**Table of Contents**

[1. Linear Regression](#_24yzzmi9zfo2) 1

[2. Logistic Regression](#_dbn0g84q2jxk) 2

[3. Decision Tree](#_g0fnx2omn2oi) 4

[4. SVM](#_bwi8375qvqty) 4

[5. Naive Bayes](#_k7t1xlgfo2st) 6

[6. kNN](#_v4kkbnebopaj) 7

[7. K-Means](#_k57fn6vkfejc) 7

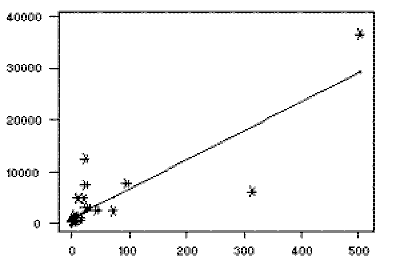
[8. Random Forest](#_qlyp8mpstral) 8

[9. Dimensionality Reduction Algorithms](#_cz76spgbx0z1) 9

[10. Gradient Boosting Algorithms](#_ykmer9ydp552) 10

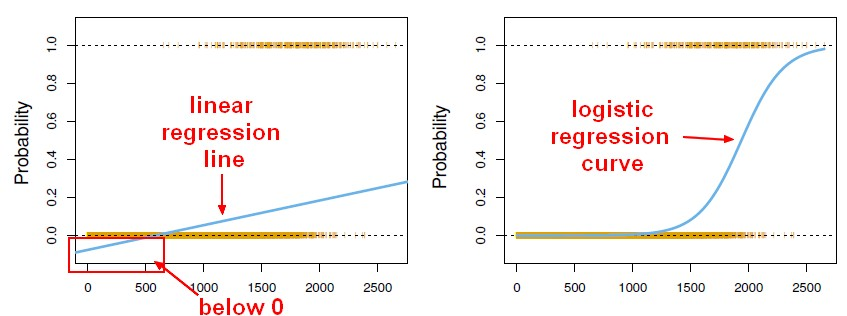
# 1. Linear Regression

* Attempts to model the relationship between two variables by fitting a linear line to observed data
* Of the form y=mx+b, where m is the slope and b is the y-intercept
* Before using a linear model to observe data, you should first determine if there is a correlation between the variables of interest (can be done by calculating the correlation coefficient)
* The most popular method to fit a regression line is the method of least squares. This method calculates the best fitting line for the observed data by minimizing the sum of squares of the vertical deviations from each data point to the line.
* **Residuals** (e) are the vertical distance between a point and the regression line. Plotting the residuals on a graph against the explanatory variable can help reveal non-linear relationships.
* **Lurking Variables** exist when the relationship between two variables is affected by the presence of an unknown, third variable.
* **Extrapolation**, or using a regression equation to predict values outside its range, is generally considered unsafe.



# 2. Logistic Regression

* Goal is to predict a categorical label
* Misleading name, actually a classification algorithm
* Named for the function used at the core of its method, the logistic function (aka sigmoid function). Denoted as 1 / (1 + e^-value). Takes on an S shaped curve (population growth in ecology)
* Whereas linear regression is unbounded (it’s a line, continues on for infinity), logistic regression is bounded between 0 and 1. Therefore, logistic regression only has a limited number of outcomes compared to linear regression (hence categorical)
* Uses maximum likelihood estimation to determine the “best” parameters to ensure that the process described by a model matches the data that was actually observed. This is usually the mean and standard deviation.



# 3. Decision Tree

* Uses a tree-like model of decisions to reach a particular endpoint. This endpoint depends on whether the tree used is being used in a classification or regression context.
* The leaf that doesn’t split anymore is often known as the decision/leaf
* Growing a tree involves deciding which features to choose and what conditions to use to split, along with knowing when to stop
  + When we calculate when to split, our goal is to minimize accuracy lost per split. This is known as the cost function, and the split that costs the least accuracy is chosen
  + This algorithm is recursive in nature as the groups formed can be subdivided using the same strategy (known as a greedy algorithm)

Steps:

1. Begin with your training dataset, which should have some feature variables and classification or regression output.
2. Determine the “best feature” in the dataset to split the data on; more on how we define “best feature” later
3. Split the data into subsets that contain the possible values for this best feature. This splitting basically defines a node on the tree i.e each node is a splitting point based on a certain feature from our data.
4. Recursively generate new tree nodes by using the subset of data created from step 3. We keep splitting until we reach a point where we have optimised, by some measure, maximum accuracy while minimising the number of splits / nodes.

* When to stop? Usually, there are many features, so there will be a large number of splits in a tree (leads to overfitting). Solution: set minimum number of training data per leaf (ex. min 10 yes’s as the final leaf, if 6 yes’s then the tree stops splitting).
  + Can also be done by setting a maximum depth of the tree
* Performance can be further improved by using **pruning** (removing branches that make use of features that have low importance). Method:
  + Start at leaves
  + Removes each node with the most popular class in that leaf, this change is kept if it doesn’t affect accuracy.

Advantages of CART

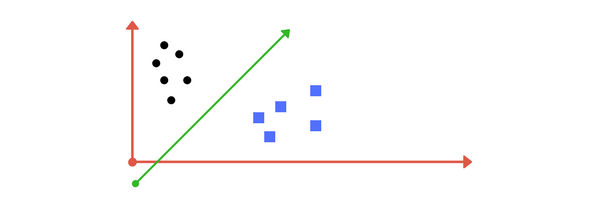
* Simple to understand, interpret, visualize.
* Decision trees *implicitly perform variable screening or feature selection.*
* Can *handle both numerical and categorical data*. Can also *handle multi-output problems.*
* Decision trees require relatively *little effort from users for data preparation.*
* *Nonlinear relationships between parameters do not affect tree performance.*

## Disadvantages of CART

* Decision-tree learners *can create over-complex trees* that do not generalize the data well. This is called *overfitting*.
* Decision trees can be unstable because *small variations in the data might result in a completely different tree being generated.* This is called *variance*, which needs to be *lowered by methods like* *bagging and boosting*.
* Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
* Decision tree learners create *biased trees if some classes dominate*. It is therefore recommended to balance the data set prior to fitting with the decision tree.

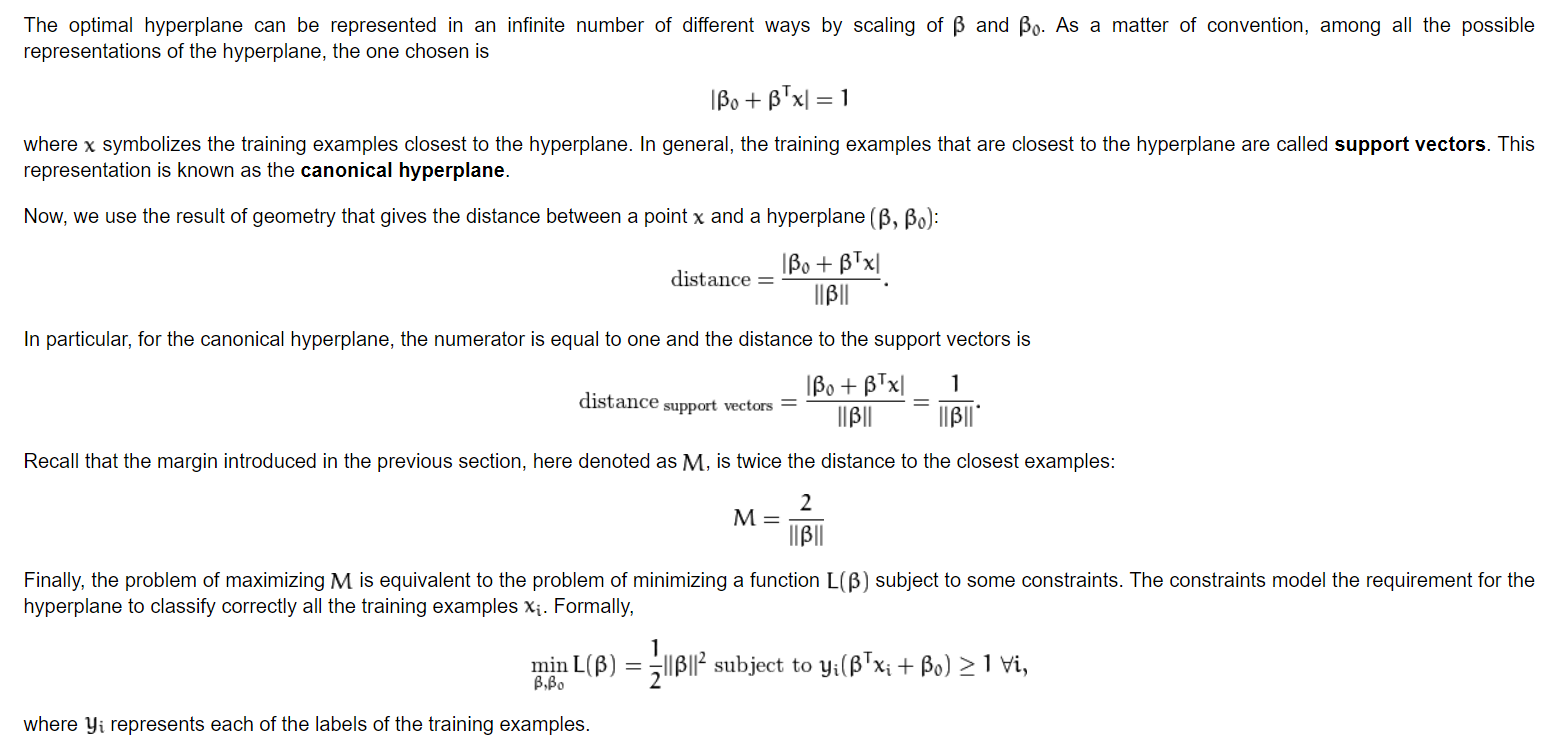
# 4. SVM

* Can be used for both classification and regression, but typically used with classification
* Given labeled training data, the algorithm outputs an optimal hyperplane (farthest that categorizes given examples



(Example of a line being found that categorizes that data into distinct sets)

* Most of the time, it is harder to find this line. Transformations, or **kernels**, are needed to map the original functions to new functions.
* Tuning Parameters: **Kernel, Regularization, Gamma, and Margin.**

****

**Kernel**

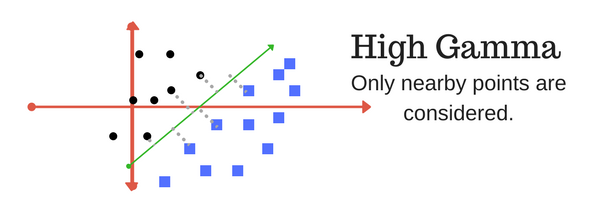
* Learning of the hyperplane in linear SVM is done by transforming the graph with linear algebra.
* **Linear Kernel:** calculated with the inner products of a new input vector (x) with all support vectors in training data
  + f(x) = B(0) + sum(ai \* (x,xi))
* **Polynomial Kernel**: K(x,xi) = 1 + sum(x \* xi)^d
* **Exponential Kernel**: exp(-gamma \* sum((x — xi²))
  + Inputs: (x)
  + Support Vector: (xi)
  + B(0): estimated from the input
* **Kernel Trick:** Calculates separation in a higher dimension

**Regularization** (denoted by c)

* Tells the SVM optimization how much you want to avoid misclassifying each training example
* For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
  + If you optimize too much, high possibility of overfitting the data.

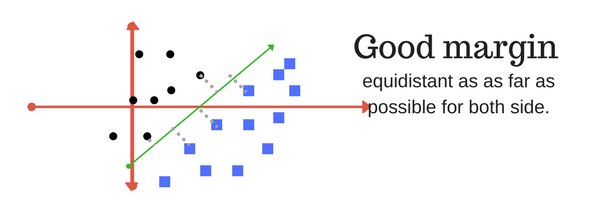
**Gamma:**

* Defines how far from the plausible separation line that points are in consideration; high gamma means close points are in consideration, and vice versa.

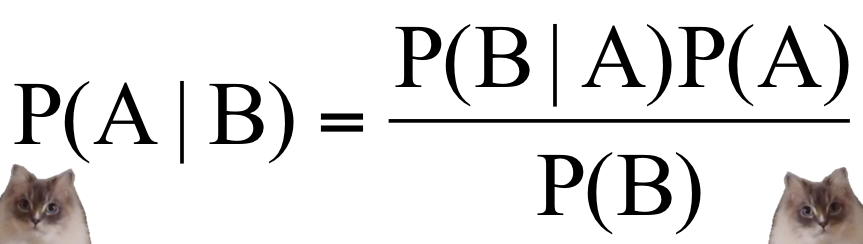


**Margin:**

* Defines the separation between class points.



# 5. Naive Bayes

****

* Assumes that the features are independent, which means that the probability of one event does not affect the probability of another event occurring.
* Often used for text classification, recommender systems, and real time prediction (due to speed)

# 6. kNN

* Assumes similar things are near each other to classify

1. Load the data
2. Initialize K to your chosen amount of neighbors
3. For each example in the data,
   1. Calculate the distance between the query example and the current example from the data
   2. Add the distance and index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

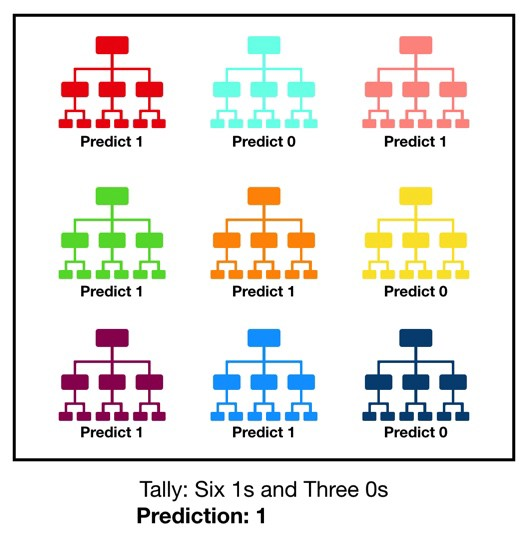
# 7. K-Means

* Partitions a set of points into K sets (clusters) such that each point in the cluster is near each other.
* Unsupervised learning because the points have no external classification

1. Starts with a group of randomly selected centroids which are used as the beginning points for each cluster
2. Iteratively calculates to optimize position of each centroid.
3. Stops optimizing clusters when:
   1. Centroids stop stabilizing - no change in their position because the clustering has been successful
   2. Defined number of iterations has been reached.

# 8. Random Forest

* Consists of a large number of decision trees known as an **ensemble**
* Each individual tree outputs a decision and the most voted is the collective decision
  + Concept is the “wisdom of the crowd” mentality
  + A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models
  + This has the effect of reducing errors
* Prerequisites
  + Actual signal in the features such that the model will perform better than random guessing
  + The predictions made by the individual trees need to have low correlations with each other
* How does random forest ensure individual trees are not correlated with each other?
  + **Bagging** (Bootstrap aggregation): Each individual tree randomly samples from the training data set with replacement
  + **Feature Randomness**: In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the operations in the left node vs those in the right node. However, random forest trees pick randomly from a set of features instead of an optimal separation, ensuring randomness and lower correlation.



# 9. Dimensionality Reduction Algorithms

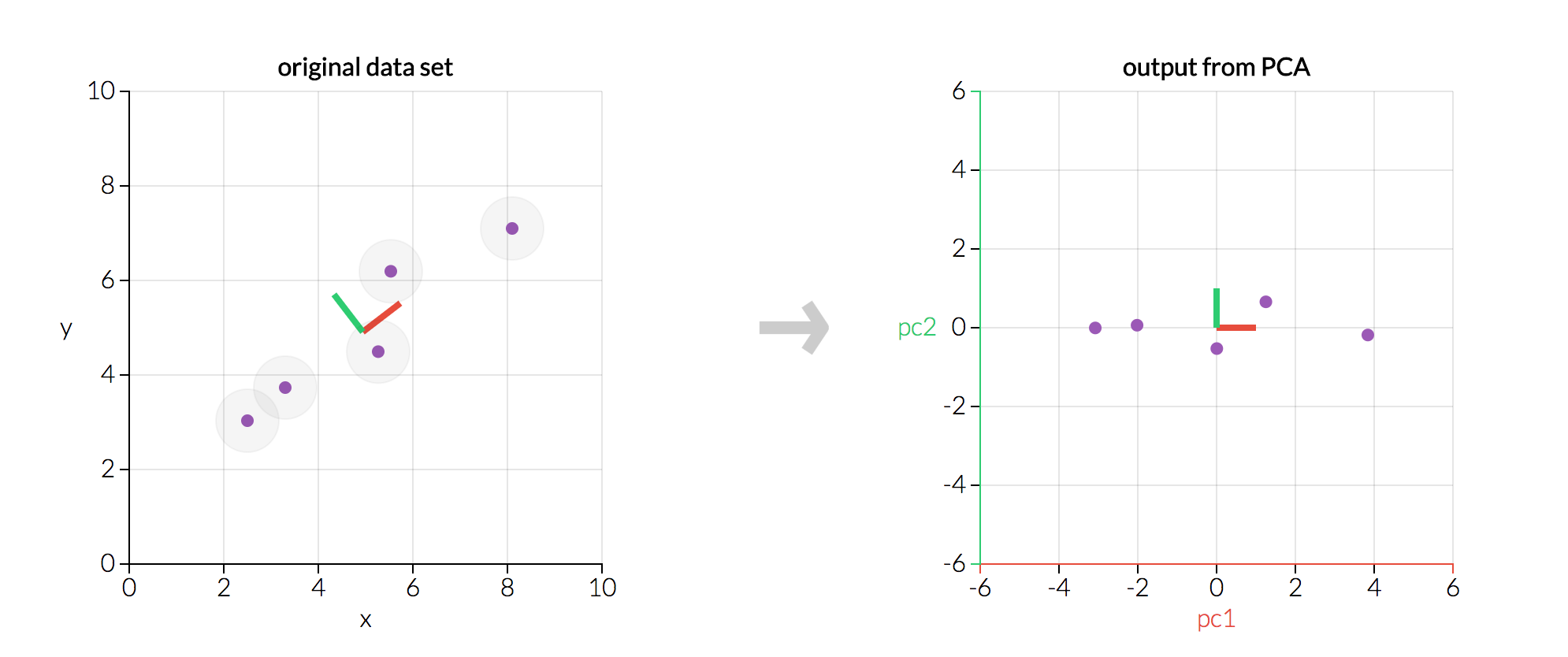
* Used when a dataset has a large number of features that are correlated with each other and hence redundant
* Advantages:
  + Reduces time and storage space required
  + Removal of multi-collinearity improves the interpretation of parameters of the machine learning model
  + Becomes easier to visualize data when reduced to 2D or 3D
  + Avoids the curse of dimensionality (as dimensionality increases, the volume of space increases so fast that the available data becomes sparse
* Disadvantages:
  + May lead to some data loss
  + May not know how many principal components to keep
* Feature Elimination vs Feature Extraction
  + **Feature Elimination:** Completely removes some features from the dataset
  + **Feature Extraction:** Creates new independent variables that are a combination of the previous independent variables, ordered by how well they predict our dependent variable

**Principal Component Analysis (PCA)**

* You should **not** use PCA if you are not comfortable making your features less readable

Algorithm:

1. Construct the covariance matrix of the data
2. Compute the eigenvectors of the matrix
3. Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of the variance of the original data.
4. Drop the eigenvectors that are relatively unimportant.



1. The two charts show the exact same data, but the right graph reflects the original data transformed so that our axes are now the principal components
2. Every principal component will always be **orthogonal** to every other principal component
   1. Thus, since our principal components are orthogonal, they are linearly independent of one another

There is also:

**Singular Value Decomposition (SVD)  
Independent Component Analysis (ICA)**

# 10. Gradient Boosting Algorithms

* Method of modifying “weak learners” (performance is only slightly better than random chance) into stronger learners

**AdaBoost**

* Method of training a decision tree in which each observation is assigned an equal weight.
* After evaluating the first tree, we increase the weights of observations that are difficult to classify and lower the weights of those that are easy to classify. New Model: Tree 1 + Tree 2
* Then, we compute the classification error from this new 2-tree ensemble method and grow a third tree to predict the revised residuals
  + Repeated for a specified number of iterations
* Predictions of the final model are a weighted sum of the predictions made by previous tree models

**Gradient Boosting**

* Differs from AdaBoost in that gradient boosting identifies shortcomings by using gradients in the loss function (y=ax+b+e, where e is the error term).
  + The loss function is a measure indicating how good a model’s coefficients are at fitting the underlying data
* Thus, the advantage is that it allows the user to optimize a self-inputted cost function that is more applicable to real world scenarios
* Involves three elements
  + Loss function to be optimized
  + Weak learner to make predictions
  + Additive model to add weak learners to minimize the loss function

1. Loss function

* Depends on the problem being solved
* Must be differentiable

1. Weak learner

* Usually decision trees, specifically, regression trees that outputs real values for splits and whose output can be added together, allowing subsequent models to be added and “correct” the residuals in the predictions
* It is common to restrain the weak learners in specific ways, such as a maximum number of layers, nodes, or leaves (to ensure learners remain weak)

1. Additive model

* Trees are added one at a time, and existing pool is not changed
* **Gradient Descent** is used to minimize the loss when adding trees
  + Gradient descent is used instead of parameters because the way that gradient descent works is that it minimizes the cost of a function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient
* The “parameters” in this case are weak learners (decision trees). After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss.
  + This is done by parameterizing the tree, then modifying the parameters of the tree and moving in the right direction (by reducing residual loss)

**Problems**

* Gradient Boosting is a greedy algorithm and can quickly overfit a training dataset
* Benefits from regularization methods that penalize various parts of the algorithm
* To improve,
  + Tree constraints
    - It is important that weak learners have accuracy but remain week
    - Can be done by constraining tree parameters:
      * Number of trees
      * Tree depth
      * Number of nodes
      * Number of observations per split
      * Minimum improvement to loss
  + Shrinkage
    - Predictions of each tree are added sequentially
    - The contribution of each tree to this sum can be weighted to slow down the learning of the algorithm, this is known as **shrinkage**
    - Effect: Learning is slowed down, requiring more trees to be added to the model, taking longer to train, providing a trade-off between number of trees and learning rate
  + Stochastic Gradient Boosting (Random Sampling)
    - Similar to how random sampling works in Random Forest, apply the same concept here to reduce correlations between trees in the sequence
  + Penalized Learning
    - Additional constraints imposed on the parameterized trees in addition to their structure
      * L1, L2 regularization of weights