1. Linear Regression

- Learning the parameters: Minimize the cost function.
- Minimizing cost function: via Gradient Descent.
- Regularization:(impose penalty term λ to shrink coefficients to prevent overfitting).
  1. **Lasso (L1)**: Can minimize some coefficients to zero which leads to feature selection. LASSO (Least Absolute Shrinkage and Selection Operator).

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

```
# import lasso regression
from sklearn.linear_model import Lasso
```

  2. **Ridge (L2)**: Shrinks all coefficients with same proportion, almost always outperforms L1.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

```
# import ridge regression
from sklearn.linear_model import Ridge
```

  3. **Elastic Net**: Combined L1 and L2 as regularizer. Overcomes limitations of L1 (i.e., given large p and small n, L1 selects at most n features before it saturates. If there are highly correlated group of variables, L1 tends to select one out of that group and ignores others) by adding a quadratic part to the pentalty ($\|\beta\|^2$):

$$\hat{\beta} = argmin_{\beta}(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1)$$

     Quadratic penalty term makes the loss function strongly convex, and therefore has a unique minimum. L1 and L2 are special cases when $\lambda_1 = \lambda, \lambda_2 = 0$ and $\lambda_1 = 0, \lambda_2 = \lambda$.
     - Procedure: For a fixed λ_2, find coefficients of L2, and then performs L1 shrinkage.
     - Due to the double amount of shrinkage (from L1 and L2), this leads to increased bias and poor predictions. To improve prediction performance, we rescale coefficients by multiplying estimated coefficients by $(1 + \lambda_2)$.
     - Choosing best lambdas (i.e. λ_1, λ_2): via cross validation .

```
# import Elastic Net
from sklearn.linear_model import ElasticNet
```

- Choosing between regularization methods: If variables are highly correlated (most variables are useful), L2. If variables are rather independent (some variables are not important), L1 as it leads to variable selection. If there are more variables then observations and high correlation between parameters, Elastic Net
- Assumes linear relationship between features and label.
- Can add polynomial and interaction features to add non-linearity.
- Loss functions/Evaluation Metrics:
  1. **Mean Absolute Error (MAE)**: Easy to understand and assigns equal weights to errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  2. **Mean Squared Error (MSE)**: Punishes larger errors, i.e. gives higher weights to larger errors.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  3. **Root Mean Squared Error (RMSE)**: More popular than MSE since it is interpretable in the y units.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Sources:

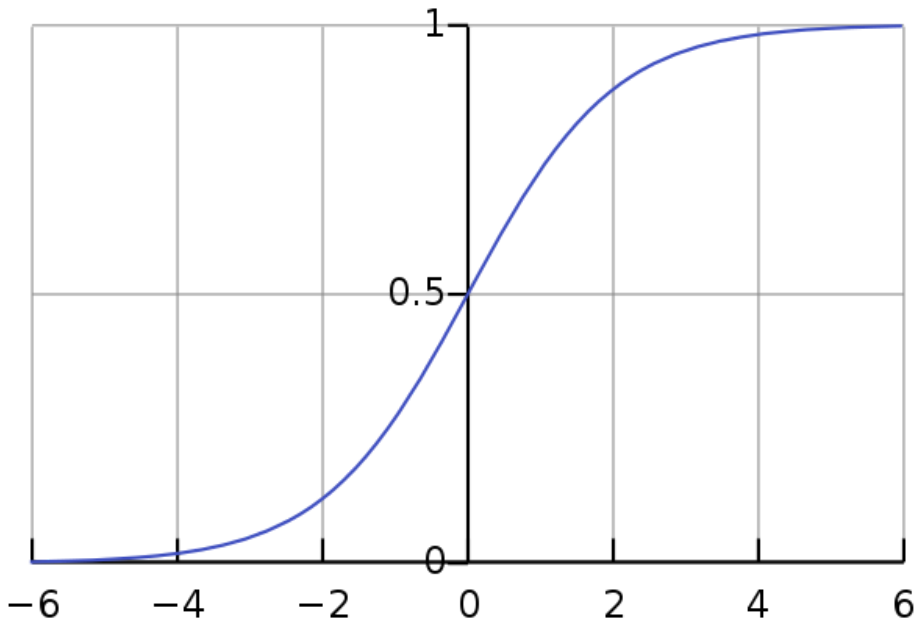
- https://en.wikipedia.org/wiki/Elastic_net_regularization (https://en.wikipedia.org/wiki/Elastic_net_regularization)

- https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization (https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization)
- https://www.youtube.com/watch?v=1dKRdX9bfIo (https://www.youtube.com/watch?v=1dKRdX9bfIo)

2. Logistic Regression

- Generalized Linear Model (GLM) for binary classification problems.
- Apply sigmoid function to the outputs of the linear models, resulting in outputs between 0 or 1. If > 0.5, usually 1, and 0 otherwise.
- It is a special case of the softmax function that deals with multi-class problems

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}, t = \beta_0 + \beta_1 x$$



- Model Evaluation: **Confusion matrix**
  - True Positives (TP)
  - True Negatives (TN)
  - False Positive (FP, Type I Error)
  - False Negative (FN, Type II Error)

  - **Accuracy**: (TP + TN) / total
    - The number of correct predictions made out of the total number of predictions.
    - Useful when target classes are well balanced, i.e. equal number of classes (50 dogs & 50 cats).
    - Not so useful when classes are unbalanced: if we had 99 images of dogs and 1 image of a cat, if our model was simply a line that always predicted a dog, we will get 99% accuracy.
    - In this situation, we would like to understand **recall** and **precision**.

  - **Misclassification Rate**: (FP + FN) / total

  - **Precision**: The ability to classify an instance as positive, when in reality it is negative. For all the instances that were labeled as positive, what percent is correct?

$$\frac{TP}{TP + FP} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + people\ incorrectly\ classified\ as\ terrorists}$$

    - The ability for a model to find only the relevant data points.
    - Number of true positives over the sum of true positive and false positive.

  - **Recall**: The ability for a classifier to find all the positive instances. For all the instances that were actually positive, what percent is classified correctly?

$$\frac{TP}{TN + FN} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + terrorists\ incorrectly\ labeld\ as\ not\ terrorists}$$

    - The ability for a model to find all the relevant data points.
    - Number of true positives over the sum of true positive and false negatives.

  - You will usually have a trade off between precision and recall.

  - While recall expresses the ability to find all relevant instances in a dataset, precision expresses the portion of the data points our model says was relevant that actually were relevant.

- **F1-Score**: Weighted harmonic mean of **precision** and **recall**. Ranges between 0 to 1, 1 being the best score. In general, F1-Scores are lower accuracy measures as they embed precision and recall into their computation. As a rult of thumb, the weighted average of F1 should be used to compare classifier models instead of global accuracy.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

  - In cases when we want to find an optimal blend of precision and recall we can combine the two metrics using what we call f1 score.
  - We use this because it punishes extreme values.
  - A classifier with a precision 1.0 and a recall of 0.0 has a simple avg of 0.5 but an F1 score of 0.
  - We can also view all correctly classified vs. incorrectly classified images in the form of a confusion matrix.

- **Support**: Number of actual occurrences of the class in the specified dataset. Imbalanced support in the training set indicates structural weakness in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support does not change between models but instead diagnoses the evaluation procewss.

		predicted condition		
	total population	prediction positive	prediction negative	Prevalence $= \frac{\Sigma \text{ condition positive}}{\Sigma \text{ total population}}$
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)	True Positive Rate (TPR), Sensitivity, Recall, Probability of Detection $= \frac{\Sigma \text{ TP}}{\Sigma \text{ condition positive}}$
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**	False Positive Rate (FPR), Fall-out, Probability of False Alarm $= \frac{\Sigma \text{ FP}}{\Sigma \text{ condition negative}}$
	Accuracy $= \frac{\Sigma \text{ TP} + \Sigma \text{ TN}}{\Sigma \text{ total population}}$	Positive Predictive Value (PPV), Precision $= \frac{\Sigma \text{ TP}}{\Sigma \text{ prediction positive}}$	False Omission Rate (FOR) $= \frac{\Sigma \text{ FN}}{\Sigma \text{ prediction negative}}$	Positive Likelihood Ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$
		False Discovery Rate (FDR) $= \frac{\Sigma \text{ FP}}{\Sigma \text{ prediction positive}}$	Negative Predictive Value (NPV) $= \frac{\Sigma \text{ TN}}{\Sigma \text{ prediction negative}}$	Negative Likelihood Ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$

Sources:

- https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization (https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization)
- https://en.wikipedia.org/wiki/Logistic_regression (https://en.wikipedia.org/wiki/Logistic_regression)

3. Naive Bayes (NB)

- Supervised Learning Algorithm based on Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- Called Naive because it builds on the 'naive' assumption that each feature are independent from each other.

- Can make different assumptions: Gaussian, Bernoulli and Multinomial.
  - **Gaussian NB**: Used under the assumption that variables are continuous and normally distributed.

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

  - **Multinomial NB**: Assumes that samples represent frequencies of events generated from Multinomial distribution, where $(p_1, \ldots, p_n)$ are the probabilities of $1, \ldots n$ events. A feature factor, $x = (x_1, \ldots, x_n)$ is a histogram with x_i indicating the number of times an event i was observed for an instance. An example of its application is text classification, with events representing the occurrence of a word in a single document (BoW:Bag of Words).

$$likelihood = p(x|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i}$$

  - **Bernoulli NB**: Assumes that features are independent and binary. Also widely used in text classification where binary term occurrence features are used instead of term frequencies.

$$likelihood = p(x|C_k) = \prod_{i=1}^{n} p_{ki}^{x_i}(1 - p_{ki})^{1-x_i}$$

- Works well with text classification (baseline model for text classficiation too).
- Extremely fast compared to other models

$$posterior = \frac{prior * likelihood}{evidence}$$

Sources:

- https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization (https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization)
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier (https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

4. KNN (K-Nearest Neighbor)

- A non-parametric method used for classification and regression.
- KNN Classification: output is a class membership. An object is assigned to the class most common among its k nearest neighbors. If k=1, then the object is assigned to the class of that single neighbor.
- KNN regression: output is the property value for the object. This value is the avg of the values of k nearest neighbors.
- Lazy learning algorithm.
- Given a datapoint, find the K nearest data points using distance metric (i.e. Euclidean). In classification, take majority label of neighbors. In regression, take mean of labels.
- We do not train the model in KNN, instead we simply compute during inference time. This is computationally expensive since each of the test example need to be compared with every training example to compare how close they are.
- There are approx. methods that are faster by partitioning train data into regions (i.e. ANNOY, Approximate Nearest Neighborhors Oh Yeah).
- When k=1 or some other small number of the model is prone to overfitting (high variance). On the other hand, when k = no. data pts or other large number, then the model is prone to underfitting (high bias). See image below:

- Steps:
  1. Calculate distance from point x (new data point) to all points in your data.
  2. Sort the points in your data by increasing distance from x.
  3. Predict the majority label of the "k" closest points
  4. Note: Choosing K will affect what class a new point is assigned to:
- **Elbow Method**: A method to determine optimal value for 'k'.
- Pros and Cons:
  - Pros: simple, easy to train / Works with any no. of classes / Easy to add more data / few parameters (i.e. k's and distnace metric)
  - Cons: High prediction cost (worse for large data) / Not good with high dimensional data / don't work well with categorical features
- When given unlabeled data without context, useful to classify data.

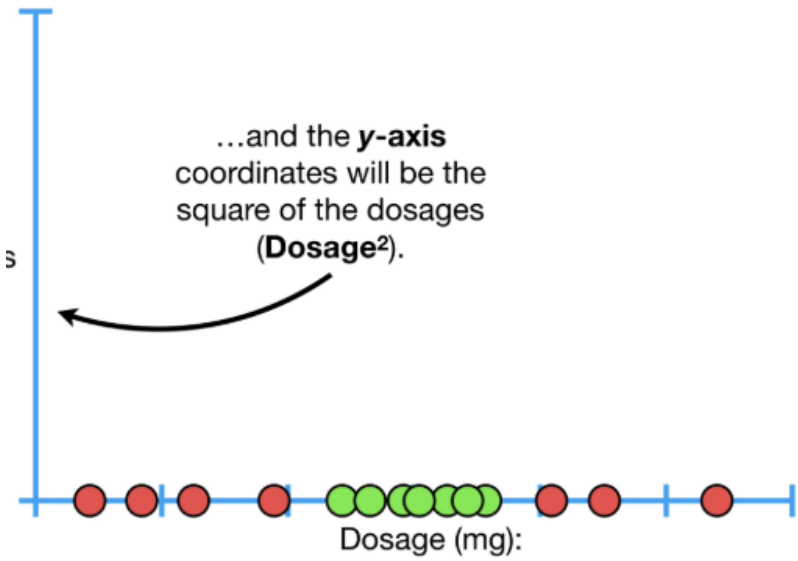
Sources:

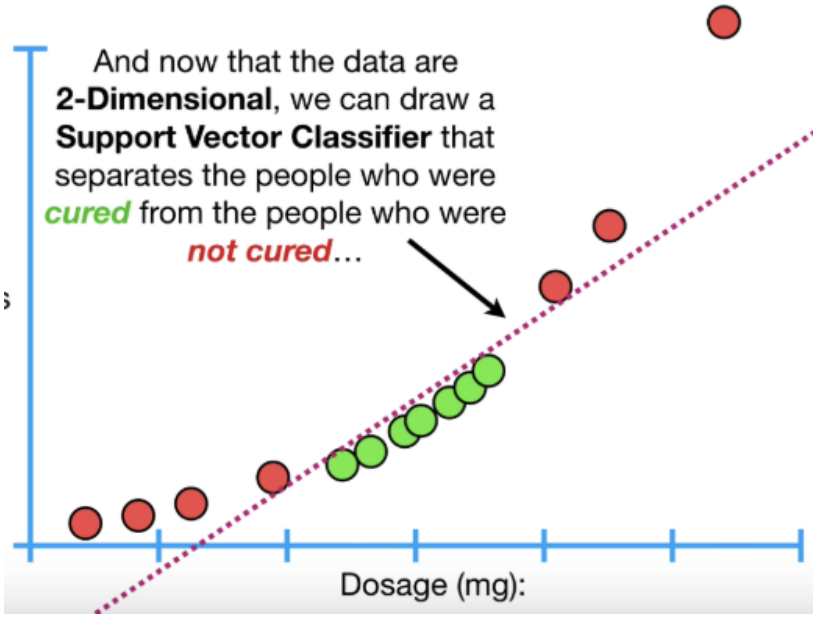
- https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization (https://github.com/ShuaiW/data-science-question-answer/blob/master/README.md#l1-vs-l2-regularization)
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)

5. SVM (Support Vector Machines)

- Can perform linear, nonlinear or outlier detection (unsupervised)
- Large margin classifier: SVM determines decision boundary and sets the boundary to be as far from the closest training point as possible.
- Support Vectors: are the closest training examples since they are the points based on where the decision boundary is drawn.
- Sensitive to **feature scaling** (method used to normalize the range of independent variables or feature data (aka data normalization).
- **Terminologies**:
  - **Margin**: Shortest distance between the observations and the threshold (i.e. decision boundary).
  - **Maximal Margin classifier**: Using the threshold that gives the largest margin to make classification.
    - Sensitive to outliers in the training set.
    - So we choose threshold to allow misclassification (bias & variance trade-off).
  - **Soft Margin**: When we allow misclassification, the distance between the observations and the threshold.
    - We use Cross Validation to determine how many misclassifications and observations to allow inside the soft margin to get better classification.
  - **Support Vector Classifier (SVC)**: When we use soft margin to determine the location of the threshold. SV comes from the fact that the observations on the edge and within the soft margin are called Support Vectors.
    - Handles outliers and allow misclassification.
    - Since they allow misclassification, they can handle overlapping classifications (when data pts overlaps each other.
    - In 2D, SVC is a line (1D subspace) and in 3D, SVC is a plane (2D subspace).
    - Does not perform well when one class of data points are sandwiched between another class of data points.
      - This is where SVM comes into play.

  - **Support Vector Machines (SVM)**: Adds another axis which are squared values of the original axis where data points lie to classify data on higher dimension by SVC.
    - Uses Kernel functions (i.e. Polynomial Kernel with parameter **d**=polynomial degree) to systematically find SVC in higher dimensions.
    - We find the optimal value of **d** using Cross Validation.
    - Another commonly used Kernel Function: Radial Basis Function (RBF)
      - RBF finds SVC in infinite dimensions
      - When given new/test data, RBF behaves like a **Weighted Nearest Neighbor** model (closest observations have a lot of influence on how we classify new observation).





- **Kernel Trick**: Calculating high dimensional relationships without actually transforming the data into higher dimension.
  - Reduces the computation complexities of SVM
  - Makes calculating relationships in the infinite dimensions possible (RBF).

- Perform **Grid Search** to find optimal parameters and better recall values.

Sources:

- https://en.wikipedia.org/wiki/Feature_scaling (https://en.wikipedia.org/wiki/Feature_scaling)
- https://www.youtube.com/watch?v=efR1C6CvhmE (https://www.youtube.com/watch?v=efR1C6CvhmE)
- https://github.com/ShuaiW/data-science-question-answer#knn (https://github.com/ShuaiW/data-science-question-answer#knn)

6. Decision Tree (DT)

- Non-parametric, supervised learning algo

- Given training set, DT divides feature space into regions. For inference, determine which region the test data fall in and take the mean label values (regression) or the majority label value (classification).

- **Construction**: From top to down, selects a variable to split the data where the target variables within each region are as homogenous as possible.
  - Two common metrics: gini impurity or information gain
  - Steps:
    1. Calculate impurity for each columns to determine root node (lowest impurity).
    2. If node itself has lowest impurity score, there is no point in splitting and becomes a leaf node.
    3. If splitting leads to an improvement, then pick separation with lowest impurity value.

- **Pros**: easy to understand and interpret

- **Cons**:
  - May lead to overfitting if depth of the tree is not set
  - A small change in the training data might lead to a tree that is significantly different.
  - sensitive to training set rotation due to its orthogonal decision boundaries.

- Consists of the **Root Node**, **Internal Nodes** and **Leaf Nodes**

Sources:

- https://github.com/ShuaiW/data-science-question-answer#knn (https://github.com/ShuaiW/data-science-question-answer#knn)
- https://www.youtube.com/watch?v=7VeUPuFGJHk&t=1s (https://www.youtube.com/watch?v=7VeUPuFGJHk&t=1s)

7. Random Forest

- Steps:
  1. Take bootstrapped sample of the data (repetition is possible).
  2. Create decision trees using data from step 1, but only use random subset of columns at each step (when building nodes).

3. Repeat steps 1 and 2 n times
4. Given all the trees, test them on the test set, i.e. given one observation, run it down through all n trees and pick answer with the most votes.

- Usually 1/3 of data (**Out-of-bag data)** will not be included in the bootstrapped dataset. Out-of-bag data will be used to test the model.
- **Out-of-bag Error**: Proportion of out-of-bag samples that were incorrectly classified.
- **Bagging**: Bootstrapping data + using the aggregate to make the decision.
- Can use out-of-bag error to compare the number of subset of columns to choose in step 2 above.
  - Usually start with square-root of the number of columns, $\sqrt{p}$, and try few settings above and below that value.
- **Dealing with missing data**: replace with the most common (mode) value or median value.
- **Refining guess**: Determine which samples are similar. Similar in a sense that they ended up in the same leaf node. Similarity check using proximity matrix (weighted frequency).
- Improve bagging further by adding some randomness. In random forest, only a subset of features are selected at random to construct a tree. Benefits: decorrelates the trees.
- For example, given a dataset, supposed that there is only one very predicative feature and a couple of moderately predicative features. In bagging trees, most of the trees will use the very predicative feature in the top split so the rest of the trees look highly correlated and similar. Taking the average of the many highly correlated results will not lead to a large decrease in variance as opposed to uncorrelated results. In random forest, for each split, we only consider a subset of features and hence reduce variance further by introducing more uncorrelated trees.
- In practice, tuning RF involves having a large number of trees (the more the merrier, but there is alway the issue of computational cost).
- `min_samples_leaf` used (minimum number of samples at the leaf node) to control the tree size and overfitting.
- Important to CV parameters.

Sources:

- https://github.com/ShuaiW/data-science-question-answer#knn (https://github.com/ShuaiW/data-science-question-answer#knn)
- https://www.youtube.com/watch?v=J4Wdy0Wc_xQ (https://www.youtube.com/watch?v=J4Wdy0Wc_xQ)
- https://www.youtube.com/watch?v=nyxTdL_4Q-Q (https://www.youtube.com/watch?v=nyxTdL_4Q-Q)

8. Boosting Tree

How it works:

It builds on weak learners in an iterative way. In every iteration, a new learner is added, while existing learners remain unchanged. All learners are weighted based on their performance (i.e. accuracy) and afte a weak learner is added, data are weighted again: examples that are misclassified gain more weights, while examples that are correctly classified lose weights. Hence, future weak learners focus more on examples that previous weak learners misclassified.

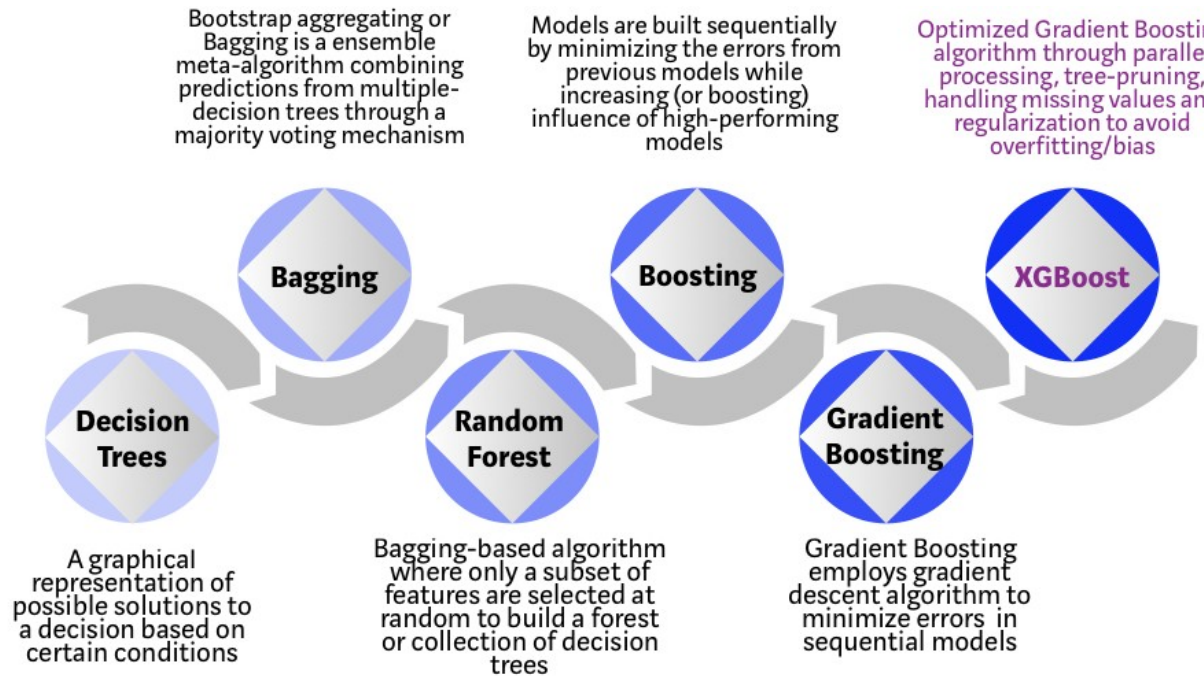
- How it is different from RF:
  1. RF grows trees in parallel, while in boosting, it is sequential.
  2. RF reduces variance, while boosting reduces errors by reducing bias.
- **Ada Boost** (Adaptive Boosting): (convert set of weak classifiers into strong ones)
  - Whereas in RF trees made with no distinct depth, in Ada Boost, trees are usually just a node with 2 learners (forest of stumps).
  - **Stump**: Tree with 1 node and 2 leaves.
    - Stumps are 'weak learners' since they only use one variable.
  - Whereas in RF all tree's final classification have same equal weights, a forest of stumps in Ada Boost have some stumps with more weights on final classification.
  - In RF, each tree is independent of another (i.e. order does not matter) whiel in Ada Boost, order matters (error from 1st stump affects the following stumps).
  - 3 ideas behind Ada Boost:
    1. AdaBoost combines a lot of 'weak' learners to classify.
    2. Some stumps weigh more than the others.
    3. Each stump is made by taking previous stump's mistakes into account.
  - For the first stump, each sample's weights are equal.
  - Total error of a stump: sum of the weights associated w/ incorrectly classified samples.

        - total error $\in [0, 1]$, where 0 = perfect stump and 1 = horrible stump
        - Determines the weight of stump:

$$\frac{1}{2}\log(\frac{1 - tot.error}{tot.error})$$

    - Increase sample weight for samples that incorrectly classified the result.
    - AdaBoost works by putting more weight on difficult to classify samples and less on those already handled well.
    - AdaBoost can be used for both classification and regression problem.

- **XGBoost** (Extreme Gradient Boosting):
    - XGBoost is a tree-based ensembled ML algorithm that uses gradient boosting framework.
        - While artificial neural networks tend to outperform for problems involving unstructured data (images, texts, etc.), tree based methods are considered best-in-class when it comes to small-medium structured data. Following is a picture describing evolution of the tree-based methods:



        - Can be used for regression, classification, ranking and other user-defined prediction problems.

Sources:

- https://github.com/ShuaiW/data-science-question-answer (https://github.com/ShuaiW/data-science-question-answer)
- https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d (https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d)
- https://www.youtube.com/watch?v=LsK-xG1cLYA&t=748s (https://www.youtube.com/watch?v=LsK-xG1cLYA&t=748s)

Additional notes:

- **Tuning parameter**: a parameter that is not estimated but guessed (i.e. Ridge Regression (L2) has a tuning parameter, λ_2). Use k-fold (i.e. 10) CV to find the best guess for the tuning parameter.
- **Cross Validation**: allows us to compare different ML methods and get a sense of how well they work in practice.
- **Bias**: Inablity for ML method to capture the true relationship
- **Variance**: Difference in fits between datasets.
- **Hyperparameter**: is a characteristic of a model that is external to the model and whose value cannot be estimated from data. Value for the hyperparameter has to be set before the learning process begins. Examples include c in SVM, k in KNN, # of hidden layers in Neural Networks.
- **Parameter**: is an internal characteristic of the model and its value can be estimated from data. Examples include coefficients of Linear model or SVC's in SVM.

- **Grid Search**: Used to find the optimal hyperparameters of a model for the most accurate predictions. Used to find better recall value.
  - C parameter: gives you low bias and high variance.
    - High value of C: low variance
    - Low value of C: low variance and high bias.
  - Kernel paramter: Gaussian radial basis /RBF
  - Gamma parameter:
    - Small gamma = gaussian with large variance.
    - Large gamma = high bias and low variance.

- **Baseline performance**: To answer to question: "What would be the success rate of the model if one were simply guessing?"

- **Bagging**: General purpose procedure for reducing the variance of ML method.

- **Bootstrapping**:

- **Understanding the evolution of tree-based methods**: Imagine you are a hiring manager interviewing candidates with excellent qualifications. Each step of the evolution of tree-based algorithm can be viewed as a version of the interview process.

  1. Decision Tree: Every hiring manager has a set of criteria when hiring a candidate, i.e. education level, number of years of experience, interview performance. Here, decision tree is analogous to a hiring manager interviewing candidates based on the criteria.
  2. Bagging: Instead of a single interviewer, there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.
  3. RF: Bagging based algorithm where only a subset of features are chosen at random. Every interviewer will only test candidates on certain randomly selected criteria.
  4. Boosting: Alternative approach where interviewer changes the evaluation criteria based on feedback from the previous interviewer. This 'boosts' the efficiency of the interview process by deploying a more dynamic evaluation process.
  5. Gradient Boosting: Special case of boosting where errors are minimized by gradient descent. This is the strategy consulting firms leverage by using case interviews to weed out less qualified candidates.
  6. XGBoost: Gradient boosting on steriods. Perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.

- **CART**: Classification And Regression Trees.

Sources:

- https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e (https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e)
- https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d (https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d)

In [ ]: