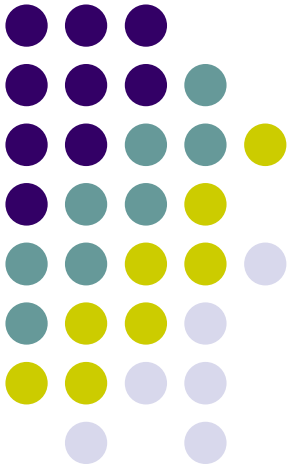
# *Big Data Analytics*

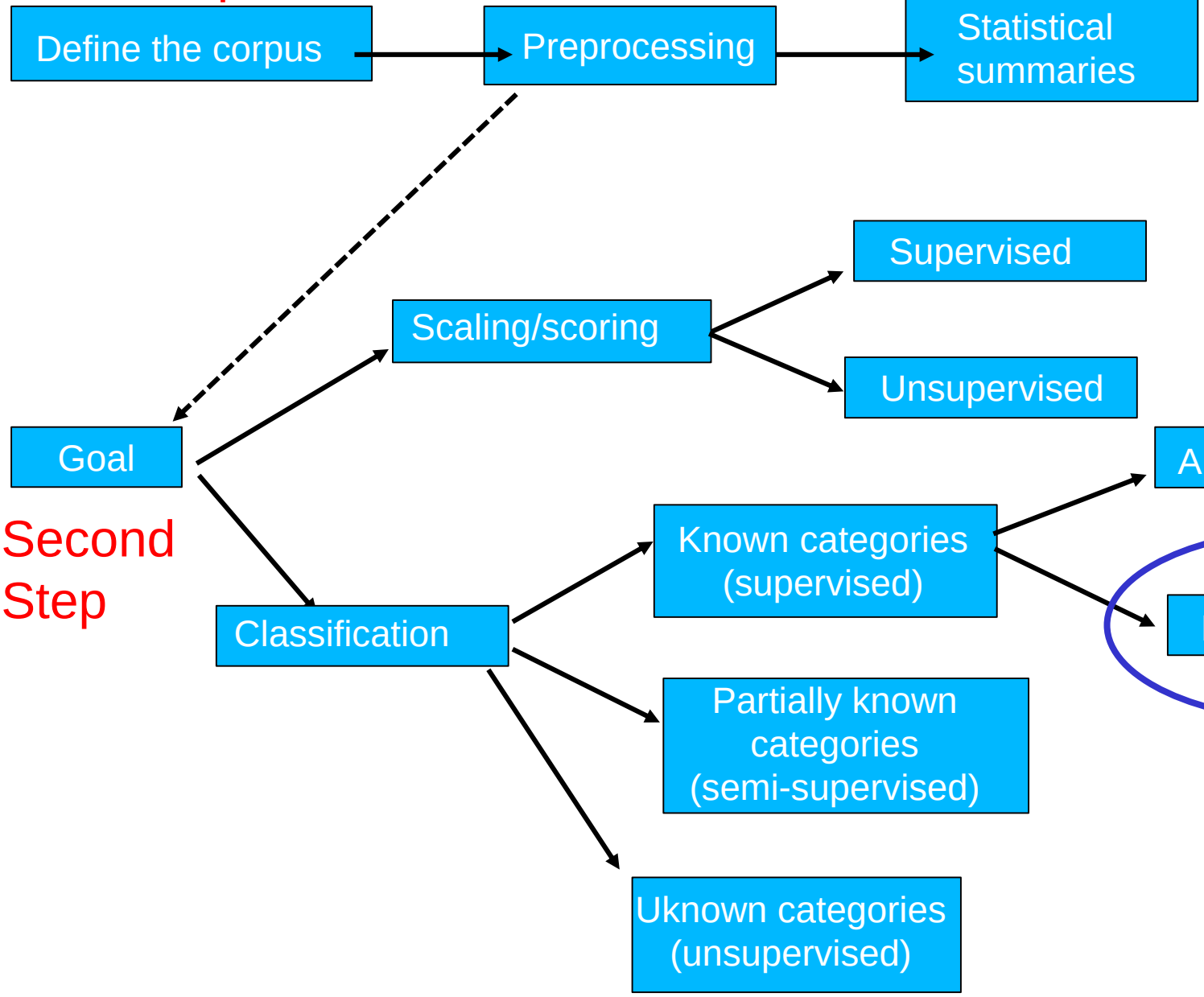
Lecture 9 (part 1)

Supervised classification methods:

A review of (some) Machine Learning Algorithms (third part)



First Step


Define the corpus
Preprocessing
Statistical summaries
Goal
Scaling/scoring
Supervised
Unsupervised
Second Step
Classification
Known categories (supervised)
Automatic tagging
Human tagging
Partially known categories (semi-supervised)
Uknown categories (unsupervised)

# References


- ✓ Olivella, Santiago, and Shoub Kelsey (2020). Machine Learning in Political Science: Supervised Learning Models. In Luigi Curini and Robert Franzese (eds.), *SAGE Handbook of Research Methods is Political Science & International Relations*, London, Sage, chapter 56

# ML

Several different possible machine learning algorithms are available out there

We will offer an intuitive introduction to the following further two ML algorithms:

1. Regularized regression
2. Gradient Boosting

# Regularized regression

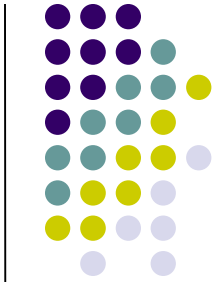
To understand the logic behind a regularized regression, let's start to think about an OLS. Assume we have:

- ✓ $i = 1, 2, \ldots, N$ observations
- ✓ Each observation $i$ presents a given value of $y_i$ (our DV)
- ✓ $j = 1, 2, \ldots, J$ independent variables (our IV)
- ✓ And $x_{ij}$ as the value of variable $j$ in observation $i$

We could build a linear regression model using the values of $\beta_0, \beta_1, \ldots, \beta_J$ that minimize the usual Sum of Squared Residuals (RSS):

$$RSS = \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{J} \beta_j x_{ij} \right)^2 \; where \; \beta_0 + \sum_{j=1}^{J} \beta_j x_{ij} = \widehat{y_i}$$

# Regularized regression

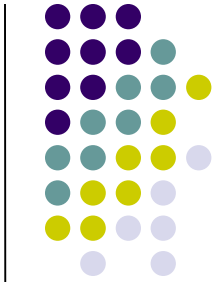


However, for many real-life data sets (and definitely for texts!) we have very *wide* data, meaning we have a large number of features, i.e., IVs (*J)* that we believe are informative in predicting some outcome

But as *J* increases, we can quickly violate some of the OLS assumptions and we require alternative approaches to provide predictive analytic solutions

Specifically, as *J* increases…

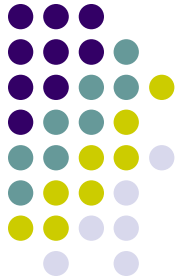
# Regularized regression



- ✓ If $J > N$, OLS estimates are not unique. Moreover, quite often the result will be computationally infeasible
- ✓ Even with $N > J$, if $J$ increases significantly (with respect to $N$) we are more likely to capture multiple features that have some multicollinearity. Coefficients for correlated features become over-inflated and can fluctuate significantly

One consequence of these large fluctuations in the coefficient terms is **overfitting**, which means we have high variance in our usual bias-variance tradeoff space

# Regularized regression



What can we do? Add a penalty, such that we now minimize:

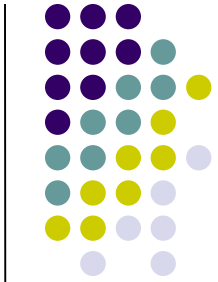
$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{J}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{J}\beta_j^2 \quad \rightarrow \text{ridge regression}$$

OR

$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{J}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{J}\left|\beta_j\right| \quad \rightarrow \text{lasso regression}$$

where $\lambda$ *(lambda)* is the penalty parameter (to be estimated)

# Regularized regression



Why adding a penalty?

**Penalty parameters** constrain the size of the coefficients such that the only way the coefficients can *increase* is **iff** we experience a *comparable decrease* in the RSS

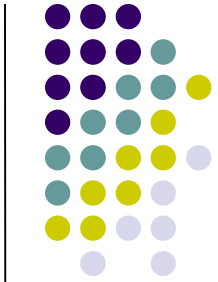
This provides more clarity in identifying the true signal in our model. As a result:

- ✓ It reduces the magnitude and fluctuations of the coefficients and will reduce the variance of our model
- ✓ It also identifies the model if J > N

Moreover…

- ✓ Non-important features are pushed closer to zero, reducing the noise in our data. And actually, some coefficients can also become zero (allowing *feature selection…*)

# Regularized regression

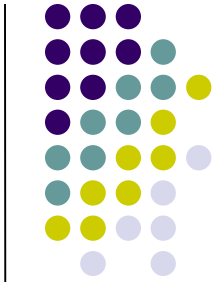


The penalty can take different forms:

*Ridge regression*: $\lambda \sum_{j=1}^{J} \beta_j^2$

- ✓ A ridge model will retain all variables (although the coefficient of some features will be greatly constrained, shrinking some of them towards zero…but not exactly to zero! Therefore a ridge regression do**es not perform any feature selection**)
- ✓ A ridge model is good if you believe there is a need to retain all features in your model, yet you still want to reduce the noise that less influential variables may create and minimize multicollinearity

# Regularized regression



The penalty can take different forms:

*Lasso*: $\lambda \sum_{j=1}^{J} |\beta_j|$

✓ Unlike ridge, the lasso will actually push coefficients to zero and **perform feature selection**

✓ This simplifies and automates the process of identifying those feature most influential to predictive accuracy

✓ If greater interpretation is necessary where you need to reduce the signal in your data to a smaller subset, then a lasso model may be preferable

✓ However, there is always a trade-off: when we remove features we can sacrifice the overall level of accuracy of the model (its prediction power)

# Regularized regression

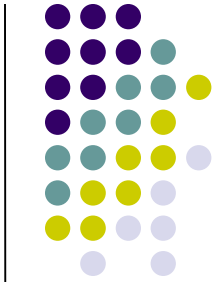
*Elastic net*

$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{J}\beta_j x_{ij}\right)^2+\lambda_1\sum_{j=1}^{J}\beta_j^2+\lambda_2\sum_{j=1}^{J}|\beta_j| \rightarrow \text{elastic net regression}$$

Best of both world?

- ✓ ridge regression penalty is more effective in systematically reducing correlated features together
- ✓ lasso models perform feature selection, but when two strongly correlated features exist, one may be pushed fully to zero while the other remains in the model. Furthermore, the process of one being in and one being out is not very systematic
- ✓ The advantage of the elastic net model is that it enables effective regularization via the ridge penalty with the feature selection characteristics of the lasso penalty

# Regularized regression

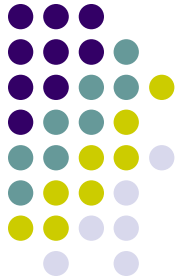


How to find best value of $\lambda$? Via cross-validation of course!

✓ Now, let's go back to text-analytics and classification: in this case, rather than minimizing the RSS, you want to minimize the *miss-classification error* and running a binomial (or multinomial) model (according to the type of DV you have)

✓ But the logic remains the same as discussed earlier!

# Gradient Boosting classifier

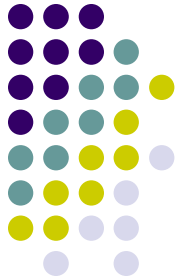


Several supervised machine learning models are founded on a single predictive model (i.e. naive Bayes, support vector machines, RR, etc.)

Alternatively, other approaches such as bagging and random forests are built on the idea of building an *ensemble of models* where each individual model predicts the outcome and then the ensemble simply averages the predicted values

The family of **boosting methods** is based on a different, constructive strategy of *ensemble formation*

# Gradient Boosting classifier



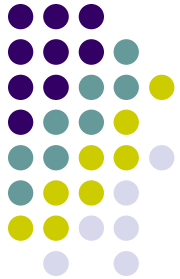
The main idea of boosting is to add new models to the ensemble ***sequentially***

At each particular iteration, as we will see, a *new weak, base-learner model* (what's that??? Give me a moment…) is trained with respect to the error of the whole ensemble learnt so far

Boosting is a framework that iteratively improves *any* weak learning model

In practice however, boosted algorithms almost **always use decision trees as the base-learner**

# Gradient Boosting classifier

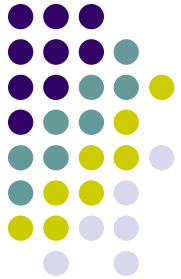


Whereas Random Forests build an ensemble of *deep independent trees*, **Gradient Boosting Machines** (GBM) build an ensemble of shallow and weak successive trees with **each tree learning and improving on the previous**

When combined, these many weak successive trees produce a powerful “committee” that are often hard to beat with other algorithms (in particular when you have a training-set relatively large in terms of documents)

Let’s see how

# Gradient Boosting classifier



**Training weak models**:

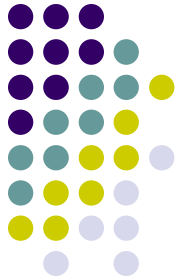
A *weak model* is one whose error rate is only slightly better than random guessing

The idea behind boosting is that each sequential model builds a simple weak model to slightly improve the remaining errors

With regards to decision trees, **shallow trees** represent a weak learner. Commonly, trees with only 1-6 splits are used

Combining many weak models (versus strong ones) has several benefits

# Gradient Boosting classifier

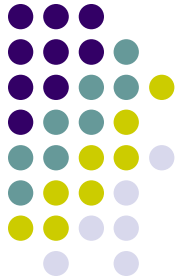


*Speed*: Constructing weak models is computationally cheap

*Accuracy improvement*: Weak models allow the algorithm to *learn slowly*; making minor adjustments in new areas where it does not perform well. In general, statistical approaches that learn slowly tend to perform well

*Avoids overfitting*: Due to making only small incremental improvements with each model in the ensemble, this allows us to stop the learning process as soon as overfitting has been detected (typically by using cross-validation)

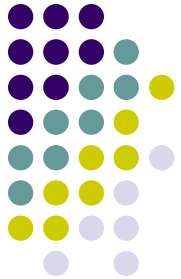
# Gradient Boosting classifier



Boosted trees are **grown therefore sequentially**; each tree is grown using information from previously grown trees

The basic algorithm for boosted classification trees can be generalized to the following where *x* represents our features and *y* represents our response…

# Gradient Boosting classifier

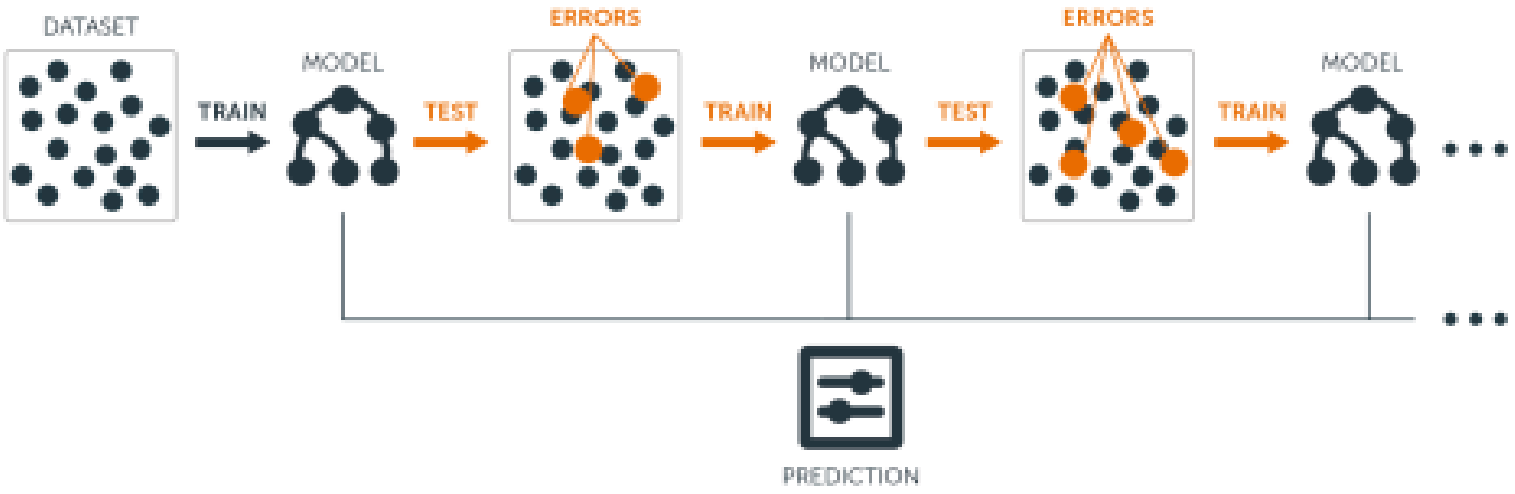


1. Fit a decision tree to the data: $F_1(x)=y$
2. We then fit the next decision tree to the classification errors of the previous: $h_1(x)=y-F_1(x)$

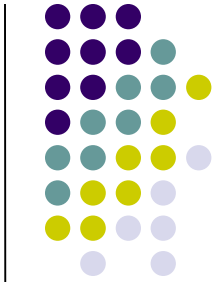
Note: the existing trees in the model do not change when a new tree is added. The added decision tree fits the residuals from the current model!

3. Add this new tree to our algorithm: $F_2(x)=F_1(x)+h_1(x)$
4. Fit the next decision tree to the classification errors of $F_2$: $h_2(x)=y-F_2(x)$
5. Add this new tree to our algorithm: $F_3(x)=F_2(x)+h_2(x)$
6. This process repeats until we have made the maximum number of trees specified or the residuals get super small

# Gradient Boosting classifier



# Gradient Boosting classifier



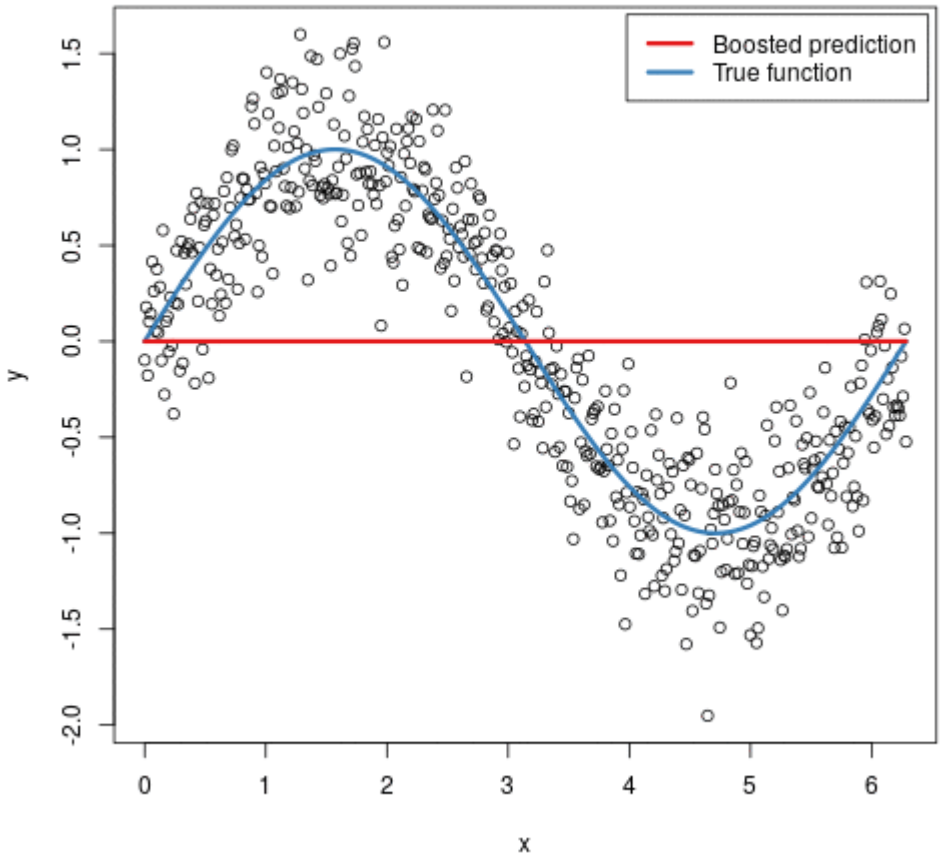
To illustrate the behavior, assume the following situation

The blue sine wave represents the true underlying function and the points represent observations that include some irreducible error

The boosted prediction (in red) illustrates the adjusted predictions after each additional **sequential tree** is added to the algorithm

Initially, there are large errors which the boosted algorithm improves upon immediately but as the predictions get closer to the true underlying function you see each additional tree make small improvements in different areas across the feature space where errors remain

# Gradient Boosting classifier

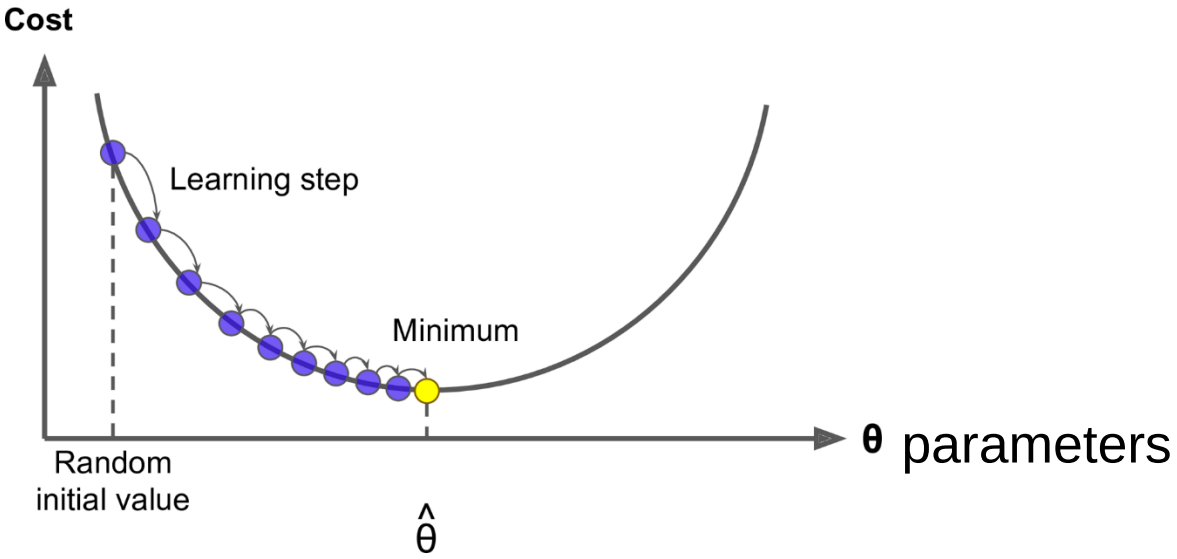


# Gradient Boosting classifier

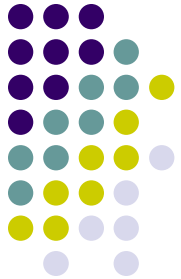
Gradient boosting is considered a ***gradient descent*** algorithm

Gradient descent is a very generic **optimization algorithm** capable of finding optimal solutions to a wide range of problems (employed also in NN and deep neural algorithms but w/o backpropagation in the GB case)

The general idea of gradient descent is to tweak parameters iteratively in order to minimize a cost function



# Gradient Boosting classifier



Gradient descent can be performed on any loss function that is differentiable (for example the mean squared error loss function, the mean absolute error, deviance, etc.)

Consequently, this allows GBMs to optimize different loss functions as desired

# Gradient Boosting classifier

When we are dealing with **classification**, such as classifying a news as positive or negative, we'll require a loss function that helps us understand how accurate our model is at classifying news who are or not positive

Typically, we either focus on *classification error rate* (calculated as (# wrong cases) / (# all cases)) or more often on the *log-loss function* (i.e., *-1 * the log of the likelihood function* - lower loss scores are therefore better)

This latter loss function, can also be expressed (when our predictions are in terms of log(odds) values) as the following: (*Observed value – Predicted value= PseudoResidual*)

Our aim, throughout our sequence of weak models, is to find the minimum of our loss function

# Gradient Boosting classifier

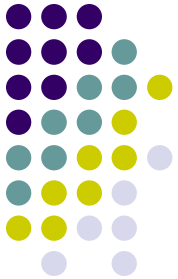
Let’s go back to our example about American/Japanese citizens

Our training set:

*Weight (lbs.)/Sex/Nationality*

195 M American
190 M American
160 F American
165 F American
165 M Japanese
160 M Japanese
130 F Japanese
140 F Japanese

# Gradient Boosting classifier



| *Nationality* | *Class* | *Gender* | *Weight* |
|---|---|---|---|
| American | 0 | M | 195 |
| American | 0 | M | 190 |
| American | 0 | F | 160 |
| American | 0 | F | 165 |
| Japanese | 1 | M | 165 |
| Japanese | 1 | M | 160 |
| Japanese | 1 | F | 130 |
| Japanese | 1 | F | 140 |

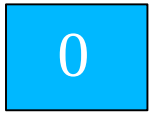
# Gradient Boosting classifier

*Step 1: Initialize model with a constant value*

Typically this implies starting with one leaf node that predicts the initial value for every citizen in our training-set

For a classification problem, it will be the log(odds) of the target value. Since four citizens in our case are Japanese, and four are Americans, the log(odds) that a citizen is Japanese would be: log(Japanese/American)=log(4/4)=log(1)=0

This becomes our initial leaf




*Initial leaf node*

# Gradient Boosting classifier

*Step 1: Initialize model with a constant value*

We can then convert the log(odds) to a probability:

$$\Pr(Japanese) = \frac{exp^{\log(odds)}}{1+exp^{\log(odds)}} = \frac{exp^{\log(0)}}{1+exp^{\log(0)}} = 0.5$$

Note that such outcome is the equivalent of the average in a classification problem, i.e., Japanese/Total=4/8=0.5

# Gradient Boosting classifier

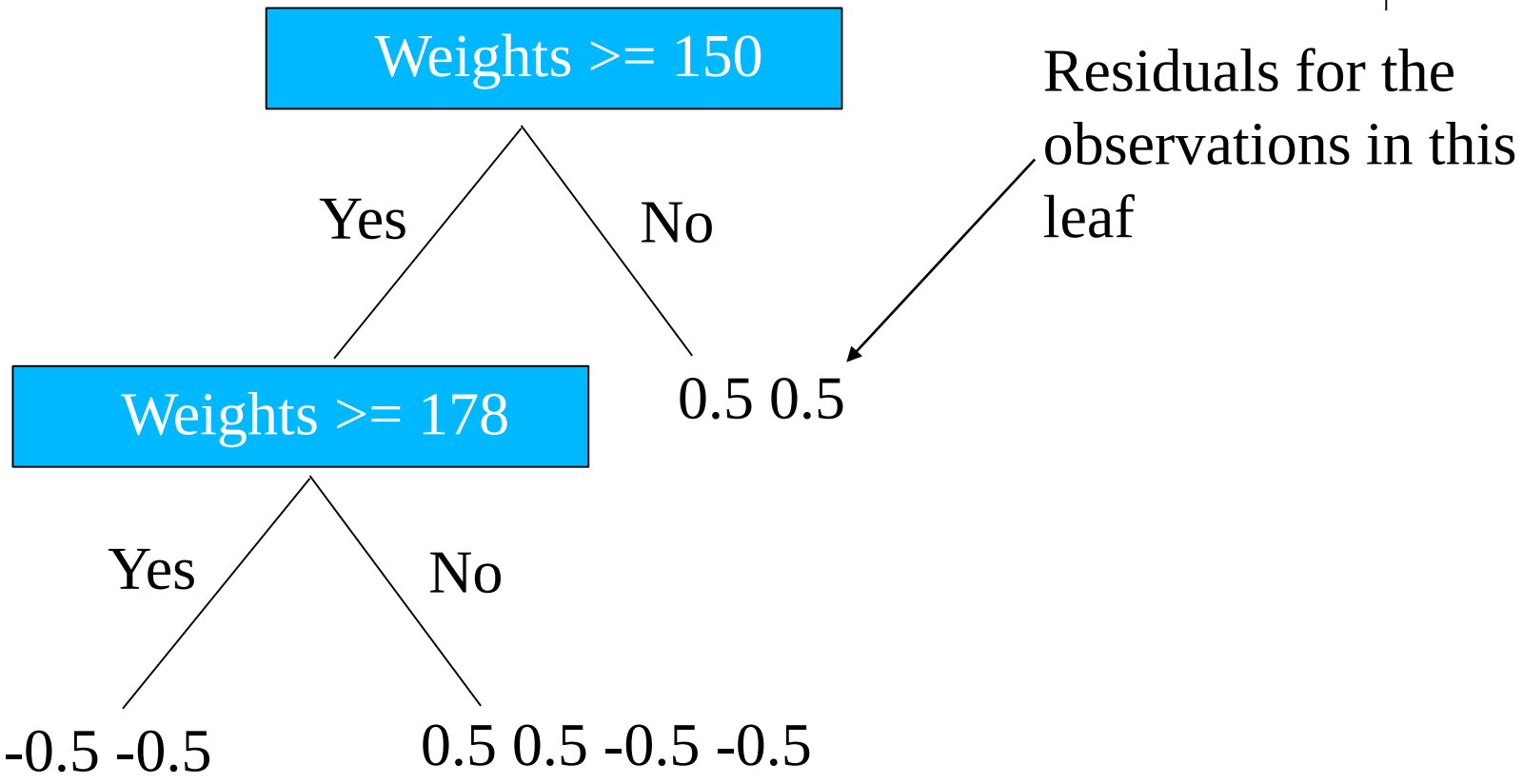
*Step 2: Estimate the (pseudo)residuals*

| Nationality | Class | Gender | Weight | Prediction | Residual |
|---|---|---|---|---|---|
| American | 0 | M | 195 | 0.5 | -0.5 |
| American | 0 | M | 190 | 0.5 | -0.5 |
| American | 0 | F | 160 | 0.5 | -0.5 |
| American | 0 | F | 165 | 0.5 | -0.5 |
| Japanese | 1 | M | 165 | 0.5 | 0.5 |
| Japanese | 1 | M | 160 | 0.5 | 0.5 |
| Japanese | 1 | F | 130 | 0.5 | 0.5 |
| Japanese | 1 | F | 140 | 0.5 | 0.5 |

Summating the loss function, i.e. if we add up the Loss Function for each observed value (in absolute value), we get 4

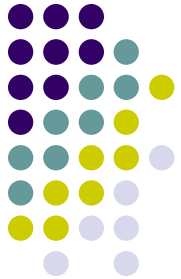
# Gradient Boosting classifier

*Step 3: Add the first weak learner model*




We use a limit of two leaves here to simplify our example, but in reality, Gradient Boost has as a default value (that can be changed) of 6 leaves

# Gradient Boosting classifier

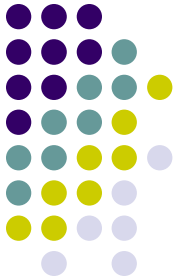


Because of the limit on leaves, one leaf can have multiple values (as it happens in our case)

Moreover, predictions are in terms of log(odds) but these leaves contain residuals

So, we can't just add the single leaf we got earlier and this tree to get new predictions because they're derived from different sources

# Gradient Boosting classifier



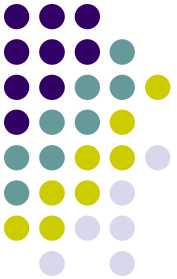
We have to use some kind of transformation

The most common form of transformation used in Gradient Boost for Classification is:

$$\frac{\sum Residual}{\sum[Previous\ Prob*(1-Previous\ Prob)]}$$

The numerator in this equation is sum of residuals in that particular leaf; the denominator is sum of (previous prediction probability for each residual ) * (1 - same previous prediction probability).

# Gradient Boosting classifier



The first leaf has two residual values that are 0.5, and since this is the first tree, the previous probability will be the value from the initial leaf, thus, same for all residuals

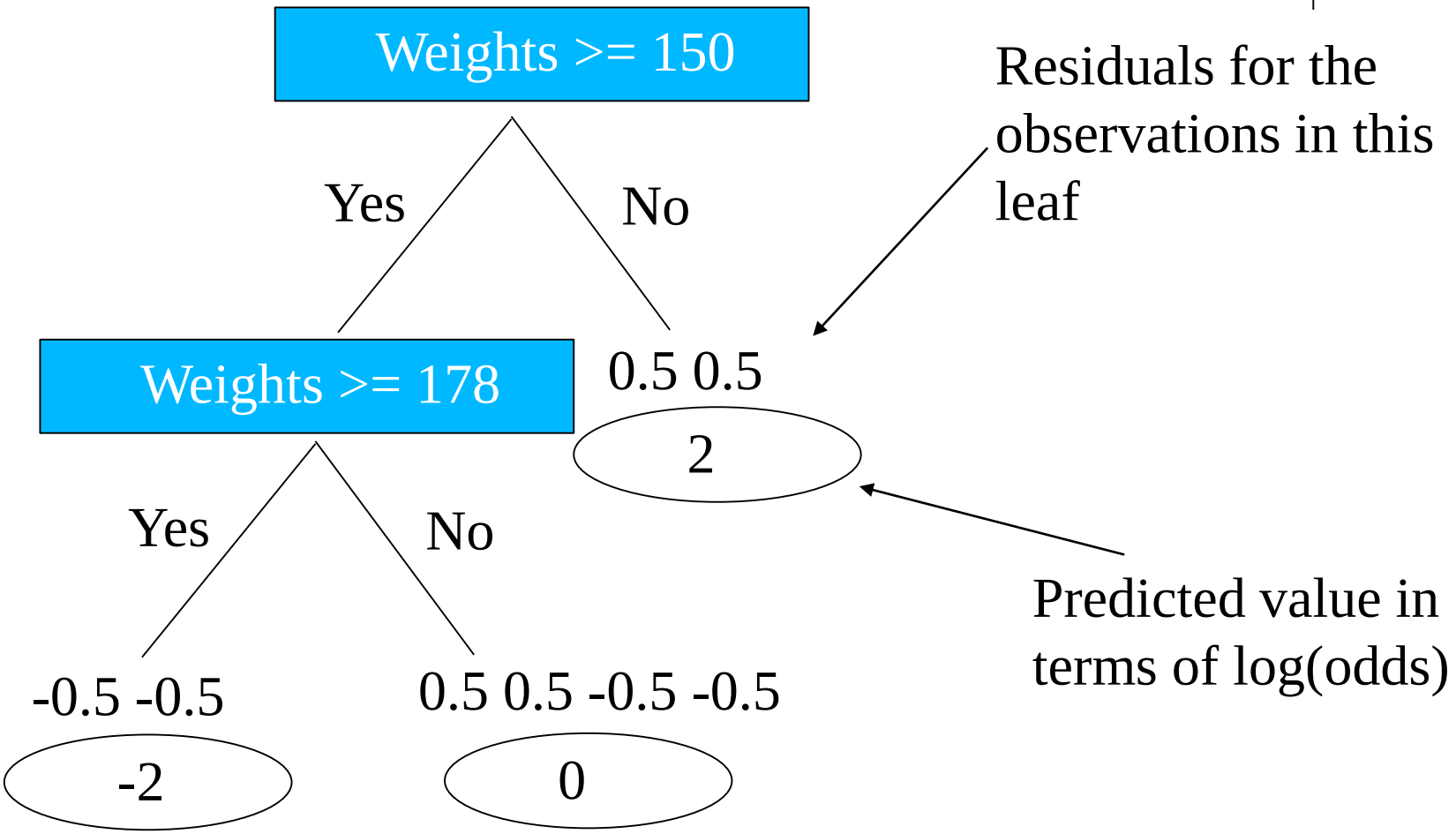
Hence: $\frac{0.5+0.5}{(0.5*(1-0.5)+(0.5*(1-0.5))}=2$

For the second leaf: $\frac{-0.5-0.5}{(0.5*(1-0.5)+(0.5*(1-0.5))}=-2$

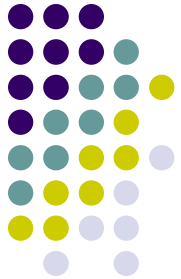
For the third one: $\frac{-0.5-0.5+0.5+0.5}{4*(0.5*(1-0.5))}=0$

# Gradient Boosting classifier

Now the transformed tree looks like:



# Gradient Boosting classifier



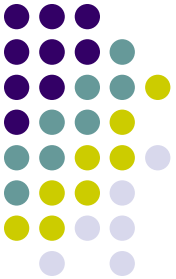
We can now add our initial leaf with our new tree (together with a *learning rate*): *Old Tree + New Tree * Learning Rate*

The *Learning Rate* is used to scale the contribution from the new tree

Empirical evidence has proven that taking lots of small steps (in the right direction) results in better prediction with a testing dataset, i.e., the dataset that the model has never seen

Learning Rate is usually a small number like 0.1

# Gradient Boosting classifier



We can now calculate new log(odds) prediction and hence a new probability

For example, for the first two citizens, Old Tree = 0. Learning Rate which remains the same for all records is equal to 0.1 and by scaling the new tree, we found above its value to be 2. Hence, substituting in the formula we get: 0+2*0.1=0.2

In terms of probabilities, this log(odds) translates into:

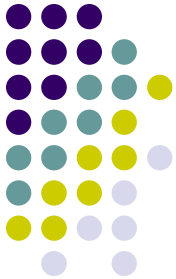
$$\Pr(Japanese) = \frac{exp^{\log(odds)}}{1+exp^{\log(odds)}} = \frac{exp^{\log(0.2)}}{1+exp^{\log(0.2)}} = 0.55$$

Similarly, we substitute and find the new log(odds) for each citizens and hence find the probability

Using the new probability, we can now calculate the new residuals

# Gradient Boosting classifier

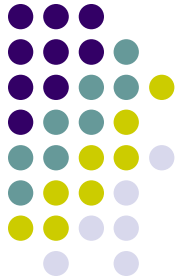
*Step 4: Estimate the new (pseudo)residuals*



| Nationality | Class | Gender | Weight | Prediction | Residual | Prediction2 | Residual2 |
|---|---|---|---|---|---|---|---|
| American | 0 | M | 195 | 0.5 | -0.5 | 0.45 | -0.45 |
| American | 0 | M | 190 | 0.5 | -0.5 | 0.45 | -0.45 |
| American | 0 | F | 160 | 0.5 | -0.5 | 0.5 | -0.5 |
| American | 0 | F | 165 | 0.5 | -0.5 | 0.5 | -0.5 |
| Japanese | 1 | M | 165 | 0.5 | 0.5 | 0.5 | 0.5 |
| Japanese | 1 | M | 160 | 0.5 | 0.5 | 0.5 | 0.5 |
| Japanese | 1 | F | 130 | 0.5 | 0.5 | 0.55 | 0.45 |
| Japanese | 1 | F | 140 | 0.5 | 0.5 | 0.55 | 0.45 |

Summating the loss function, i.e. if we add up the Loss Function for each observed value (in absolute value), we get now 3.8. We have improved! Now let's add a second weak tree, let's predict, let's get new residuals; and then let's add a third weak tree, etc.

# Gradient Boosting classifier

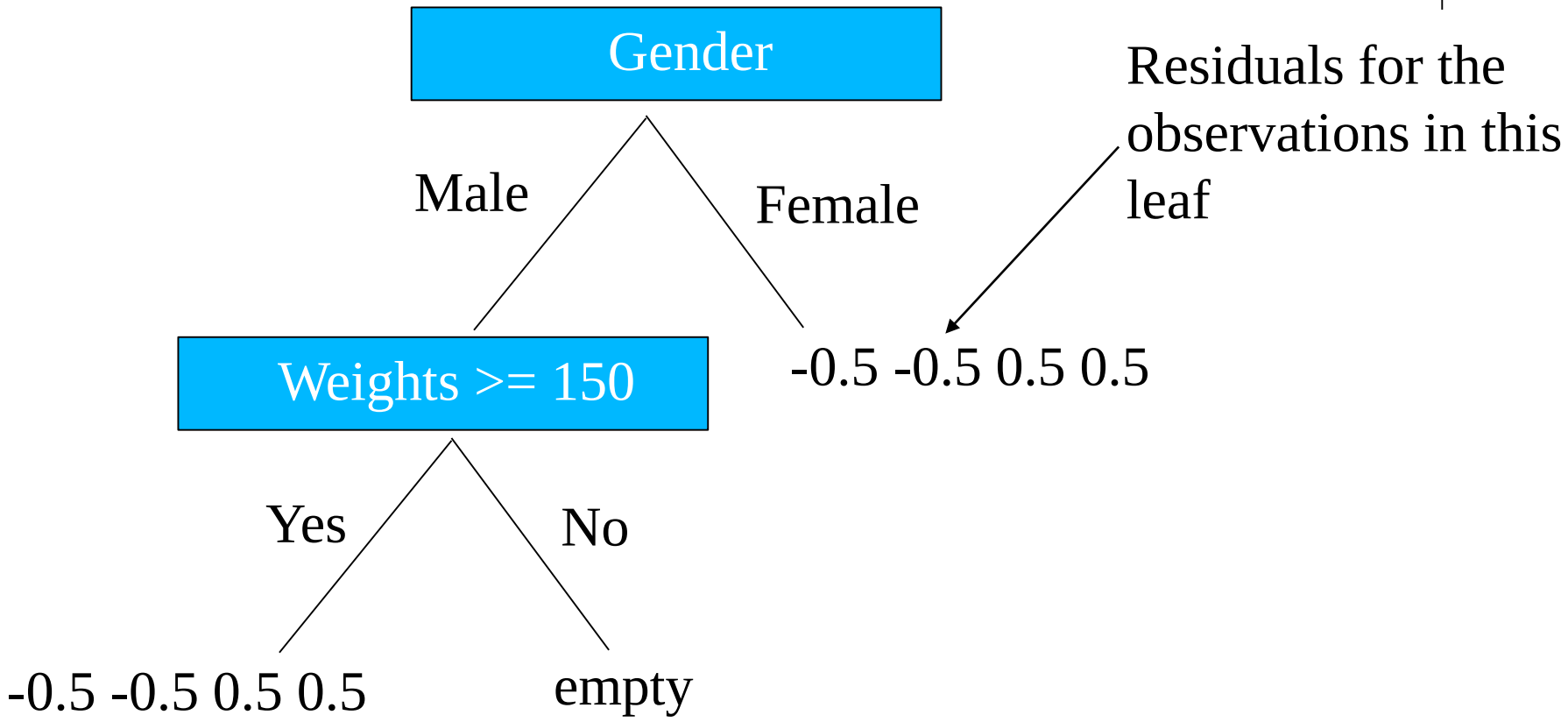


The challenge of the gradient descent algorithm is that each time we run a weak learner model, this must decrease the loss function (we want to minimize it!)

For example, compare this weak tree with the previous one

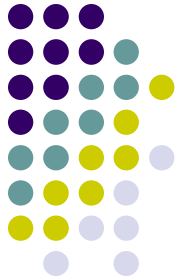
# Gradient Boosting classifier

*Step 3: Add the first weak learner model*




Clearly this weak learner model cannot improve our loss function (given the new log(odds) for each predictions is still 0, given that $\frac{-0.5-0.5+0.5+0.5}{4*(0.5*(1-0.5))}$=0), so it won't be selected…

# Gradient Boosting classifier



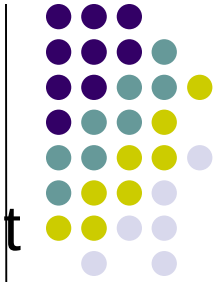
GBC are computationally expensive - they often require many trees (>1000) which can be time and memory exhaustive

The high flexibility results in many parameters that interact and influence heavily the behavior of the approach (number of iterations, tree depth, regularization parameters, etc.)

This requires a large *grid search* during tuning to find the best combination of hyperparameters given a specific task at hand

We will discuss about it in the Lab class

# Gradient Boosting classifier

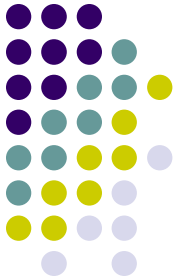


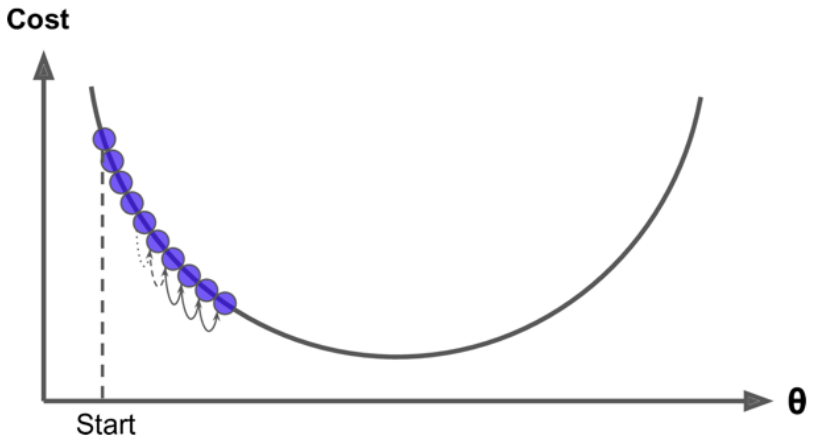
For example: an important parameter in gradient descent is the size of the steps which is determined by the *learning rate* (eta) discussed above. It controls how quickly the algorithm proceeds down the gradient descent

If the learning rate is too small, the algorithm will take many iterations to find the minimum (this reduces the chance of overfitting but also increases the time to find the optimal fit)

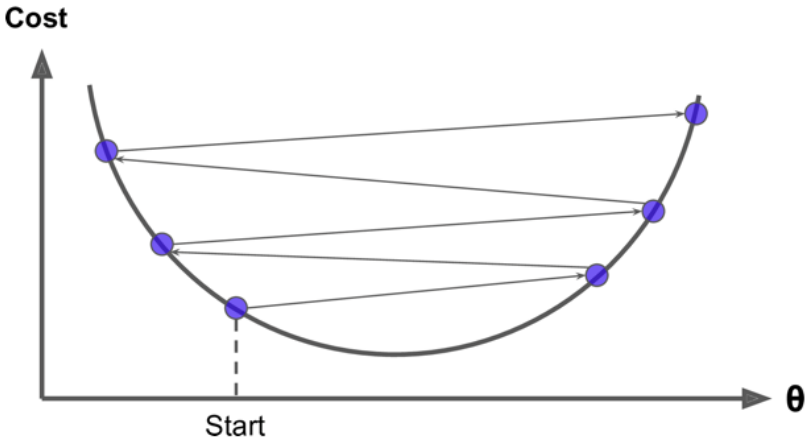
On the other hand, if the learning rate is too high, you might jump cross the minimum and end up further away than when you started

# Gradient Boosting classifier



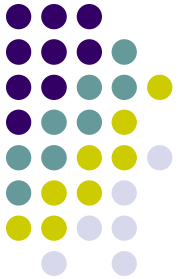



a) too small



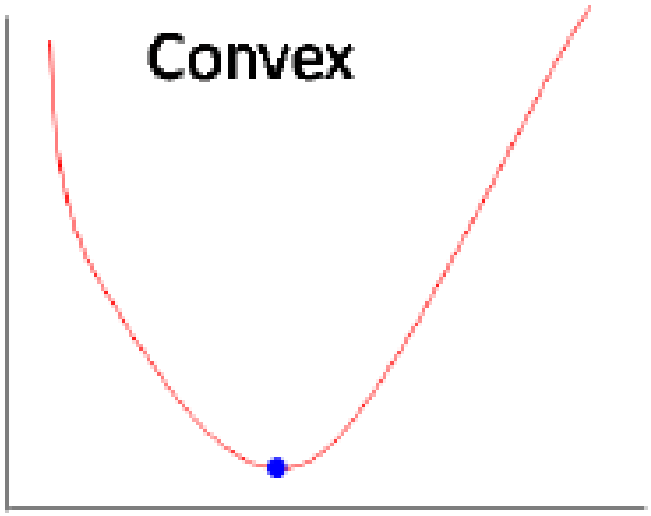

a) too big

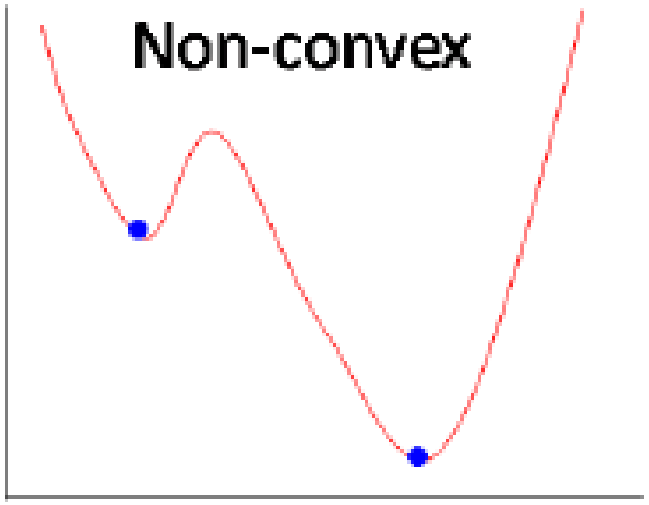
# Gradient Boosting classifier



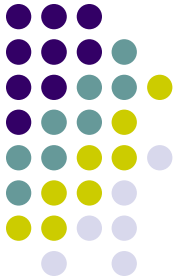
Moreover, not all cost functions are convex (bowl shaped)

There may be local minimas, plateaus, and other irregular terrain of the loss function that makes finding the global minimum difficult



# Gradient Boosting classifier



***Stochastic gradient descent*** can help us address this problem by sampling a fraction of the training observations (typically without replacement) and growing the next tree using that subsample

This makes the algorithm faster. Moreover, the stochastic nature of random sampling also adds some random nature in descending the loss function gradient

Although this randomness does not necessarily allow per-se the algorithm to find the absolute global minimum, it can actually help the algorithm jump out of local minima and off plateaus and get near the global minimum

# R pakcages to install

*install.packages("glmnet", repos='http://cran.us.r-project.org')*

*install.packages("xgboost", repos='http://cran.us.r-project.org')*

*install.packages("Ckmeans.1d.dp", repos='http://cran.us.r-project.org')*