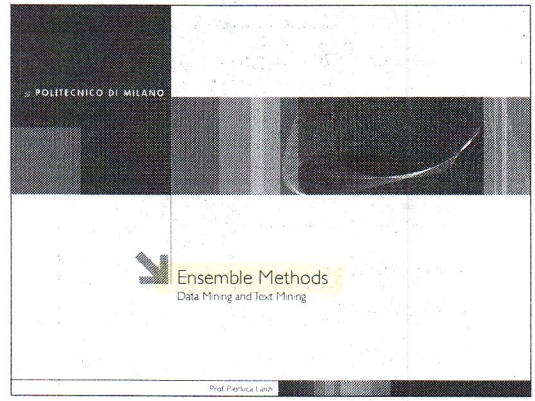
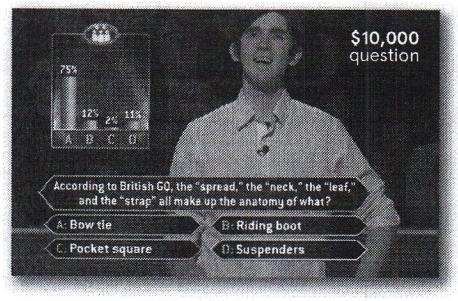
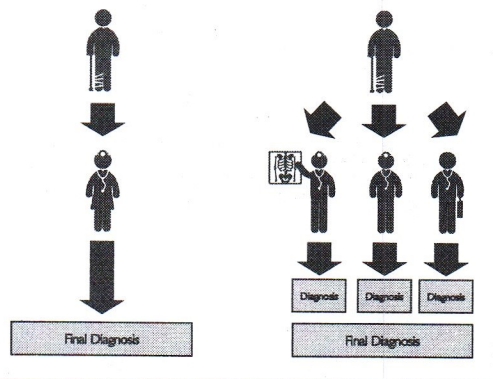


What are Ensemble Methods?



https://medium.com/fathominfo/especially-big-data-gameshow-edition-8f8faee5d479



Ensemble Methods

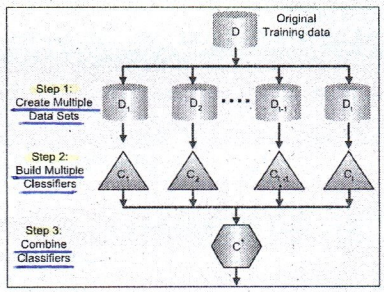
Generate a set of classifiers from the training data

Predict class label of previously unseen cases by aggregating predictions made by multiple classifiers

Majority vote for classification

Averaging for regression

What is the General Idea?



Building models ensembles

- **Basic idea**
  - Build different "experts", let them vote
- **Advantage**
  - Often improves predictive performance
- **Disadvantage**
  - Usually produces output that is very hard to analyze

It's difficult to create a narrative behind different opinions

Why does ensembles work?

- Same reason we ask for a second opinion from another doctor:
- Good chance one expert makes a mistake in her prediction
- But not much chance that a set of experts all will
- Unless there's a lot of group think going on of course

Why does it work?

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\epsilon = 0.35$ (which is pretty high)
- Assume classifiers are independent
- The probability that the ensemble makes a wrong prediction is

$$P(X \geq 13 \mid p = \epsilon, n = 25) = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

majority voting

meaning that the data they receive are independent one from another

how can we generate several models using the same data and the same approach (e.g., Trees)?

we might …

randomize the selection of training data?

randomize the approach?

…

Bagging

What is Bagging? (Bootstrap Aggregation)

- **Analogy**
  - Diagnosis based on multiple doctors' majority vote
- **Training**
  - Given a set D of d tuples, at each iteration i, a training set Di of d tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model Mi is learned for each training set Di
- **Classification (classify an unknown sample X)**
  - Each classifier Mi returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to X
- **Combining predictions by voting/averaging**
  - The simplest way is to give equal weight to each model
- **Predicting continuous values**
  - It can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

we want data to be ll : if we split the dataset in 2 one model will use 50% of the data and the other the other 50%.. This is not ll ! The performance of one classifier depends on the fact that the other classifier has the remaining data. We want to avoid this.

Bagging Algorithm

Model generation

```
Let n be the number of instances in the training data
For each of t iterations:
   Sample n instances from training set
   (with replacement)
   Apply the learning algorithm to the sample
Store the resulting model
```

Classification

```
For each of the t models:
   Predict class of instance using model
Return class that is predicted most often
```


Bagging Regressor Pseudocode - Fitting


```
def BaggingFit(X,y,base_model,no_models):
   m,n = X.shape

   models = [ None ] * no_models

   for i in range(no_models):

   # generate the array of bootstrapped examples
   ind = np.floor( m * np.random.rand(m) ).astype(int)

   # create a copy of the base model
   model = sklearn.base.clone(base_model)

   # fit the classifier
   model.fit(X[ind,:], y[ind])

   # memorize the model for later prediction
   models[i] = model

   # return the array of models
   return models
```

bootstrap

Bagging Regressor Pseudocode - Predicting


```
def BaggingPredict(models,X,use_average=False):
    no_models = len(models)
    m = X.shape[0]
    predict = np.zeros( (m, no_models) )
    for i in range(no_models):
        predict[:,i] = models[i].predict(X)


    if use_average:
        predict = np.mean(predict, axis=1)
    else:
        # Make overall prediction by majority vote
        predict = np.mean(predict, axis=1) > 0 # if +1 vs -1

    return predict
```


Bagging works because it reduces variance by voting/averaging

usually, the more classifiers the better

it can help a lot if data are noisy

however, in some pathological hypothetical situations the overall error might increase

When Does Bagging Work? Stable vs Unstable Classifiers

- A learning algorithm is unstable, if small changes to the training set cause large changes in the learned classifier
- If the learning algorithm is unstable, then bagging almost always improves performance
- Bagging stable classifiers is not a good idea
- Unstable classifiers
  - Decision trees, regression trees, linear regression, neural networks are examples of unstable classifiers
- Stable classifiers
  - K-nearest neighbors (with larger k), models with strong regularization

the information gain may immediately change the decision based on small values of distribution of the classes

the more uncorrelated the trees, the better the variance reduction

Random Forests

random forests are ensembles of unpruned decision tree learners with randomized selection of features at each split

What is a Random Forest?

- Random forests (RF) are a combination of tree predictors
- Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest
- The generalization error of a forest of tree classifiers depends on
  - The strength of the individual trees in the forest (how good are they on average?)
  - The correlation between them (how much group-think is there in the predictions)
- Using a random selection of features to split each node yields error rates that compare favorably to Adaboost, and are more robust with respect to noise

How do Random Forests Work?

- D = training set
- k = number of trees in forest

F = set of variables to test for split
n = number of variables

- for i = 1 to k do
  - build data set D_i by sampling with replacement from D
  - learn tree T_i from D_i using at each node only a subset F of the n variables
  - save tree T_i as it is, do not perform any pruning
- Output is computed as the majority voting (for classification) or average (for prediction) of the set of k generated trees

before we build anything we may decide that for each tree in the forest we use (e.g.) 70% of the variables. Each tree will have 70% of the variables and different data. Notice that two different trees may have different variables.

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

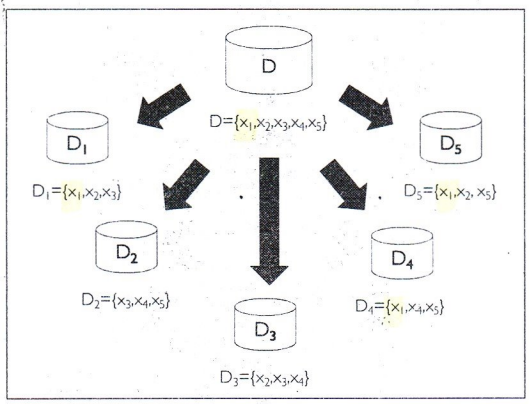
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Out of Bag Samples

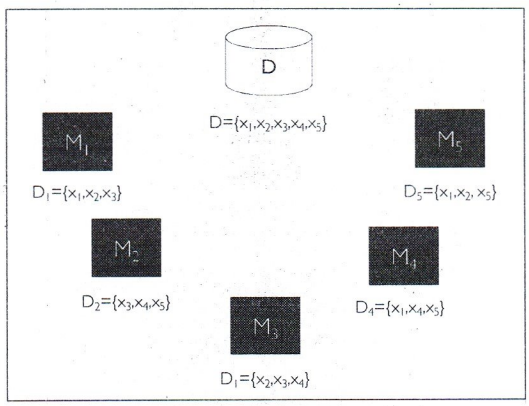
- For each observation (x_i, y_i), construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which the observation did not appear
- The OOB error estimate is almost identical to that obtained by n-fold crossvalidation and related to the leave-one-out evaluation
- Thus, random forests can be fit in one sequence, with cross-validation being performed along the way
- Once the OOB error stabilizes, the training can be terminated

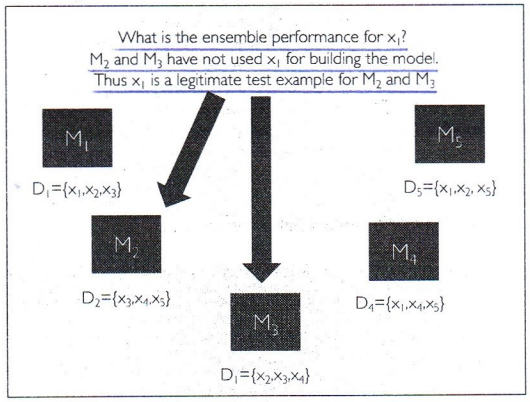
Tracking the OOB error leads to the decision of (e.g.) how many trees do we need. If we start with N trees and the error is high we add more trees. At a certain point adding trees won't change too much ⇒ training terminated

Roudom forest with 5 predictors:
3 values (listed)
+ 2 copies among the 3 values



x_1 is used for train in D_1, D_4, D_5 and for test in D_2, D_3.



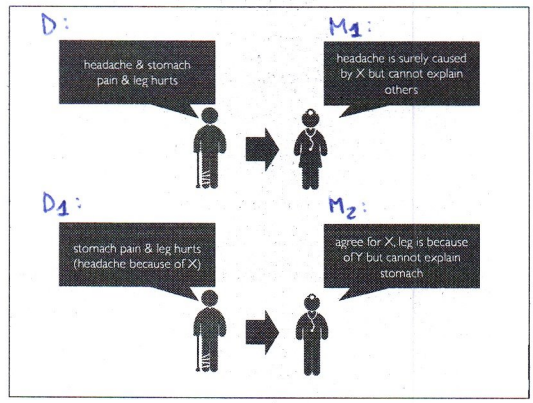


Properties of Random Forests

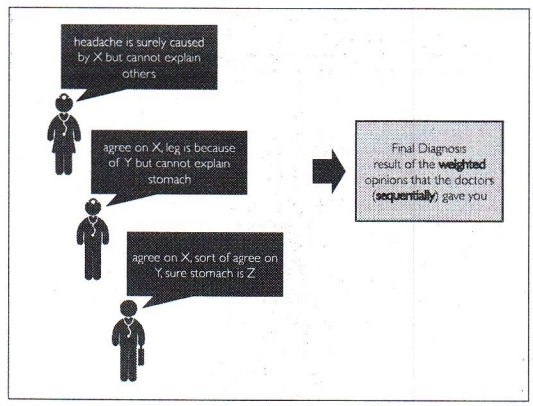
30

- Easy to use ("off-the-shelve"), only 2 parameters (no. of trees, %variables for split)
- Very high accuracy
- No overfitting if selecting large number of trees (choose high)
- Insensitive to choice of split% (~20%)
- Returns an estimate of variable importance
- Random forests are an effective tool in prediction.
- Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set.
- Random inputs and random features produce good results in classification - less so in regression.
- For larger data sets, we can gain accuracy by combining random features with boosting.

Boosting



By iteration we try to create models that try to solve mistakes made by the previous models. We obtain a sequence of models.



It applies to classification with 2 classes (+1/-1) and it produces a classifier which is a combination of weak classifiers

AdaBoost

34

- AdaBoost computes a strong classifier as a combination of weak classifiers,

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad \in \{-1, +1\}$$

- Where $h_t(x)$ is the output of the t^{th} weak classifier t
- α_t is weight assigned to the t^{th} weak classifier based on its estimated error ϵ_t as

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

- Note that this "confidence" weight is just the log odds of a correct prediction!

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

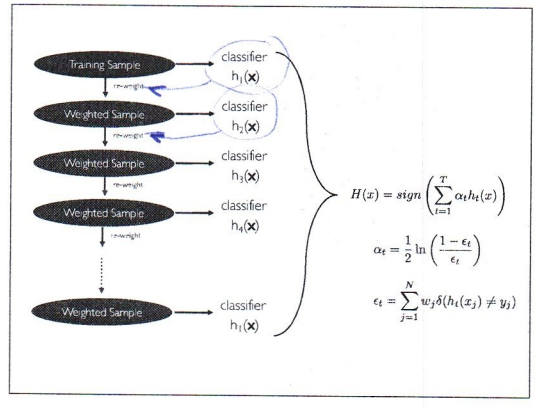
What is Boosting?

35

- **Analogy**
  - Consult several doctors, sequentially trying to find a doctor that can explain what other doctors cannot
  - Final diagnose is based on a weighted combination of all the diagnoses
  - Weights are assigned based on the accuracy of previous diagnoses
- **How does Boosting work?**
  - Weights are assigned to each training example
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier M_{i+1} to pay more attention to the training tuples that were misclassified by M_i
  - The final M^* combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

We adjust the generation of the dataset focusing on the error that the latest classifier did. We update the probability distribution of extracting the datapoints along the way.



It's like when we're preparing an exam. At the beginning we practice on all the exercise, as we go on we focus on those with which we have problems (however we still work on the others (less) because we don't want to forget them)

AdaBoost Example (1)

37

- Suppose our data consist of ten points with a +1/-1 label

x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

- At the beginning, all the data points have the same weight and thus the same probability of being selected for the training set. With 10 data points the weight is 1/10 or 0.1

w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

AdaBoost Example (2)

38

- Suppose that in the first round, we generated the training data using the weights and generated a weak classifier h_1 that returns +1 if x<3 (bootstrap)

				h_1						
w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1
	+1	**+1**	**+1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**

← targets
← predictions of the overall model

- The error ϵ_1 on the weighted dataset is 2/10 and α_1 is 0.69
- The weights are updated by multiplying them with
  - $\exp(-\alpha_1)$ if the example was correctly classified
  - $\exp(\alpha_1)$ if the example was incorrectly classified
- Finally, the weights are normalized since they have to sum up to 1

the weights of the misclassified increase because we want to bootstrap a lot of them in the next train

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

AdaBoost Example (3)

39

- The new weights are thus,

w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

- Suppose now that we computed the second weak classifier h_2 as

								h_2		
w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1
	-1	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**+1**	**+1**

- That has an error ϵ_2 of 0.1875 and α_2 is 0.7332

(these are evaluated on the ensamble of 1+2)

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

AdaBoost Example (4)

40

- The new weights are thus,

w	0.167	0.167	0.167	0.038	0.038	0.038	0.038	0.038	0.154	0.154
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1

- Finally, we sample the dataset using the weights and compute the third weak classifier h_3 as

									h_3	
w	0.167	0.167	0.167	0.038	0.038	0.038	0.038	0.038	0.154	0.154
x	0	1	2	3	4	5	6	7	8	9
y	+1	+1	+1	-1	-1	-1	-1	-1	+1	+1
	+1	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**	**+1**

- That has an error ϵ_3 of 0.19 and α_3 of 0.72

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

AdaBoost Example (5)

41

- The final model is thus computed as,

$$H(x) = sign(0.69h_1(x) + 0.73h_2(x) + 0.72h_3(x))$$

Prof. Pierluca Lanzi — POLITECNICO DI MILANO

AdaBoost.M1

Model generation

```
Assign equal weight to each training instance
For t iterations:
  Apply learning algorithm to weighted dataset,
     store resulting model
  Compute model's error e on weighted dataset
  If e = 0 or e ≥ 0.5:
     Terminate model generation
  For each instance in dataset:
     If classified correctly by model:
        Multiply instance's weight by e/(1-e)
  Normalize weight of all instances
```

Classification

```
Assign weight = 0 to all classes
For each of the t (or less) models:
For the class this model predicts
     add -log e/(1-e) to this class's weight
Return class with highest weight
```


Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

More on Boosting

- Boosting needs weights ... but
- Can adapt learning algorithm ... or
- Can apply boosting without weights
  - Resample with probability determined by weights
  - Disadvantage: not all instances are used
  - Advantage: if error > 0.5, can resample again
- Stems from computational learning theory
- Theoretical result: training error decreases exponentially
- It works if base classifiers are not too complex, and their error doesn't become too large too quickly
- Boosting works with weak learners only condition: error doesn't exceed 0.5
- In practice, boosting sometimes overfits (in contrast to bagging)

Boosting

- Also uses voting/averaging
- Weights models according to performance
- Iterative: new models are influenced by the performance of previously built ones
  - Encourage new model to become an "expert" for instances misclassified by earlier models
  - Intuitive justification: models should be experts that complement each other
- Several variants
  - Boosting by sampling, the weights are used to sample the data for training
  - Boosting by weighting, the weights are used by the learning algorithm

Gradient Boosting

focused on the target:
we build a model to predict the target (classification/regression) and then we look at the difference between the target function and the prediction (:= the RESIDUAL). The second model is focused on predicting the RESIDUAL: the second model is not focused on the entire. Then the second model is added to the 1st and they predict better the target.

We repeat this process over and over again until we find a good approximation of the target function.

We generate the 1st model $f_0(x)$ based on the "entire".

Then we generate other M models on the (progressive) residuals (actually we generate a model based on the residuals, we update the model and we repeat)

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Why is it Called Gradient Boosting? (Intuition using MSE)

48

- To compute a model to predict a target value y_i that minimizes a loss function, for instance the mean square error
$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$
- We can adjust $\hat{y}_i$ to try to reduce the error using the gradient
$$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$
- The gradient for the MSE is proportional to,
$$y_i - \hat{y}_i$$
- Each learner is estimating the gradient of the loss. Larger α means larger steps, smaller α smaller steps and smoothing effect

Gradient Boosting

49

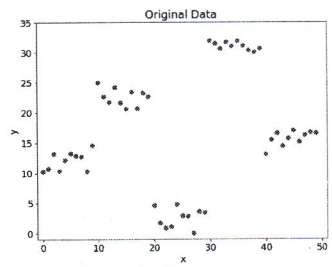
- Build a sequence of predictors by repeating three simple steps
  1. Learn a basic predictor
  2. Compute the gradient of a loss function wrt the predictor
  3. Compute a model to predict the residual
  4. Updated the predictor with the new model
  5. Goto 2
- The predictor is increasingly accurate and increasingly complex

This can be very simple (and therefore fast): for example, for regression it could be the average

Gradient Boosting Example – Original Data

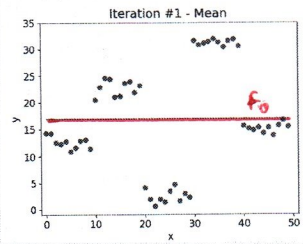
50

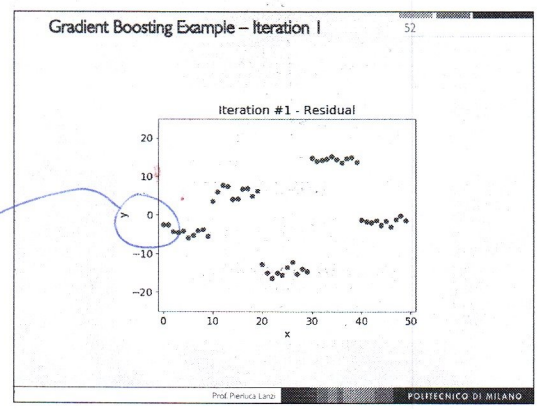
(regression)



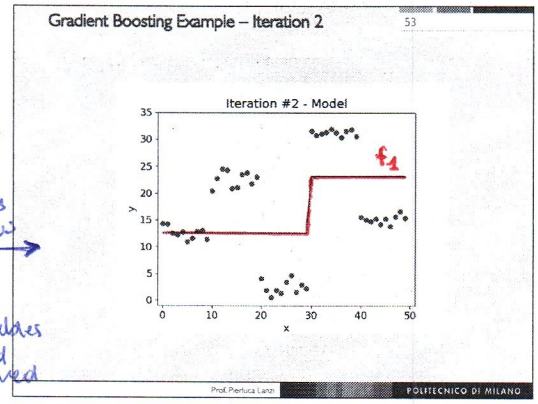
Gradient Boosting Example – Iteration 1

51





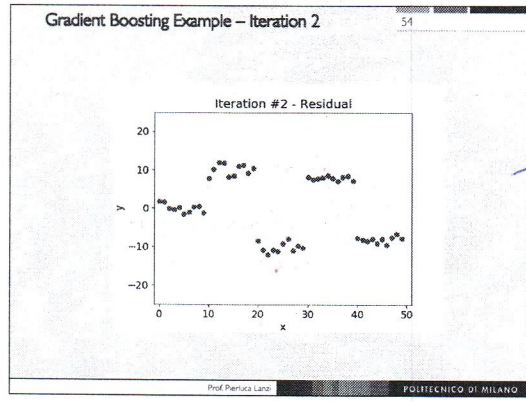
the residuals (previous values − mean) are now centered around 0



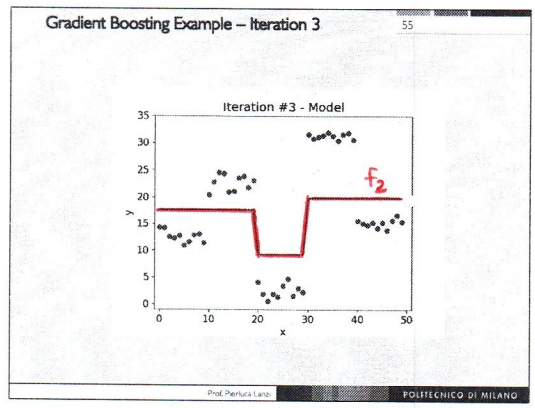
We build a second classifier, a little bit more complex than the mean, to predict the residuals. Once we built the model for the residuals we sum it to the previous and we obtain →

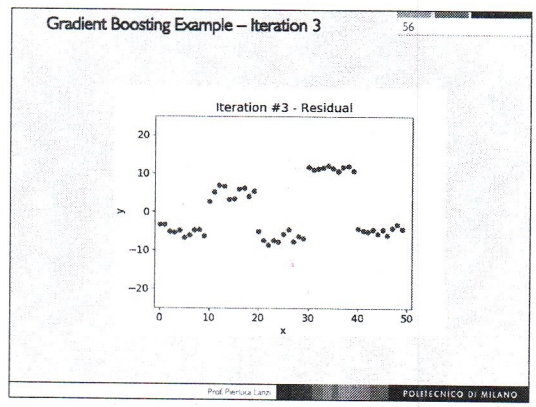
Here we se the overall f_1 model in the output variables space, however the second model is built and learned in the residual space.

Note: the more convenient model that we can use for modelling the residuals is a stump (one level decision tree):



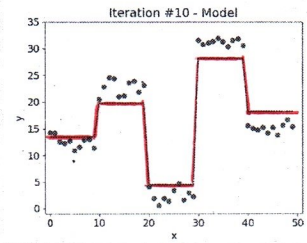
If we're proceding well we should see a decrease in the amplitude of the residuals





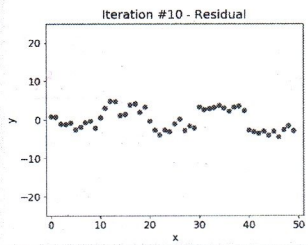
Gradient Boosting Example – Iteration 10

57



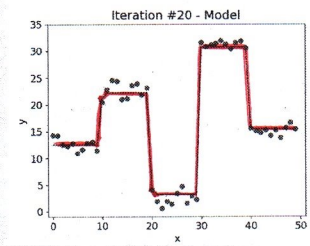
Gradient Boosting Example – Iteration 10

58



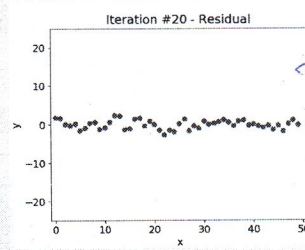
Gradient Boosting Example – Iteration 20

59



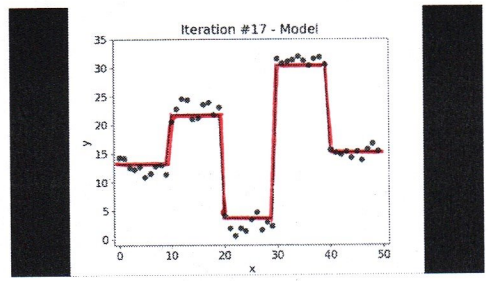
Gradient Boosting Example – Iteration 20

60



we see that the residuals converge to something centered in zero and very close to zero

If we get a better α:



Gradient Boosting Example
https://youtu.be/rvM648cedXU

Gradient Boosting: Pseudocode for training

62

```
# number of models
n_boost = 50
# array of models
learner = [None]* n_boost

# start with the basic predictor (the mean)
mu = mean(y)
dy = y - mu

for k in range(n_boost):
    # fit the residual
    learner[k] = model.fit(X,dy)

    # set the learning rate
    alpha[k] = 1

    # update the residual
    dy = dy - alpha[k] * learner[k].predict(X)
```

as first model (f_0) we use the mean

first residual (target − prediction)

we're doing the fit of the input data, but we're not targeting the original targeting (y), we're targeting the residuals (dy)

tipically is $1/k$ or $1/\log(k)$ because we want the importance of each predictor to decrease in time

Gradient Boosting: Pseudocode for testing

63

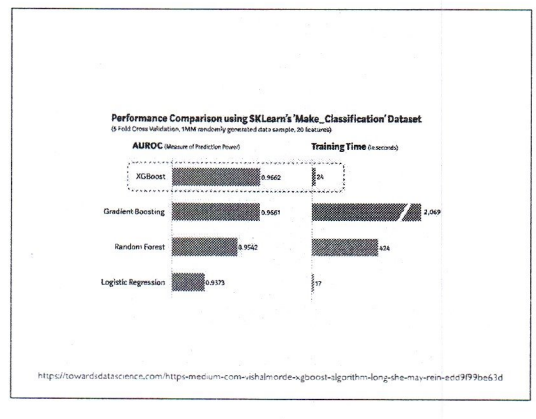
```
# X_test and y_test are the test data
predict = zeros(len(y_test))+mu

for k in range(n_boost):
    predict = predict + alpha[k]*learner[k].predict(X_test)
```

we sum up all the predictions of all the possible predictors

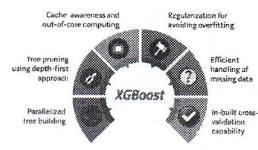
eXtreme Gradient Boosting (XGBoost)

64

- Efficient and scalable implementation of gradient boosting applied to classification and regression trees
- Used by most of the winning solutions and runner up of the recent data science competitions
- It only deals with numerical variables
- Features
  - Novel tree learning algorithm to handle sparse data
  - Approximate tree learning using quantile sketch
  - More than ten times faster on single machines
  - More regularized model formalization to control over-fitting, which gives it better performance.



66

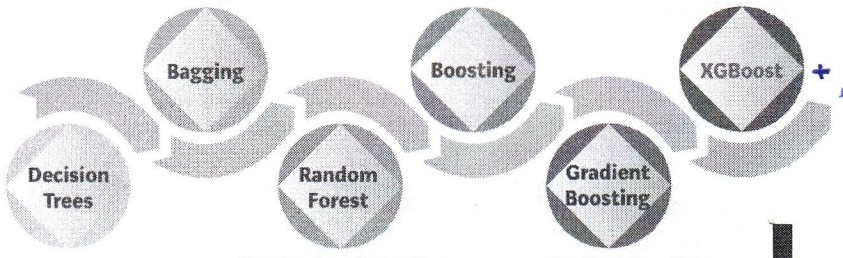


The focus is on the points where the model made mistakes

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism

Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models

Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias



+ Cat Boost for categorical

A graphical representation of possible solutions to a decision based on certain conditions

Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees

moreover we eliminate pruning: the trees are highly uncorrelated

Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

The focus is on the target function and on the residuals

So far all the ensambles we saw were made by the same type of models (they target different things but they're all trees / all logistic regressions/..)
→ homogeneous ensambles
What if we stack together heterogeneous models?

STACKED GENERALIZATION (Stacking)

Stacking Generalization (Stacking) 69

- Train a learning algorithm to combine the predictions of a heterogeneous set of learning algorithms
  - First, a set of base learners are trained over the data (to generate level-0 models)
  - Then, a meta learner is trained using the predictions of base classifiers as additional inputs (to generate a level-1 model)
- Level-0 models are generated different learning schemes
- Typically yields performance better than any single one of the trained models

Prof. Pierluca Lanzi POLITECNICO DI MILANO

One way to combine them is to put the different models as a kind of input in a new learning algorithm that we train to combine the outputs of the models.
We build a meta learner that takes the outputs of the models and learns which models to select in some situations. like: "in that part of the space the model i is better than the model j".
Sometimes this is also called: PATCHWORK MODEL (because we cover in different ways different part of the space)

The meta model is trained on the outputs of the original models (≠ from the homogen. ensambles because there the ensamble method was fixed (majority voting, average, ..))

We want this because we want to capture different regularities

Generating the level-1 training data 70

- The training dataset for the meta-model is usually generated using k-fold cross-validation of the base models
- The out-of-fold predictions are used as the basis for the training dataset for the meta-model.
- The training data for the meta-model may also include the inputs to the base models, e.g. input elements of the training data.
- This can provide an additional context to the meta-model as to how to best combine the predictions from the meta-model.

Prof. Pierluca Lanzi POLITECNICO DI MILANO

we can be in 2 scenarios:

1. data → model 1 → output; data → model 2 → output; data → model K → output → meta
2. data → model 1 → output; data → model 2 → output; data → model K → output; data → meta

Generating the level-1 training data

- Training data for level-1 model contains predictions of level-0 models as attributes; class attribute remains the same.
- Note that, we cannot use the level-0 models' predictions on their training data as attribute values for the level-1 data
- Instead, we must generate the level-1 training data by running a cross-validation for each of the level-0 algorithms
- Then, the predictions obtained for the test instances encountered during the cross-validation are collected and become the training test for the meta-model

Prof. Pierluca Lanzi POLITECNICO DI MILANO

More on stacking

- Stacking is hard to analyze theoretically
- If possible, better to have base learners that output probabilities since these give more information to the meta learner
- Which algorithm as meta-learner?
  - Any learner can be used, in principle
  - Prefer "relatively global, smooth" models to
- Note that stacking can be trivially applied to numeric prediction to reduce the risk of overfitting

this is more a rule of thumb, an intuition, it has no theoretical background

- Let's say you want to do 2-fold stacking:
- Split the train set in 2 parts: train_a and train_b
- Fit a first-stage model on train_a and create predictions for train_b
- Fit the same model on train_b and create predictions for train_a
- Finally fit the model on the entire train set and create predictions for the test set.
- Now train a second-stage stacker model on the probabilities from the first-stage model(s).
- A stacker model gets more information on the problem space by using the first-stage predictions as features, than if it was trained in isolation.
- https://mlwave.com/kaggle-ensembling-guide/

Warning

All the approaches discussed here can be applied both to classification and prediction!

Summary

Summary

- **Ensemble methods**
  - Generate a set of classifiers from the training data
  - Combine the predictions of the classifiers to achieve better performance (on average) than any of the base classifiers
- **Common techniques**
  - Bagging: train members of ensemble on random copies of data
  - Boosting: train members of ensemble on reweighted data
  - Stacking: train a learner to combine predictions of other models