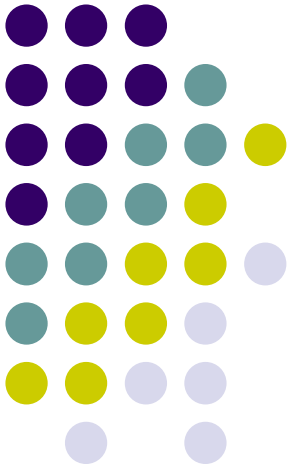
Big Data Analytics

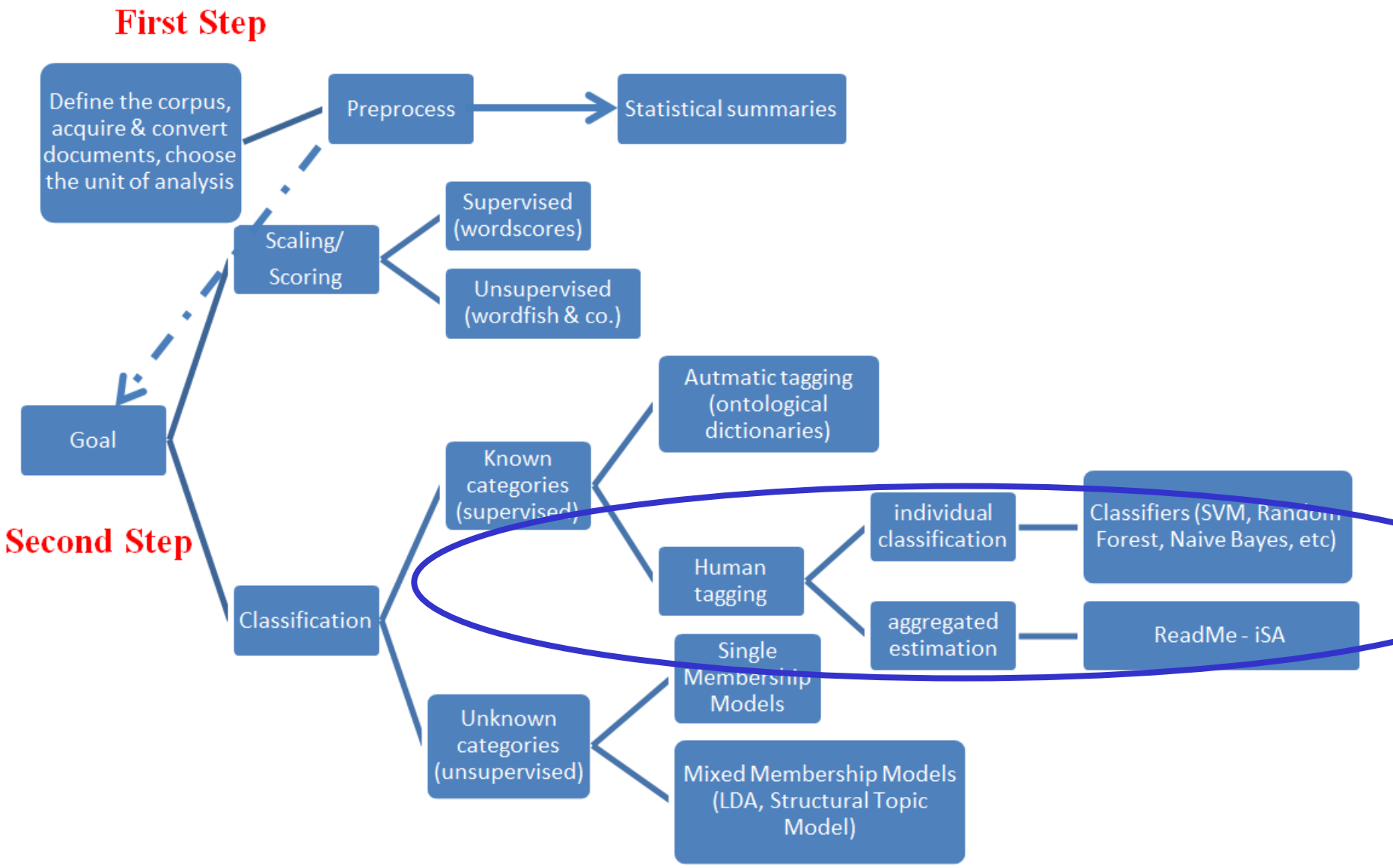
Lecture 9 (part 1)

Supervised classification methods:

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



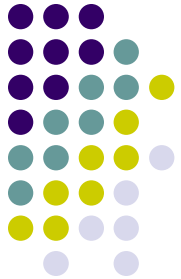
Our Course Map



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

Individual methods

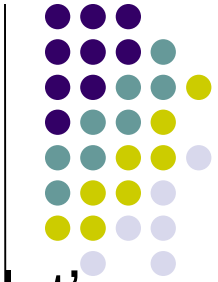


Several different possible machine learning algorithms are available out there

Today we will offer an intuitive introduction to the following two further algorithms:

- Regularized regression
- Gradient Boosting

Regularized regression



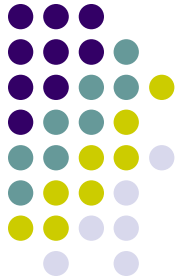
To understand the logic behind a regularized regression, let's start to think about an OLS. Assume we have:

- ✓ $i = 1, 2, \ldots, N$ 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):

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

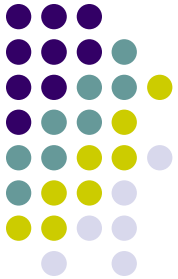
Regularized regression



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., IV (*J)* that we believe are informative in predicting some outcome

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…

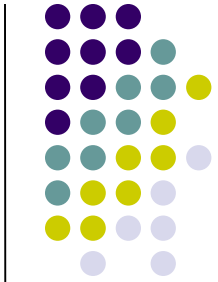
Regularized regression



- ✓ If $J > N$, OLS estimates are not unique. Moreover, many instances the result will be computationally infeasible
- ✓ Even with $N > J$, as 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:

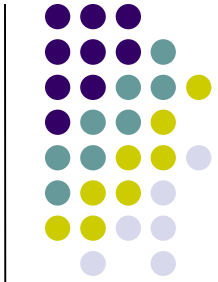
$$\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{J}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{J}\beta_j^2 \quad \rightarrow \text{ridge regression}$$

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 λ *(lambda)* is the penalty parameter (to be estimated)

Regularized regression



Why adding a penalty?

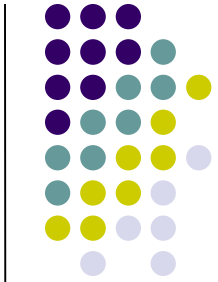
Penalty parameters constrain the size of the coefficients such that the only way the coefficients can increase is if we experience a comparable decrease in the RSS. This provides us more clarity in identifying the true signal in our model. As a result:

- ✓ It reduces the fluctuations of the coefficients
- ✓ It 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

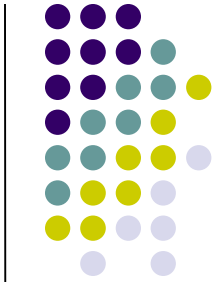


The penalty can take different forms:

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 does 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: $\lambda \sum_{j=1}^{J} |\beta_j|$ where some coefficients become zero

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

✓ 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

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

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

Regularized regression

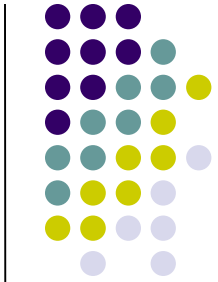
Elastic net

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

Best of both world?

- ✓ 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
- ✓ ridge regression penalty is more effective in systematically reducing correlated features together
- ✓ 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

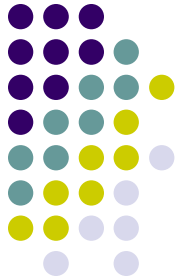


How to find best value of λ? Via cross-validation of course!

✓ 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

✓ But the logic remains the same as discussed earlier!

Gradient Boosting classifier

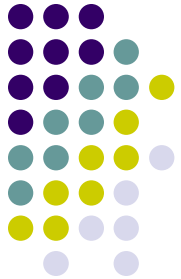


Several supervised machine learning models are founded on a single predictive model (i.e. naive Bayes, support vector machines, RR, etc.)

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

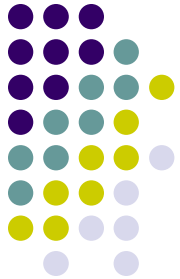


Whereas Random Forests build an ensemble of *deep independent trees*, **Gradient Boosting Machines** (GBM) build an ensemble of shallow and weak *successive trees* with **each tree learning and improving on the previous**

When combined, these many weak successive trees produce a powerful “committee”

Let’s see how

Gradient Boosting classifier



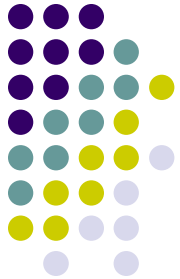
Training weak models:

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



Which advantages of doing it?

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)

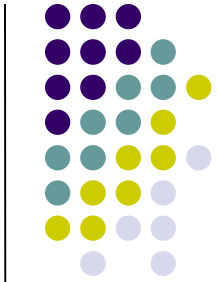
Gradient Boosting classifier

Boosted trees are **grown 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

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: $h_1(x)=y-F_1(x)$
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. Continue this process until some mechanism (i.e. cross validation) tells us to stop

Gradient Boosting classifier



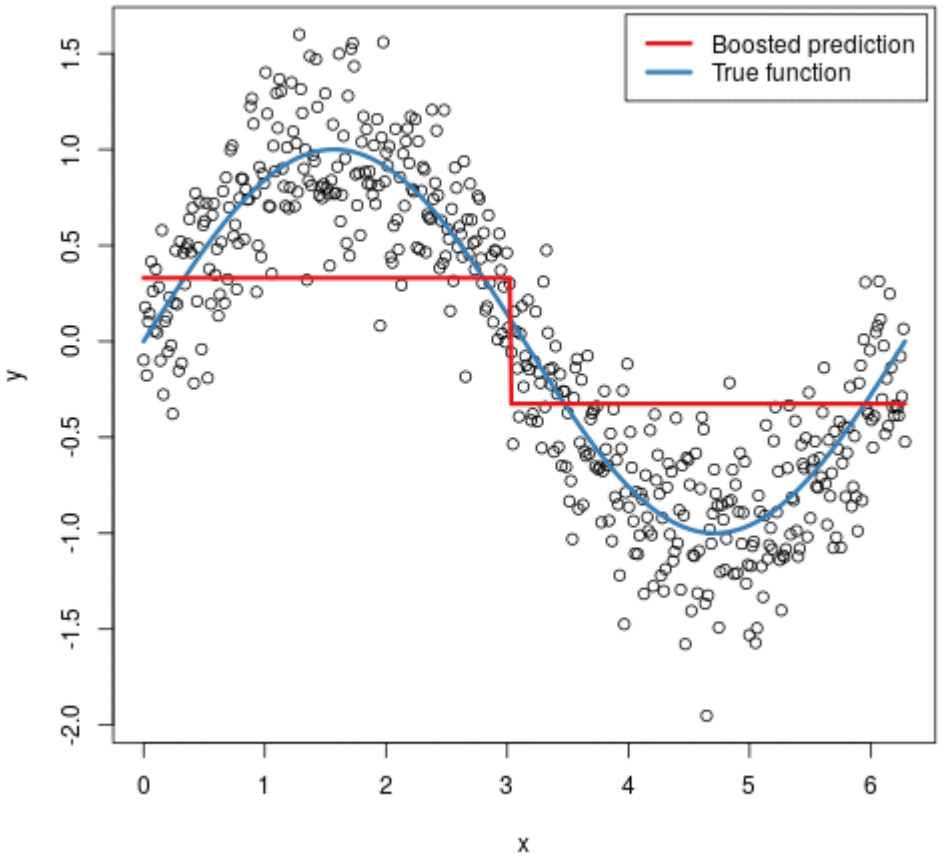
To illustrate the behavior, assume the following situation

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

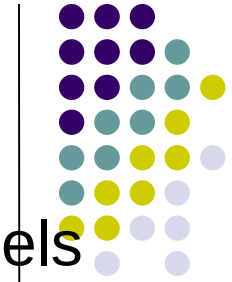
The boosted prediction (in red) illustrates the adjusted predictions after each additional **sequential tree** is added to the algorithm

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



The main idea of boosting is therefore to add new models to the ensemble ***sequentially.*** At each particular iteration, a new *weak*, base-learner model 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

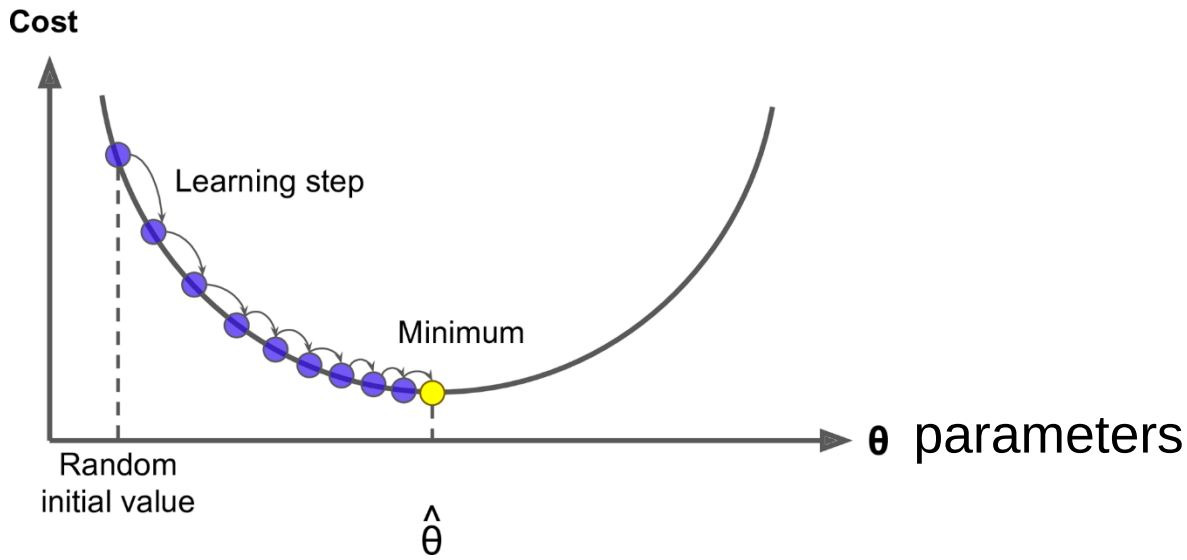
In practice however, boosted algorithms almost **always use decision trees as the base-learner**

Gradient Boosting classifier

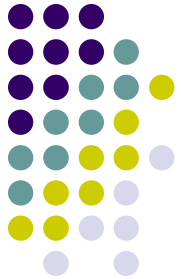
Gradient boosting is considered a ***gradient descent*** algorithm

Gradient descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems

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



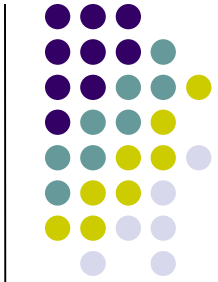
Gradient Boosting classifier



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

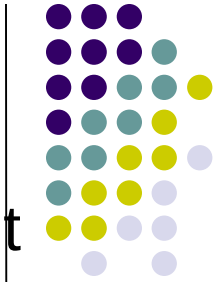
Gradient Boosting classifier



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. We will discuss about it in the next class

Gradient Boosting classifier

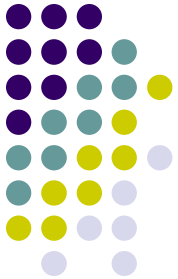


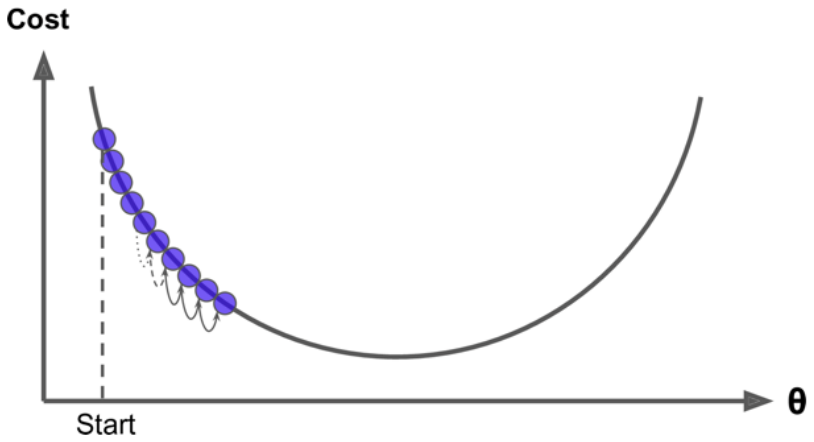
For example: an important parameter in gradient descent is the size of the steps which is determined by the *learning rate* (eta). It controls how quickly the algorithm proceeds down the gradient descent

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)

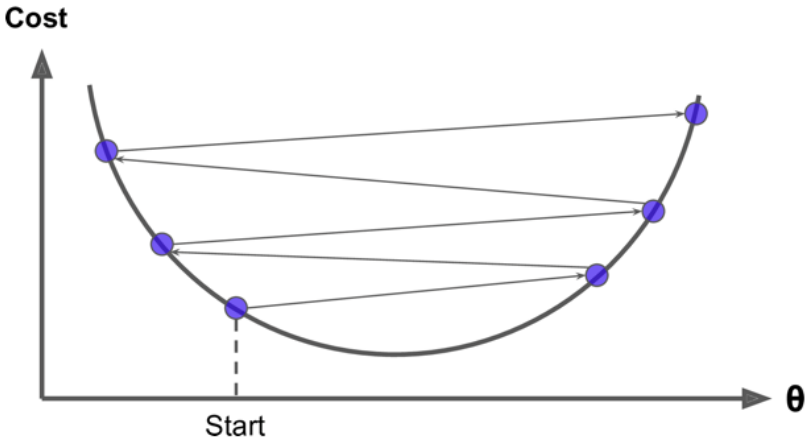
On the other hand, if the learning rate is too high, you might jump cross the minimum and end up further away than when you started

Gradient Boosting classifier



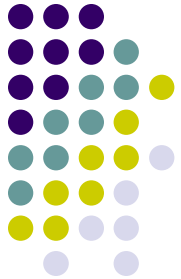


a) too small



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

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 but the stochastic nature of random sampling also adds some random nature in descending the loss function gradient. Although this randomness does not allow 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')