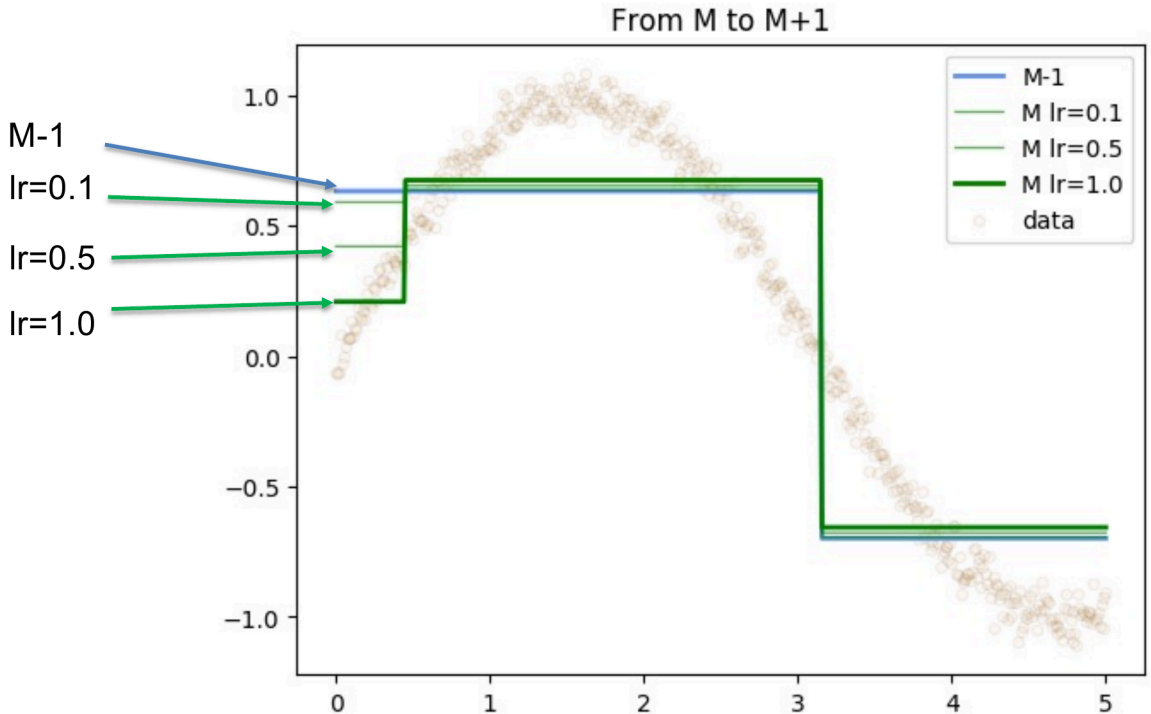
https://github.com/ioskn/mldl_htwg/blob/master/boosting/Least_Square_Boosting.ipynb

Effect of learning rate



Both learning rate and early stopping needs to be tuned for optimal result

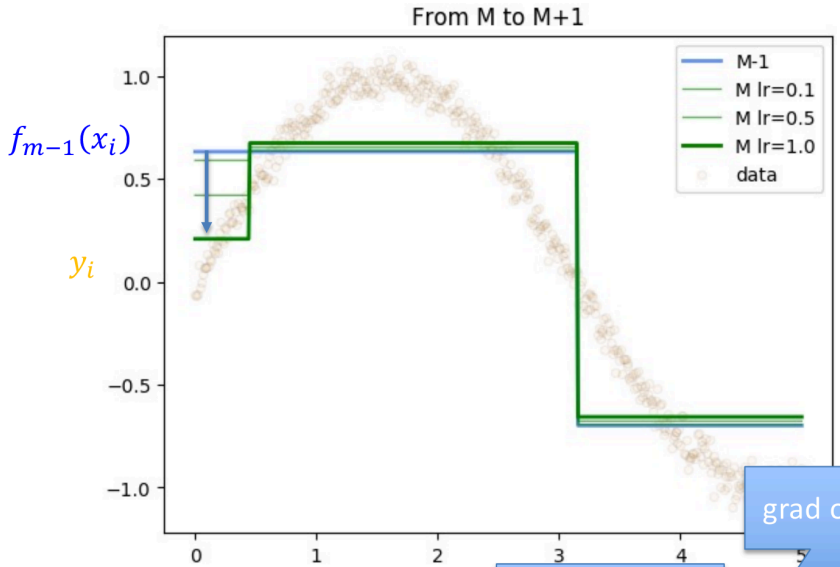
Learning rate as partial step to best solution



We move in the direction of the optimal fit (lr=1.0). But, we do not go the full step.

The gradient in gradient boosting

Connection between gradient and residuals



grad of MSE(fancy written)

grad of MSE

Residuals

$$\underbrace{f_{m-1}(x_i)}_{w_i} - y_i = w_i - y_i = \frac{\partial}{\partial w_i}\frac{1}{2}(w_i - y_i)^2 = -\frac{\partial}{\partial f_{m-1}(x_i)} L_{MSE}(y_i, f_{m-1}(x_i))$$

w_i (just a number for region)

With a squared loss: The residuals are the negative gradients of the loss.

Gradient Boosting (Cookbook)

- In gradient boosting residuals are replaced by the gradients of the loss
  - For MSE this leads to the residuals
  - For other loss functions *generalized residuals* are created. These point to the direction in with to move to improve the loss.

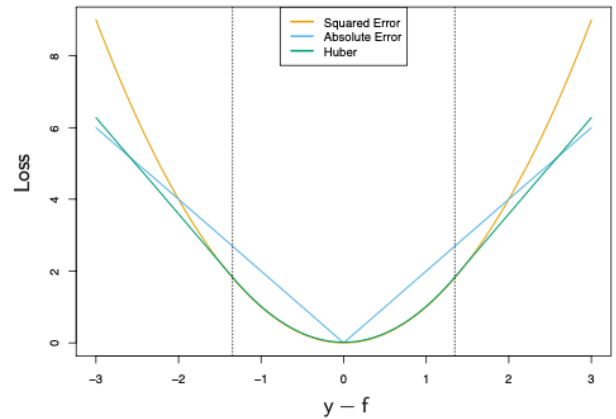
$$r_i = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

- A tree is fitted to the generalized residuals of the trees of stage *m-1*
  - *Note: this is in contrast to neural networks and most other models in which the gradients are propagated into the model*

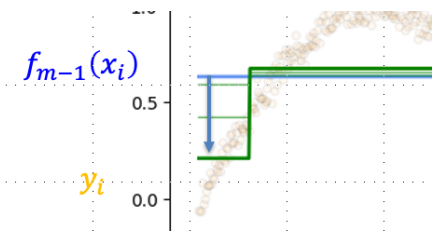
$$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} \big(r_i - \beta b(x_i; \gamma)\big)^2$$

Heuristic loss functions for regression

- MSE
  - L = $\frac{1}{N}\sum\left(y_i - f(x_i)\right)^2$
- MAE
  - More robust against outliers
  - L = $\frac{1}{N}\sum|y_i - f(x_i)|$
- Huber Loss interpolates



$$L(y, f(x)) = \begin{cases} [y - f(x)]^2 & \text{for } |y - f(x)| \le \delta, \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise.} \end{cases}$$



$$-\frac{\partial}{\partial f_{m-1}(x_i)} L_{MSE}(y_i, f_{m-1}(x_i))$$

Adaboost

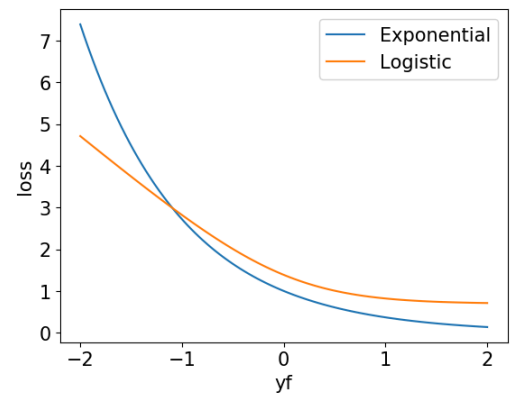
- Adaboost

$$F_M(x) = \text{sign} \sum_{m=1}^{M} \alpha_m f_m(x)$$

Corresponds to the exponential loss function (per training example *i*)

  - L = $e^{-f(x_i) \cdot y_i}$ remember $y_i \in \{-1, +1\}$ coding
  - See ELS 10.5

- Also shown is logistic



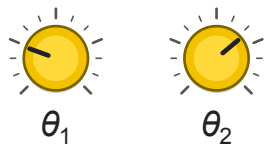
Pause bis 12:15 WS 2025

Probabilistic Predictions

Recap: Probabilistic Modelling

- We model a random variable $Y_{|x}$

$x \rightarrow$ model parametrized with parameters $\theta \rightarrow$ parameters of distribution

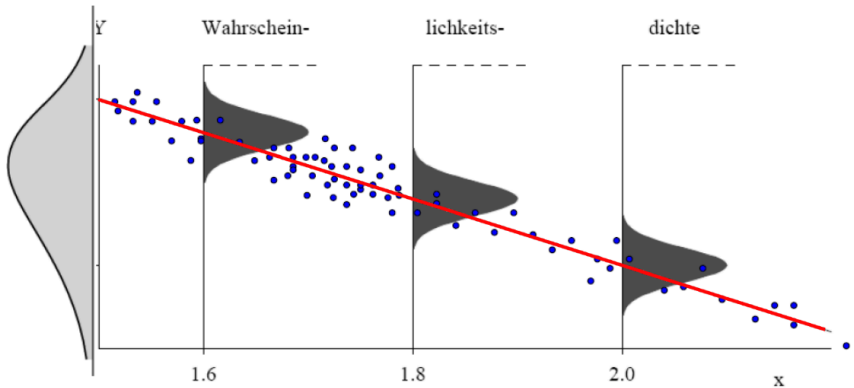


$$p_\theta(y|x) = \text{pdf}(\text{para} = f(x))$$

Examples of probabilistic models:

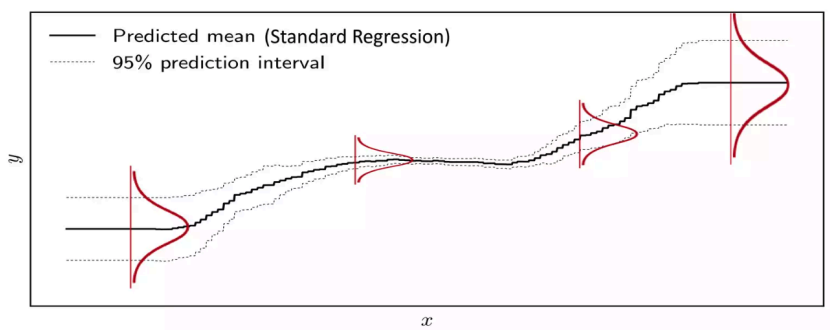
- Linear regression
  - $p_\theta(y|x) = \text{N}(y; \mu = w \cdot x, \sigma = const.)$
- Logistic regression
  - $p_\theta(y|x) = \text{Bern}(y; p = \text{sigmoid}(w \cdot x))$
- LLM
  - $p_\theta(y|x) = \text{Categorical}(\text{NN}_\theta(\text{x}))$

Linear Regression



$\mu = a \cdot x + b$ and $\sigma = \text{const}$

Mean is linear
Variance is constant



Mean is complex
Variance is also not fixed

NGBoost package

Predicting more than one parameter

- Boosting
  - $f_M(x) = \sum_{i=1}^{M} \gamma_m f_m(x)$
- Can be extended to the multivariate case
  - $f_{Mj}(x) = \sum_{m=1}^{M} \gamma_{mj} f_{mj}(x)$ j $= 1, \ldots p$ p Number of parameters
- The functions $f_{Mj}(x)$ control the parameters of a probability distribution
- Probabilistic regression
  - Output is Gaussian. For linear regression
    - $\mu = a \cdot x + b$ and $\sigma = \text{const}$
  - Boosting, we control mean and sigma
    - $p_\theta(y|x) = \mathrm{N}(y; \mu = f_{M1}(x), \sigma = \exp(f_{M2}(x)))$
    - Note that the functions $f_{M.}$ are unrestricted
    - To make them positive use exponential

Loss Function (Regression Like Problems)

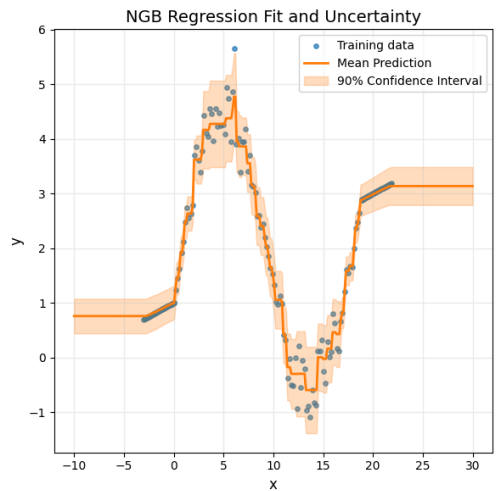
- For Boosting, we need a loss function
  - $L\big(y_i, f_{Mj}(x_i)\big)$
- Simply calculate the NLL
  - $L\big(y_i, f_{Mj}(x_i)\big) = -\log(p_\theta(y_i|x_i))$
- Example NGBoost-Library (loss functions)
  - Bernoulli
  - Cauchy
  - Exponential
  - Normal
  - NormalFixedMean
  - NormalFixedVar
  - Multivariate Normal
  - …

Example ngboost

```
from ngboost import NGBRegressor
from ngboost.distns import Normal, NormalFixedVar
ngb = NGBRegressor(Dist=Normal).fit(x_train, y_train.flatten(),
    X_val=x_val, Y_val=y_val.flatten(), early_stopping_rounds=2)
```

✓ 0.8s

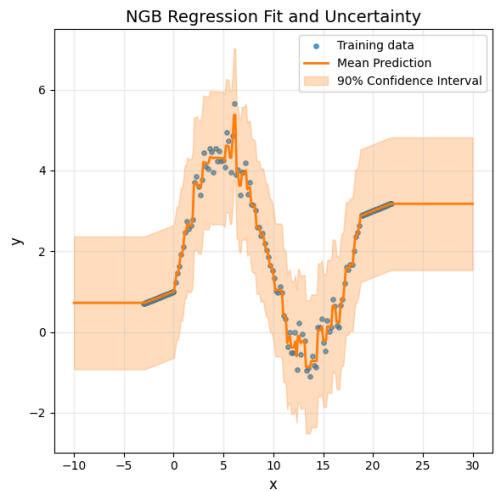
```
[iter 0] loss=1.7871 val_loss=1.9602 scale=1.0000 norm=1.3748
[iter 100] loss=0.8822 val_loss=1.0353 scale=2.0000 norm=1.1738
[iter 200] loss=-0.0348 val_loss=0.3559 scale=2.0000 norm=0.9988
== Early stopping achieved.
== Best iteration / VAL245 (val_loss=0.2414)
```




```
from ngboost import NGBRegressor
from ngboost.distns import Normal, NormalFixedVar
ngb = NGBRegressor(Dist=NormalFixedVar).fit(x_train, y_train.flatten())
```

✓ 1.2s

```
[iter 0] loss=1.9631 val_loss=0.0000 scale=2.0000 norm=2.5937
[iter 100] loss=0.9771 val_loss=0.0000 scale=2.0000 norm=0.4674
[iter 200] loss=0.9324 val_loss=0.0000 scale=2.0000 norm=0.1722
[iter 300] loss=0.9268 val_loss=0.0000 scale=2.0000 norm=0.1234
[iter 400] loss=0.9243 val_loss=0.0000 scale=2.0000 norm=0.1014
```



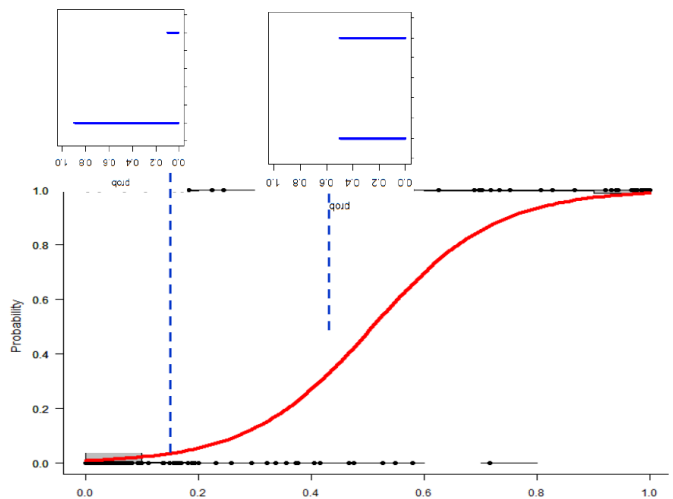
https://github.com/ioskn/mldl_htwg/blob/master/boosting/ProbBoosting.ipynb

Classification (2 Class)

Logistic Regression

$par = a \cdot x + b$

$p_\theta\left(y|x\right) = Bern(y; p = sigmoid(par))$



Boosting (just replace parameter)

$par = f_{M1}\left(x\right)$

$p_\theta\left(y|x\right) = Bern(y; p = sigmoid(par))$

Example ngboost

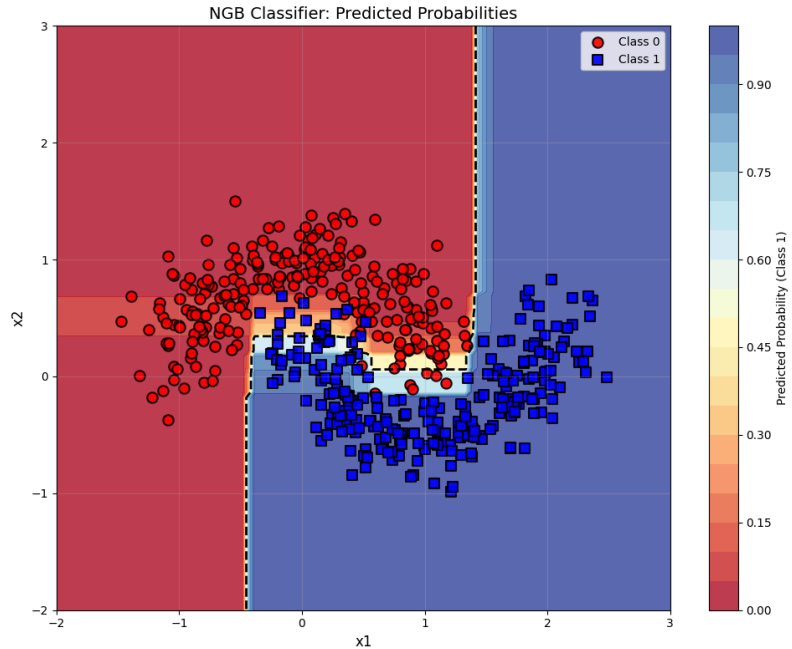
```
### Fitting a NGBClassifier

from ngboost import NGBClassifier
from ngboost.distns import Bernoulli

ngb_clf = NGBClassifier(Dist=Bernoulli, n_estimators=1000, learning_rate=0.01)
ngb_clf.fit(x_train, y_train)
```

✓ 1.3s

```
[iter 0] loss=0.6931 val_loss=0.0000 scale=4.0000 norm=8.0000
[iter 100] loss=0.1366 val_loss=0.0000 scale=2.0000 norm=2.4314
[iter 200] loss=0.1101 val_loss=0.0000 scale=1.0000 norm=1.1805
[iter 300] loss=0.1058 val_loss=0.0000 scale=0.0001 norm=0.0001
[iter 400] loss=0.1058 val_loss=0.0000 scale=0.0001 norm=0.0001
[iter 500] loss=0.1058 val_loss=0.0000 scale=0.0001 norm=0.0001
[iter 600] loss=0.1058 val_loss=0.0000 scale=0.0001 norm=0.0001
```



https://github.com/ioskn/mldl_htwg/blob/master/boosting/ProbBoosting.ipynb

Classification (more then two classes)

- Categorical distribution
  - $\overrightarrow{p(y|x)} = \text{categorical}(p_1, p_2, \ldots, p_k)$ a vector with the probabilities for the classes $1, \ldots, k$
  - The parameter p_j is probability for j-th class
- Linking the p_j to the boosting functions $f_{Mj}(x)$
  - $f_{Mj}(x) \in \mathbb{R}$
  - But p_j are probabilities > 0 and must sum up to 1
  - Solution “softmax” (see deep learning)
    - $p_j = \frac{\exp(f_{Mj}(x))}{\sum \exp(f_{Mj}(x))}$

```
# Fit a NGBClassifier with validation set
# We use only 2 trees (otherwise performance is too good)
ngb_clf = NGBClassifier(Dist=distns.k_categorical(3), n_estim
    x_train, y_train
)
np.round(ngb_clf.predict_proba(x_val[0:5]),3)
```

✓ 0.0s

```
[iter 0] loss=1.0986 val_loss=0.0000 scale=32.0000 norm=109.2548
```

← Probabilities for the classes

```
array([[0.697, 0.152, 0.152],
       [0.153, 0.153, 0.694],
       [0.152, 0.697, 0.152],
       [0.152, 0.697, 0.152],
       [0.697, 0.152, 0.152]])
```

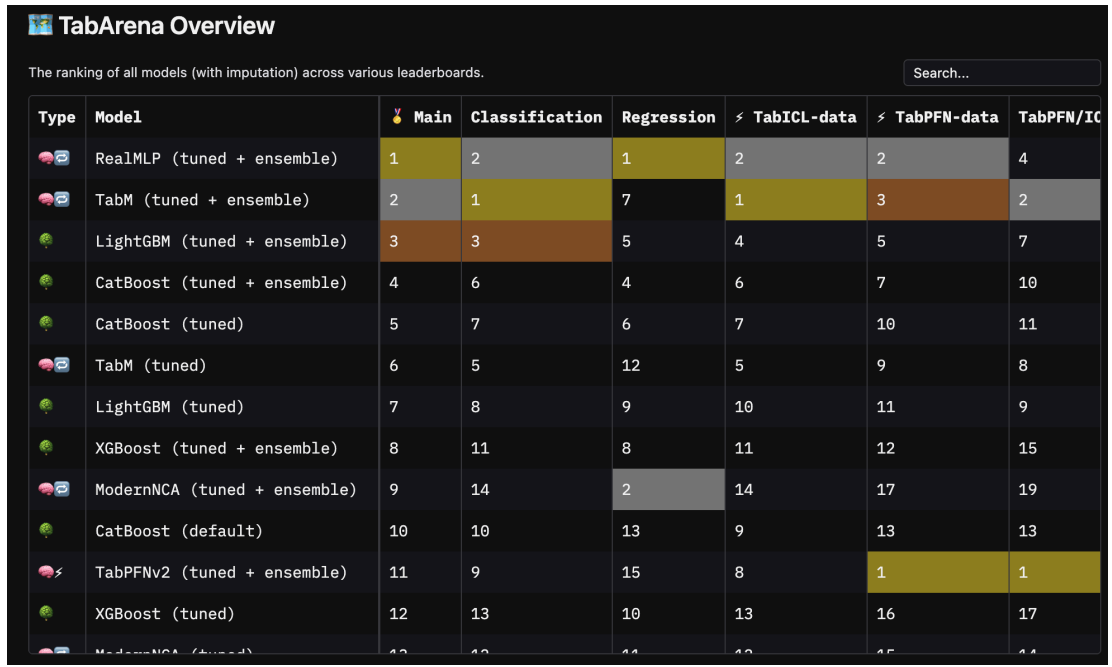
Notes on NGBoost

- Paper `https://arxiv.org/abs/1910.03225`
- Gradient Boosting for all probabilistic models
- Input
  - Base learner (e.g. Regression Tree)
  - Parametric probability distribution (Normal, Laplace, etc.)
  - Scoring rule (Log-Likelihood, CRPS)
    - CRPS is a robust alternative to Log-Likelihood
- Special Trick
  - For more than one dimension, like the two in
    - $p_\theta(y|x) = \mathrm{N}(y; \mu = f_{M1}(x), \sigma = \exp(f_{M2}(x)))$
  - Normal Gradient makes problems, they are replaced it with natural gradient hence NGBoost*

*Or is NGBoost because of famous author Andrew NG?

State of The Art Methods

https://huggingface.co/spaces/TabArena/leaderboard



TabArena Overview

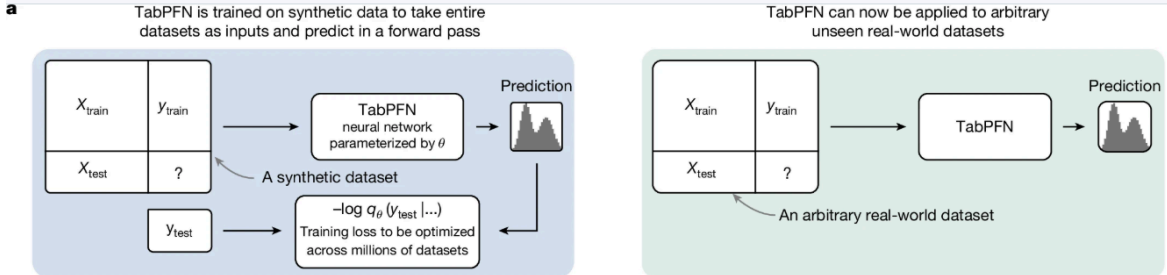
The ranking of all models (with imputation) across various leaderboards.

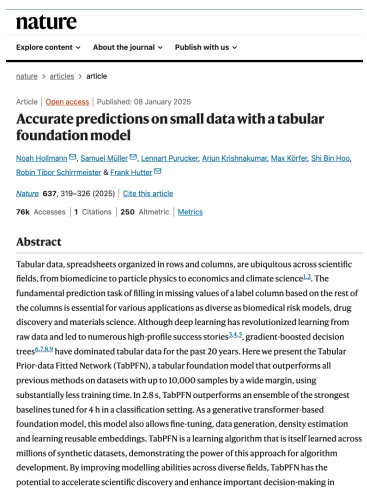
Search...

Type	Model	Main	Classification	Regression	TabICL-data	TabPFN-data	TabPFN/IC
	RealMLP (tuned + ensemble)	1	2	1	2	2	4
	TabM (tuned + ensemble)	2	1	7	1	3	2
	LightGBM (tuned + ensemble)	3	3	5	4	5	7
	CatBoost (tuned + ensemble)	4	6	4	6	7	10
	CatBoost (tuned)	5	7	6	7	10	11
	TabM (tuned)	6	5	12	5	9	8
	LightGBM (tuned)	7	8	9	10	11	9
	XGBoost (tuned + ensemble)	8	11	8	11	12	15
	ModernNCA (tuned + ensemble)	9	14	2	14	17	19
	CatBoost (default)	10	10	13	9	13	13
	TabPFNv2 (tuned + ensemble)	11	9	15	8	1	1
	XGBoost (tuned)	12	13	10	13	16	17

TabArena a benchmarking system for machine-learning models on tabular data.
Many methods are tuned and ensembled.
Deep Learning models (often pre-trained on existing data) are entering the field. In contrast to popular believe DL only for text images?

TabPFN





nature

Explore content · About the journal · Publish with us

nature › articles › article

Article | Open access | Published: 08 January 2025

Accurate predictions on small data with a tabular foundation model

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister & Frank Hutter

Nature **637**, 319–326 (2025) | Cite this article

76k Accesses | **1** Citations | **250** Altmetric | Metrics

Abstract

Tabular data, spreadsheets organized in rows and columns, are ubiquitous across scientific fields, from biomedicine to particle physics to economics and climate science[1,2]. The fundamental prediction task of filling in missing values of a label column based on the rest of the columns is essential for various applications as diverse as biomedical risk models, drug discovery and materials science. Although deep learning has revolutionized learning from raw data and led to numerous high-profile success stories[3–5], gradient-boosted decision trees[6,7,8,9] have dominated tabular data for the past 20 years. Here we present the Tabular Prior-data Fitted Network (TabPFN), a tabular foundation model that outperforms all previous methods on datasets with up to 10,000 samples by a wide margin, using substantially less training time. In 2.8 s, TabPFN outperforms an ensemble of the strongest baselines tuned for 4 h in a classification setting. As a generative transformer-based foundation model, this model also allows fine-tuning, data generation, density estimation and learning reusable embeddings. TabPFN is a learning algorithm that is itself learned across millions of synthetic datasets, demonstrating the power of this approach for algorithm development. By improving modelling abilities across diverse fields, TabPFN has the potential to accelerate scientific discovery and enhance important decision-making in various domains.

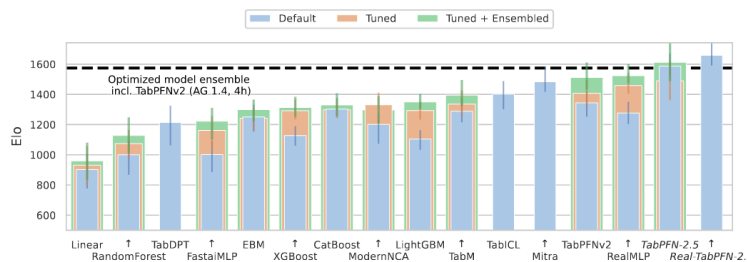


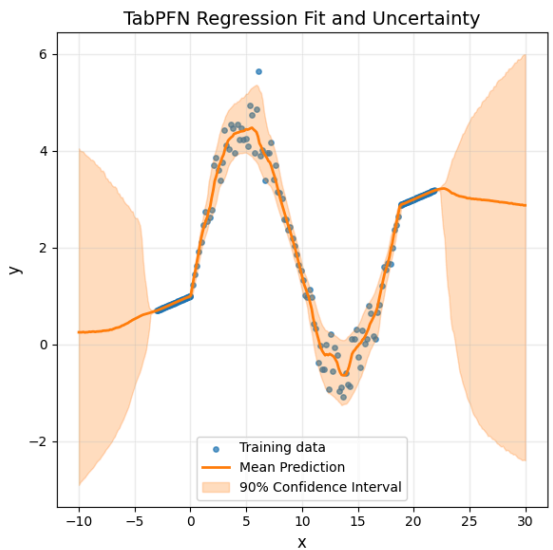
Figure 1: TabPFN-2.5 performance on the standard TabArena-lite benchmark [1], TabPFNv2 classification subset. TabPFN-2.5 outperforms any other model in a forward pass, and marks a strong leap from TabPFNv2. When fine-tuned on real data, Real-TabPFN-2.5 shows even stronger performance. The horizontal dotted line stands for AutoGluon 1.4 extreme mode tuned for 4 hours, an ensemble of models including TabPFNv2.

http://arxiv.org/abs/2511.08667 (Released 6 Nov)

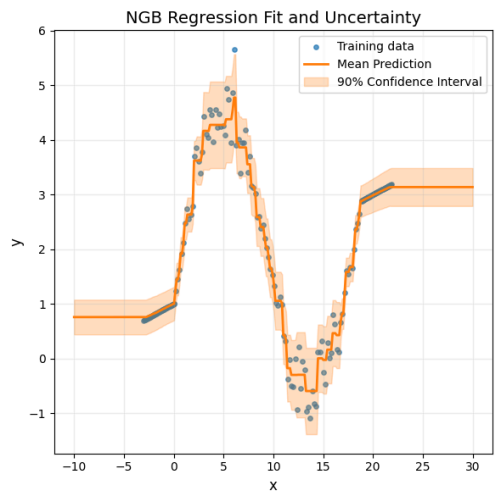
TabPFN

```python
from tabpfn import TabPFNRegressor
reg = TabPFNRegressor()
x_train, y_train = dgp(100)
x_val = np.linspace(-10, 30, 300).reshape(-1, 1)
reg.fit(x_train, y_train)
#preds = reg.predict(x_val, output_type="full")
quantile_predictions = reg.predict(x_val, output_type='quantiles', quantiles=[0.05, 0.5, 0.95])
```

✓ 4.9s Python



NGBoost for comparison



https://github.com/ioskn/mldl_htwg/blob/master/boosting/ProbBoosting.ipynb