# 32.1 Ensemble models

Ensembles :

↳ group of things

(or)

Collection of things

In ML : multiple models used together

$m_1, m_2, m_3 \dots m_k$ → base models

(SVM, logistic, DT, KNN etc.)

Combine these model to create a "more powerful model"

4 types

↳ Bagging (Bootstrapped Aggregation)

↳ Boosting → high performing

↳ Stacking → V. powerful

↳ Cascading → kaggle competitions

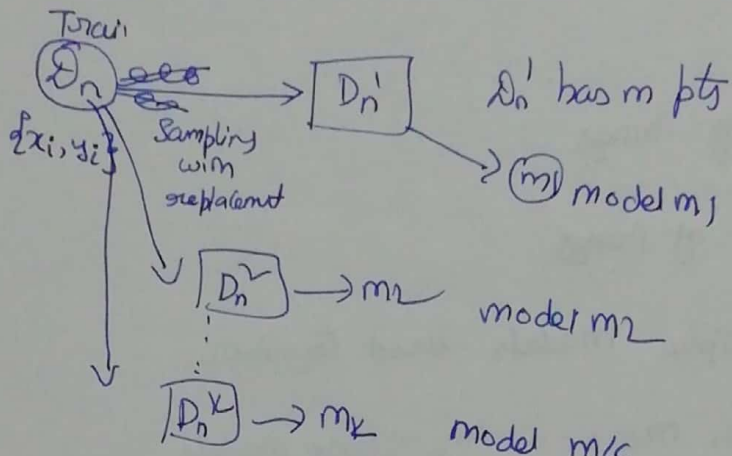
Key aspect : $m_1, m_2, m_3 \dots m_k$

more different the models are the better you can combine them.

{ Problem : $m_i$ - Expert

## 32.2 Bootstrapped Aggregation (Bagging)

**Intuition:**



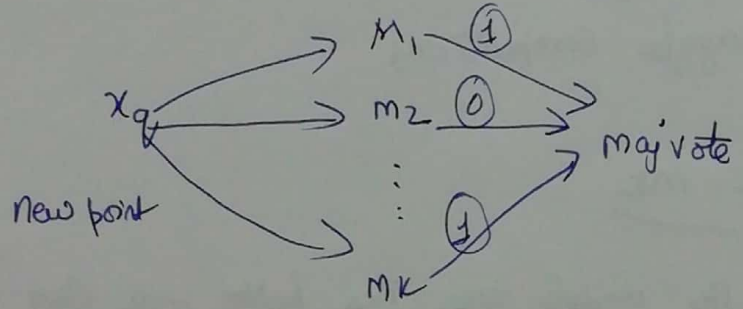

Models are built using $D_n'$ of size m ($m \leq n$)

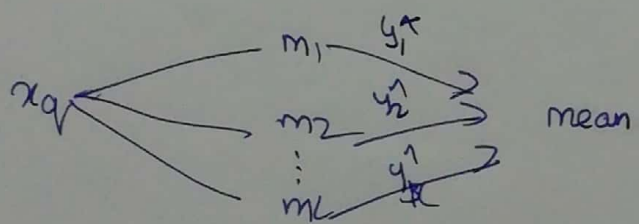
⇒ Each model $M_i$ has seen a different subset of data

→ Each sample is called bootstrap sample

Aggregation: classification : majority vote

Regression : mean/median

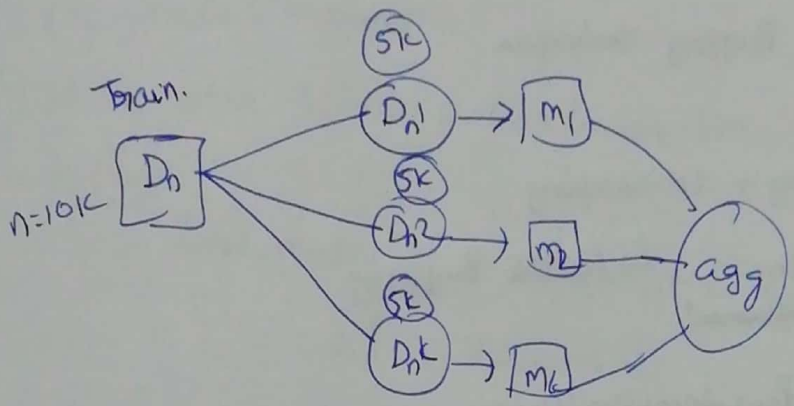



$\{y_1, y_2, y_K\}$

mean of $\{y_1, y_2, y_K\}$ is our output

## Bootstrap Sampling




Variance :- How much does a model change with changes in training data.

Let's assume ~~we~~ first we have n=10K, now we have changed 100 pts in Strain. These 100 pts might effect ~~we~~ few subset of the ~~mod~~ sample. It will effect few models not all.

Bagging → Can reduce the variance in the model with out impacting the bias

model Error :- Bias + Var.

bare-model ($m_i$) → low bias, & high-Var model

Bagging ($m_i$'s) → low-bias & reduced Variance.

Bagging says take a bunch of (low-bias, high-Var) models

↓

Combine them using bagging

↓

(low bias, reduced-variance)

DT ~~of~~ with depth of large

↳ high Variance

↳ low bias.

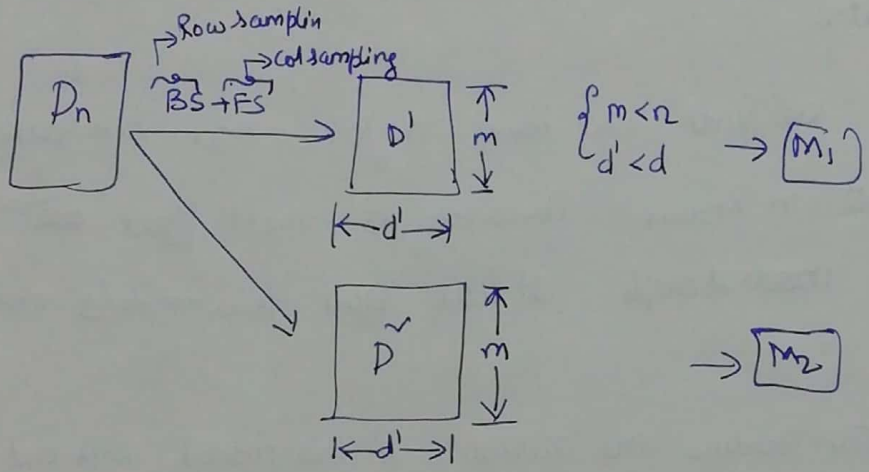
## 32.3 Random Forest & their Construction

→ RF → most popular Bagging technique

RF : DT + Bagging + Col. sampling
- DT → base
- Bagging → Row sampling with replacement
- Col. sampling → feature Bagging

Bootstrap sampling : Row sampling / Bagging

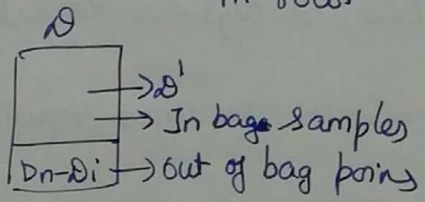
column sampling : Feature sampling / Bagging




$M_i$'s ÷ DT of reasonable depth

$M_i \longrightarrow D^i \rightarrow$ d' columns, m rows

$D_n - D^i$ ← minus




↳ we can be used as a CV dataset for model $M_i$

## Summary

RF : DT base models (→ of reasonable depth (or) fully grown tree)
- + row sampling with replacement
- + col. sampling
- + agg (majority vote) (or) (mean & median)

## 32.4 Bias and Variance Trade offs.

RF: reduce Variance

↳ low bias because, base learner ($M_i$'s) are low biased

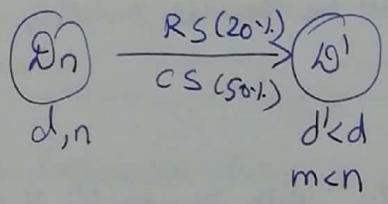
↳ (Var)

final model $M = agg(M_1, M_2, M_3 \cdots M_k)$

$k \uparrow$ ; Variance $\downarrow$

$K \downarrow$ ; Variance $\uparrow$

→ we have seen 2000 learners

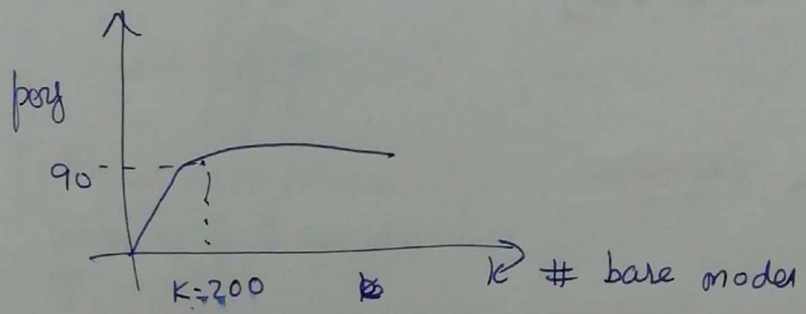



$\frac{d'}{d}$ = col sampling rate

$\frac{m}{n}$ = Row sampling rate

col. S.R ↓
row S.R ↓
→ low variance model & Variance ↓

$D_n$ → Training data → $D_1 - M_1$
$D_2 \rightarrow M_2$




Hyper parameter

k → # base learners

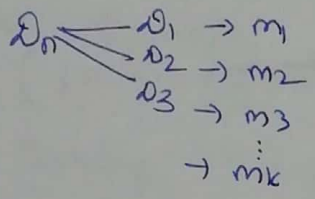
col. sampling rate

Row sampling rate

32.5 Train & run time Complexity

Imagine we are training RF with K base learner (DT) (fully grown)

Train : $O(n \log d * K)$




→ In modern computing, we will have multicore, Each model can be trained on a single core.

If 4 core system, four models can be trained parallely

→ Trivially parallizable

→ when we have large amount of data with reasonable # features

Runtime : $O(depth * K)$

↓

large (10 to 20)

Space complexity : $O(DT * K)$ → if else

## 32.6 Bagging code Sample

→ Sklearn.ensemble. RandomForestClassifier.

→ n_estimators = K (# of models)

→ max_depth = None
↳ we build DT to deep

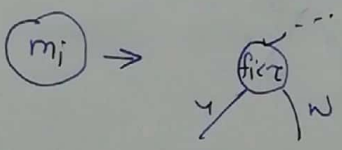
→ bootstrap = True

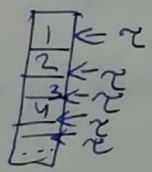
→ oob_score = False
↳ If we make it true
for every model that is build, it will return the oob-score.

## 32.7 Extremely Randomized Trees

→ Sklearn implemented ExtraTreesClassifier and ExtraTreesRegressor

→ In RF we do col. sampling, row sampling and agg

→ ExtremeTrees → 


RF/DT
- fi : real valued value.
- sort fi 

- Try out all the possible values of fi to determine r.

→ Instead of Trying all the values, what if we try out random sample of possible values to determine r

→ Extreme Trees: col-sampling, r-sampling + agg + randomization when selecting 'r'

→ Randomization is a way to reduce variance.

→ Extreme trees better than RF in terms of variance

## 32.8 Random Trees: Cases

Random Forest: DT + RS + CS + agg

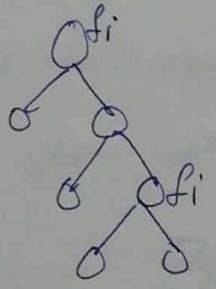
- DT + RS + CS + agg → to reduce variance
- DT → do not handle → large-dim → Cat-features with many categories

Bias-Variance of RF → DT → (depth)
→ (k) = # of base learn
→ deep trees

② Feature Imp:

DT: $f_i$ : - overall reduction in Entropy & $I_G$ because of this feature @ Various level in the DT

RF




RF: Overall reduction in H(Y) & $I_G$ because of feature $f_i$ @ various level of Each of $M_i$'s

## 32.9 Boosting Intuition

32.9 : Boosting Intuition.

Bagging : high variance, low bias + (randomization + aggregation) ← reduce-var.
(RS, RS)

Boosting : low variance, high bias + additively combine
↳ reduce bias while keeping our variance low

$Error = bias^2 + var + \epsilon$

Core idea : reduce bias

$D_{Train} = \{x_i, y_i\}_{i=1}^{n}$

Step 0

$D_{train}$ → Build a model ($M_0$)
↳ high bias & low variance.
Eg : DT which is shallow/less deep
(high variance, low bias)

high-bias → large Training Error

(a) $D_{train} \longrightarrow M_0$
$(x_i, y_i)_{i=1}^{n}$ $\quad y = h_0(x)$

(b) $y_i - h_0(x) = Error_i$ ← Sq. Error
← log Error
← hinge Error.

$\{x_i, y_i, Error_i\}_{i=1}^{n}$

Step : 1 : Instead of taking $D_{train}$ we will take $\{x_i, Error_i\}_{i=1}^{n}$

$M_1 \leftarrow \{x_i, Error_i\}_{i=1}^{n}$

here $Error_i$ is the Error which we calculated in step 0

$h_1(x) = h_1(x_i) = Error_i = y_i - h_0(x_i)$ } it tries to predict Error.

$F_1(x)$ : model at the End of stage 1

$F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$ } : we can think as weighted sum of 2-base models

(2) $\{x_i, \text{Error}_i\} \rightarrow (m_2) \; h_2(x)$

$\uparrow y_i - F_1(x_i)$ → here $F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)$

$F_2(x) = \alpha_0 h_0(x) + \alpha_2 h_2(x) + \alpha_1 h_1(x)$

At the end of stage K :-

$$F_K(x) = \sum_{i=0}^{K} \alpha_i h_i(x)$$

Trained to fit the residual Error @ End of the prev. stage.

additive weighted model

$h_i(x) \leftarrow \{x_i, \text{Error}_i\}$ ; $y_i - F_{i-1}(x)$

↑ residual Error @ End of stage $(i-1)$

$$F_K(x) = \sum_{i=0}^{K} \alpha_i h_i(x)$$

→ Since we are calculating Consequetive models based on the Error from the previous stage.

→ Let K=10

→ it ends up having a low residual Error.

→ At Every stage we are reducing Error and bias in model is also reduces

There are many boosting Technique, we will see

→ Gradient Boosting DT

→ Ada Boosting

↳ Face detection

32.10 Residuals, loss functions, gradients

At the End we got

$$F_k(x) = \sum_{i=1}^{k} \alpha_i h_i(x)$$

Residual @ End of Stage k → (Error i) ⇒ $y_i - F_k(x)$

residual

If we want to train a model $M_{k+1}$, we will train using

$$M_{k+1} \leftarrow \{x_i, Error_i\}$$

Loss-minimization

In logistic Regression we minimized logistic loss
In linear " Square loss
SVM " hinge loss.

$$\mathcal{L}(y_i, F_k(x_i)) = (y_i - F_k(x_i))^2$$

→ sq. loss

Let's call $F_k(x_i) = z_i$

$$\frac{\partial \mathcal{L}}{\partial F_k(x_i)} = \frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial}{\partial z_i}(y_i - z_i)^2$$

$$= (-1) * 2 * (y_i - z_i)$$

$$\frac{\partial \mathcal{L}}{\partial z_i} = -2(y_i - z_i)$$

→ negative gradient $\left\{ -\frac{\partial \mathcal{L}}{\partial F_k(x_i)} = 2(y_i - F_k(x_i)) \right.$

Error i

negative derivative of loss function w.r.t $F_k(x_i)$

residual.

* negative gradient/derivative ≈ Residual

neg. gradient is a pseudo residual (or) Body residual
↳ false

$h_i(x) \leftarrow x_i, \text{Error}_i = y_i - F_{i-1}(x)$

Instead of having a Error at any point i as a residual of point i
what if we replace Error i with pseudo residual

$$-\frac{\partial L}{\partial F_{i-1}(x)}$$

→ Why we have to train on pseudo residuals why not only residuals

Let us have any loss function which is differentiable

RF → min hing loss
↳ we cannot perform using hinge loss, RF use Entropy / $I_G$

GB → min (any loss) function
↳ which is differentiable
↑ Super powerfull

At any stage i

$F_{i-1}(x)$

$(x_i, \text{Error}_i) \rightarrow (M_i)$
$h_i(x)$

pseudo. residual

$$-\frac{\partial L}{\partial F_{i-1}(x)}$$

GB: If we have base models with high bias → we will combine them using pseudo residuals $\left(\frac{-\partial L}{\partial F_{i-1}(x)}\right)$

22.11 Gradient Boosting

Input : Training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$

↑ Logistic Loss
↳ Sq.. loss
↳ hinge loss

Algorithm

1. Initialize model with a constant value :

$$h_0(x) = F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma).$$

find a constant gamma($\gamma$) which minimize my loss $y_i$ over $\gamma$

If we are doing regression problem, we will use squared loss

$$\underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \gamma)^2 \qquad \gamma = \text{mean}(y_i)$$

2. For $m = 1$ to $M$ $\rightarrow h_1(x), h_2(x), h_3(x) \dots h_m(x)$

1. Compute so-called pseudo-residuals :

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x) = F_{m-1}(x)} \quad \text{for } i = 1, \dots, n$$

2. Fit a base learner (eg. tree), $h_m(x)$ to pseudo-residuals.

i.e., train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$

↑ Grad; m

$$F_m(x) = h_0(x) + \gamma_1 h_1(x) + \gamma_2 h_2(x) + \dots \gamma_m h_m(x)$$

3. Compute multiple $\gamma_m$ by solving the following one-dimensional optimization problem.

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

4. Update the model

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

## 32.12 Regularization by shrinkage

$$F_m(x) = h_0(x) + \sum_{m=1}^{M} \gamma_m h_m(x)$$

m = # of base model → (Cross Validation)

as m↑ = overfit ↑ = Var ↑
↳ bias ↓

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

→ In addition to whatever weightage we found using gradient boosting, we are going to reduce that weight by multiplying with a constant "v"

=

→ As v increase. chances of overfitting ↑, variance ↑

v↑ → variance ↑

v↓ → variance ↓

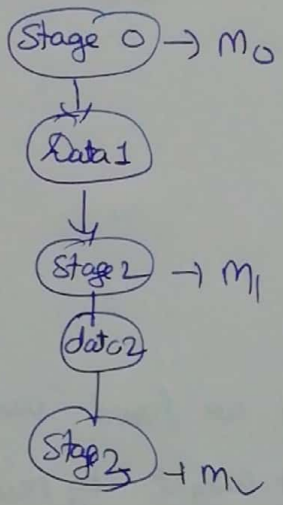
v = [0 to 1]

→ PPl will generally overfit using GBT

## 32.13 Train & Run time Complexity

Train : $O(n \lg n d * M)$ $\quad m = \#$ base_learner.

→ GBDT is not Easy to parallelize




### Runtime

GBDT = $O(depth * m)$

⇑

Small in GB

### Space Complexity

↳ Space it would take to store Each tree & $\gamma_m$'s for Each base learners

→ Good for low latency as the depth of the trees are very less

→ We can choose any loss function.

Analytics Vidhya

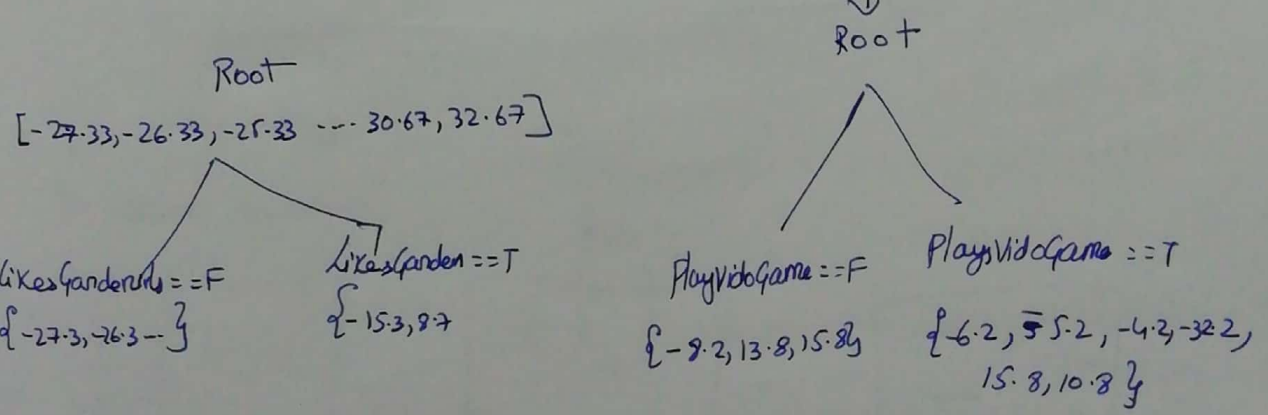
The accuracy of a predictive model can be boosted in two ways. Either by Embracing feature Engineering & by applying boosting algorithms straight away.

There are multiple boosting algorithms

Gradient Boosting
XG Boosting
Ada Boost
Gentle Boost etc.,

Square Error2 (http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

| Age | F0 | PseudoResidual0 | h0 | gamma0 | F1 | Pseudoresidual1 | h1 | gama1 | F2 |
|---|---|---|---|---|---|---|---|---|---|
| 13 | 40.33 | -27.33 | -21.08 | 1 | 19.25 | -6.25 | -3.567 | 1 | 15.68 |
| 14 | " | -26.33 | -21.08 | 1 | 19.25 | -5.25 | -3.567 | 1 | 15.68 |
| 15 | " | -25.33 | -21.08 | 1 | 19.25 | -4.25 | " | 1 | 15.68 |
| 25 | " | -15.33 | 16.87 | 1 | 57.2 | -32.2 | " | 1 | 53.63 |
| 35 | " | -5.333 | -21.08 | 1 | 19.25 | 15.75 | " | 1 | 15.68 |
| 49 | " | 8.667 | 16.87 | 1 | 57.2 | -8.2 | 7.133 | 1 | 64.33 |
| 68 | " | 27.67 | 16.87 | 1 | 57.2 | 10.8 | -3.567 | 1 | 53.63 |
| 71 | " | 30.67 | 16.87 | 1 | 57.2 | 13.8 | 7.133 | 1 | 64.33 |
| 73 | " | 32.67 | 16.87 | 1 | 57.2 | 15.8 | 7.133 | 1 | " |



## 32.14 XGBoost : Boosting + Randomization

Sklearn : GBDT + $\gamma$·S

XGBoost : GBDT + $\gamma$·S + C·S

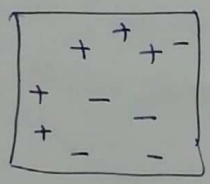
→ `pip3 install XGboost` (or) sudo pip3 install XGboost

https://github/dmlc/xgboost/blob/master/demo/guide-python/sklearn_examples.py

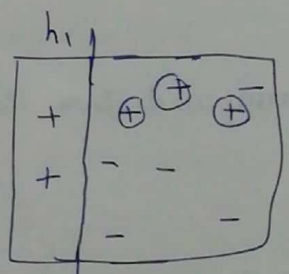
## 32.15 Ada Boost: Geometric intuition

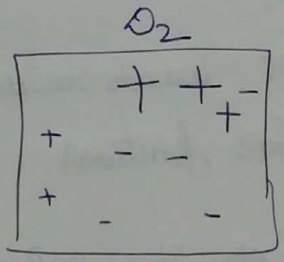
Adaptive Boosting

→ Typically used in image processing, face detection.



The first round:

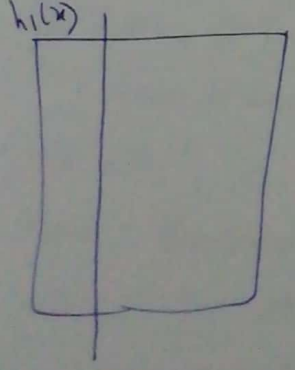


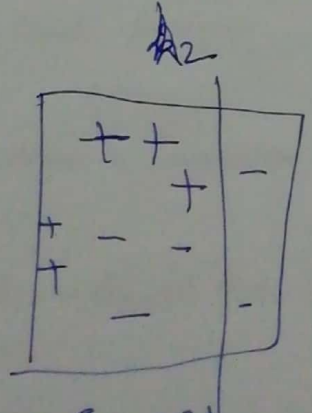



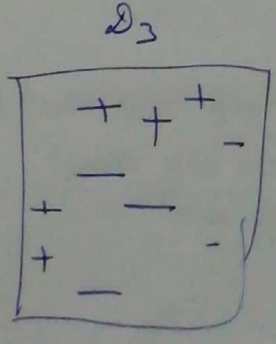

$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

The second round:



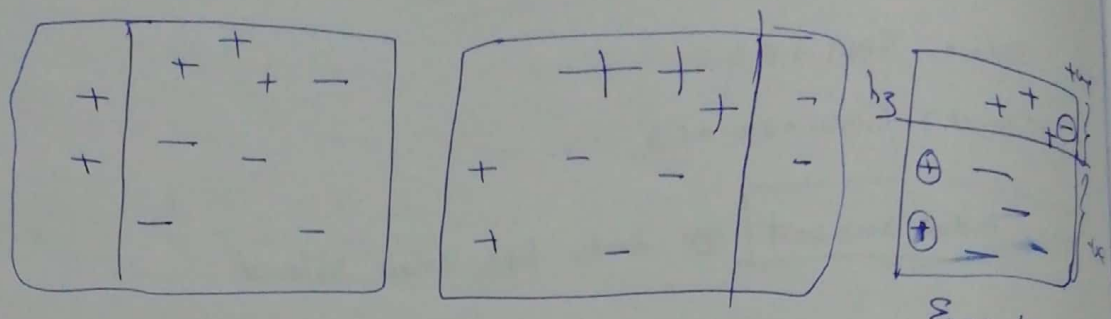




$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

The Third Round:



$\xi_3 = 0.14$

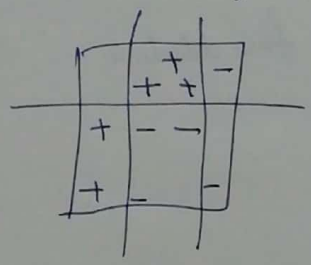
$\alpha_3 = 0.92$

The final classifier

$$\alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3 = F_3(x)$$

$$H_{final} = sign\left(\underset{0.42}{\alpha_1}\ \square + \underset{0.65}{\alpha_2}\ \square + \underset{0.92}{\alpha_3}\ \square\right)$$

$$=$$



$\alpha_i \rightarrow \gamma_i$

In the case of GBDT pseudo residuals (Computed from the negative of the gradient from loss function)

→ misclassified points are given more weightage

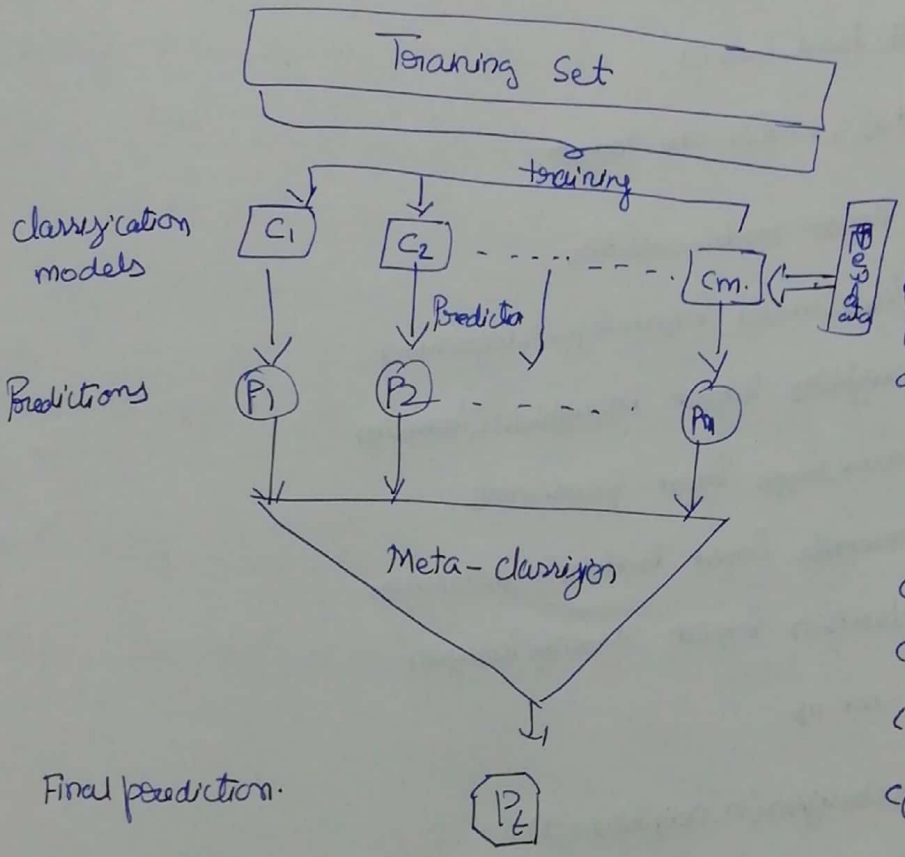
† At every stage we are adapting the model in the previous stage.

## 32.16 Stacked models

→ One of the form of Ensembling

→ https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

```
from mlxtend.classifier import StackingClassifier
```




ideally, more models are parallel & independent of each other.

$C_1$ : lrSVM

$C_2$ = RBFSVM

$C_3$ = GBT

$C_4$ : NB

$C_5$ = k-NN (k=5)

$C_6$ = k-NN (k=3)

### Algorithm

Input : Training Data $D = \{x_i, y_i\}_{i=1}^{m}$ $(x_i \in R^d, y_i \in Y)$

Output : An Ensemble classifier H

1: Step 1 : learn first-level classifiers.
2: for $t \leftarrow 1$ to T do
3: learn a base classifier $h_t$ based on D — $\{h_1, h_2, h_3 \dots h_T\}$ ↳ with good bias-var trade off
4: End for
5: Step 2 : Construct new data set from D
6: for $i \leftarrow 1$ m do
7: Construct a new dataset that contains $\{x_i', y_i\}$ $x_i' = [h_1(x), h_2(x) \dots h_m(x)]$
8: End for

9: Step3: learn a second-level classifier

10: Learn a new classifier h' based on the newly constructed data-set

11: return $H(x) = h'(h_1(x), h_2(x), \ldots h_T(x))$

Code : Pip3 install mlxtend

```
from sklearn import datasets
iris = datasets.load_iris()
X, y = iris.data[:, 1:3], iris.target
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
import numpy as np

clf1 = KNeighborsClassifier(n_neighbors = 1)
clf2 = RandomForestClassifier(random_state = 1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers = [clf1, clf2, clf3],
                          meta_classifier = lr)

print('3-fold Cross Validation: \n')

for clf, label in ([clf1, clf2, clf3, sclf], ['KNN',
                                              'RandomForest',
                                              'Naive Bayes',
                                              'Stacking Classifier']):
```

```
scores = model_selection.cross_val_score(clf, X, y, cv=3,
                                         scoring='accuracy')
print("Accuracy: %0.2f (+/- %0.2f) [%s]"
      % (scores.mean(), scores.std(), label))
```

O/P

Accuracy : 0.91 (+/- 0.01) [KNN]

" : 0.91 (+/- 0.06) [Random Forest]

" : 0.92 (+/- 0.03) [Naive Bayes]

: 0.95 (+/- 0.03) [Stacking Classifier]

-> Here Stacking classifier has the highest accuracy over the base learners

-> We can use probabilities?

```
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          use_probas=True,
                          average_probas=False,
                          meta_classifier=lr)
```

Key points

(1) Each of the base models $h_1(x)$, $h_2(x)$, $h_3(x)$, more different the models are, the better it is

(2) Kaggle: stacking is generally used

(3) Real world: cost of stacking is actually lot

Run/Evaluation ↑

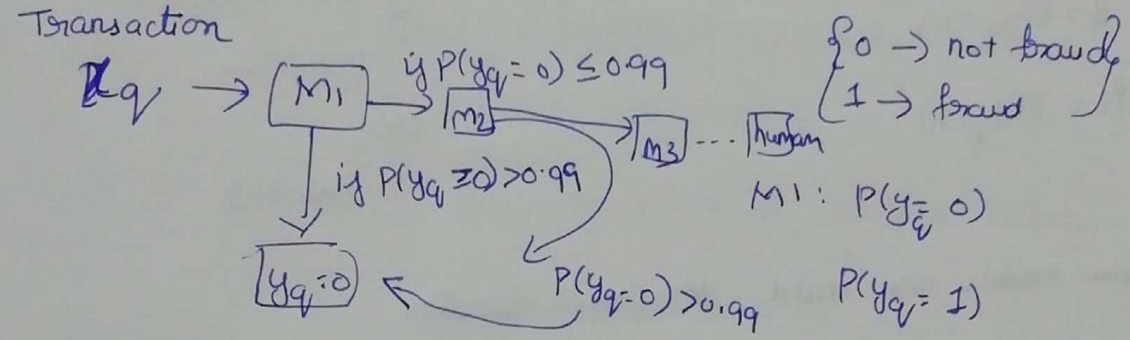
Training time ↑

In real world stacking being used less often than boosting (GB) & bagging (RF)

32.17 Cascading classifiers

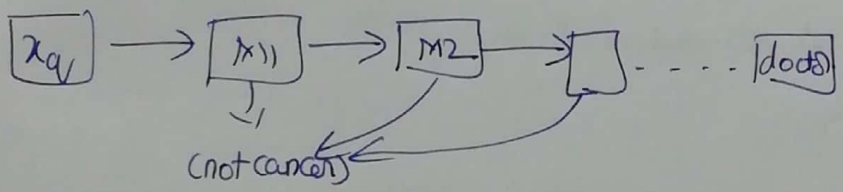
→ we have seen Bagging, boosting, stacking

→ Eg: Predicts Credit Card transaction is fraudulent or not.

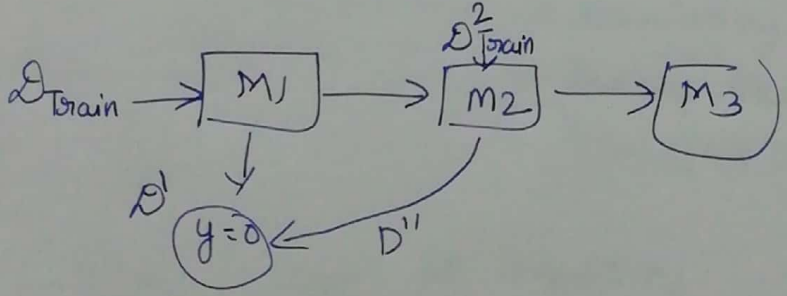



Cascade-model :-

Typically used when the cost of making a mistake is high




→ we should be slightly careful.



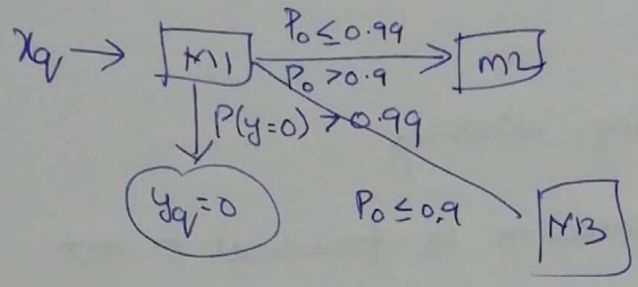

$D_{train} \rightarrow M_1$

$D^2_{train} \rightarrow M_2$

$D^2_{train} = D_{Train} - D'$ points

$D^3_{train} \rightarrow M_2$

$D^3_{train} \rightarrow D^2_{train} - D''$ points




$P_0 = P(y_q = 0)$

$P_1 = P(y_q = 1)$

If $P_0 > 0.99$

$y_q = 0$

Else

if $P_0 > 0.9$ & $P_0 \leq 0.99$

use M2

Else

use M3

→ Cascade models are build for fraud detection

→ In some case studies (20-levels) are used

team will work for years

medical / fraud.

32.18 Kaggle Vs Real World

Ensembles ÷ Kaggle, Competition.

(1) Kaggle cares about only one metric

Competition → 1st (log-loss, AUC, F1-score)
↳ 2nd

In the real world we have look at multiple metrics

→ primary metric
↳ Secondary metric.

Problem ÷ Improve Sales of Amazon.
↳ which metric we have reduce.

(1) (2) Business metric → Translate it to one or more ML-metrics.

In kaggle

(2) Very Complex Ensembles → to improve one metric

↑ impractical → low-latency
↳ Training time
↳ interpretability

(3) Kaggle ÷ great for Competitions

↳ data cleaning
data preprocessing
feature Engineering

## Random Forest

1) a) max_features :

- Auto/None
- Sqrt
- log2
- 0.2

b) n_estimators : higher the better

c) min_sample_leaf :

Smaller leaf makes the model more prone to capturing noise in train data. Generally I prefer a minimum leaf size of more than 50.

2. Features which will make the model training Easier :

n_jobs : This parameter tells the Engine how many processors is it allowed to use

'-1' : no restriction

1 : use 1 one processor

Random state :

oob_score : This is a random forest CV method. It is similar to leave one out validation technique, however this is so much faster.

# Gradient Boosting

## Tree Specific Parameters

→ min_samples_split
- ↳ used to control overfitting
- ↳ higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- ↳ To high will lead to underfitting.

→ min_samples_leaf / minimum_weight_fraction_leaf :
- ↳ Control overfitting
- ↳ Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

→ max_depth / max_leaf_nodes
- ↳ Control overfitting
- ↳ higher depth will allow model to learn relations very specific to a particular sample
- ↳ should be tuned using CV
- ↳ If max_leaf_nodes are defined, it will ignore max_depth.

→ max_features
- ↳ sqrt
- ↳ log3
- ↳ Auto
- ↳ 0.3 & 0.4

↳ higher values lead to overfitting

## Boosting Parameters

→ Learning rate

- This determines the impact of each tree on the final outcome
- GBM works by starting with an initial estimate which is updated using the output of each tree.
- The learning parameter controls the magnitude of this change in the estimates.
- Lower values are preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize ~~that~~ well
- Lower values would need higher number of trees. Computationally expensive.

### n_estimators

- The # of sequential trees to be modeled
- Though the GBM is fairly robust at higher # of trees but it can still overfit a point
- CV should be used for tuning

### Subsample

- The fraction of observations to be selected for each tree. Selection is done by random sampling
- Typical values ~0.8 generally work fine but can be fine tuned further.

Fix learning rate & number of estimators for tuning tree based parameters

min_sample_split : 500 : Should be ~0.5 to 1% of total values. Since this is a imbalanced class problem, we'll take a small value from the range.

min_samples_leaf : 50 : Can be selected based on intuition

max_depth : 8 : Should be chosen (5-8) based on # of observations & predictors. This has 87K rows & 49 columns.

max_features : Sqrt

Subsample : 0.8 Commonly used start value.

# XGBoosting

## Boosting Parameters

### eta

* Analogous to learning rate in GBM
* Makes the model more robust by shrinking the weights on each step
* Typically final values to be used: 0.01 → 0.2

### min_child_weight

* Defines the minimum sum of weights of all observations required in a child
* This is similar to min_child_leaf in GBM but not exactly. This refers to min "sum of weights" of observations while GBM has min "number of observation"

→ ↑ overfitting

→ ↑↑ underfitting (use CV)

### Max_depth / max_leaf_nodes

→ Typically 3 to 10

### Max_leaf nodes / max_depth

→ maximum number of terminal nodes or leaves in a tree.

→

### gamma

→ A node is split only when resulting split gives a positive reduction in the loss function.

Gamma specifies minimum loss reduction required to make a split.

→ Make algo conservative

→ Can vary depending on the loss function.

Subsample

Typically : 0.5 - 1

Colsample_bytree

- Similar to max_features in GBM

typically : 0.5 - 1

lambda

→ L2 regularisation term on weights (Ridge Regression)

→ This used to handle the regularisation part of XGBoost.

Though many datascientists don't use it often, it should be explored to reduce overfitting

alpha

→ L1 regularisation term on weight (Lasso regression)

→ Can be use in case of very high dimensionality so that the algorithm run faster when implemented

Scale_pos_weight

A value greater than 0 should be used in case of high class imbalance as it helps in faster convergence.

## Learning Task Parameters:

These parameters are used to define the optimization objective the metric to be calculated at each step.

1. Objective [default = reg: linear]

This defines the loss function to be minimized. Mostly used values are

* binary: logistic : logistic regression, returns predicted probability (not class)
* multi: softmax : multiclass classification using the softmax objective, returns predicted class (not probabilities)
* multi: softprob : same as softmax, but returns predicted probability of each data point belonging to each class.

2. eval_metric : [default according to objective]

→ The metric to be used for validation data
→ The default values are rmse for regression & error for classification
→ Typical values are

→ rmse
→ mae
→ logloss ⊖ : negative log-likelihood
→ error : Binary classification error rate (0.5 threshold)
→ merror : Multiclass classification error rate
→ mlogloss : " logloss
→ auc : area under curve.

3. Seed

* The random number seed.
* Can be used for generating reproducible results and also for parameter tuning

# Pipe line

```
from sklearn.feature_selection import ColumnSelector
from sklearn.pipeline import make_pipeline

pipe1 = make_pipeline (ColumnSelector (cols = (0, 2)), LogisticRegression())
pipe2 = "          ("          (cols = (1, 2, 3)),          ")

sclf = StackingClassifier (classifiers = [pipe1, pipe2],
                           meta-classifier = LogisticRegression())

sclf.fit (X, y)
```