

Data Science Cheatsheet

Compiled by Maverick Lin (<http://mavericklin.com>)
Last Updated April 19, 2020

What is Data Science?

Multi-disciplinary field that brings together concepts from computer science, statistics/machine learning, and data analysis to understand and extract insights from the ever-increasing amounts of data.

Two paradigms of data research.

1. **Hypothesis-Driven:** Given a problem, what kind of data do we need to help solve it?
2. **Data-Driven:** Given some data, what interesting problems can be solved with it?

The heart of data science is to always ask questions. Always be curious about the world.

1. What can we learn from this data?
2. What actions can we take once we find whatever it is we are looking for?

Types of Data

Structured: Data that has predefined structures. e.g. tables, spreadsheets, or relational databases.

Unstructured Data: Data with no predefined structure, comes in any size or form, cannot be easily stored in tables. e.g. blobs of text, images, audio

Quantitative Data: Numerical. e.g. height, weight

Categorical Data: Data that can be labeled or divided into groups. e.g. race, sex, hair color.

Big Data: Massive datasets, or data that contains greater *variety* arriving in increasing *volumes* and with ever-higher *velocity* (3 Vs). Cannot fit in the memory of a single machine.

Data Sources/Formats

Most Common Data Formats CSV, XML, SQL, JSON, Protocol Buffers

Data Sources Companies/Proprietary Data, APIs, Government, Academic, Web Scraping/Crawling

Main Types of Problems

Two problems arise repeatedly in data science.

Classification: Assigning something to a discrete set of possibilities. e.g. spam or non-spam, Democrat or Republican, blood type (A, B, AB, O)

Regression: Predicting a numerical value. e.g. someone's income, next year GDP, stock price

Probability Overview

Probability theory provides a framework for reasoning about likelihood of events.

Terminology

Experiment: procedure that yields one of a possible set of outcomes e.g. repeatedly tossing a die or coin

Sample Space S: set of possible outcomes of an experiment e.g. if tossing a die, $S = (1,2,3,4,5,6)$

Event E: set of outcomes of an experiment e.g. event that a roll is 5, or the event that sum of 2 rolls is 7

Probability of an Outcome s or P(s): number that satisfies 2 properties

1. for each outcome s, $0 \leq P(s) \leq 1$
2. $\sum p(s) = 1$

Probability of Event E: sum of the probabilities of the outcomes of the experiment: $p(E) = \sum_{s \in E} p(s)$

Random Variable V: numerical function on the outcomes of a probability space

Expected Value of Random Variable V: $E(V) = \sum_{s \in S} p(s) * V(s)$

Independence, Conditional, Compound

Independent Events: A and B are independent iff:

$$\begin{aligned} P(A \cap B) &= P(A)P(B) \\ P(A|B) &= P(A) \\ P(B|A) &= P(B) \end{aligned}$$

Conditional Probability: $P(A|B) = P(A, B) / P(B)$

Bayes Theorem: $P(A|B) = P(B|A)P(A) / P(B)$

Joint Probability: $P(A, B) = P(B|A)P(A)$

Marginal Probability: $P(A)$

Probability Distributions

Probability Density Function (PDF) Gives the probability that a rv takes on the value x: $p_X(x) = P(X = x)$

Cumulative Density Function (CDF) Gives the probability that a random variable is less than or equal to x: $F_X(x) = P(X \leq x)$

Note: The PDF and the CDF of a given random variable contain exactly the same information.

Descriptive Statistics

Provides a way of capturing a given data set or sample. There are two main types: **centrality** and **variability** measures.

Centrality

Arithmetic Mean Useful to characterize symmetric distributions without outliers $\mu_X = \frac{1}{n} \sum x$

Geometric Mean Useful for averaging ratios. Always less than arithmetic mean $= \sqrt[n]{a_1 a_2 \dots a_n}$

Median Exact middle value among a dataset. Useful for skewed distribution or data with outliers.

Mode Most frequent element in a dataset.

Variability

Standard Deviation Measures the squares differences between the individual elements and the mean

$$\sigma = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N-1}}$$

Variance $V = \sigma^2$

Interpreting Variance

Variance is an inherent part of the universe. It is impossible to obtain the same results after repeated observations of the same event due to random noise/error. Variance can be explained away by attributing to sampling or measurement errors. Other times, the variance is due to the random fluctuations of the universe.

Correlation Analysis

Correlation coefficients $r(X, Y)$ is a statistic that measures the degree that Y is a function of X and vice versa. Correlation values range from -1 to 1, where 1 means fully correlated, -1 means negatively-correlated, and 0 means no correlation.

Pearson Coefficient Measures the degree of the relationship between linearly related variables

$$r = \frac{Cov(X, Y)}{\sigma(X)\sigma(Y)}$$

Spearman Rank Coefficient Computed on ranks and depicts monotonic relationships

Note: Correlation does not imply causation!

Data Cleaning

Data Cleaning is the process of turning raw data into a clean and analyzable data set. "Garbage in, garbage out." Make sure garbage doesn't get put in.

Errors vs. Artifacts

1. **Errors:** information that is lost during acquisition and can never be recovered e.g. power outage, crashed servers
2. **Artifacts:** systematic problems that arise from the data cleaning process. these problems can be corrected but we must first discover them

Data Compatibility

Data compatibility problems arise when merging datasets. Make sure you are comparing "apples to apples" and not "apples to oranges". Main types of conversions/unifications:

- **units** (metric vs. imperial)
- **numbers** (decimals vs. integers),
- **names** (John Smith vs. Smith, John),
- **time/dates** (UNIX vs. UTC vs. GMT),
- **currency** (currency type, inflation-adjusted, dividends)

Data Imputation

Process of dealing with missing values. The proper methods depend on the type of data we are working with. General methods include:

- Drop all records containing missing data
- Heuristic-Based: make a reasonable guess based on knowledge of the underlying domain
- Mean Value: fill in missing data with the mean
- Random Value
- Nearest Neighbor: fill in missing data using similar data points
- Interpolation: use a method like linear regression to predict the value of the missing data

Outlier Detection

Outliers can interfere with analysis and often arise from mistakes during data collection. It makes sense to run a "sanity check".

Miscellaneous

Lowercasing, removing non-alphanumeric, repairing, unicode, removing unknown characters

Note: When cleaning data, always maintain both the raw data and the cleaned version(s). The raw data should be kept intact and preserved for future use. Any type of data cleaning/analysis should be done on a copy of the raw data.

Feature Engineering

Feature engineering is the process of using domain knowledge to create features or input variables that help machine learning algorithms perform better. Done correctly, it can help increase the predictive power of your models. Feature engineering is more of an art than science. FE is one of the most important steps in creating a good model. As Andrew Ng puts it:

"Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."

Continuous Data

Raw Measures: data that hasn't been transformed yet

Rounding: sometimes precision is noise; round to nearest integer, decimal etc..

Scaling: log, z-score, minmax scale

Imputation: fill in missing values using mean, median, model output, etc..

Binning: transforming numeric features into categorical ones (or binned) e.g. values between 1-10 belong to A, between 10-20 belong to B, etc.

Interactions: interactions between features: e.g. subtraction, addition, multiplication, statistical test

Statistical: log/power transform (helps turn skewed distributions more normal), Box-Cox

Row Statistics: number of NaN's, 0's, negative values, max, min, etc

Dimensionality Reduction: using PCA, clustering, factor analysis etc

Discrete Data

Encoding: since some ML algorithms cannot work on categorical data, we need to turn categorical data into numerical data or vectors

Ordinal Values: convert each distinct feature into a random number (e.g. [r,g,b] becomes [1,2,3])

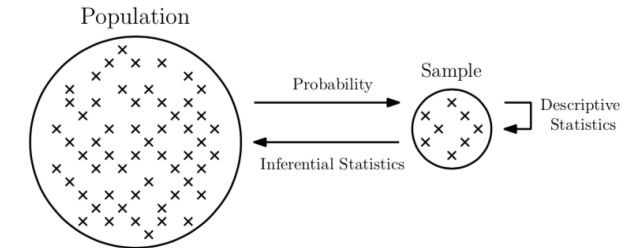
One-Hot Encoding: each of the m features becomes a vector of length m with containing only one 1 (e.g. [r, g, b] becomes [[1,0,0],[0,1,0],[0,0,1]])

Feature Hashing Scheme: turns arbitrary features into indices in a vector or matrix

Embeddings: if using words, convert words to vectors (word embeddings)

Statistical Analysis

Process of statistical reasoning: there is an underlying population of possible things we can potentially observe and only a small subset of them are actually sampled (ideally at random). Probability theory describes what properties our sample should have given the properties of the population, but **statistical inference** allows us to deduce what the full population is like after analyzing the sample.



Sampling From Distributions

Inverse Transform Sampling Sampling points from a given probability distribution is sometimes necessary to run simulations or whether your data fits a particular distribution. The general technique is called *inverse transform sampling* or Smirnov transform. First draw a random number p between $[0,1]$. Compute value x such that the CDF equals p : $F_X(x) = p$. Use x as the value to be the random value drawn from the distribution described by $F_X(x)$.

Monte Carlo Sampling In higher dimensions, correctly sampling from a given distribution becomes more tricky. Generally want to use Monte Carlo methods, which typically follow these rules: define a domain of possible inputs, generate random inputs from a probability distribution over the domain, perform a deterministic calculation, and analyze the results.

Classic Statistical Distributions

Binomial Distribution (Discrete)

Assume X is distributed $\text{Bin}(n, p)$. X is the number of "successes" that we will achieve in n independent trials, where each trial is either a success or failure and each success occurs with the same probability p and each failure occurs with probability $q=1-p$.

PDF: $P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$

EV: $\mu = np$ Variance = npq

Normal/Gaussian Distribution (Continuous)

Assume X is distributed $\mathcal{N}(\mu, \sigma^2)$. It is a bell-shaped and symmetric distribution. Bulk of the values lie close to the mean and no value is too extreme. Generalization of the binomial distribution as $n \rightarrow \infty$.

PDF: $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$

EV: μ Variance: σ^2

Implications: 68%-95%-99% rule. 68% of probability mass fall within 1σ of the mean, 95% within 2σ , and 99.7% within 3σ .

Poisson Distribution (Discrete)

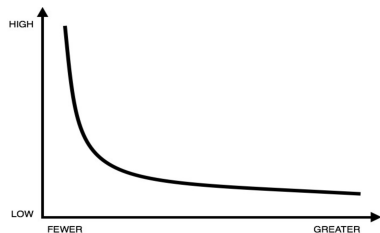
Assume X is distributed $\text{Pois}(\lambda)$. Poisson expresses the probability of a given number of events occurring in a fixed interval of time/space if these events occur independently and with a known constant rate λ .

PDF: $P(x) = \frac{e^{-\lambda} \lambda^x}{x!}$ EV: λ Variance = λ

Power Law Distributions (Discrete)

Many data distributions have much longer tails than the normal or Poisson distributions. In other words, the change in one quantity varies as a *power* of another quantity. It helps measure the inequality in the world. e.g. wealth, word frequency and Pareto Principle (80/20 Rule)

PDF: $P(X=x) = cx^{-\alpha}$, where α is the law's exponent and c is the normalizing constant



Modeling- Overview

Modeling is the process of incorporating information into a tool which can forecast and make predictions. Usually, we are dealing with statistical modeling where we want to analyze relationships between variables. Formally, we want to estimate a function $f(X)$ such that:

$$Y = f(X) + \epsilon$$

where $X = (X_1, X_2, \dots, X_p)$ represents the input variables, Y represents the output variable, and ϵ represents random error.

Statistical learning is set of approaches for estimating this $f(X)$.

Why Estimate $f(X)$?

Prediction: once we have a good estimate $\hat{f}(X)$, we can use it to make predictions on new data. We treat \hat{f} as a black box, since we only care about the accuracy of the predictions, not why or how it works.

Inference: we want to understand the relationship between X and Y . We can no longer treat \hat{f} as a black box since we want to understand how Y changes with respect to $X = (X_1, X_2, \dots, X_p)$

More About ϵ

The error term ϵ is composed of the reducible and irreducible error, which will prevent us from ever obtaining a perfect \hat{f} estimate.

- **Reducible:** error that can potentially be reduced by using the most appropriate statistical learning technique to estimate f . The goal is to minimize the reducible error.
- **Irreducible:** error that cannot be reduced no matter how well we estimate f . Irreducible error is unknown and unmeasurable and will always be an upper bound for ϵ .

Note: There will always be trade-offs between model flexibility (prediction) and model interpretability (inference). This is just another case of the bias-variance trade-off. Typically, as flexibility increases, interpretability decreases. Much of statistical learning/modeling is finding a way to balance the two.

Modeling- Philosophies

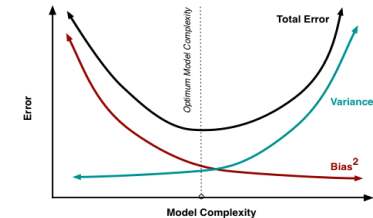
Modeling is the process of incorporating information into a tool which can forecast and make predictions. Designing and validating models is important, as well as evaluating the performance of models. Note that the best forecasting model may not be the most accurate one.

Philosophies of Modeling

Occam's Razor Philosophical principle that the simplest explanation is the best explanation. In modeling, if we are given two models that predicts equally well, we should choose the simpler one. Choosing the more complex one can often result in overfitting.

Bias Variance Trade-Off Inherent part of predictive modeling, where models with lower bias will have higher variance and vice versa. Goal is to achieve low bias and low variance.

- **Bias:** error from incorrect assumptions to make target function easier to learn (high bias \rightarrow missing relevant relations or underfitting)
- **Variance:** error from sensitivity to fluctuations in the dataset, or how much the target estimate would differ if different training data was used (high variance \rightarrow modeling noise or overfitting)



No Free Lunch Theorem No single machine learning algorithm is better than all the others on all problems. It is common to try multiple models and find one that works best for a particular problem.

Thinking Like Nate Silver

1. Think Probabilistically Probabilistic forecasts are more meaningful than concrete statements and should be reported as probability distributions (including σ along with mean prediction μ).

2. Incorporate New Information Use live models, which continually updates using new information. To update, use Bayesian reasoning to calculate how probabilities change in response to new evidence.

3. Look For Consensus Forecast Use multiple distinct sources of evidence. Some models operate this way, such as boosting and bagging, which uses large number of weak classifiers to produce a strong one.

Modeling- Taxonomy

There are many different types of models. It is important to understand the trade-offs and when to use a certain type of model.

Parametric vs. Nonparametric

- **Parametric:** models that first make an assumption about a function form, or shape, of f (linear). Then fits the model. This reduces estimating f to just estimating set of parameters, but if our assumption was wrong, will lead to bad results.
- **Non-Parametric:** models that don't make any assumptions about f , which allows them to fit a wider range of shapes; but may lead to overfitting

Supervised vs. Unsupervised

- **Supervised:** models that fit input variables $x_i = (x_1, x_2, \dots, x_n)$ to a known output variables $y_i = (y_1, y_2, \dots, y_n)$
- **Unsupervised:** models that take in input variables $x_i = (x_1, x_2, \dots, x_n)$, but they do not have an associated output to supervise the training. The goal is understand relationships between the variables or observations.

Blackbox vs. Descriptive

- **Blackbox:** models that make decisions, but we do not know what happens "under the hood" e.g. deep learning, neural networks
- **Descriptive:** models that provide insight into *why* they make their decisions e.g. linear regression, decision trees

First-Principle vs. Data-Driven

- **First-Principle:** models based on a prior belief of how the system under investigation works, incorporates domain knowledge (ad-hoc)
- **Data-Driven:** models based on observed correlations between input and output variables

Deterministic vs. Stochastic

- **Deterministic:** models that produce a single "prediction" e.g. yes or no, true or false
- **Stochastic:** models that produce probability distributions over possible events

Flat vs. Hierarchical

- **Flat:** models that solve problems on a single level, no notion of subproblems
- **Hierarchical:** models that solve several different nested subproblems

Modeling- Evaluation Metrics

Need to determine how good our model is. Best way to assess models is out-of-sample predictions (data points your model has never seen).

Classification

	Predicted Yes	Predicted No
Actual Yes	True Positives (TP)	False Negatives (FN)
Actual No	False Positives (FP)	True Negatives (TN)

Accuracy: ratio of correct predictions over total predictions. Misleading when class sizes are substantially different. $accuracy = \frac{TP+TN}{TP+TN+FN+FP}$

Precision: how often the classifier is correct when it predicts positive: $precision = \frac{TP}{TP+FP}$

Recall: how often the classifier is correct for all positive instances: $recall = \frac{TP}{TP+FN}$

F-Score: single measurement to describe performance: $F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

ROC Curves: plots true positive rates and false positive rates for various thresholds, or where the model determines if a data point is positive or negative (e.g. if >0.8 , classify as positive). Best possible area under the ROC curve (AUC) is 1, while random is 0.5, or the main diagonal line.

Regression

Errors are defined as the difference between a prediction y' and the actual result y .

Absolute Error: $\Delta = y' - y$

Squared Error: $\Delta^2 = (y' - y)^2$

Mean-Squared Error: $MSE = \frac{1}{n} \sum_{i=1}^n (y'_i - y_i)^2$

Root Mean-Squared Error: $RMSD = \sqrt{MSE}$

Absolute Error Distribution: Plot absolute error distribution: should be symmetric, centered around 0, bell-shaped, and contain rare extreme outliers.

Modeling- Evaluation Environment

Evaluation metrics provides use with the tools to estimate errors, but what should be the process to obtain the best estimate? Resampling involves repeatedly drawing samples from a training set and refitting a model to each sample, which provides us with additional information compared to fitting the model once, such as obtaining a better estimate for the test error.

Key Concepts

Training Data: data used to fit your models or the set used for learning

Validation Data: data used to tune the parameters of a model

Test Data: data used to evaluate how good your model is. Ideally your model should never touch this data until final testing/evaluation

Cross Validation

Class of methods that estimate test error by holding out a subset of training data from the fitting process.

Validation Set: split data into training set and validation set. Train model on training and estimate test error using validation. e.g. 80-20 split

Leave-One-Out CV (LOOCV): split data into training set and validation set, but the validation set consists of 1 observation. Then repeat $n-1$ times until all observations have been used as validation. Test error is the average of these n test error estimates.

k-Fold CV: randomly divide data into k groups (folds) of approximately equal size. First fold is used as validation and the rest as training. Then repeat k times and find average of the k estimates.

Bootstrapping

Methods that rely on random sampling with replacement. Bootstrapping helps with quantifying uncertainty associated with a given estimate or model.

Amplifying Small Data Sets

What can we do if we don't have enough data?

- **Create Negative Examples-** e.g. classifying presidential candidates, most people would be unqualified so label most as unqualified
- **Synthetic Data-** create additional data by adding noise to the real data

Linear Regression

Linear regression is a simple and useful tool for predicting a quantitative response. The relationship between input variables $X = (X_1, X_2, \dots, X_p)$ and output variable Y takes the form:

$$Y \approx \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon$$

$\beta_0 \dots \beta_p$ are the unknown coefficients (parameters) which we are trying to determine. The best coefficients will lead us to the best "fit", which can be found by minimizing the *residual sum squares* (RSS), or the sum of the squared differences between the actual i th value and the predicted i th value. $RSS = \sum_{i=1}^n e_i^2$, where $e_i = y_i - \hat{y}_i$

How to find best fit?

Matrix Form: We can solve the closed-form equation for coefficient vector w : $w = (X^T X)^{-1} X^T Y$. X represents the input data and Y represents the output data. This method is used for smaller matrices, since inverting a matrix is computationally expensive.

Gradient Descent: First-order optimization algorithm. We can find the minimum of a *convex* function by starting at an arbitrary point and repeatedly take steps in the downward direction, which can be found by taking the negative direction of the gradient. After several iterations, we will eventually converge to the minimum. In our case, the minimum corresponds to the coefficients with the minimum error, or the best line of fit. The learning rate α determines the size of the steps we take in the downward direction.

Gradient descent algorithm in two dimensions. Repeat until convergence.

1. $w_0^{t+1} := w_0^t - \alpha \frac{\partial}{\partial w_0} J(w_0, w_1)$
2. $w_1^{t+1} := w_1^t - \alpha \frac{\partial}{\partial w_1} J(w_0, w_1)$

For non-convex functions, gradient descent no longer guarantees an optimal solution since there may be local minimas. Instead, we should run the algorithm from different starting points and use the best local minima we find for the solution.

Stochastic Gradient Descent: instead of taking a step after sampling the *entire* training set, we take a small batch of training data at random to determine our next step. Computationally more efficient and may lead to faster convergence.

Linear Regression II

Improving Linear Regression

Subset/Feature Selection: approach involves identifying a subset of the p predictors that we believe to be best related to the response. Then we fit model using the reduced set of variables.

- Best, Forward, and Backward Subset Selection

Shrinkage/Regularization: all variables are used, but estimated coefficients are shrunk towards zero relative to the least squares estimate. λ represents the tuning parameter- as λ increases, flexibility decreases \rightarrow decreased variance but increased bias. The tuning parameter is key in determining the sweet spot between under and over-fitting. In addition, while Ridge will always produce a model with p variables, Lasso can force coefficients to be equal to zero.

- Lasso (L1): $\min RSS + \lambda \sum_{j=1}^p |\beta_j|$
- Ridge (L2): $\min RSS + \lambda \sum_{j=1}^p \beta_j^2$

Dimension Reduction: projecting p predictors into a M -dimensional subspace, where $M < p$. This is achieved by computing M different linear combinations of the variables. Can use PCA.

Miscellaneous: Removing outliers, feature scaling, removing multicollinearity (correlated variables)

Evaluating Model Accuracy

Residual Standard Error (RSE): $RSE = \sqrt{\frac{1}{n-2} RSS}$. Generally, the smaller the better.

R^2 : Measure of fit that represents the proportion of variance explained, or the *variability in Y that can be explained using X* . It takes on a value between 0 and 1. Generally the higher the better. $R^2 = 1 - \frac{RSS}{TSS}$, where Total Sum of Squares (TSS) = $\sum (y_i - \bar{y})^2$

Evaluating Coefficient Estimates

Standard Error (SE) of the coefficients can be used to perform hypothesis tests on the coefficients:

H_0 : No relationship between X and Y , H_a : Some relationship exists. A p-value can be obtained and can be interpreted as follows: a small p-value indicates that a relationship between the predictor (X) and the response (Y) exists. Typical p-value cutoffs are around 5 or 1 %.

Logistic Regression

Logistic regression is used for classification, where the response variable is categorical rather than numerical.

The model works by predicting the probability that Y belongs to a particular category by first fitting the data to a linear regression model, which is then passed to the logistic function (below). The logistic function will always produce a S-shaped curve, so regardless of X , we can always obtain a sensible answer (between 0 and 1). If the probability is above a certain predetermined threshold (e.g. $P(\text{Yes}) > 0.5$), then the model will predict Yes.

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}$$

How to find best coefficients?

Maximum Likelihood: The coefficients $\beta_0 \dots \beta_p$ are unknown and must be estimated from the training data. We seek estimates for $\beta_0 \dots \beta_p$ such that the predicted probability $\hat{p}(x_i)$ of each observation is a number close to one if its observed in a certain class and close to zero otherwise. This is done by maximizing the likelihood function:

$$l(\beta_0, \beta_1) = \prod_{i: y_i=1} p(x_i) \prod_{i': y_{i'}=1} (1 - p(x_{i'}))$$

Potential Issues

Imbalanced Classes: imbalance in classes in training data lead to poor classifiers. It can result in a lot of false positives and also lead to few training data. Solutions include forcing balanced data by removing observations from the larger class, replicate data from the smaller class, or heavily weigh the training examples toward instances of the larger class.

Multi-Class Classification: the more classes you try to predict, the harder it will be for the classifier to be effective. It is possible with logistic regression, but another approach, such as Linear Discriminant Analysis (LDA), may prove better.

Distance/Network Methods

Interpreting examples as points in space provides a way to find natural groupings or clusters among data e.g. which stars are the closest to our sun? Networks can also be built from point sets (vertices) by connecting related points.

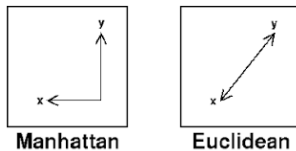
Measuring Distances/Similarity Measure

There are several ways of measuring distances between points a and b in d dimensions- with closer distances implying similarity.

Minkowski Distance Metric: $d_k(a, b) = \sqrt[k]{\sum_{i=1}^d |a_i - b_i|^k}$

The parameter k provides a way to tradeoff between the largest and the total dimensional difference. In other words, larger values of k place more emphasis on large differences between feature values than smaller values. Selecting the right k can significantly impact the the meaningfulness of your distance function. The most popular values are 1 and 2.

- Manhattan ($k=1$): city block distance, or the sum of the absolute difference between two points
- Euclidean ($k=2$): straight line distance



Weighted Minkowski: $d_k(a, b) = \sqrt[k]{\sum_{i=1}^d w_i |a_i - b_i|^k}$, in some scenarios, not all dimensions are equal. Can convey this idea using w_i . Generally not a good idea- should normalize data by Z-scores before computing distances.

Cosine Similarity: $\cos(a, b) = \frac{a \cdot b}{|a||b|}$, calculates the similarity between 2 non-zero vectors, where $a \cdot b$ is the dot product (normalized between 0 and 1), higher values imply more similar vectors

Kullback-Leibler Divergence: $KL(A||B) = \sum_{i=1}^d a_i \log_2 \frac{a_i}{b_i}$ KL divergence measures the distances between probability distributions by measuring the uncertainty gained or uncertainty lost when replacing distribution A with distribution B. However, this is not a metric but forms the basis for the Jensen-Shannon Divergence Metric.

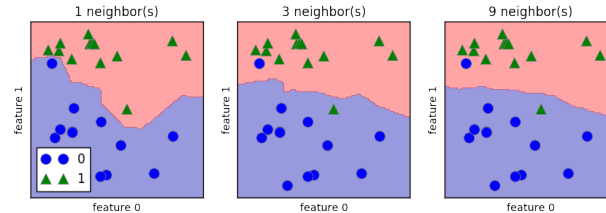
Jensen-Shannon: $JS(A, B) = \frac{1}{2}KL(A||M) + \frac{1}{2}KL(M||B)$, where M is the average of A and B. The JS function is the right metric for calculating distances between probability distributions

Nearest Neighbor Classification

Distance functions allow us to identify the points closest to a given target, or the *nearest neighbors (NN)* to a given point. The advantages of NN include simplicity, interpretability and non-linearity.

k-Nearest Neighbors

Given a positive integer k and a point x_0 , the KNN classifier first identifies k points in the training data most similar to x_0 , then estimates the conditional probability of x_0 being in class j as the fraction of the k points whose values belong to j . The optimal value for k can be found using cross validation.



KNN Algorithm

1. Compute distance $D(a, b)$ from point b to all points
2. Select k closest points and their labels
3. Output class with most frequent labels in k points

Optimizing KNN

Comparing a query point a in d dimensions against n training examples computes with a runtime of $O(nd)$, which can cause lag as points reach millions or billions. Popular choices to speed up KNN include:

- **Vernoi Diagrams:** partitioning plane into regions based on distance to points in a specific subset of the plane
- **Grid Indexes:** carve up space into d -dimensional boxes or grids and calculate the NN in the same cell as the point
- **Locality Sensitive Hashing (LSH):** abandons the idea of finding the exact nearest neighbors. Instead, batch up nearby points to quickly find the most appropriate bucket B for our query point. LSH is defined by a hash function $h(p)$ that takes a point/vector as input and produces a number/ code as output, such that it is likely that $h(a) = h(b)$ if a and b are close to each other, and $h(a) \neq h(b)$ if they are far apart.

Clustering

Clustering is the problem of grouping points by similarity using distance metrics, which ideally reflect the similarities you are looking for. Often items come from logical "sources" and clustering is a good way to reveal those origins. Perhaps the first thing to do with any data set. Possible applications include: hypothesis development, modeling over smaller subsets of data, data reduction, outlier detection.

K-Means Clustering

Simple and elegant algorithm to partition a dataset into K distinct, non-overlapping clusters.

1. Choose a K . Randomly assign a number between 1 and K to each observation. These serve as initial cluster assignments
2. Iterate until cluster assignments stop changing
 - (a) For each of the K clusters, compute the cluster centroid. The k th cluster centroid is the vector of the p feature means for the observations in the k th cluster.
 - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using distance metric).

Since the results of the algorithm depends on the initial random assignments, it is a good idea to repeat the algorithm from different random initializations to obtain the best overall results. Can use MSE to determine which cluster assignment is better.

Hierarchical Clustering

Alternative clustering algorithm that does not require us to commit to a particular K . Another advantage is that it results in a nice visualization called a **dendrogram**. Observations that fuse at bottom are similar, where those at the top are quite different- we draw conclusions based on the location on the vertical rather than horizontal axis.

1. Begin with n observations and a measure of all the $\frac{(n-1)}{2}$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For $i = n, n-1, \dots, 2$
 - (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates height in dendrogram where fusion should be placed.
 - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using distance metric).

Linkage: Complete (max dissimilarity), Single (min), Average, Centroid (between centroids of cluster A and B)

Machine Learning Part I

Comparing ML Algorithms

Power and Expressibility: ML methods differ in terms of complexity. Linear regression fits linear functions while NN define piecewise-linear separation boundaries. More complex models can provide more accurate models, but at the risk of overfitting.

Interpretability: some models are more transparent and understandable than others (white box vs. black box models)

Ease of Use: some models feature few parameters/decisions (linear regression/NN), while others require more decision making to optimize (SVMs)

Training Speed: models differ in how fast they fit the necessary parameters

Prediction Speed: models differ in how fast they make predictions given a query

Method	Power of Expression	Ease of Interpretation	Ease of Use	Training Speed	Prediction Speed
Linear Regression	5	9	9	9	9
Nearest Neighbor	5	9	8	10	2
Naive Bayes	4	8	7	9	8
Decision Trees	8	8	7	7	9
Support Vector Machines	8	6	6	7	7
Boosting	9	6	6	6	6
Graphical Models	9	8	3	4	4
Deep Learning	10	3	4	3	7

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.

Problem: Suppose we need to classify vector $X = x_1 \dots x_n$ into m classes, $C_1 \dots C_m$. We need to compute the probability of each possible class given X , so we can assign X the label of the class with highest probability. We can calculate a probability using the Bayes' Theorem:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

Where:

1. $P(C_i)$: the prior probability of belonging to class i
2. $P(X)$: normalizing constant, or probability of seeing the given input vector over all possible input vectors
3. $P(X|C_i)$: the conditional probability of seeing input vector X given we know the class is C_i

The prediction model will formally look like:

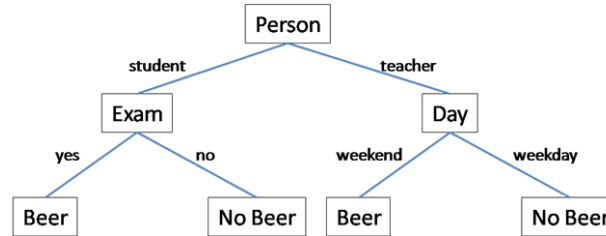
$$C(X) = \operatorname{argmax}_{i \in \text{classes}(t)} \frac{P(X|C_i)P(C_i)}{P(X)}$$

where $C(X)$ is the prediction returned for input X .

Machine Learning Part II

Decision Trees

Binary branching structure used to classify an arbitrary input vector X . Each node in the tree contains a simple feature comparison against some field ($x_i > 42?$). Result of each comparison is either true or false, which determines if we should proceed along to the left or right child of the given node. Also known as sometimes called classification and regression trees (CART).



Advantages: Non-linearity, support for categorical variables, easy to interpret, application to regression.

Disadvantages: Prone to overfitting, instable (not robust to noise), high variance, low bias

Note: rarely do models just use one decision tree. Instead, we aggregate many decision trees using methods like ensembling, bagging, and boosting.

Ensembles, Bagging, Random Forests, Boosting

Ensemble learning is the strategy of combining many different classifiers/models into one predictive model. It revolves around the idea of voting: a so-called "wisdom of crowds" approach. The most predicted class will be the final prediction.

Bagging: ensemble method that works by taking B bootstrapped subsamples of the training data and constructing B trees, each tree training on a distinct subsample as

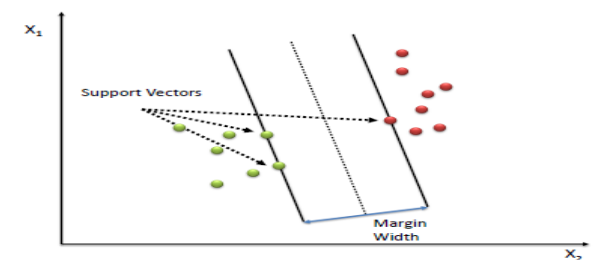
Random Forests: builds on bagging by decorrelating the trees. We do everything the same like in bagging, but when we build the trees, everytime we consider a split, a random sample of the p predictors is chosen as split candidates, not the full set (typically $m \approx \sqrt{p}$). When $m = p$, then we are just doing bagging.

Boosting: the main idea is to improve our model where it is not performing well by using information from previously constructed classifiers. Slow learner. Has 3 tuning parameters: number of classifiers B , learning parameter λ , interaction depth d (controls interaction order of model).

Machine Learning Part III

Support Vector Machines

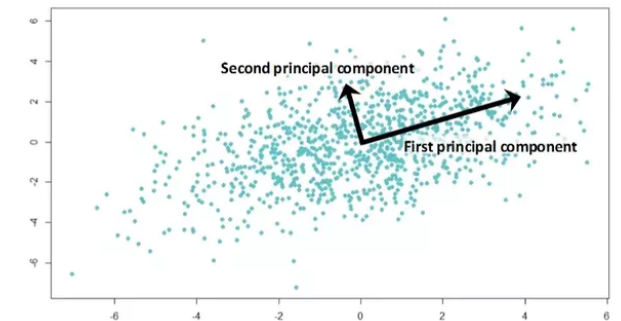
Work by constructing a hyperplane that separates points between two classes. The hyperplane is determined using the maximal margin hyperplane, which is the hyperplane that is the maximum distance from the training observations. This distance is called the margin. Points that fall on one side of the hyperplane are classified as -1 and the other $+1$.



Principal Component Analysis (PCA)

Principal components allow us to summarize a set of correlated variables with a smaller set of variables that collectively explain most of the variability in the original set. Essentially, we are "dropping" the least important feature variables.

Principal Component Analysis is the process by which principal components are calculated and the use of them to analyzing and understanding the data. PCA is an unsupervised approach and is used for dimensionality reduction, feature extraction, and data visualization. Variables after performing PCA are independent. Scaling variables is also important while performing PCA.



Machine Learning Part IV

ML Terminology and Concepts

Features: input data/variables used by the ML model

Feature Engineering: transforming input features to be more useful for the models. e.g. mapping categories to buckets, normalizing between -1 and 1, removing null

Train/Eval/Test: training is data used to optimize the model, evaluation is used to assess the model on new data during training, test is used to provide the final result

Classification/Regression: regression is prediction a number (e.g. housing price), classification is prediction from a set of categories (e.g. predicting red/blue/green)

Linear Regression: predicts an output by multiplying and summing input features with weights and biases

Logistic Regression: similar to linear regression but predicts a probability

Overfitting: model performs great on the input data but poorly on the test data (combat by dropout, early stopping, or reduce # of nodes or layers)

Bias/Variance: how much output is determined by the features. more variance often can mean overfitting, more bias can mean a bad model

Regularization: variety of approaches to reduce overfitting, including adding the weights to the loss function, randomly dropping layers (dropout)

Ensemble Learning: training multiple models with different parameters to solve the same problem

A/B testing: statistical way of comparing 2+ techniques to determine which technique performs better and also if difference is statistically significant

Baseline Model: simple model/heuristic used as reference point for comparing how well a model is performing

Bias: prejudice or favoritism towards some things, people, or groups over others that can affect collection/sampling and interpretation of data, the design of a system, and how users interact with a system

Dynamic Model: model that is trained online in a continuously updating fashion

Static Model: model that is trained offline

Normalization: process of converting an actual range of values into a standard range of values, typically -1 to +1

Independently and Identically Distributed (i.i.d): data drawn from a distribution that doesn't change, and where each value drawn doesn't depend on previously drawn values; ideal but rarely found in real life

Hyperparameters: the "knobs" that you tweak during successive runs of training a model

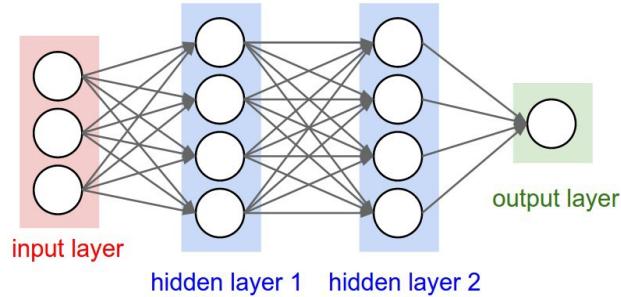
Generalization: refers to a model's ability to make correct predictions on new, previously unseen data as opposed to the data used to train the model

Cross-Entropy: quantifies the difference between two probability distributions

Deep Learning Part I

What is Deep Learning?

Deep learning is a subset of machine learning. One popular DL technique is based on Neural Networks (NN), which loosely mimic the human brain and the code structures are arranged in layers. Each layer's input is the previous layer's output, which yields progressively higher-level features and defines a hierarchy. A Deep Neural Network is just a NN that has more than 1 hidden layer.



Recall that statistical learning is all about approximating $f(X)$. Neural networks are known as **universal approximators**, meaning no matter how complex a function is, there exists a NN that can (approximately) do the job. We can increase the approximation (or complexity) by adding more hidden layers and neurons.

Popular Architectures

There are different kinds of NNs that are suitable for certain problems, which depend on the NN's architecture.

Linear Classifier: takes input features and combines them with weights and biases to predict output value

DNN: deep neural net, contains intermediate layers of nodes that represent "hidden features" and activation functions to represent non-linearity

CNN: convolutional NN, has a combination of convolutional, pooling, dense layers. popular for image classification.

Transfer Learning: use existing trained models as starting points and add additional layers for the specific use case. idea is that highly trained existing models know general features that serve as a good starting point for training a small network on specific examples

RNN: recurrent NN, designed for handling a sequence of inputs that have "memory" of the sequence. LSTMs are a fancy version of RNNs, popular for NLP

GAN: general adversarial NN, one model creates fake examples, and another model is served both fake example and real examples and is asked to distinguish

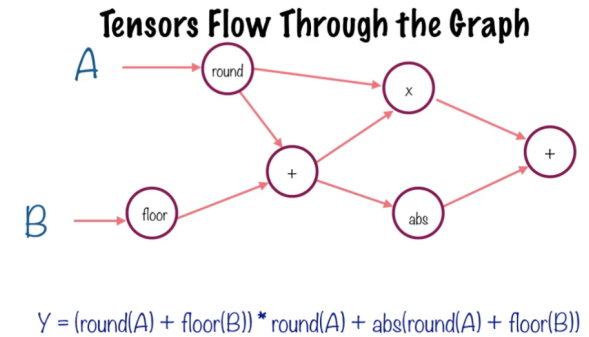
Wide and Deep: combines linear classifiers with deep neural net classifiers, "wide" linear parts represent memorizing specific examples and "deep" parts represent understanding high level features

Deep Learning Part II

Tensorflow

Tensorflow is an open source software library for numerical computation using data flow graphs. Everything in TF is a graph, where nodes represent operations on data and edges represent the data. Phase 1 of TF is building up a computation graph and phase 2 is executing it. It is also distributed, meaning it can run on either a cluster of machines or just a single machine.

TF is extremely popular/suitable for working with Neural Networks, since the way TF sets up the computational graph pretty much resembles a NN.



Tensors

In a graph, tensors are the edges and are multidimensional data arrays that flow through the graph. Central unit of data in TF and consists of a set of primitive values shaped into an array of any number of dimensions.

A tensor is characterized by its rank (# dimensions in tensor), shape (# of dimensions and size of each dimension), data type (data type of each element in tensor).

Placeholders and Variables

Variables: best way to represent shared, persistent state manipulated by your program. These are the parameters of the ML model are altered/trained during the training process. Training variables.

Placeholders: way to specify inputs into a graph that hold the place for a Tensor that will be fed at runtime. They are assigned once, do not change after. Input nodes

Deep Learning Part III

Deep Learning Terminology and Concepts

Neuron: node in a NN, typically taking in multiple input values and generating one output value, calculates the output value by applying an activation function (non-linear transformation) to a weighted sum of input values

Weights: edges in a NN, the goal of training is to determine the optimal weight for each feature; if weight = 0, corresponding feature does not contribute

Neural Network: composed of neurons (simple building blocks that actually “learn”), contains activation functions that makes it possible to predict non-linear outputs

Activation Functions: mathematical functions that introduce non-linearity to a network e.g. RELU, tanh

Sigmoid Function: function that maps very negative numbers to a number very close to 0, huge numbers close to 1, and 0 to .5. Useful for predicting probabilities

Gradient Descent/Backpropagation: fundamental loss optimizer algorithms, of which the other optimizers are usually based. Backpropagation is similar to gradient descent but for neural nets

Optimizer: operation that changes the weights and biases to reduce loss e.g. Adagrad or Adam

Weights / Biases: weights are values that the input features are multiplied by to predict an output value. Biases are the value of the output given a weight of 0.

Converge: algorithm that converges will eventually reach an optimal answer, even if very slowly. An algorithm that doesn't converge may never reach an optimal answer.

Learning Rate: rate at which optimizers change weights and biases. High learning rate generally trains faster but risks not converging, whereas a lower rate trains slower

Numerical Instability: issues with very large/small values due to limits of floating point numbers in computers

Embeddings: mapping from discrete objects, such as words, to vectors of real numbers. useful because classifiers/neural networks work well on vectors of real numbers

Convolutional Layer: series of convolutional operations, each acting on a different slice of the input matrix

Dropout: method for regularization in training NNs, works by removing a random selection of some units in a network layer for a single gradient step

Early Stopping: method for regularization that involves ending model training early

Gradient Descent: technique to minimize loss by computing the gradients of loss with respect to the model's parameters, conditioned on training data

Pooling: Reducing a matrix (or matrices) created by an earlier convolutional layer to a smaller matrix. Pooling usually involves taking either the maximum or average value across the pooled area

Big Data- Hadoop Overview

Data can no longer fit in memory on one machine (monolithic), so a new way of computing was devised using a group of computers to process this “big data” (distributed). Such a group is called a cluster, which makes up server farms. All of these servers have to be coordinated in the following ways: partition data, coordinate computing tasks, handle fault tolerance/recovery, and allocate capacity to process.

Hadoop

Hadoop is an open source *distributed* processing framework that manages data processing and storage for big data applications running in clustered systems. It is comprised of 3 main components:

- **Hadoop Distributed File System (HDFS):** a distributed file system that provides high-throughput access to application data by partitioning data across many machines
- **YARN:** framework for job scheduling and cluster resource management (task coordination)
- **MapReduce:** YARN-based system for parallel processing of large data sets on multiple machines

HDFS

Each disk on a different machine in a cluster is comprised of 1 master node and the rest are workers/data nodes. The **master node** manages the overall file system by storing the directory structure and the metadata of the files. The **data nodes** physically store the data. Large files are broken up and distributed across multiple machines, which are also replicated across multiple machines to provide fault tolerance.

MapReduce

Parallel programming paradigm which allows for processing of huge amounts of data by running processes on multiple machines. Defining a MapReduce job requires two stages: map and reduce.

- **Map:** operation to be performed in parallel on small portions of the dataset. the output is a key-value pair $\langle K, V \rangle$
- **Reduce:** operation to combine the results of Map

YARN- Yet Another Resource Negotiator

Coordinates tasks running on the cluster and assigns new nodes in case of failure. Comprised of 2 subcomponents: the resource manager and the node manager. The **resource manager** runs on a single master node and schedules tasks across nodes. The **node manager** runs on all other nodes and manages tasks on the individual node.

Big Data- Hadoop Ecosystem

An entire ecosystem of tools have emerged around Hadoop, which are based on interacting with HDFS. Below are some popular ones:

Hive: data warehouse software built on top of Hadoop that facilitates reading, writing, and managing large datasets residing in distributed storage using SQL-like queries (HiveQL). Hive abstracts away underlying MapReduce jobs and returns HDFS in the form of tables (not HDFS).

Pig: high level scripting language (Pig Latin) that enables writing complex data transformations. It pulls unstructured/incomplete data from sources, cleans it, and places it in a database/data warehouses. Pig performs ETL into data warehouse while Hive queries from data warehouse to perform analysis (GCP: DataFlow).

Spark: framework for writing fast, distributed programs for data processing and analysis. Spark solves similar problems as Hadoop MapReduce but with a fast in-memory approach. It is an unified engine that supports SQL queries, streaming data, machine learning and graph processing. Can operate separately from Hadoop but integrates well with Hadoop. Data is processed using Resilient Distributed Datasets (RDDs), which are immutable, lazily evaluated, and tracks lineage.

Hbase: non-relational, NoSQL, column-oriented database management system that runs on top of HDFS. Well suited for sparse data sets (GCP: BigTable)

Flink/Kafka: stream processing framework. Batch streaming is for bounded, finite datasets, with periodic updates, and delayed processing. Stream processing is for unbounded datasets, with continuous updates, and immediate processing. Stream data and stream processing must be decoupled via a message queue. Can group streaming data (windows) using tumbling (non-overlapping time), sliding (overlapping time), or session (session gap) windows.

Beam: programming model to define and execute data processing pipelines, including ETL, batch and stream (continuous) processing. After building the pipeline, it is executed by one of Beam's distributed processing back-ends (Apache Apex, Apache Flink, Apache Spark, and Google Cloud Dataflow). Modeled as a Directed Acyclic Graph (DAG).

Oozie: workflow scheduler system to manage Hadoop jobs

Sqoop: transferring framework to transfer large amounts of data into HDFS from relational databases (MySQL)

Graph Theory

Graph theory is the study of graphs, which are structures used to model relationships between objects. For example, we can model friendships, computers, social networks, and transportation systems all as graphs. These graphs can then be analyzed to uncover hidden patterns or connections that were previously unidentifiable or calculate statistical properties of the networks and predict how the networks will evolve over time.

Formally, a graph $G = (V, E)$ consists of a set V of vertices (nodes) and a set E of edges (connects two nodes together). An edge represents a relationship between the nodes it connects (friendship between 2 people, connection between 2 computers, etc.). A directed graph is where the edges have a direction or order (otherwise undirected).

A weighted graph is a graph where the edges show the intensity of the relationships using weights:

- **Binary Weight:** 0 or 1 weight, tells us if a link exists between 2 nodes
- **Numeric Weight:** expresses how strong the connection is between a node and other nodes
- **Normalized Weight:** variant of numeric weight where all the outgoing edges of a node sum to 1

A graph can be represented:

- **Graphically:** a picture that displays all the nodes, edges, and weights
- **Mathematically:** an adjacency matrix A of size (n, n) (n nodes) and $a_{i,j} = 1$ if a link exists between nodes i and j , 0 otherwise. A weight matrix W expresses the edge weights between nodes of a network. An adjacency list is an abstract representation of the adjacency matrix, and provides a list of all the connections present in the network (weight list is similar).

Applications of graph theory include:

- **Route Optimization:** model the transportation of a commodity from one place to another
- **Job Scheduling:** model and find the optimal scheduling of jobs or tasks
- **Fraud Detection:** model fraud transactions and uncover rings of fraudsters working together
- **Sociology and Economics:** model groups of people to see how they will act and evolve over time
- **Epidemiology:** model how a disease will spread through a network and how fast it will spread

Python's NetworkX and Spark's GraphX offer graph capabilities.

SQL Part I

Structured Query Language (SQL) is a declarative language used to access & manipulate data in databases. Usually the database is a Relational Database Management System (RDBMS), which stores data arranged in relational database tables. A table is arranged in columns and rows, where columns represent characteristics of stored data and rows represent actual data entries.

Basic Queries

- filter columns: **SELECT** col1, col3... **FROM** table1
- filter the rows: **WHERE** col4 = 1 **AND** col5 = 2
- aggregate the data: **GROUP BY**...
- limit aggregated data: **HAVING** count(*) > 1
- order of the results: **ORDER BY** col2

Useful Keywords for **SELECT**

DISTINCT- return unique results

BETWEEN a **AND** b- limit the range, the values can be numbers, text, or dates

LIKE- pattern search within the column text

IN (a, b, c) - check if the value is contained among given

Data Modification

- update specific data with the **WHERE** clause:

UPDATE table1 **SET** col1 = 1 **WHERE** col2 = 2

- insert values manually

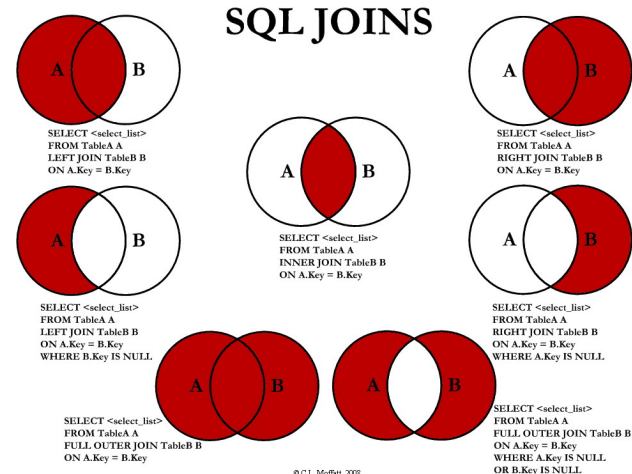
INSERT INTO table1 (col1,col3) **VALUES** (val1,val3);

- by using the results of a query

INSERT INTO table1 (col1,col3) **SELECT** col,col2 **FROM** table2;

Joins

The JOIN clause is used to combine rows from two or more tables, based on a related column between them.



Python- Data Structures

Data structures are a way of storing and manipulating data and each data structure has its own strengths and weaknesses. Combined with algorithms, data structures allow us to efficiently solve problems. It is important to know the main types of data structures that you will need to efficiently solve problems.

Lists: or arrays, ordered sequences of objects, mutable

```
>>> l = [42, 3.14, "hello", "world"]
```

Tuples: like lists, but immutable

```
>>> t = (42, 3.14, "hello", "world")
```

Dictionaries: hash tables, key-value pairs, unsorted

```
>>> d = {"life": 42, "pi": 3.14}
```

Sets: mutable, unordered sequence of unique elements. frozensets are just immutable sets

```
>>> s = set([42, 3.14, "hello", "world"])
```

Collections Module

deque: double-ended queue, generalization of stacks and queues; supports append, appendLeft, pop, rotate, etc

```
>>> s = deque([42, 3.14, "hello", "world"])
```

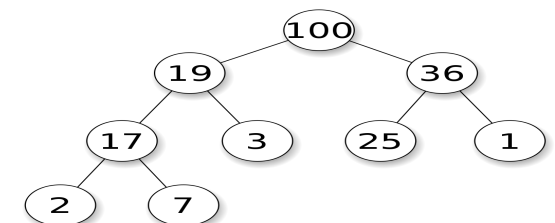
Counter: dict subclass, unordered collection where elements are stored as keys and counts stored as values

```
>>> c = Counter('apple')
>>> print(c)
Counter({'p': 2, 'a': 1, 'l': 1, 'e': 1})
```

heapq Module

Heap Queue: priority queue, heaps are binary trees for which every parent node has a value greater than or equal to any of its children (max-heap), order is important; supports push, pop, pushpop, heapify, replace functionality

```
>>> heap = []
>>> for n in data:
...     heappush(heap, n)
>>> heap
[0, 1, 3, 6, 2, 8, 4, 7, 9, 5]
```



Recommended Resources

- Data Science Design Manual
(www.springer.com/us/book/9783319554433)
- Introduction to Statistical Learning
(www-bcf.usc.edu/~gareth/ISL/)
- Probability Cheatsheet
(www.wzchen.com/probability-cheatsheet/)
- Google's Machine Learning Crash Course
(developers.google.com/machine-learning/crash-course/)