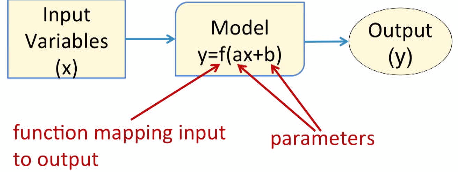
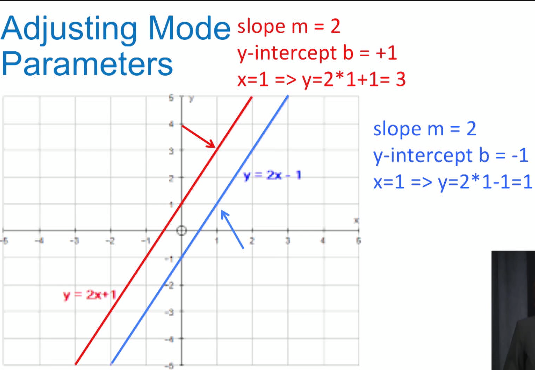
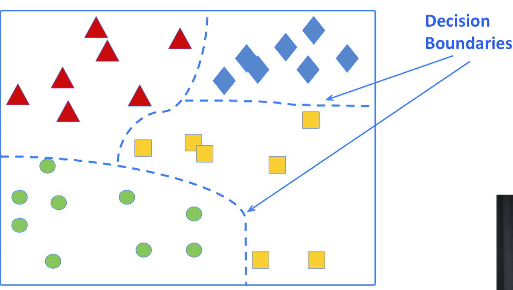
* Machine Learning = field of study focusing construction of CPU systems (**models**) that can *learn from data* w/out being explicitly programmed to
* i.e. learn to perform a specific task by analyzing lots of examples for a particular problem
* ex: recognize image of cat after being shown lots of images of cats
* Algorithms are programmed to learn *from data*, NOT from a set of given instructions
* i.e. determines what features are important in recognizing a cat
* therefore its important note the amount + the quality of data provided to a ML model
* ML models can be used to discover hidden trends + patterns which can lead to valuable insights into data to allow for data-driven decisions to be made for particular problems
* **ML Model =** mathematical mode/**parametric function** that maps inputs to outputs



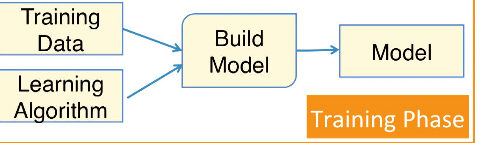
* i.e. model has **input/parameters** + uses equations to determine the relationship between inputs and outputs
* **Parameters** are used by the model to modify inputs the generate output
* They are adjusted to correct/refine the input-output relationship
* i.e. adjusting for **a** and **b** above will adjust how **x** maps to **y**
* given x, a model uses its parameters (a + b) to determine y

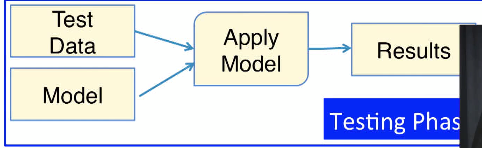


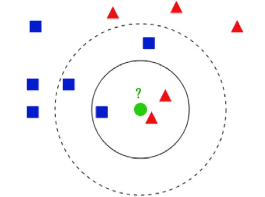
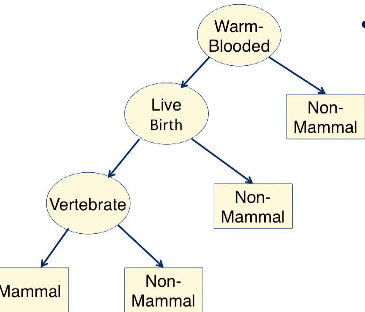
* Parameters of ML models are adjusted/estimated from the data using a learning algorithm
* The above process = **model building, models creation, model training, model fitting, etc.**
* Model parameters are adjusted in order to *reduce error* (getting outputs to match the desired outputs/targets as much as possible in classification or regression)
* Applications:
* Credit card fraud detection
* Each transaction is analyzed against transaction history to determine if its legitimate or not
* Suspicious activity 🡪 big ticket items in a category you’ve never shown interest in, Point of sales in foreign countries
* If suspicious, transaction could be denied or you may be asked to confirm the purchase
* Handwritten digit recognition 🡪 read #’s on checks to determine bank accounts, deposit amount, etc.
* ML models can sift through different variations of digits to find similar patterns to distinguished 1 from 9
* Website recommendations 🡪 list of related items after purchases
* **Data mining** (process of finding patterns in databases + data warehouses) is similar to ML + they both often use the similar algorithms and techniques
* **Predictive** **Analytics** is also similar 🡪 analyze data to predict future outcomes (sales forecasting, customer behavior)
* **Classification** 🡪 predict category/label of input data
* Examples:
* Weather as sunny, rainy, windy, cloudy
* Input data = sensor data specifying temperature, relative humidity, atmospheric pressure, wind speed + direction, etc.
* Target = the categories
* Tumor as benign or malignant (**binary classification**)
* ID handwritten digits as 1 of 10 categories (0-9)
* Given specific values for different features, the model is tasked w/ coming up w/ the category/classification/label of the record
* Output = target/label/outcome/class variable/class/category
* Target is ALWAYS categorical
* Can be **binary** (only 2 possible classifications) or **multi-class** (multinomial/multi-label)
* Think of classification as carving up an input space into regions corresponding to different labels

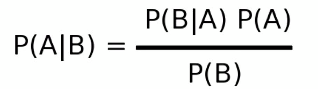


* The model formed the **decision boundaries** used to define regions separating the shapes
* Classifications are based on these regions
* To build a classification model/**classifier**, we need to use data to adjust model parameters to form decision boundaries to separate target classes
* 2 phases of building a classifier (+ other ML models)
* 1) **Training** 🡪 model is constructed + parameters are adjusted using training data
* 2) **Testing** 🡪 learned model is applied to *new* data, never-before-seen (not in training)

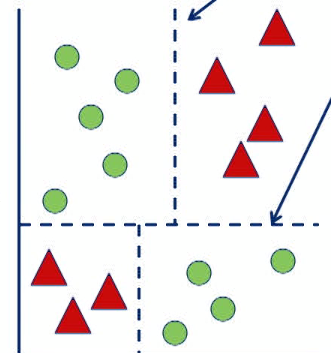




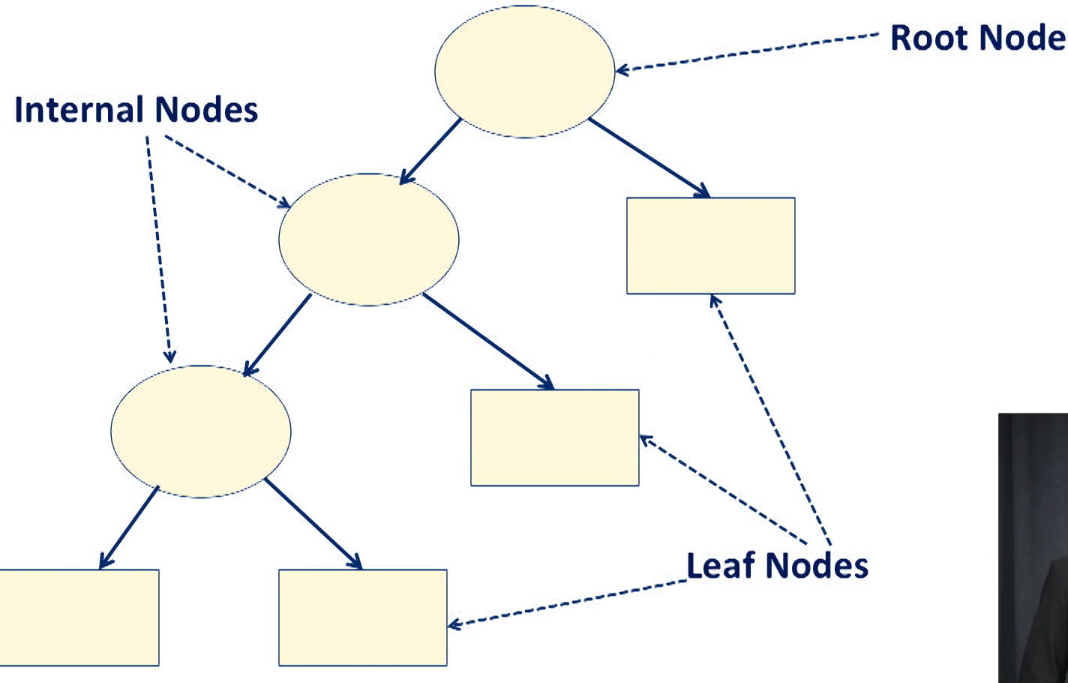
* Model is then evaluated on how it performs to test data
* **Goal** = have classifier perform well on both training + testing data = get model outputs to match targets as much as possible
* To do this, a classifier adjusts is parameters by applying a learning algorithm in the training phase
* Models used to build a classifier:
* **k-nearest neighbor (kNN) -** relies on the notion that samples w/ similar values for inputs *likely* belong to the same class
* 
* i.e. classification is dependent on target values of neighboring points
* **Decision tree** 🡪 uses a tree-like structure to represent multiple decision paths
* Each path leads to a different way to classify an input sample
* **Random forest 🡪** multiple decision trees
* 
* **Naïve Bayes** 🡪 uses probabilistic approach to classification
* Bayes’ theorem is used to capture the relationship between input data + output class
* *compares probability of an event in the presence of another event*



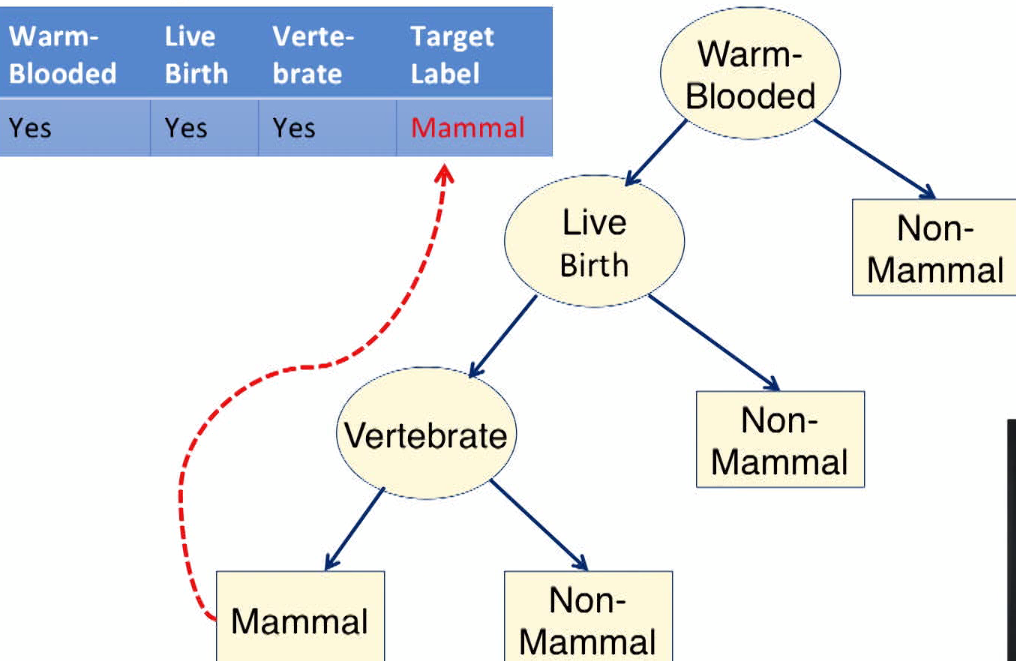
* **Regression** 🡪 predicting numeric value
* Ex: predict stock price in $
* Predicting stock prices as “rise” or “fall” 🡪 classification
* Ex: Predict *amount* of rain in a region
* Predicting “rain” or “not rain” tomorrow 🡪 classification
* **Cluster** 🡪 group similar items in a dataset into groups
* Ex: customer segmentation
* seniors, adults, teens 🡪 different likes/dislikes + purchase behavior
* can provide different targeted marketing to each
* **Association analysis 🡪** come up w/ a set of rules to capture associations between items/events
* Rules are used to determine when they occur together
* Ex: **Market Basket analysis** 🡪 used to understand customer purchase behavior
* Ex: Revealing that customers w/ checking accounts tend to be interested in other investment vehicles, such as money market accounts + this info can be used for cross-selling
* i.e. selling money market accounts to customers w/ CD’s
* Ex: Amazon recommender systems based on items usually purchased together
* Ex: Identification of web pages often accessed together 🡪 can generate more offers on related web pages
* **Supervised learning** 🡪 target/outcome is provided (**labelled data**)
* Classification + regression, generally
* **Unsupervised learning** 🡪 target/outcome is NOT known
* Clustering and association analysis, generally
* **Sample/Record/Observation =** instance/example of an **entity** of the data (i.e. a row in a dataset)
* **Variables/Features/Attribute/Dimension/Field** = different info pieces about samples
* Each has a data type associated w/ it 🡪 numerical, categorical, string, binary, date, etc.
* Numeric/Quantitative
* Can be measured and/or sorted
* Can be continuous or integers, and be positive, negative, or both
* Categorical/qualitative/Nominal
* Variable w/ labels, names, categories, etc. that describe characteristics of an entity
* **Scitkit-learn 🡪** ML library in Python
* Open-source + built on top of Numpy, SciPy, matplotlib
* Rapidly developed + improved by an active community of developers (800+ contributors)
* Includes end-to-end ML 🡪 entire data science process (i.e. cleaning, transformations, + ML)
* Supports whole data science process via functions for transformations, cleaning, prep, scaling, normalization, feature engineering, + missing value handling
* **Utility functions** for transforming raw features to suitable format
* **API** for scaling (remove mean + keep unit variance), normalization to have a unit norm, **binarization**, **One Hot Encoding** for categorical features, generating **high-order features**, building custom transforms, handling missing values
* Also has built-in functions for many ML algorithms, ready for modeling + analysis
* Still requires some expertise to use these for the right tasks, but many online resources smooth out the learning curve
* Documentation includes tutorials
* Has specialized implementations for **dimensionality reduction algorithms**
* Enables us to reduce features while preserving variance
* **Principal component analysis (PCA), single value decomposition, factor analysis, independent component analysis, matrix factorization, latent dirichlet allocation**
* Has methods for model selection
* **Cross-validation** methods
* Library functions for **tuning hyper parameters**
* **Model evaluation mechanisms** to measure model performance
* Plotting methods for visualizing scores to evaluate models
* **Decision Trees Goal** 🡪> split data into subsets where each subset belongs to only 1 class by dividing the input space into pure regions (regions w/ samples from only 1 class)
* Completely pure subsets are most likely not possible w/ real data
* Goal = get subsets *as pure as possible* (that contain as many samples of a single class as possible)



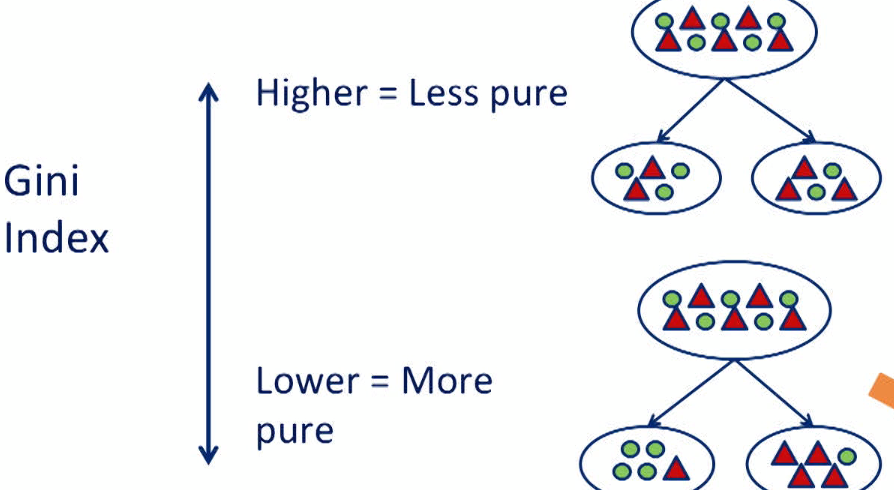
* Dividing regions via **decision boundaries** into regions that are as pure as possible
* Actual decision tree = hierarchical structure w/ **nodes** + **directed edges**
* **Root node** = node at top
* **Leaf/terminal nodes** = nodes at bottom
* **Internal nodes =** nodes neither the root nor a leaf
* Root + internal nodes have test conditions, and leaf nodes has a class label associated w/ them
* A classification decision is made by traversing the tree via test conditions at each node, starting at the root and ending at a leaf



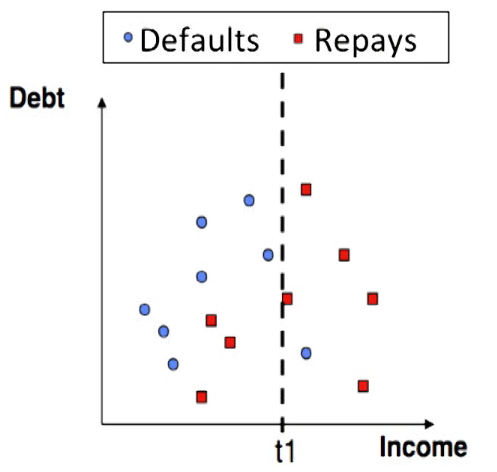
* **Depth** of a node = # of edges from the root node to that node
* Depth of root node = 0
* Depth of the *whole tree* is the # of edges in the longest path from the root to leaf
* **Size** of a decision tree = # of nodes in the tree



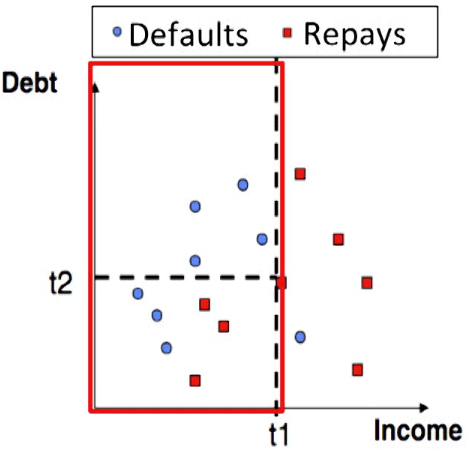
* DT’s are built w/ ALL samples at a single node (root) whilst adding additional nodes when data is split into subsets
* 1) All samples at root node 🡪 2) Partition samples based on input to create the purest subsets (as many samples as possible belonging to just 1 class = **homogenous**) 🡪 3) Repeat to partition into successive, more pure subsets until stopping criteria are satisfied
* Algorithm for constructing at DT model = **induction algorithm** (induction can be used to describe the process of building a DT)
* At each split, the induction algorithm only considers the *best* way to split the particular portion of data = **greedy approach**
* Solves a subset of a problem at a time + is necessary when solving the entire problem is not feasible (computable in a reasonable amount of time/space)
* It’s not feasible to build a DT outright given a dataset, so it must be built piecemeal by determining the best way to split a node at each step + combining these decisions together to make a tree
* Data is partitioned to result in subset as pure as possible (more homogenous = single class)
* Measure purity of a split by measuring the **impurity 🡪** specifies how mixed a resulting subset is
* Want homogenous class labels in a subset, so want the subset that minimizes the impurity
* **Gini index =** common impurity measure used to determine best splits (want lower Gini index score)



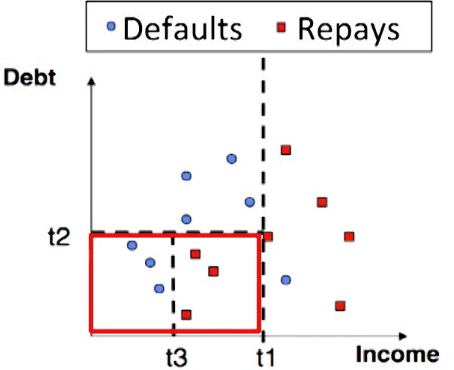
* Other impurity measures = **information gain (entropy), misclassification rate**
* Another factor in determining best partition of a node = which variable is best to split on?
* Tree tests all variables to determine best split of the node using Gini index to compare various possibilities
* Induction will partition nodes repeatedly to get more + more homogenous subsets and stops based on several criteria options
* When all samples in a node have the same class label (as pure as possible)
* 100% purity 🡪 more splitting will not increase classification accuracy
* Difficult to achieve w/ real data 🡪 usually modified to 90% homogenous
* When # of samples in a node reaches/falls below a minimum value
* # of samples is too small to make any difference in classification w/ more splits
* Change in impurity measure is smaller than a threshold
* Max tree depth is reached
* Controls complexity of the resulting DT
* Ex: Classify applicants as likely to repay a loan or not based on income + debt they have



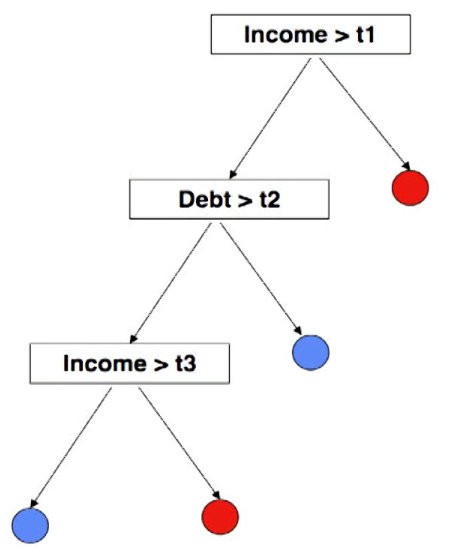
* 1) Can split dataset into subsets where income is > or < t1
* See more red > t1, and more blue < t1 (not completely homogenous)
* B/c right subset almost perfectly predicts lenders that *will* repay, we can predict those data points as red



* 2) Then split subset into datasets where debt > t2 or < t2
* See red < t2, blue in *both* regions
* Therefore, say that data points w/ debt > 2 will *not* repay the loan



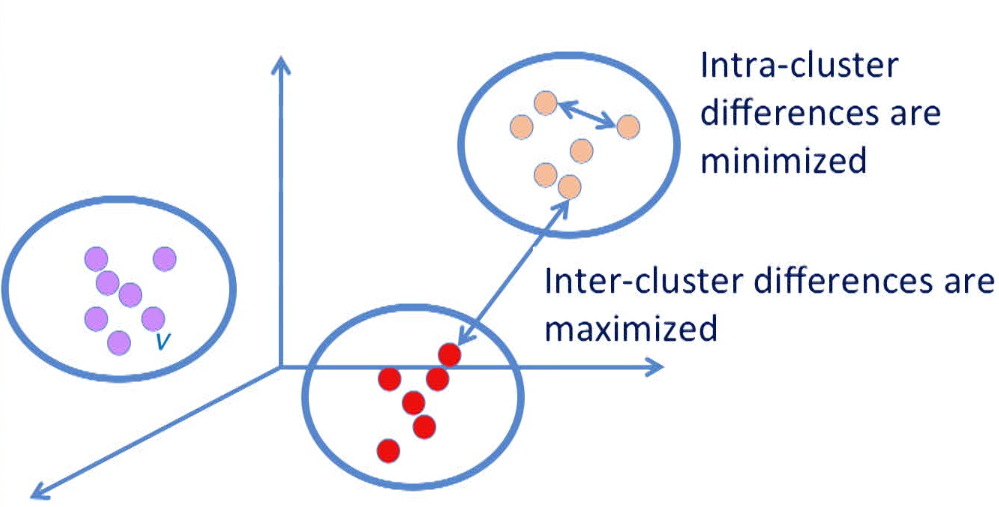
* 3) Split final subset into those w/ income > t3 and income < t3
* Best split says those w/ income > t3 = red and those w/ income < t3 = blue
* Final Tree: implemented decision boundaries from above implemented



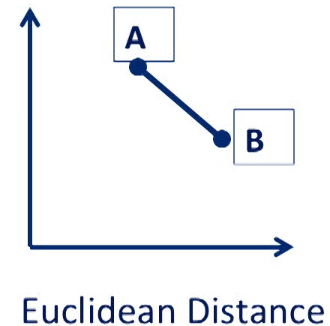
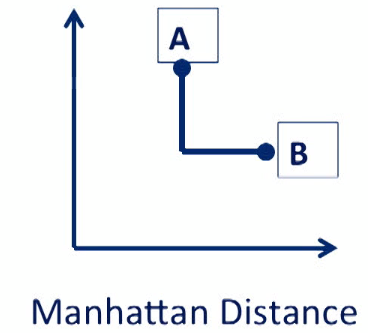
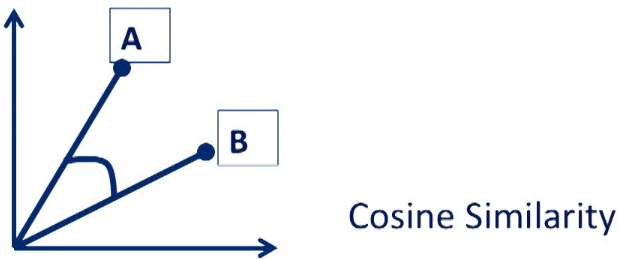
* Label for each region in input space is determined by the label of the majority of the samples, which are reflected in the leaf nodes of the decision tree
* Notice that the decision boundaries in the input space are parallel to the axis formed by the variable = **rectilinear**
* Decisions boundaries are rectilinear b/c each split only considers a single variable
* Some variants of the induction algorithm consider more than 1 attribute when splitting a value, but each split must consider all combinations of combined variables
* Therefore these variations are computationally expensive/intensive
* DT’s are often simple to understand/interpret (1 of their biggest advantages)
* Can see which variables are important to the classification + understand how the classification is performed
* For this reason, many start out w/ DT’s to get a feel for the problem, even if they end up using a more sophisticated model
* DT’s are relatively inexpensive/fast
* Note: Greedy approach determines best way to split a proportion *at given node*, but does NOT guarantee best solution for overall dataset
* Rectilinear decision boundaries can limit expressiveness of a resulting model
* may not be able to solve complicated classification problems that require more complex decision boundaries to be formed

***CLUSTERING***

* Clustering goal = organize similar items into groups/clusters by segmenting datasets
* Can then analyze clusters more carefully
* Common application = customer segmentation based on purchase history for targeted marketing
* Others: weather patterns for a region, news articles into topics to ID trending topics, discovering hotspots for different types of crimes from police reports to provide sufficient police presence for problem areas



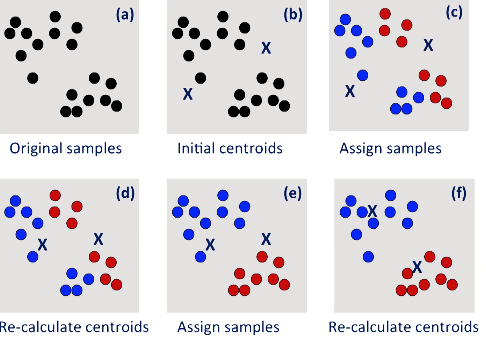
* Which group an item is placed in is based on some measure of similarity
* Want the differences between *samples* to be *minimized* (blue arrow between yellow dots above) + the differences between *clusters* to be *maximized* (blue arrow between red + yellow dots)
* Want samples w/in clusters to be as close as possible + samples between clusters to be as far as possible
* Common similarity measures:
* **Euclidean distance =** distance along a straight line between 2 points
* **Manhattan distance =** calculated on a strictly horizontal + vertical path (can only step along x or y axis in 2D case)
* **Cosine Similarity** = cosine of the angle between points A + B

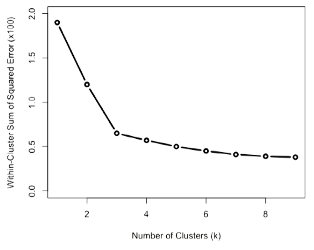
* Since distance measures are often used to measure similarity between samples in clustering algorithms, it may be necessary to **normalize** input variables so that no 1 value dominates the similarity calculation.
* **Normalizing =** putall values on same scale that they all have equal weighting in the calculation
* **Scaling** is very necessary w/ variables on very different scales (height vs. weight) to put them on a common value range
* Clustering is **Unsupervised =** no target value for any sample/record in the dataset
* In general, those is no “correct” clustering results + the best ones are dependent on the application + how the clusters will be used
* There are some numerical measures to compare clusters, but since there are no labels to signify if a sample was clustered “correctly”, there’s no ground truth to determine if a set of clusters is truly “correct” or not
* Clusters don’t come w/ labels 🡪 even when we end up w/ 5 clusters at the end of the algorithm, we don’t know what they represent
* Only by analyzing samples in each cluster further can we come up w/ reasonable labels for them
* **Interpretation and analysis of clusters are required to make sense of the results**
* Most obvious way results of clusters can be used via segmentation, wherein analysis of segments can provide insights into preferences and purchasing history (customers who like sci-fi, non-fiction, or children’s books) to help provide more effective marketing
* Can also use clusters to ID new datasets 🡪 see which cluster a new sample is closes to via a similarity measure between the sample + the centers of all clusters
* *Once cluster labels are determined, samples in each cluster can be used as labeled data for another classification task*
* Cluster labels would be the **target class** for input data from new samples
* *So clusters can provide much-needed label data for classification problems*
* Another use of cluster results is for **anomaly detections** 🡪 samples are very far away/different from *any* cluster center
* Anomalies require further analysis + studied more carefully (may be removed based on the application, such as age = 150)

***K-MEAN CLUSTERING***

* k-means is a classic clustering algorithm that begins w/ selecting k initial **centroids** = center of a cluster
* Then we assign each data point to the centroid closest to it (calculate distance between DP and each centroid and pick the shortest one)
* Then calculate mean of each cluster to determine *new* *centroids* and repeat these 2 steps until some stopping criteria is reached.



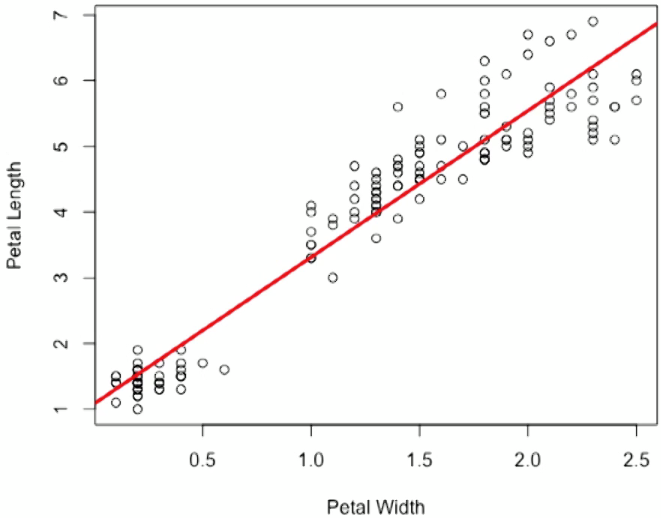
* One issue w/ k-means is that final clusters are sensitive to the choices of initial centroids
* Cluster results w/ 1 set of initial centroids could be different from another set of initial centroids
* Easiest and most widely approach to selecting a good initial # of centroids is to run k-means several times w/ different initial centroids chosen randomly
* Then we choose the centroids that gave the “best” clustering results
* **Within-Cluster Sum of Squared Errors** (**WCSSE**) can be used to evaluate cluster results
* Error = distance between DP + centroid 🡪 square it 🡪 the sum all squared errors of a cluster
* Then sum up SSE for *all clusters*
* Want the smallest WCSSE, which is the best solution *numerically*
* This does NOT guarantee that cluster result 2 is better than cluster result 1
* Smallest WCSSE is not always the best solution for an application at hand
* Domain knowledge and interpretation are critical for determining what a cluster should represent and how they’ll be used
* There is NO mathematically-ground truth for determining which set of clusters gives the best result
* ***NOTE****: Increasing k automatically reduces WCSSE, so it should be used w/ caution as a measure of performance*
* Only makes sense to use it when comparing 2 cluster results w/ same k value from same dataset
* There are several other method for evaluating cluster results, as well
* Choosing optimal value for k is always an important question for k-means
* There are several methods for determining this
* Visualization can be used to view the dataset to see if there are any natural grouping of samples
* Scatterplots and dimensionality reduction are good here
* Good k-values are application-dependent = domain knowledge of an application can drive the selection for the value of k
* Ex: clustering types of products a customer is purchasing, natural choice for k = # of broad product categories you offer
* Ex: clustering geographic regions of respondents to a survey, natural choice for k = # of regions you’re interested in analyzing
* Data-driven methods can calculate some metric for different values of k to determine the best k
* Ex: **Elbow method** = Plot WCSSE for different values of k to see how the error measure changes for each value of k



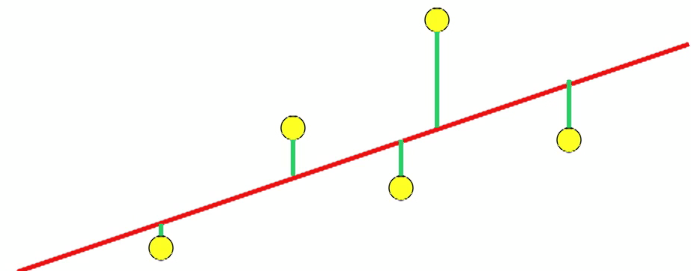
* The “bend” in the error curve indicates a drop in gain from adding more clusters
* This curve suggests a k value of 3 is where marginal gains start
* The elbow *cannot always be unambiguously determined, especially w/ complex data*
* Curve can have multiple suggestions for values which can be used as a guideline for a range of values to try for k
* 1 obvious **stopping criterion** is when *there are no changes to the centroids*
* i.e. no samples will change cluster assignments + recalculating centroids doesn’t result in a change
* We could relax this stopping criterion to instead say to stop when the # of samples changing clusters is below a certain threshold, such as 1%
* So clusters are only changing by a couple of samples, resulting in minimal changes to clustering results
* At the end of k-means, we have a set of clusters, each w/ a centroid
* Each centroid is the mean of the samples assigned to that cluster (think of centroid as a representative sample for that cluster)
* To examine cluster results, examine cluster centroids
* Comparing variables from each cluster reveals how alike/different clusters are and provides insight into what each cluster represents
* Ex: if the value for age is different for different customer clusters, this indicates that the clusters are different customers groups segmented by age

***REGRESSION ANALYSIS***

* Compared to classification, instead of categories, w/ regression we’re predicting numerical values (# of kids, stock price, profits, forecasting high and low temperatures, power usage for a smart grid, average housing price for an area, demand for new product, inventory needs, etc.)
* This is a supervised learning algorithm wherein the target/outcome is known/provided
* Regression also involved 2 phases, training + testing, just like classification w/ the goal to have the model perform well on training data + generalize well on new data (test)
* Training data is used to train the model = adjust parameters to learn the input-to-output mapping
* Test data is used to evaluate the model (on new or leftover data)
* A new dataset for this is **validation datasets** 🡪 used to determine when to stop training in order to avoid model **overfitting** to the training data
* **Linear regression** is a simple, powerful, and popular ML algorithm
* It captures the relationship between the inputs and the numerical output and models the relationship linearly



* Want to predict petal length based on petal width (given width, what is length?)
* Can build a LR model to capture this linear relationship between the input width + output length
* It works by finding the **regression line** = best-fit straight line through the sample DP’s
* In a simple case w/ just 1 input, we have a simple line 🡪 y = mx + b where m (slope) and b (intercept) are parameters that we adjust in training
* The regression line can be determine via the **least squares method**



* Want the minimum sum of the squared errors (**residuals**) from the prediction line (red) and the actual DP’s values (yellow dots)
* Goal of LR = find best fitting-straight line through the data via the least square method to make sum of residuals as small as possible
* Once built, we can use the LR model to make predictions given new data (new petal width) based on the regression line created

