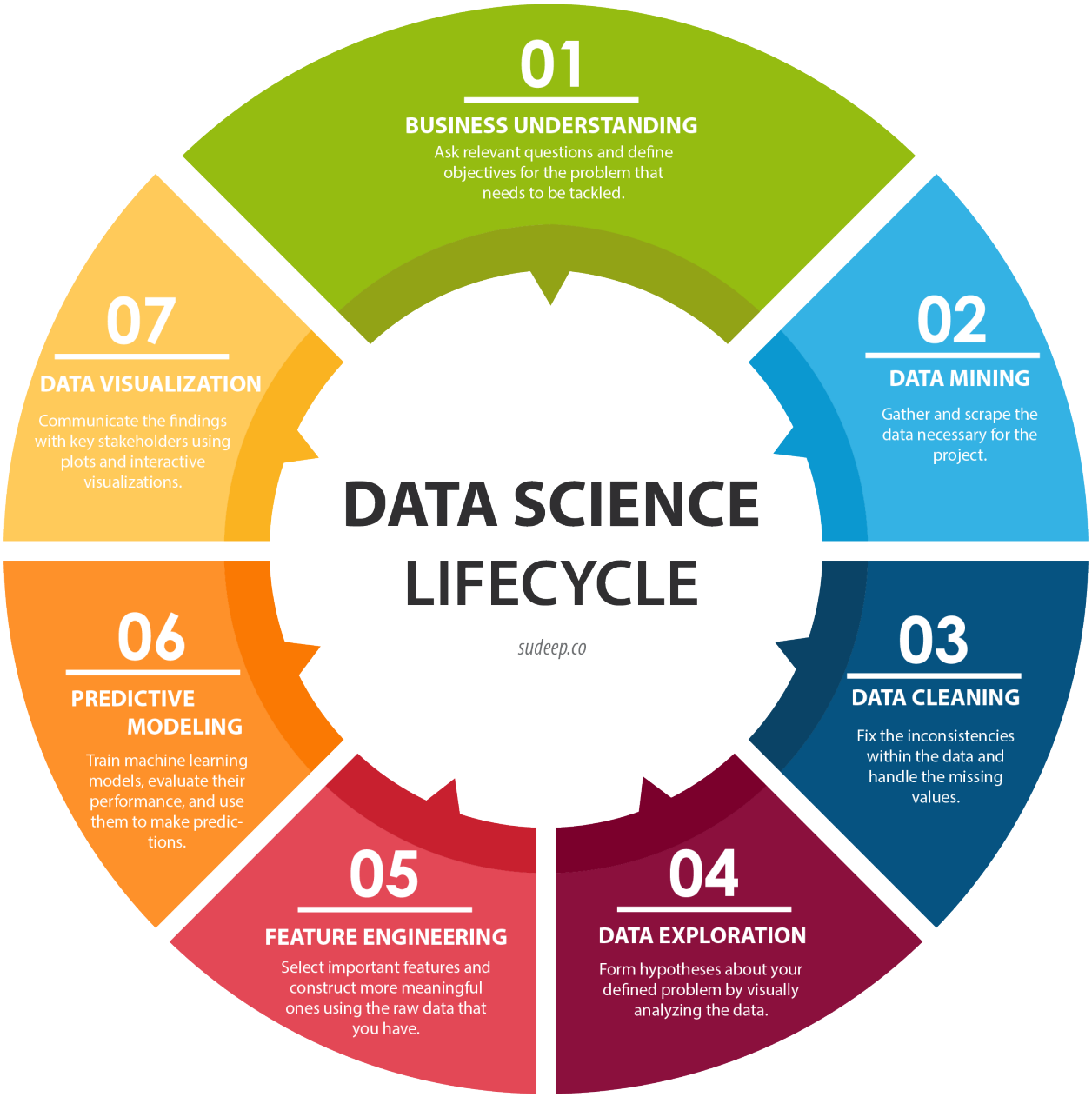

01
BUSINESS UNDERSTANDING
Ask relevant questions and define objectives for the problem that needs to be tackled.
02
DATA MINING
Gather and scrape the data necessary for the project.
03
DATA CLEANING
Fix the inconsistencies within the data and handle the missing values.
04
DATA EXPLORATION
Form hypotheses about your defined problem by visually analyzing the data.
05
FEATURE ENGINEERING
Select important features and construct more meaningful ones using the raw data that you have.
06
PREDICTIVE MODELING
Train machine learning models, evaluate their performance, and use them to make predictions.
07
DATA VISUALIZATION
Communicate the findings with key stakeholders using plots and interactive visualizations.
DATA SCIENCE LIFECYCLE
sudeep.co

Data Science Cheatsheet

Compiled by Maverick Lin (http://mavericklin.com)

Last Updated August 13, 2018

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/Fomats

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, $0 \leq \text{P(s)} \leq 1$
2. $\sum \text{p(s)} = 1$

Probability of Event E: sum of the probabilities of the outcomes of the experiment: $\text{p(E)} = \sum_{s \subset E} \text{p(s)}$
Random Variable V: numerical function on the outcomes of a probability space
Expected Value of Random Variable V: $\text{E(V)} = \sum_{s \subset S} \text{p(s)} * \text{V(s)}$

Independence, Conditional, Compound

Independent Events: A and B are independent iff:

$$\text{P(A} \cap \text{B)} = \text{P(A)P(B)}$$
$$\text{P}(A|B) = \text{P(A)}$$
$$\text{P}(B|A) = \text{P(B)}$$

Conditional Probability: $\text{P}(A|B) = \text{P(A,B)/P(B)}$
Bayes Theorem: $\text{P}(A|B) = \text{P}(B|A)\text{P(A)/P(B)}$
Joint Probability: $\text{P(A,B)} = \text{P}(B|A)\text{P(A)}$
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 $= \sqrt[n]{a_1 a_2 ... a_3}$
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

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

Variance $\text{V} = \sigma^2$

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

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

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, unidecode, 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

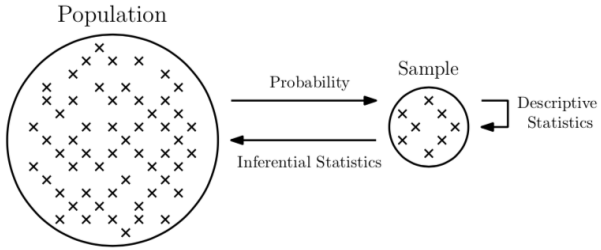
Raw Measures: data that hasn't been transformed yet
Rounding: sometimes precision is noise; round to nearest integer, decimal etc..
Scaling: log, z-score, minmax scale
Imputation: fill in missing values using mean, median, model output, etc..
Binning: transforming numeric features into categorical ones (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.

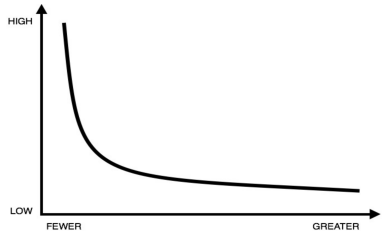
Classic Statistical Distributions

Binomial Distribution (Discrete)
Assume X is distributed 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: $P(X = x) = \binom{n}{k} p^x (1-p)^{n-x}$
EV: $\mu = np$ Variance = npq

Normal/Gaussian Distribution (Continuous)
Assume X in 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 $\to \infty$.
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 Pois(λ). 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 = $(X_1, X_2, ...X_p)$ 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 = $(X_1, X_2, ...X_p)$

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.

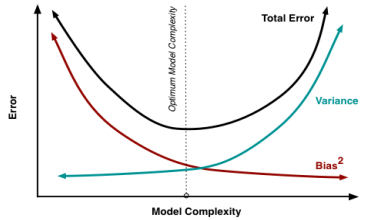
Modeling- Philosophies

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 $\to$ 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 $\to$ 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. Ssome 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 $x_i = (x_1, x_2, ...x_n)$ to a known output variables $y_i = (y_1, y_2, ...y_n)$
- **Unsupervised**: models that take in input variables $x_i = (x_1, x_2, ...x_n)$, 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: $\mathrm{F} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$
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: $\Delta = y\prime - y$
Squared Error: $\Delta^2 = (y\prime - y)^2$
Mean-Squared Error: $MSE = \frac{1}{n}\sum_{i=1}^{n}(y\prime_i - y_i)^2$
Root Mean-Squared Error: $\mathrm{RMSD} = \sqrt{MSE}$
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 erro 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 it 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 = $(X_1, X_2, ...X_p)$ and output variable Y takes the form:

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

$\beta_0 ... \beta_p$ 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 differences between the actual ith value and the predicted ith value. RSS = $\sum_{i=1}^{n} e_i$, 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 solutions 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 shrunken 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(Yes) > 0.5), then the model will predict Yes.

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

How to find best coefficients?

Maximum Likelihood: The coefficients $\beta_0 ... \beta_p$ are unknown and must be estimated from the training data. We seek estimates for $\beta_0 ... \beta_p$ 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 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

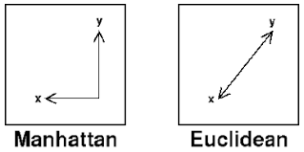
Interpreting examples as points in space provides a way to find natural groupings or clusters among data e.g. which stars are the closest to our sun? Networks can also be built from point sets (vertices) by connecting related points.

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

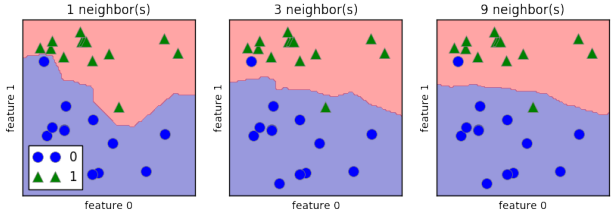
Kullback-Leibler Divergence: $KL(A||B) = \sum_{i=i}^{d} a_i log_2 \frac{a_i}{b_i}$
KL divergence measures the distances between probability distributions by measuring the uncertainty gained or uncertainty lost when replacing distribution A with distribution B. However, this is not a metric but forms 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)!= 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 kth cluster centroid is the vector of the p feature means for the observations in the kth 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 $\frac{(n)n-1}{2}$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For i = n, n-1, ...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 = $x_1...x_n$ into m classes, $C_1...C_m$. 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:

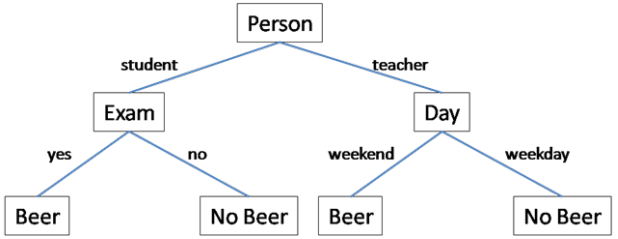
$$C(X) = argmax_{i \in classes(t)} \frac{P(X|C_i)P(C_i)}{P(X)}$$

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

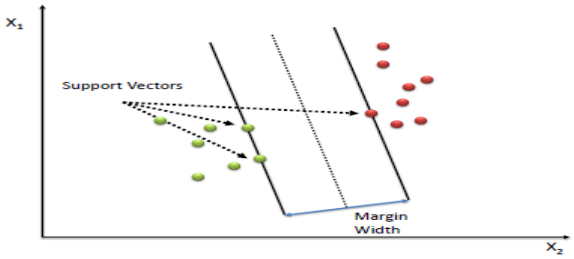
Random Forests: builds on bagging by decorrelating the trees. We do everything the same like in bagging, but when we build the trees, everytime we consider a split, a random sample of the p predictors is chosen as split candidates, not the full set (typically m $\approx \sqrt{p}$). When m = p, then we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

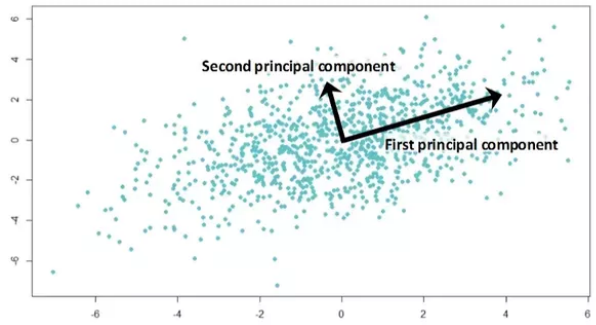
Work by constructing a hyperplane that separates points between two classes. The hyperplane is determined using the maximal margin hyperplane, which is the hyperplane that is the maximum distance from the training observations. This distance is called the margin. Points that fall on one side of the hyperplane 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

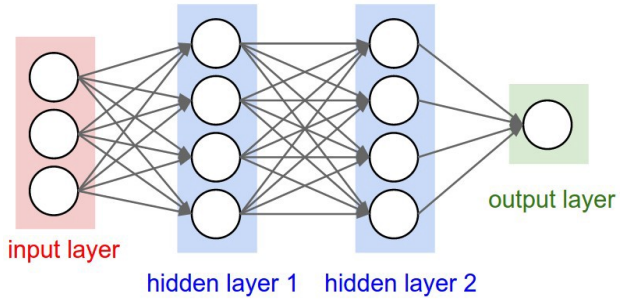
ML Terminology and Concepts

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 asses 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

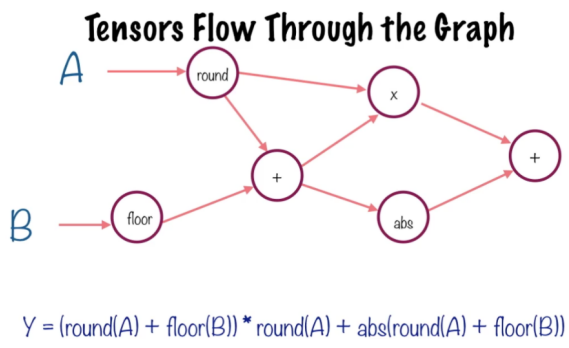
There are different kinds of NNs that are suitable for certain problems, which depend on the NN's architecture.

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 (nonlinear 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 $< K, V >$
- **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 o 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)

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

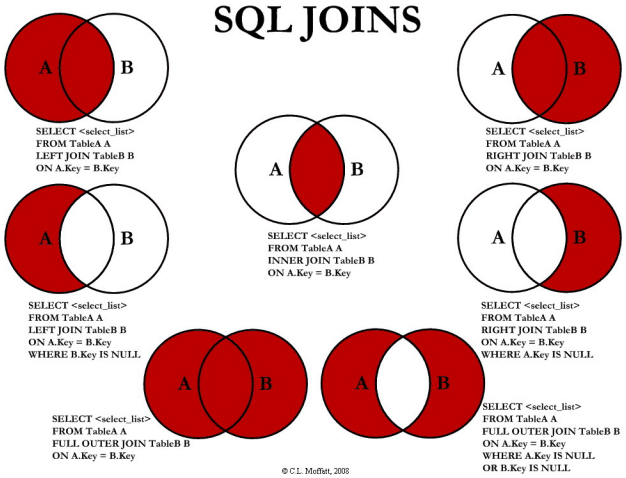
Useful Keywords for **SELECT**
DISTINCT- return unique results
BETWEEN a AND b- limit the range, the values can be numbers, text, or dates
LIKE- pattern search within the column text
IN (a, b, c) - check if the value is contained among given

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"])
```

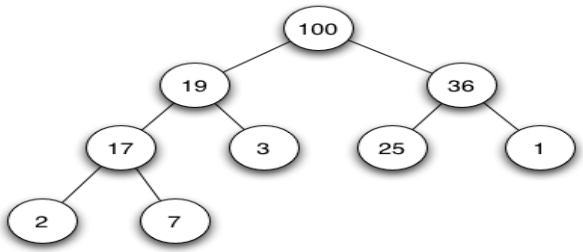
Counter: dict subclass, unordered collection where elements are stored as keys and counts stored as values

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

heqpq 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/`)

Business Science Problem Framework

View Business as a Machine

Step 1:
Isolate business unit

Step 2:
Define objectives.
Define machine in terms of people and processes

Step 3:
Collect outcomes in terms of feedback.
Feedback identifies problems.

Understand the Drivers

Step 1:
Investigate if objectives are being met

Step 2:
Synthesize outcomes

Step 3:
Hypothesize drivers

Measure the Drivers

Step 1:
Collect Data

Step 2:
Develop KPIs

Uncover Problems & Opportunities

Step 1:
Evaluate performance vs KPIs

Step 2:
Highlight potential problem areas

Step 3:
Review process and consider what could be missed or needed to answer questions

Encode Algorithms

Step 1:
Develop algorithms to predict and explain problem

Step 2:
Tie financial value of individual decisions to optimize for profit

Step 3:
Use recommendation algorithms to improve decisions

Measure Results

Step 1:
Capture outcomes after decision making system is implemented

Step 2:
Synthesize results in terms of good and bad outcomes identifying what was done and what happened

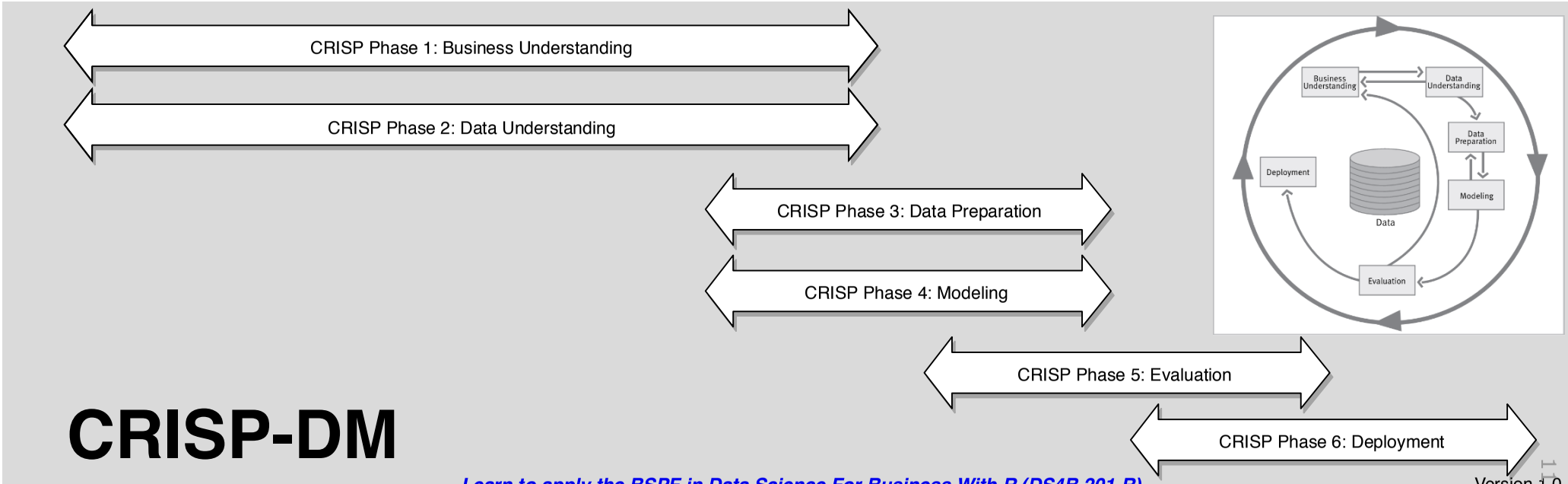
Step 3:
Visualize outcomes over time to determine progress

Report Financial Impact

Step 1:
Measure actual results.

Step 2:
Tie to financial benefits

Step 3:
Report financial benefit of algorithms to key stakeholders



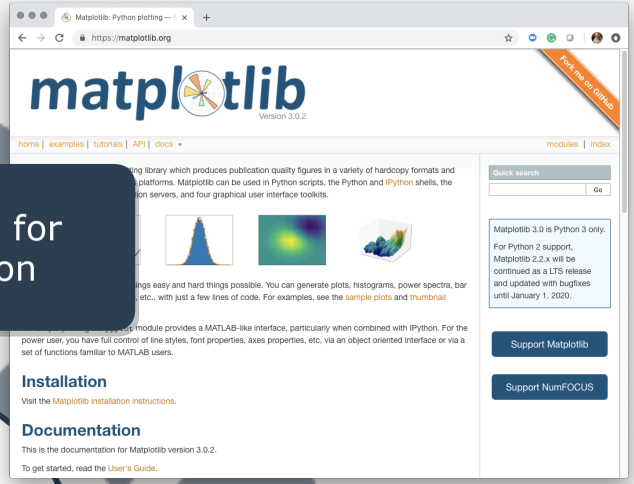
CRISP-DM

Learn to apply the BSPF in Data Science For Business With R (DS4B 201-R)

Version 1.0

Data Science with Python Workflow

If you want to learn Python and this workflow ***for business analysis***, take the **Python For Business Analysis (DS4B 101-P) course through Business Science University.**



Click the links for Documentation

CS = Cheat Sheet

Pandas (CS)

Import → **Tidy** → **Transform** ⇄ **Visualize** / **Model** → **Communicate**

Data
I/O tools

Web
Beautiful Soup
Scrappy

Pandas
data structures
group by
joins
reshape (pivot)

Jupyter & Pycharm

Pandas
text
time series
categorical
missing

matplotlib
seaborn

Scikit-Learn
Featuretools

JupyterLab
Jupyter Notebook
Django
Flask

Important Resources

- **Anaconda Distribution:** https://www.anaconda.com/download/
- **Python Documentation:** https://docs.python.org/
- **Python Standard Library:** https://docs.python.org/3/library

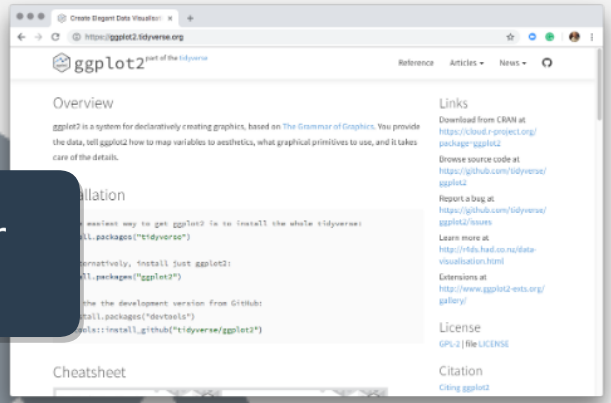
"Data Science Education for the Enterprise"

Business Science University
university.business-science.io

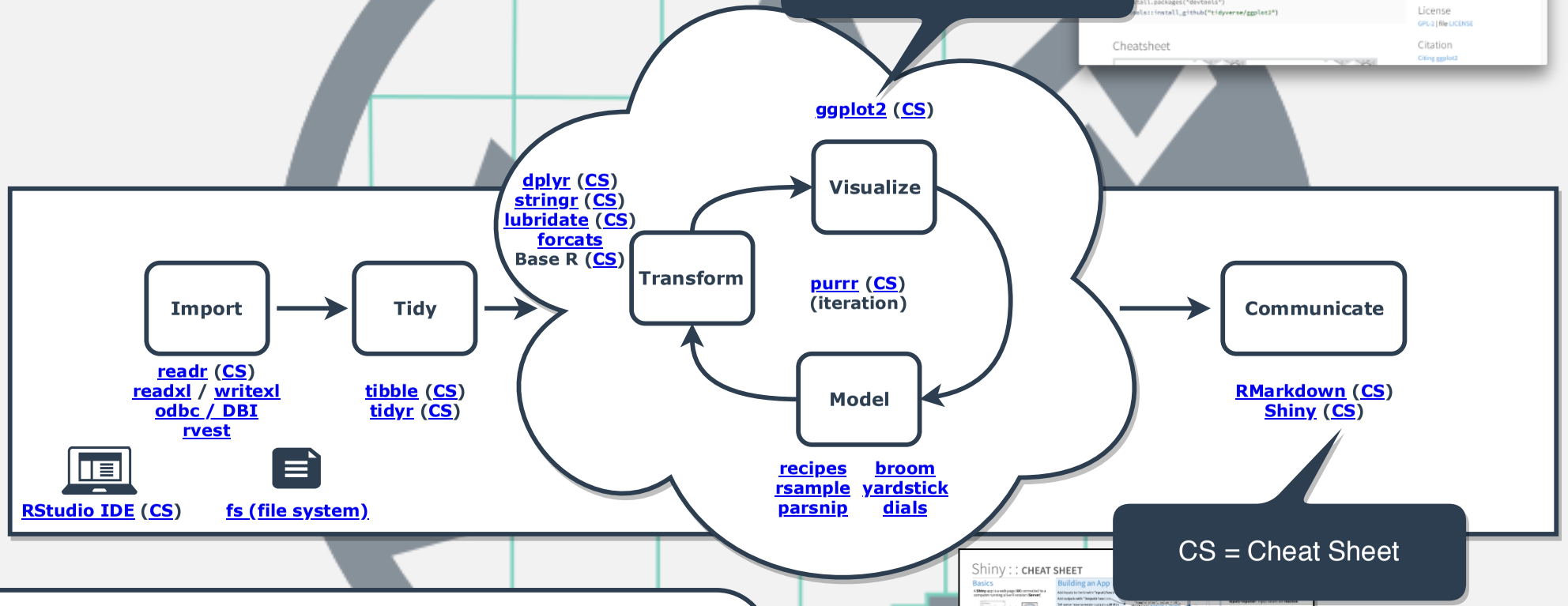
version: 0.0

Data Science with R Workflow

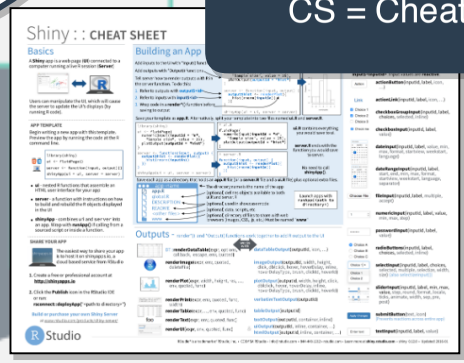
The Data Science With R Workflow is available in the book: R For Data Science. If you want to learn R and this workflow ***for business analysis***, take the **R For Business Analysis (DS4B 101-R) course through Business Science University.**



Click the links for Documentation



CS = Cheat Sheet



Important Resources

- **R For Data Science Book:** http://r4ds.had.co.nz/
- **Rmarkdown Book:** https://bookdown.org/yihui/rmarkdown/
- **Data Visualization Book**: https://rkabacoff.github.io/datavis/
- **More Cheatsheets:** https://www.rstudio.com/resources/cheatsheets/
- **tidyverse packages:** https://www.tidyverse.org/
- **Connecting to databases:** https://db.rstudio.com/
- **RMarkdown website:** https://rmarkdown.rstudio.com/
- **Shiny web applications website:** http://shiny.rstudio.com/
- **Jenny Bryan's purrr tutorial:** https://jennybryan.org/

"Data Science Education for the Enterprise"

Business Science University
university.business-science.io

version: 1.1

Data Science with R

Special Topics

Text Analysis & NLP

- Text Mining with R (Book): tidytext
- NLP:
  - H2O word2vec: Word embeddings
  - text2vec: fast vectorization, topic modeling
  - udpipe: UDPipe C++ lib in R

Machine Learning

- Multi-Threaded/Scalable/Production ML:
  - H2O (CS)
  - Extreme Gradient Boosting: xgboost
  - R + Spark: sparklyr (CS)
  - Sparkling Water (Spark + H2O): rsparkling
- ML (Tidy): parsnip
- ML: caret (CS)

Time Series Analysis

- Time-aware tibbles: tibbletime & tsibble
- Convert between classes: timetk & tsbox
- Time Series Index Summary: timetk
- Generating Future Series: timetk

Forecasting

- ARIMA, ETS, etc: forecast & fable
- Tidy, glance, augment for forecast models: sweep
- Converting forecast prediction to tibble: sweep

Anomaly Detection

- Identify anomalies: anomalize

Network Analysis

- Network Data Transformations (Tidy): tidygraph
- Network Data Transformations: igraph

Network Viz

- Static:
  - ggraph - Graph plotting utilities for ggplot2
- Interactive (JavaScript):
  - networkD3 - D3 Networks in R
  - plotly - plotly.js (network graphs) in R

Deep Learning

- R Interface to TensorFlow Homepage:
  - Keras (CS)
  - TF Estimators
  - TensorFlow (Core)

Speed & Scale

- Fastest Single-Node Speed: data.table (CS)
- Distributed Cluster (Spark): sparklyr (CS)

Financial Analysis

- Getting financial data: tidyquant & quantmod
- Quantitative Analysis: tidyquant & xts/TTR
- Portfolio Analysis: tidyquant & PerformanceAnalytics

Financial & Time Viz

- Static:
  - tidyquant - Financial ggplot2 geoms
- Interactive:
  - highcharter - highchart.js in R
  - dygraphs - xts plotting
  - plotly - plotly.js (financial) in R

Geospatial Analysis

- Geocoding (getting lat/long, bboxes, & sf's):
  - ggmap - Google API (requires key)
  - osmdata - OpenStreet Overpass API
  - tmaptools - OpenStreet Nominatum API
- Simple Features (sf objects): sf (CS) (tidy)
- Spatial Objects (sp objects): sp (non-tidy)

Geospatial Viz

- Static:
  - ggmap - Google API (requires key)
  - osmplotr - Impressive Maps via OSM
  - tmap - Thematic Maps
  - cartography (CS) - Thematic Maps
- Interactive (JavaScript):
  - leaflet (CS) - leaflet.js in R
  - plotly - plotly.js (maps) in R

Interoperability

- Python: reticulate
- Java: rJava
- C++: Rcpp

Miscellaneous Tools

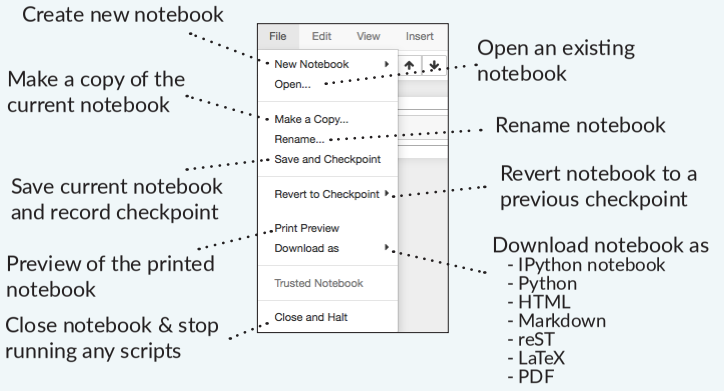
- **Interactive Plotting:** htmlwidgets for R
- **Building R Packages:** R packages Book
  - Pkg Development Tools: devtools (CS)
  - R Templates: usethis
  - Build Web Doc's: pkgdown
- **Advanced Concepts** (Advanced R Book)
  - rlang & Tidy Evaluation (CS)
- **Making Blogs & Books:**
  - Make a Website/Blog: blogdown
  - Write a Web Book: bookdown
- **Posting Code (GitHub, Stack Overflow):** reprex

Python For Data Science *Cheat Sheet*

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

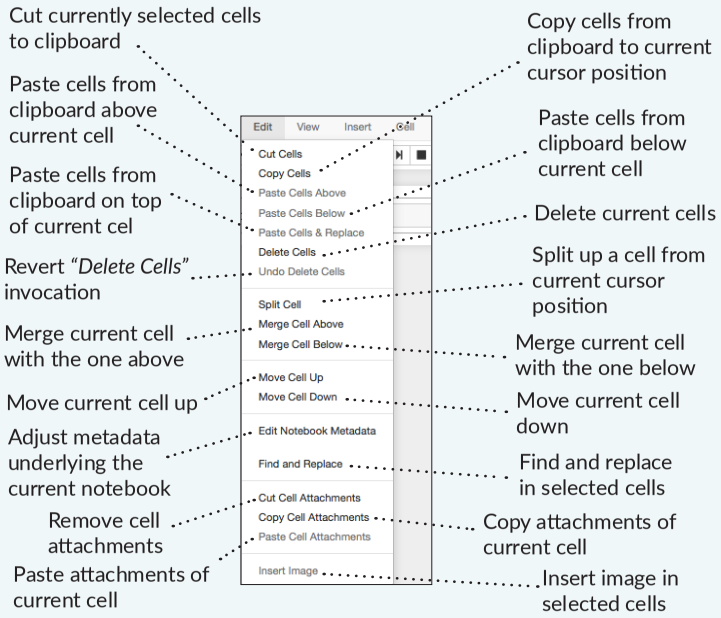
Saving/Loading Notebooks



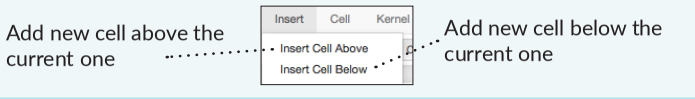
Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells



Insert Cells

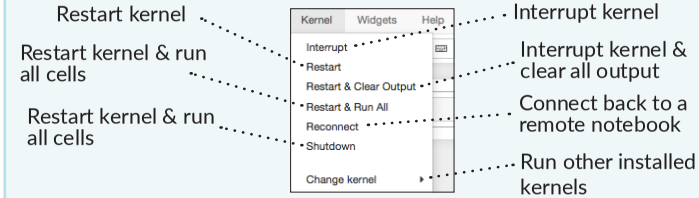


Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython | IRkernel | IJulia

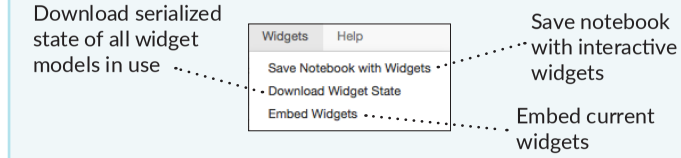
Installing Jupyter Notebook will automatically install the IPython kernel.



Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

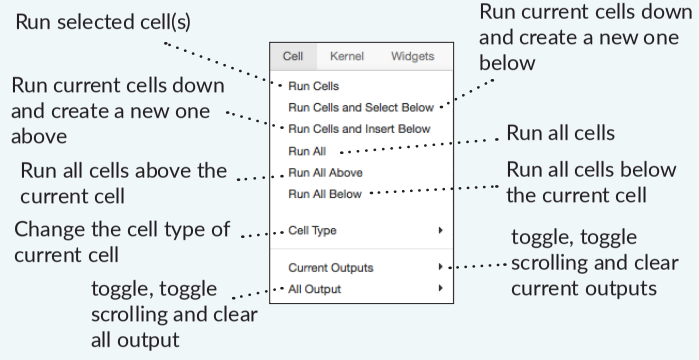


Command Mode:

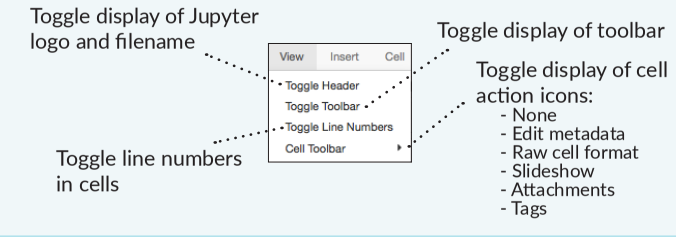
Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

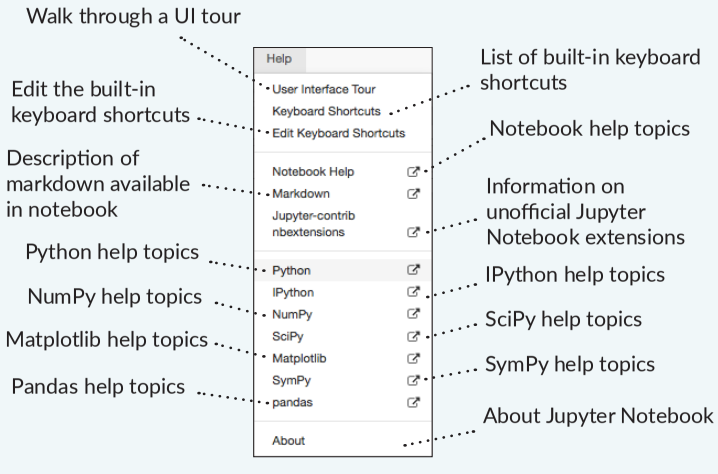
Executing Cells



View Cells



Asking For Help



Data Science Cheat Sheet

Python Basics

BASICS, PRINTING AND GETTING HELP

`x = 3` - Assign **3** to the variable **x**
`print(x)` - Print the value of **x**
`type(x)` - Return the type of the variable **x** (in this case, **int** for integer)
`help(x)` - Show documentation for the **str** data type
`help(print)` - Show documentation for the **print()** function

READING FILES

```
f = open("my_file.txt","r")
file_as_string = f.read()
```

- Open the file **my_file.txt** and assign its contents to **s**

```
import csv
f = open("my_dataset.csv","r")
csvreader = csv.reader(f)
csv_as_list = list(csvreader)
```

- Open the CSV file **my_dataset.csv** and assign its data to the list of lists **csv_as_list**

STRINGS

`s = "hello"` - Assign the string **"hello"** to the variable **s**

```
s = """She said,
"there's a good idea."
"""
```

- Assign a multi-line string to the variable **s**. Also used to create strings that contain both **"** and **'** characters

`len(s)` - Return the number of characters in **s**
`s.startswith("hel")` - Test whether **s** starts with the substring **"hel"**
`s.endswith("lo")` - Test whether **s** ends with the substring **"lo"**
`"{} plus {} is {}".format(3,1,4)` - Return the string with the values **3**, **1**, and **4** inserted
`s.replace("e","z")` - Return a new string based on **s** with all occurances of **"e"** replaced with **"z"**
`s.split(" ")` - Split the string **s** into a list of strings, separating on the character **" "** and return that list

NUMERIC TYPES AND MATHEMATICAL OPERATIONS

`i = int("5")` - Convert the string **"5"** to the integer **5** and assign the result to **i**
`f = float("2.5")` - Convert the string **"2.5"** to the float value **2.5** and assign the result to **f**
`5 + 5` - Addition
`5 - 5` - Subtraction
`10 / 2` - Division
`5 * 2` - Multiplication
`3 ** 2` - Raise **3** to the power of **2** (or 3^2)
`27 ** (1/3)` - The **3**rd root of **27** (or $\sqrt[3]{27}$)
`x += 1` - Assign the value of **x + 1** to **x**
`x -= 1` - Assign the value of **x - 1** to **x**

LISTS

`l = [100,21,88,3]` - Assign a list containing the integers **100**, **21**, **88**, and **3** to the variable **l**
`l = list()` - Create an empty list and assign the result to **l**
`l[0]` - Return the first value in the list **l**
`l[-1]` - Return the last value in the list **l**
`l[1:3]` - Return a slice (list) containing the second and third values of **l**
`len(l)` - Return the number of elements in **l**
`sum(l)` - Return the sum of the values of **l**
`min(l)` - Return the minimum value from **l**
`max(l)` - Return the maximum value from **l**
`l.append(16)` - Append the value **16** to the end of **l**
`l.sort()` - Sort the items in **l** in ascending order
`" ".join(["A","B","C","D"])` - Converts the list **["A", "B", "C", "D"]** into the string **"A B C D"**

DICTIONARIES

`d = {"CA":"Canada","GB":"Great Britain", "IN":"India"}` - Create a dictionary with keys of **"CA"**, **"GB"**, and **"IN"** and corresponding values of of **"Canada"**, **"Great Britain"**, and **"India"**
`d["GB"]` - Return the value from the dictionary **d** that has the key **"GB"**
`d.get("AU","Sorry")` - Return the value from the dictionary **d** that has the key **"AU"**, or the string **"Sorry"** if the key **"AU"** is not found in **d**
`d.keys()` - Return a list of the keys from **d**
`d.values()` - Return a list of the values from **d**
`d.items()` - Return a list of **(key, value)** pairs from **d**

MODULES AND FUNCTIONS

The body of a function is defined through indentation.

`import random` - Import the module **random**
`from math import sqrt` - Import the function **sqrt** from the module **math**

```
def calculate(addition_one,addition_two,
exponent=1,factor=1):
    result = (value_one + value_two) ** exponent * factor
    return result
```

- Define a new function **calculate** with two required and two optional named arguments which calculates and returns a result.

`addition(3,5,factor=10)` - Run the **addition** function with the values **3** and **5** and the named argument **10**

BOOLEAN COMPARISONS

`x == 5` - Test whether **x** is equal to **5**
`x != 5` - Test whether **x** is not equal to **5**
`x > 5` - Test whether **x** is greater than **5**
`x < 5` - Test whether **x** is less than **5**
`x >= 5` - Test whether **x** is greater than or equal to **5**
`x <= 5` - Test whether **x** is less than or equal to **5**
`x == 5 or name == "alfred"` - Test whether **x** is equal to **5** or **name** is equal to **"alfred"**
`x == 5 and name == "alfred"` - Test whether **x** is equal to **5** and **name** is equal to **"alfred"**
`5 in l` - Checks whether the value **5** exists in the list **l**
`"GB" in d` - Checks whether the value **"GB"** exists in the keys for **d**

IF STATEMENTS AND LOOPS

The body of if statements and loops are defined through indentation.

```
if x > 5:
    print("{} is greater than five".format(x))
elif x < 0:
    print("{} is negative".format(x))
else:
    print("{} is between zero and five".format(x))
```

- Test the value of the variable **x** and run the code body based on the value

```
for value in l:
    print(value)
```

- Iterate over each value in **l**, running the code in the body of the loop with each iteration

```
while x < 10:
    x += 1
```

- Run the code in the body of the loop until the value of **x** is no longer less than **10**

Data Science Cheat Sheet

Python - Intermediate

KEY BASICS, PRINTING AND GETTING HELP

This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet

`s` - A Python string variable
`i` - A Python integer variable
`f` - A Python float variable
`l` - A Python list variable
`d` - A Python dictionary variable

LISTS

`l.pop(3)` - Returns the fourth item from `l` and deletes it from the list
`l.remove(x)` - Removes the first item in `l` that is equal to `x`
`l.reverse()` - Reverses the order of the items in `l`
`l[1::2]` - Returns every second item from `l`, commencing from the `1`st item
`l[-5:]` - Returns the last 5 items from `l` specific axis

STRINGS

`s.lower()` - Returns a lowercase version of `s`
`s.title()` - Returns `s` with the first letter of every word capitalized
`"23".zfill(4)` - Returns `"0023"` by left-filling the string with `0`'s to make it's length `4`.
`s.splitlines()` - Returns a list by splitting the string on any newline characters.

Python strings share some common methods with lists

`s[:5]` - Returns the first `5` characters of `s`
`"fri" + "end"` - Returns `"friend"`
`"end" in s` - Returns `True` if the substring `"end"` is found in `s`

RANGE

Range objects are useful for creating sequences of integers for looping.

`range(5)` - Returns a sequence from `0` to `4`
`range(2000,2018)` - Returns a sequence from `2000` to `2017`
`range(0,11,2)` - Returns a sequence from `0` to `10`, with each item incrementing by `2`
`range(0,-10,-1)` - Returns a sequence from `0` to `-9`
`list(range(5))` - Returns a list from `0` to `4`

DICTIONARIES

`max(d, key=d.get)` - Return the key that corresponds to the largest value in `d`
`min(d, key=d.get)` - Return the key that corresponds to the smallest value in `d`

SETS

`my_set = set(l)` - Return a `set` object containing the unique values from `l`
`len(my_set)` - Returns the number of objects in `my_set` (or, the number of unique values from `l`)
`a in my_set` - Returns `True` if the value `a` exists in `my_set`

REGULAR EXPRESSIONS

`import re` - Import the Regular Expressions module
`re.search("abc",s)` - Returns a `match` object if the regex `"abc"` is found in `s`, otherwise `None`
`re.sub("abc","xyz",s)` - Returns a string where all instances matching regex `"abc"` are replaced by `"xyz"`

LIST COMPREHENSION

A one-line expression of a for loop

`[i ** 2 for i in range(10)]` - Returns a list of the squares of values from `0` to `9`
`[s.lower() for s in l_strings]` - Returns the list `l_strings`, with each item having had the `.lower()` method applied
`[i for i in l_floats if i < 0.5]` - Returns the items from `l_floats` that are less than `0.5`

FUNCTIONS FOR LOOPING

```
for i, value in enumerate(l):
  print("The value of item {} is {}".
  format(i,value))
```

- Iterate over the list `l`, printing the index location of each item and its value

```
for one, two in zip(l_one,l_two):
  print("one: {}, two: {}".format(one,two))
```

- Iterate over two lists, `l_one` and `l_two` and print each value

```
while x < 10:
  x += 1
```

- Run the code in the body of the loop until the value of `x` is no longer less than `10`

DATETIME

`import datetime as dt` - Import the `datetime` module
`now = dt.datetime.now()` - Assign `datetime` object representing the current time to `now`
`wks4 = dt.datetime.timedelta(weeks=4)` - Assign a `timedelta` object representing a timespan of 4 weeks to `wks4`
`now - wks4` - Return a `datetime` object representing the time 4 weeks prior to `now`
`newyear_2020 = dt.datetime(year=2020, month=12, day=31)` - Assign a `datetime` object representing December 25, 2020 to `newyear_2020`
`newyear_2020.strftime("%A, %b %d, %Y")` - Returns `"Thursday, Dec 31, 2020"`
`dt.datetime.strptime('Dec 31, 2020',"%b %d, %Y")` - Return a `datetime` object representing December 31, 2020

RANDOM

`import random` - Import the `random` module
`random.random()` - Returns a random float between `0.0` and `1.0`
`random.randint(0,10)` - Returns a random integer between `0` and `10`
`random.choice(l)` - Returns a random item from the list `l`

COUNTER

`from collections import Counter` - Import the `Counter` class
`c = Counter(l)` - Assign a `Counter` (dict-like) object with the counts of each unique item from `l`, to `c`
`c.most_common(3)` - Return the 3 most common items from `l`

TRY/EXCEPT

Catch and deal with Errors

`l_ints = [1, 2, 3, "", 5]` - Assign a list of integers with one missing value to `l_ints`

```
l_floats = []
for i in l_ints:
  try:
    l_floats.append(float(i))
  except:
    l_floats.append(i)
```

- Convert each value of `l_ints` to a float, catching and handling `ValueError: could not convert string to float:` where values are missing.

Data Science Cheat Sheet

NumPy

KEY

We'll use shorthand in this cheat sheet

`arr` - A numpy Array object

IMPORTS

Import these to start

```
import numpy as np
```

IMPORTING/EXPORTING

`np.loadtxt('file.txt')` - From a text file

`np.genfromtxt('file.csv',delimiter=',')` - From a CSV file

`np.savetxt('file.txt',arr,delimiter=' ')` - Writes to a text file

`np.savetxt('file.csv',arr,delimiter=',')` - Writes to a CSV file

CREATING ARRAYS

`np.array([1,2,3])` - One dimensional array

`np.array([(1,2,3),(4,5,6)])` - Two dimensional array

`np.zeros(3)` - 1D array of length **3** all values **0**

`np.ones((3,4))` - **3**x**4** array with all values **1**

`np.eye(5)` - **5**x**5** array of **0** with **1** on diagonal (Identity matrix)

`np.linspace(0,100,6)` - Array of **6** evenly divided values from **0** to **100**

`np.arange(0,10,3)` - Array of values from **0** to less than **10** with step **3** (eg **[0,3,6,9]**)

`np.full((2,3),8)` - **2**x**3** array with all values **8**

`np.random.rand(4,5)` - **4**x**5** array of random floats between **0**-**1**

`np.random.rand(6,7)*100` - **6**x**7** array of random floats between **0**-**100**

`np.random.randint(5,size=(2,3))` - **2**x**3** array with random ints between **0**-**4**

INSPECTING PROPERTIES

`arr.size` - Returns number of elements in `arr`

`arr.shape` - Returns dimensions of `arr` (rows, columns)

`arr.dtype` - Returns type of elements in `arr`

`arr.astype(dtype)` - Convert `arr` elements to type `dtype`

`arr.tolist()` - Convert `arr` to a Python list

`np.info(np.eye)` - View documentation for `np.eye`

COPYING/SORTING/RESHAPING

`np.copy(arr)` - Copies `arr` to new memory

`arr.view(dtype)` - Creates view of `arr` elements with type `dtype`

`arr.sort()` - Sorts `arr`

`arr.sort(axis=0)` - Sorts specific axis of `arr`

`two_d_arr.flatten()` - Flattens 2D array `two_d_arr` to 1D

`arr.T` - Transposes `arr` (rows become columns and vice versa)

`arr.reshape(3,4)` - Reshapes `arr` to **3** rows, **4** columns without changing data

`arr.resize((5,6))` - Changes `arr` shape to **5**x**6** and fills new values with **0**

ADDING/REMOVING ELEMENTS

`np.append(arr,values)` - Appends `values` to end of `arr`

`np.insert(arr,2,values)` - Inserts `values` into `arr` before index **2**

`np.delete(arr,3,axis=0)` - Deletes row on index **3** of `arr`

`np.delete(arr,4,axis=1)` - Deletes column on index **4** of `arr`

COMBINING/SPLITTING

`np.concatenate((arr1,arr2),axis=0)` - Adds `arr2` as rows to the end of `arr1`

`np.concatenate((arr1,arr2),axis=1)` - Adds `arr2` as columns to end of `arr1`

`np.split(arr,3)` - Splits `arr` into **3** sub-arrays

`np.hsplit(arr,5)` - Splits `arr` horizontally on the **5**th index

INDEXING/SLICING/SUBSETTING

`arr[5]` - Returns the element at index **5**

`arr[2,5]` - Returns the 2D array element on index **[2][5]**

`arr[1]=4` - Assigns array element on index **1** the value **4**

`arr[1,3]=10` - Assigns array element on index **[1][3]** the value **10**

`arr[0:3]` - Returns the elements at indices **0,1,2** (On a 2D array: returns rows **0,1,2**)

`arr[0:3,4]` - Returns the elements on rows **0,1,2** at column **4**

`arr[:2]` - Returns the elements at indices **0,1** (On a 2D array: returns rows **0,1**)

`arr[:,1]` - Returns the elements at index **1** on all rows

`arr<5` - Returns an array with boolean values

`(arr1<3) & (arr2>5)` - Returns an array with boolean values

`~arr` - Inverts a boolean array

`arr[arr<5]` - Returns array elements smaller than **5**

SCALAR MATH

`np.add(arr,1)` - Add **1** to each array element

`np.subtract(arr,2)` - Subtract **2** from each array element

`np.multiply(arr,3)` - Multiply each array element by **3**

`np.divide(arr,4)` - Divide each array element by **4** (returns `np.nan` for division by zero)

`np.power(arr,5)` - Raise each array element to the **5**th power

VECTOR MATH

`np.add(arr1,arr2)` - Elementwise add `arr2` to `arr1`

`np.subtract(arr1,arr2)` - Elementwise subtract `arr2` from `arr1`

`np.multiply(arr1,arr2)` - Elementwise multiply `arr1` by `arr2`

`np.divide(arr1,arr2)` - Elementwise divide `arr1` by `arr2`

`np.power(arr1,arr2)` - Elementwise raise `arr1` raised to the power of `arr2`

`np.array_equal(arr1,arr2)` - Returns `True` if the arrays have the same elements and shape

`np.sqrt(arr)` - Square root of each element in the array

`np.sin(arr)` - Sine of each element in the array

`np.log(arr)` - Natural log of each element in the array

`np.abs(arr)` - Absolute value of each element in the array

`np.ceil(arr)` - Rounds up to the nearest int

`np.floor(arr)` - Rounds down to the nearest int

`np.round(arr)` - Rounds to the nearest int

STATISTICS

`np.mean(arr,axis=0)` - Returns mean along specific axis

`arr.sum()` - Returns sum of `arr`

`arr.min()` - Returns minimum value of `arr`

`arr.max(axis=0)` - Returns maximum value of specific axis

`np.var(arr)` - Returns the variance of array

`np.std(arr,axis=1)` - Returns the standard deviation of specific axis

`arr.corrcoef()` - Returns correlation coefficient of array

Data Science Cheat Sheet

Pandas

KEY

We'll use shorthand in this cheat sheet

`df` - A pandas DataFrame object

`s` - A pandas Series object

IMPORTS

Import these to start

```
import pandas as pd
import numpy as np
```

IMPORTING DATA

`pd.read_csv(filename)` - From a CSV file

`pd.read_table(filename)` - From a delimited text file (like TSV)

`pd.read_excel(filename)` - From an Excel file

`pd.read_sql(query, connection_object)` - Reads from a SQL table/database

`pd.read_json(json_string)` - Reads from a JSON formatted string, URL or file.

`pd.read_html(url)` - Parses an html URL, string or file and extracts tables to a list of dataframes

`pd.read_clipboard()` - Takes the contents of your clipboard and passes it to `read_table()`

`pd.DataFrame(dict)` - From a dict, keys for columns names, values for data as lists

EXPORTING DATA

`df.to_csv(filename)` - Writes to a CSV file

`df.to_excel(filename)` - Writes to an Excel file

`df.to_sql(table_name, connection_object)` - Writes to a SQL table

`df.to_json(filename)` - Writes to a file in JSON format

`df.to_html(filename)` - Saves as an HTML table

`df.to_clipboard()` - Writes to the clipboard

CREATE TEST OBJECTS

Useful for testing

`pd.DataFrame(np.random.rand(20,5))` - 5 columns and 20 rows of random floats

`pd.Series(my_list)` - Creates a series from an iterable `my_list`

`df.index = pd.date_range('1900/1/30', periods=df.shape[0])` - Adds a date index

VIEWING/INSPECTING DATA

`df.head(n)` - First `n` rows of the DataFrame

`df.tail(n)` - Last `n` rows of the DataFrame

`df.shape()` - Number of rows and columns

`df.info()` - Index, Datatype and Memory information

`df.describe()` - Summary statistics for numerical columns

`s.value_counts(dropna=False)` - Views unique values and counts

`df.apply(pd.Series.value_counts)` - Unique values and counts for all columns

SELECTION

`df[col]` - Returns column with label `col` as Series

`df[[col1, col2]]` - Returns Columns as a new DataFrame

`s.iloc[0]` - Selection by position

`s.loc[0]` - Selection by index

`df.iloc[0,:]` - First row

`df.iloc[0,0]` - First element of first column

DATA CLEANING

`df.columns = ['a','b','c']` - Renames columns

`pd.isnull()` - Checks for null Values, Returns Boolean Array

`pd.notnull()` - Opposite of `s.isnull()`

`df.dropna()` - Drops all rows that contain null values

`df.dropna(axis=1)` - Drops all columns that contain null values

`df.dropna(axis=1,thresh=n)` - Drops all rows have have less than `n` non null values

`df.fillna(x)` - Replaces all null values with `x`

`s.fillna(s.mean())` - Replaces all null values with the mean (mean can be replaced with almost any function from the statistics section)

`s.astype(float)` - Converts the datatype of the series to float

`s.replace(1,'one')` - Replaces all values equal to `1` with `'one'`

`s.replace([1,3],['one','three'])` - Replaces all `1` with `'one'` and `3` with `'three'`

`df.rename(columns=lambda x: x + 1)` - Mass renaming of columns

`df.rename(columns={'old_name': 'new_name'})` - Selective renaming

`df.set_index('column_one')` - Changes the index

`df.rename(index=lambda x: x + 1)` - Mass renaming of index

FILTER, SORT, & GROUPBY

`df[df[col] > 0.5]` - Rows where the `col` column is greater than `0.5`

`df[(df[col] > 0.5) & (df[col] < 0.7)]` - Rows where `0.7 > col > 0.5`

`df.sort_values(col1)` - Sorts values by `col1` in ascending order

`df.sort_values(col2,ascending=False)` - Sorts values by `col2` in descending order

`df.sort_values([col1,col2], ascending=[True,False])` - Sorts values by `col1` in ascending order then `col2` in descending order

`df.groupby(col)` - Returns a groupby object for values from one column

`df.groupby([col1,col2])` - Returns a groupby object values from multiple columns

`df.groupby(col1)[col2].mean()` - Returns the mean of the values in `col2`, grouped by the values in `col1` (mean can be replaced with almost any function from the statistics section)

`df.pivot_table(index=col1,values=[col2,col3],aggfunc=mean)` - Creates a pivot table that groups by `col1` and calculates the mean of `col2` and `col3`

`df.groupby(col1).agg(np.mean)` - Finds the average across all columns for every unique column 1 group

`df.apply(np.mean)` - Applies a function across each column

`df.apply(np.max, axis=1)` - Applies a function across each row

JOIN/COMBINE

`df1.append(df2)` - Adds the rows in `df1` to the end of `df2` (columns should be identical)

`pd.concat([df1, df2],axis=1)` - Adds the columns in `df1` to the end of `df2` (rows should be identical)

`df1.join(df2,on=col1,how='inner')` - SQL-style joins the columns in `df1` with the columns on `df2` where the rows for `col` have identical values. `how` can be one of `'left'`, `'right'`, `'outer'`, `'inner'`

STATISTICS

These can all be applied to a series as well.

`df.describe()` - Summary statistics for numerical columns

`df.mean()` - Returns the mean of all columns

`df.corr()` - Returns the correlation between columns in a DataFrame

`df.count()` - Returns the number of non-null values in each DataFrame column

`df.max()` - Returns the highest value in each column

`df.min()` - Returns the lowest value in each column

`df.median()` - Returns the median of each column

`df.std()` - Returns the standard deviation of each column

Data Science Cheat Sheet

Python Regular Expressions

SPECIAL CHARACTERS

`^` | Matches the expression to its right at the start of a string. It matches every such instance before each `\n` in the string.

`$` | Matches the expression to its left at the end of a string. It matches every such instance before each `\n` in the string.

`.` | Matches any character except line terminators like `\n`.

`\` | Escapes special characters or denotes character classes.

`A|B` | Matches expression `A` or `B`. If A is matched first, B is left untried.

`+` | Greedily matches the expression to its left 1 or more times.

`*` | Greedily matches the expression to its left 0 or more times.

`?` | Greedily matches the expression to its left 0 or 1 times. But if `?` is added to qualifiers (`+`, `*`, and `?` itself) it will perform matches in a non-greedy manner.

`{m}` | Matches the expression to its left `m` times, and not less.

`{m,n}` | Matches the expression to its left `m` to `n` times, and not less.

`{m,n}?` | Matches the expression to its left `m` times, and ignores `n`. See `?` above.

CHARACTER CLASSES [A.K.A. SPECIAL SEQUENCES]

`\w` | Matches alphanumeric characters, which means `a-z`, `A-Z`, and `0-9`. It also matches the underscore, `_`.

`\d` | Matches digits, which means `0-9`.

`\D` | Matches any non-digits.

`\s` | Matches whitespace characters, which include the `\t`, `\n`, `\r`, and space characters.

`\S` | Matches non-whitespace characters.

`\b` | Matches the boundary (or empty string) at the start and end of a word, that is, between `\w` and `\W`.

`\B` | Matches where `\b` does not, that is, the boundary of `\w` characters.

`\A` | Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.

`\Z` | Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.

SETS

`[ ]` | Contains a set of characters to match.

`[amk]` | Matches either `a`, `m`, or `k`. It does not match `amk`.

`[a-z]` | Matches any alphabet from `a` to `z`.

`[a\-z]` | Matches `a`, `-`, or `z`. It matches `-` because `\` escapes it.

`[a-]` | Matches `a` or `-`, because `-` is not being used to indicate a series of characters.

`[-a]` | As above, matches `a` or `-`.

`[a-z0-9]` | Matches characters from `a` to `z` and also from `0` to `9`.

`[(+*)]` | Special characters become literal inside a set, so this matches `(`, `+`, `*`, and `)`.

`[^ab5]` | Adding `^` excludes any character in the set. Here, it matches characters that are not `a`, `b`, or `5`.

GROUPS

`( )` | Matches the expression inside the parentheses and groups it.

`(?)` | Inside parentheses like this, `?` acts as an extension notation. Its meaning depends on the character immediately to its right.

`(?PAB)` | Matches the expression `AB`, and it can be accessed with the group name.

`(?aiLmsux)` | Here, `a`, `i`, `L`, `m`, `s`, `u`, and `x` are flags:
- `a` — Matches ASCII only
- `i` — Ignore case
- `L` — Locale dependent
- `m` — Multi-line
- `s` — Matches all
- `u` — Matches unicode
- `x` — Verbose

`(?:A)` | Matches the expression as represented by `A`, but unlike `(?PAB)`, it cannot be retrieved afterwards.

`(?#...)` | A comment. Contents are for us to read, not for matching.

`A(?=B)` | Lookahead assertion. This matches the expression `A` only if it is followed by `B`.

`A(?!B)` | Negative lookahead assertion. This matches the expression `A` only if it is not followed by `B`.

`(?<=B)A` | Positive lookbehind assertion. This matches the expression `A` only if `B` is immediately to its left. This can only matched fixed length expressions.

`(?<!B)A` | Negative lookbehind assertion. This matches the expression `A` only if `B` is not immediately to its left. This can only matched fixed length expressions.

`(?P=name)` | Matches the expression matched by an earlier group named "name".

`(...)\1` | The number `1` corresponds to the first group to be matched. If we want to match more instances of the same expression, simply use its number instead of writing out the whole expression again. We can use from `1` up to `99` such groups and their corresponding numbers.

POPULAR PYTHON RE MODULE FUNCTIONS

`re.findall(A, B)` | Matches all instances of an expression `A` in a string `B` and returns them in a list.

`re.search(A, B)` | Matches the first instance of an expression `A` in a string `B`, and returns it as a re match object.

`re.split(A, B)` | Split a string `B` into a list using the delimiter `A`.

`re.sub(A, B, C)` | Replace `A` with `B` in the string `C`.

Python For Data Science *Cheat Sheet*

Importing Data

Learn Python for data science Interactively at www.DataCamp.com

Importing Data in Python

Most of the time, you'll use either **NumPy** or **pandas** to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager `with`

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t',
                       comment='#',
                       na_values=[""])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata Files

```
>>> data = pd.read_stata('urbanpop.dta')
```

Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager `with`

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to an a NumPy array

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys
Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure
Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls
%cd ..
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

`os` Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file
Delete an existing file
Create a new directory

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

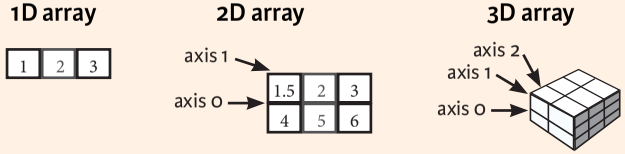
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                          Array-wise sum
>>> a.min()                          Array-wise minimum value
>>> b.max(axis=0)                    Maximum value of an array row
>>> b.cumsum(axis=1)                 Cumulative sum of the elements
>>> a.mean()                         Mean
>>> b.median()                       Median
>>> a.corrcoef()                     Correlation coefficient
>>> np.std(b)                        Standard deviation
```

Copying Arrays

```
>>> h = a.view()     Create a view of the array with the same data
>>> np.copy(a)       Create a copy of the array
>>> h = a.copy()     Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()         Sort an array
>>> c.sort(axis=0)   Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting


```
>>> a[2]
  3
```

Select the element at the 2nd index


```
>>> b[1,2]
  6.0
```

Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing


```
>>> a[0:2]
  array([1, 2])
```

Select items at index 0 and 1


```
>>> b[0:2,1]
  array([ 2.,  5.])
```

Select items at rows 0 and 1 in column 1


```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
```

Same as `[1,:,:]`

```
>>> a[ : :-1]
  array([3, 2, 1])
```

Reversed array `a`

Boolean Indexing


```
>>> a[a<2]
  array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
  array([ 4. , 2. , 6. , 1.5])
```

Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
  array([[ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)     Permute array dimensions
>>> i.T                     Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()               Flatten the array
>>> g.reshape(3,-2)         Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))         Return a new array with shape (2,6)
>>> np.append(h,g)          Append items to an array
>>> np.insert(a, 1, 5)      Insert items in an array
>>> np.delete(a,[1])        Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd index
  [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

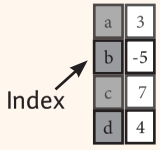
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

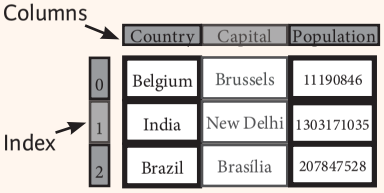
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country    Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat([0],[0])
'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at([0], ['Country'])
'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital    Brasília
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0     Brussels
1    New Delhi
2     Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a     10.0
b      NaN
c      5.0
d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c    5.0
d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Python For Data Science *Cheat Sheet*

Pandas

Learn Python for Data Science **Interactively** at www.DataCamp.com

Reshaping Data

Pivot

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

Spread rows into columns

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

Spread rows into columns

Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

Gather columns into rows

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Iteration

```
>>> df.iteritems()
>>> df.iterrows()
```

(Column-index, Series) pairs
(Row-index, Series) pairs

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

Select cols with any vals >1
Select cols with vals > 1
Select cols with NaN
Select cols without NaN

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

Find same elements
Filter on values
Select specific elements

Where

```
>>> s.where(s > 0)
```

Subset the data

Query

```
>>> df6.query('second > first')
```

Query DataFrame

Setting/Resetting Index

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

Set the index
Reset the index
Rename DataFrame

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528
3	Brazil	Brasília	207847528

Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

0	3
1	3
2	3
3	3
4	3

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

Duplicate Data

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),
                              'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

Drop NaN values
Fill NaN values with a predetermined value
Replace values with others

Combining Data

data1

	X1	X2
a	11.432
b	1.303
c	99.906

data2

	X1	X3
a	20.784
b	NaN
d	20.784

Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

Dates

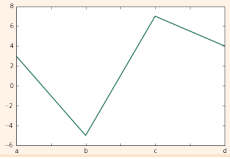
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

Visualization

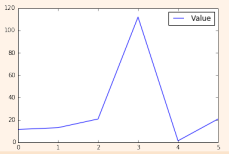
Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```



```
>>> df2.plot()
>>> plt.show()
```



Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

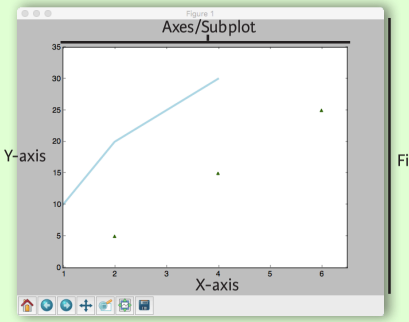
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Figure

Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	Create a figure and one subplot

Seaborn styles

`>>> sns.set()`	(Re)set the seaborn default
`>>> sns.set_style("whitegrid")`	Set the matplotlib parameters
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	Set the matplotlib parameters
`>>> sns.axes_style("whitegrid")`	Return a dict of params or use with `with` to temporarily set the style

Context Functions

`>>> sns.set_context("talk")`	Set context to `"talk"`
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

`>>> sns.set_palette("husl",3)`	Define the color palette
`>>> sns.color_palette("husl")`	Use with `with` to temporarily set palette
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	Set your own color palette

3 Plotting With Seaborn

Axis Grids

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	Draw a categorical plot onto a Facetgrid
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	Plot data and regression model fits across a FacetGrid
`>>> h = sns.PairGrid(iris)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	
`>>> sns.pairplot(iris)`	Plot pairwise bivariate distributions
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	Grid for bivariate plot with marginal univariate plots
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	Plot bivariate distribution

Categorical Plots

Scatterplot

`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	Scatterplot with one categorical variable
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	Categorical scatterplot with non-overlapping points

Bar Chart

`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	Show count of observations

Point Plot

`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	Show point estimates and confidence intervals as rectangular bars

Boxplot

`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	Boxplot
`>>> sns.boxplot(data=iris,orient="h")`	Boxplot with wide-form data

Violinplot

`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	Violin plot

Regression Plots

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	Plot data and a linear regression model fit

Distribution Plots

`>>> plot = sns.distplot(data.y, kde=False, color="b")`	Plot univariate distribution

Matrix Plots

`>>> sns.heatmap(uniform_data,vmin=0,vmax=1)`	Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

`>>> g.despine(left=True)`	Remove left spine
`>>> g.set_ylabels("Survived")`	Set the labels of the y-axis
`>>> g.set_xticklabels(rotation=45)`	Set the tick labels for x
`>>> g.set_axis_labels("Survived", "Sex")`	Set the axis labels
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	Set the limit and ticks of the x-and y-axis

Plot

`>>> plt.title("A Title")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear an entire figure
`>>> plt.close()`	Close a window

Python For Data Science *Cheat Sheet*

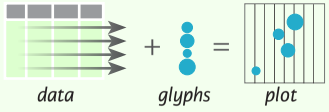
Bokeh

Learn Bokeh Interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:

Python lists, NumPy arrays, Pandas DataFrames and other sequences of values

2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                          Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",      Step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)   Step 3
>>> output_file("lines.html")                    Step 4
>>> show(p)                                      Step 5
```

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

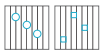
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
                x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

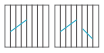
3 Renderers & Visual Customizations

Glyphs



Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                        transform=color_mapper),
             legend='Origin')
```

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" ,[p1, r1]),("Two",[r2])],
                    location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Linked Plots

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100,
                tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200,
                tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

4 Output & Export

Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

HTML

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
```

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> save(p1)
```

```
>>> show(layout)
>>> save(layout)
```

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

`>>> np.mgrid[0:5,0:5]`	Create a dense meshgrid
`>>> np.ogrid[0:2,0:2]`	Create an open meshgrid
`>>> np.r_[[3,[0]*5,-1:1:10j]`	Stack arrays vertically (row-wise)
`>>> np.c_[b,c]`	Create stacked column-wise arrays

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.hstack((b,c))`	Stack arrays horizontally (column-wise)
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.hsplit(c,2)`	Split the array horizontally at the 2nd index
`>>> np.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

`>>> from numpy import poly1d`	
`>>> p = poly1d([3,4,5])`	Create a polynomial object

Vectorizing Functions

`>>> def myfunc(a):`
`  if a < 0:`
`  return a*2`
`  else:`
`  return a/2`	

`>>> np.vectorize(myfunc)`	Vectorize functions

Type Handling

`>>> np.real(c)`	Return the real part of the array elements
`>>> np.imag(c)`	Return the imaginary part of the array elements
`>>> np.real_if_close(c,tol=1000)`	Return a real array if complex parts close to 0
`>>> np.cast['f'](np.pi)`	Cast object to a data type

Other Useful Functions

`>>> np.angle(b,deg=True)`	Return the angle of the complex argument
`>>> g = np.linspace(0,np.pi,num=5)`	Create an array of evenly spaced values (number of samples)
`>>> g [3:] += np.pi`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

`>>> A.I`	Inverse
`>>> linalg.inv(A)`	Inverse
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
`>>> np.trace(A)`	Trace

Norm

`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	L inf norm (max row sum)

Rank

`>>> np.linalg.matrix_rank(C)`	Matrix rank

Determinant

`>>> linalg.det(A)`	Determinant

Solving linear problems

`>>> linalg.solve(A,b)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(D,E)`	Least-squares solution to linear matrix equation

Generalized inverse

`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

`>>> F = np.eye(3, k=1)`	Create a 2X2 identity matrix
`>>> G = np.mat(np.identity(2))`	Create a 2x2 identity matrix
`>>> C[C > 0.5] = 0`	
`>>> H = sparse.csr_matrix(C)`	Compressed Sparse Row matrix
`>>> I = sparse.csc_matrix(D)`	Compressed Sparse Column matrix
`>>> J = sparse.dok_matrix(A)`	Dictionary Of Keys matrix
`>>> E.todense()`	Sparse matrix to full matrix
`>>> sparse.isspmatrix_csc(A)`	Identify sparse matrix

Sparse Matrix Routines

Inverse

`>>> sparse.linalg.inv(I)`	Inverse

Norm

`>>> sparse.linalg.norm(I)`	Norm

Solving linear problems

`>>> sparse.linalg.spsolve(H,I)`	Solver for sparse matrices

Sparse Matrix Functions

`>>> sparse.linalg.expm(I)`	Sparse matrix exponential

Matrix Functions

Addition

`>>> np.add(A,D)`	Addition

Subtraction

`>>> np.subtract(A,D)`	Subtraction

Division

`>>> np.divide(A,D)`	Division

Multiplication

`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product

Exponential Functions

`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)

Logarithm Function

`>>> linalg.logm(A)`	Matrix logarithm

Trigonometric Tunctions

`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent

Hyperbolic Trigonometric Functions

`>>> linalg.sinhm(D)`	Hypberbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent

Matrix Sign Function

`>>> np.sigm(A)`	Matrix sign function

Matrix Square Root

`>>> linalg.sqrtm(A)`	Matrix square root

Arbitrary Functions

`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues

Singular Value Decomposition

`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD

LU Decomposition

`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Python For Data Science *Cheat Sheet*

PySpark - SQL Basics

Learn Python for data science Interactively at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
```

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +--------+---+
  |    name|age|
  +--------+---+
  |    Mine| 28|
  |   Filip| 29|
  |Jonathan| 30|
  +--------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName |lastName|         phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
               "firstName",
               "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)
      .show()
>>> df.select(df['age'] > 24).show()
```

When

```
>>> df.select("firstName",
               F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]
                .collect()
```

Like

```
>>> df.select("firstName",
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName",
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th")) \
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \
                       .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \
      .show()
```

Show all entries in `firstName` column

Show all entries in `firstName`, `age` and `type`

Show all entries in `firstName` and `age`, add `1` to the entries of `age`

Show all entries where `age` >24

Show `firstName` and 0 or 1 depending on `age` >30

Show `firstName` if in the given options

Show `firstName`, and `lastName` is `TRUE` if `lastName` is like `Smith`

Show `firstName`, and `TRUE` if `lastName` starts with `Sm`

Show last names ending in `th`

Return substrings of `firstName`

Show `age`: values are `TRUE` if between 22 and 24

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age")\
      .count() \
      .show()
```

Group by `age`, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of `age`, only keep those records of which the values are >24

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
      .replace(10, 20) \
      .show()
```

Replace null values
Return new `df` omitting rows with null values
Return new `df` replacing one value with another

Repartitioning

```
>>> df.repartition(10)\
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

`df` with 10 partitions

`df` with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert `df` into an RDD
Convert `df` into a RDD of string
Return the contents of `df` as Pandas `DataFrame`

Write & Save to Files

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

Inspect Data

```
>>> df.dtypes
>>> df.show()
>>> df.head()
>>> df.first()
>>> df.take(2)
>>> df.schema
```

Return `df` column names and data types
Display the content of `df`
Return first `n` rows
Return first row
Return the first `n` rows
Return the schema of `df`

```
>>> df.describe().show()
>>> df.columns
>>> df.count()
>>> df.distinct().count()
>>> df.printSchema()
>>> df.explain()
```

Compute summary statistics
Return the columns of `df`
Count the number of rows in `df`
Count the number of distinct rows in `df`
Print the schema of `df`
Print the (logical and physical) plans

Python For Data Science *Cheat Sheet*

PySpark - RDD Basics

Learn Python for data science Interactively at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version                  Retrieve SparkContext version
>>> sc.pythonVer                Retrieve Python version
>>> sc.master                   Master URL to connect to
>>> str(sc.sparkHome)           Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())         Retrieve name of the Spark User running SparkContext
>>> sc.appName                  Return application name
>>> sc.applicationId            Retrieve application ID
>>> sc.defaultParallelism       Return default level of parallelism
>>> sc.defaultMinPartitions     Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()      List the number of partitions
>>> rdd.count()                 Count RDD instances
 3
>>> rdd.countByKey()            Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()          Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()          Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()                  Sum of RDD elements
 4950
>>> sc.parallelize([]).isEmpty()  Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max()                  Maximum value of RDD elements
  99
>>> rdd3.min()                  Minimum value of RDD elements
  0
>>> rdd3.mean()                 Mean value of RDD elements
  49.5
>>> rdd3.stdev()                Standard deviation of RDD elements
  28.866070047722118
>>> rdd3.variance()             Compute variance of RDD elements
  833.25
>>> rdd3.histogram(3)           Compute histogram by bins
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                Summary statistics (count, mean, stdev, max & min)
```

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element and flatten the result

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)   Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Selecting Data

Getting

```
>>> rdd.collect()               Return a list with all RDD elements
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                 Take first 2 RDD elements
  [('a', 7), ('a', 2)]
>>> rdd.first()                 Take first RDD element
  ('a', 7)
>>> rdd.top(2)                  Take top 2 RDD elements
  [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()   Return sampled subset of rdd3
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x)   Filter the RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()   Return distinct RDD values
  ['a',2,'b',7]
>>> rdd.keys().collect()        Return (key,value) RDD's keys
  ['a', 'a', 'b']
```

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)              Apply a function to all RDD elements
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)   Merge the rdd values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)   Merge the rdd values
  ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)   Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()            Group rdd by key
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)   Aggregate RDD elements of each partition and then the results
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)   Aggregate values of each RDD key
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)            Aggregate the elements of each partition, and then the results
  4950
>>> rdd.foldByKey(0, add)       Merge the values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)   Create tuples of RDD elements by applying a function
       .collect()
```

Mathematical Operations

```
>>> rdd.subtract(rdd2)          Return each rdd value not contained in rdd2
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)     Return each (key,value) pair of rdd2 with no matching key in rdd
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()   Return the Cartesian product of rdd and rdd2
```

Sort

```
>>> rdd2.sortBy(lambda x: x[1])   Sort RDD by given function
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()            Sort (key, value) RDD by key
       .collect()
  [('a',2),('b',1),('d',1)]
```

Repartitioning

```
>>> rdd.repartition(4)          New RDD with 4 partitions
>>> rdd.coalesce(1)             Decrease the number of partitions in the RDD to 1
```

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Python for Data Science Cheat Sheet spaCy

Learn more Python for data science interactively at www.datacamp.com

DataCamp

About spaCy

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. **Documentation:** spacy.io

```
$ pip install spacy
```

```
import spacy
```

Statistical models

Download statistical models

Predict part-of-speech tags, dependency labels, named entities and more. See here for available models: spacy.io/models

```
$ python -m spacy download en_core_web_sm
```

Check that your installed models are up to date

```
$ python -m spacy validate
```

Loading statistical models

```
import spacy
# Load the installed model "en_core_web_sm"
nlp = spacy.load("en_core_web_sm")
```

Documents and tokens

Processing text

Processing text with the `nlp` object returns a `Doc` object that holds all information about the tokens, their linguistic features and their relationships

```
doc = nlp("This is a text")
```

Accessing token attributes

```
doc = nlp("This is a text")
# Token texts
[token.text for token in doc]
# ['This', 'is', 'a', 'text']
```

Spans

Accessing spans

Span indices are **exclusive**. So `doc[2:4]` is a span starting at token 2, up to – but not including! – token 4.

```
doc = nlp("This is a text")
span = doc[2:4]
span.text
# 'a text'
```

Creating a span manually

```
# Import the Span object
from spacy.tokens import Span
# Create a Doc object
doc = nlp("I live in New York")
# Span for "New York" with label GPE (geopolitical)
span = Span(doc, 3, 5, label="GPE")
span.text
# 'New York'
```

Linguistic features

Attributes return label IDs. For string labels, use the attributes with an underscore. For example, `token.pos_`.

Part-of-speech tags

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Coarse-grained part-of-speech tags
[token.pos_ for token in doc]
# ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
# Fine-grained part-of-speech tags
[token.tag_ for token in doc]
# ['DT', 'VBZ', 'DT', 'NN', '.']
```

Syntactic dependencies

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Dependency labels
[token.dep_ for token in doc]
# ['nsubj', 'ROOT', 'det', 'attr', 'punct']
# Syntactic head token (governor)
[token.head.text for token in doc]
# ['is', 'is', 'text', 'is', 'is']
```

Named entities

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("Larry Page founded Google")
# Text and label of named entity span
[(ent.text, ent.label_) for ent in doc.ents]
# [('Larry Page', 'PERSON'), ('Google', 'ORG')]
```

Syntax iterators

Sentences

USUALLY NEEDS THE DEPENDENCY PARSER

```
doc = nlp("This a sentence. This is another one.")
# doc.sents is a generator that yields sentence spans
[sent.text for sent in doc.sents]
# ['This is a sentence.', 'This is another one.']
```

Base noun phrases

NEEDS THE TAGGER AND PARSER

```
doc = nlp("I have a red car")
# doc.noun_chunks is a generator that yields spans
[chunk.text for chunk in doc.noun_chunks]
# ['I', 'a red car']
```

Label explanations

```
spacy.explain("RB")
# 'adverb'
spacy.explain("GPE")
# 'Countries, cities, states'
```

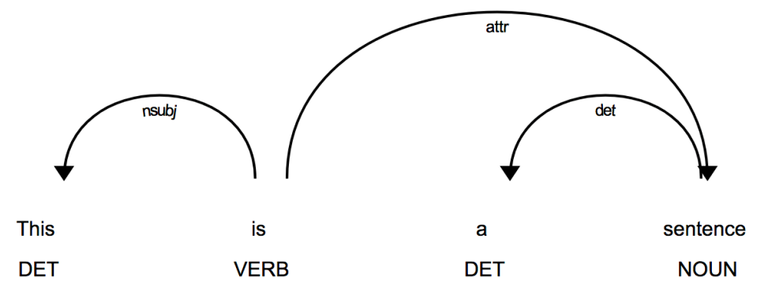
Visualizing

If you're in a Jupyter notebook, use `displacy.render`. Otherwise, use `displacy.serve` to start a web server and show the visualization in your browser.

```
from spacy import displacy
```

Visualize dependencies

```
doc = nlp("This is a sentence")
displacy.render(doc, style="dep")
```



Visualize named entities

```
doc = nlp("Larry Page founded Google")
displacy.render(doc, style="ent")
```

Larry Page PERSON founded Google ORG

Word vectors and similarity

To use word vectors, you need to install the larger models ending in `md` or `lg`, for example `en_core_web_lg`.

Comparing similarity

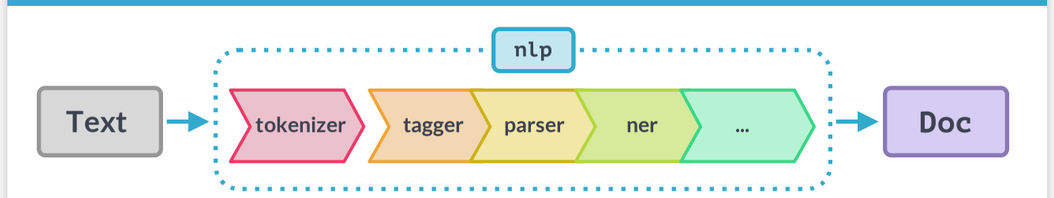
```
doc1 = nlp("I like cats")
doc2 = nlp("I like dogs")
# Compare 2 documents
doc1.similarity(doc2)
# Compare 2 tokens
doc1[2].similarity(doc2[2])
# Compare tokens and spans
doc1[0].similarity(doc2[1:3])
```

Accessing word vectors

```
# Vector as a numpy array
doc = nlp("I like cats")
# The L2 norm of the token's vector
doc[2].vector
doc[2].vector_norm
```

Pipeline components

Functions that take a `Doc` object, modify it and return it.



Pipeline information

```
nlp = spacy.load("en_core_web_sm")
nlp.pipe_names
# ['tagger', 'parser', 'ner']
nlp.pipeline
# [('tagger', <spacy.pipeline.Tagger>),
#  ('parser', <spacy.pipeline.DependencyParser>),
#  ('ner', <spacy.pipeline.EntityRecognizer>)]
```

Custom components

```
# Function that modifies the doc and returns it
def custom_component(doc):
    print("Do something to the doc here!")
    return doc

# Add the component first in the pipeline
nlp.add_pipe(custom_component, first=True)
```

Components can be added `first`, `last` (default), or `before` or `after` an existing component.

Extension attributes

Custom attributes that are registered on the global `Doc`, `Token` and `Span` classes and become available as `._`.

```
from spacy.tokens import Doc, Token, Span
doc = nlp("The sky over New York is blue")
```

Attribute extensions

WITH DEFAULT VALUE

```
# Register custom attribute on Token class
Token.set_extension("is_color", default=False)
# Overwrite extension attribute with default value
doc[6]._.is_color = True
```

Property extensions

WITH GETTER & SETTER

```
# Register custom attribute on Doc class
get_reversed = lambda doc: doc.text[::-1]
Doc.set_extension("reversed", getter=get_reversed)
# Compute value of extension attribute with getter
doc._.reversed
# 'eulb si kroY weN revo yks ehT'
```

Method extensions

CALLABLE METHOD

```
# Register custom attribute on Span class
has_label = lambda span, label: span.label_ == label
Span.set_extension("has_label", method=has_label)
# Compute value of extension attribute with method
doc[3:5].has_label("GPE")
# True
```

Rule-based matching

Using the matcher

```
# Matcher is initialized with the shared vocab
from spacy.matcher import Matcher
# Each dict represents one token and its attributes
matcher = Matcher(nlp.vocab)
# Add with ID, optional callback and pattern(s)
pattern = [{"LOWER": "new"}, {"LOWER": "york"}]
matcher.add("CITIES", None, pattern)
# Match by calling the matcher on a Doc object
doc = nlp("I live in New York")
matches = matcher(doc)
# Matches are (match_id, start, end) tuples
for match_id, start, end in matches:
    # Get the matched span by slicing the Doc
    span = doc[start:end]
    print(span.text)
# 'New York'
```

Rule-based matching

Token patterns

```
# "love cats", "loving cats", "loved cats"
pattern1 = [{"LEMMA": "love"}, {"LOWER": "cats"}]
# "10 people", "twenty people"
pattern2 = [{"LIKE_NUM": True}, {"TEXT": "people"}]
# "book", "a cat", "the sea" (noun + optional article)
pattern3 = [{"POS": "DET", "OP": "?"}, {"POS": "NOUN"}]
```

Operators and quantifiers

Can be added to a token dict as the `"OP"` key.

`!`	Negate pattern and match **exactly 0 times**.
`?`	Make pattern optional and match **0 or 1 times**.
`+`	Require pattern to match **1 or more times**.
`*`	Allow pattern to match **0 or more times**.

Glossary

Tokenization	Segmenting text into words, punctuation etc.
Lemmatization	Assigning the base forms of words, for example: "was" → "be" or "rats" → "rat".
Sentence Boundary Detection	Finding and segmenting individual sentences.
Part-of-speech (POS) Tagging	Assigning word types to tokens like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Named Entity Recognition (NER)	Labeling named "real-world" objects, like persons, companies or locations.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Statistical model	Process for making predictions based on examples.
Training	Updating a statistical model with new examples.

Python 3 Cheat Sheet

Latest version on :
https://perso.limsi.fr/pointal/python:memento

Base Types

integer, float, boolean, string, bytes

```
int 783  0 -192  0b010 0o642 0xF3
          zero    binary  octal  hexa
float 9.23 0.0  -1.7e-6
                        ×10^-6
bool True  False
str "One\nTwo"   escaped new line
'I\'m'   escaped '
"""X\tY\tZ
1\t2\t3"""   escaped tab
bytes b"toto\xfe\775"
        hexadecimal  octal
```

Multiline string:

immutables

Container Types

- **ordered sequences**, fast index access, repeatable values

```
list [1,5,9]   ["x",11,8.9]   ["mot"]   []
tuple (1,5,9)   11,"y",7.4   ("mot",)   ()
```

Non modifiable values (immutables) — *expression with only comas →* `tuple`

```
str bytes   (ordered sequences of chars / bytes)
                                        "" b""
```

- **key containers**, no *a priori* order, fast key access, each key is unique

```
dictionary dict {"key":"value"}   dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection set {"key1","key2"}   {1,9,3,0}   set()
```

keys=hashable values (base types, immutables…) — `frozenset` *immutable set* — *empty*

Identifiers

for variables, functions, modules, classes… names

`a…zA…Z_` followed by `a…zA…Z_0…9`

- diacritics allowed but should be avoided
- language keywords forbidden
- lower/UPPER case discrimination

```
☺ a toto x7 y_max BigOne
☹ 8y and for
```

Variables assignment

`=` — assignment ⇔ **binding** of a *name* with a *value*

1) evaluation of right side expression value
2) assignment in order with left side names

```
x=1.2+8+sin(y)
a=b=c=0   assignment to same value
y,z,r=9.2,-7.6,0   multiple assignments
a,b=b,a   values swap
a,*b=seq
*a,b=seq   unpacking of sequence in item and list
x+=3   increment ⇔ x=x+3
x-=2   decrement ⇔ x=x-2
x=None   « undefined » constant value
del x   remove name x
```

and `*=` `/=` `%=` …

Conversions

`type(`*expression*`)`

```
int("15") → 15
int("3f",16) → 63   can specify integer number base in 2nd parameter
int(15.56) → 15   truncate decimal part
float("-11.24e8") → -1124000000.0
round(15.56,1)→15.6   rounding to 1 decimal (0 decimal → integer number)
bool(x)  False for null x, empty container x , None or False x ; True for other x
str(x)→"…"   representation string of x for display (cf. formatting on the back)
chr(64)→'@'   ord('@')→64   code ↔ char
repr(x)→"…"   literal representation string of x
bytes([72,9,64]) → b'H\t@'
list("abc") → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
separator str and sequence of str → assembled str
   ':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str splitted on whitespaces → list of str
   "words with  spaces".split() → ['words','with','spaces']
str splitted on separator str → list of str
   "1,4,8,2".split(",") → ['1','4','8','2']
sequence of one type → list of another type (via list comprehension)
   [int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Sequence Containers Indexing

for lists, tuples, strings, bytes…

```
negative index   -5  -4  -3  -2  -1
positive index    0   1   2   3   4
lst=[10, 20, 30, 40, 50]
positive slice    0   1   2   3   4   5
negative slice   -5  -4  -3  -2  -1
```

Items count `len(lst)→5`

index from 0 (here from 0 to 4)

Individual access to **items** via `lst[`*index*`]`

```
lst[0]→10   ⇒ first one   lst[1]→20
lst[-1]→50   ⇒ last one   lst[-2]→40
```

On mutable sequences (`list`), remove with `del lst[3]` *and modify with assignment* `lst[4]=25`

Access to **sub-sequences** via `lst[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]   lst[::-2]→[50,30,10]   lst[-3:-1]→[30,40]   lst[3:]→[40,50]
lst[::2]→[10,30,50]   lst[:]→[10,20,30,40,50]   shallow copy of sequence
```

Missing slice indication → from start / up to end.
On mutable sequences (`list`), remove with `del lst[3:5]` *and modify with assignment* `lst[1:4]=[15,25]`

Boolean Logic

Comparisons : `<` `>` `<=` `>=` `==` `!=` (≤ ≥ = ≠) *(boolean results)*

`a and b` logical and *both simulta-neously*

`a or b` logical or *one or other or both*

pitfall : `and` *and* `or` *return* *value* *of* `a` *or of* `b` *(under shortcut evaluation).*
⇒ *ensure that* `a` *and* `b` *are booleans.*

`not a` logical not

`True` `False` True and False constants

Statements Blocks

```
parent statement:
    statement block 1…
    ⋮
parent statement:
    statement block2…
    ⋮
next statement after block 1
```

indentation !

configure editor to insert 4 spaces in place of an indentation tab.

Modules/Names Imports

module `truc` ⇔ *file* `truc.py`

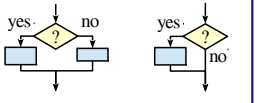
```
from monmod import nom1,nom2 as fct
    →direct access to names, renaming with as
import monmod →access via monmod.nom1 …
```

modules and packages searched in *python path* (cf `sys.path`)

Conditional Statement

statement block executed only if a condition is true

```
if logical condition:
    statements block
```



Can go with several *elif*, *elif*... and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

with a var `x`*:*
`if bool(x)==True:` ⇔ `if x:`
`if bool(x)==False:` ⇔ `if not x:`

Maths

floating numbers… approximated values

Operators: `+` `-` `*` `/` `//` `%` `**`
Priority `(…)` × ÷ ↑ ↑ a^b integer ÷ ÷ remainder

`@` → matrix × *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

usual order of operations

angles in radians

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

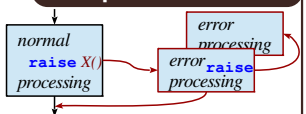
modules `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, *etc. (cf. doc)*

Exceptions on Errors

Signaling an error:
`raise` *ExcClass(…)*

Errors processing:

```
try:
    →normal procesising block
except Exception as e:
    →error processing block
```



`finally` *block for final processing in all cases.*

Conditional Loop Statement

*statements block executed **as long as** condition is true*

```
while logical condition:
    statements block
```


```
s = 0          # initializations before the loop
i = 1          # condition with a least one variable value (here i)
while i <= 100:
    s = s + i**2
    i = i + 1  # make condition variable change !
print("sum:",s)
```

Algo: $s = \sum_{i=1}^{i=100} i^2$

beware of infinite loops!

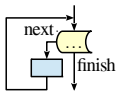
Loop Control

`break` immediate exit
`continue` next iteration
`else` block for ***normal*** loop exit.

Iterative Loop Statement

*statements block executed **for each** item of a container or iterator*

```
for var in sequence:
    statements block
```



Go over sequence's **values**

```
s = "Some text"   # initializations before the loop
cnt = 0
for c in s:       # loop variable, assignment managed by for statement
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

Algo: count number of `e` in the string.

loop on dict/set ⇔ loop on keys sequences
use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

good habit : don't modify loop variable

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

Display

```
print("v=",3,"cm :",x,",",y+4)
```

items to display : literal values, variables, expressions

`print` options:

- `sep=" "` items separator, default space
- `end="\n"` end of print, default new line
- `file=sys.stdout` print to file, default standard output

Input

```
s = input("Instructions:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Generic Operations on Containers

`len(c)` → items count
`min(c)` `max(c)` `sum(c)`
`sorted(c)` → `list` sorted *copy*
`val in c` → boolean, membership operator `in` (absence `not in`)
`enumerate(c)` → *iterator* on (index, value)
`zip(c1,c2…)` → *iterator* on tuples containing c_i items at same index
`all(c)` → `True` if **all** `c` items evaluated to true, else `False`
`any(c)` → `True` if **at least one** item of `c` evaluated true, else `False`

*Note: For dictionaries and sets, these operations use **keys**.*

*Specific to **ordered sequences containers** (lists, tuples, strings, bytes…)*

`reversed(c)` → *inversed iterator* `c*5`→ duplicate `c+c2`→ concatenate
`c.index(val)` → *position* `c.count(val)` → *events count*

```
import copy
copy.copy(c) → shallow copy of container
copy.deepcopy(c) → deep copy of container
```

Operations on Lists

modify original list

`lst.append(val)`	add item at end
`lst.extend(seq)`	add sequence of items at end
`lst.insert(idx,val)`	insert item at index
`lst.remove(val)`	remove first item with value *val*
`lst.pop([idx])` →*value*	remove & return item at index *idx* (default last)
`lst.sort()` `lst.reverse()`	sort / reverse liste *in place*

Operations on Dictionaries

`d[key]=value` `d.clear()`
`d[key]` → *value* `del d[key]`
`d.update(d2)` update/add associations
`d.keys()` `d.values()` `d.items()` →*iterable views on keys/values/associations*
`d.pop(key[,default])` → *value*
`d.popitem()` → *(key,value)*
`d.get(key[,default])` → *value*
`d.setdefault(key[,default])` →*value*

Operations on Sets

Operators:
`|` → union (vertical bar char)
`&` → intersection
`-` `^` → difference/symmetric diff.
`<` `<=` `>` `>=` → inclusion relations
Operators also exist as methods.

`s.update(s2)` `s.copy()`
`s.add(key)` `s.remove(key)`
`s.discard(key)` `s.clear()`
`s.pop()`

Files

storing data on disk, and reading it back

```
f = open("file.txt","w",encoding="utf8")
```

- file **variable** for operations
- **name** of file on disk (+path…) — cf. modules `os`, `os.path` and `pathlib`
- opening **mode**: `'r'` read, `'w'` write, `'a'` append, …`'+'` `'x'` `'b'` `'t'`
- **encoding** of chars for *text files*: `utf8` `ascii` `latin1` …

writing

`f.write("coucou")`
`f.writelines(list of lines)`

reading

read empty string if end of file
`f.read([n])` → *next chars* — *if n not specified, read up to end !*
`f.readlines([n])` → `list` *of next lines*
`f.readline()` → *next line*

*text mode `t` by default (read/write `str`), possible binary mode `b` (read/write `bytes`). **Convert from/to required type** !*

`f.close()` dont forget to **close the file** after use !

`f.flush()` write cache
`f.truncate([size])` resize

reading/writing progress sequentially in the file, modifiable with:
`f.tell()` →*position*
`f.seek(position[,origin])`

Very common: opening with a guarded block (automatic closing) and reading loop on lines of a text file:

```
with open(…) as f:
    for line in f :
        # processing of line
```

Integer Sequences

`range([start,] end [,step])`

start default 0, *end* not included in sequence, *step* signed, default 1

`range(5)` → 0 1 2 3 4
`range(3,8)` → 3 4 5 6 7
`range(len(seq))` → *sequence of index of values in seq*
`range(2,12,3)` → 2 5 8 11
`range(20,5,-5)` → 20 15 10

range provides an immutable sequence of int constructed as needed

Function Definition

function name (identifier)
named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`
**args variable positional arguments (→`tuple`), default values,*
***kwargs variable named arguments (→`dict`)*

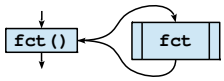
Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value — *one argument per parameter*

this is the use of function name *with parentheses* which does the call

*Advanced: *sequence **dict*



Operations on Strings

`s.startswith(prefix[,start[,end]])` `s.endswith(suffix[,start[,end]])` `s.strip([chars])`
`s.count(sub[,start[,end]])` `s.partition(sep)` → *(before,sep,after)*
`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`
`s.is…()` *tests on chars categories (ex. `s.isalpha()`)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center([width,fill])`
`s.ljust([width,fill])` `s.rjust([width,fill])` `s.zfill([width])`
`s.encode(encoding)` `s.split([sep])` `s.join(seq)`

Formatting

formating directives — values to format

```
"modele{} {} {}".format(x,y,r) → str
```

`"{`*selection* `:` *formatting* `!` *conversion*`}"`

- **Selection** :
  `2`
  `nom`
  `0.nom`
  `4[key]`
  `0[2]`

Examples:

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="I'm")
→'"I\'m"'
```

- **Formatting** :

fill char *alignment* *sign* *mini width*`.`*precision~maxwidth* *type*

`<` `>` `^` `=` `+` `-` *space* — `0` at start for filling with 0

integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…
float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default),
string: `s` … `%` percent

- **Conversion** : `s` (readable text) or `r` (literal representation)

Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Lists

A list stores a series of items in a particular order. You access items using an index, or within a loop.

Make a list

```
bikes = ['trek', 'redline', 'giant']
```

Get the first item in a list

```
first_bike = bikes[0]
```

Get the last item in a list

```
last_bike = bikes[-1]
```

Looping through a list

```
for bike in bikes:
    print(bike)
```

Adding items to a list

```
bikes = []
bikes.append('trek')
bikes.append('redline')
bikes.append('giant')
```

Making numerical lists

```
squares = []
for x in range(1, 11):
    squares.append(x**2)
```

Lists (cont.)

List comprehensions

```
squares = [x**2 for x in range(1, 11)]
```

Slicing a list

```
finishers = ['sam', 'bob', 'ada', 'bea']
first_two = finishers[:2]
```

Copying a list

```
copy_of_bikes = bikes[:]
```

Tuples

Tuples are similar to lists, but the items in a tuple can't be modified.

Making a tuple

```
dimensions = (1920, 1080)
```

If statements

If statements are used to test for particular conditions and respond appropriately.

Conditional tests

```
equals            x == 42
not equal         x != 42
greater than      x > 42
  or equal to     x >= 42
less than         x < 42
  or equal to     x <= 42
```

Conditional test with lists

```
'trek' in bikes
'surly' not in bikes
```

Assigning boolean values

```
game_active = True
can_edit = False
```

A simple if test

```
if age >= 18:
    print("You can vote!")
```

If-elif-else statements

```
if age < 4:
    ticket_price = 0
elif age < 18:
    ticket_price = 10
else:
    ticket_price = 15
```

Dictionaries

Dictionaries store connections between pieces of information. Each item in a dictionary is a key-value pair.

A simple dictionary

```
alien = {'color': 'green', 'points': 5}
```

Accessing a value

```
print("The alien's color is " + alien['color'])
```

Adding a new key-value pair

```
alien['x_position'] = 0
```

Looping through all key-value pairs

```
fav_numbers = {'eric': 17, 'ever': 4}
for name, number in fav_numbers.items():
    print(name + ' loves ' + str(number))
```

Looping through all keys

```
fav_numbers = {'eric': 17, 'ever': 4}
for name in fav_numbers.keys():
    print(name + ' loves a number')
```

Looping through all the values

```
fav_numbers = {'eric': 17, 'ever': 4}
for number in fav_numbers.values():
    print(str(number) + ' is a favorite')
```

User input

Your programs can prompt the user for input. All input is stored as a string.

Prompting for a value

```
name = input("What's your name? ")
print("Hello, " + name + "!")
```

Prompting for numerical input

```
age = input("How old are you? ")
age = int(age)

pi = input("What's the value of pi? ")
pi = float(pi)
```

While loops

A while loop repeats a block of code as long as a certain condition is true.

A simple while loop

```
current_value = 1
while current_value <= 5:
    print(current_value)
    current_value += 1
```

Letting the user choose when to quit

```
msg = ''
while msg != 'quit':
    msg = input("What's your message? ")
    print(msg)
```

Functions

Functions are named blocks of code, designed to do one specific job. Information passed to a function is called an argument, and information received by a function is called a parameter.

A simple function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

Passing an argument

```
def greet_user(username):
    """Display a personalized greeting."""
    print("Hello, " + username + "!")

greet_user('jesse')
```

Default values for parameters

```
def make_pizza(topping='bacon'):
    """Make a single-topping pizza."""
    print("Have a " + topping + " pizza!")

make_pizza()
make_pizza('pepperoni')
```

Returning a value

```
def add_numbers(x, y):
    """Add two numbers and return the sum."""
    return x + y

sum = add_numbers(3, 5)
print(sum)
```

Classes

A class defines the behavior of an object and the kind of information an object can store. The information in a class is stored in attributes, and functions that belong to a class are called methods. A child class inherits the attributes and methods from its parent class.

Creating a dog class

```
class Dog():
    """Represent a dog."""

    def __init__(self, name):
        """Initialize dog object."""
        self.name = name

    def sit(self):
        """Simulate sitting."""
        print(self.name + " is sitting.")

my_dog = Dog('Peso')

print(my_dog.name + " is a great dog!")
my_dog.sit()
```

Inheritance

```
class SARDog(Dog):
    """Represent a search dog."""

    def __init__(self, name):
        """Initialize the sardog."""
        super().__init__(name)

    def search(self):
        """Simulate searching."""
        print(self.name + " is searching.")

my_dog = SARDog('Willie')

print(my_dog.name + " is a search dog.")
my_dog.sit()
my_dog.search()
```

Infinite Skills

If you had infinite programming skills, what would you build?

As you're learning to program, it's helpful to think about the real-world projects you'd like to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. If you haven't done so already, take a few minutes and describe three projects you'd like to create.

Working with files

Your programs can read from files and write to files. Files are opened in read mode ('r') by default, but can also be opened in write mode ('w') and append mode ('a').

Reading a file and storing its lines

```
filename = 'siddhartha.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line)
```

Writing to a file

```
filename = 'journal.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

Appending to a file

```
filename = 'journal.txt'
with open(filename, 'a') as file_object:
    file_object.write("\nI love making games.")
```

Exceptions

Exceptions help you respond appropriately to errors that are likely to occur. You place code that might cause an error in the try block. Code that should run in response to an error goes in the except block. Code that should run only if the try block was successful goes in the else block.

Catching an exception

```
prompt = "How many tickets do you need? "
num_tickets = input(prompt)

try:
    num_tickets = int(num_tickets)
except ValueError:
    print("Please try again.")
else:
    print("Your tickets are printing.")
```

Zen of Python

Simple is better than complex

If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet - Lists

What are lists?

A list stores a series of items in a particular order. Lists allow you to store sets of information in one place, whether you have just a few items or millions of items. Lists are one of Python's most powerful features readily accessible to new programmers, and they tie together many important concepts in programming.

Defining a list

Use square brackets to define a list, and use commas to separate individual items in the list. Use plural names for lists, to make your code easier to read.

Making a list

```
users = ['val', 'bob', 'mia', 'ron', 'ned']
```

Accessing elements

Individual elements in a list are accessed according to their position, called the index. The index of the first element is 0, the index of the second element is 1, and so forth. Negative indices refer to items at the end of the list. To get a particular element, write the name of the list and then the index of the element in square brackets.

Getting the first element

```
first_user = users[0]
```

Getting the second element

```
second_user = users[1]
```

Getting the last element

```
newest_user = users[-1]
```

Modifying individual items

Once you've defined a list, you can change individual elements in the list. You do this by referring to the index of the item you want to modify.

Changing an element

```
users[0] = 'valerie'
users[-2] = 'ronald'
```

Adding elements

You can add elements to the end of a list, or you can insert them wherever you like in a list.

Adding an element to the end of the list

```
users.append('amy')
```

Starting with an empty list

```
users = []
users.append('val')
users.append('bob')
users.append('mia')
```

Inserting elements at a particular position

```
users.insert(0, 'joe')
users.insert(3, 'bea')
```

Removing elements

You can remove elements by their position in a list, or by the value of the item. If you remove an item by its value, Python removes only the first item that has that value.

Deleting an element by its position

```
del users[-1]
```

Removing an item by its value

```
users.remove('mia')
```

Popping elements

If you want to work with an element that you're removing from the list, you can "pop" the element. If you think of the list as a stack of items, pop() takes an item off the top of the stack. By default pop() returns the last element in the list, but you can also pop elements from any position in the list.

Pop the last item from a list

```
most_recent_user = users.pop()
print(most_recent_user)
```

Pop the first item in a list

```
first_user = users.pop(0)
print(first_user)
```

List length

The len() function returns the number of items in a list.

Find the length of a list

```
num_users = len(users)
print("We have " + str(num_users) + " users.")
```

Sorting a list

The sort() method changes the order of a list permanently. The sorted() function returns a copy of the list, leaving the original list unchanged. You can sort the items in a list in alphabetical order, or reverse alphabetical order. You can also reverse the original order of the list. Keep in mind that lowercase and uppercase letters may affect the sort order.

Sorting a list permanently

```
users.sort()
```

Sorting a list permanently in reverse alphabetical order

```
users.sort(reverse=True)
```

Sorting a list temporarily

```
print(sorted(users))
print(sorted(users, reverse=True))
```

Reversing the order of a list

```
users.reverse()
```

Looping through a list

Lists can contain millions of items, so Python provides an efficient way to loop through all the items in a list. When you set up a loop, Python pulls each item from the list one at a time and stores it in a temporary variable, which you provide a name for. This name should be the singular version of the list name.

The indented block of code makes up the body of the loop, where you can work with each individual item. Any lines that are not indented run after the loop is completed.

Printing all items in a list

```
for user in users:
    print(user)
```

Printing a message for each item, and a separate message afterwards

```
for user in users:
    print("Welcome, " + user + "!")

print("Welcome, we're glad to see you all!")
```

The range() function

You can use the range() function to work with a set of numbers efficiently. The range() function starts at 0 by default, and stops one number below the number passed to it. You can use the list() function to efficiently generate a large list of numbers.

Printing the numbers 0 to 1000

```
for number in range(1001):
    print(number)
```

Printing the numbers 1 to 1000

```
for number in range(1, 1001):
    print(number)
```

Making a list of numbers from 1 to a million

```
numbers = list(range(1, 1000001))
```

Simple statistics

There are a number of simple statistics you can run on a list containing numerical data.

Finding the minimum value in a list

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
youngest = min(ages)
```

Finding the maximum value

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
oldest = max(ages)
```

Finding the sum of all values

```
ages = [93, 99, 66, 17, 85, 1, 35, 82, 2, 77]
total_years = sum(ages)
```

Slicing a list

You can work with any set of elements from a list. A portion of a list is called a slice. To slice a list start with the index of the first item you want, then add a colon and the index after the last item you want. Leave off the first index to start at the beginning of the list, and leave off the last index to slice through the end of the list.

Getting the first three items

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
first_three = finishers[:3]
```

Getting the middle three items

```
middle_three = finishers[1:4]
```

Getting the last three items

```
last_three = finishers[-3:]
```

Copying a list

To copy a list make a slice that starts at the first item and ends at the last item. If you try to copy a list without using this approach, whatever you do to the copied list will affect the original list as well.

Making a copy of a list

```
finishers = ['kai', 'abe', 'ada', 'gus', 'zoe']
copy_of_finishers = finishers[:]
```

List comprehensions

You can use a loop to generate a list based on a range of numbers or on another list. This is a common operation, so Python offers a more efficient way to do it. List comprehensions may look complicated at first; if so, use the for loop approach until you're ready to start using comprehensions.

To write a comprehension, define an expression for the values you want to store in the list. Then write a for loop to generate input values needed to make the list.

Using a loop to generate a list of square numbers

```
squares = []
for x in range(1, 11):
    square = x**2
    squares.append(square)
```

Using a comprehension to generate a list of square numbers

```
squares = [x**2 for x in range(1, 11)]
```

Using a loop to convert a list of names to upper case

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = []
for name in names:
    upper_names.append(name.upper())
```

Using a comprehension to convert a list of names to upper case

```
names = ['kai', 'abe', 'ada', 'gus', 'zoe']

upper_names = [name.upper() for name in names]
```

Styling your code

Readability counts

- Use four spaces per indentation level.
- Keep your lines to 79 characters or fewer.
- Use single blank lines to group parts of your program visually.

Tuples

A tuple is like a list, except you can't change the values in a tuple once it's defined. Tuples are good for storing information that shouldn't be changed throughout the life of a program. Tuples are designated by parentheses instead of square brackets. (You can overwrite an entire tuple, but you can't change the individual elements in a tuple.)

Defining a tuple

```
dimensions = (800, 600)
```

Looping through a tuple

```
for dimension in dimensions:
    print(dimension)
```

Overwriting a tuple

```
dimensions = (800, 600)
print(dimensions)

dimensions = (1200, 900)
```

Visualizing your code

When you're first learning about data structures such as lists, it helps to visualize how Python is working with the information in your program. pythontutor.com is a great tool for seeing how Python keeps track of the information in a list. Try running the following code on pythontutor.com, and then run your own code.

Build a list and print the items in the list

```
dogs = []
dogs.append('willie')
dogs.append('hootz')
dogs.append('peso')
dogs.append('goblin')

for dog in dogs:
    print("Hello " + dog + "!")
print("I love these dogs!")

print("\nThese were my first two dogs:")
old_dogs = dogs[:2]
for old_dog in old_dogs:
    print(old_dog)

del dogs[0]
dogs.remove('peso')
print(dogs)
```

Beginner's Python Cheat Sheet — Dictionaries

What are dictionaries?

Python's dictionaries allow you to connect pieces of related information. Each piece of information in a dictionary is stored as a key-value pair. When you provide a key, Python returns the value associated with that key. You can loop through all the key-value pairs, all the keys, or all the values.

Defining a dictionary

Use curly braces to define a dictionary. Use colons to connect keys and values, and use commas to separate individual key-value pairs.

Making a dictionary

```
alien_0 = {'color': 'green', 'points': 5}
```

Accessing values

To access the value associated with an individual key give the name of the dictionary and then place the key in a set of square brackets. If the key you're asking for is not in the dictionary, an error will occur.

You can also use the get() method, which returns None instead of an error if the key doesn't exist. You can also specify a default value to use if the key is not in the dictionary.

Getting the value associated with a key

```
alien_0 = {'color': 'green', 'points': 5}

print(alien_0['color'])
print(alien_0['points'])
```

Getting the value with get()

```
alien_0 = {'color': 'green'}

alien_color = alien_0.get('color')
alien_points = alien_0.get('points', 0)

print(alien_color)
print(alien_points)
```

Adding new key-value pairs

You can store as many key-value pairs as you want in a dictionary, until your computer runs out of memory. To add a new key-value pair to an existing dictionary give the name of the dictionary and the new key in square brackets, and set it equal to the new value.

This also allows you to start with an empty dictionary and add key-value pairs as they become relevant.

Adding a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}

alien_0['x'] = 0
alien_0['y'] = 25
alien_0['speed'] = 1.5
```

Adding to an empty dictionary

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
```

Modifying values

You can modify the value associated with any key in a dictionary. To do so give the name of the dictionary and enclose the key in square brackets, then provide the new value for that key.

Modifying values in a dictionary

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

# Change the alien's color and point value.
alien_0['color'] = 'yellow'
alien_0['points'] = 10
print(alien_0)
```

Removing key-value pairs

You can remove any key-value pair you want from a dictionary. To do so use the del keyword and the dictionary name, followed by the key in square brackets. This will delete the key and its associated value.

Deleting a key-value pair

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)

del alien_0['points']
print(alien_0)
```

Visualizing dictionaries

Try running some of these examples on pythontutor.com.

Looping through a dictionary

You can loop through a dictionary in three ways: you can loop through all the key-value pairs, all the keys, or all the values.

A dictionary only tracks the connections between keys and values; it doesn't track the order of items in the dictionary. If you want to process the information in order, you can sort the keys in your loop.

Looping through all key-value pairs

```
# Store people's favorite languages.
fav_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
    }

# Show each person's favorite language.
for name, language in fav_languages.items():
    print(name + ": " + language)
```

Looping through all the keys

```
# Show everyone who's taken the survey.
for name in fav_languages.keys():
    print(name)
```

Looping through all the values

```
# Show all the languages that have been chosen.
for language in fav_languages.values():
    print(language)
```

Looping through all the keys in order

```
# Show each person's favorite language,
#   in order by the person's name.
for name in sorted(fav_languages.keys()):
    print(name + ": " + language)
```

Dictionary length

You can find the number of key-value pairs in a dictionary.

Finding a dictionary's length

```
num_responses = len(fav_languages)
```

Nesting — A list of dictionaries

It's sometimes useful to store a set of dictionaries in a list; this is called nesting.

Storing dictionaries in a list

```
# Start with an empty list.
users = []

# Make a new user, and add them to the list.
new_user = {
    'last': 'fermi',
    'first': 'enrico',
    'username': 'efermi',
    }
users.append(new_user)

# Make another new user, and add them as well.
new_user = {
    'last': 'curie',
    'first': 'marie',
    'username': 'mcurie',
    }
users.append(new_user)

# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

You can also define a list of dictionaries directly, without using append():

```
# Define a list of users, where each user
#   is represented by a dictionary.
users = [
    {
        'last': 'fermi',
        'first': 'enrico',
        'username': 'efermi',
    },
    {
        'last': 'curie',
        'first': 'marie',
        'username': 'mcurie',
    },
]

# Show all information about each user.
for user_dict in users:
    for k, v in user_dict.items():
        print(k + ": " + v)
    print("\n")
```

Nesting — Lists in a dictionary

Storing a list inside a dictionary alows you to associate more than one value with each key.

Storing lists in a dictionary

```
# Store multiple languages for each person.
fav_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
}

# Show all responses for each person.
for name, langs in fav_languages.items():
    print(name + ": ")
    for lang in langs:
        print("- " + lang)
```

Nesting — A dictionary of dictionaries

You can store a dictionary inside another dictionary. In this case each value associated with a key is itself a dictionary.

Storing dictionaries in a dictionary

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
        },
    }

for username, user_dict in users.items():
    print("\nUsername: " + username)
    full_name = user_dict['first'] + " "
    full_name += user_dict['last']
    location = user_dict['location']

    print("\tFull name: " + full_name.title())
    print("\tLocation: " + location.title())
```

Levels of nesting

Nesting is extremely useful in certain situations. However, be aware of making your code overly complex. If you're nesting items much deeper than what you see here there are probably simpler ways of managing your data, such as using classes.

Using an OrderedDict

Standard Python dictionaries don't keep track of the order in which keys and values are added; they only preserve the association between each key and its value. If you want to preserve the order in which keys and values are added, use an OrderedDict.

Preserving the order of keys and values

```
from collections import OrderedDict

# Store each person's languages, keeping
#  track of who respoded first.
fav_languages = OrderedDict()

fav_languages['jen'] = ['python', 'ruby']
fav_languages['sarah'] = ['c']
fav_languages['edward'] = ['ruby', 'go']
fav_languages['phil'] = ['python', 'haskell']

# Display the results, in the same order they
#  were entered.
for name, langs in fav_languages.items():
    print(name + ":")
    for lang in langs:
        print("- " + lang)
```

Generating a million dictionaries

You can use a loop to generate a large number of dictionaries efficiently, if all the dictionaries start out with similar data.

A million aliens

```
aliens = []

# Make a million green aliens, worth 5 points
#  each. Have them all start in one row.
for alien_num in range(1000000):
    new_alien = {}
    new_alien['color'] = 'green'
    new_alien['points'] = 5
    new_alien['x'] = 20 * alien_num
    new_alien['y'] = 0
    aliens.append(new_alien)

# Prove the list contains a million aliens.
num_aliens = len(aliens)

print("Number of aliens created:")
print(num_aliens)
```

Beginner's Python Cheat Sheet — If Statements and While Loops

What are if statements? What are while loops?

If statements allow you to examine the current state of a program and respond appropriately to that state. You can write a simple if statement that checks one condition, or you can create a complex series of if statements that idenitfy the exact conditions you're looking for.

While loops run as long as certain conditions remain true. You can use while loops to let your programs run as long as your users want them to.

Conditional Tests

A conditional test is an expression that can be evaluated as True or False. Python uses the values True and False to decide whether the code in an if statement should be executed.

Checking for equality

A single equal sign assigns a value to a variable. A double equal sign (==) checks whether two values are equal.

```
>>> car = 'bmw'
>>> car == 'bmw'
True
>>> car = 'audi'
>>> car == 'bmw'
False
```

Ignoring case when making a comparison

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Checking for inequality

```
>>> topping = 'mushrooms'
>>> topping != 'anchovies'
True
```

Numerical comparisons

Testing numerical values is similar to testing string values.

Testing equality and inequality

```
>>> age = 18
>>> age == 18
True
>>> age != 18
False
```

Comparison operators

```
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >= 21
False
```

Checking multiple conditions

You can check multiple conditions at the same time. The and operator returns True if all the conditions listed are True. The or operator returns True if any condition is True.

Using and to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 and age_1 >= 21
False
>>> age_1 = 23
>>> age_0 >= 21 and age_1 >= 21
True
```

Using or to check multiple conditions

```
>>> age_0 = 22
>>> age_1 = 18
>>> age_0 >= 21 or age_1 >= 21
True
>>> age_0 = 18
>>> age_0 >= 21 or age_1 >= 21
False
```

Boolean values

A boolean value is either True or False. Variables with boolean values are often used to keep track of certain conditions within a program.

Simple boolean values

```
game_active = True
can_edit = False
```

If statements

Several kinds of if statements exist. Your choice of which to use depends on the number of conditions you need to test. You can have as many elif blocks as you need, and the else block is always optional.

Simple if statement

```
age = 19

if age >= 18:
    print("You're old enough to vote!")
```

If-else statements

```
age = 17

if age >= 18:
    print("You're old enough to vote!")
else:
    print("You can't vote yet.")
```

The if-elif-else chain

```
age = 12

if age < 4:
   price = 0
elif age < 18:
    price = 5
else:
    price = 10

print("Your cost is $" + str(price) + ".")
```

Conditional tests with lists

You can easily test whether a certain value is in a list. You can also test whether a list is empty before trying to loop through the list.

Testing if a value is in a list

```
>>> players = ['al', 'bea', 'cyn', 'dale']
>>> 'al' in players
True
>>> 'eric' in players
False
```

Conditional tests with lists (cont.)

Testing if a value is not in a list

```
banned_users = ['ann', 'chad', 'dee']
user = 'erin'

if user not in banned_users:
    print("You can play!")
```

Checking if a list is empty

```
players = []

if players:
    for player in players:
        print("Player: " + player.title())
else:
    print("We have no players yet!")
```

Accepting input

You can allow your users to enter input using the input() statement. In Python 3, all input is stored as a string.

Simple input

```
name = input("What's your name? ")
print("Hello, " + name + ".")
```

Accepting numerical input

```
age = input("How old are you? ")
age = int(age)

if age >= 18:
    print("\nYou can vote!")
else:
    print("\nYou can't vote yet.")
```

Accepting input in Python 2.7

Use raw_input() in Python 2.7. This function interprets all input as a string, just as input() does in Python 3.

```
name = raw_input("What's your name? ")
print("Hello, " + name + ".")
```

While loops

A while loop repeats a block of code as long as a condition is True.

Counting to 5

```
current_number = 1

while current_number <= 5:
    print(current_number)
    current_number += 1
```

While loops (cont.)

Letting the user choose when to quit

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

message = ""
while message != 'quit':
    message = input(prompt)

    if message != 'quit':
        print(message)
```

Using a flag

```
prompt = "\nTell me something, and I'll "
prompt += "repeat it back to you."
prompt += "\nEnter 'quit' to end the program. "

active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Using break to exit a loop

```
prompt = "\nWhat cities have you visited?"
prompt += "\nEnter 'quit' when you're done. "

while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print("I've been to " + city + "!")
```

Accepting input with Sublime Text

Sublime Text doesn't run programs that prompt the user for input. You can use Sublime Text to write programs that prompt for input, but you'll need to run these programs from a terminal.

Breaking out of loops

You can use the break statement and the continue statement with any of Python's loops. For example you can use break to quit a for loop that's working through a list or a dictionary. You can use continue to skip over certain items when looping through a list or dictionary as well.

While loops (cont.)

Using continue in a loop

```
banned_users = ['eve', 'fred', 'gary', 'helen']

prompt = "\nAdd a player to your team."
prompt += "\nEnter 'quit' when you're done. "

players = []
while True:
    player = input(prompt)
    if player == 'quit':
        break
    elif player in banned_users:
        print(player + " is banned!")
        continue
    else:
        players.append(player)

print("\nYour team:")
for player in players:
    print(player)
```

Avoiding infinite loops

Every while loop needs a way to stop running so it won't continue to run forever. If there's no way for the condition to become False, the loop will never stop running.

An infinite loop

```
while True:
    name = input("\nWho are you? ")
    print("Nice to meet you, " + name + "!")
```

Removing all instances of a value from a list

The remove() method removes a specific value from a list, but it only removes the first instance of the value you provide. You can use a while loop to remove all instances of a particular value.

Removing all cats from a list of pets

```
pets = ['dog', 'cat', 'dog', 'fish', 'cat',
        'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Functions

What are functions?

Functions are named blocks of code designed to do one specific job. Functions allow you to write code once that can then be run whenever you need to accomplish the same task. Functions can take in the information they need, and return the information they generate. Using functions effectively makes your programs easier to write, read, test, and fix.

Defining a function

The first line of a function is its definition, marked by the keyword def. The name of the function is followed by a set of parentheses and a colon. A docstring, in triple quotes, describes what the function does. The body of a function is indented one level.

To call a function, give the name of the function followed by a set of parentheses.

Making a function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

Passing information to a function

Information that's passed to a function is called an argument; information that's received by a function is called a parameter. Arguments are included in parentheses after the function's name, and parameters are listed in parentheses in the function's definition.

Passing a single argument

```
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username + "!")

greet_user('jesse')
greet_user('diana')
greet_user('brandon')
```

Positional and keyword arguments

The two main kinds of arguments are positional and keyword arguments. When you use positional arguments Python matches the first argument in the function call with the first parameter in the function definition, and so forth.

With keyword arguments, you specify which parameter each argument should be assigned to in the function call. When you use keyword arguments, the order of the arguments doesn't matter.

Using positional arguments

```
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
```

Using keyword arguments

```
def describe_pet(animal, name):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet(animal='hamster', name='harry')
describe_pet(name='willie', animal='dog')
```

Default values

You can provide a default value for a parameter. When function calls omit this argument the default value will be used. Parameters with default values must be listed after parameters without default values in the function's definition so positional arguments can still work correctly.

Using a default value

```
def describe_pet(name, animal='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    print("Its name is " + name + ".")

describe_pet('harry', 'hamster')
describe_pet('willie')
```

Using None to make an argument optional

```
def describe_pet(animal, name=None):
    """Display information about a pet."""
    print("\nI have a " + animal + ".")
    if name:
        print("Its name is " + name + ".")

describe_pet('hamster', 'harry')
describe_pet('snake')
```

Return values

A function can return a value or a set of values. When a function returns a value, the calling line must provide a variable in which to store the return value. A function stops running when it reaches a return statement.

Returning a single value

```
def get_full_name(first, last):
    """Return a neatly formatted full name."""
    full_name = first + ' ' + last
    return full_name.title()

musician = get_full_name('jimi', 'hendrix')
print(musician)
```

Returning a dictionary

```
def build_person(first, last):
    """Return a dictionary of information
    about a person.
    """
    person = {'first': first, 'last': last}
    return person

musician = build_person('jimi', 'hendrix')
print(musician)
```

Returning a dictionary with optional values

```
def build_person(first, last, age=None):
    """Return a dictionary of information
    about a person.
    """
    person = {'first': first, 'last': last}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', 27)
print(musician)

musician = build_person('janis', 'joplin')
print(musician)
```

Visualizing functions

Try running some of these examples on pythontutor.com.

Passing a list to a function

You can pass a list as an argument to a function, and the function can work with the values in the list. Any changes the function makes to the list will affect the original list. You can prevent a function from modifying a list by passing a copy of the list as an argument.

Passing a list as an argument

```
def greet_users(names):
    """Print a simple greeting to everyone."""
    for name in names:
        msg = "Hello, " + name + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

Allowing a function to modify a list

The following example sends a list of models to a function for printing. The original list is emptied, and the second list is filled.

```
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)

# Store some unprinted designs,
#  and print each of them.
unprinted = ['phone case', 'pendant', 'ring']
printed = []
print_models(unprinted, printed)

print("\nUnprinted:", unprinted)
print("Printed:", printed)
```

Preventing a function from modifying a list

The following example is the same as the previous one, except the original list is unchanged after calling print_models().

```
def print_models(unprinted, printed):
    """3d print a set of models."""
    while unprinted:
        current_model = unprinted.pop()
        print("Printing " + current_model)
        printed.append(current_model)

# Store some unprinted designs,
#  and print each of them.
original = ['phone case', 'pendant', 'ring']
printed = []

print_models(original[:], printed)
print("\nOriginal:", original)
print("Printed:", printed)
```

Passing an arbitrary number of arguments

*Sometimes you won't know how many arguments a function will need to accept. Python allows you to collect an arbitrary number of arguments into one parameter using the * operator. A parameter that accepts an arbitrary number of arguments must come last in the function definition.*

*The ** operator allows a parameter to collect an arbitrary number of keyword arguments.*

Collecting an arbitrary number of arguments

```
def make_pizza(size, *toppings):
    """Make a pizza."""
    print("\nMaking a " + size + " pizza.")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)

# Make three pizzas with different toppings.
make_pizza('small', 'pepperoni')
make_pizza('large', 'bacon bits', 'pineapple')
make_pizza('medium', 'mushrooms', 'peppers',
        'onions', 'extra cheese')
```

Collecting an arbitrary number of keyword arguments

```
def build_profile(first, last, **user_info):
    """Build a user's profile dictionary."""
    # Build a dict with the required keys.
    profile = {'first': first, 'last': last}

    # Add any other keys and values.
    for key, value in user_info.items():
        profile[key] = value

    return profile

# Create two users with different kinds
#   of information.
user_0 = build_profile('albert', 'einstein',
        location='princeton')
user_1 = build_profile('marie', 'curie',
        location='paris', field='chemistry')

print(user_0)
print(user_1)
```

What's the best way to structure a function?

As you can see there are many ways to write and call a function. When you're starting out, aim for something that simply works. As you gain experience you'll develop an understanding of the more subtle advantages of different structures such as positional and keyword arguments, and the various approaches to importing functions. For now if your functions do what you need them to, you're doing well.

Modules

You can store your functions in a separate file called a module, and then import the functions you need into the file containing your main program. This allows for cleaner program files. (Make sure your module is stored in the same directory as your main program.)

Storing a function in a module

File: pizza.py

```
def make_pizza(size, *toppings):
    """Make a pizza."""
    print("\nMaking a " + size + " pizza.")
    print("Toppings:")
    for topping in toppings:
        print("- " + topping)
```

Importing an entire module

File: making_pizzas.py
Every function in the module is available in the program file.

```
import pizza

pizza.make_pizza('medium', 'pepperoni')
pizza.make_pizza('small', 'bacon', 'pineapple')
```

Importing a specific function

Only the imported functions are available in the program file.

```
from pizza import make_pizza

make_pizza('medium', 'pepperoni')
make_pizza('small', 'bacon', 'pineapple')
```

Giving a module an alias

```
import pizza as p

p.make_pizza('medium', 'pepperoni')
p.make_pizza('small', 'bacon', 'pineapple')
```

Giving a function an alias

```
from pizza import make_pizza as mp

mp('medium', 'pepperoni')
mp('small', 'bacon', 'pineapple')
```

Importing all functions from a module

Don't do this, but recognize it when you see it in others' code. It can result in naming conflicts, which can cause errors.

```
from pizza import *

make_pizza('medium', 'pepperoni')
make_pizza('small', 'bacon', 'pineapple')
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet - Classes

What are classes?

Classes are the foundation of object-oriented programming. Classes represent real-world things you want to model in your programs: for example dogs, cars, and robots. You use a class to make objects, which are specific instances of dogs, cars, and robots. A class defines the general behavior that a whole category of objects can have, and the information that can be associated with those objects.

Classes can inherit from each other – you can write a class that extends the functionality of an existing class. This allows you to code efficiently for a wide variety of situations.

Creating and using a class

Consider how we might model a car. What information would we associate with a car, and what behavior would it have? The information is stored in variables called attributes, and the behavior is represented by functions. Functions that are part of a class are called methods.

The Car class

```
class Car():
    """A simple attempt to model a car."""

    def __init__(self, make, model, year):
        """Initialize car attributes."""
        self.make = make
        self.model = model
        self.year = year

        # Fuel capacity and level in gallons.
        self.fuel_capacity = 15
        self.fuel_level = 0

    def fill_tank(self):
        """Fill gas tank to capacity."""
        self.fuel_level = self.fuel_capacity
        print("Fuel tank is full.")

    def drive(self):
        """Simulate driving."""
        print("The car is moving.")
```

Creating and using a class (cont.)

Creating an object from a class

```
my_car = Car('audi', 'a4', 2016)
```

Accessing attribute values

```
print(my_car.make)
print(my_car.model)
print(my_car.year)
```

Calling methods

```
my_car.fill_tank()
my_car.drive()
```

Creating multiple objects

```
my_car = Car('audi', 'a4', 2016)
my_old_car = Car('subaru', 'outback', 2013)
my_truck = Car('toyota', 'tacoma', 2010)
```

Modifying attributes

You can modify an attribute's value directly, or you can write methods that manage updating values more carefully.

Modifying an attribute directly

```
my_new_car = Car('audi', 'a4', 2016)
my_new_car.fuel_level = 5
```

Writing a method to update an attribute's value

```
def update_fuel_level(self, new_level):
    """Update the fuel level."""
    if new_level <= self.fuel_capacity:
        self.fuel_level = new_level
    else:
        print("The tank can't hold that much!")
```

Writing a method to increment an attribute's value

```
def add_fuel(self, amount):
    """Add fuel to the tank."""
    if (self.fuel_level + amount
            <= self.fuel_capacity):
        self.fuel_level += amount
        print("Added fuel.")
    else:
        print("The tank won't hold that much.")
```

Naming conventions

In Python class names are written in CamelCase and object names are written in lowercase with underscores. Modules that contain classes should still be named in lowercase with underscores.

Class inheritance

If the class you're writing is a specialized version of another class, you can use inheritance. When one class inherits from another, it automatically takes on all the attributes and methods of the parent class. The child class is free to introduce new attributes and methods, and override attributes and methods of the parent class.

To inherit from another class include the name of the parent class in parentheses when defining the new class.

The __init__() method for a child class

```
class ElectricCar(Car):
    """A simple model of an electric car."""

    def __init__(self, make, model, year):
        """Initialize an electric car."""
        super().__init__(make, model, year)

        # Attributes specific to electric cars.
        # Battery capacity in kWh.
        self.battery_size = 70
        # Charge level in %.
        self.charge_level = 0
```

Adding new methods to the child class

```
class ElectricCar(Car):
    --snip--
    def charge(self):
        """Fully charge the vehicle."""
        self.charge_level = 100
        print("The vehicle is fully charged.")
```

Using child methods and parent methods

```
my_ecar = ElectricCar('tesla', 'model s', 2016)

my_ecar.charge()
my_ecar.drive()
```

Finding your workflow

There are many ways to model real world objects and situations in code, and sometimes that variety can feel overwhelming. Pick an approach and try it – if your first attempt doesn't work, try a different approach.

Class inheritance (cont.)

Overriding parent methods

```
class ElectricCar(Car):
    --snip--
    def fill_tank(self):
        """Display an error message."""
        print("This car has no fuel tank!")
```

Instances as attributes

A class can have objects as attributes. This allows classes to work together to model complex situations.

A Battery class

```
class Battery():
    """A battery for an electric car."""

    def __init__(self, size=70):
        """Initialize battery attributes."""
        # Capacity in kWh, charge level in %.
        self.size = size
        self.charge_level = 0

    def get_range(self):
        """Return the battery's range."""
        if self.size == 70:
            return 240
        elif self.size == 85:
            return 270
```

Using an instance as an attribute

```
class ElectricCar(Car):
    --snip--

    def __init__(self, make, model, year):
        """Initialize an electric car."""
        super().__init__(make, model, year)

        # Attribute specific to electric cars.
        self.battery = Battery()

    def charge(self):
        """Fully charge the vehicle."""
        self.battery.charge_level = 100
        print("The vehicle is fully charged.")
```

Using the instance

```
my_ecar = ElectricCar('tesla', 'model x', 2016)

my_ecar.charge()
print(my_ecar.battery.get_range())
my_ecar.drive()
```

Importing classes

Class files can get long as you add detailed information and functionality. To help keep your program files uncluttered, you can store your classes in modules and import the classes you need into your main program.

Storing classes in a file

car.py

```
"""Represent gas and electric cars."""

class Car():
    """A simple attempt to model a car."""
    --snip—

class Battery():
    """A battery for an electric car."""
    --snip--

class ElectricCar(Car):
    """A simple model of an electric car."""
    --snip--
```

Importing individual classes from a module

my_cars.py

```
from car import Car, ElectricCar

my_beetle = Car('volkswagen', 'beetle', 2016)
my_beetle.fill_tank()
my_beetle.drive()

my_tesla = ElectricCar('tesla', 'model s',
2016)
my_tesla.charge()
my_tesla.drive()
```

Importing an entire module

```
import car

my_beetle = car.Car(
        'volkswagen', 'beetle', 2016)
my_beetle.fill_tank()
my_beetle.drive()

my_tesla = car.ElectricCar(
        'tesla', 'model s', 2016)
my_tesla.charge()
my_tesla.drive()
```

Importing all classes from a module

(Don't do this, but recognize it when you see it.)

```
from car import *

my_beetle = Car('volkswagen', 'beetle', 2016)
```

Classes in Python 2.7

Classes should inherit from object

```
class ClassName(object):
```

The Car class in Python 2.7

```
class Car(object):
```

Child class __init__() method is different

```
class ChildClassName(ParentClass):
    def __init__(self):
        super(ClassName, self).__init__()
```

The ElectricCar class in Python 2.7

```
class ElectricCar(Car):
    def __init__(self, make, model, year):
        super(ElectricCar, self).__init__(
                make, model, year)
```

Storing objects in a list

A list can hold as many items as you want, so you can make a large number of objects from a class and store them in a list.

Here's an example showing how to make a fleet of rental cars, and make sure all the cars are ready to drive.

A fleet of rental cars

```
from car import Car, ElectricCar

# Make lists to hold a fleet of cars.
gas_fleet = []
electric_fleet = []

# Make 500 gas cars and 250 electric cars.
for _ in range(500):
    car = Car('ford', 'focus', 2016)
    gas_fleet.append(car)
for _ in range(250):
    ecar = ElectricCar('nissan', 'leaf', 2016)
    electric_fleet.append(ecar)

# Fill the gas cars, and charge electric cars.
for car in gas_fleet:
    car.fill_tank()
for ecar in electric_fleet:
    ecar.charge()

print("Gas cars:", len(gas_fleet))
print("Electric cars:", len(electric_fleet))
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Files and Exceptions

What are files? What are exceptions?

Your programs can read information in from files, and they can write data to files. Reading from files allows you to work with a wide variety of information; writing to files allows users to pick up where they left off the next time they run your program. You can write text to files, and you can store Python structures such as lists in data files.

Exceptions are special objects that help your programs respond to errors in appropriate ways. For example if your program tries to open a file that doesn't exist, you can use exceptions to display an informative error message instead of having the program crash.

Reading from a file

To read from a file your program needs to open the file and then read the contents of the file. You can read the entire contents of the file at once, or read the file line by line. The with statement makes sure the file is closed properly when the program has finished accessing the file.

Reading an entire file at once

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    contents = f_obj.read()

print(contents)
```

Reading line by line

Each line that's read from the file has a newline character at the end of the line, and the print function adds its own newline character. The rstrip() method gets rid of the the extra blank lines this would result in when printing to the terminal.

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    for line in f_obj:
        print(line.rstrip())
```

Reading from a file (cont.)

Storing the lines in a list

```
filename = 'siddhartha.txt'

with open(filename) as f_obj:
    lines = f_obj.readlines()

for line in lines:
    print(line.rstrip())
```

Writing to a file

Passing the 'w' argument to open() tells Python you want to write to the file. Be careful; this will erase the contents of the file if it already exists. Passing the 'a' argument tells Python you want to append to the end of an existing file.

Writing to an empty file

```
filename = 'programming.txt'

with open(filename, 'w') as f:
    f.write("I love programming!")
```

Writing multiple lines to an empty file

```
filename = 'programming.txt'

with open(filename, 'w') as f:
    f.write("I love programming!\n")
    f.write("I love creating new games.\n")
```

Appending to a file

```
filename = 'programming.txt'

with open(filename, 'a') as f:
    f.write("I also love working with data.\n")
    f.write("I love making apps as well.\n")
```

File paths

When Python runs the open() function, it looks for the file in the same directory where the program that's being excuted is stored. You can open a file from a subfolder using a relative path. You can also use an absolute path to open any file on your system.

Opening a file from a subfolder

```
f_path = "text_files/alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()

for line in lines:
    print(line.rstrip())
```

File paths (cont.)

Opening a file using an absolute path

```
f_path = "/home/ehmatthes/books/alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()
```

Opening a file on Windows

Windows will sometimes interpret forward slashes incorrectly. If you run into this, use backslashes in your file paths.

```
f_path = "C:\Users\ehmatthes\books\alice.txt"

with open(f_path) as f_obj:
    lines = f_obj.readlines()
```

The try-except block

When you think an error may occur, you can write a try-except block to handle the exception that might be raised. The try block tells Python to try running some code, and the except block tells Python what to do if the code results in a particular kind of error.

Handling the ZeroDivisionError exception

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Handling the FileNotFoundError exception

```
f_name = 'siddhartha.txt'

try:
    with open(f_name) as f_obj:
        lines = f_obj.readlines()
except FileNotFoundError:
    msg = "Can't find file {0}.".format(f_name)
    print(msg)
```

Knowing which exception to handle

It can be hard to know what kind of exception to handle when writing code. Try writing your code without a try block, and make it generate an error. The traceback will tell you what kind of exception your program needs to handle.

The else block

The try block should only contain code that may cause an error. Any code that depends on the try block running successfully should be placed in the else block.

Using an else block

```
print("Enter two numbers. I'll divide them.")

x = input("First number: ")
y = input("Second number: ")

try:
    result = int(x) / int(y)
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print(result)
```

Preventing crashes from user input

Without the except block in the following example, the program would crash if the user tries to divide by zero. As written, it will handle the error gracefully and keep running.

```
"""A simple calculator for division only."""

print("Enter two numbers. I'll divide them.")
print("Enter 'q' to quit.")

while True:
    x = input("\nFirst number: ")
    if x == 'q':
        break
    y = input("Second number: ")
    if y == 'q':
        break

    try:
        result = int(x) / int(y)
    except ZeroDivisionError:
        print("You can't divide by zero!")
    else:
        print(result)
```

Deciding which errors to report

Well-written, properly tested code is not very prone to internal errors such as syntax or logical errors. But every time your program depends on something external such as user input or the existence of a file, there's a possibility of an exception being raised.

It's up to you how to communicate errors to your users. Sometimes users need to know if a file is missing; sometimes it's better to handle the error silently. A little experience will help you know how much to report.

Failing silently

Sometimes you want your program to just continue running when it encounters an error, without reporting the error to the user. Using the pass statement in an else block allows you to do this.

Using the pass statement in an else block

```
f_names = ['alice.txt', 'siddhartha.txt',
        'moby_dick.txt', 'little_women.txt']

for f_name in f_names:
    # Report the length of each file found.
    try:
        with open(f_name) as f_obj:
            lines = f_obj.readlines()
    except FileNotFoundError:
        # Just move on to the next file.
        pass
    else:
        num_lines = len(lines)
        msg = "{0} has {1} lines.".format(
            f_name, num_lines)
        print(msg)
```

Avoid bare except blocks

Exception-handling code should catch specific exceptions that you expect to happen during your program's execution. A bare except block will catch all exceptions, including keyboard interrupts and system exits you might need when forcing a program to close.

If you want to use a try block and you're not sure which exception to catch, use Exception. It will catch most exceptions, but still allow you to interrupt programs intentionally.

Don't use bare except blocks

```
try:
    # Do something
except:
    pass
```

Use Exception instead

```
try:
    # Do something
except Exception:
    pass
```

Printing the exception

```
try:
    # Do something
except Exception as e:
    print(e, type(e))
```

Storing data with json

The json module allows you to dump simple Python data structures into a file, and load the data from that file the next time the program runs. The JSON data format is not specific to Python, so you can share this kind of data with people who work in other languages as well.

Knowing how to manage exceptions is important when working with stored data. You'll usually want to make sure the data you're trying to load exists before working with it.

Using json.dump() to store data

```
"""Store some numbers."""

import json

numbers = [2, 3, 5, 7, 11, 13]

filename = 'numbers.json'
with open(filename, 'w') as f_obj:
    json.dump(numbers, f_obj)
```

Using json.load() to read data

```
"""Load some previously stored numbers."""

import json

filename = 'numbers.json'
with open(filename) as f_obj:
    numbers = json.load(f_obj)

print(numbers)
```

Making sure the stored data exists

```
import json

f_name = 'numbers.json'

try:
    with open(f_name) as f_obj:
        numbers = json.load(f_obj)
except FileNotFoundError:
    msg = "Can't find {0}.".format(f_name)
    print(msg)
else:
    print(numbers)
```

Practice with exceptions

Take a program you've already written that prompts for user input, and add some error-handling code to the program.

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Testing Your Code

Why test your code?

When you write a function or a class, you can also write tests for that code. Testing proves that your code works as it's supposed to in the situations it's designed to handle, and also when people use your programs in unexpected ways. Writing tests gives you confidence that your code will work correctly as more people begin to use your programs. You can also add new features to your programs and know that you haven't broken existing behavior.

A unit test verifies that one specific aspect of your code works as it's supposed to. A test case is a collection of unit tests which verify your code's behavior in a wide variety of situations.

Testing a function: A passing test

Python's unittest module provides tools for testing your code. To try it out, we'll create a function that returns a full name. We'll use the function in a regular program, and then build a test case for the function.

A function to test

Save this as full_names.py

```
def get_full_name(first, last):
    """Return a full name."""
    full_name = "{0} {1}".format(first, last)
    return full_name.title()
```

Using the function

Save this as names.py

```
from full_names import get_full_name

janis = get_full_name('janis', 'joplin')
print(janis)

bob = get_full_name('bob', 'dylan')
print(bob)
```

Testing a function (cont.)

Building a testcase with one unit test

To build a test case, make a class that inherits from `unittest.TestCase` *and write methods that begin with* `test_`. *Save this as test_full_names.py*

```
import unittest
from full_names import get_full_name

class NamesTestCase(unittest.TestCase):
    """Tests for names.py."""

    def test_first_last(self):
        """Test names like Janis Joplin."""
        full_name = get_full_name('janis',
                'joplin')
        self.assertEqual(full_name,
                'Janis Joplin')

unittest.main()
```

Running the test

Python reports on each unit test in the test case. The dot reports a single passing test. Python informs us that it ran 1 test in less than 0.001 seconds, and the OK lets us know that all unit tests in the test case passed.

```
.
--------------------------------------
Ran 1 test in 0.000s

OK
```

Testing a function: A failing test

Failing tests are important; they tell you that a change in the code has affected existing behavior. When a test fails, you need to modify the code so the existing behavior still works.

Modifying the function

We'll modify `get_full_name()` *so it handles middle names, but we'll do it in a way that breaks existing behavior.*

```
def get_full_name(first, middle, last):
    """Return a full name."""
    full_name = "{0} {1} {2}".format(first,
                                     middle, last)
    return full_name.title()
```

Using the function

```
from full_names import get_full_name

john = get_full_name('john', 'lee', 'hooker')
print(john)

david = get_full_name('david', 'lee', 'roth')
print(david)
```

A failing test (cont.)

Running the test

When you change your code, it's important to run your existing tests. This will tell you whether the changes you made affected existing behavior.

```
E
======================================================
ERROR: test_first_last (__main__.NamesTestCase)
Test names like Janis Joplin.
------------------------------------------------------
Traceback (most recent call last):
  File "test_full_names.py", line 10,
    in test_first_last
    'joplin')
TypeError: get_full_name() missing 1 required
  positional argument: 'last'

------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)
```

Fixing the code

When a test fails, the code needs to be modified until the test passes again. (Don't make the mistake of rewriting your tests to fit your new code.) Here we can make the middle name optional.

```
def get_full_name(first, last, middle=''):
    """Return a full name."""
    if middle:
        full_name = "{0} {1} {2}".format(first,
                                         middle, last)
    else:
        full_name = "{0} {1}".format(first,
                                     last)
    return full_name.title()
```

Running the test

Now the test should pass again, which means our original functionality is still intact.

```
.
--------------------------------------
Ran 1 test in 0.000s

OK
```

Adding new tests

You can add as many unit tests to a test case as you need. To write a new test, add a new method to your test case class.

Testing middle names

We've shown that `get_full_name()` works for first and last names. Let's test that it works for middle names as well.

```
import unittest
from full_names import get_full_name

class NamesTestCase(unittest.TestCase):
    """Tests for names.py."""

    def test_first_last(self):
        """Test names like Janis Joplin."""
        full_name = get_full_name('janis',
                'joplin')
        self.assertEqual(full_name,
                'Janis Joplin')

    def test_middle(self):
        """Test names like David Lee Roth."""
        full_name = get_full_name('david',
                'roth', 'lee')
        self.assertEqual(full_name,
                'David Lee Roth')

unittest.main()
```

Running the tests

The two dots represent two passing tests.

```
..
--------------------------------------
Ran 2 tests in 0.000s

OK
```

A variety of assert methods

Python provides a number of assert methods you can use to test your code.

Verify that a==b, or a != b

```
assertEqual(a, b)
assertNotEqual(a, b)
```

Verify that x is True, or x is False

```
assertTrue(x)
assertFalse(x)
```

Verify an item is in a list, or not in a list

```
assertIn(item, list)
assertNotIn(item, list)
```

Testing a class

Testing a class is similar to testing a function, since you'll mostly be testing your methods.

A class to test

Save as accountant.py

```
class Accountant():
    """Manage a bank account."""

    def __init__(self, balance=0):
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount
```

Building a testcase

For the first test, we'll make sure we can start out with different initial balances. Save this as test_accountant.py.

```
import unittest
from accountant import Accountant

class TestAccountant(unittest.TestCase):
    """Tests for the class Accountant."""

    def test_initial_balance(self):
        # Default balance should be 0.
        acc = Accountant()
        self.assertEqual(acc.balance, 0)

        # Test non-default balance.
        acc = Accountant(100)
        self.assertEqual(acc.balance, 100)

unittest.main()
```

Running the test

```
.
--------------------------------------
Ran 1 test in 0.000s

OK
```

When is it okay to modify tests?

In general you shouldn't modify a test once it's written. When a test fails it usually means new code you've written has broken existing functionality, and you need to modify the new code until all existing tests pass.

If your original requirements have changed, it may be appropriate to modify some tests. This usually happens in the early stages of a project when desired behavior is still being sorted out.

The setUp() method

When testing a class, you usually have to make an instance of the class. The `setUp()` method is run before every test. Any instances you make in `setUp()` are available in every test you write.

Using setUp() to support multiple tests

The instance `self.acc` can be used in each new test.

```
import unittest
from accountant import Accountant

class TestAccountant(unittest.TestCase):
    """Tests for the class Accountant."""

    def setUp(self):
        self.acc = Accountant()

    def test_initial_balance(self):
        # Default balance should be 0.
        self.assertEqual(self.acc.balance, 0)

        # Test non-default balance.
        acc = Accountant(100)
        self.assertEqual(acc.balance, 100)

    def test_deposit(self):
        # Test single deposit.
        self.acc.deposit(100)
        self.assertEqual(self.acc.balance, 100)

        # Test multiple deposits.
        self.acc.deposit(100)
        self.acc.deposit(100)
        self.assertEqual(self.acc.balance, 300)

    def test_withdrawal(self):
        # Test single withdrawal.
        self.acc.deposit(1000)
        self.acc.withdraw(100)
        self.assertEqual(self.acc.balance, 900)

unittest.main()
```

Running the tests

```
...
--------------------------------------
Ran 3 tests in 0.001s

OK
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet - Pygame

What is Pygame?

Pygame is a framework for making games using Python. Making games is fun, and it's a great way to expand your programming skills and knowledge. Pygame takes care of many of the lower-level tasks in building games, which lets you focus on the aspects of your game that make it interesting.

Installing Pygame

Pygame runs on all systems, but setup is slightly different on each OS. The instructions here assume you're using Python 3, and provide a minimal installation of Pygame. If these instructions don't work for your system, see the more detailed notes at http://ehmatthes.github.io/pcc/.

Pygame on Linux

```
$ sudo apt-get install python3-dev mercurial
    libsdl-image1.2-dev libsdl2-dev
    libsdl-ttf2.0-dev
$ pip install --user
    hg+http://bitbucket.org/pygame/pygame
```

Pygame on OS X

This assumes you've used Homebrew to install Python 3.

```
$ brew install hg sdl sdl_image sdl_ttf
$ pip install --user
    hg+http://bitbucket.org/pygame/pygame
```

Pygame on Windows

Find an installer at https://bitbucket.org/pygame/pygame/downloads/ or http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame that matches your version of Python. Run the installer file if it's a .exe or .msi file. If it's a .whl file, use pip to install Pygame:

```
> python -m pip install --user
    pygame-1.9.2a0-cp35-none-win32.whl
```

Testing your installation

To test your installation, open a terminal session and try to import Pygame. If you don't get any error messages, your installation was successful.

```
$ python
>>> import pygame
>>>
```

Starting a game

The following code sets up an empty game window, and starts an event loop and a loop that continually refreshes the screen.

An empty game window

```
import sys
import pygame as pg

def run_game():
    # Initialize and set up screen.
    pg.init()
    screen = pg.display.set_mode((1200, 800))
    pg.display.set_caption("Alien Invasion")

    # Start main loop.
    while True:
        # Start event loop.
        for event in pg.event.get():
            if event.type == pg.QUIT:
                sys.exit()

        # Refresh screen.
        pg.display.flip()

run_game()
```

Setting a custom window size

The display.set_mode() function accepts a tuple that defines the screen size.

```
screen_dim = (1200, 800)
screen = pg.display.set_mode(screen_dim)
```

Setting a custom background color

Colors are defined as a tuple of red, green, and blue values. Each value ranges from 0-255.

```
bg_color = (230, 230, 230)
screen.fill(bg_color)
```

Pygame rect objects

Many objects in a game can be treated as simple rectangles, rather than their actual shape. This simplifies code without noticeably affecting game play. Pygame has a rect object that makes it easy to work with game objects.

Getting the screen rect object

We already have a screen object; we can easily access the rect object associated with the screen.

```
screen_rect = screen.get_rect()
```

Finding the center of the screen

Rect objects have a center attribute which stores the center point.

```
screen_center = screen_rect.center
```

Pygame rect objects (cont.)

Useful rect attributes

Once you have a rect object, there are a number of attributes that are useful when positioning objects and detecting relative positions of objects. (You can find more attributes in the Pygame documentation.)

```
# Individual x and y values:
screen_rect.left, screen_rect.right
screen_rect.top, screen_rect.bottom
screen_rect.centerx, screen_rect.centery
screen_rect.width, screen_rect.height

# Tuples
screen_rect.center
screen_rect.size
```

Creating a rect object

You can create a rect object from scratch. For example a small rect object that's filled in can represent a bullet in a game. The Rect() class takes the coordinates of the upper left corner, and the width and height of the rect. The draw.rect() function takes a screen object, a color, and a rect. This function fills the given rect with the given color.

```
bullet_rect = pg.Rect(100, 100, 3, 15)
color = (100, 100, 100)
pg.draw.rect(screen, color, bullet_rect)
```

Working with images

Many objects in a game are images that are moved around the screen. It's easiest to use bitmap (.bmp) image files, but you can also configure your system to work with jpg, png, and gif files as well.

Loading an image

```
ship = pg.image.load('images/ship.bmp')
```

Getting the rect object from an image

```
ship_rect = ship.get_rect()
```

Positioning an image

With rects, it's easy to position an image wherever you want on the screen, or in relation to another object. The following code positions a ship object at the bottom center of the screen.

```
ship_rect.midbottom = screen_rect.midbottom
```

Working with images (cont.)

Drawing an image to the screen

Once an image is loaded and positioned, you can draw it to the screen with the blit() method. The blit() method acts on the screen object, and takes the image object and image rect as arguments.

```
# Draw ship to screen.
screen.blit(ship, ship_rect)
```

The blitme() method

Game objects such as ships are often written as classes. Then a blitme() method is usually defined, which draws the object to the screen.

```
def blitme(self):
    """Draw ship at current location."""
    self.screen.blit(self.image, self.rect)
```

Responding to keyboard input

Pygame watches for events such as key presses and mouse actions. You can detect any event you care about in the event loop, and respond with any action that's appropriate for your game.

Responding to key presses

Pygame's main event loop registers a KEYDOWN event any time a key is pressed. When this happens, you can check for specific keys.

```
for event in pg.event.get():
    if event.type == pg.KEYDOWN:
        if event.key == pg.K_RIGHT:
            ship_rect.x += 1
        elif event.key == pg.K_LEFT:
            ship_rect.x -= 1
        elif event.key == pg.K_SPACE:
            ship.fire_bullet()
        elif event.key == pg.K_q:
            sys.exit()
```

Responding to released keys

When the user releases a key, a KEYUP event is triggered.

```
if event.type == pg.KEYUP:
    if event.key == pg.K_RIGHT:
        ship.moving_right = False
```

Pygame documentation

The Pygame documentation is really helpful when building your own games. The home page for the Pygame project is at http://pygame.org/, and the home page for the documentation is at http://pygame.org/docs/.

The most useful part of the documentation are the pages about specific parts of Pygame, such as the Rect() class and the sprite module. You can find a list of these elements at the top of the help pages.

Responding to mouse events

Pygame's event loop registers an event any time the mouse moves, or a mouse button is pressed or released.

Responding to the mouse button

```
for event in pg.event.get():
    if event.type == pg.MOUSEBUTTONDOWN:
        ship.fire_bullet()
```

Finding the mouse position

The mouse position is returned as a tuple.

```
mouse_pos = pg.mouse.get_pos()
```

Clicking a button

You might want to know if the cursor is over an object such as a button. The rect.collidepoint() method returns true when a point is inside a rect object.

```
if button_rect.collidepoint(mouse_pos):
    start_game()
```

Hiding the mouse

```
pg.mouse.set_visible(False)
```

Pygame groups

Pygame has a Group class which makes working with a group of similar objects easier. A group is like a list, with some extra functionality that's helpful when building games.

Making and filling a group

An object that will be placed in a group must inherit from Sprite.

```
from pygame.sprite import Sprite, Group

def Bullet(Sprite):
    ...
    def draw_bullet(self):
        ...
    def update(self):
        ...

bullets = Group()

new_bullet = Bullet()
bullets.add(new_bullet)
```

Looping through the items in a group

The sprites() method returns all the members of a group.

```
for bullet in bullets.sprites():
    bullet.draw_bullet()
```

Calling update() on a group

Calling update() on a group automatically calls update() on each member of the group.

```
bullets.update()
```

Pygame groups (cont.)

Removing an item from a group

It's important to delete elements that will never appear again in the game, so you don't waste memory and resources.

```
bullets.remove(bullet)
```

Detecting collisions

You can detect when a single object collides with any member of a group. You can also detect when any member of one group collides with a member of another group.

Collisions between a single object and a group

The spritecollideany() function takes an object and a group, and returns True if the object overlaps with any member of the group.

```
if pg.sprite.spritecollideany(ship, aliens):
    ships_left -= 1
```

Collisions between two groups

The sprite.groupcollide() function takes two groups, and two booleans. The function returns a dictionary containing information about the members that have collided. The booleans tell Pygame whether to delete the members of either group that have collided.

```
collisions = pg.sprite.groupcollide(
        bullets, aliens, True, True)

score += len(collisions) * alien_point_value
```

Rendering text

You can use text for a variety of purposes in a game. For example you can share information with players, and you can display a score.

Displaying a message

The following code defines a message, then a color for the text and the background color for the message. A font is defined using the default system font, with a font size of 48. The font.render() function is used to create an image of the message, and we get the rect object associated with the image. We then center the image on the screen and display it.

```
msg = "Play again?"
msg_color = (100, 100, 100)
bg_color = (230, 230, 230)

f = pg.font.SysFont(None, 48)
msg_image = f.render(msg, True, msg_color,
        bg_color)
msg_image_rect = msg_image.get_rect()
msg_image_rect.center = screen_rect.center
screen.blit(msg_image, msg_image_rect)
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — matplotlib

What is matplotlib?

Data visualization involves exploring data through visual representations. The matplotlib package helps you make visually appealing representations of the data you're working with. matplotlib is extremely flexible; these examples will help you get started with a few simple visualizations.

Installing matplotlib

matplotlib runs on all systems, but setup is slightly different depending on your OS. If the minimal instructions here don't work for you, see the more detailed instructions at http://ehmatthes.github.io/pcc/. You should also consider installing the Anaconda distrubution of Python from https://continuum.io/downloads/, which includes matplotlib.

matplotlib on Linux

```
$ sudo apt-get install python3-matplotlib
```

matplotlib on OS X

Start a terminal session and enter `import matplotlib` *to see if it's already installed on your system. If not, try this command:*

```
$ pip install --user matplotlib
```

matplotlib on Windows

You first need to install Visual Studio, which you can do from https://dev.windows.com/. The Community edition is free. Then go to https://pypi.python.org/pypi/matplotlib/ or http://www.lfd.uic.edu/~gohlke/pythonlibs/#matplotlib and download an appropriate installer file.

Line graphs and scatter plots

Making a line graph

```
import matplotlib.pyplot as plt

x_values = [0, 1, 2, 3, 4, 5]
squares = [0, 1, 4, 9, 16, 25]
plt.plot(x_values, squares)
plt.show()
```

Line graphs and scatter plots (cont.)

Making a scatter plot

The scatter() function takes a list of x values and a list of y values, and a variety of optional arguments. The `s=10` *argument controls the size of each point.*

```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]

plt.scatter(x_values, squares, s=10)
plt.show()
```

Customizing plots

Plots can be customized in a wide variety of ways. Just about any element of a plot can be customized.

Adding titles and labels, and scaling axes

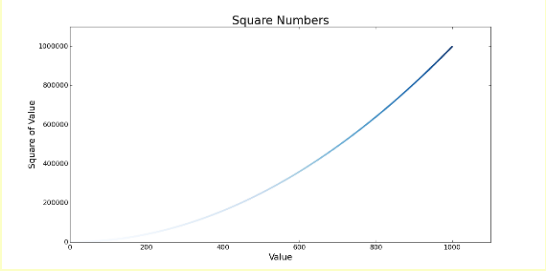
```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]
plt.scatter(x_values, squares, s=10)

plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=18)
plt.ylabel("Square of Value", fontsize=18)
plt.tick_params(axis='both', which='major',
                labelsize=14)
plt.axis([0, 1100, 0, 1100000])

plt.show()
```

Using a colormap

A colormap varies the point colors from one shade to another, based on a certain value for each point. The value used to determine the color of each point is passed to the `c` *argument, and the* `cmap` *argument specifies which colormap to use.*

The `edgecolor='none'` *argument removes the black outline from each point.*

```
plt.scatter(x_values, squares, c=squares,
        cmap=plt.cm.Blues, edgecolor='none',
        s=10)
```



Customizing plots (cont.)

Emphasizing points

You can plot as much data as you want on one plot. Here we re-plot the first and last points larger to emphasize them.

```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]
plt.scatter(x_values, squares, c=squares,
        cmap=plt.cm.Blues, edgecolor='none',
        s=10)

plt.scatter(x_values[0], squares[0], c='green',
        edgecolor='none', s=100)
plt.scatter(x_values[-1], squares[-1], c='red',
        edgecolor='none', s=100)

plt.title("Square Numbers", fontsize=24)
--snip--
```

Removing axes

You can customize or remove axes entirely. Here's how to access each axis, and hide it.

```
plt.axes().get_xaxis().set_visible(False)
plt.axes().get_yaxis().set_visible(False)
```

Setting a custom figure size

You can make your plot as big or small as you want. Before plotting your data, add the following code. The `dpi` *argument is optional; if you don't know your system's resolution you can omit the argument and adjust the* `figsize` *argument accordingly.*

```
plt.figure(dpi=128, figsize=(10, 6))
```

Saving a plot

The matplotlib viewer has an interactive save button, but you can also save your visualizations programmatically. To do so, replace `plt.show()` *with* `plt.savefig()`*. The* `bbox_inches='tight'` *argument trims extra whitespace from the plot.*

```
plt.savefig('squares.png', bbox_inches='tight')
```

Online resources

The matplotlib gallery and documentation are at http://matplotlib.org/. Be sure to visit the examples, gallery, and pyplot links.

Multiple plots

You can make as many plots as you want on one figure. When you make multiple plots, you can emphasize relationships in the data. For example you can fill the space between two sets of data.

Plotting two sets of data

Here we use `plt.scatter()` *twice to plot square numbers and cubes on the same figure.*

```
import matplotlib.pyplot as plt

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

plt.scatter(x_values, squares, c='blue',
        edgecolor='none', s=20)
plt.scatter(x_values, cubes, c='red',
        edgecolor='none', s=20)

plt.axis([0, 11, 0, 1100])
plt.show()
```

Filling the space between data sets

The `fill_between()` *method fills the space between two data sets. It takes a series of x-values and two series of y-values. It also takes a* `facecolor` *to use for the fill, and an optional* `alpha` *argument that controls the color's transparency.*

```
plt.fill_between(x_values, cubes, squares,
        facecolor='blue', alpha=0.25)
```

Working with dates and times

Many interesting data sets have a date or time as the x-value. Python's datetime module helps you work with this kind of data.

Generating the current date

The `datetime.now()` *function returns a datetime object representing the current date and time.*

```
from datetime import datetime as dt

today = dt.now()
date_string = dt.strftime(today, '%m/%d/%Y')
print(date_string)
```

Generating a specific date

You can also generate a datetime object for any date and time you want. The positional order of arguments is year, month, and day. The hour, minute, second, and microsecond arguments are optional.

```
from datetime import datetime as dt

new_years = dt(2017, 1, 1)
fall_equinox = dt(year=2016, month=9, day=22)
```

Working with dates and times (cont.)

Datetime formatting arguments

The `strftime()` *function generates a formatted string from a datetime object, and the* `strptime()` *function genereates a datetime object from a string. The following codes let you work with dates exactly as you need to.*

```
%A    Weekday name, such as Monday
%B    Month name, such as January
%m    Month, as a number (01 to 12)
%d    Day of the month, as a number (01 to 31)
%Y    Four-digit year, such as 2016
%y    Two-digit year, such as 16
%H    Hour, in 24-hour format (00 to 23)
%I    Hour, in 12-hour format (01 to 12)
%p    AM or PM
%M    Minutes (00 to 59)
%S    Seconds (00 to 61)
```

Converting a string to a datetime object

```
new_years = dt.strptime('1/1/2017', '%m/%d/%Y')
```

Converting a datetime object to a string

```
ny_string = dt.strftime(new_years, '%B %d, %Y')
print(ny_string)
```

Plotting high temperatures

The following code creates a list of dates and a corresponding list of high temperatures. It then plots the high temperatures, with the date labels displayed in a specific format.

```
from datetime import datetime as dt

import matplotlib.pyplot as plt
from matplotlib import dates as mdates

dates = [
    dt(2016, 6, 21), dt(2016, 6, 22),
    dt(2016, 6, 23), dt(2016, 6, 24),
    ]

highs = [57, 68, 64, 59]

fig = plt.figure(dpi=128, figsize=(10,6))
plt.plot(dates, highs, c='red')
plt.title("Daily High Temps", fontsize=24)
plt.ylabel("Temp (F)", fontsize=16)

x_axis = plt.axes().get_xaxis()
x_axis.set_major_formatter(
        mdates.DateFormatter('%B %d %Y')
        )
fig.autofmt_xdate()

plt.show()
```

Multiple plots in one figure

You can include as many individual graphs in one figure as you want. This is useful, for example, when comparing related datasets.

Sharing an x-axis

The following code plots a set of squares and a set of cubes on two separate graphs that share a common x-axis.

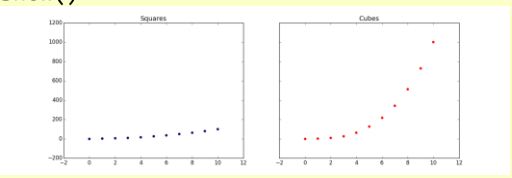
The `plt.subplots()` *function returns a figure object and a tuple of axes. Each set of axes corresponds to a separate plot in the figure. The first two arguments control the number of rows and columns generated in the figure.*

```
import matplotlib.pyplot as plt

x_vals = list(range(11))
squares = [x**2 for x in x_vals]
cubes = [x**3 for x in x_vals]

fig, axarr = plt.subplots(2, 1, sharex=True)

axarr[0].scatter(x_vals, squares)
axarr[0].set_title('Squares')

axarr[1].scatter(x_vals, cubes, c='red')
axarr[1].set_title('Cubes')

plt.show()
```

Sharing a y-axis

To share a y-axis, we use the `sharey=True` *argument.*

```
import matplotlib.pyplot as plt

x_vals = list(range(11))
squares = [x**2 for x in x_vals]
cubes = [x**3 for x in x_vals]

fig, axarr = plt.subplots(1, 2, sharey=True)

axarr[0].scatter(x_vals, squares)
axarr[0].set_title('Squares')

axarr[1].scatter(x_vals, cubes, c='red')
axarr[1].set_title('Cubes')

plt.show()
```



More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Pygal

What is Pygal?

Data visualization involves exploring data through visual representations. Pygal helps you make visually appealing representations of the data you're working with. Pygal is particularly well suited for visualizations that will be presented online, because it supports interactive elements.

Installing Pygal

Pygal can be installed using pip.

Pygal on Linux and OS X

```
$ pip install --user pygal
```

Pygal on Windows

```
> python -m pip install --user pygal
```

Line graphs, scatter plots, and bar graphs

To make a plot with Pygal, you specify the kind of plot and then add the data.

Making a line graph

To view the output, open the file squares.svg in a browser.

```
import pygal

x_values = [0, 1, 2, 3, 4, 5]
squares = [0, 1, 4, 9, 16, 25]

chart = pygal.Line()
chart.force_uri_protocol = 'http'
chart.add('x^2', squares)
chart.render_to_file('squares.svg')
```

Adding labels and a title

```
--snip--
chart = pygal.Line()
chart.force_uri_protocol = 'http'
chart.title = "Squares"
chart.x_labels = x_values
chart.x_title = "Value"
chart.y_title = "Square of Value"
chart.add('x^2', squares)
chart.render_to_file('squares.svg')
```

Line graphs, scatter plots, and bar graphs (cont.)

Making a scatter plot

The data for a scatter plot needs to be a list containing tuples of the form (x, y). The `stroke=False` argument tells Pygal to make an XY chart with no line connecting the points.

```
import pygal

squares = [
    (0, 0), (1, 1), (2, 4), (3, 9),
    (4, 16), (5, 25),
    ]

chart = pygal.XY(stroke=False)
chart.force_uri_protocol = 'http'
chart.add('x^2', squares)
chart.render_to_file('squares.svg')
```

Using a list comprehension for a scatter plot

A list comprehension can be used to effficiently make a dataset for a scatter plot.

```
squares = [(x, x**2) for x in range(1000)]
```

Making a bar graph

A bar graph requires a list of values for the bar sizes. To label the bars, pass a list of the same length to `x_labels`.

```
import pygal

outcomes = [1, 2, 3, 4, 5, 6]
frequencies = [18, 16, 18, 17, 18, 13]

chart = pygal.Bar()
chart.force_uri_protocol = 'http'
chart.x_labels = outcomes
chart.add('D6', frequencies)
chart.render_to_file('rolling_dice.svg')
```

Making a bar graph from a dictionary

Since each bar needs a label and a value, a dictionary is a great way to store the data for a bar graph. The keys are used as the labels along the x-axis, and the values are used to determine the height of each bar.

```
import pygal

results = {
    1:18, 2:16, 3:18,
    4:17, 5:18, 6:13,
    }

chart = pygal.Bar()
chart.force_uri_protocol = 'http'
chart.x_labels = results.keys()
chart.add('D6', results.values())
chart.render_to_file('rolling_dice.svg')
```

Multiple plots

You can add as much data as you want when making a visualization.

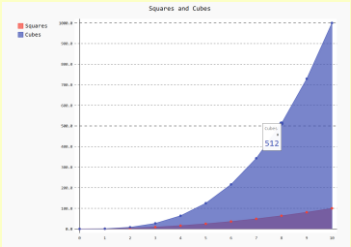
Plotting squares and cubes

```
import pygal

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

chart = pygal.Line()
chart.force_uri_protocol = 'http'
chart.title = "Squares and Cubes"
chart.x_labels = x_values

chart.add('Squares', squares)
chart.add('Cubes', cubes)
chart.render_to_file('squares_cubes.svg')
```

Filling the area under a data series

Pygal allows you to fill the area under or over each series of data. The default is to fill from the x-axis up, but you can fill from any horizontal line using the `zero` argument.

```
chart = pygal.Line(fill=True, zero=0)
```



Online resources

The documentation for Pygal is available at http://www.pygal.org/.

Enabling interactive features

If you're viewing svg output in a browser, Pygal needs to render the output file in a specific way. The `force_uri_protocol` attribute for chart objects needs to be set to `'http'`.

Styling plots

Pygal lets you customize many elements of a plot. There are some excellent default themes, and many options for styling individual plot elements.

Using built-in styles

To use built-in styles, import the style and make an instance of the style class. Then pass the style object with the `style` argument when you make the chart object.

```
import pygal
from pygal.style import LightGreenStyle

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

chart_style = LightGreenStyle()
chart = pygal.Line(style=chart_style)
chart.force_uri_protocol = 'http'
chart.title = "Squares and Cubes"
chart.x_labels = x_values

chart.add('Squares', squares)
chart.add('Cubes', cubes)
chart.render_to_file('squares_cubes.svg')
```

Parametric built-in styles

Some built-in styles accept a custom color, then generate a theme based on that color.

```
from pygal.style import LightenStyle

--snip--
chart_style = LightenStyle('#336688')
chart = pygal.Line(style=chart_style)
--snip--
```

Customizing individual style properties

Style objects have a number of properties you can set individually.

```
chart_style = LightenStyle('#336688')
chart_style.plot_background = '#CCCCCC'
chart_style.major_label_font_size = 20
chart_style.label_font_size = 16
--snip--
```

Custom style class

You can start with a bare style class, and then set only the properties you care about.

```
chart_style = Style()
chart_style.colors = [
    '#CCCCCC', '#AAAAAA', '#888888']
chart_style.plot_background = '#EEEEEE'

chart = pygal.Line(style=chart_style)
--snip--
```

Styling plots (cont.)

Configuration settings

Some settings are controlled by a `Config` object.

```
my_config = pygal.Config()
my_config.show_y_guides = False
my_config.width = 1000
my_config.dots_size = 5

chart = pygal.Line(config=my_config)
--snip--
```

Styling series

You can give each series on a chart different style settings.

```
chart.add('Squares', squares, dots_size=2)
chart.add('Cubes', cubes, dots_size=3)
```

Styling individual data points

You can style individual data points as well. To do so, write a dictionary for each data point you want to customize. A `'value'` key is required, and other properies are optional.

```
import pygal

repos = [
    {
        'value': 20506,
        'color': '#3333CC',
        'xlink': 'http://djangoproject.com/',
    },
    20054,
    12607,
    11827,
    ]

chart = pygal.Bar()
chart.force_uri_protocol = 'http'
chart.x_labels = [
    'django', 'requests', 'scikit-learn',
    'tornado',
    ]
chart.y_title = 'Stars'
chart.add('Python Repos', repos)
chart.render_to_file('python_repos.svg')
```

Plotting global datasets

Pygal can generate world maps, and you can add any data you want to these maps. Data is indicated by coloring, by labels, and by tooltips that show data when users hover over each country on the map.

Installing the world map module

The world map module is not included by default in Pygal 2.0. It can be installed with pip:

```
$ pip install --user pygal_maps_world
```

Making a world map

The following code makes a simple world map showing the countries of North America.

```
from pygal.maps.world import World

wm = World()
wm.force_uri_protocol = 'http'
wm.title = 'North America'
wm.add('North America', ['ca', 'mx', 'us'])

wm.render_to_file('north_america.svg')
```

Showing all the country codes

In order to make maps, you need to know Pygal's country codes. The following example will print an alphabetical list of each country and its code.

```
from pygal.maps.world import COUNTRIES

for code in sorted(COUNTRIES.keys()):
    print(code, COUNTRIES[code])
```

Plotting numerical data on a world map

To plot numerical data on a map, pass a dictionary to `add()` instead of a list.

```
from pygal.maps.world import World

populations = {
    'ca': 34126000,
    'us': 309349000,
    'mx': 113423000,
    }

wm = World()
wm.force_uri_protocol = 'http'
wm.title = 'Population of North America'
wm.add('North America', populations)

wm.render_to_file('na_populations.svg')
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Django

What is Django?

Django is a web framework which helps you build interactive websites using Python. With Django you define the kind of data your site needs to work with, and you define the ways your users can work with that data.

Installing Django

It's usualy best to install Django to a virtual environment, where your project can be isolated from your other Python projects. Most commands assume you're working in an active virtual environment.

Create a virtual environment

```
$ python -m venv ll_env
```

Activate the environment (Linux and OS X)

```
$ source ll_env/bin/activate
```

Activate the environment (Windows)

```
> ll_env\Scripts\activate
```

Install Django to the active environment

```
(ll_env)$ pip install Django
```

Creating a project

To start a project we'll create a new project, create a database, and start a development server.

Create a new project

```
$ django-admin.py startproject learning_log .
```

Create a database

```
$ python manage.py migrate
```

View the project

After issuing this command, you can view the project at http://localhost:8000/.

```
$ python manage.py runserver
```

Create a new app

A Django project is made up of one or more apps.

```
$ python manage.py startapp learning_logs
```

Working with models

The data in a Django project is structured as a set of models.

Defining a model

To define the models for your app, modify the file models.py that was created in your app's folder. The __str__() method tells Django how to represent data objects based on this model.

```
from django.db import models

class Topic(models.Model):
    """A topic the user is learning about."""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(
            auto_now_add=True)

    def __str__(self):
        return self.text
```

Activating a model

To use a model the app must be added to the tuple `INSTALLED_APPS`*, which is stored in the project's settings.py file.*

```
INSTALLED_APPS = (
    --snip--
    'django.contrib.staticfiles',

    # My apps
    'learning_logs',
    )
```

Migrating the database

The database needs to be modified to store the kind of data that the model represents.

```
$ python manage.py makemigrations learning_logs
$ python manage.py migrate
```

Creating a superuser

A superuser is a user account that has access to all aspects of the project.

```
$ python manage.py createsuperuser
```

Registering a model

You can register your models with Django's admin site, which makes it easier to work with the data in your project. To do this, modify the app's admin.py file. View the admin site at http://localhost:8000/admin/.

```
from django.contrib import admin

from learning_logs.models import Topic

admin.site.register(Topic)
```

Building a simple home page

Users interact with a project through web pages, and a project's home page can start out as a simple page with no data. A page usually needs a URL, a view, and a template.

Mapping a project's URLs

The project's main urls.py file tells Django where to find the urls.py files associated with each app in the project.

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('learning_logs.urls',
            namespace='learning_logs')),
    ]
```

Mapping an app's URLs

An app's urls.py file tells Django which view to use for each URL in the app. You'll need to make this file yourself, and save it in the app's folder.

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.index, name='index'),
    ]
```

Writing a simple view

A view takes information from a request and sends data to the browser, often through a template. View functions are stored in an app's views.py file. This simple view function doesn't pull in any data, but it uses the template index.html to render the home page.

```
from django.shortcuts import render

def index(request):
    """The home page for Learning Log."""
    return render(request,
            'learning_logs/index.html')
```

Online resources

The documentation for Django is available at http://docs.djangoproject.com/. The Django documentation is thorough and user-friendly, so check it out!

Building a simple home page (cont.)

Writing a simple template

A template sets up the structure for a page. It's a mix of html and template code, which is like Python but not as powerful. Make a folder called templates inside the project folder. Inside the templates folder make another folder with the same name as the app. This is where the template files should be saved.

```
<p>Learning Log</p>

<p>Learning Log helps you keep track of your
learning, for any topic you're learning
about.</p>
```

Template inheritance

Many elements of a web page are repeated on every page in the site, or every page in a section of the site. By writing one parent template for the site, and one for each section, you can easily modify the look and feel of your entire site.

The parent template

The parent template defines the elements common to a set of pages, and defines blocks that will be filled by individual pages.

```
<p>
  <a href="{% url 'learning_logs:index' %}">
    Learning Log
  </a>
</p>

{% block content %}{% endblock content %}
```

The child template

The child template uses the `{% extends %}` *template tag to pull in the structure of the parent template. It then defines the content for any blocks defined in the parent template.*

```
{% extends 'learning_logs/base.html' %}

{% block content %}
  <p>
    Learning Log helps you keep track
    of your learning, for any topic you're
    learning about.
  </p>
{% endblock content %}
```

Template indentation

Python code is usually indented by four spaces. In templates you'll often see two spaces used for indentation, because elements tend to be nested more deeply in templates.

Another model

A new model can use an existing model. The ForeignKey attribute establishes a connection between instances of the two related models. Make sure to migrate the database after adding a new model to your app.

Defining a model with a foreign key

```
class Entry(models.Model):
    """Learning log entries for a topic."""
    topic = models.ForeignKey(Topic)
    text = models.TextField()
    date_added = models.DateTimeField(
            auto_now_add=True)

    def __str__(self):
        return self.text[:50] + "..."
```

Building a page with data

Most pages in a project need to present data that's specific to the current user.

URL parameters

A URL often needs to accept a parameter telling it which data to access from the database. The second URL pattern shown here looks for the ID of a specific topic and stores it in the parameter `topic_id`.

```
urlpatterns = [
    url(r'^$', views.index, name='index'),
    url(r'^topics/(?P<topic_id>\d+)/$',
            views.topic, name='topic'),
    ]
```

Using data in a view

The view uses a parameter from the URL to pull the correct data from the database. In this example the view is sending a `context` *dictionary to the template, containing data that should be displayed on the page.*

```
def topic(request, topic_id):
    """Show a topic and all its entries."""
    topic = Topics.objects.get(id=topic_id)
    entries = topic.entry_set.order_by(
            '-date_added')
    context = {
        'topic': topic,
        'entries': entries,
        }
    return render(request,
        'learning_logs/topic.html', context)
```

Restarting the development server

If you make a change to your project and the change doesn't seem to have any effect, try restarting the server:

```
$ python manage.py runserver
```

Building a page with data (cont.)

Using data in a template

The data in the view function's `context` *dictionary is available within the template. This data is accessed using template variables, which are indicated by doubled curly braces.*

The vertical line after a template variable indicates a filter. In this case a filter called `date` *formats date objects, and the filter* `linebreaks` *renders paragraphs properly on a web page.*

```
{% extends 'learning_logs/base.html' %}

{% block content %}

  <p>Topic: {{ topic }}</p>

  <p>Entries:</p>
  <ul>
  {% for entry in entries %}
    <li>
    <p>
      {{ entry.date_added|date:'M d, Y H:i' }}
    </p>
    <p>
      {{ entry.text|linebreaks }}
    </p>
    </li>
  {% empty %}
    <li>There are no entries yet.</li>
  {% endfor %}
  </ul>

{% endblock content %}
```

The Django shell

You can explore the data in your project from the command line. This is helpful for developing queries and testing code snippets.

Start a shell session

```
$ python manage.py shell
```

Access data from the project

```
>>> from learning_logs.models import Topic
>>> Topic.objects.all()
[<Topic: Chess>, <Topic: Rock Climbing>]
>>> topic = Topic.objects.get(id=1)
>>> topic.text
'Chess'
```

More cheat sheets available at
ehmatthes.github.io/pcc/

Beginner's Python Cheat Sheet — Django, Part 2

Users and forms

Most web applications need to let users create accounts. This lets users create and work with their own data. Some of this data may be private, and some may be public. Django's forms allow users to enter and modify their data.

User accounts

User accounts are handled by a dedicated app called `users`*. Users need to be able to register, log in, and log out. Django automates much of this work for you.*

Making a users app

After making the app, be sure to add `'users'` *to* `INSTALLED_APPS` *in the project's settings.py file.*

```
$ python manage.py startapp users
```

Including URLS for the users app

Add a line to the project's urls.py file so the `users` *app's URLs are included in the project.*

```
urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^users/', include('users.urls',
            namespace='users')),
    url(r'', include('learning_logs.urls',
            namespace='learning_logs')),
    ]
```

Using forms in Django

There are a number of ways to create forms and work with them. You can use Django's defaults, or completely customize your forms. For a simple way to let users enter data based on your models, use a `ModelForm`*. This creates a form that allows users to enter data that will populate the fields on a model.*

The register view on the back of this sheet shows a simple approach to form processing. If the view doesn't receive data from a form, it responds with a blank form. If it receives `POST` *data from a form, it validates the data and then saves it to the database.*

User accounts (cont.)

Defining the URLs

Users will need to be able to log in, log out, and register. Make a new urls.py file in the `users` *app folder. The login view is a default view provided by Django.*

```
from django.conf.urls import url
from django.contrib.auth.views import login

from . import views

urlpatterns = [
    url(r'^login/$', login,
        {'template_name': 'users/login.html'},
        name='login'),
    url(r'^logout/$', views.logout_view,
        name='logout'),
    url(r'^register/$', views.register,
        name='register'),
    ]
```

The login template

The login view is provided by default, but you need to provide your own login template. The template shown here displays a simple login form, and provides basic error messages. Make a templates folder in the users folder, and then make a users folder in the templates folder. Save this file as login.html.

The tag `{% csrf_token %}` *helps prevent a common type of attack with forms. The* `{{ form.as_p }}` *element displays the default login form in paragraph format. The* `<input>` *element named* `next` *redirects the user to the home page after a successful login.*

```
{% extends "learning_logs/base.html" %}

{% block content %}
  {% if form.errors %}
  <p>
    Your username and password didn't match.
    Please try again.
  </p>
  {% endif %}

  <form method="post"
      action="{% url 'users:login' %}">
    {% csrf token %}
    {{ form.as_p }}
    <button name="submit">log in</button>

    <input type="hidden" name="next"
      value="{% url 'learning_logs:index' %}"/>
  </form>

{% endblock content %}
```

User accounts (cont.)

Showing the current login status

You can modify the base.html template to show whether the user is currently logged in, and to provide a link to the login and logout pages. Django makes a `user` *object available to every template, and this template takes advantage of this object.*

The `user.is_authenticated` *tag allows you to serve specific content to users depending on whether they have logged in or not. The* `{{ user.username }}` *property allows you to greet users who have logged in. Users who haven't logged in see links to register or log in.*

```
<p>
  <a href="{% url 'learning_logs:index' %}">
    Learning Log
  </a>
  {% if user.is_authenticated %}
    Hello, {{ user.username }}.
    <a href="{% url 'users:logout' %}">
      log out
    </a>
  {% else %}
    <a href="{% url 'users:register' %}">
      register
    </a> -
    <a href="{% url 'users:login' %}">
      log in
    </a>
  {% endif %}
</p>

{% block content %}{% endblock content %}
```

The logout view

The `logout_view()` *function uses Django's* `logout()` *function and then redirects the user back to the home page. Since there is no logout page, there is no logout template. Make sure to write this code in the views.py file that's stored in the users app folder.*

```
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from django.contrib.auth import logout

def logout_view(request):
    """Log the user out."""
    logout(request)
    return HttpResponseRedirect(
        reverse('learning_logs:index'))
```

User accounts (cont.)

The register view

The register view needs to display a blank registration form when the page is first requested, and then process completed registration forms. A successful registration logs the user in and redirects to the home page.

```
from django.contrib.auth import login
from django.contrib.auth import authenticate
from django.contrib.auth.forms import \
        UserCreationForm

def register(request):
    """Register a new user."""
    if request.method != 'POST':
        # Show blank registration form.
        form = UserCreationForm()
    else:
        # Process completed form.
        form = UserCreationForm(
                data=request.POST)

        if form.is_valid():
            new_user = form.save()
            # Log in, redirect to home page.
            pw = request.POST['password1']
            authenticated_user = authenticate(
                username=new_user.username,
                password=pw
            )
            login(request, authenticated_user)
            return HttpResponseRedirect(
                reverse('learning_logs:index'))

    context = {'form': form}
    return render(request,
        'users/register.html', context)
```

Styling your project

The django-bootstrap3 app allows you to use the Bootstrap library to make your project look visually appealing. The app provides tags that you can use in your templates to style individual elements on a page. Learn more at http://django-bootstrap3.readthedocs.io/.

Deploying your project

Heroku lets you push your project to a live server, making it available to anyone with an internet connection. Heroku offers a free service level, which lets you learn the deployment process without any commitment. You'll need to install a set of heroku tools, and use git to track the state of your project. See http://devcenter.heroku.com/, and click on the Python link.

User accounts (cont.)

The register template

The register template displays the registration form in paragraph formats.

```
{% extends 'learning_logs/base.html' %}

{% block content %}

  <form method='post'
      action="{% url 'users:register' %}">

    {% csrf_token %}
    {{ form.as_p }}

    <button name='submit'>register</button>
    <input type='hidden' name='next'
      value="{% url 'learning_logs:index' %}"/>

  </form>

{% endblock content %}
```

Connecting data to users

Users will have data that belongs to them. Any model that should be connected directly to a user needs a field connecting instances of the model to a specific user.

Making a topic belong to a user

Only the highest-level data in a hierarchy needs to be directly connected to a user. To do this import the `User` model, and add it as a foreign key on the data model.

After modifying the model you'll need to migrate the database. You'll need to choose a user ID to connect each existing instance to.

```
from django.db import models
from django.contrib.auth.models import User

class Topic(models.Model):
    """A topic the user is learning about."""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(
            auto_now_add=True)
    owner = models.ForeignKey(User)

    def __str__(self):
        return self.text
```

Querying data for the current user

In a view, the request object has a `user` attribute. You can use this attribute to query for the user's data. The `filter()` function then pulls the data that belongs to the current user.

```
topics = Topic.objects.filter(
                owner=request.user)
```

Connecting data to users (cont.)

Restricting access to logged-in users

Some pages are only relevant to registered users. The views for these pages can be protected by the `@login_required` decorator. Any view with this decorator will automatically redirect non-logged in users to an appropriate page. Here's an example views.py file.

```
from django.contrib.auth.decorators import /
        login_required
--snip--

@login_required
def topic(request, topic_id):
    """Show a topic and all its entries."""
```

Setting the redirect URL

The `@login_required` decorator sends unauthorized users to the login page. Add the following line to your project's settings.py file so Django will know how to find your login page.

```
LOGIN_URL = '/users/login/'
```

Preventing inadvertent access

Some pages serve data based on a parameter in the URL. You can check that the current user owns the requested data, and return a 404 error if they don't. Here's an example view.

```
from django.http import Http404

--snip--
def topic(request, topic_id):
    """Show a topic and all its entries."""
    topic = Topics.objects.get(id=topic_id)
    if topic.owner != request.user:
        raise Http404
    --snip--
```

Using a form to edit data

If you provide some initial data, Django generates a form with the user's existing data. Users can then modify and save their data.

Creating a form with initial data

The instance parameter allows you to specify initial data for a form.

```
form = EntryForm(instance=entry)
```

Modifying data before saving

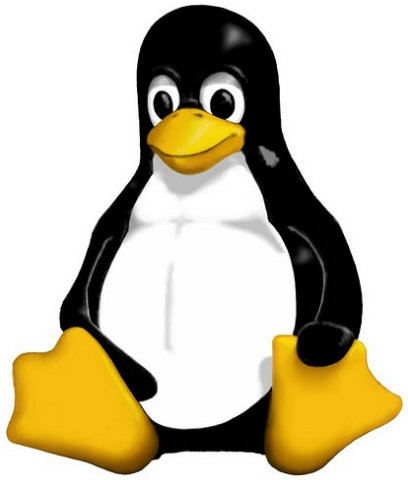
The argument `commit=False` allows you to make changes before writing data to the database.

```
new_topic = form.save(commit=False)
new_topic.owner = request.user
new_topic.save()
```

More cheat sheets available at
ehmatthes.github.io/pcc/

freeworld.posterous.com

Linux Bash Shell Cheat Sheet



(works with about every distribution, except for apt-get which is Ubuntu/Debian exclusive)

Legend:

Everything in “<>” is to be replaced, ex: <fileName> --> iLovePeanuts.txt
Don't include the '=' in your commands
'..' means that more than one file can be affected with only one command ex: rm file.txt file2.txt movie.mov .. ..

Linux Bash Shell Cheat Sheet

Basic Commands

Basic Terminal Shortcuts

CTRL L = Clear the terminal
CTRL D = Logout
SHIFT Page Up/Down = Go up/down the terminal
CTRL A = Cursor to start of line
CTRL E = Cursor the end of line
CTRL U = Delete left of the cursor
CTRL K = Delete right of the cursor
CTRL W = Delete word on the left
CTRL Y = Paste (after CTRL U,K or W)
TAB = auto completion of file or command
CTRL R = reverse search history
!! = repeat last command
CTRL Z = stops the current command (resume with fg in foreground or bg in background)

Basic Terminal Navigation

ls -a = list all files and folders
ls <folderName> = list files in folder
ls -lh = Detailed list, Human readable
ls -l *.jpg = list jpeg files only
ls -lh <fileName> = Result for file only

cd <folderName> = change directory
if folder name has spaces use “ “
cd / = go to root
cd .. = go up one folder, tip: ../../../

du -h: Disk usage of folders, human readable
du -ah: “ “ “ files & folders, Human readable
du -sh: only show disc usage of folders

pwd = print working directory

man <command> = shows manual (RTFM)

Basic file manipulation

cat <fileName> = show content of file
(less, more)
head = from the top
-n <#oflines> <fileName>

tail = from the bottom
-n <#oflines> <fileName>

mkdir = create new folder
mkdir myStuff ..
mkdir myStuff/pictures/ ..

cp image.jpg newimage.jpg = copy and rename a file
cp image.jpg <folderName>/ = copy to folder
cp image.jpg folder/sameImageNewName.jpg
cp -R stuff otherStuff = copy and rename a folder
cp *.txt stuff/ = copy all of *<file type> to folder

mv file.txt Documents/ = move file to a folder
mv <folderName> <folderName2> = move folder in folder
mv filename.txt filename2.txt = rename file
mv <fileName> stuff/newfileName
mv <folderName>/ .. = move folder up in hierarchy

rm <fileName> .. = delete file (s)
rm -i <fileName> .. = ask for confirmation each file
rm -f <fileName> = force deletion of a file
rm -r <foldername>/ = delete folder

touch <fileName> = create or update a file

ln file1 file2 = physical link
ln -s file1 file2 = symbolic link

Linux Bash Shell Cheat Sheet

Basic Commands

Researching Files

The slow method (sometimes very slow):

```
locate <text> = search the content of all the files
locate <fileName> = search for a file
sudo updatedb = update database of files

find = the best file search tool (fast)
find -name “<fileName>”
find -name “text” = search for files who start with the word text
find -name “*text” = “      “    “    “   end   “    “   “    “
```

Advanced Search:

```
Search from file Size (in ~)
      find ~ -size +10M = search files bigger than.. (M,K,G)

Search from last access
      find -name “<filetype>” -atime -5
            ('-' = less than, '+' = more than and nothing = exactly)

Search only files or directory’s
      find -type d --> ex: find /var/log -name "syslog" -type d
      find -type f = files

More info: man find, man locate
```

Extract, sort and filter data

```
grep <someText> <fileName> = search for text in file
      -i = Doesn't consider uppercase words
      -I = exclude binary files
grep -r <text> <folderName>/ = search for file names
      with occurrence of the text
```

With regular expressions:

```
grep -E ^<text> <fileName> = search start of lines
with the word text
grep -E <0-4> <fileName> =shows lines containing numbers 0-4
grep -E <a-zA-Z> <fileName> = retrieve all lines
with alphabetical letters

sort = sort the content of files
sort <fileName> = sort alphabetically
sort -o <file> <outputFile> = write result to a file
sort -r <fileName> = sort in reverse
sort -R <fileName> = sort randomly
sort -n <fileName> = sort numbers

wc = word count
wc <fileName> = nbr of line, nbr of words, byte size
      -l (lines), -w (words), -c (byte size), -m
      (number of characters)

cut = cut a part of a file
-c --> ex: cut -c 2-5 names.txt
      (cut the characters 2 to 5 of each line)
-d (delimiter)         (-d & -f good for .csv files)
-f (# of field to cut)

more info: man cut, man sort, man grep
```

Linux Bash Shell Cheat Sheet

Basic Commands

Time settings

date = view & modify time (on your computer)

View:
date “+%H” --> If it's 9 am, then it will show 09
date “+%H:%M:%Ss” = (hours, minutes, seconds)
%Y = years
Modify:
MMDDhhmmYYYY
Month | Day | Hours | Minutes | Year

sudo date 031423421997 = March 14th 1997, 23:42

Execute programs at another time

use 'at' to execute programs in the future

Step 1, write in the terminal: at <timeOfExecution> ENTER
ex --> **at 16:45** or **at 13:43 7/23/11** (to be more precise)
or after a certain delay:
at now +5 minutes (hours, days, weeks, months, years)
Step 2: <ENTER COMMAND> ENTER
repeat step 2 as many times you need
Step 3: CTRL D to close input

atq = show a list of jobs waiting to be executed

atrm = delete a job n°<x>
ex (delete job #42) --> atrm 42

sleep = pause between commands
with ';' you can chain commands, ex: touch file; rm file
you can make a pause between commands (**m**inutes, **h**ours, **d**ays)
ex --> touch file; sleep 10; rm file <-- 10 seconds

(continued)

crontab = execute a command regularly
-e = modify the crontab
-l = view current crontab
-r = delete you crontab
In crontab the syntax is
<Minutes> <Hours> <Day of month> <Day of week (0-6, 0 = Sunday)> <COMMAND>

ex, create the file movies.txt every day at 15:47:
47 15 * * * touch /home/bob/movies.txt
* * * * * --> every minute
at 5:30 in the morning, from the 1st to 15th each month:
30 5 1-15 * *
at midnight on Mondays, Wednesdays and Thursdays:
0 0 * * 1,3,4
every two hours:
0 */2 * * *
every 10 minutes Monday to Friday:
*/10 * * * 1-5

Execute programs in the background

Add a '&' at the end of a command
ex --> cp bigMovieFile.mp4 &

nohup: ignores the HUP signal when closing the console (process will still run if the terminal is closed)
ex --> nohup cp bigMovieFile.mp4

jobs = know what is running in the background

fg = put a background process to foreground
ex: fg (process 1), f%2 (process 2) f%3, ...

Linux Bash Shell Cheat Sheet

Basic Commands

Process Management

```
w = who is logged on and what they are doing

tload = graphic representation of system load average
       (quit with CTRL C)

ps = Static process list
       -ef --> ex: ps -ef | less
       -ejH --> show process hierarchy
       -u --> process's from current user

top = Dynamic process list
While in top:
```

- q to close top
- h to show the help
- k to kill a process

```
CTRL C to top a current terminal process

kill = kill a process
       You need the PID # of the process
              ps -u <AccountName> | grep <Application>
       Then
              kill <PID> .. .. ..
kill -9 <PID> = violent kill

killall = kill multiple process's
       ex --> killall locate

extras:
       sudo halt <-- to close computer
       sudo reboot <-- to reboot
```

Create and modify user accounts

```
sudo adduser bob = root creates new user
sudo passwd <AccountName> = change a user's password
sudo deluser <AccountName> = delete an account

addgroup friends = create a new user group
delgroup friends = delete a user group

usermod -g friends <Account> = add user to a group
usermod -g bob boby = change account name
usermod -aG friends bob = add groups to a user with-
out loosing the ones he's already in
```

File Permissions

```
chown = change the owner of a file
       ex --> chown bob hello.txt
chown user:bob report.txt = changes the user owning
report.txt to 'user' and the group owning it to 'bob'
-R = recursively affect all the sub folders
       ex --> chown -R bob:bob /home/Daniel

chmod = modify user access/permission – simple way
       u = user
       g = group
       o = other

       d = directory (if element is a directory)
       l = link (if element is a file link)
       r = read (read permissions)
       w = write (write permissions)
       x = eXecute (only useful for scripts and
       programs)
```

Linux Bash Shell Cheat Sheet

Basic Commands

File Permissions (continued)

```
'+' means add a right
'-' means delete a right
'=' means affect a right

ex --> chmod g+w someFile.txt
       (add to current group the right to modify someFile.txt)

more info: man chmod
```

Flow redirection

```
Redirect results of commands:

'>' at the end of a command to redirect the result to a file
       ex --> ps -ejH > process.txt
'>>' to redirect the result to the end of a file
```

Redirect errors:

```
'2>' at the end of the command to redirect the result to a file
       ex --> cut -d , -f 1 file.csv > file 2> errors.log
'2>&1' to redirect the errors the same way as the standard output

Read progressively from the keyboard

<Command> << <wordToTerminateInput>
       ex --> sort << END <-- This can be anything you want
              > Hello
              > Alex
              > Cinema
              > Game
              > Code
              > Ubuntu
              > END
```

Flow Redirection (continued)

```
terminal output:
Alex
Cinema
Code
Game
Ubuntu

       Another example --> wc -m << END
```

Chain commands

```
'|' at the end of a command to enter another one
       ex --> du | sort -nr | less
```

Archive and compress data

```
Archive and compress data the long way:

Step 1, put all the files you want to compress in
the same folder: ex --> mv *.txt folder/

Step 2, Create the tar file:
tar -cvf my_archive.tar folder/
       -c : creates a .tar archive
       -v : tells you what is happening (verbose)
       -f : assembles the archive into one file

       Step 3.1, create gzip file (most current):
gzip my_archive.tar
       to decompress: gunzip my_archive.tar.gz

Step 3.2, or create a bzip2 file (more powerful but slow):
bzip2 my_archive.tar
       to decompress: bunzip2 my_archive.tar.bz2
```

Linux Bash Shell Cheat Sheet

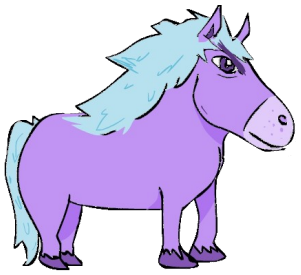
Basic Commands

Archive and compress data (continued)

```
step 4, to decompress the .tar file:
       tar -xvf archive.tar archive.tar

Archive and compress data the fast way:

gzip: tar -zcvf my_archive.tar.gz folder/
      decompress: tar -zcvf my_archive.tar.gz Documents/

bzip2: tar -jcvf  my_archive.tar.gz folder/
      decompress: tar -jxvf archive.tar.bz2 Documents/

Show the content of .tar, .gz or .bz2  without decompressing it:

gzip:
      gzip -ztf archive.tar.gz
bzip2:
      bzip2 -jtf archive.tar.bz2
tar:
      tar -tf archive.tar

tar extra:
      tar -rvf archive.tar file.txt = add a file to the .tar

You can also directly compress a single file and view the file
without decompressing:

Step 1, use gzip or bzip2 to compress the file:
      gzip numbers.txt

Step 2, view the file without decompressing it:
      zcat = view the entire file in the console (same as cat)
      zmore = view one screen at a time the content of the file (same as more)
      zless = view one line of the file at a time (same as less)
```

Installing software

```
When software is available in the repositories:
sudo apt-get install <nameOfSoftware>
      ex--> sudo apt-get install aptitude

If you download it from the Internets in .gz format
(or bz2) - “Compiling from source”
Step 1, create a folder to place the file:
      mkdir /home/username/src <-- then cd to it

Step 2, with 'ls' verify that the file is there
(if not, mv ../file.tar.gz /home/username/src/)

Step 3, decompress the file (if .zip: unzip <file>)
<--
Step 4, use 'ls', you should see a new directory
Step 5, cd to the new directory
Step 6.1, use ls to verify you have an INSTALL file,
then: more INSTALL
If you don't have an INSTALL file:
Step 6.2, execute ./configure <-- creates a makefile
Step 6.2.1, run make <-- builds application binaries
Step 6.2.2 : switch to root --> su
Step 6.2.3 : make install <-- installs the software
Step 7, read the readme file
```



DVC Cheat Sheet

https://github.com/iterative/dvc

https://dvc.org/chat

Basics

Initializing

Initialize a DVC environment

```
$ dvc init
```

Remote

Set up a remote to keep and share data files

```
$ dvc remote add -d myremote /path
```

**Possible remotes include local, s3, gs, azure, ssh, hdfs and http.*

Show all available remotes

```
$ dvc remote list
```

Modify remote settings

```
$ dvc remote modify myremote
```

**Use if remote requires extra configuration*

Adding Files

Add files under DVC control

```
$ dvc add filename
```

**Use --no-commit to stop adding the file to the cache.*

Share Data

Push all data files to the remote storage

```
$ dvc push
```

Push outputs of a specific .dvc file

```
$ dvc push filename.dvc
```

Retrieve Data

Download files from the remote storage

```
$ dvc pull
```

Download files from a specific .dvc file

```
$ dvc pull filename.dvc
```

Checkout files from cache into working space

```
$ dvc checkout
```

The Pipeline

Add transformations and generate a stage file from a given command

```
$ dvc run -d dependencyfile \
  -o outputfile python command.py
```

**Use --file to specify the name of the generated .dvc file.*

**Use --metrics to output a file containing the metric.*

Metrics

Collect and display project metrics

```
$ dvc metrics show
```

**Use --all to show the metrics in all branches.*

Visualizing

Show stages in a pipeline

```
$ dvc pipeline show --ascii file.dvc
```

**Add --commands or --outs to show more detail.*

Show connected pipelines of DVC stages

```
$ dvc pipeline list
```

Reproducing

Reproduce outputs defined in .dvc file

```
$ dvc repro filename.dvc
```

**Name a .dvc file "Dvcfile" to be use by dvc repro by default*

Other Commands

Set/unset cache directory location

```
$ dvc cache dir /path
```

Commit outputs to cache

```
$ dvc commit
```

**Use if you specified --no-commit in dvc add/run/repro*

Config repository or global options

```
$ dvc config
```

**Config the default remote using core.remote myremote*

**Config core (loglevel, remote), cache and state settings*

Fetch files from the remote to the local cache

```
$ dvc fetch file.dvc
```

Remove unused objects from cache

```
$ dvc gc
```

Import file from URL to local directory

```
$ dvc import url /path
```

**Supported schemes include local, s3, gs, azure, ssh, hdfs and http.*

Remove data files tracked by dvc

```
$ dvc remove filename.dvc
```

Show changed stages in the pipeline

```
$ dvc status
```

Made by Carl Handlin based on the documentation for DVC at https://dvc.org/doc

Git Cheat Sheet

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

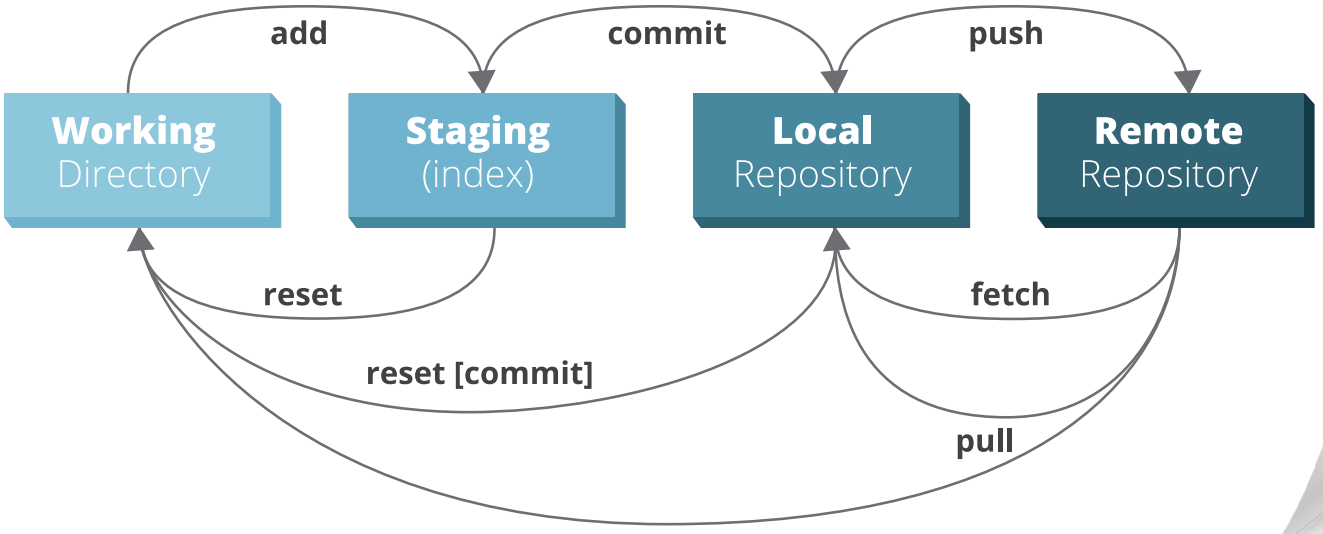
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



Git Cheat Sheet

Git Basics

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via `HTTP` or `SSH`.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

Undoing Changes

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

Rewriting Git History

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, a branch name, a tag, or a relative reference to `HEAD`.
`git reflog`	Show a log of changes to the local repository's `HEAD`. Add `--relative-date` flag to show date info or `--all` to show all refs.

Git Branches

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

Remote Repositories

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Additional Options +

git config

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog log --graph --oneline` will set `git glog` equivalent to `git log --graph --oneline`.
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

git log

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `git log -5` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author="<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

git diff

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

git reset

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

git rebase

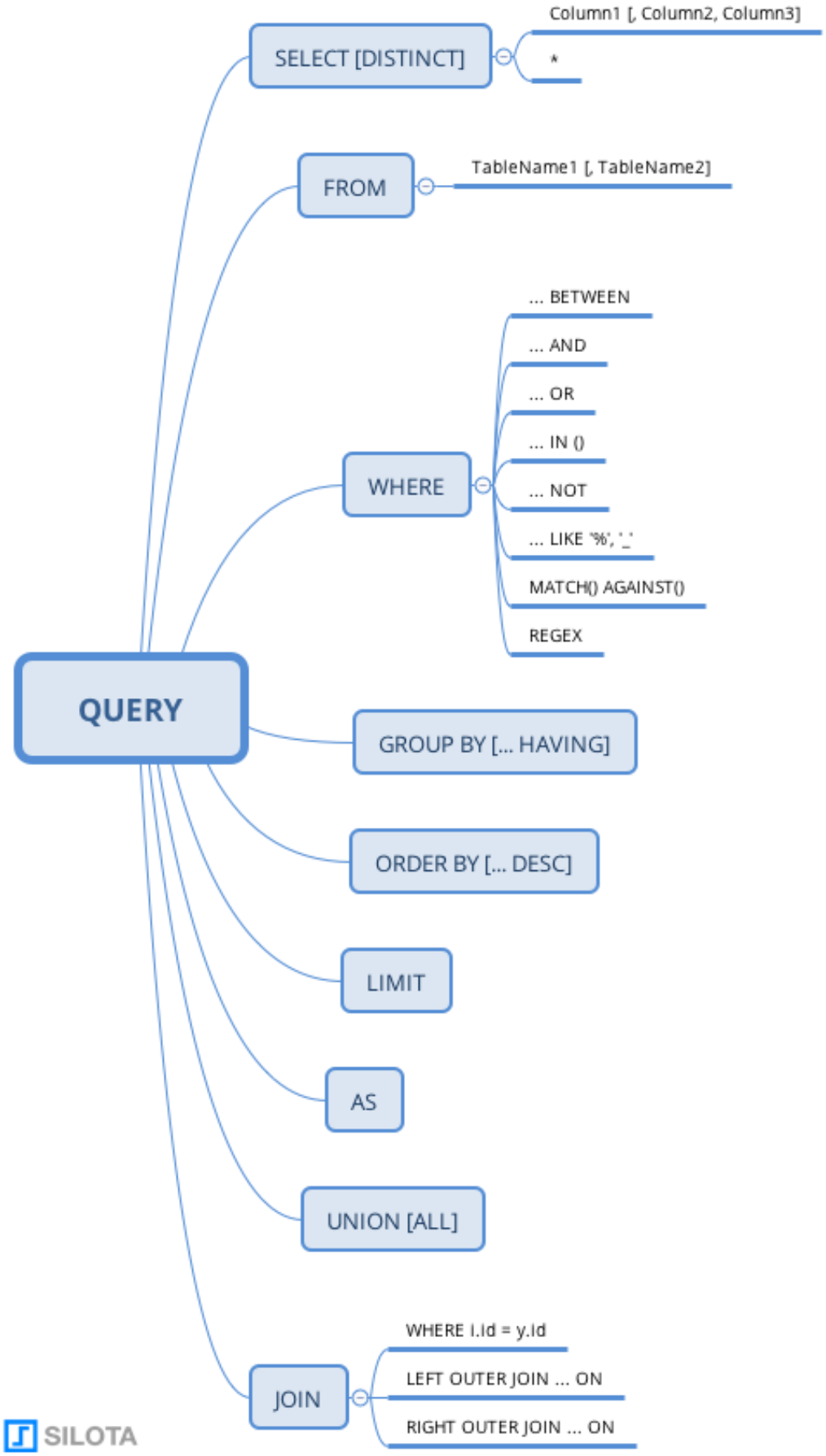
`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

git pull

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

git push

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all flag`. The `--tags` flag sends all of your local tags to the remote repo.


QUERY
SELECT [DISTINCT]
Column1 [, Column2, Column3]
*
FROM
TableName1 [, TableName2]
WHERE
... BETWEEN
... AND
... OR
... IN ()
... NOT
... LIKE '%', '_'
MATCH() AGAINST()
REGEX
GROUP BY [... HAVING]
ORDER BY [... DESC]
LIMIT
AS
UNION [ALL]
JOIN
WHERE l.id = y.id
LEFT OUTER JOIN ... ON
RIGHT OUTER JOIN ... ON
SILOTA

SQL cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Basic Queries

```
-- filter your columns
   SELECT col1, col2, 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 **SELECTS**:

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

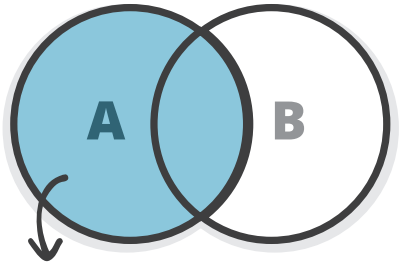
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

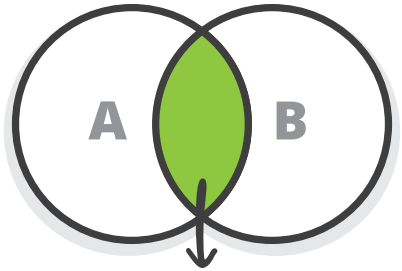
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

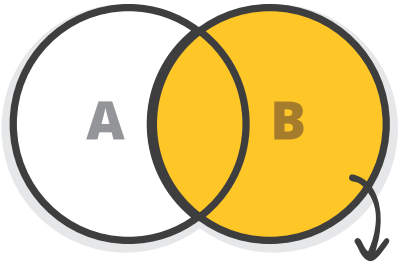
The Joy of JOINs



LEFT OUTER JOIN - *all rows from table A, even if they do not exist in table B*



INNER JOIN - *fetch the results that exist in both tables*



RIGHT OUTER JOIN - *all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value

SQL COMMANDS
CHEAT SHEET

SQL Commands

The commands in SQL are called Queries and they are of two types:

- Data Definition Query: The statements which defines the structure of a database, create tables, specify their keys, indexes and so on
- Data manipulation queries: These are the queries which can be edited.

  E.g.: Select, update and insert operation

Command	Syntax	Description
ALTER table	ALTER TABLE table_name ADD column_name datatype;	It is used to add columns to a table in a database
AND	SELECT column_name(s) FROM table_name WHERE column_1 = value_1 AND column_2 = value_2;	It is an operator that is used to combine two conditions
AS	SELECT column_name AS 'Alias' FROM table_name;	It is an keyword in SQL that is used to rename a column or table using an alias name
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value_1 AND value_2;	It is an operator used to filter the result within a certain range
CASE	SELECT column_name, CASE WHEN condition THEN 'Result_1' WHEN condition THEN 'Result_2' ELSE 'Result_3' END FROM table_name;	It is a statement used to create different outputs inside a SELECT statement
COUNT	SELECT COUNT(column_name) FROM table_name;	It is a function that takes the name of a column as argument and counts the number of rows when the column is not NULL
Create TABLE	CREATE TABLE table_name (column_1 datatype, column_2 datatype, column_3 datatype);	It is used to create a new table in a database and specify the name of the table and columns inside it

Command	Syntax	Description
GROUP BY	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;	It is an clause in SQL used for aggregate functions in collaboration with the SELECT statement
HAVING	SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name HAVING COUNT(*) > value;	It is used in SQL because the WHERE keyword cannot be used in aggregating functions
INNER JOIN	SELECT column_name(s) FROM table_1 JOIN table_2 ON table_1.column_name = table_2.column_name;	It is used to combine rows from different tables if the Join condition goes TRUE
INSERT	INSERT INTO table_name (column_1, column_2, column_3) VALUES (value_1, 'value_2', value_3);	It is used to add new rows to a table
IS NULL/ IS NOT NULL	SELECT column_name(s) FROM table_name WHERE column_name IS NULL;	It is a operator used with the WHERE clause to check for the empty values
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;	It is an special operator used with the WHERE clause to search for a specific pattern in a column
LIMIT	SELECT column_name(s) FROM table_name LIMIT number;	It is a clause to specify the maximum number of rows the result set must have
MAX	SELECT MAX(column_name) FROM table_name;	It is a function that takes number of columns as an argument and return the largest value among them
MIN	SELECT MIN(column_name) FROM table_name;	It is a function that takes number of columns as an argument and return the smallest value among them
OR	SELECT column_name FROM table_name WHERE column_name = value_1 OR column_name = value_2;	It is an operator that is used to filter the result set to contain only the rows where either condition is TRUE
ORDER BY	SELECT column_name FROM table_name ORDER BY column_name ASC | DESC;	It is a clause used to sort the result set by a particular column either numerically or alphabetically

Command	Syntax	Description
OUTER JOIN	SELECT column_name(s) FROM table_1 LEFT JOIN table_2 ON table_1.column_name = table_2.column_name;	It is sued to combine rows from different tables even if the condition is NOT TRUE
ROUND	SELECT ROUND(column_name, integer) FROM table_name;	It is a function that takes the column name and a integer as an argument, and rounds the values in a column to the number of decimal places specified by an integer
SELECT	SELECT column_name FROM table_name;	It is a statement that is used to fetch data from a database
SELECT DISTINCT	SELECT DISTINCT column_name FROM table_name;	It is used to specify that the statement is a query which returns unique values in specified columns
SUM	SELECT SUM(column_name) FROM table_name;	It is function used to return sum of values from a particular column
UPDATE	UPDATE table_name SET some_column = some_value WHERE some_column = some_value;	It is used to edit rows in a table
WHERE	SELECT column_name(s) FROM table_name WHERE column_name operator value;	It is a clause used to filter the result set to include the rows which where the condition is TRUE
WITH	WITH temporary_name AS (SELECT * FROM table_name) SELECT * FROM temporary_name WHERE column_name operator value;	It is used to store the result of a particular query in a temporary table using an alias
DELETE	DELETE FROM table_name WHERE some_column = some_value;	It is used to remove the rows from a table
AVG	SELECT AVG(column_name) FROM table_name;	It is used to aggregate a numeric column and return its average

Commands and syntax for querying data from Single Table	Commands and syntax for querying data from Multiple Table
SELECT c1 FROM t To select the data in Column c1 from table t	SELECT c1, c2 FROM t1 INNER JOIN t2 on condition Select column c1 and c2 from table t1 and perform an inner join between t1 and t2
SELECT * FROM t To select all rows and columns from table t	SELECT c1, c2 FROM t1 LEFT JOIN t2 on condition Select column c1 and c2 from table t1 and perform a left join between t1 and t2
SELECT c1 FROM t WHERE c1 = 'test' To select data in column c1 from table t, where c1=test	SELECT c1, c2 FROM t1 RIGHT JOIN t2 on condition Select column c1 and c2 from table t1 and perform a right join between t1 and t2
SELECT c1 FROM t ORDER BY c1 ASC (DESC) To select data in column c1 from table t either in ascending or descending order	SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 on condition Select column c1 and c2 from table t1 and perform a full outer join between t1 and t2
SELECT c1 FROM t ORDER BY c1LIMIT n OFFSET offset To skip the offset of rows and return the next n rows	SELECT c1, c2 FROM t1 CROSS JOIN t2 Select column c1 and c2 from table t1 and produce a Cartesian product of rows in a table
SELECT c1, aggregate(c2) FROM t GROUP BY c1 To group rows using an aggregate function	SELECT c1, c2 FROM t1, t2 Select column c1 and c2 from table t1 and produce a Cartesian product of rows in a table
SELECT c1, aggregate(c2) FROM t GROUP BY c1HAVING condition Group rows using an aggregate function and filter these groups using 'HAVING' clause	SELECT c1, c2 FROM t1 A INNER JOIN t2 B on condition Select column c1 and c2 from table t1 and join it to itself using INNER JOIN clause

FURTHERMORE:

SQL Developer, SQL DBA Training Masters Program

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t ;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
   c1 INT, c2 INT, c3 VARCHAR,
   PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
   c1 INT PRIMARY KEY,
   c2 INT,
   FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
   c1 INT, c1 INT,
   UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
       (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
    c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- **BEFORE** – invoke before the event occurs
- **AFTER** – invoke after the event occurs

EVENT
- **INSERT** – invoke for INSERT
- **UPDATE** – invoke for UPDATE
- **DELETE** – invoke for DELETE

TRIGGER_TYPE
- **FOR EACH ROW**
- **FOR EACH STATEMENT**

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger

VIP Refresher: Probabilities and Statistics

Afshine Amidi and Shervine Amidi

August 6, 2018

Introduction to Probability and Combinatorics

❒ **Sample space** – The set of all possible outcomes of an experiment is known as the sample space of the experiment and is denoted by S.

❒ **Event** – Any subset E of the sample space is known as an event. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in E, then we say that E has occurred.

❒ **Axioms of probability** – For each event E, we denote $P(E)$ as the probability of event E occuring. By noting $E_1,...,E_n$ mutually exclusive events, we have the 3 following axioms:

$$(1) \quad \boxed{0 \leqslant P(E) \leqslant 1} \quad (2) \quad \boxed{P(S) = 1} \quad (3) \quad \boxed{P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{i=1}^{n} P(E_i)}$$

❒ **Permutation** – A permutation is an arrangement of r objects from a pool of n objects, in a given order. The number of such arrangements is given by $P(n,r)$, defined as:

$$\boxed{P(n,r) = \frac{n!}{(n-r)!}}$$

❒ **Combination** – A combination is an arrangement of r objects from a pool of n objects, where the order does not matter. The number of such arrangements is given by $C(n,r)$, defined as:

$$\boxed{C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!}}$$

Remark: we note that for $0 \leqslant r \leqslant n$, we have $P(n,r) \geqslant C(n,r)$.

Conditional Probability

❒ **Bayes' rule** – For events A and B such that $P(B) > 0$, we have:

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

Remark: we have $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$.

❒ **Partition** – Let $\{A_i, i \in [\![1,n]\!]\}$ be such that for all i, $A_i \neq \varnothing$. We say that $\{A_i\}$ is a partition if we have:

$$\boxed{\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^{n} A_i = S}$$

Remark: for any event B in the sample space, we have $P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$.

❒ **Extended form of Bayes' rule** – Let $\{A_i, i \in [\![1,n]\!]\}$ be a partition of the sample space. We have:

$$\boxed{P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)}}$$

❒ **Independence** – Two events A and B are independent if and only if we have:

$$\boxed{P(A \cap B) = P(A)P(B)}$$

Random Variables

❒ **Random variable** – A random variable, often noted X, is a function that maps every element in a sample space to a real line.

❒ **Cumulative distribution function (CDF)** – The cumulative distribution function F, which is monotonically non-decreasing and is such that $\lim_{x \to -\infty} F(x) = 0$ and $\lim_{x \to +\infty} F(x) = 1$, is defined as:

$$\boxed{F(x) = P(X \leqslant x)}$$

Remark: we have $P(a < X \leqslant B) = F(b) - F(a)$.

❒ **Probability density function (PDF)** – The probability density function f is the probability that X takes on values between two adjacent realizations of the random variable.

❒ **Relationships involving the PDF and CDF** – Here are the important properties to know in the discrete (D) and the continuous (C) cases.

Case	CDF F	PDF f	Properties of PDF
(D)	$F(x) = \sum_{x_i \leqslant x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leqslant f(x_j) \leqslant 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^{x} f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geqslant 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

❒ **Variance** – The variance of a random variable, often noted $\text{Var}(X)$ or σ^2, is a measure of the spread of its distribution function. It is determined as follows:

$$\boxed{\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2}$$

❒ **Standard deviation** – The standard deviation of a random variable, often noted σ, is a measure of the spread of its distribution function which is compatible with the units of the actual random variable. It is determined as follows:

$$\boxed{\sigma = \sqrt{\text{Var}(X)}}$$

❒ **Expectation and Moments of the Distribution** – Here are the expressions of the expected value $E[X]$, generalized expected value $E[g(X)]$, k^{th} moment $E[X^k]$ and characteristic function $\psi(\omega)$ for the discrete and continuous cases:

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	$\psi(\omega)$
(D)	$\sum_{i=1}^{n} x_i f(x_i)$	$\sum_{i=1}^{n} g(x_i) f(x_i)$	$\sum_{i=1}^{n} x_i^k f(x_i)$	$\sum_{i=1}^{n} f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} x f(x) dx$	$\int_{-\infty}^{+\infty} g(x) f(x) dx$	$\int_{-\infty}^{+\infty} x^k f(x) dx$	$\int_{-\infty}^{+\infty} f(x) e^{i\omega x} dx$

Remark: we have $e^{i\omega x} = \cos(\omega x) + i \sin(\omega x)$.

❒ **Revisiting the k^{th} moment** – The k^{th} moment can also be computed with the characteristic function as follows:

$$\boxed{E[X^k] = \frac{1}{i^k} \left[\frac{\partial^k \psi}{\partial \omega^k} \right]_{\omega=0}}$$

❒ **Transformation of random variables** – Let the variables X and Y be linked by some function. By noting f_X and f_Y the distribution function of X and Y respectively, we have:

$$\boxed{f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|}$$

❒ **Leibniz integral rule** – Let g be a function of x and potentially c, and a, b boundaries that may depend on c. We have:

$$\boxed{\frac{\partial}{\partial c} \left(\int_a^b g(x) dx \right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x) dx}$$

❒ **Chebyshev's inequality** – Let X be a random variable with expected value μ and standard deviation σ. For $k, \sigma > 0$, we have the following inequality:

$$\boxed{P(|X - \mu| \geqslant k\sigma) \leqslant \frac{1}{k^2}}$$

Jointly Distributed Random Variables

❒ **Conditional density** – The conditional density of X with respect to Y, often noted $f_{X|Y}$, is defined as follows:

$$\boxed{f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}}$$

❒ **Independence** – Two random variables X and Y are said to be independent if we have:

$$\boxed{f_{XY}(x,y) = f_X(x) f_Y(y)}$$

❒ **Marginal density and cumulative distribution** – From the joint density probability function f_{XY}, we have:

Case	Marginal density	Cumulative function
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i, y_j)$	$F_{XY}(x,y) = \sum_{x_i \leqslant x} \sum_{y_j \leqslant y} f_{XY}(x_i, y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y) dy$	$F_{XY}(x,y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f_{XY}(x', y') dx' dy'$

❒ **Distribution of a sum of independent random variables** – Let $Y = X_1 + ... + X_n$ with $X_1, ..., X_n$ independent. We have:

$$\boxed{\psi_Y(\omega) = \prod_{k=1}^{n} \psi_{X_k}(\omega)}$$

❒ **Covariance** – We define the covariance of two random variables X and Y, that we note σ_{XY}^2 or more commonly $\text{Cov}(X,Y)$, as follows:

$$\boxed{\text{Cov}(X,Y) \triangleq \sigma_{XY}^2 = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y}$$

❒ **Correlation** – By noting σ_X, σ_Y the standard deviations of X and Y, we define the correlation between the random variables X and Y, noted ρ_{XY}, as follows:

$$\boxed{\rho_{XY} = \frac{\sigma_{XY}^2}{\sigma_X \sigma_Y}}$$

Remarks: For any X, Y, we have $\rho_{XY} \in [-1,1]$. If X and Y are independent, then $\rho_{XY} = 0$.

❒ **Main distributions** – Here are the main distributions to have in mind:

Type	Distribution	PDF	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n,p)$ Binomial	$P(X=x) = \binom{n}{x} p^x q^{n-x}$ $x \in [\![0,n]\!]$	$(pe^{i\omega} + q)^n$	np	npq
	$X \sim \text{Po}(\mu)$ Poisson	$P(X=x) = \frac{\mu^x}{x!} e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	μ	μ
(C)	$X \sim \mathcal{U}(a,b)$ Uniform	$f(x) = \frac{1}{b-a}$ $x \in [a,b]$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu,\sigma)$ Gaussian	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	μ	σ^2
	$X \sim \text{Exp}(\lambda)$ Exponential	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Parameter estimation

❒ **Random sample** – A random sample is a collection of n random variables $X_1, ..., X_n$ that are independent and identically distributed with X.

❒ **Estimator** – An estimator $\hat{\theta}$ is a function of the data that is used to infer the value of an unknown parameter θ in a statistical model.

❒ **Bias** – The bias of an estimator $\hat{\theta}$ is defined as being the difference between the expected value of the distribution of $\hat{\theta}$ and the true value, i.e.:

$$\boxed{\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta}$$

Remark: an estimator is said to be unbiased when we have $E[\hat{\theta}] = \theta$.

❒ **Sample mean and variance** – The sample mean and the sample variance of a random sample are used to estimate the true mean μ and the true variance σ^2 of a distribution, are noted $\overline{X}$ and s^2 respectively, and are such that:

$$\boxed{\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i} \quad \text{and} \quad \boxed{s^2 = \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}$$

❒ **Central Limit Theorem** – Let us have a random sample $X_1, ..., X_n$ following a given distribution with mean μ and variance σ^2, then we have:

$$\boxed{\overline{X} \underset{n\to+\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)}$$

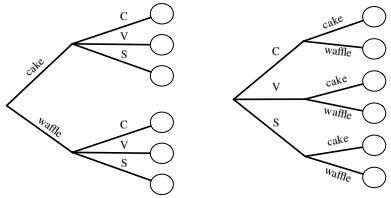
Probability Cheatsheet v2.0

Compiled by William Chen (http://wzchen.com) and Joe Blitzstein, with contributions from Sebastian Chiu, Yuan Jiang, Yuqi Hou, and Jessy Hwang. Material based on Joe Blitzstein's (@stat110) lectures (http://stat110.net) and Blitzstein/Hwang's Introduction to Probability textbook (http://bit.ly/introprobability). Please share comments, suggestions, and errors at http://github.com/wzchen/probability_cheatsheet.

Last Updated September 4, 2015

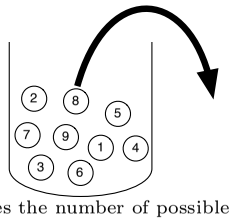
Counting

Multiplication Rule



Let's say we have a compound experiment (an experiment with multiple components). If the 1st component has n_1 possible outcomes, the 2nd component has n_2 possible outcomes, ..., and the rth component has n_r possible outcomes, then overall there are $n_1 n_2 \ldots n_r$ possibilities for the whole experiment.

Sampling Table



The sampling table gives the number of possible samples of size k out of a population of size n, under various assumptions about how the sample is collected.

	Order Matters	Not Matter
With Replacement	n^k	$\binom{n+k-1}{k}$
Without Replacement	$\frac{n!}{(n-k)!}$	$\binom{n}{k}$

Naive Definition of Probability

If all outcomes are equally likely, the probability of an event A happening is:

$$P_{\text{naive}}(A) = \frac{\text{number of outcomes favorable to } A}{\text{number of outcomes}}$$

Thinking Conditionally

Independence

Independent Events A and B are independent if knowing whether A occurred gives no information about whether B occurred. More formally, A and B (which have nonzero probability) are independent if and only if one of the following equivalent statements holds:

$$P(A \cap B) = P(A)P(B)$$
$$P(A|B) = P(A)$$
$$P(B|A) = P(B)$$

Conditional Independence A and B are conditionally independent given C if $P(A \cap B|C) = P(A|C)P(B|C)$. Conditional independence does not imply independence, and independence does not imply conditional independence.

Unions, Intersections, and Complements

De Morgan's Laws A useful identity that can make calculating probabilities of unions easier by relating them to intersections, and vice versa. Analogous results hold with more than two sets.

$$(A \cup B)^c = A^c \cap B^c$$
$$(A \cap B)^c = A^c \cup B^c$$

Joint, Marginal, and Conditional

Joint Probability $P(A \cap B)$ or $P(A, B)$ – Probability of A and B.

Marginal (Unconditional) Probability $P(A)$ – Probability of A.

Conditional Probability $P(A|B) = P(A, B)/P(B)$ – Probability of A, given that B occurred.

Conditional Probability *is* Probability $P(A|B)$ is a probability function for any fixed B. Any theorem that holds for probability also holds for conditional probability.

Probability of an Intersection or Union

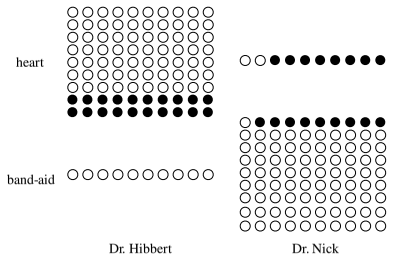
Intersections via Conditioning

$$P(A, B) = P(A)P(B|A)$$
$$P(A, B, C) = P(A)P(B|A)P(C|A, B)$$

Unions via Inclusion-Exclusion

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$
$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C).$$

Simpson's Paradox



It is possible to have

$$P(A \mid B, C) < P(A \mid B^c, C) \text{ and } P(A \mid B, C^c) < P(A \mid B^c, C^c)$$
$$\text{yet also } P(A \mid B) > P(A \mid B^c).$$

Law of Total Probability (LOTP)

Let $B_1, B_2, B_3, \ldots B_n$ be a *partition* of the sample space (i.e., they are disjoint and their union is the entire sample space).

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \cdots + P(A|B_n)P(B_n)$$
$$P(A) = P(A \cap B_1) + P(A \cap B_2) + \cdots + P(A \cap B_n)$$

For **LOTP with extra conditioning**, just add in another event C!

$$P(A|C) = P(A|B_1, C)P(B_1|C) + \cdots + P(A|B_n, C)P(B_n|C)$$
$$P(A|C) = P(A \cap B_1|C) + P(A \cap B_2|C) + \cdots + P(A \cap B_n|C)$$

Special case of LOTP with B and B^c as partition:

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$
$$P(A) = P(A \cap B) + P(A \cap B^c)$$

Bayes' Rule

Bayes' Rule, and with extra conditioning (just add in C!)

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

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

We can also write

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

Odds Form of Bayes' Rule

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

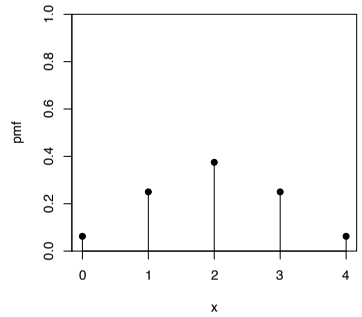
The *posterior odds* of A are the *likelihood ratio* times the *prior odds*.

Random Variables and their Distributions

PMF, CDF, and Independence

Probability Mass Function (PMF) Gives the probability that a *discrete* random variable takes on the value x.

$$p_X(x) = P(X = x)$$

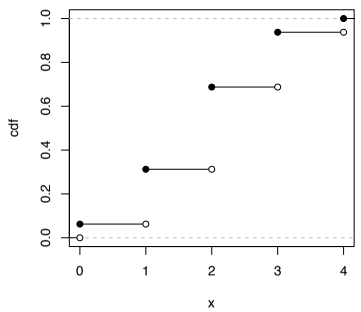


The PMF satisfies

$$p_X(x) \geq 0 \text{ and } \sum_x p_X(x) = 1$$

Cumulative Distribution Function (CDF) Gives the probability that a random variable is less than or equal to x.

$$F_X(x) = P(X \leq x)$$



The CDF is an increasing, right-continuous function with

$$F_X(x) \to 0 \text{ as } x \to -\infty \text{ and } F_X(x) \to 1 \text{ as } x \to \infty$$

Independence Intuitively, two random variables are independent if knowing the value of one gives no information about the other. Discrete r.v.s X and Y are independent if for *all* values of x and y

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

Expected Value and Indicators

Expected Value and Linearity

Expected Value (a.k.a. *mean*, *expectation*, or *average*) is a weighted average of the possible outcomes of our random variable. Mathematically, if $x_1, x_2, x_3, \ldots$ are all of the distinct possible values that X can take, the expected value of X is

$$E(X) = \sum_i x_i P(X = x_i)$$

X	Y	$X+Y$
3	4	7
2	2	4
6	8	14
10	23	33
1	−3	−2
1	0	1
5	9	14
4	1	5
...	...	...
$\frac{1}{n}\sum_{i=1}^{n} x_i$	$\frac{1}{n}\sum_{i=1}^{n} y_i$	$\frac{1}{n}\sum_{i=1}^{n}(x_i+y_i)$
$E(X)$	$E(Y)$	$E(X+Y)$

Linearity For any r.v.s X and Y, and constants a, b, c,

$$E(aX + bY + c) = aE(X) + bE(Y) + c$$

Same distribution implies same mean If X and Y have the same distribution, then $E(X) = E(Y)$ and, more generally,

$$E(g(X)) = E(g(Y))$$

Conditional Expected Value is defined like expectation, only conditioned on any event A.

$$E(X|A) = \sum_x x P(X = x|A)$$

Indicator Random Variables

Indicator Random Variable is a random variable that takes on the value 1 or 0. It is always an indicator of some event: if the event occurs, the indicator is 1; otherwise it is 0. They are useful for many problems about counting how many events of some kind occur. Write

$$I_A = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Note that $I_A^2 = I_A, I_A I_B = I_{A \cap B}$, and $I_{A \cup B} = I_A + I_B - I_A I_B$.

Distribution $I_A \sim \text{Bern}(p)$ where $p = P(A)$.

Fundamental Bridge The expectation of the indicator for event A is the probability of event A: $E(I_A) = P(A)$.

Variance and Standard Deviation

$$\text{Var}(X) = E\left(X - E(X)\right)^2 = E(X^2) - (E(X))^2$$

$$\text{SD}(X) = \sqrt{\text{Var}(X)}$$

Continuous RVs, LOTUS, UoU

Continuous Random Variables (CRVs)

What's the probability that a CRV is in an interval? Take the difference in CDF values (or use the PDF as described later).

$$P(a \leq X \leq b) = P(X \leq b) - P(X \leq a) = F_X(b) - F_X(a)$$

For $X \sim \mathcal{N}(\mu, \sigma^2)$, this becomes

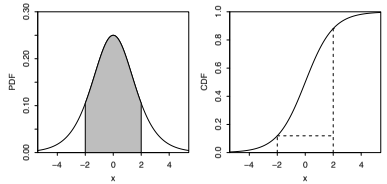
$$P(a \leq X \leq b) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)$$

What is the Probability Density Function (PDF)? The PDF f is the derivative of the CDF F.

$$F'(x) = f(x)$$

A PDF is nonnegative and integrates to 1. By the fundamental theorem of calculus, to get from PDF back to CDF we can integrate:

$$F(x) = \int_{-\infty}^{x} f(t)dt$$



To find the probability that a CRV takes on a value in an interval, integrate the PDF over that interval.

$$F(b) - F(a) = \int_a^b f(x)dx$$

How do I find the expected value of a CRV? Analogous to the discrete case, where you sum x times the PMF, for CRVs you integrate x times the PDF.

$$E(X) = \int_{-\infty}^{\infty} x f(x)dx$$

LOTUS

Expected value of a function of an r.v. The expected value of X is defined this way:

$$E(X) = \sum_x x P(X = x) \text{ (for discrete } X)$$

$$E(X) = \int_{-\infty}^{\infty} x f(x)dx \text{ (for continuous } X)$$

The **Law of the Unconscious Statistician (LOTUS)** states that you can find the expected value of a *function of a random variable*, $g(X)$, in a similar way, by replacing the x in front of the PMF/PDF by $g(x)$ but still working with the PMF/PDF of X:

$$E(g(X)) = \sum_x g(x) P(X = x) \text{ (for discrete } X)$$

$$E(g(X)) = \int_{-\infty}^{\infty} g(x) f(x)dx \text{ (for continuous } X)$$

What's a function of a random variable? A function of a random variable is also a random variable. For example, if X is the number of bikes you see in an hour, then $g(X) = 2X$ is the number of bike wheels you see in that hour and $h(X) = \binom{X}{2} = \frac{X(X-1)}{2}$ is the number of *pairs* of bikes such that you see both of those bikes in that hour.

What's the point? You don't need to know the PMF/PDF of $g(X)$ to find its expected value. All you need is the PMF/PDF of X.

Universality of Uniform (UoU)

When you plug any CRV into its own CDF, you get a Uniform(0,1) random variable. When you plug a Uniform(0,1) r.v. into an inverse CDF, you get an r.v. with that CDF. For example, let's say that a random variable X has CDF

$$F(x) = 1 - e^{-x}, \text{ for } x > 0$$

By UoU, if we plug X into this function then we get a uniformly distributed random variable.

$$F(X) = 1 - e^{-X} \sim \text{Unif}(0, 1)$$

Similarly, if $U \sim \text{Unif}(0, 1)$ then $F^{-1}(U)$ has CDF F. The key point is that for any continuous random variable X, we can transform it into a Uniform random variable and back by using its CDF.

Moments and MGFs

Moments

Moments describe the shape of a distribution. Let X have mean μ and standard deviation σ, and $Z = (X - \mu)/\sigma$ be the *standardized* version of X. The kth moment of X is $\mu_k = E(X^k)$ and the kth standardized moment of X is $m_k = E(Z^k)$. The mean, variance, skewness, and kurtosis are important summaries of the shape of a distribution.

Mean $E(X) = \mu_1$

Variance $\text{Var}(X) = \mu_2 - \mu_1^2$

Skewness $\text{Skew}(X) = m_3$

Kurtosis $\text{Kurt}(X) = m_4 - 3$

Moment Generating Functions

MGF For any random variable X, the function

$$M_X(t) = E(e^{tX})$$

is the **moment generating function (MGF)** of X, if it exists for all t in some open interval containing 0. The variable t could just as well have been called u or v. It's a bookkeeping device that lets us work with the *function* M_X rather than the *sequence* of moments.

Why is it called the Moment Generating Function? Because the kth derivative of the moment generating function, evaluated at 0, is the kth moment of X.

$$\mu_k = E(X^k) = M_X^{(k)}(0)$$

This is true by Taylor expansion of e^{tX} since

$$M_X(t) = E(e^{tX}) = \sum_{k=0}^{\infty} \frac{E(X^k)t^k}{k!} = \sum_{k=0}^{\infty} \frac{\mu_k t^k}{k!}$$

MGF of linear functions If we have $Y = aX + b$, then

$$M_Y(t) = E(e^{t(aX+b)}) = e^{bt}E(e^{(at)X}) = e^{bt}M_X(at)$$

Uniqueness *If it exists, the MGF uniquely determines the distribution.* This means that for any two random variables X and Y, they are distributed the same (their PMFs/PDFs are equal) if and only if their MGFs are equal.

Summing Independent RVs by Multiplying MGFs. If X and Y are independent, then

$$M_{X+Y}(t) = E(e^{t(X+Y)}) = E(e^{tX})E(e^{tY}) = M_X(t) \cdot M_Y(t)$$

The MGF of the sum of two random variables is the product of the MGFs of those two random variables.

Joint PDFs and CDFs

Joint Distributions

The **joint CDF** of X and Y is

$$F(x, y) = P(X \leq x, Y \leq y)$$

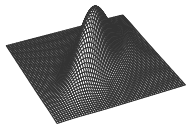
In the discrete case, X and Y have a **joint PMF**

$$p_{X,Y}(x, y) = P(X = x, Y = y).$$

In the continuous case, they have a **joint PDF**

$$f_{X,Y}(x, y) = \frac{\partial^2}{\partial x \partial y} F_{X,Y}(x, y).$$

The joint PMF/PDF must be nonnegative and sum/integrate to 1.



Conditional Distributions

Conditioning and Bayes' rule for discrete r.v.s

$$P(Y = y|X = x) = \frac{P(X = x, Y = y)}{P(X = x)} = \frac{P(X = x|Y = y)P(Y = y)}{P(X = x)}$$

Conditioning and Bayes' rule for continuous r.v.s

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x, y)}{f_X(x)} = \frac{f_{X|Y}(x|y)f_Y(y)}{f_X(x)}$$

Hybrid Bayes' rule

$$f_X(x|A) = \frac{P(A|X = x)f_X(x)}{P(A)}$$

Marginal Distributions

To find the distribution of one (or more) random variables from a joint PMF/PDF, sum/integrate over the unwanted random variables.

Marginal PMF from joint PMF

$$P(X = x) = \sum_y P(X = x, Y = y)$$

Marginal PDF from joint PDF

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y)dy$$

Independence of Random Variables

Random variables X and Y are independent if and only if any of the following conditions holds:

- Joint CDF is the product of the marginal CDFs
- Joint PMF/PDF is the product of the marginal PMFs/PDFs
- Conditional distribution of Y given X is the marginal distribution of Y

Write $X \perp\!\!\!\perp Y$ to denote that X and Y are independent.

Multivariate LOTUS

LOTUS in more than one dimension is analogous to the 1D LOTUS. For discrete random variables:

$$E(g(X, Y)) = \sum_x \sum_y g(x, y)P(X = x, Y = y)$$

For continuous random variables:

$$E(g(X, Y)) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y)f_{X,Y}(x, y)dxdy$$

Covariance and Transformations

Covariance and Correlation

Covariance is the analog of variance for two random variables.

$$\text{Cov}(X, Y) = E\left((X - E(X))(Y - E(Y))\right) = E(XY) - E(X)E(Y)$$

Note that

$$\text{Cov}(X, X) = E(X^2) - (E(X))^2 = \text{Var}(X)$$

Correlation is a standardized version of covariance that is always between -1 and 1.

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}}$$

Covariance and Independence If two random variables are independent, then they are uncorrelated. The converse is not necessarily true (e.g., consider $X \sim \mathcal{N}(0, 1)$ and $Y = X^2$).

$$X \perp\!\!\!\perp Y \longrightarrow \text{Cov}(X, Y) = 0 \longrightarrow E(XY) = E(X)E(Y)$$

Covariance and Variance The variance of a sum can be found by

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y)$$

$$\text{Var}(X_1 + X_2 + \cdots + X_n) = \sum_{i=1}^{n} \text{Var}(X_i) + 2\sum_{i<j} \text{Cov}(X_i, X_j)$$

If X and Y are independent then they have covariance 0, so

$$X \perp\!\!\!\perp Y \Longrightarrow \text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$$

If $X_1, X_2, \ldots, X_n$ are identically distributed and have the same covariance relationships (often by **symmetry**), then

$$\text{Var}(X_1 + X_2 + \cdots + X_n) = n\text{Var}(X_1) + 2\binom{n}{2}\text{Cov}(X_1, X_2)$$

Covariance Properties For random variables W, X, Y, Z and constants a, b:

$$\begin{aligned}
\text{Cov}(X, Y) &= \text{Cov}(Y, X) \\
\text{Cov}(X + a, Y + b) &= \text{Cov}(X, Y) \\
\text{Cov}(aX, bY) &= ab\text{Cov}(X, Y) \\
\text{Cov}(W + X, Y + Z) &= \text{Cov}(W, Y) + \text{Cov}(W, Z) + \text{Cov}(X, Y) \\
&\quad + \text{Cov}(X, Z)
\end{aligned}$$

Correlation is location-invariant and scale-invariant For any constants a, b, c, d with a and c nonzero,

$$\text{Corr}(aX + b, cY + d) = \text{Corr}(X, Y)$$

Transformations

One Variable Transformations Let's say that we have a random variable X with PDF $f_X(x)$, but we are also interested in some function of X. We call this function $Y = g(X)$. Also let $y = g(x)$. If g is differentiable and strictly increasing (or strictly decreasing), then the PDF of Y is

$$f_Y(y) = f_X(x)\left|\frac{dx}{dy}\right| = f_X(g^{-1}(y))\left|\frac{d}{dy}g^{-1}(y)\right|$$

The derivative of the inverse transformation is called the **Jacobian**.

Two Variable Transformations Similarly, let's say we know the joint PDF of U and V but are also interested in the random vector (X, Y) defined by $(X, Y) = g(U, V)$. Let

$$\frac{\partial(u, v)}{\partial(x, y)} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}$$

be the **Jacobian matrix**. If the entries in this matrix exist and are continuous, and the determinant of the matrix is never 0, then

$$f_{X,Y}(x, y) = f_{U,V}(u, v)\left|\left|\frac{\partial(u, v)}{\partial(x, y)}\right|\right|$$

The inner bars tells us to take the matrix's determinant, and the outer bars tell us to take the absolute value. In a 2×2 matrix,

$$\left|\left| \begin{matrix} a & b \\ c & d \end{matrix} \right|\right| = |ad - bc|$$

Convolutions

Convolution Integral If you want to find the PDF of the sum of two independent CRVs X and Y, you can do the following integral:

$$f_{X+Y}(t) = \int_{-\infty}^{\infty} f_X(x)f_Y(t - x)dx$$

Example Let $X, Y \sim \mathcal{N}(0, 1)$ be i.i.d. Then for each fixed t,

$$f_{X+Y}(t) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \frac{1}{\sqrt{2\pi}} e^{-(t-x)^2/2} dx$$

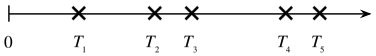
By completing the square and using the fact that a Normal PDF integrates to 1, this works out to $f_{X+Y}(t)$ being the $\mathcal{N}(0, 2)$ PDF.

Poisson Process

Definition We have a **Poisson process** of rate λ arrivals per unit time if the following conditions hold:

1. The number of arrivals in a time interval of length t is Pois(λt).
2. Numbers of arrivals in disjoint time intervals are independent.

For example, the numbers of arrivals in the time intervals $[0, 5]$, $(5, 12)$, and $[13, 23)$ are independent with Pois(5λ), Pois(7λ), Pois(10λ) distributions, respectively.



Count-Time Duality Consider a Poisson process of emails arriving in an inbox at rate λ emails per hour. Let T_n be the time of arrival of the nth email (relative to some starting time 0) and N_t be the number of emails that arrive in $[0, t]$. Let's find the distribution of T_1. The event $T_1 > t$, the event that you have to wait more than t hours to get the first email, is the same as the event $N_t = 0$, which is the event that there are no emails in the first t hours. So

$$P(T_1 > t) = P(N_t = 0) = e^{-\lambda t} \longrightarrow P(T_1 \leq t) = 1 - e^{-\lambda t}$$

Thus we have $T_1 \sim$ Expo(λ). By the memoryless property and similar reasoning, the interarrival times between emails are i.i.d. Expo(λ), i.e., the differences $T_n - T_{n-1}$ are i.i.d. Expo(λ).

Order Statistics

Definition Let's say you have n i.i.d. r.v.s $X_1, X_2, \ldots, X_n$. If you arrange them from smallest to largest, the ith element in that list is the ith order statistic, denoted $X_{(i)}$. So $X_{(1)}$ is the smallest in the list and $X_{(n)}$ is the largest in the list.

Note that the order statistics are *dependent*, e.g., learning $X_{(4)} = 42$ gives us the information that $X_{(1)}, X_{(2)}, X_{(3)}$ are ≤ 42 and $X_{(5)}, X_{(6)}, \ldots, X_{(n)}$ are ≥ 42.

Distribution Taking n i.i.d. random variables $X_1, X_2, \ldots, X_n$ with CDF $F(x)$ and PDF $f(x)$, the CDF and PDF of $X_{(i)}$ are:

$$F_{X_{(i)}}(x) = P(X_{(i)} \leq x) = \sum_{k=i}^{n} \binom{n}{k} F(x)^k (1 - F(x))^{n-k}$$

$$f_{X_{(i)}}(x) = n\binom{n-1}{i-1} F(x)^{i-1} (1 - F(x))^{n-i} f(x)$$

Uniform Order Statistics The jth order statistic of i.i.d. $U_1, \ldots, U_n \sim$ Unif$(0, 1)$ is $U_{(j)} \sim$ Beta$(j, n - j + 1)$.

Conditional Expectation

Conditioning on an Event We can find $E(Y|A)$, the expected value of Y given that event A occurred. A very important case is when A is the event $X = x$. Note that $E(Y|A)$ is a *number*. For example:

- The expected value of a fair die roll, given that it is prime, is $\frac{1}{3} \cdot 2 + \frac{1}{3} \cdot 3 + \frac{1}{3} \cdot 5 = \frac{10}{3}$.
- Let Y be the number of successes in 10 independent Bernoulli trials with probability p of success. Let A be the event that the first 3 trials are all successes. Then

  $$E(Y|A) = 3 + 7p$$

  since the number of successes among the last 7 trials is Bin$(7, p)$.
- Let $T \sim$ Expo$(1/10)$ be how long you have to wait until the shuttle comes. Given that you have already waited t minutes, the expected additional waiting time is 10 more minutes, by the memoryless property. That is, $E(T|T > t) = t + 10$.

Discrete Y	Continuous Y
$E(Y) = \sum_y yP(Y = y)$	$E(Y) = \int_{-\infty}^{\infty} yf_Y(y)dy$
$E(Y\|A) = \sum_y yP(Y = y\|A)$	$E(Y\|A) = \int_{-\infty}^{\infty} yf(y\|A)dy$

Conditioning on a Random Variable We can also find $E(Y|X)$, the expected value of Y given the random variable X. This is *a function of the random variable* X. It is *not* a number except in certain special cases such as if $X \perp\!\!\!\perp Y$. To find $E(Y|X)$, find $E(Y|X = x)$ and then plug in X for x. For example:

- If $E(Y|X = x) = x^3 + 5x$, then $E(Y|X) = X^3 + 5X$.
- Let Y be the number of successes in 10 independent Bernoulli trials with probability p of success and X be the number of successes among the first 3 trials. Then $E(Y|X) = X + 7p$.
- Let $X \sim \mathcal{N}(0, 1)$ and $Y = X^2$. Then $E(Y|X = x) = x^2$ since if we know $X = x$ then we know $Y = x^2$. And $E(X|Y = y) = 0$ since if we know $Y = y$ then we know $X = \pm\sqrt{y}$, with equal probabilities (by symmetry). So $E(Y|X) = X^2, E(X|Y) = 0$.

Properties of Conditional Expectation

1. $E(Y|X) = E(Y)$ if $X \perp\!\!\!\perp Y$
2. $E(h(X)W|X) = h(X)E(W|X)$ (**taking out what's known**) In particular, $E(h(X)|X) = h(X)$.
3. $E(E(Y|X)) = E(Y)$ (**Adam's Law**, a.k.a. Law of Total Expectation)

Adam's Law (a.k.a. Law of Total Expectation) can also be written in a way that looks analogous to LOTP. For any events $A_1, A_2, \ldots, A_n$ that partition the sample space,

$$E(Y) = E(Y|A_1)P(A_1) + \cdots + E(Y|A_n)P(A_n)$$

For the special case where the partition is A, A^c, this says

$$E(Y) = E(Y|A)P(A) + E(Y|A^c)P(A^c)$$

Eve's Law (a.k.a. Law of Total Variance)

$$\text{Var}(Y) = E(\text{Var}(Y|X)) + \text{Var}(E(Y|X))$$

MVN, LLN, CLT

Law of Large Numbers (LLN)

Let $X_1, X_2, X_3 \ldots$ be i.i.d. with mean μ. The **sample mean** is

$$\bar{X}_n = \frac{X_1 + X_2 + X_3 + \cdots + X_n}{n}$$

The **Law of Large Numbers** states that as $n \to \infty$, $\bar{X}_n \to \mu$ with probability 1. For example, in flips of a coin with probability p of Heads, let X_j be the indicator of the jth flip being Heads. Then LLN says the proportion of Heads converges to p (with probability 1).

Central Limit Theorem (CLT)

Approximation using CLT

We use $\dot{\sim}$ to denote *is approximately distributed*. We can use the **Central Limit Theorem** to approximate the distribution of a random variable $Y = X_1 + X_2 + \cdots + X_n$ that is a sum of n i.i.d. random variables X_i. Let $E(Y) = \mu_Y$ and $\text{Var}(Y) = \sigma_Y^2$. The CLT says

$$Y \dot{\sim} \mathcal{N}(\mu_Y, \sigma_Y^2)$$

If the X_i are i.i.d. with mean μ_X and variance σ_X^2, then $\mu_Y = n\mu_X$ and $\sigma_Y^2 = n\sigma_X^2$. For the sample mean $\bar{X}_n$, the CLT says

$$\bar{X}_n = \frac{1}{n}(X_1 + X_2 + \cdots + X_n) \dot{\sim} \mathcal{N}(\mu_X, \sigma_X^2/n)$$

Asymptotic Distributions using CLT

We use $\xrightarrow{D}$ to denote *converges in distribution to* as $n \to \infty$. The CLT says that if we standardize the sum $X_1 + \cdots + X_n$ then the distribution of the sum converges to $\mathcal{N}(0, 1)$ as $n \to \infty$:

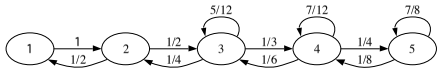
$$\frac{1}{\sigma\sqrt{n}}(X_1 + \cdots + X_n - n\mu_X) \xrightarrow{D} \mathcal{N}(0, 1)$$

In other words, the CDF of the left-hand side goes to the standard Normal CDF, Φ. In terms of the sample mean, the CLT says

$$\frac{\sqrt{n}(\bar{X}_n - \mu_X)}{\sigma_X} \xrightarrow{D} \mathcal{N}(0, 1)$$

Markov Chains

Definition



A Markov chain is a random walk in a **state space**, which we will assume is finite, say $\{1, 2, \ldots, M\}$. We let X_t denote which element of the state space the walk is visiting at time t. The Markov chain is the sequence of random variables tracking where the walk is at all points in time, $X_0, X_1, X_2, \ldots$. By definition, a Markov chain must satisfy the **Markov property**, which says that if you want to predict where the chain will be at a future time, if we know the present state then the entire past history is irrelevant. *Given the present, the past and future are conditionally independent.* In symbols,

$$P(X_{n+1} = j|X_0 = i_0, X_1 = i_1, \ldots, X_n = i) = P(X_{n+1} = j|X_n = i)$$

State Properties

A state is either recurrent or transient.

- If you start at a **recurrent state**, then you will always return back to that state at some point in the future. ♪*You can check-out any time you like, but you can never leave.* ♪
- Otherwise you are at a **transient state**. There is some positive probability that once you leave you will never return. ♪*You don't have to go home, but you can't stay here.* ♪

A state is either periodic or aperiodic.

- If you start at a **periodic state** of period k, then the GCD of the possible numbers of steps it would take to return back is $k > 1$.
- Otherwise you are at an **aperiodic state**. The GCD of the possible numbers of steps it would take to return back is 1.

Transition Matrix

Let the state space be $\{1, 2, \ldots, M\}$. The transition matrix Q is the $M \times M$ matrix where element q_{ij} is the probability that the chain goes from state i to state j in one step:

$$q_{ij} = P(X_{n+1} = j | X_n = i)$$

To find the probability that the chain goes from state i to state j in exactly m steps, take the (i, j) element of Q^m.

$$q_{ij}^{(m)} = P(X_{n+m} = j | X_n = i)$$

If X_0 is distributed according to the row vector PMF $\vec{p}$, i.e., $p_j = P(X_0 = j)$, then the PMF of X_n is $\vec{p}Q^n$.

Chain Properties

A chain is **irreducible** if you can get from anywhere to anywhere. If a chain (on a finite state space) is irreducible, then all of its states are recurrent. A chain is **periodic** if any of its states are periodic, and is **aperiodic** if none of its states are periodic. In an irreducible chain, all states have the same period.

A chain is **reversible** with respect to $\vec{s}$ if $s_i q_{ij} = s_j q_{ji}$ for all i, j. Examples of reversible chains include any chain with $q_{ij} = q_{ji}$, with $\vec{s} = (\frac{1}{M}, \frac{1}{M}, \ldots, \frac{1}{M})$, and random walk on an undirected network.

Stationary Distribution

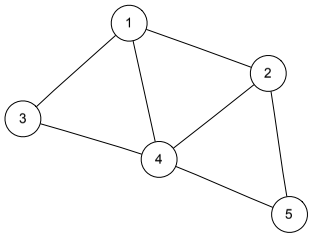
Let us say that the vector $\vec{s} = (s_1, s_2, \ldots, s_M)$ be a PMF (written as a row vector). We will call $\vec{s}$ the **stationary distribution** for the chain if $\vec{s}Q = \vec{s}$. As a consequence, if X_t has the stationary distribution, then all future $X_{t+1}, X_{t+2}, \ldots$ also have the stationary distribution.

For irreducible, aperiodic chains, the stationary distribution exists, is unique, and s_i is the long-run probability of a chain being at state i. The expected number of steps to return to i starting from i is $1/s_i$.

To find the stationary distribution, you can solve the matrix equation $(Q' - I)\vec{s}' = 0$. The stationary distribution is uniform if the columns of Q sum to 1.

Reversibility Condition Implies Stationarity If you have a PMF $\vec{s}$ and a Markov chain with transition matrix Q, then $s_i q_{ij} = s_j q_{ji}$ for all states i, j implies that $\vec{s}$ is stationary.

Random Walk on an Undirected Network



If you have a collection of **nodes**, pairs of which can be connected by undirected **edges**, and a Markov chain is run by going from the current node to a uniformly random node that is connected to it by an edge, then this is a random walk on an undirected network. The stationary distribution of this chain is proportional to the **degree sequence** (this is the sequence of degrees, where the degree of a node is how many edges are attached to it). For example, the stationary distribution of random walk on the network shown above is proportional to $(3, 3, 2, 4, 2)$, so it's $(\frac{3}{14}, \frac{3}{14}, \frac{3}{14}, \frac{4}{14}, \frac{2}{14})$.

Continuous Distributions

Uniform Distribution

Let us say that U is distributed Unif(a, b). We know the following:

Properties of the Uniform For a Uniform distribution, the probability of a draw from any interval within the support is proportional to the length of the interval. See *Universality of Uniform* and *Order Statistics* for other properties.

Example William throws darts really badly, so his darts are uniform over the whole room because they're equally likely to appear anywhere. William's darts have a Uniform distribution on the surface of the room. The Uniform is the only distribution where the probability of hitting in any specific region is proportional to the length/area/volume of that region, and where the density of occurrence in any one specific spot is constant throughout the whole support.

Normal Distribution

Let us say that X is distributed $\mathcal{N}(\mu, \sigma^2)$. We know the following:

Central Limit Theorem The Normal distribution is ubiquitous because of the Central Limit Theorem, which states that the sample mean of i.i.d. r.v.s will approach a Normal distribution as the sample size grows, regardless of the initial distribution.

Location-Scale Transformation Every time we shift a Normal r.v. (by adding a constant) or rescale a Normal (by multiplying by a constant), we change it to another Normal r.v. For any Normal $X \sim \mathcal{N}(\mu, \sigma^2)$, we can transform it to the standard $\mathcal{N}(0, 1)$ by the following transformation:

$$Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1)$$

Standard Normal The Standard Normal, $Z \sim \mathcal{N}(0, 1)$, has mean 0 and variance 1. Its CDF is denoted by Φ.

Exponential Distribution

Let us say that X is distributed Expo(λ). We know the following:

Story You're sitting on an open meadow right before the break of dawn, wishing that airplanes in the night sky were shooting stars, because you could really use a wish right now. You know that shooting stars come on average every 15 minutes, but a shooting star is not "due" to come just because you've waited so long. Your waiting time is memoryless; the additional time until the next shooting star comes does not depend on how long you've waited already.

Example The waiting time until the next shooting star is distributed Expo(4) hours. Here $\lambda = 4$ is the **rate parameter**, since shooting stars arrive at a rate of 1 per 1/4 hour on average. The expected time until the next shooting star is $1/\lambda = 1/4$ hour.

Expos as a rescaled Expo(1)

$$Y \sim \text{Expo}(\lambda) \to X = \lambda Y \sim \text{Expo}(1)$$

Memorylessness The Exponential Distribution is the only continuous memoryless distribution. The memoryless property says that for $X \sim \text{Expo}(\lambda)$ and any positive numbers s and t,

$$P(X > s + t | X > s) = P(X > t)$$

Equivalently,

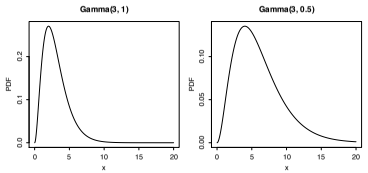
$$X - a | (X > a) \sim \text{Expo}(\lambda)$$

For example, a product with an Expo(λ) lifetime is always "as good as new" (it doesn't experience wear and tear). Given that the product has survived a years, the additional time that it will last is still Expo(λ).

Min of Expos If we have independent $X_i \sim \text{Expo}(\lambda_i)$, then $\min(X_1, \ldots, X_k) \sim \text{Expo}(\lambda_1 + \lambda_2 + \cdots + \lambda_k)$.

Max of Expos If we have i.i.d. $X_i \sim \text{Expo}(\lambda)$, then $\max(X_1, \ldots, X_k)$ has the same distribution as $Y_1 + Y_2 + \cdots + Y_k$, where $Y_j \sim \text{Expo}(j\lambda)$ and the Y_j are independent.

Gamma Distribution

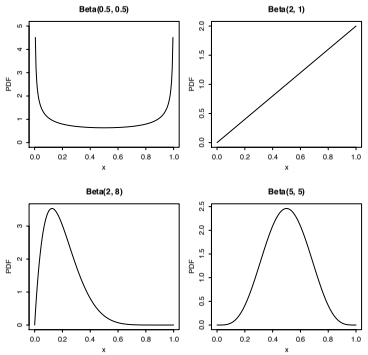


Let us say that X is distributed Gamma(a, λ). We know the following:

Story You sit waiting for shooting stars, where the waiting time for a star is distributed Expo(λ). You want to see n shooting stars before you go home. The total waiting time for the nth shooting star is Gamma(n, λ).

Example You are at a bank, and there are 3 people ahead of you. The serving time for each person is Exponential with mean 2 minutes. Only one person at a time can be served. The distribution of your waiting time until it's your turn to be served is Gamma$(3, \frac{1}{2})$.

Beta Distribution



Conjugate Prior of the Binomial In the Bayesian approach to statistics, parameters are viewed as random variables, to reflect our uncertainty. The *prior* for a parameter is its distribution before observing data. The *posterior* is the distribution for the parameter after observing data. Beta is the *conjugate* prior of the Binomial because if you have a Beta-distributed prior on p in a Binomial, then the posterior distribution on p given the Binomial data is also Beta-distributed. Consider the following two-level model:

$$X | p \sim \text{Bin}(n, p)$$
$$p \sim \text{Beta}(a, b)$$

Then after observing $X = x$, we get the posterior distribution

$$p | (X = x) \sim \text{Beta}(a + x, b + n - x)$$

Order statistics of the Uniform See *Order Statistics*.

Beta-Gamma relationship If $X \sim \text{Gamma}(a, \lambda)$, $Y \sim \text{Gamma}(b, \lambda)$, with $X \perp\!\!\!\perp Y$ then

- $\frac{X}{X+Y} \sim \text{Beta}(a, b)$
- $X + Y \perp\!\!\!\perp \frac{X}{X+Y}$

This is known as the **bank–post office result**.

χ^2 (Chi-Square) Distribution

Let us say that X is distributed χ^2_n. We know the following:

Story A Chi-Square(n) is the sum of the squares of n independent standard Normal r.v.s.

Properties and Representations

$$X \text{ is distributed as } Z_1^2 + Z_2^2 + \cdots + Z_n^2 \text{ for i.i.d. } Z_i \sim \mathcal{N}(0,1)$$

$$X \sim \text{Gamma}(n/2, 1/2)$$

Discrete Distributions

Distributions for four sampling schemes

	Replace	No Replace
Fixed # trials (n)	Binomial (Bern if $n = 1$)	HGeom
Draw until r success	NBin (Geom if $r = 1$)	NHGeom

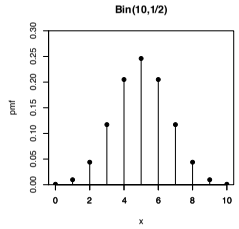
Bernoulli Distribution

The Bernoulli distribution is the simplest case of the Binomial distribution, where we only have one trial ($n = 1$). Let us say that X is distributed Bern(p). We know the following:

Story A trial is performed with probability p of "success", and X is the indicator of success: 1 means success, 0 means failure.

Example Let X be the indicator of Heads for a fair coin toss. Then $X \sim \text{Bern}(\frac{1}{2})$. Also, $1 - X \sim \text{Bern}(\frac{1}{2})$ is the indicator of Tails.

Binomial Distribution



Let us say that X is distributed Bin(n, p). We know the following:

Story X is the number of "successes" that we will achieve in n independent trials, where each trial is either a success or a failure, each with the same probability p of success. We can also write X as a sum of multiple independent Bern(p) random variables. Let $X \sim \text{Bin}(n, p)$ and $X_j \sim \text{Bern}(p)$, where all of the Bernoullis are independent. Then

$$X = X_1 + X_2 + X_3 + \cdots + X_n$$

Example If Jeremy Lin makes 10 free throws and each one independently has a $\frac{3}{4}$ chance of getting in, then the number of free throws he makes is distributed Bin(10, $\frac{3}{4}$).

Properties Let $X \sim \text{Bin}(n, p), Y \sim \text{Bin}(m, p)$ with $X \perp\!\!\!\perp Y$.

- **Redefine success** $n - X \sim \text{Bin}(n, 1 - p)$
- **Sum** $X + Y \sim \text{Bin}(n + m, p)$
- **Conditional** $X|(X + Y = r) \sim \text{HGeom}(n, m, r)$
- **Binomial-Poisson Relationship** Bin(n, p) is approximately Pois(λ) if p is small.
- **Binomial-Normal Relationship** Bin(n, p) is approximately $\mathcal{N}(np, np(1-p))$ if n is large and p is not near 0 or 1.

Geometric Distribution

Let us say that X is distributed Geom(p). We know the following:

Story X is the number of "failures" that we will achieve before we achieve our first success. Our successes have probability p.

Example If each pokeball we throw has probability $\frac{1}{10}$ to catch Mew, the number of failed pokeballs will be distributed Geom($\frac{1}{10}$).

First Success Distribution

Equivalent to the Geometric distribution, except that it includes the first success in the count. This is 1 more than the number of failures. If $X \sim \text{FS}(p)$ then $E(X) = 1/p$.

Negative Binomial Distribution

Let us say that X is distributed NBin(r, p). We know the following:

Story X is the number of "failures" that we will have before we achieve our rth success. Our successes have probability p.

Example Thundershock has 60% accuracy and can faint a wild Raticate in 3 hits. The number of misses before Pikachu faints Raticate with Thundershock is distributed NBin(3, 0.6).

Hypergeometric Distribution

Let us say that X is distributed HGeom(w, b, n). We know the following:

Story In a population of w desired objects and b undesired objects, X is the number of "successes" we will have in a draw of n objects, without replacement. The draw of n objects is assumed to be a **simple random sample** (all sets of n objects are equally likely).

Examples Here are some HGeom examples.

- Let's say that we have only b Weedles (failure) and w Pikachus (success) in Viridian Forest. We encounter n Pokemon in the forest, and X is the number of Pikachus in our encounters.
- The number of Aces in a 5 card hand.
- You have w white balls and b black balls, and you draw n balls. You will draw X white balls.
- You have w white balls and b black balls, and you draw n balls without replacement. The number of white balls in your sample is HGeom(w, b, n); the number of black balls is HGeom(b, w, n).
- **Capture-recapture** A forest has N elk, you capture n of them, tag them, and release them. Then you recapture a new sample of size m. How many tagged elk are now in the new sample? HGeom($n, N - n, m$)

Poisson Distribution

Let us say that X is distributed Pois(λ). We know the following:

Story There are rare events (low probability events) that occur many different ways (high possibilities of occurences) at an average rate of λ occurrences per unit space or time. The number of events that occur in that unit of space or time is X.

Example A certain busy intersection has an average of 2 accidents per month. Since an accident is a low probability event that can happen many different ways, it is reasonable to model the number of accidents in a month at that intersection as Pois(2). Then the number of accidents that happen in two months at that intersection is distributed Pois(4).

Properties Let $X \sim \text{Pois}(\lambda_1)$ and $Y \sim \text{Pois}(\lambda_2)$, with $X \perp\!\!\!\perp Y$.

1. **Sum** $X + Y \sim \text{Pois}(\lambda_1 + \lambda_2)$
2. **Conditional** $X|(X + Y = n) \sim \text{Bin}\left(n, \frac{\lambda_1}{\lambda_1+\lambda_2}\right)$
3. **Chicken-egg** If there are $Z \sim \text{Pois}(\lambda)$ items and we randomly and independently "accept" each item with probability p, then the number of accepted items $Z_1 \sim \text{Pois}(\lambda p)$, and the number of rejected items $Z_2 \sim \text{Pois}(\lambda(1-p))$, and $Z_1 \perp\!\!\!\perp Z_2$.

Multivariate Distributions

Multinomial Distribution

Let us say that the vector $\vec{X} = (X_1, X_2, X_3, \ldots, X_k) \sim \text{Mult}_k(n, \vec{p})$ where $\vec{p} = (p_1, p_2, \ldots, p_k)$.

Story We have n items, which can fall into any one of the k buckets independently with the probabilities $\vec{p} = (p_1, p_2, \ldots, p_k)$.

Example Let us assume that every year, 100 students in the Harry Potter Universe are randomly and independently sorted into one of four houses with equal probability. The number of people in each of the houses is distributed $\text{Mult}_4(100, \vec{p})$, where $\vec{p} = (0.25, 0.25, 0.25, 0.25)$. Note that $X_1 + X_2 + \cdots + X_4 = 100$, and they are dependent.

Joint PMF For $n = n_1 + n_2 + \cdots + n_k$,

$$P(\vec{X} = \vec{n}) = \frac{n!}{n_1! n_2! \ldots n_k!} p_1^{n_1} p_2^{n_2} \ldots p_k^{n_k}$$

Marginal PMF, Lumping, and Conditionals Marginally, $X_i \sim \text{Bin}(n, p_i)$ since we can define "success" to mean category i. If you lump together multiple categories in a Multinomial, then it is still Multinomial. For example, $X_i + X_j \sim \text{Bin}(n, p_i + p_j)$ for $i \neq j$ since we can define "success" to mean being in category i or j. Similarly, if $k = 6$ and we lump categories 1-2 and lump categories 3-5, then

$$(X_1 + X_2, X_3 + X_4 + X_5, X_6) \sim \text{Mult}_3(n, (p_1 + p_2, p_3 + p_4 + p_5, p_6))$$

Conditioning on some X_j also still gives a Multinomial:

$$X_1, \ldots, X_{k-1} | X_k = n_k \sim \text{Mult}_{k-1}\left(n - n_k, \left(\frac{p_1}{1-p_k}, \ldots, \frac{p_{k-1}}{1-p_k}\right)\right)$$

Variances and Covariances We have $X_i \sim \text{Bin}(n, p_i)$ marginally, so $\text{Var}(X_i) = np_i(1-p_i)$. Also, $\text{Cov}(X_i, X_j) = -np_ip_j$ for $i \neq j$.

Multivariate Uniform Distribution

See the univariate Uniform for stories and examples. For the 2D Uniform on some region, probability is proportional to area. Every point in the support has equal density, of value $\frac{1}{\text{area of region}}$. For the 3D Uniform, probability is proportional to volume.

Multivariate Normal (MVN) Distribution

A vector $\vec{X} = (X_1, X_2, \ldots, X_k)$ is Multivariate Normal if every linear combination is Normally distributed, i.e., $t_1X_1 + t_2X_2 + \cdots + t_kX_k$ is Normal for any constants $t_1, t_2, \ldots, t_k$. The parameters of the Multivariate Normal are the **mean vector** $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_k)$ and the **covariance matrix** where the (i, j) entry is $\text{Cov}(X_i, X_j)$.

Properties The Multivariate Normal has the following properties.

- Any subvector is also MVN.
- If any two elements within an MVN are uncorrelated, then they are independent.
- The joint PDF of a Bivariate Normal (X, Y) with $\mathcal{N}(0, 1)$ marginal distributions and correlation $\rho \in (-1, 1)$ is

$$f_{X,Y}(x, y) = \frac{1}{2\pi\tau} \exp\left(-\frac{1}{2\tau^2}(x^2 + y^2 - 2\rho xy)\right),$$

with $\tau = \sqrt{1 - \rho^2}$.

Distribution Properties

Important CDFs

Standard Normal Φ

Exponential(λ) $F(x) = 1 - e^{-\lambda x}$, for $x \in (0, \infty)$

Uniform(0,1) $F(x) = x$, for $x \in (0, 1)$

Convolutions of Random Variables

A convolution of n random variables is simply their sum. For the following results, let X and Y be *independent*.

1. $X \sim \text{Pois}(\lambda_1)$, $Y \sim \text{Pois}(\lambda_2) \longrightarrow X + Y \sim \text{Pois}(\lambda_1 + \lambda_2)$
2. $X \sim \text{Bin}(n_1, p)$, $Y \sim \text{Bin}(n_2, p) \longrightarrow X + Y \sim \text{Bin}(n_1 + n_2, p)$. $\text{Bin}(n, p)$ can be thought of as a sum of i.i.d. $\text{Bern}(p)$ r.v.s.
3. $X \sim \text{Gamma}(a_1, \lambda)$, $Y \sim \text{Gamma}(a_2, \lambda)$ $\longrightarrow X + Y \sim \text{Gamma}(a_1 + a_2, \lambda)$. $\text{Gamma}(n, \lambda)$ with n an integer can be thought of as a sum of i.i.d. $\text{Expo}(\lambda)$ r.v.s.
4. $X \sim \text{NBin}(r_1, p)$, $Y \sim \text{NBin}(r_2, p)$ $\longrightarrow X + Y \sim \text{NBin}(r_1 + r_2, p)$. $\text{NBin}(r, p)$ can be thought of as a sum of i.i.d. $\text{Geom}(p)$ r.v.s.
5. $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$, $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$ $\longrightarrow X + Y \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$

Special Cases of Distributions

1. $\text{Bin}(1, p) \sim \text{Bern}(p)$
2. $\text{Beta}(1, 1) \sim \text{Unif}(0, 1)$
3. $\text{Gamma}(1, \lambda) \sim \text{Expo}(\lambda)$
4. $\chi^2_n \sim \text{Gamma}\left(\frac{n}{2}, \frac{1}{2}\right)$
5. $\text{NBin}(1, p) \sim \text{Geom}(p)$

Inequalities

1. **Cauchy-Schwarz** $|E(XY)| \leq \sqrt{E(X^2)E(Y^2)}$
2. **Markov** $P(X \geq a) \leq \frac{E|X|}{a}$ for $a > 0$
3. **Chebyshev** $P(|X - \mu| \geq a) \leq \frac{\sigma^2}{a^2}$ for $E(X) = \mu, \text{Var}(X) = \sigma^2$
4. **Jensen** $E(g(X)) \geq g(E(X))$ for g convex; reverse if g is concave

Formulas

Geometric Series

$$1 + r + r^2 + \cdots + r^{n-1} = \sum_{k=0}^{n-1} r^k = \frac{1 - r^n}{1 - r}$$

$$1 + r + r^2 + \cdots = \frac{1}{1 - r} \text{ if } |r| < 1$$

Exponential Function (e^x)

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

Gamma and Beta Integrals

You can sometimes solve complicated-looking integrals by pattern-matching to a gamma or beta integral:

$$\int_0^\infty x^{t-1} e^{-x}\, dx = \Gamma(t) \qquad \int_0^1 x^{a-1}(1-x)^{b-1}\, dx = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Also, $\Gamma(a + 1) = a\Gamma(a)$, and $\Gamma(n) = (n - 1)!$ if n is a positive integer.

Euler's Approximation for Harmonic Sums

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \log n + 0.577\ldots$$

Stirling's Approximation for Factorials

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Miscellaneous Definitions

Medians and Quantiles Let X have CDF F. Then X has median m if $F(m) \geq 0.5$ and $P(X \geq m) \geq 0.5$. For X continuous, m satisfies $F(m) = 1/2$. In general, the ath quantile of X is $\min\{x : F(x) \geq a\}$; the median is the case $a = 1/2$.

log Statisticians generally use log to refer to natural log (i.e., base e).

i.i.d r.v.s Independent, identically-distributed random variables.

Example Problems

Contributions from Sebastian Chiu

Calculating Probability

A textbook has n typos, which are randomly scattered amongst its n pages, independently. You pick a random page. What is the probability that it has no typos? **Answer:** There is a $\left(1 - \frac{1}{n}\right)$ probability that any specific typo isn't on your page, and thus a $\boxed{\left(1 - \frac{1}{n}\right)^n}$ probability that there are no typos on your page. For n large, this is approximately $e^{-1} = 1/e$.

Linearity and Indicators (1)

In a group of n people, what is the expected number of distinct birthdays (month and day)? What is the expected number of birthday matches? **Answer:** Let X be the number of distinct birthdays and I_j be the indicator for the jth day being represented.

$$E(I_j) = 1 - P(\text{no one born on day } j) = 1 - (364/365)^n$$

By linearity, $\boxed{E(X) = 365\left(1 - (364/365)^n\right)}$. Now let Y be the number of birthday matches and J_i be the indicator that the ith pair of people have the same birthday. The probability that any two specific people share a birthday is $1/365$, so $\boxed{E(Y) = \binom{n}{2}/365}$.

Linearity and Indicators (2)

This problem is commonly known as the ***hat-matching problem****.* There are n people at a party, each with hat. At the end of the party, they each leave with a random hat. What is the expected number of people who leave with the right hat? **Answer:** Each hat has a $1/n$ chance of going to the right person. By linearity, the average number of hats that go to their owners is $\boxed{n(1/n) = 1}$.

Linearity and First Success

This problem is commonly known as the ***coupon collector problem****.* There are n coupon types. At each draw, you get a uniformly random coupon type. What is the expected number of coupons needed until you have a complete set? **Answer:** Let N be the number of coupons needed; we want $E(N)$. Let $N = N_1 + \cdots + N_n$, where N_1 is the draws to get our first new coupon, N_2 is the *additional* draws needed to draw our second new coupon and so on. By the story of the First Success, $N_2 \sim \text{FS}((n-1)/n)$ (after collecting first coupon type, there's $(n-1)/n$ chance you'll get something new). Similarly, $N_3 \sim \text{FS}((n-2)/n)$, and $N_j \sim \text{FS}((n-j+1)/n)$. By linearity,

$$E(N) = E(N_1) + \cdots + E(N_n) = \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} = \boxed{n\sum_{j=1}^{n} \frac{1}{j}}$$

This is approximately $n(\log(n) + 0.577)$ by Euler's approximation.

Orderings of i.i.d. random variables

I call 2 UberX's and 3 Lyfts at the same time. If the time it takes for the rides to reach me are i.i.d., what is the probability that all the Lyfts will arrive first? **Answer:** Since the arrival times of the five cars are i.i.d., all 5! orderings of the arrivals are equally likely. There are 3!2! orderings that involve the Lyfts arriving first, so the probability that the Lyfts arrive first is $\boxed{\frac{3!2!}{5!} = 1/10}$. Alternatively, there are $\binom{5}{3}$ ways to choose 3 of the 5 slots for the Lyfts to occupy, where each of the choices are equally likely. One of these choices has all 3 of the Lyfts arriving first, so the probability is $\boxed{1/\binom{5}{3} = 1/10}$.

Expectation of Negative Hypergeometric

What is the expected number of cards that you draw before you pick your first Ace in a shuffled deck (not counting the Ace)? **Answer:** Consider a non-Ace. Denote this to be card j. Let I_j be the indicator that card j will be drawn before the first Ace. Note that $I_j = 1$ says that j is before all 4 of the Aces in the deck. The probability that this occurs is $1/5$ by symmetry. Let X be the number of cards drawn before the first Ace. Then $X = I_1 + I_2 + \ldots + I_{48}$, where each indicator corresponds to one of the 48 non-Aces. Thus,

$$E(X) = E(I_1) + E(I_2) + \ldots + E(I_{48}) = 48/5 = \boxed{9.6}.$$

Minimum and Maximum of RVs

What is the CDF of the maximum of n independent Unif(0,1) random variables? **Answer:** Note that for r.v.s $X_1, X_2, \ldots, X_n$,

$$P(\min(X_1, X_2, \ldots, X_n) \geq a) = P(X_1 \geq a, X_2 \geq a, \ldots, X_n \geq a)$$

Similarly,

$$P(\max(X_1, X_2, \ldots, X_n) \leq a) = P(X_1 \leq a, X_2 \leq a, \ldots, X_n \leq a)$$

We will use this principle to find the CDF of $U_{(n)}$, where $U_{(n)} = \max(U_1, U_2, \ldots, U_n)$ and $U_i \sim \text{Unif}(0, 1)$ are i.i.d.

$$\begin{aligned} P(\max(U_1, U_2, \ldots, U_n) \leq a) &= P(U_1 \leq a, U_2 \leq a, \ldots, U_n \leq a) \\ &= P(U_1 \leq a)P(U_2 \leq a)\ldots P(U_n \leq a) \\ &= \boxed{a^n} \end{aligned}$$

for $0 < a < 1$ (and the CDF is 0 for $a \leq 0$ and 1 for $a \geq 1$).

Pattern-matching with e^x Taylor series

For $X \sim \text{Pois}(\lambda)$, find $E\left(\frac{1}{X+1}\right)$. **Answer:** By LOTUS,

$$E\left(\frac{1}{X+1}\right) = \sum_{k=0}^{\infty} \frac{1}{k+1} \frac{e^{-\lambda}\lambda^k}{k!} = \frac{e^{-\lambda}}{\lambda} \sum_{k=0}^{\infty} \frac{\lambda^{k+1}}{(k+1)!} = \boxed{\frac{e^{-\lambda}}{\lambda}(e^\lambda - 1)}$$

Adam's Law and Eve's Law

William really likes speedsolving Rubik's Cubes. But he's pretty bad at it, so sometimes he fails. On any given day, William will attempt $N \sim \text{Geom}(s)$ Rubik's Cubes. Suppose each time, he has probability p of solving the cube, independently. Let T be the number of Rubik's Cubes he solves during a day. Find the mean and variance of T.
Answer: Note that $T|N \sim \text{Bin}(N,p)$. So by Adam's Law,

$$E(T) = E(E(T|N)) = E(Np) = \boxed{\frac{p(1-s)}{s}}$$

Similarly, by Eve's Law, we have that

$$\text{Var}(T) = E(\text{Var}(T|N)) + \text{Var}(E(T|N)) = E(Np(1-p)) + \text{Var}(Np)$$
$$= \frac{p(1-p)(1-s)}{s} + \frac{p^2(1-s)}{s^2} = \boxed{\frac{p(1-s)(p+s(1-p))}{s^2}}$$

MGF – Finding Moments

Find $E(X^3)$ for $X \sim \text{Expo}(\lambda)$ using the MGF of X. **Answer:** The MGF of an $\text{Expo}(\lambda)$ is $M(t) = \frac{\lambda}{\lambda - t}$. To get the third moment, we can take the third derivative of the MGF and evaluate at $t = 0$:

$$\boxed{E(X^3) = \frac{6}{\lambda^3}}$$

But a much nicer way to use the MGF here is via pattern recognition: note that $M(t)$ looks like it came from a geometric series:

$$\frac{1}{1 - \frac{t}{\lambda}} = \sum_{n=0}^{\infty} \left(\frac{t}{\lambda}\right)^n = \sum_{n=0}^{\infty} \frac{n!}{\lambda^n} \frac{t^n}{n!}$$

The coefficient of $\frac{t^n}{n!}$ here is the nth moment of X, so we have $E(X^n) = \frac{n!}{\lambda^n}$ for all nonnegative integers n.

Markov chains (1)

Suppose X_n is a two-state Markov chain with transition matrix

$$Q = \begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} \end{array}$$

Find the stationary distribution $\vec{s} = (s_0, s_1)$ of X_n by solving $\vec{s}Q = \vec{s}$, and show that the chain is reversible with respect to $\vec{s}$. **Answer:** The equation $\vec{s}Q = \vec{s}$ says that

$$s_0 = s_0(1-\alpha) + s_1\beta \text{ and } s_1 = s_0(\alpha) + s_0(1-\beta)$$

By solving this system of linear equations, we have

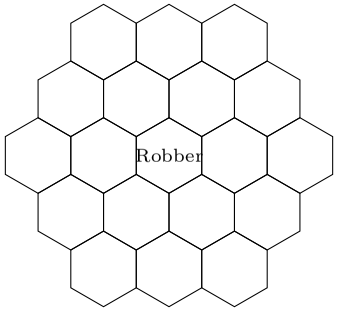
$$\boxed{\vec{s} = \left(\frac{\beta}{\alpha+\beta}, \frac{\alpha}{\alpha+\beta}\right)}$$

To show that the chain is reversible with respect to $\vec{s}$, we must show $s_i q_{ij} = s_j q_{ji}$ for all i, j. This is done if we can show $s_0 q_{01} = s_1 q_{10}$. And indeed,

$$s_0 q_{01} = \frac{\alpha\beta}{\alpha+\beta} = s_1 q_{10}$$

Markov chains (2)

William and Sebastian play a modified game of Settlers of Catan, where every turn they randomly move the robber (which starts on the center tile) to one of the adjacent hexagons.



(a) Is this Markov chain irreducible? Is it aperiodic? **Answer:** $\boxed{\text{Yes to both.}}$ The Markov chain is irreducible because it can get from anywhere to anywhere else. The Markov chain is aperiodic because the robber can return back to a square in $2, 3, 4, 5, \ldots$ moves, and the GCD of those numbers is 1.

(b) What is the stationary distribution of this Markov chain? **Answer:** Since this is a random walk on an undirected graph, the stationary distribution is proportional to the degree sequence. The degree for the corner pieces is 3, the degree for the edge pieces is 4, and the degree for the center pieces is 6. To normalize this degree sequence, we divide by its sum. The sum of the degrees is $6(3) + 6(4) + 7(6) = 84$. Thus the stationary probability of being on a corner is $3/84 = 1/28$, on an edge is $4/84 = 1/21$, and in the center is $6/84 = 1/14$.

(c) What fraction of the time will the robber be in the center tile in this game, in the long run? **Answer:** By the above, $\boxed{1/14}$.

(d) What is the expected amount of moves it will take for the robber to return to the center tile? **Answer:** Since this chain is irreducible and aperiodic, to get the expected time to return we can just invert the stationary probability. Thus on average it will take $\boxed{14}$ turns for the robber to return to the center tile.

Problem-Solving Strategies

Contributions from Jessy Hwang, Yuan Jiang, Yuqi Hou

1. **Getting started.** Start by *defining relevant events and random variables.* ("Let A be the event that I pick the fair coin"; "Let X be the number of successes.") Clear notion is important for clear thinking! Then decide what it is that you're supposed to be finding, in terms of your notation ("I want to find $P(X = 3|A)$"). Think about what type of object your answer should be (a number? A random variable? A PMF? A PDF?) and what it should be in terms of.
   Try simple and extreme cases. To make an abstract experiment more concrete, try *drawing a picture* or making up numbers that could have happened. Pattern recognition: does the structure of the problem resemble something we've seen before?
2. **Calculating probability of an event.** Use counting principles if the naive definition of probability applies. Is the probability of the complement easier to find? Look for symmetries. Look for something to condition on, then apply Bayes' Rule or the Law of Total Probability.
3. **Finding the distribution of a random variable.** First make sure you need the full distribution not just the mean (see next item). Check the *support* of the random variable: what values can it take on? Use this to rule out distributions that don't fit. Is there a *story* for one of the named distributions that fits the problem at hand? Can you write the random variable as a function of an r.v. with a known distribution, say $Y = g(X)$?
4. **Calculating expectation.** If it has a named distribution, check out the table of distributions. If it's a function of an r.v. with a named distribution, try LOTUS. If it's a count of something, try breaking it up into indicator r.v.s. If you can condition on something natural, consider using Adam's law.
5. **Calculating variance.** Consider independence, named distributions, and LOTUS. If it's a count of something, break it up into a sum of indicator r.v.s. If it's a sum, use properties of covariance. If you can condition on something natural, consider using Eve's Law.
6. **Calculating $E(X^2)$.** Do you already know $E(X)$ or $\text{Var}(X)$? Recall that $\text{Var}(X) = E(X^2) - (E(X))^2$. Otherwise try LOTUS.
7. **Calculating covariance.** Use the properties of covariance. If you're trying to find the covariance between two components of a Multinomial distribution, X_i, X_j, then the covariance is $-np_ip_j$ for $i \neq j$.
8. **Symmetry.** If $X_1, \ldots, X_n$ are i.i.d., consider using symmetry.
9. **Calculating probabilities of orderings.** Remember that all $n!$ ordering of i.i.d. continuous random variables $X_1, \ldots, X_n$ are equally likely.
10. **Determining independence.** There are several equivalent definitions. Think about simple and extreme cases to see if you can find a counterexample.
11. **Do a painful integral.** If your integral looks painful, see if you can write your integral in terms of a known PDF (like Gamma or Beta), and use the fact that PDFs integrate to 1?
12. **Before moving on.** Check some simple and extreme cases, check whether the answer seems plausible, check for biohazards.

Biohazards

Contributions from Jessy Hwang

1. **Don't misuse the naive definition of probability.** When answering "What is the probability that in a group of 3 people, no two have the same birth month?", it is *not* correct to treat the people as indistinguishable balls being placed into 12 boxes, since that assumes the list of birth months {January, January, January} is just as likely as the list {January, April, June}, even though the latter is six times more likely.
2. **Don't confuse unconditional, conditional, and joint probabilities.** In applying $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, it is *not* correct to say "$P(B) = 1$ because we know B happened"; $P(B)$ is the *prior* probability of B. Don't confuse $P(A|B)$ with $P(A, B)$.
3. **Don't assume independence without justification.** In the matching problem, the probability that card 1 is a match and card 2 is a match is not $1/n^2$. Binomial and Hypergeometric are often confused; the trials are independent in the Binomial story and dependent in the Hypergeometric story.
4. **Don't forget to do sanity checks.** Probabilities must be between 0 and 1. Variances must be ≥ 0. Supports must make sense. PMFs must sum to 1. PDFs must integrate to 1.
5. **Don't confuse random variables, numbers, and events.** Let X be an r.v. Then $g(X)$ is an r.v. for any function g. In particular, X^2, $|X|$, $F(X)$, and $I_{X>3}$ are r.v.s. $P(X^2 < X|X \geq 0)$, $E(X)$, $\text{Var}(X)$, and $g(E(X))$ are numbers. $X = 2$ and $F(X) \geq -1$ are events. It does not make sense to write $\int_{-\infty}^{\infty} F(X)dx$, because $F(X)$ is a random variable. It does not make sense to write $P(X)$, because X is not an event.

6. **Don't confuse a random variable with its distribution.** To get the PDF of X^2, you can't just square the PDF of X. The right way is to use transformations. To get the PDF of $X+Y$, you can't just add the PDF of X and the PDF of Y. The right way is to compute the convolution.

7. **Don't pull non-linear functions out of expectations.** $E(g(X))$ does not equal $g(E(X))$ in general. The St. Petersburg paradox is an extreme example. See also Jensen's inequality. The right way to find $E(g(X))$ is with LOTUS.

Recommended Resources

- Introduction to Probability Book (`http://bit.ly/introprobability`)
- Stat 110 Online (`http://stat110.net`)
- Stat 110 Quora Blog (`https://stat110.quora.com/`)
- Quora Probability FAQ (`http://bit.ly/probabilityfaq`)
- R Studio (`https://www.rstudio.com`)
- LaTeX File (`github.com/wzchen/probability_cheatsheet`)

Please share this cheatsheet with friends!
`http://wzchen.com/probability-cheatsheet`

Distributions in R

Command	What it does
`help(distributions)`	shows documentation on distributions
`dbinom(k,n,p)`	PMF $P(X=k)$ for $X \sim \text{Bin}(n,p)$
`pbinom(x,n,p)`	CDF $P(X \leq x)$ for $X \sim \text{Bin}(n,p)$
`qbinom(a,n,p)`	ath quantile for $X \sim \text{Bin}(n,p)$
`rbinom(r,n,p)`	vector of r i.i.d. $\text{Bin}(n,p)$ r.v.s
`dgeom(k,p)`	PMF $P(X=k)$ for $X \sim \text{Geom}(p)$
`dhyper(k,w,b,n)`	PMF $P(X=k)$ for $X \sim \text{HGeom}(w,b,n)$
`dnbinom(k,r,p)`	PMF $P(X=k)$ for $X \sim \text{NBin}(r,p)$
`dpois(k,r)`	PMF $P(X=k)$ for $X \sim \text{Pois}(r)$
`dbeta(x,a,b)`	PDF $f(x)$ for $X \sim \text{Beta}(a,b)$
`dchisq(x,n)`	PDF $f(x)$ for $X \sim \chi^2_n$
`dexp(x,b)`	PDF $f(x)$ for $X \sim \text{Expo}(b)$
`dgamma(x,a,r)`	PDF $f(x)$ for $X \sim \text{Gamma}(a,r)$
`dlnorm(x,m,s)`	PDF $f(x)$ for $X \sim \mathcal{LN}(m,s^2)$
`dnorm(x,m,s)`	PDF $f(x)$ for $X \sim \mathcal{N}(m,s^2)$
`dt(x,n)`	PDF $f(x)$ for $X \sim t_n$
`dunif(x,a,b)`	PDF $f(x)$ for $X \sim \text{Unif}(a,b)$

The table above gives R commands for working with various named distributions. Commands analogous to `pbinom`, `qbinom`, and `rbinom` work for the other distributions in the table. For example, `pnorm`, `qnorm`, and `rnorm` can be used to get the CDF, quantiles, and random generation for the Normal. For the Multinomial, `dmultinom` can be used for calculating the joint PMF and `rmultinom` can be used for generating random vectors. For the Multivariate Normal, after installing and loading the `mvtnorm` package `dmvnorm` can be used for calculating the joint PDF and `rmvnorm` can be used for generating random vectors.

Table of Distributions

Distribution	PMF/PDF and Support	Expected Value	Variance	MGF
Bernoulli Bern(p)	$P(X=1)=p$ $P(X=0)=q=1-p$	p	pq	$q+pe^t$
Binomial Bin(n,p)	$P(X=k)=\binom{n}{k}p^k q^{n-k}$ $k\in\{0,1,2,\dots n\}$	np	npq	$(q+pe^t)^n$
Geometric Geom(p)	$P(X=k)=q^k p$ $k\in\{0,\ 1,\ 2,\ \dots\}$	q/p	q/p^2	$\frac{p}{1-qe^t}$, $qe^t<1$
Negative Binomial NBin(r,p)	$P(X=n)=\binom{r+n-1}{r-1}p^r q^n$ $n\in\{0,\ 1,\ 2,\ \dots\}$	rq/p	rq/p^2	$(\frac{p}{1-qe^t})^r$, $qe^t<1$
Hypergeometric HGeom(w,b,n)	$P(X=k)=\binom{w}{k}\binom{b}{n-k}/\binom{w+b}{n}$ $k\in\{0,1,2,\dots,n\}$	$\mu=\frac{nw}{b+w}$	$\left(\frac{w+b-n}{w+b-1}\right)n\frac{\mu}{n}(1-\frac{\mu}{n})$	messy
Poisson Pois(λ)	$P(X=k)=\frac{e^{-\lambda}\lambda^k}{k!}$ $k\in\{0,\ 1,\ 2,\ \dots\}$	λ	λ	$e^{\lambda(e^t-1)}$
Uniform Unif(a,b)	$f(x)=\frac{1}{b-a}$ $x\in(a,b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$	$\frac{e^{tb}-e^{ta}}{t(b-a)}$
Normal $\mathcal{N}(\mu,\sigma^2)$	$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/(2\sigma^2)}$ $x\in(-\infty,\infty)$	μ	σ^2	$e^{t\mu+\frac{\sigma^2t^2}{2}}$
Exponential Expo(λ)	$f(x)=\lambda e^{-\lambda x}$ $x\in(0,\infty)$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$	$\frac{\lambda}{\lambda-t}$, $t<\lambda$
Gamma Gamma(a,λ)	$f(x)=\frac{1}{\Gamma(a)}(\lambda x)^a e^{-\lambda x}\frac{1}{x}$ $x\in(0,\infty)$	$\frac{a}{\lambda}$	$\frac{a}{\lambda^2}$	$\left(\frac{\lambda}{\lambda-t}\right)^a$, $t<\lambda$
Beta Beta(a,b)	$f(x)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}x^{a-1}(1-x)^{b-1}$ $x\in(0,1)$	$\mu=\frac{a}{a+b}$	$\frac{\mu(1-\mu)}{(a+b+1)}$	messy
Log-Normal $\mathcal{LN}(\mu,\sigma^2)$	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-(\log x-\mu)^2/(2\sigma^2)}$ $x\in(0,\infty)$	$\theta=e^{\mu+\sigma^2/2}$	$\theta^2(e^{\sigma^2}-1)$	doesn't exist
Chi-Square χ^2_n	$\frac{1}{2^{n/2}\Gamma(n/2)}x^{n/2-1}e^{-x/2}$ $x\in(0,\infty)$	n	$2n$	$(1-2t)^{-n/2}$, $t<1/2$
Student-t t_n	$\frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)}(1+x^2/n)^{-(n+1)/2}$ $x\in(-\infty,\infty)$	0 if $n>1$	$\frac{n}{n-2}$ if $n>2$	doesn't exist

z	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
+0	.50000	.50399	.50798	.51197	.51595	.51994	.52392	.52790	.53188	.53586
+0.1	.53983	.54380	.54776	.55172	.55567	.55966	.56360	.56749	.57142	.57535
+0.2	.57926	.58317	.58706	.59095	.59483	.59871	.60257	.60642	.61026	.61409
+0.3	.61791	.62172	.62552	.62930	.63307	.63683	.64058	.64431	.64803	.65173
+0.4	.65542	.65910	.66276	.66640	.67003	.67364	.67724	.68082	.68439	.68793
+0.5	.69146	.69497	.69847	.70194	.70540	.70884	.71226	.71566	.71904	.72240
+0.6	.72575	.72907	.73237	.73565	.73891	.74215	.74537	.74857	.75175	.75490
+0.7	.75804	.76115	.76424	.76730	.77035	.77337	.77637	.77935	.78230	.78524
+0.8	.78814	.79103	.79389	.79673	.79955	.80234	.80511	.80785	.81057	.81327
+0.9	.81594	.81859	.82121	.82381	.82639	.82894	.83147	.83398	.83646	.83891
+1	.84134	.84375	.84614	.84849	.85083	.85314	.85543	.85769	.85993	.86214
+1.1	.86433	.86650	.86864	.87076	.87286	.87493	.87698	.87900	.88100	.88298
+1.2	.88493	.88686	.88877	.89065	.89251	.89435	.89617	.89796	.89973	.90147
+1.3	.90320	.90490	.90658	.90824	.90988	.91149	.91308	.91466	.91621	.91774
+1.4	.91924	.92073	.92220	.92364	.92507	.92647	.92785	.92922	.93056	.93189
+1.5	.93319	.93448	.93574	.93699	.93822	.93943	.94062	.94179	.94295	.94408
+1.6	.94520	.94630	.94738	.94845	.94950	.95053	.95154	.95254	.95352	.95449
+1.7	.95543	.95637	.95728	.95818	.95907	.95994	.96080	.96164	.96246	.96327
+1.8	.96407	.96485	.96562	.96638	.96712	.96784	.96856	.96926	.96995	.97062
+1.9	.97128	.97193	.97257	.97320	.97381	.97441	.97500	.97558	.97615	.97670
+2	.97725	.97778	.97831	.97882	.97932	.97982	.98030	.98077	.98124	.98169
+2.1	.98214	.98257	.98300	.98341	.98382	.98422	.98461	.98500	.98537	.98574
+2.2	.98610	.98645	.98679	.98713	.98745	.98778	.98809	.98840	.98870	.98899
+2.3	.98928	.98956	.98983	.99010	.99036	.99061	.99086	.99111	.99134	.99158
+2.4	.99180	.99202	.99224	.99245	.99266	.99286	.99305	.99324	.99343	.99361
+2.5	.99379	.99396	.99413	.99430	.99446	.99461	.99477	.99492	.99506	.99520
+2.6	.99534	.99547	.99560	.99573	.99585	.99598	.99609	.99621	.99632	.99643
+2.7	.99653	.99664	.99674	.99683	.99693	.99702	.99711	.99720	.99728	.99736
+2.8	.99744	.99752	.99760	.99767	.99774	.99781	.99788	.99795	.99801	.99807
+2.9	.99813	.99819	.99825	.99831	.99836	.99841	.99846	.99851	.99856	.99861
+3	.99865	.99869	.99874	.99878	.99882	.99886	.99889	.99893	.99896	.99900
+3.1	.99903	.99906	.99910	.99913	.99916	.99918	.99921	.99924	.99926	.99929
+3.2	.99931	.99934	.99936	.99938	.99940	.99942	.99944	.99946	.99948	.99950
+3.3	.99952	.99953	.99955	.99957	.99958	.99960	.99961	.99962	.99964	.99965
+3.4	.99966	.99968	.99969	.99970	.99971	.99972	.99973	.99974	.99975	.99976
+3.5	.99977	.99978	.99978	.99979	.99980	.99981	.99981	.99982	.99983	.99983
+3.6	.99984	.99985	.99985	.99986	.99986	.99987	.99987	.99988	.99988	.99989
+3.7	.99989	.99990	.99990	.99990	.99991	.99991	.99992	.99992	.99992	.99992
+3.8	.99993	.99993	.99993	.99994	.99994	.99994	.99994	.99995	.99995	.99995
+3.9	.99995	.99995	.99996	.99996	.99996	.99996	.99996	.99996	.99997	.99997
+4	.99997	.99997	.99997	.99997	.99997	.99997	.99998	.99998	.99998	.99998

z	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-0	.50000	.49601	.49202	.48803	.48405	.48006	.47608	.47210	.46812	.46414
-0.1	.46017	.45620	.45224	.44828	.44433	.44034	.43640	.43251	.42858	.42465
-0.2	.42074	.41683	.41294	.40905	.40517	.40129	.39743	.39358	.38974	.38591
-0.3	.38209	.37828	.37448	.37070	.36693	.36317	.35942	.35569	.35197	.34827
-0.4	.34458	.34090	.33724	.33360	.32997	.32636	.32276	.31918	.31561	.31207
-0.5	.30854	.30503	.30153	.29806	.29460	.29116	.28774	.28434	.28096	.27760
-0.6	.27425	.27093	.26763	.26435	.26109	.25785	.25463	.25143	.24825	.24510
-0.7	.24196	.23885	.23576	.23270	.22965	.22663	.22363	.22065	.21770	.21476
-0.8	.21186	.20897	.20611	.20327	.20045	.19766	.19489	.19215	.18943	.18673
-0.9	.18406	.18141	.17879	.17619	.17361	.17106	.16853	.16602	.16354	.16109
-1	.15866	.15625	.15386	.15151	.14917	.14686	.14457	.14231	.14007	.13786
-1.1	.13567	.13350	.13136	.12924	.12714	.12507	.12302	.12100	.11900	.11702
-1.2	.11507	.11314	.11123	.10935	.10749	.10565	.10383	.10204	.10027	.09853
-1.3	.09680	.09510	.09342	.09176	.09012	.08851	.08692	.08534	.08379	.08226
-1.4	.08076	.07927	.07780	.07636	.07493	.07353	.07215	.07078	.06944	.06811
-1.5	.06681	.06552	.06426	.06301	.06178	.06057	.05938	.05821	.05705	.05592
-1.6	.05480	.05370	.05262	.05155	.05050	.04947	.04846	.04746	.04648	.04551
-1.7	.04457	.04363	.04272	.04182	.04093	.04006	.03920	.03836	.03754	.03673
-1.8	.03593	.03515	.03438	.03362	.03288	.03216	.03144	.03074	.03005	.02938
-1.9	.02872	.02807	.02743	.02680	.02619	.02559	.02500	.02442	.02385	.02330
-2	.02275	.02222	.02169	.02118	.02068	.02018	.01970	.01923	.01876	.01831
-2.1	.01786	.01743	.01700	.01659	.01618	.01578	.01539	.01500	.01463	.01426
-2.2	.01390	.01355	.01321	.01287	.01255	.01222	.01191	.01160	.01130	.01101
-2.3	.01072	.01044	.01017	.00990	.00964	.00939	.00914	.00889	.00866	.00842
-2.4	.00820	.00798	.00776	.00755	.00734	.00714	.00695	.00676	.00657	.00639
-2.5	.00621	.00604	.00587	.00570	.00554	.00539	.00523	.00508	.00494	.00480
-2.6	.00466	.00453	.00440	.00427	.00415	.00402	.00391	.00379	.00368	.00357
-2.7	.00347	.00336	.00326	.00317	.00307	.00298	.00289	.00280	.00272	.00264
-2.8	.00256	.00248	.00240	.00233	.00226	.00219	.00212	.00205	.00199	.00193
-2.9	.00187	.00181	.00175	.00169	.00164	.00159	.00154	.00149	.00144	.00139
-3	.00135	.00131	.00126	.00122	.00118	.00114	.00111	.00107	.00104	.00100
-3.1	.00097	.00094	.00090	.00087	.00084	.00082	.00079	.00076	.00074	.00071
-3.2	.00069	.00066	.00064	.00062	.00060	.00058	.00056	.00054	.00052	.00050
-3.3	.00048	.00047	.00045	.00043	.00042	.00040	.00039	.00038	.00036	.00035
-3.4	.00034	.00032	.00031	.00030	.00029	.00028	.00027	.00026	.00025	.00024
-3.5	.00023	.00022	.00022	.00021	.00020	.00019	.00019	.00018	.00017	.00017
-3.6	.00016	.00015	.00015	.00014	.00014	.00013	.00013	.00012	.00012	.00011
-3.7	.00011	.00010	.00010	.00010	.00009	.00009	.00008	.00008	.00008	.00008
-3.8	.00007	.00007	.00007	.00006	.00006	.00006	.00006	.00005	.00005	.00005
-3.9	.00005	.00005	.00004	.00004	.00004	.00004	.00004	.00004	.00003	.00003
-4	.00003	.00003	.00003	.00003	.00003	.00003	.00002	.00002	.00002	.00002

VIP Refresher: Linear Algebra and Calculus

Afshine Amidi and Shervine Amidi

October 6, 2018

General notations

❒ **Vector** – We note $x \in \mathbb{R}^n$ a vector with n entries, where $x_i \in \mathbb{R}$ is the i^{th} entry:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

❒ **Matrix** – We note $A \in \mathbb{R}^{m \times n}$ a matrix with m rows and n columns, where $A_{i,j} \in \mathbb{R}$ is the entry located in the i^{th} row and j^{th} column:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Remark: the vector x defined above can be viewed as a $n \times 1$ matrix and is more particularly called a column-vector.

❒ **Identity matrix** – The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones in its diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remark: for all matrices $A \in \mathbb{R}^{n \times n}$, we have $A \times I = I \times A = A$.

❒ **Diagonal matrix** – A diagonal matrix $D \in \mathbb{R}^{n \times n}$ is a square matrix with nonzero values in its diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

Remark: we also note D as $diag(d_1,...,d_n)$.

Matrix operations

❒ **Vector-vector multiplication** – There are two types of vector-vector products:

- inner product: for $x,y \in \mathbb{R}^n$, we have:

$$\boxed{x^T y = \sum_{i=1}^{n} x_i y_i \in \mathbb{R}}$$

- outer product: for $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, we have:

$$\boxed{xy^T = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \cdots & x_m y_n \end{pmatrix} \in \mathbb{R}^{m \times n}}$$

❒ **Matrix-vector multiplication** – The product of matrix $A \in \mathbb{R}^{m \times n}$ and vector $x \in \mathbb{R}^n$ is a vector of size $\mathbb{R}^m$, such that:

$$\boxed{Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} x_i \in \mathbb{R}^m}$$

where $a_{r,i}^T$ are the vector rows and $a_{c,j}$ are the vector columns of A, and x_i are the entries of x.

❒ **Matrix-matrix multiplication** – The product of matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is a matrix of size $\mathbb{R}^{n \times p}$, such that:

$$\boxed{AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} b_{r,i}^T \in \mathbb{R}^{n \times p}}$$

where $a_{r,i}^T, b_{r,i}^T$ are the vector rows and $a_{c,j}, b_{c,j}$ are the vector columns of A and B respectively.

❒ **Transpose** – The transpose of a matrix $A \in \mathbb{R}^{m \times n}$, noted A^T, is such that its entries are flipped:

$$\boxed{\forall i,j, \quad A_{i,j}^T = A_{j,i}}$$

Remark: for matrices A,B, we have $(AB)^T = B^T A^T$.

❒ **Inverse** – The inverse of an invertible square matrix A is noted A^{-1} and is the only matrix such that:

$$\boxed{AA^{-1} = A^{-1}A = I}$$

Remark: not all square matrices are invertible. Also, for matrices A,B, we have $(AB)^{-1} = B^{-1}A^{-1}$

❒ **Trace** – The trace of a square matrix A, noted $\text{tr}(A)$, is the sum of its diagonal entries:

$$\boxed{\text{tr}(A) = \sum_{i=1}^{n} A_{i,i}}$$

Remark: for matrices A,B, we have $tr(A^T) = tr(A)$ and $tr(AB) = tr(BA)$

❒ **Determinant** – The determinant of a square matrix $A \in \mathbb{R}^{n \times n}$, noted $|A|$ or $\det(A)$ is expressed recursively in terms of $A_{\backslash i,\backslash j}$, which is the matrix A without its i^{th} row and j^{th} column, as follows:

$$\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} |A_{\backslash i, \backslash j}|$$

Remark: A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

Matrix properties

❒ **Symmetric decomposition** – A given matrix A can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}$$

❒ **Norm** – A norm is a function $N : V \longrightarrow [0, +\infty[$ where V is a vector space, and such that for all x,$y \in V$, we have:

- $N(x + y) \leqslant N(x) + N(y)$
- $N(ax) = |a|N(x)$ for a scalar
- if $N(x) = 0$, then $x = 0$

For $x \in V$, the most commonly used norms are summed up in the table below:

Norm	Notation	Definition	Use case
Manhattan, L^1	$\|\|x\|\|_1$	$\sum_{i=1}^{n} \|x_i\|$	LASSO regularization
Euclidean, L^2	$\|\|x\|\|_2$	$\sqrt{\sum_{i=1}^{n} x_i^2}$	Ridge regularization
p-norm, L^p	$\|\|x\|\|_p$	$\left(\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$	Hölder inequality
Infinity, L^∞	$\|\|x\|\|_\infty$	$\max_i \|x_i\|$	Uniform convergence

❒ **Linearly dependence** – A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others.
Remark: if no vector can be written this way, then the vectors are said to be linearly independent.

❒ **Matrix rank** – The rank of a given matrix A is noted $\text{rank}(A)$ and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of A.

❒ **Positive semi-definite matrix** – A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD) and is noted $A \succeq 0$ if we have:

$$A = A^T \quad \text{and} \quad \forall x \in \mathbb{R}^n, \quad x^T A x \geqslant 0$$

Remark: similarly, a matrix A is said to be positive definite, and is noted $A \succ 0$, if it is a PSD matrix which satisfies for all non-zero vector x, $x^T A x > 0$.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n \times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$Az = \lambda z$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n \times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n \times n}$. By noting $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$, we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

❒ **Singular-value decomposition** – For a given matrix A of dimensions $m \times n$, the singular-value decomposition (SVD) is a factorization technique that guarantees the existence of U $m \times m$ unitary, Σ $m \times n$ diagonal and V $n \times n$ unitary matrices, such that:

$$A = U \Sigma V^T$$

Matrix calculus

❒ **Gradient** – Let $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ be a function and $A \in \mathbb{R}^{m \times n}$ be a matrix. The gradient of f with respect to A is a $m \times n$ matrix, noted $\nabla_A f(A)$, such that:

$$\left(\nabla_A f(A)\right)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}$$

Remark: the gradient of f is only defined when f is a function that returns a scalar.

❒ **Hessian** – Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and $x \in \mathbb{R}^n$ be a vector. The hessian of f with respect to x is a $n \times n$ symmetric matrix, noted $\nabla_x^2 f(x)$, such that:

$$\left(\nabla_x^2 f(x)\right)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

Remark: the hessian of f is only defined when f is a function that returns a scalar.

❒ **Gradient operations** – For matrices A,B,C, the following gradient properties are worth having in mind:

$$\nabla_A \text{tr}(AB) = B^T \qquad \nabla_{A^T} f(A) = (\nabla_A f(A))^T$$

$$\nabla_A \text{tr}(ABA^T C) = CAB + C^T AB^T \qquad \nabla_A |A| = |A|(A^{-1})^T$$

Limits

Definitions

Precise Definition : We say $\lim_{x\to a} f(x) = L$ if for every $\varepsilon > 0$ there is a $\delta > 0$ such that whenever $0 < |x-a| < \delta$ then $|f(x) - L| < \varepsilon$.

"Working" Definition : We say $\lim_{x\to a} f(x) = L$ if we can make $f(x)$ as close to L as we want by taking x sufficiently close to a (on either side of a) without letting $x = a$.

Right hand limit : $\lim_{x\to a^+} f(x) = L$. This has the same definition as the limit except it requires $x > a$.

Left hand limit : $\lim_{x\to a^-} f(x) = L$. This has the same definition as the limit except it requires $x < a$.

Limit at Infinity : We say $\lim_{x\to\infty} f(x) = L$ if we can make $f(x)$ as close to L as we want by taking x large enough and positive.

There is a similar definition for $\lim_{x\to-\infty} f(x) = L$ except we require x large and negative.

Infinite Limit : We say $\lim_{x\to a} f(x) = \infty$ if we can make $f(x)$ arbitrarily large (and positive) by taking x sufficiently close to a (on either side of a) without letting $x = a$.

There is a similar definition for $\lim_{x\to a} f(x) = -\infty$ except we make $f(x)$ arbitrarily large and negative.

Relationship between the limit and one-sided limits

$$\lim_{x\to a} f(x) = L \;\Rightarrow\; \lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x) = L \qquad \lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x) = L \;\Rightarrow\; \lim_{x\to a} f(x) = L$$

$$\lim_{x\to a^+} f(x) \neq \lim_{x\to a^-} f(x) \;\Rightarrow\; \lim_{x\to a} f(x) \text{ Does Not Exist}$$

Properties

Assume $\lim_{x\to a} f(x)$ and $\lim_{x\to a} g(x)$ both exist and c is any number then,

1. $\lim_{x\to a}\left[c f(x)\right] = c\lim_{x\to a} f(x)$
2. $\lim_{x\to a}\left[f(x) \pm g(x)\right] = \lim_{x\to a} f(x) \pm \lim_{x\to a} g(x)$
3. $\lim_{x\to a}\left[f(x)g(x)\right] = \lim_{x\to a} f(x)\ \lim_{x\to a} g(x)$
4. $\lim_{x\to a}\left[\dfrac{f(x)}{g(x)}\right] = \dfrac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)}$ provided $\lim_{x\to a} g(x) \neq 0$
5. $\lim_{x\to a}\left[f(x)\right]^n = \left[\lim_{x\to a} f(x)\right]^n$
6. $\lim_{x\to a}\left[\sqrt[n]{f(x)}\right] = \sqrt[n]{\lim_{x\to a} f(x)}$

Basic Limit Evaluations at $\pm\infty$

Note : $\operatorname{sgn}(a) = 1$ if $a > 0$ and $\operatorname{sgn}(a) = -1$ if $a < 0$.

1. $\lim_{x\to\infty} \mathbf{e}^x = \infty$ & $\lim_{x\to-\infty} \mathbf{e}^x = 0$
2. $\lim_{x\to\infty} \ln(x) = \infty$ & $\lim_{x\to 0^+} \ln(x) = -\infty$
3. If $r > 0$ then $\lim_{x\to\infty} \dfrac{b}{x^r} = 0$
4. If $r > 0$ and x^r is real for negative x then $\lim_{x\to-\infty} \dfrac{b}{x^r} = 0$
5. n even : $\lim_{x\to\pm\infty} x^n = \infty$
6. n odd : $\lim_{x\to\infty} x^n = \infty$ & $\lim_{x\to-\infty} x^n = -\infty$
7. n even : $\lim_{x\to\pm\infty} a x^n + \cdots + b x + c = \operatorname{sgn}(a)\infty$
8. n odd : $\lim_{x\to\infty} a x^n + \cdots + b x + c = \operatorname{sgn}(a)\infty$
9. n odd : $\lim_{x\to-\infty} a x^n + \cdots + c x + d = -\operatorname{sgn}(a)\infty$

Evaluation Techniques

Continuous Functions

If $f(x)$ is continuous at a then $\lim_{x\to a} f(x) = f(a)$

Continuous Functions and Composition

$f(x)$ is continuous at b and $\lim_{x\to a} g(x) = b$ then

$$\lim_{x\to a} f(g(x)) = f\left(\lim_{x\to a} g(x)\right) = f(b)$$

Factor and Cancel

$$\lim_{x\to 2}\frac{x^2+4x-12}{x^2-2x} = \lim_{x\to 2}\frac{(x-2)(x+6)}{x(x-2)}$$

$$= \lim_{x\to 2}\frac{x+6}{x} = \frac{8}{2} = 4$$

Rationalize Numerator/Denominator

$$\lim_{x\to 9}\frac{3-\sqrt{x}}{x^2-81} = \lim_{x\to 9}\frac{3-\sqrt{x}}{x^2-81}\,\frac{3+\sqrt{x}}{3+\sqrt{x}}$$

$$= \lim_{x\to 9}\frac{9-x}{\left(x^2-81\right)\left(3+\sqrt{x}\right)} = \lim_{x\to 9}\frac{-1}{(x+9)\left(3+\sqrt{x}\right)}$$

$$= \frac{-1}{(18)(6)} = -\frac{1}{108}$$

Combine Rational Expressions

$$\lim_{h\to 0}\frac{1}{h}\left(\frac{1}{x+h}-\frac{1}{x}\right) = \lim_{h\to 0}\frac{1}{h}\left(\frac{x-(x+h)}{x(x+h)}\right)$$

$$= \lim_{h\to 0}\frac{1}{h}\left(\frac{-h}{x(x+h)}\right) = \lim_{h\to 0}\frac{-1}{x(x+h)} = -\frac{1}{x^2}$$

L'Hospital's Rule

If $\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{0}{0}$ or $\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{\pm\infty}{\pm\infty}$ then,

$$\lim_{x\to a}\frac{f(x)}{g(x)} = \lim_{x\to a}\frac{f'(x)}{g'(x)}$$

a is a number, ∞ or $-\infty$

Polynomials at Infinity

$p(x)$ and $q(x)$ are polynomials. To compute $\lim_{x\to\pm\infty}\frac{p(x)}{q(x)}$ factor largest power of x in $q(x)$ out of both $p(x)$ and $q(x)$ then compute limit.

$$\lim_{x\to-\infty}\frac{3x^2-4}{5x-2x^2} = \lim_{x\to-\infty}\frac{x^2\left(3-\frac{4}{x^2}\right)}{x^2\left(\frac{5}{x}-2\right)} = \lim_{x\to-\infty}\frac{3-\frac{4}{x^2}}{\frac{5}{x}-2} = -\frac{3}{2}$$

Piecewise Function

$\lim_{x\to -2} g(x)$ where $g(x) = \begin{cases} x^2+5 & \text{if } x<-2 \\ 1-3x & \text{if } x\ge -2 \end{cases}$

Compute two one sided limits,

$$\lim_{x\to -2^-} g(x) = \lim_{x\to -2^-} x^2+5 = 9$$

$$\lim_{x\to -2^+} g(x) = \lim_{x\to -2^+} 1-3x = 7$$

One sided limits are different so $\lim_{x\to -2} g(x)$ doesn't exist. If the two one sided limits had been equal then $\lim_{x\to -2} g(x)$ would have existed and had the same value.

Some Continuous Functions

Partial list of continuous functions and the values of x for which they are continuous.

1. Polynomials for all x.
2. Rational function, except for x's that give division by zero.
3. $\sqrt[n]{x}$ (n odd) for all x.
4. $\sqrt[n]{x}$ (n even) for all $x \ge 0$.
5. $\mathbf{e}^x$ for all x.
6. $\ln x$ for $x > 0$.
7. $\cos(x)$ and $\sin(x)$ for all x.
8. $\tan(x)$ and $\sec(x)$ provided $x \ne \cdots, -\frac{3\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{2}, \frac{3\pi}{2}, \cdots$
9. $\cot(x)$ and $\csc(x)$ provided $x \ne \cdots, -2\pi, -\pi, 0, \pi, 2\pi, \cdots$

Intermediate Value Theorem

Suppose that $f(x)$ is continuous on [a, b] and let M be any number between $f(a)$ and $f(b)$. Then there exists a number c such that $a < c < b$ and $f(c) = M$.

Derivatives

Definition and Notation

If $y = f(x)$ then the derivative is defined to be $f'(x) = \lim_{h\to 0} \dfrac{f(x+h) - f(x)}{h}$.

If $y = f(x)$ then all of the following are equivalent notations for the derivative.

$$f'(x) = y' = \frac{df}{dx} = \frac{dy}{dx} = \frac{d}{dx}\left(f(x)\right) = Df(x)$$

If $y = f(x)$ all of the following are equivalent notations for derivative evaluated at $x = a$.

$$f'(a) = y'\Big|_{x=a} = \left.\frac{df}{dx}\right|_{x=a} = \left.\frac{dy}{dx}\right|_{x=a} = Df(a)$$

Interpretation of the Derivative

If $y = f(x)$ then,

1. $m = f'(a)$ is the slope of the tangent line to $y = f(x)$ at $x = a$ and the equation of the tangent line at $x = a$ is given by $y = f(a) + f'(a)(x - a)$.
2. $f'(a)$ is the instantaneous rate of change of $f(x)$ at $x = a$.
3. If $f(x)$ is the position of an object at time x then $f'(a)$ is the velocity of the object at $x = a$.

Basic Properties and Formulas

If $f(x)$ and $g(x)$ are differentiable functions (the derivative exists), c and n are any real numbers,

1. $(cf)' = cf'(x)$
2. $(f \pm g)' = f'(x) \pm g'(x)$
3. $(fg)' = f'g + fg'$ – **Product Rule**
4. $\left(\dfrac{f}{g}\right)' = \dfrac{f'g - fg'}{g^2}$ – **Quotient Rule**
5. $\dfrac{d}{dx}(c) = 0$
6. $\dfrac{d}{dx}\left(x^n\right) = nx^{n-1}$ – **Power Rule**
7. $\dfrac{d}{dx}\left(f(g(x))\right) = f'(g(x))g'(x)$
   This is the **Chain Rule**

Common Derivatives

$$\frac{d}{dx}(x) = 1$$
$$\frac{d}{dx}(\sin x) = \cos x$$
$$\frac{d}{dx}(\cos x) = -\sin x$$
$$\frac{d}{dx}(\tan x) = \sec^2 x$$
$$\frac{d}{dx}(\sec x) = \sec x \tan x$$
$$\frac{d}{dx}(\csc x) = -\csc x \cot x$$
$$\frac{d}{dx}(\cot x) = -\csc^2 x$$
$$\frac{d}{dx}\left(\sin^{-1} x\right) = \frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\left(\cos^{-1} x\right) = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\left(\tan^{-1} x\right) = \frac{1}{1+x^2}$$
$$\frac{d}{dx}\left(a^x\right) = a^x \ln(a)$$
$$\frac{d}{dx}\left(\mathbf{e}^x\right) = \mathbf{e}^x$$
$$\frac{d}{dx}\left(\ln(x)\right) = \frac{1}{x}, \quad x > 0$$
$$\frac{d}{dx}\left(\ln|x|\right) = \frac{1}{x}, \quad x \neq 0$$
$$\frac{d}{dx}\left(\log_a(x)\right) = \frac{1}{x \ln a}, \quad x > 0$$

Chain Rule Variants

The chain rule applied to some specific functions.

1. $\frac{d}{dx}\left(\left[f(x)\right]^n\right) = n\left[f(x)\right]^{n-1} f'(x)$
2. $\frac{d}{dx}\left(\mathbf{e}^{f(x)}\right) = f'(x)\mathbf{e}^{f(x)}$
3. $\frac{d}{dx}\left(\ln\left[f(x)\right]\right) = \frac{f'(x)}{f(x)}$
4. $\frac{d}{dx}\left(\sin\left[f(x)\right]\right) = f'(x)\cos\left[f(x)\right]$
5. $\frac{d}{dx}\left(\cos\left[f(x)\right]\right) = -f'(x)\sin\left[f(x)\right]$
6. $\frac{d}{dx}\left(\tan\left[f(x)\right]\right) = f'(x)\sec^2\left[f(x)\right]$
7. $\frac{d}{dx}\left(\sec\left[f(x)\right]\right) = f'(x)\sec\left[f(x)\right]\tan\left[f(x)\right]$
8. $\frac{d}{dx}\left(\tan^{-1}\left[f(x)\right]\right) = \frac{f'(x)}{1+\left[f(x)\right]^2}$

Higher Order Derivatives

The Second Derivative is denoted as $f''(x) = f^{(2)}(x) = \frac{d^2 f}{dx^2}$ and is defined as $f''(x) = \left(f'(x)\right)'$, *i.e.* the derivative of the first derivative, $f'(x)$.

The n[th] Derivative is denoted as $f^{(n)}(x) = \frac{d^n f}{dx^n}$ and is defined as $f^{(n)}(x) = \left(f^{(n-1)}(x)\right)'$, *i.e.* the derivative of the $(n\text{-}1)^{st}$ derivative, $f^{(n-1)}(x)$.

Implicit Differentiation

Find y' if $\mathbf{e}^{2x-9y} + x^3y^2 = \sin(y) + 11x$. Remember $y = y(x)$ here, so products/quotients of x and y will use the product/quotient rule and derivatives of y will use the chain rule. The "trick" is to differentiate as normal and every time you differentiate a y you tack on a y' (from the chain rule). After differentiating solve for y'.

$$\begin{aligned}
&\mathbf{e}^{2x-9y}(2-9y') + 3x^2y^2 + 2x^3y\,y' = \cos(y)\,y' + 11\\
&2\mathbf{e}^{2x-9y} - 9y'\mathbf{e}^{2x-9y} + 3x^2y^2 + 2x^3y\,y' = \cos(y)\,y' + 11 \qquad \Rightarrow \qquad y' = \frac{11 - 2\mathbf{e}^{2x-9y} - 3x^2y^2}{2x^3y - 9\mathbf{e}^{2x-9y} - \cos(y)}\\
&\left(2x^3y - 9\mathbf{e}^{2x-9y} - \cos(y)\right)y' = 11 - 2\mathbf{e}^{2x-9y} - 3x^2y^2
\end{aligned}$$

Increasing/Decreasing – Concave Up/Concave Down

Critical Points

$x = c$ is a critical point of $f(x)$ provided either **1.** $f'(c) = 0$ or **2.** $f'(c)$ doesn't exist.

Increasing/Decreasing

1. If $f'(x) > 0$ for all x in an interval I then $f(x)$ is increasing on the interval I.
2. If $f'(x) < 0$ for all x in an interval I then $f(x)$ is decreasing on the interval I.
3. If $f'(x) = 0$ for all x in an interval I then $f(x)$ is constant on the interval I.

Concave Up/Concave Down

1. If $f''(x) > 0$ for all x in an interval I then $f(x)$ is concave up on the interval I.
2. If $f''(x) < 0$ for all x in an interval I then $f(x)$ is concave down on the interval I.

Inflection Points

$x = c$ is a inflection point of $f(x)$ if the concavity changes at $x = c$.

Extrema

Absolute Extrema

1. $x = c$ is an absolute maximum of $f(x)$ if $f(c) \ge f(x)$ for all x in the domain.
2. $x = c$ is an absolute minimum of $f(x)$ if $f(c) \le f(x)$ for all x in the domain.

Fermat's Theorem

If $f(x)$ has a relative (or local) extrema at $x = c$, then $x = c$ is a critical point of $f(x)$.

Extreme Value Theorem

If $f(x)$ is continuous on the closed interval $[a,b]$ then there exist numbers c and d so that, **1.** $a \le c, d \le b$, **2.** $f(c)$ is the abs. max. in $[a,b]$, **3.** $f(d)$ is the abs. min. in $[a,b]$.

Finding Absolute Extrema

To find the absolute extrema of the continuous function $f(x)$ on the interval $[a,b]$ use the following process.

1. Find all critical points of $f(x)$ in $[a,b]$.
2. Evaluate $f(x)$ at all points found in Step 1.
3. Evaluate $f(a)$ and $f(b)$.
4. Identify the abs. max. (largest function value) and the abs. min.(smallest function value) from the evaluations in Steps 2 & 3.

Relative (local) Extrema

1. $x = c$ is a relative (or local) maximum of $f(x)$ if $f(c) \ge f(x)$ for all x near c.
2. $x = c$ is a relative (or local) minimum of $f(x)$ if $f(c) \le f(x)$ for all x near c.

1st Derivative Test

If $x = c$ is a critical point of $f(x)$ then $x = c$ is

1. a rel. max. of $f(x)$ if $f'(x) > 0$ to the left of $x = c$ and $f'(x) < 0$ to the right of $x = c$.
2. a rel. min. of $f(x)$ if $f'(x) < 0$ to the left of $x = c$ and $f'(x) > 0$ to the right of $x = c$.
3. not a relative extrema of $f(x)$ if $f'(x)$ is the same sign on both sides of $x = c$.

2nd Derivative Test

If $x = c$ is a critical point of $f(x)$ such that $f'(c) = 0$ then $x = c$

1. is a relative maximum of $f(x)$ if $f''(c) < 0$.
2. is a relative minimum of $f(x)$ if $f''(c) > 0$.
3. may be a relative maximum, relative minimum, or neither if $f''(c) = 0$.

Finding Relative Extrema and/or Classify Critical Points

1. Find all critical points of $f(x)$.
2. Use the 1st derivative test or the 2nd derivative test on each critical point.

Mean Value Theorem

If $f(x)$ is continuous on the closed interval $[a,b]$ and differentiable on the open interval (a,b) then there is a number $a < c < b$ such that $f'(c) = \dfrac{f(b) - f(a)}{b - a}$.

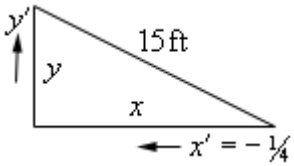
Newton's Method

If x_n is the n^{th} guess for the root/solution of $f(x) = 0$ then $(n+1)^{\text{st}}$ guess is $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ provided $f'(x_n)$ exists.

Related Rates

Sketch picture and identify known/unknown quantities. Write down equation relating quantities and differentiate with respect to *t* using implicit differentiation (*i.e.* add on a derivative every time you differentiate a function of *t*). Plug in known quantities and solve for the unknown quantity.

Ex. A 15 foot ladder is resting against a wall. The bottom is initially 10 ft away and is being pushed towards the wall at $\frac{1}{4}$ ft/sec. How fast is the top moving after 12 sec?



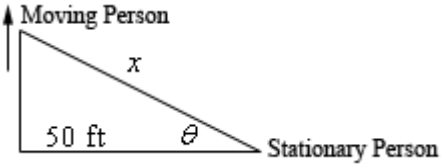
x' is negative because x is decreasing. Using Pythagorean Theorem and differentiating,

$$x^2 + y^2 = 15^2 \quad \Rightarrow \quad 2x\,x' + 2y\,y' = 0$$

After 12 sec we have $x = 10 - 12\left(\frac{1}{4}\right) = 7$ and so $y = \sqrt{15^2 - 7^2} = \sqrt{176}$. Plug in and solve for y'.

$$7\left(-\tfrac{1}{4}\right) + \sqrt{176}\; y' = 0 \;\Rightarrow\; y' = \frac{7}{4\sqrt{176}} \text{ ft/sec}$$

Ex. Two people are 50 ft apart when one starts walking north. The angle θ changes at 0.01 rad/min. At what rate is the distance between them changing when $\theta = 0.5$ rad?



We have $\theta' = 0.01$ rad/min. and want to find x'. We can use various trig fcns but easiest is,

$$\sec\theta = \frac{x}{50} \quad \Rightarrow \quad \sec\theta\tan\theta\;\theta' = \frac{x'}{50}$$

We know $\theta = 0.5$ so plug in θ' and solve.

$$\sec(0.5)\tan(0.5)(0.01) = \frac{x'}{50}$$

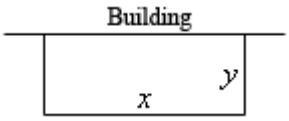
$$x' = 0.3112 \text{ ft/sec}$$

Remember to have calculator in radians!

Optimization

Sketch picture if needed, write down equation to be optimized and constraint. Solve constraint for one of the two variables and plug into first equation. Find critical points of equation in range of variables and verify that they are min/max as needed.

Ex. We're enclosing a rectangular field with 500 ft of fence material and one side of the field is a building. Determine dimensions that will maximize the enclosed area.



Maximize $A = xy$ subject to constraint of $x + 2y = 500$. Solve constraint for x and plug into area.

$$x = 500 - 2y \;\Rightarrow\; \begin{aligned} A &= y(500 - 2y) \\ &= 500y - 2y^2 \end{aligned}$$

Differentiate and find critical point(s).

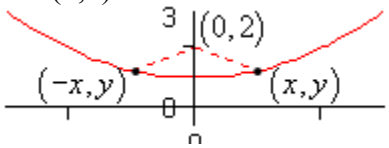
$$A' = 500 - 4y \;\Rightarrow\; y = 125$$

By 2nd deriv. test this is a rel. max. and so is the answer we're after. Finally, find x.

$$x = 500 - 2(125) = 250$$

The dimensions are then 250 x 125.

Ex. Determine point(s) on $y = x^2 + 1$ that are closest to (0,2).



Minimize $f = d^2 = (x-0)^2 + (y-2)^2$ and the constraint is $y = x^2 + 1$. Solve constraint for x^2 and plug into the function.

$$x^2 = y - 1 \;\Rightarrow\; f = x^2 + (y-2)^2$$

$$= y - 1 + (y-2)^2 = y^2 - 3y + 3$$

Differentiate and find critical point(s).

$$f' = 2y - 3 \quad \Rightarrow \quad y = \tfrac{3}{2}$$

By the 2nd derivative test this is a rel. min. and so all we need to do is find x value(s).

$$x^2 = \tfrac{3}{2} - 1 = \tfrac{1}{2} \quad \Rightarrow \quad x = \pm\tfrac{1}{\sqrt{2}}$$

The 2 points are then $\left(\frac{1}{\sqrt{2}}, \frac{3}{2}\right)$ and $\left(-\frac{1}{\sqrt{2}}, \frac{3}{2}\right)$.

Integrals

Definitions

Definite Integral: Suppose $f(x)$ is continuous on $[a,b]$. Divide $[a,b]$ into n subintervals of width Δx and choose x_i^* from each interval. Then $\int_a^b f(x)\,dx = \lim_{n\to\infty}\sum_{i=1}^{\infty} f(x_i^*)\Delta x$.

Anti-Derivative : An anti-derivative of $f(x)$ is a function, $F(x)$, such that $F'(x) = f(x)$.

Indefinite Integral : $\int f(x)\,dx = F(x) + c$ where $F(x)$ is an anti-derivative of $f(x)$.

Fundamental Theorem of Calculus

Part I : If $f(x)$ is continuous on $[a,b]$ then $g(x) = \int_a^x f(t)\,dt$ is also continuous on $[a,b]$ and $g'(x) = \frac{d}{dx}\int_a^x f(t)\,dt = f(x)$.

Part II : $f(x)$ is continuous on $[a,b]$, $F(x)$ is an anti-derivative of $f(x)$ (*i.e.* $F(x) = \int f(x)\,dx$) then $\int_a^b f(x)\,dx = F(b) - F(a)$.

Variants of Part I :

$$\frac{d}{dx}\int_a^{u(x)} f(t)\,dt = u'(x) f\left[u(x)\right]$$

$$\frac{d}{dx}\int_{v(x)}^b f(t)\,dt = -v'(x) f\left[v(x)\right]$$

$$\frac{d}{dx}\int_{v(x)}^{u(x)} f(t)\,dt = u'(x) f\left[u(x)\right] - v'(x) f\left[v(x)\right]$$

Properties

$\int f(x) \pm g(x)\,dx = \int f(x)\,dx \pm \int g(x)\,dx$

$\int_a^b f(x) \pm g(x)\,dx = \int_a^b f(x)\,dx \pm \int_a^b g(x)\,dx$

$\int_a^a f(x)\,dx = 0$

$\int_a^b f(x)\,dx = -\int_b^a f(x)\,dx$

$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx$ for any value of c.

$\int cf(x)\,dx = c\int f(x)\,dx$, c is a constant

$\int_a^b cf(x)\,dx = c\int_a^b f(x)\,dx$, c is a constant

$\int_a^b c\,dx = c(b-a)$

$\left|\int_a^b f(x)\,dx\right| \le \int_a^b |f(x)|\,dx$

If $f(x) \ge g(x)$ on $a \le x \le b$ then $\int_a^b f(x)\,dx \ge \int_a^b g(x)\,dx$

If $f(x) \ge 0$ on $a \le x \le b$ then $\int_a^b f(x)\,dx \ge 0$

If $m \le f(x) \le M$ on $a \le x \le b$ then $m(b-a) \le \int_a^b f(x)\,dx \le M(b-a)$

Common Integrals

$\int k\,dx = kx + c$

$\int x^n\,dx = \frac{1}{n+1}x^{n+1} + c, n \ne -1$

$\int x^{-1}\,dx = \int \frac{1}{x}\,dx = \ln|x| + c$

$\int \frac{1}{ax+b}\,dx = \frac{1}{a}\ln|ax+b| + c$

$\int \ln u\,du = u\ln(u) - u + c$

$\int \mathbf{e}^u\,du = \mathbf{e}^u + c$

$\int \cos u\,du = \sin u + c$

$\int \sin u\,du = -\cos u + c$

$\int \sec^2 u\,du = \tan u + c$

$\int \sec u\tan u\,du = \sec u + c$

$\int \csc u\cot u\,du = -\csc u + c$

$\int \csc^2 u\,du = -\cot u + c$

$\int \tan u\,du = \ln|\sec u| + c$

$\int \sec u\,du = \ln|\sec u + \tan u| + c$

$\int \frac{1}{a^2+u^2}\,du = \frac{1}{a}\tan^{-1}\left(\frac{u}{a}\right) + c$

$\int \frac{1}{\sqrt{a^2-u^2}}\,du = \sin^{-1}\left(\frac{u}{a}\right) + c$

Standard Integration Techniques

Note that at many schools all but the Substitution Rule tend to be taught in a Calculus II class.

***u* Substitution :** The substitution $u = g(x)$ will convert $\int_a^b f(g(x))g'(x)\,dx = \int_{g(a)}^{g(b)} f(u)\,du$ using $du = g'(x)\,dx$. For indefinite integrals drop the limits of integration.

Ex. $\int_1^2 5x^2\cos(x^3)\,dx$

$u = x^3 \;\Rightarrow\; du = 3x^2dx \;\Rightarrow\; x^2dx = \frac{1}{3}du$

$x = 1 \;\Rightarrow\; u = 1^3 = 1 \;\because\; x = 2 \;\Rightarrow\; u = 2^3 = 8$

$$\int_1^2 5x^2\cos(x^3)\,dx = \int_1^8 \tfrac{5}{3}\cos(u)\,du$$
$$= \tfrac{5}{3}\sin(u)\Big|_1^8 = \tfrac{5}{3}(\sin(8) - \sin(1))$$

Integration by Parts : $\int u\,dv = uv - \int v\,du$ and $\int_a^b u\,dv = uv\Big|_a^b - \int_a^b v\,du$. Choose *u* and *dv* from integral and compute *du* by differentiating *u* and compute *v* using $v = \int dv$.

Ex. $\int xe^{-x}\,dx$

$u = x \quad dv = e^{-x} \quad\Rightarrow\quad du = dx \quad v = -e^{-x}$

$$\int xe^{-x}\,dx = -xe^{-x} + \int e^{-x}\,dx = -xe^{-x} - e^{-x} + c$$

Ex. $\int_3^5 \ln x\,dx$

$u = \ln x \quad dv = dx \quad\Rightarrow\quad du = \frac{1}{x}dx \quad v = x$

$$\int_3^5 \ln x\,dx = x\ln x\Big|_3^5 - \int_3^5 dx = (x\ln(x) - x)\Big|_3^5$$
$$= 5\ln(5) - 3\ln(3) - 2$$

Products and (some) Quotients of Trig Functions

For $\int \sin^n x\cos^m x\,dx$ we have the following :

1. ***n* odd.** Strip 1 sine out and convert rest to cosines using $\sin^2 x = 1 - \cos^2 x$, then use the substitution $u = \cos x$.
2. ***m* odd.** Strip 1 cosine out and convert rest to sines using $\cos^2 x = 1 - \sin^2 x$, then use the substitution $u = \sin x$.
3. ***n* and *m* both odd.** Use either 1. or 2.
4. ***n* and *m* both even.** Use double angle and/or half angle formulas to reduce the integral into a form that can be integrated.

For $\int \tan^n x\sec^m x\,dx$ we have the following :

1. ***n* odd.** Strip 1 tangent and 1 secant out and convert the rest to secants using $\tan^2 x = \sec^2 x - 1$, then use the substitution $u = \sec x$.
2. ***m* even.** Strip 2 secants out and convert rest to tangents using $\sec^2 x = 1 + \tan^2 x$, then use the substitution $u = \tan x$.
3. ***n* odd and *m* even.** Use either 1. or 2.
4. ***n* even and *m* odd.** Each integral will be dealt with differently.

Trig Formulas **:** $\sin(2x) = 2\sin(x)\cos(x)$, $\cos^2(x) = \frac{1}{2}(1 + \cos(2x))$, $\sin^2(x) = \frac{1}{2}(1 - \cos(2x))$

Ex. $\int \tan^3 x\sec^5 x\,dx$

$$\int \tan^3 x\sec^5 x\,dx = \int \tan^2 x\sec^4 x\tan x\sec x\,dx$$
$$= \int (\sec^2 x - 1)\sec^4 x\tan x\sec x\,dx$$
$$= \int (u^2 - 1)u^4\,du \qquad (u = \sec x)$$
$$= \tfrac{1}{7}\sec^7 x - \tfrac{1}{5}\sec^5 x + c$$

Ex. $\int \frac{\sin^5 x}{\cos^3 x}\,dx$

$$\int \frac{\sin^5 x}{\cos^3 x}\,dx = \int \frac{\sin^4 x\sin x}{\cos^3 x}\,dx = \int \frac{(\sin^2 x)^2\sin x}{\cos^3 x}\,dx$$
$$= \int \frac{(1-\cos^2 x)^2\sin x}{\cos^3 x}\,dx \qquad (u = \cos x)$$
$$= -\int \frac{(1-u^2)^2}{u^3}\,du = -\int \frac{1 - 2u^2 + u^4}{u^3}\,du$$
$$= \tfrac{1}{2}\sec^2 x + 2\ln|\cos x| - \tfrac{1}{2}\cos^2 x + c$$

Trig Substitutions : If the integral contains the following root use the given substitution and formula to convert into an integral involving trig functions.

$\sqrt{a^2-b^2x^2} \;\Rightarrow\; x=\frac{a}{b}\sin\theta$	$\sqrt{b^2x^2-a^2} \;\Rightarrow\; x=\frac{a}{b}\sec\theta$	$\sqrt{a^2+b^2x^2} \;\Rightarrow\; x=\frac{a}{b}\tan\theta$
$\cos^2\theta=1-\sin^2\theta$	$\tan^2\theta=\sec^2\theta-1$	$\sec^2\theta=1+\tan^2\theta$

Ex. $\int \frac{16}{x^2\sqrt{4-9x^2}}\,dx$

$x=\frac{2}{3}\sin\theta \;\Rightarrow\; dx=\frac{2}{3}\cos\theta\,d\theta$

$\sqrt{4-9x^2}=\sqrt{4-4\sin^2\theta}=\sqrt{4\cos^2\theta}=2\left|\cos\theta\right|$

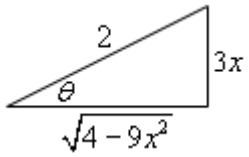
Recall $\sqrt{x^2}=\left|x\right|$. Because we have an indefinite integral we'll assume positive and drop absolute value bars. If we had a definite integral we'd need to compute θ's and remove absolute value bars based on that and,

$$\left|x\right|=\begin{cases} x & \text{if } x\ge 0\\ -x & \text{if } x<0\end{cases}$$

In this case we have $\sqrt{4-9x^2}=2\cos\theta$.

$$\int \frac{16}{\frac{4}{9}\sin^2\theta(2\cos\theta)}\left(\tfrac{2}{3}\cos\theta\right)d\theta=\int\frac{12}{\sin^2\theta}\,d\theta$$
$$=\int 12\csc^2\theta\,d\theta=-12\cot\theta+c$$

Use Right Triangle Trig to go back to x's. From substitution we have $\sin\theta=\frac{3x}{2}$ so,



From this we see that $\cot\theta=\frac{\sqrt{4-9x^2}}{3x}$. So,

$$\int \frac{16}{x^2\sqrt{4-9x^2}}\,dx=-\frac{4\sqrt{4-9x^2}}{x}+c$$

Partial Fractions : If integrating $\int \frac{P(x)}{Q(x)}\,dx$ where the degree of $P(x)$ is smaller than the degree of $Q(x)$. Factor denominator as completely as possible and find the partial fraction decomposition of the rational expression. Integrate the partial fraction decomposition (P.F.D.). For each factor in the denominator we get term(s) in the decomposition according to the following table.

Factor in $Q(x)$	Term in P.F.D	Factor in $Q(x)$	Term in P.F.D
$ax+b$	$\frac{A}{ax+b}$	$(ax+b)^k$	$\frac{A_1}{ax+b}+\frac{A_2}{(ax+b)^2}+\cdots+\frac{A_k}{(ax+b)^k}$
ax^2+bx+c	$\frac{Ax+B}{ax^2+bx+c}$	$(ax^2+bx+c)^k$	$\frac{A_1x+B_1}{ax^2+bx+c}+\cdots+\frac{A_kx+B_k}{(ax^2+bx+c)^k}$

Ex. $\int \frac{7x^2+13x}{(x-1)(x^2+4)}\,dx$

$$\int \frac{7x^2+13x}{(x-1)(x^2+4)}\,dx=\int \frac{4}{x-1}+\frac{3x+16}{x^2+4}\,dx$$
$$=\int \frac{4}{x-1}+\frac{3x}{x^2+4}+\frac{16}{x^2+4}\,dx$$
$$=4\ln\left|x-1\right|+\tfrac{3}{2}\ln\left(x^2+4\right)+8\tan^{-1}\left(\tfrac{x}{2}\right)$$

Here is partial fraction form and recombined.

$$\frac{7x^2+13x}{(x-1)(x^2+4)}=\frac{A}{x-1}+\frac{Bx+C}{x^2+4}=\frac{A(x^2+4)+(Bx+C)(x-1)}{(x-1)(x^2+4)}$$

Set numerators equal and collect like terms.

$$7x^2+13x=(A+B)x^2+(C-B)x+4A-C$$

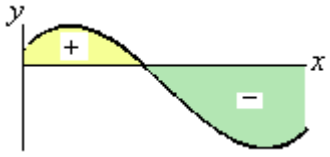
Set coefficients equal to get a system and solve to get constants.

$$A+B=7 \qquad C-B=13 \qquad 4A-C=0$$
$$A=4 \qquad B=3 \qquad C=16$$

An alternate method that *sometimes* works to find constants. Start with setting numerators equal in previous example : $7x^2+13x=A\left(x^2+4\right)+\left(Bx+C\right)\left(x-1\right)$. Chose *nice* values of x and plug in. For example if $x=1$ we get $20=5A$ which gives $A=4$. This won't always work easily.

Applications of Integrals

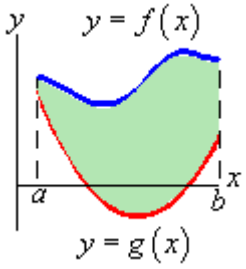
Net Area : $\int_a^b f(x)\,dx$ represents the net area between $f(x)$ and the x-axis with area above x-axis positive and area below x-axis negative.



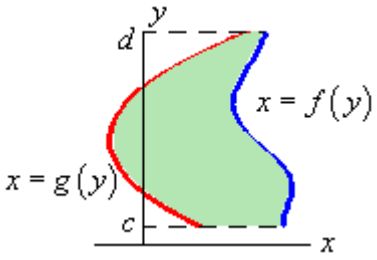
Area Between Curves : The general formulas for the two main cases for each are,

$y = f(x) \Rightarrow A = \int_a^b [\text{upper function}] - [\text{lower function}]\,dx$ & $x = f(y) \Rightarrow A = \int_c^d [\text{right function}] - [\text{left function}]\,dy$

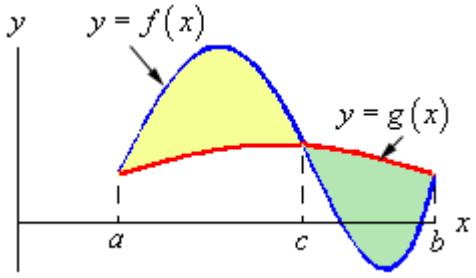
If the curves intersect then the area of each portion must be found individually. Here are some sketches of a couple possible situations and formulas for a couple of possible cases.



$A = \int_a^b f(x) - g(x)\,dx$



$A = \int_c^d f(y) - g(y)\,dy$



$A = \int_a^c f(x) - g(x)\,dx + \int_c^b g(x) - f(x)\,dx$

Volumes of Revolution : The two main formulas are $V = \int A(x)\,dx$ and $V = \int A(y)\,dy$. Here is some general information about each method of computing and some examples.

Rings

$A = \pi\left((\text{outer radius})^2 - (\text{inner radius})^2\right)$

Limits: x/y of right/bot ring to x/y of left/top ring

Horz. Axis use $f(x)$, $g(x)$, $A(x)$ and dx.

Vert. Axis use $f(y)$, $g(y)$, $A(y)$ and dy.

Cylinders

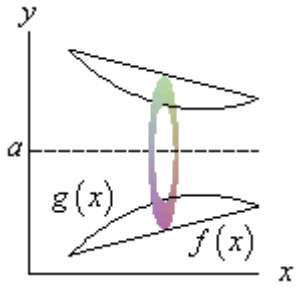
$A = 2\pi\,(\text{radius})(\text{width / height})$

Limits : x/y of inner cyl. to x/y of outer cyl.

Horz. Axis use $f(y)$, $g(y)$, $A(y)$ and dy.

Vert. Axis use $f(x)$, $g(x)$, $A(x)$ and dx.

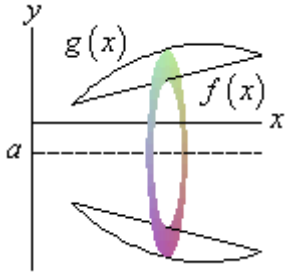
Ex. Axis : $y = a > 0$



outer radius : $a - f(x)$

inner radius : $a - g(x)$

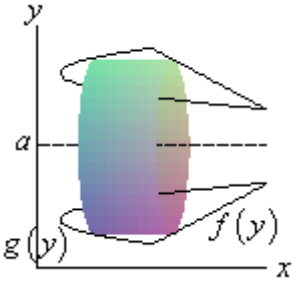
Ex. Axis : $y = a \le 0$



outer radius: $|a| + g(x)$

inner radius: $|a| + f(x)$

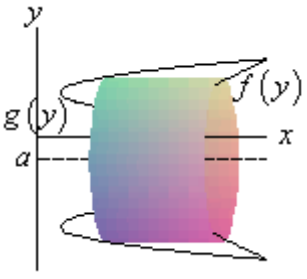
Ex. Axis : $y = a > 0$



radius : $a - y$

width : $f(y) - g(y)$

Ex. Axis : $y = a \le 0$



radius : $|a| + y$

width : $f(y) - g(y)$

These are only a few cases for horizontal axis of rotation. If axis of rotation is the x-axis use the $y = a \le 0$ case with $a = 0$. For vertical axis of rotation ($x = a > 0$ and $x = a \le 0$) interchange x and y to get appropriate formulas.

Work : If a force of $F(x)$ moves an object in $a \le x \le b$, the work done is $W = \int_a^b F(x)\,dx$

Average Function Value : The average value of $f(x)$ on $a \le x \le b$ is $f_{avg} = \frac{1}{b-a}\int_a^b f(x)\,dx$

Arc Length Surface Area : Note that this is often a Calc II topic. The three basic formulas are,

$L = \int_a^b ds$ $\quad SA = \int_a^b 2\pi y\,ds$ (rotate about x-axis) $\quad SA = \int_a^b 2\pi x\,ds$ (rotate about y-axis)

where ds is dependent upon the form of the function being worked with as follows.

$ds = \sqrt{1+\left(\frac{dy}{dx}\right)^2}\,dx$ if $y = f(x),\ a \le x \le b$ $\quad ds = \sqrt{\left(\frac{dx}{dt}\right)^2+\left(\frac{dy}{dt}\right)^2}\,dt$ if $x = f(t), y = g(t),\ a \le t \le b$

$ds = \sqrt{1+\left(\frac{dx}{dy}\right)^2}\,dy$ if $x = f(y),\ a \le y \le b$ $\quad ds = \sqrt{r^2+\left(\frac{dr}{d\theta}\right)^2}\,d\theta$ if $r = f(\theta),\ a \le \theta \le b$

With surface area you *may* have to substitute in for the x or y depending on your choice of ds to match the differential in the ds. With parametric and polar you will always need to substitute.

Improper Integral

An improper integral is an integral with one or more infinite limits and/or discontinuous integrands. Integral is called convergent if the limit exists and has a finite value and divergent if the limit doesn't exist or has infinite value. This is typically a Calc II topic.

Infinite Limit

1. $\int_a^\infty f(x)\,dx = \lim_{t\to\infty}\int_a^t f(x)\,dx$
2. $\int_{-\infty}^b f(x)\,dx = \lim_{t\to-\infty}\int_t^b f(x)\,dx$
3. $\int_{-\infty}^\infty f(x)\,dx = \int_{-\infty}^c f(x)\,dx + \int_c^\infty f(x)\,dx$ provided BOTH integrals are convergent.

Discontinuous Integrand

1. Discont. at a: $\int_a^b f(x)\,dx = \lim_{t\to a^+}\int_t^b f(x)\,dx$
2. Discont. at b : $\int_a^b f(x)\,dx = \lim_{t\to b^-}\int_a^t f(x)\,dx$
3. Discontinuity at $a < c < b$: $\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx$ provided both are convergent.

Comparison Test for Improper Integrals : If $f(x) \ge g(x) \ge 0$ on $[a,\infty)$ then,

1. If $\int_a^\infty f(x)\,dx$ conv. then $\int_a^\infty g(x)\,dx$ conv.
2. If $\int_a^\infty g(x)\,dx$ divg. then $\int_a^\infty f(x)\,dx$ divg.

Useful fact : If $a > 0$ then $\int_a^\infty \frac{1}{x^p}\,dx$ converges if $p > 1$ and diverges for $p \le 1$.

Approximating Definite Integrals

For given integral $\int_a^b f(x)\,dx$ and a n (must be even for Simpson's Rule) define $\Delta x = \frac{b-a}{n}$ and divide $[a,b]$ into n subintervals $[x_0,x_1]$, $[x_1,x_2]$, … , $[x_{n-1},x_n]$ with $x_0 = a$ and $x_n = b$ then,

Midpoint Rule : $\int_a^b f(x)\,dx \approx \Delta x\left[f(x_1^*)+f(x_2^*)+\cdots+f(x_n^*)\right]$, x_i^* is midpoint $[x_{i-1},x_i]$

Trapezoid Rule : $\int_a^b f(x)\,dx \approx \frac{\Delta x}{2}\left[f(x_0)+2f(x_1)++2f(x_2)+\cdots+2f(x_{n-1})+f(x_n)\right]$

Simpson's Rule : $\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left[f(x_0)+4f(x_1)+2f(x_2)+\cdots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)\right]$

Microsoft Azure Machine Learning: Algorithm Cheat Sheet

This cheat sheet helps you choose the best Azure Machine Learning algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.

START

Finding unusual data points

Discovering structure

Predicting categories

Predicting values

Three or more

Two

ANOMALY DETECTION

- One-class SVM — >100 features, aggressive boundary
- PCA-based anomaly detection — Fast training

CLUSTERING

- K-means

MULTICLASS CLASSIFICATION

- Fast training, linear model — Multiclass logistic regression
- Accuracy, long training times — Multiclass neural network
- Accuracy, fast training — Multiclass decision forest
- Accuracy, small memory footprint — Multiclass decision jungle
- Depends on the two-class classifier, see notes below — One-v-all multiclass

REGRESSION

- Ordinal regression — Data in rank ordered categories
- Poisson regression — Predicting event counts
- Fast forest quantile regression — Predicting a distribution
- Linear regression — Fast training, linear model
- Bayesian linear regression — Linear model, small data sets
- Neural network regression — Accuracy, long training time
- Decision forest regression — Accuracy, fast training
- Boosted decision tree regression — Accuracy, fast training

TWO-CLASS CLASSIFICATION

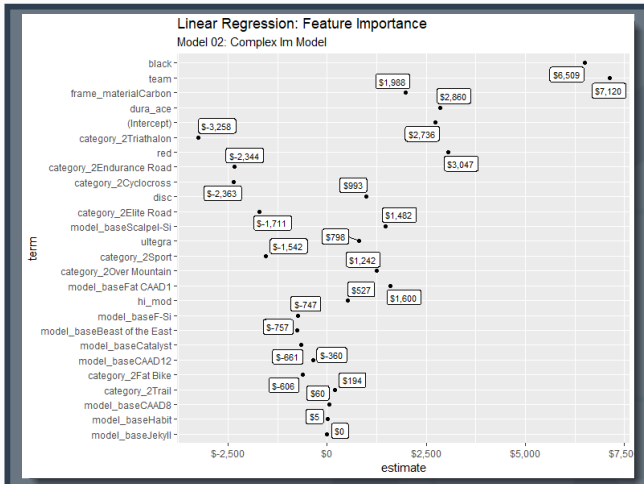
- Two-class SVM — >100 features, linear model
- Two-class averaged perceptron — Fast training, linear model
- Two-class logistic regression — Fast training, linear model
- Two-class Bayes point machine — Fast training, linear model

- Accuracy, fast training — Two-class decision forest
- Accuracy, fast training — Two-class boosted decision tree
- Accuracy, small memory footprint — Two-class decision jungle
- >100 features — Two-class locally deep SVM
- Accuracy, long training times — Two-class neural network

Created by the Azure Machine Learning Team
Email: AzurePoster@microsoft.com
Download this poster: http://aka.ms/MLCheatSheet

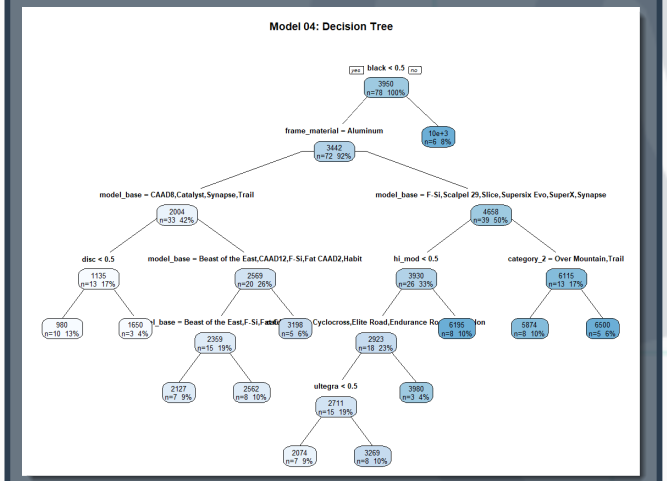
Microsoft

Regression (Machine Learning)



Linear Regression with Multiple Predictors

Each predictor (term) is interpretable meaning the value (estimate) indicates an increase/decrease in the target



Decision Tree (Regression)

Decisions are based on binary rules that split the nodes and terminate at leaves. Regression trees estimate the value at each node.

Summary:

- **Common Applications in Business**: Used to predict a *numeric value* (e.g. forecasting sales, estimating prices, etc).
- **Key Concept:** Data is usually in a rectangular format (like a spreadsheet) with one column that is a *target* (e.g. price) and other columns that are *predictors* (e.g. product category)
- **Gotchas:**
  - **Preprocessing:** Knowing when to preprocess data (normalize) prior to machine learning step
  - **Feature Engineering:** Getting good features is more important than applying complex models.
- **Parameter Tuning:** Higher complexity models have many parameters that can be tuned.
- **Interpretability:** Some models are more explainable than others, meaning the estimates for each feature means something in relation to the target. Other models are not interpretable and require additional tools (e.g. LIME) to explain.

Terminology:

- **Supervised vs Unsupervised:** Regression is a supervised technique that requires training with a "**target**" (e.g. price of product or sales by month). The algorithm learns by identifying relationships between the target & the **predictors** (attributes related to the target like category of product or month of sales).
- **Classification vs Regression:** Classification aims to predict classes (either binary yes/no or multi-class categorical). Regression aims to predict a numeric value (e.g. product price = $4,233).
- **Preprocessing**: Many algorithms require preprocessing, which transforms the data into a format more suitable for the machine learning algorithm. A common example is "**standardization**" or scaling the feature to be in a range of [0,1] (or close to it).
- **Hyper Parameter & Tuning:** Machine learning algorithms have many parameters that can be adjusted (e.g. learning rate in GBM). Tuning is the process of systematically finding the optimum parameter values.
- **Cross Validation**: Machine learning algorithms should be tuned on a validation set as opposed to a test set. Cross-validation is the process of splitting the training set into multiple sets using a portion of the training set for tuning.
- **Performance Metrics (Regression)**: Common performance metrics are **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**. These measures provide an estimate of model performance to compare models to each other.

R Cheat Sheet

Parsnip (Machine Learning):

- **Model List** (start here first)
- **Linear Regression & GLM**
- **Decision Tree**
- **Random Forest**
- **Boosted Trees** (XGBoost)
- SVM: **Poly** & **Radial**

Keras (Deep Learning)

H2O (ML & DL Framework)

MLR (ML Framework)

Python Cheat Sheet

Scikit-Learn (Machine Learning):

- **Linear Regression**
- **GLM (Elastic Net)**
- **Decision Tree (Regressor)**
- **Random Forest (Regressor)**
- **AdaBoost (Regressor)** , **XGBoost**
- **SVM (Regressor)**

Keras (Deep Learning)

H2O (ML & DL Framework)

Resources

- Business Analysis With R Course (DS4B 101-R) - Modeling - Week 6
- Business Science Problem Framework
- Ultimate R Cheat Sheet I Ultimate Python Cheat Sheet

Data Science Courses for Business

Business Science University
university.business-science.io

version: 1.1

Machine Learning Algorithms - Regression

Key Attributes Table

Popular Algorithms	Type	Key Concepts	Feature Range Standardization [0, 1]	Results Interpretable?	Key Parameters
Linear Regression	Linear	Simplest method - Uses OLS to reduce error and find the	Not Required	Yes - Model terms indicate magnitude / direction of each features contribution	N/A
GLM (Generalized Linear Model) LASSO, Ridge Regression, Elastic Net	Linear	Linear method that penalizes irrelevant features using a concept called "Regularization", where the weight of irrelevant features is reduced to make their effect on the model lower. L1 Regularization - Called LASSO regression L2 Regularization - Called Ridge Regression Elastic Net: Combines L1 & L2 Regularization	Required (but see Application Note below). **Application Note**: In practice, some algorithms (i.e. R's glmnet::glmnet()) implement standardization internally and re-scale prior to returning term estimates and predictions. This means that features need not be scaled prior to use.	Yes, if standardization is performed internally to algorithm. Model terms indicate magnitude / direction of each features contribution	Penalty (alpha) - How much to penalize the parameters Mixture (L1 Ratio) - Ratio between L1 and L2 Regularization
Decision Tree	Tree-Based (Non-Linear)	A decision tree is a set of decision rules. Each rule is considered a node with a split being a binary decision. The decisions terminate at a leaf.	Not Required	Yes - Decision Tree Plots show rule-based decisions that show how to arrive at model prediction	Max Tree Depth - How many splits for the longest tree Min Samples Per Leaf / Node - How many samples in each end node (leaf) Cost Complexity (Cp) / Min Impurity - Instructs when to stop (create a leaf) if additional information gain is not above a Cp threshold
Random Forest	Tree-Based (Non-Linear)	Ensemble learning method where many trees are created on sub-samples of data set and combined using averaging. This process controls overfitting, typicallying leading to a more accurate model. However, because the models are combined, the decision rules become incomprehensible. This process is often called "Bagging".	Not Required	No (see Application Note) **Application Note:** Feature importance can be obtained with additional methods for global (Variable Importance) and local (e.g. LIME) model understanding.	See Decision Tree Key Parameters, and: Replacement - whether or not to draw samples with replacement Number of Features - How many columns to use when sampling Number of Trees - How many trees to average
GBM (Gradient Boosted Machine) XGBoost	Tree-Based (Non-Linear)	Implements a technique called "Boosting" to build decision trees of weak prediction models and generalizes using a loss function. The weak learners converge to a strong learner.	Not Required	No (see Application Note) **Application Note:** Feature importance can be obtained with additional methods for global (Variable Importance) and local (e.g. LIME) model understanding.	See RandomForest Key Parameters, and: Learning Rate (eta) - The rate that the boosting algorithm adapts Loss Reduction (gamma) - The loss function to use during splitting Sample Size - The proportion of data exposed to the model during each iteration
SVM (Support Vector Machine)	Kernel Basis (Polynomial or Radial) (Non-Linear)	An algorithm that uses a kernel to transform the feature space to linearly seperable boundaries, and then applies a margin penalizing points that are incorrectly measured outside of the margin. The kernel transformation (i.e. radial, polynomial) makes it possible to perform linear separations within non-linear data.	Required (but see Application Note below). **Application Note**: In practice, some algorithms (i.e. R's kernlabs::ksvm()) implement standardization internally and re-scale prior to returning term estimates and predictions. This means that features need not be scaled prior to use.	No (see Application Note) **Application Note:** Feature importance can be obtained with additional methods for global (Variable Importance) and local (e.g. LIME) model understanding.	Kernel - Polynomial or Radial Basis Function Cost / Regularization - Cost of predicting sample on wrong side of the SVM margin Margin (Epsilon) - Specifies region where no penalty is applied Degree (Polynomial) - Degree of Polynomial. Use 1 for linear, 2 or more for flexible (quadratic) Scale Factor (Polynomial) - Factor to adjust bias/variance Gamma or Sigma (Radial) - Factor to adjust bias/variance
Deep Learning (Neural Network)	Neural Network (Non-Linear)	Learning algorithms with input and output and layers in between where the model parameters are learned. The user develops the architecture of the neural network, and the algorithm learns the model through iteratively seeking to minimize a cost function.	Required	No (see Application Note) **Application Note:** Feature importance can be obtained with additional methods for global (Variable Importance) and local (e.g. LIME) model understanding.	Many tuning parameters & architecture decisions

VIP Cheatsheet: Supervised Learning

Afshine Amidi and Shervine Amidi

September 9, 2018

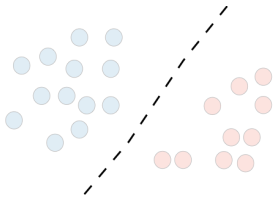
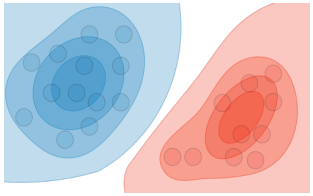
Introduction to Supervised Learning

Given a set of data points $\{x^{(1)},...,x^{(m)}\}$ associated to a set of outcomes $\{y^{(1)},...,y^{(m)}\}$, we want to build a classifier that learns how to predict y from x.

❒ **Type of prediction** – The different types of predictive models are summed up in the table below:

	Regression	**Classifier**
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

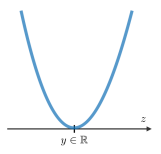
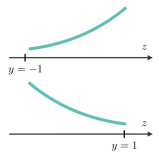
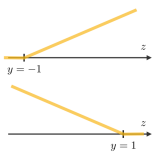
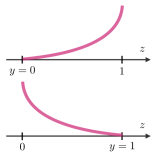
❒ **Type of model** – The different models are summed up in the table below:

	Discriminative model	**Generative model**
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Notations and general concepts

❒ **Hypothesis** – The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$, the model prediction output is $h_\theta(x^{(i)})$.

❒ **Loss function** – A loss function is a function $L:(z,y)\in\mathbb{R}\times Y\longmapsto L(z,y)\in\mathbb{R}$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are. The common loss functions are summed up in the table below:

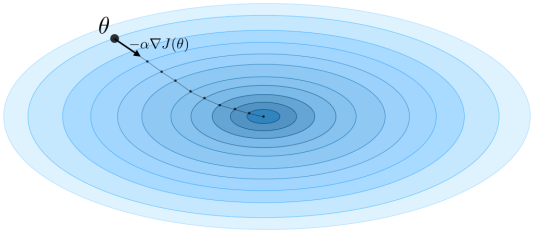
Least squared	**Logistic**	**Hinge**	**Cross-entropy**
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

❒ **Cost function** – The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$J(\theta)=\sum_{i=1}^{m}L(h_\theta(x^{(i)}),y^{(i)})$$

❒ **Gradient descent** – By noting $\alpha\in\mathbb{R}$ the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function J as follows:

$$\theta\longleftarrow\theta-\alpha\nabla J(\theta)$$



Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

❒ **Likelihood** – The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through maximizing the likelihood. In practice, we use the log-likelihood $\ell(\theta)=\log(L(\theta))$ which is easier to optimize. We have:

$$\theta^{\text{opt}}=\arg\max_\theta L(\theta)$$

❒ **Newton's algorithm** – The Newton's algorithm is a numerical method that finds θ such that $\ell'(\theta)=0$. Its update rule is as follows:

$$\theta\leftarrow\theta-\frac{\ell'(\theta)}{\ell''(\theta)}$$

Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta\leftarrow\theta-\left(\nabla^2_\theta\ell(\theta)\right)^{-1}\nabla_\theta\ell(\theta)$$

Linear regression

We assume here that $y|x;\theta \sim \mathcal{N}(\mu,\sigma^2)$

❒ Normal equations – By noting X the matrix design, the value of θ that minimizes the cost function is a closed-form solution such that:

$$\boxed{\theta = (X^TX)^{-1}X^Ty}$$

❒ LMS algorithm – By noting α the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of m data points, which is also known as the Widrow-Hoff learning rule, is as follows:

$$\boxed{\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^{m} \left[y^{(i)} - h_\theta(x^{(i)})\right] x_j^{(i)}}$$

Remark: the update rule is a particular case of the gradient ascent.

❒ LWR – Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by $w^{(i)}(x)$, which is defined with parameter $\tau \in \mathbb{R}$ as:

$$\boxed{w^{(i)}(x) = \exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)}$$

Classification and logistic regression

❒ Sigmoid function – The sigmoid function g, also known as the logistic function, is defined as follows:

$$\forall z \in \mathbb{R}, \quad \boxed{g(z) = \frac{1}{1+e^{-z}} \in]0,1[}$$

❒ Logistic regression – We assume here that $y|x;\theta \sim \text{Bernoulli}(\phi)$. We have the following form:

$$\boxed{\phi = p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^Tx)} = g(\theta^Tx)}$$

Remark: there is no closed form solution for the case of logistic regressions.

❒ Softmax regression – A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, we set $\theta_K = 0$, which makes the Bernoulli parameter ϕ_i of each class i equal to:

$$\boxed{\phi_i = \frac{\exp(\theta_i^Tx)}{\sum_{j=1}^{K}\exp(\theta_j^Tx)}}$$

Generalized Linear Models

❒ Exponential family – A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter, also called the canonical parameter or link function, η, a sufficient statistic $T(y)$ and a log-partition function $a(\eta)$ as follows:

$$\boxed{p(y;\eta) = b(y)\exp(\eta T(y) - a(\eta))}$$

Remark: we will often have $T(y) = y$. Also, $\exp(-a(\eta))$ can be seen as a normalization parameter that will make sure that the probabilities sum to one.

Here are the most common exponential distributions summed up in the following table:

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1+\exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^η	$\frac{1}{y!}$
Geometric	$\log(1-\phi)$	y	$\log\left(\frac{e^\eta}{1-e^\eta}\right)$	1

❒ Assumptions of GLMs – Generalized Linear Models (GLM) aim at predicting a random variable y as a function fo $x \in \mathbb{R}^{n+1}$ and rely on the following 3 assumptions:

(1) $\boxed{y|x;\theta \sim \text{ExpFamily}(\eta)}$ (2) $\boxed{h_\theta(x) = E[y|x;\theta]}$ (3) $\boxed{\eta = \theta^Tx}$

Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

Support Vector Machines

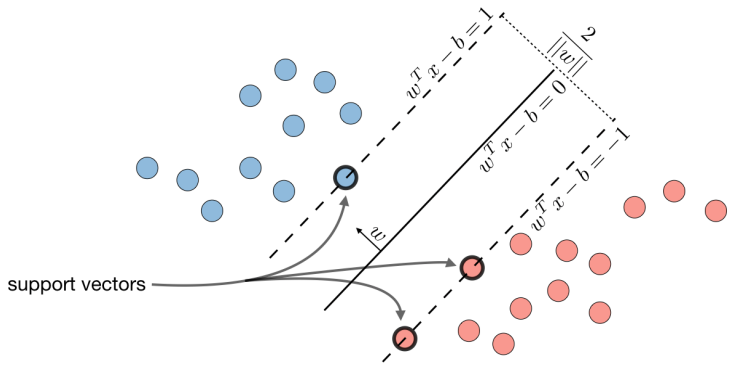
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

❒ Optimal margin classifier – The optimal margin classifier h is such that:

$$\boxed{h(x) = \text{sign}(w^Tx - b)}$$

where $(w,b) \in \mathbb{R}^n \times \mathbb{R}$ is the solution of the following optimization problem:

$$\boxed{\min \frac{1}{2}||w||^2} \quad \text{such that} \quad \boxed{y^{(i)}(w^Tx^{(i)} - b) \geqslant 1}$$



Remark: the line is defined as $\boxed{w^T x - b = 0}$.

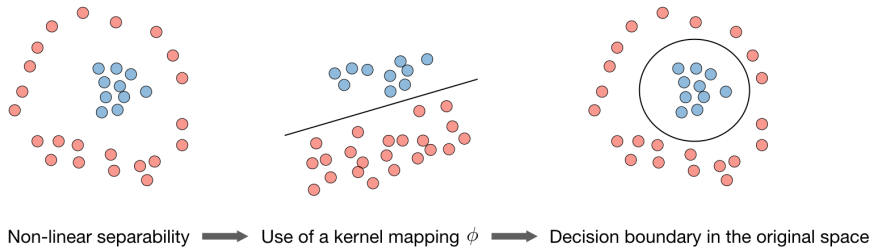
❒ **Hinge loss** – The hinge loss is used in the setting of SVMs and is defined as follows:

$$\boxed{L(z,y) = [1-yz]_+ = \max(0,1-yz)}$$

❒ **Kernel** – Given a feature mapping ϕ, we define the kernel K to be defined as:

$$\boxed{K(x,z) = \phi(x)^T \phi(z)}$$

In practice, the kernel K defined by $K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used.



Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated. Instead, only the values $K(x,z)$ are needed.

❒ **Lagrangian** – We define the Lagrangian $\mathcal{L}(w,b)$ as follows:

$$\boxed{\mathcal{L}(w,b) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)}$$

Remark: the coefficients β_i are called the Lagrange multipliers.

Generative Learning

A generative model first tries to learn how the data is generated by estimating $P(x|y)$, which we can then use to estimate $P(y|x)$ by using Bayes' rule.

Gaussian Discriminant Analysis

❒ **Setting** – The Gaussian Discriminant Analysis assumes that y and $x|y=0$ and $x|y=1$ are such that:

$$\boxed{y \sim \text{Bernoulli}(\phi)}$$

$$\boxed{x|y=0 \sim \mathcal{N}(\mu_0,\Sigma)} \quad \text{and} \quad \boxed{x|y=1 \sim \mathcal{N}(\mu_1,\Sigma)}$$

❒ **Estimation** – The following table sums up the estimates that we find when maximizing the likelihood:

$\widehat{\phi}$	$\widehat{\mu_j}$ $(j=0,1)$	$\widehat{\Sigma}$
$\frac{1}{m}\sum_{i=1}^{m} 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T$

Naive Bayes

❒ **Assumption** – The Naive Bayes model supposes that the features of each data point are all independent:

$$\boxed{P(x|y) = P(x_1,x_2,...|y) = P(x_1|y)P(x_2|y)... = \prod_{i=1}^{n} P(x_i|y)}$$

❒ **Solutions** – Maximizing the log-likelihood gives the following solutions, with $k \in \{0,1\}$, $l \in [\![1,L]\!]$

$$\boxed{P(y=k) = \frac{1}{m} \times \#\{j|y^{(j)}=k\}} \quad \text{and} \quad \boxed{P(x_i=l|y=k) = \frac{\#\{j|y^{(j)}=k \text{ and } x_i^{(j)}=l\}}{\#\{j|y^{(j)}=k\}}}$$

Remark: Naive Bayes is widely used for text classification and spam detection.

Tree-based and ensemble methods

These methods can be used for both regression and classification problems.

❒ **CART** – Classification and Regression Trees (CART), commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

❒ **Random forest** – It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

Remark: random forests are a type of ensemble methods.

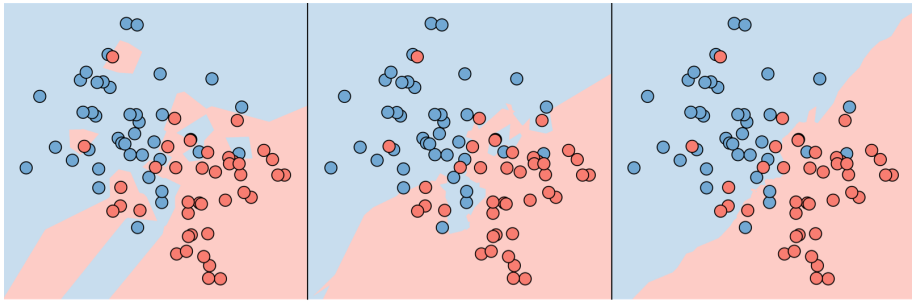
❒ **Boosting** – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

Adaptive boosting	Gradient boosting
- High weights are put on errors to improve at the next boosting step - Known as Adaboost	- Weak learners trained on remaining errors

Other non-parametric approaches

❒ **k-nearest neighbors** – The k-nearest neighbors algorithm, commonly known as k-NN, is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.

Remark: The higher the parameter k, the higher the bias, and the lower the parameter k, the higher the variance.

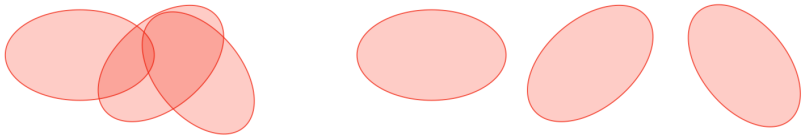


$k = 1$ $k = 3$ $k = 11$

Learning Theory

❒ **Union bound** – Let $A_1, ..., A_k$ be k events. We have:

$$\boxed{P(A_1 \cup ... \cup A_k) \leqslant P(A_1) + ... + P(A_k)}$$



$A_1 \cup A_2 \cup A_3$ A_1 A_2 A_3

❒ **Hoeffding inequality** – Let $Z_1, .., Z_m$ be m iid variables drawn from a Bernoulli distribution of parameter ϕ. Let $\widehat{\phi}$ be their sample mean and $\gamma > 0$ fixed. We have:

$$\boxed{P(|\phi - \widehat{\phi}| > \gamma) \leqslant 2\exp(-2\gamma^2 m)}$$

Remark: this inequality is also known as the Chernoff bound.

❒ **Training error** – For a given classifier h, we define the training error $\widehat{\epsilon}(h)$, also known as the empirical risk or empirical error, to be as follows:

$$\boxed{\widehat{\epsilon}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{\{h(x^{(i)}) \neq y^{(i)}\}}}$$

❒ **Probably Approximately Correct (PAC)** – PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:

- the training and testing sets follow the same distribution
- the training examples are drawn independently

❒ **Shattering** – Given a set $S = \{x^{(1)},...,x^{(d)}\}$, and a set of classifiers $\mathcal{H}$, we say that $\mathcal{H}$ shatters S if for any set of labels $\{y^{(1)}, ..., y^{(d)}\}$, we have:

$$\boxed{\exists h \in \mathcal{H}, \quad \forall i \in [\![1,d]\!], \quad h(x^{(i)}) = y^{(i)}}$$

❒ **Upper bound theorem** – Let $\mathcal{H}$ be a finite hypothesis class such that $|\mathcal{H}| = k$ and let δ and the sample size m be fixed. Then, with probability of at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + 2\sqrt{\frac{1}{2m}\log\left(\frac{2k}{\delta}\right)}}$$

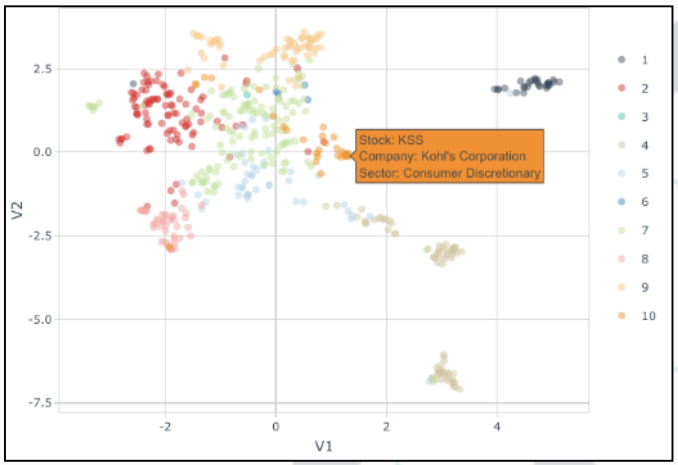
❒ **VC dimension** – The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class $\mathcal{H}$, noted $\text{VC}(\mathcal{H})$ is the size of the largest set that is shattered by $\mathcal{H}$.

Remark: the VC dimension of $\mathcal{H} = \{$set of linear classifiers in 2 dimensions$\}$ is 3.

❒ **Theorem (Vapnik)** – Let $\mathcal{H}$ be given, with $\text{VC}(\mathcal{H}) = d$ and m the number of training examples. With probability at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + O\left(\sqrt{\frac{d}{m}\log\left(\frac{m}{d}\right) + \frac{1}{m}\log\left(\frac{1}{\delta}\right)}\right)}$$

Segmentation & Clustering



Combining K-Means & UMAP to visualize clusters by Stock Price Movements

Summary:

- **Common Applications in Business**: Can be used for finding segments within Customers, Companies, etc.
- **Key Concept:** Transform data into a matrix enabling trends to be compared across units of measure (e.g. user-item matrix)
- **Gotchas:** Data must be normalized or standardized to enable comparison. This often requires calculation proportions of values by customer, company, etc to ensure the larger values do not dominate the trend mining operation.
- **How Many Components/Clusters?** Use a *Scree Plot* to determine the proportion of variance explained or total within sum of squares

Type	Popular Methods	Uses	Data Treatment
Clustering	K-Means Hierarchical Clustering	**Group Detection:** Methods use a measure of similarity (e.g. Euclidean distance) to detect groups within data set	Standardized or normalized
Dimensionality Reduction	PCA UMAP tSNE	**Reduce Width of Data:** Performing Machine Learning on wide data can drastically increase the time for algorithms to converge. Dimensionality reduction can be applied as a preprocessing step to reduce the width (number of columns) of the data but still maintain a high proportion of the overall structure. **Visualization:** Visualizing the first two components as X and Y often can enable cluster visualization. Combining with clustering techniques can provide a useful method of visualization.	Standardized or normalized

R Cheat Sheet

K-Means

```
set.seed(0)

kmeans_obj <- kmeans(X, centers = 4)
```

UMAP

```
library(umap)

umap_obj <- umap(X)
```

Python Cheat Sheet

K-Means

```
from sklearn.cluster import KMeans

kmeans = KMeans(
    n_clusters=4,
    random_state=0).fit(X)
```

UMAP

```
import umap

reducer = umap.UMAP()

embedding = reducer.fit_transform(X)
```

Resources

- Business Analysis With R Course (DS4B 101-R) - Modeling - Week 6
- Business Science Problem Framework
- Ultimate R Cheat Sheet I Ultimate Python Cheat Sheet

Data Science Courses for Business

Business Science University
university.business-science.io

version: 1.0

Segmentation & Clustering

How to apply K-Means & UMAP step-by-step

Clustering Workflow

Collect Data

```
> sp_500_prices_tbl
# A tibble: 1,225,765 x 8
   symbol date       open  high   low close   volume adjusted
   <chr>  <date>    <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
 1 MSFT   2009-01-02  19.5  20.4  19.4  20.3 50084000     15.9
 2 MSFT   2009-01-05  20.2  20.7  20.1  20.5 61475200     16.0
 3 MSFT   2009-01-06  20.8  21    20.6  20.8 58083400     16.2
 4 MSFT   2009-01-07  20.2  20.3  19.5  19.5 72709900     15.2
 5 MSFT   2009-01-08  19.6  20.2  19.5  20.1 70255400     15.7
 6 MSFT   2009-01-09  20.2  20.3  19.4  19.5 49815300     15.2
 7 MSFT   2009-01-12  19.7  19.8  19.3  19.5 52163500     15.2
 8 MSFT   2009-01-13  19.5  20.0  19.5  19.8 65843500     15.5
 9 MSFT   2009-01-14  19.5  19.7  19.0  19.1 80257500     14.9
10 MSFT   2009-01-15  19.1  19.3  18.5  19.2 96169800     15.0
# … with 1,225,755 more rows
```

Standardize / Normalize

```
> sp_500_daily_returns_tbl
# A tibble: 141,340 x 3
   symbol date       pct_return
   <chr>  <date>          <dbl>
 1 MSFT   2018-01-03   0.00465
 2 MSFT   2018-01-04   0.00880
 3 MSFT   2018-01-05   0.0124
 4 MSFT   2018-01-08   0.00102
 5 MSFT   2018-01-09  -0.000680
 6 MSFT   2018-01-10  -0.00453
 7 MSFT   2018-01-11   0.00296
 8 MSFT   2018-01-12   0.0173
 9 MSFT   2018-01-16  -0.0140
10 MSFT   2018-01-17   0.0203
# … with 141,330 more rows
```

Spread to User-Item Format

```
> stock_date_matrix_tbl
# A tibble: 502 x 283
   symbol `2018-01-03` `2018-01-04` `2018-01-05` `2
   <chr>         <dbl>        <dbl>        <dbl>
 1 A            0.0254     -0.00750       0.0160
 2 AAL         -0.0123      0.00630    -0.000380
 3 AAP          0.00905     0.0369      0.0106
 4 AAPL        -0.000174    0.00465     0.0114
 5 ABBV         0.0156     -0.00570     0.0174
 6 ABC          0.00372    -0.00222     0.0121
 7 ABMD         0.0173      0.0175      0.0154
 8 ABT          0.00221    -0.00170     0.00289
 9 ACN          0.00462     0.0118      0.00825
10 ADBE         0.0188      0.0120      0.0116
# … with 492 more rows, and 272 more variables: `2
```

K-Means

Obtain cluster assignments

UMAP

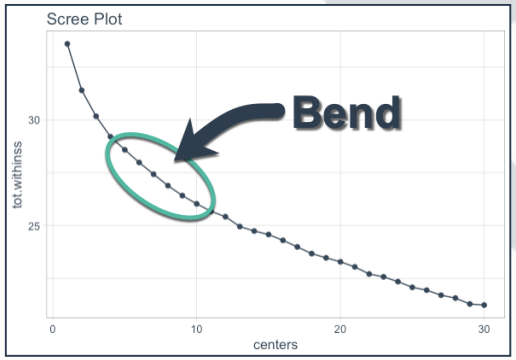
Make 2D Projection

K-Means: Scree Plot

Used to pick a value for K clusters for K-means algorithm.

Iteratively calculate "tot.withinss" for values of K.

Look for a bend.

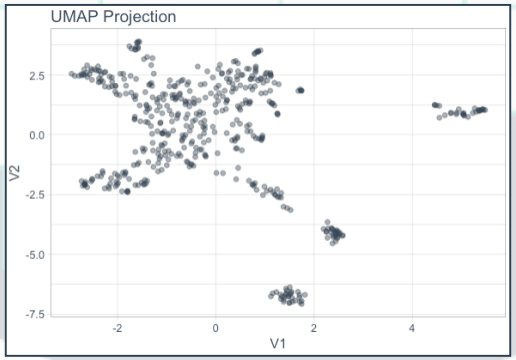


UMAP: 2D Projection

Fast dimensionality reduction algorithm that can be used for visualization.

Better than PCA - PCA is linear, UMAP is nonlinear

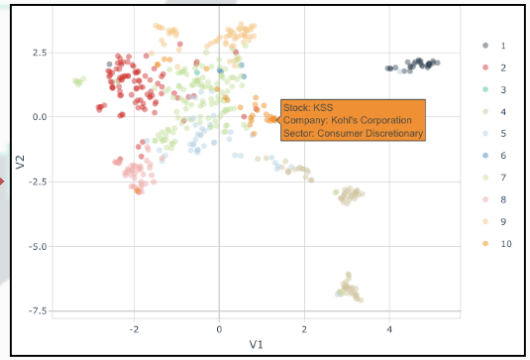
Better than tSNE - tSNE is slow



Combine

Plot the K-Means cluster assignments with the UMAP 2D Projection to obtain a visual.

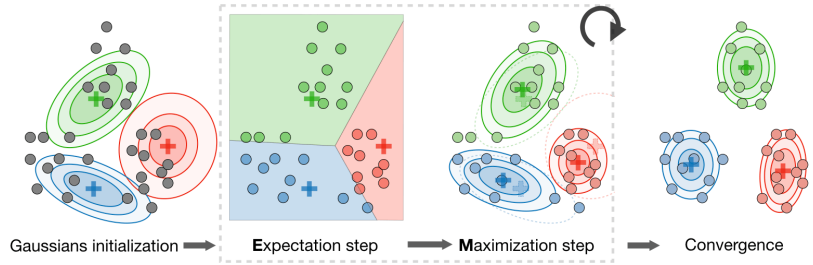
Add interactivity to enable exploration.



VIP Cheatsheet: Unsupervised Learning

Afshine Amidi and Shervine Amidi

September 9, 2018



Introduction to Unsupervised Learning

❒ **Motivation** – The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x^{(1)},...,x^{(m)}\}$.

❒ **Jensen's inequality** – Let f be a convex function and X a random variable. We have the following inequality:

$$\boxed{E[f(X)] \geqslant f(E[X])}$$

Expectation-Maximization

❒ **Latent variables** – Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted z. Here are the most common settings where there are latent variables:

Setting	Latent variable z	$x\|z$	Comments
Mixture of k Gaussians	Multinomial(ϕ)	$\mathcal{N}(\mu_j,\Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0,I)$	$\mathcal{N}(\mu+\Lambda z,\psi)$	$\mu_j \in \mathbb{R}^n$

❒ **Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $Q_i(z^{(i)})$ that each data point $x^{(i)}$ came from a particular cluster $z^{(i)}$ as follows:

$$\boxed{Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)};\theta)}$$

- M-step: Use the posterior probabilities $Q_i(z^{(i)})$ as cluster specific weights on data points $x^{(i)}$ to separately re-estimate each cluster model as follows:

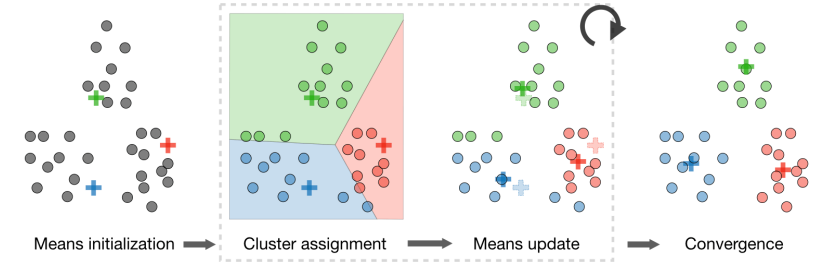
$$\boxed{\theta_i = \underset{\theta}{\text{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right) dz^{(i)}}$$

k-means clustering

We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j.

❒ **Algorithm** – After randomly initializing the cluster centroids $\mu_1,\mu_2,...,\mu_k \in \mathbb{R}^n$, the k-means algorithm repeats the following step until convergence:

$$\boxed{c^{(i)} = \underset{j}{\arg\min}||x^{(i)} - \mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}}$$



❒ **Distortion function** – In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$\boxed{J(c,\mu) = \sum_{i=1}^m ||x^{(i)} - \mu_{c^{(i)}}||^2}$$

Hierarchical clustering

❒ **Algorithm** – It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

❒ **Types** – There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

❒ **Silhouette coefficient** – By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$\boxed{s = \frac{b-a}{\max(a,b)}}$$

❒ **Calinski-Harabaz index** – By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matrices respectively defined as

$$B_k = \sum_{j=1}^{k} n_{c^{(i)}}(\mu_{c^{(i)}} - \mu)(\mu_{c^{(i)}} - \mu)^T, \qquad W_k = \sum_{i=1}^{m}(x^{(i)} - \mu_{c^{(i)}})(x^{(i)} - \mu_{c^{(i)}})^T$$

the Calinski-Harabaz index $s(k)$ indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$\boxed{s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N-k}{k-1}}$$

Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n\times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$\boxed{Az = \lambda z}$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n\times n}$. By noting $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, we have:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

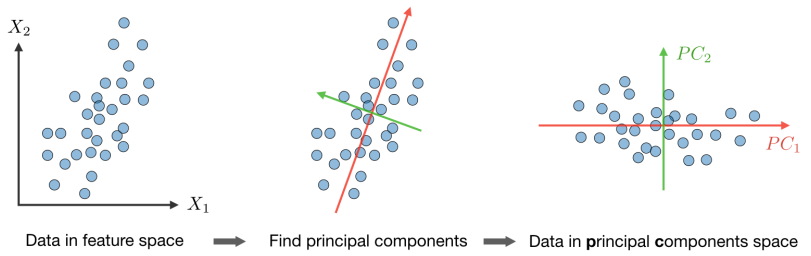
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

❒ **Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:

- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

- Step 2: Compute $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} \in \mathbb{R}^{n\times n}$, which is symmetric with real eigenvalues.
- Step 3: Compute $u_1, ..., u_k \in \mathbb{R}^n$ the k orthogonal principal eigenvectors of Σ, i.e. the orthogonal eigenvectors of the k largest eigenvalues.
- Step 4: Project the data on $\text{span}_{\mathbb{R}}(u_1,...,u_k)$. This procedure maximizes the variance among all k-dimensional spaces.



Independent component analysis

It is a technique meant to find the underlying generating sources.

❒ **Assumptions** – We assume that our data x has been generated by the n-dimensional source vector $s = (s_1,...,s_n)$, where s_i are independent random variables, via a mixing and non-singular matrix A as follows:

$$\boxed{x = As}$$

The goal is to find the unmixing matrix $W = A^{-1}$ by an update rule.

❒ **Bell and Sejnowski ICA algorithm** – This algorithm finds the unmixing matrix W by following the steps below:

- Write the probability of $x = As = W^{-1}s$ as:

$$p(x) = \prod_{i=1}^{n} p_s(w_i^T x) \cdot |W|$$

- Write the log likelihood given our training data $\{x^{(i)}, i \in [\![1,m]\!]\}$ and by noting g the sigmoid function as:

$$l(W) = \sum_{i=1}^{m}\left(\sum_{j=1}^{n}\log\left(g'(w_j^T x^{(i)})\right) + \log|W|\right)$$

Therefore, the stochastic gradient ascent learning rule is such that for each training example $x^{(i)}$, we update W as follows:

$$\boxed{W \longleftarrow W + \alpha \left(\begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)\,T} + (W^T)^{-1} \right)}$$

VIP Cheatsheet: Machine Learning Tips

Afshine Amidi and Shervine Amidi

September 9, 2018

Metrics

Given a set of data points $\{x^{(1)}, ..., x^{(m)}\}$, where each $x^{(i)}$ has n features, associated to a set of outcomes $\{y^{(1)}, ..., y^{(m)}\}$, we want to assess a given classifier that learns how to predict y from x.

Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

❒ **Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:

		Predicted class +	**Predicted** class –
Actual class	+	**TP** True Positives	**FN** False Negatives Type II error
	–	**FP** False Positives Type I error	**TN** True Negatives

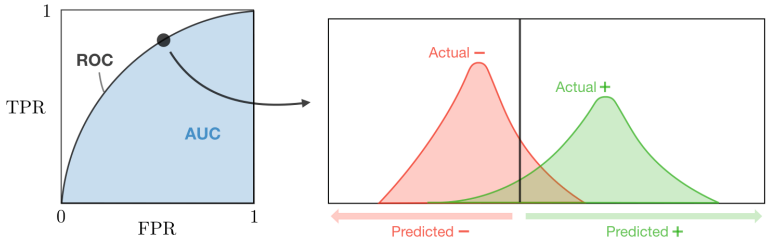
❒ **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$	Overall performance of model
Precision	$\frac{\text{TP}}{\text{TP}+\text{FP}}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Coverage of actual positive sample
Specificity	$\frac{\text{TN}}{\text{TN}+\text{FP}}$	Coverage of actual negative sample
F1 score	$\frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$	Hybrid metric useful for unbalanced classes

❒ **ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Recall, sensitivity
False Positive Rate FPR	$\frac{\text{FP}}{\text{TN}+\text{FP}}$	1-specificity

❒ **AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Regression

❒ **Basic metrics** – Given a regression model f, the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$\text{SS}_{\text{tot}} = \sum_{i=1}^{m}(y_i - \overline{y})^2$	$\text{SS}_{\text{reg}} = \sum_{i=1}^{m}(f(x_i) - \overline{y})^2$	$\text{SS}_{\text{res}} = \sum_{i=1}^{m}(y_i - f(x_i))^2$

❒ **Coefficient of determination** – The coefficient of determination, often noted R^2 or r^2, provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

❒ **Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{\text{SS}_{\text{res}} + 2(n+1)\widehat{\sigma^2}}{m}$	$2\left[(n+2) - \log(L)\right]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

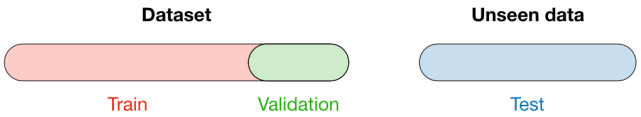
where L is the likelihood and $\widehat{\sigma}^2$ is an estimate of the variance associated with each response.

Model selection

❒ **Vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

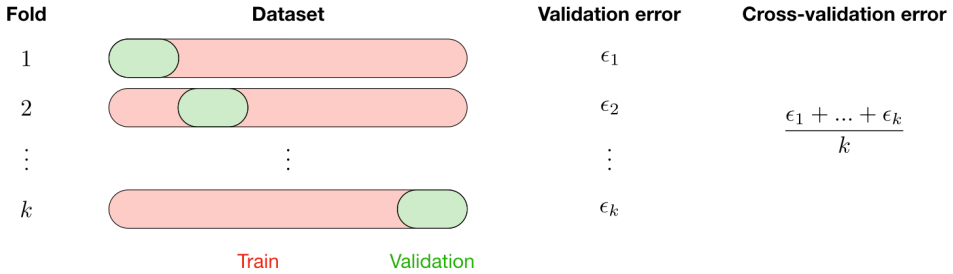
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



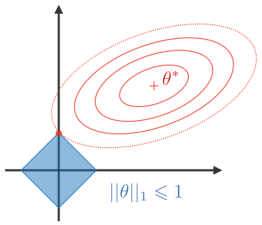
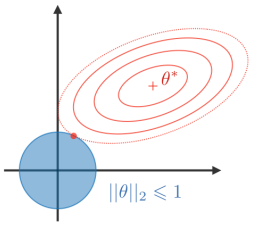
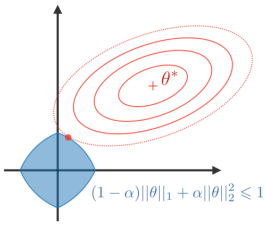
❒ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
- Training on $k-1$ folds and assessment on the remaining one - Generally $k=5$ or 10	- Training on $n-p$ observations and assessment on the p remaining ones - Case $p=1$ is called leave-one-out

The most commonly used method is called k-fold cross-validation and splits the training data into k folds to validate the model on one fold while training the model on the $k-1$ other folds, all of this k times. The error is then averaged over the k folds and is named cross-validation error.



❒ **Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$...+\lambda\|\|\theta\|\|_1$ $\lambda\in\mathbb{R}$	$...+\lambda\|\|\theta\|\|_2^2$ $\lambda\in\mathbb{R}$	$...+\lambda\left[(1-\alpha)\|\|\theta\|\|_1+\alpha\|\|\theta\|\|_2^2\right]$ $\lambda\in\mathbb{R}$, $\alpha\in[0,1]$

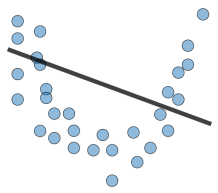
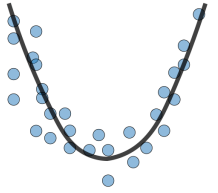
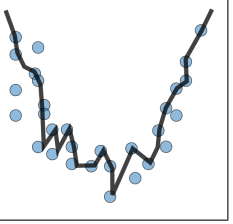
❒ **Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

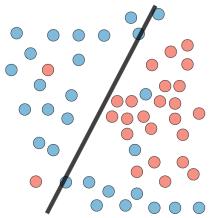
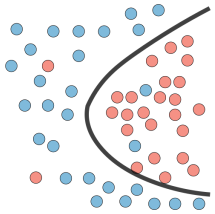
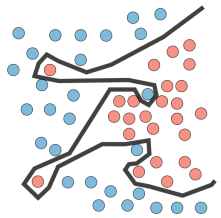
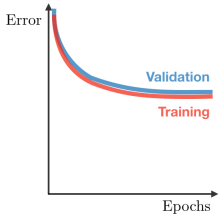
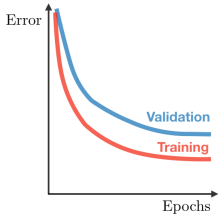
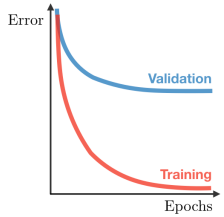
Diagnostics

❒ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

❒ **Variance** – The variance of a model is the variability of the model prediction for given data points.

❒ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

❒ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❒ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

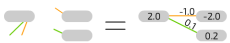
```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

An informative chart to build

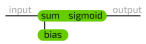
Neural Network Cells

Fixed Weight
(fixed at 1)

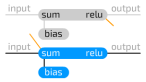
Weights



Recurrent Weights
(grouped by colour)

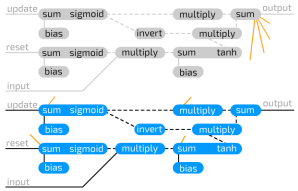


Feed Forward Cell
(basic cell)



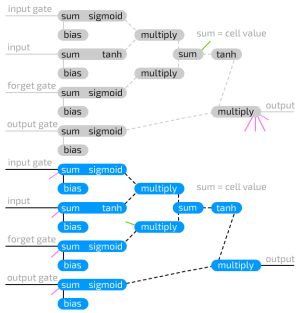
Recurrent Cell
(previous iteration)

Recurrent Cell



GRU Cell
(previous iteration)

GRU Cell

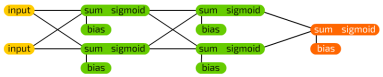


LSTM Cell
(previous iteration)

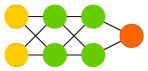
LSTM Cell

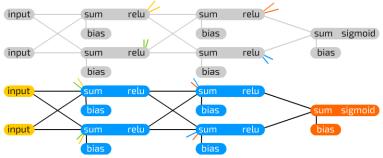
An informative chart to build

Neural Network Graphs

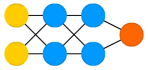


Deep Feed Forward Example

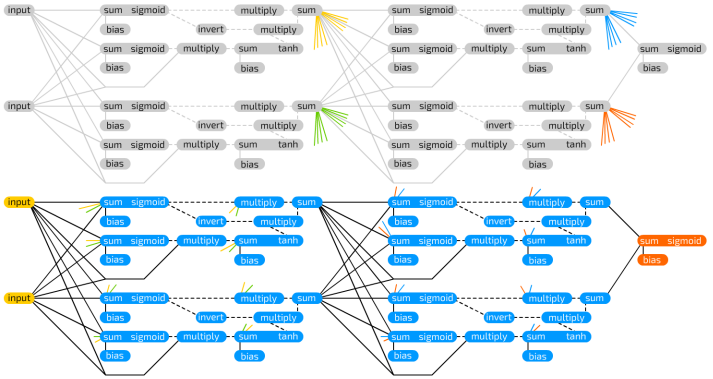




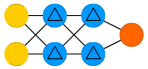
Deep Recurrent Example
(previous iteration)



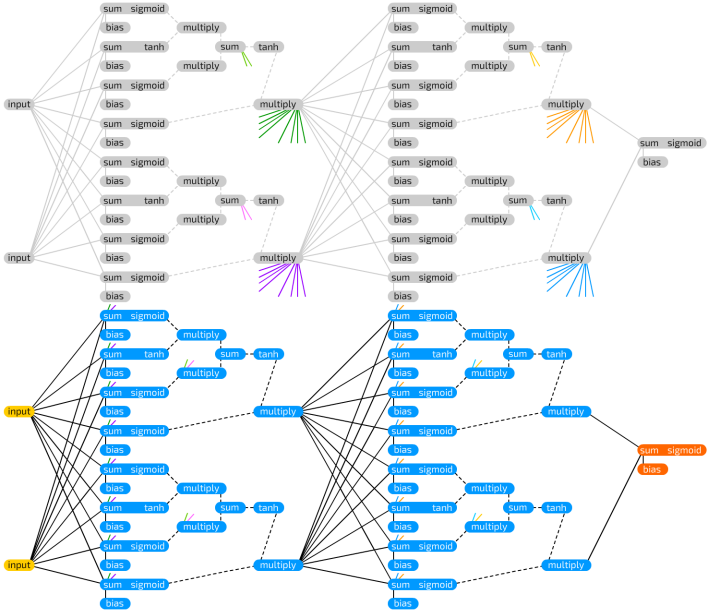
Deep Recurrent Example



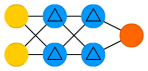
Deep GRU Example
(previous iteration)



Deep GRU Example



Deep LSTM Example
(previous iteration)



Deep LSTM Example

VIP Cheatsheet: Deep Learning

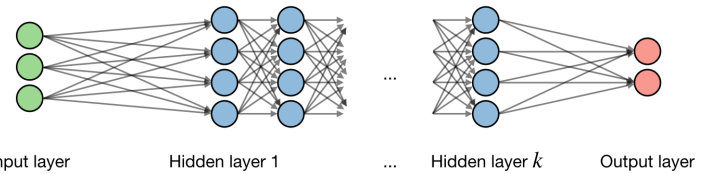
Afshine Amidi and Shervine Amidi

September 15, 2018

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

❒ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

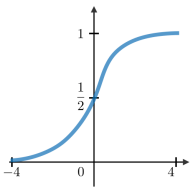
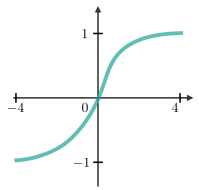
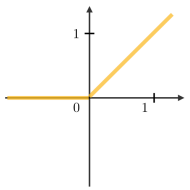
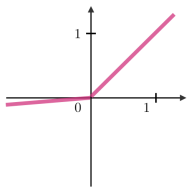


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

where we note w, b, z the weight, bias and output respectively.

❒ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

❒ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

❒ **Learning rate** – The learning rate, often noted η, indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

As a result, the weight is updated as follows:

$$\boxed{w \longleftarrow w - \eta \frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

❒ **Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1-p$.

Convolutional Neural Networks

❒ **Convolutional layer requirement** – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$\boxed{N = \frac{W - F + 2P}{S} + 1}$$

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Recurrent Neural Networks

❒ **Types of gates** – Here are the different types of gates that we encounter in a typical recurrent neural network:

Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

❒ **LSTM** – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

Reinforcement Learning and Control

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

❒ **Markov decision processes** – A Markov decision process (MDP) is a 5-tuple $(S,A,\{P_{sa}\},\gamma,R)$ where:

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $\{P_{sa}\}$ are the state transition probabilities for $s \in \mathcal{S}$ and $a \in \mathcal{A}$
- $\gamma \in [0,1[$ is the discount factor
- $R : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ or $R : \mathcal{S} \longrightarrow \mathbb{R}$ is the reward function that the algorithm wants to maximize

❒ **Policy** – A policy π is a function $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ that maps states to actions.

Remark: we say that we execute a given policy π if given a state s we take the action $a = \pi(s)$.

❒ **Value function** – For a given policy π and a given state s, we define the value function V^π as follows:

$$\boxed{V^\pi(s) = E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|s_0 = s,\pi\right]}$$

❒ **Bellman equation** – The optimal Bellman equations characterizes the value function V^{π^*} of the optimal policy π^*:

$$\boxed{V^{\pi^*}(s) = R(s) + \max_{a \in \mathcal{A}} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')}$$

Remark: we note that the optimal policy π^* for a given state s is such that:

$$\boxed{\pi^*(s) = \underset{a \in \mathcal{A}}{\text{argmax}} \sum_{s' \in \mathcal{S}} P_{sa}(s') V^*(s')}$$

❒ **Value iteration algorithm** – The value iteration algorithm is in two steps:

- We initialize the value:

$$\boxed{V_0(s) = 0}$$

- We iterate the value based on the values before:

$$\boxed{V_{i+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}} \gamma P_{sa}(s') V_i(s')\right]}$$

❒ **Maximum likelihood estimate** – The maximum likelihood estimates for the state transition probabilities are as follows:

$$\boxed{P_{sa}(s') = \frac{\#\text{times took action } a \text{ in state } s \text{ and got to } s'}{\#\text{times took action } a \text{ in state } s}}$$

❒ **Q-learning** – Q-learning is a model-free estimation of Q, which is done as follows:

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]}$$

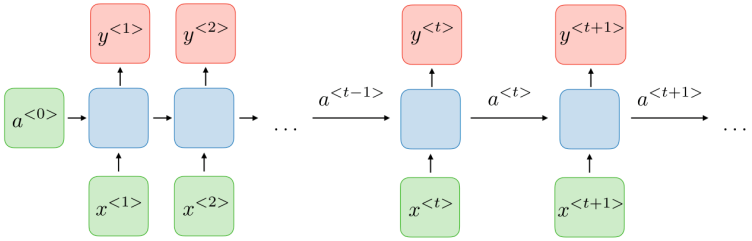
VIP Cheatsheet: Recurrent Neural Networks

Afshine Amidi and Shervine Amidi

November 26, 2018

Overview

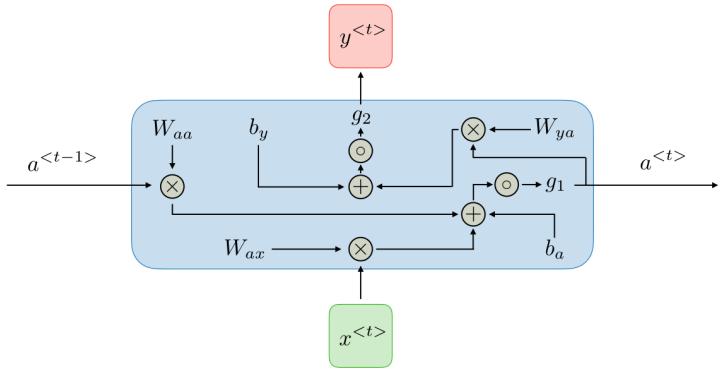
❒ Architecture of a traditional RNN – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

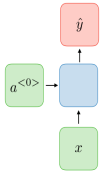
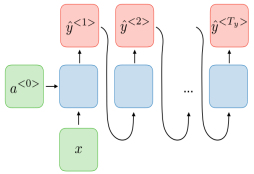
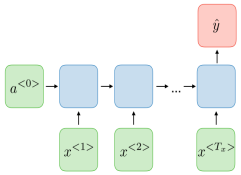
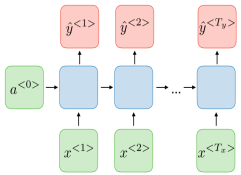
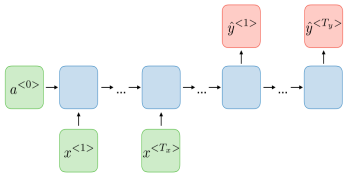
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
- Possibility of processing input of any length - Model size not increasing with size of input - Computation takes into account historical information - Weights are shared across time	- Computation being slow - Difficulty of accessing information from a long time ago - Cannot consider any future input for the current state

❒ Applications of RNNs – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

❒ Loss function – In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time

steps is defined based on the loss at every time step as follows:

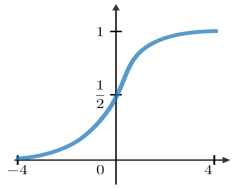
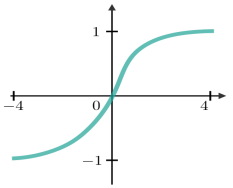
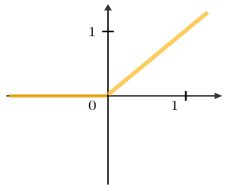
$$\mathcal{L}(\widehat{y},y)=\sum_{t=1}^{T_y}\mathcal{L}(\widehat{y}^{<t>},y^{<t>})$$

❒ **Backpropagation through time** – Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\frac{\partial\mathcal{L}^{(T)}}{\partial W}=\sum_{t=1}^{T}\left.\frac{\partial\mathcal{L}^{(T)}}{\partial W}\right|_{(t)}$$

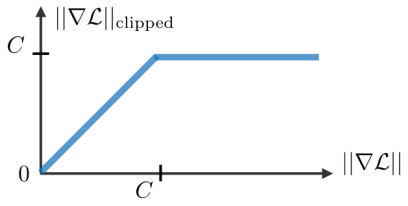
Handling long term dependencies

❒ **Commonly used activation functions** – The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z)=\frac{1}{1+e^{-z}}$	$g(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}}$	$g(z)=\max(0,z)$
		

❒ **Vanishing/exploding gradient** – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

❒ **Gradient clipping** – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



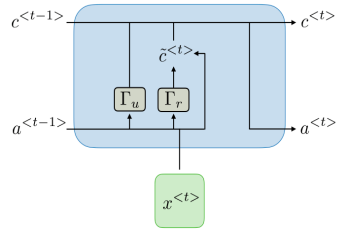
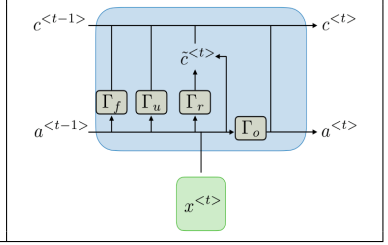
❒ **Types of gates** – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

$$\Gamma=\sigma(Wx^{<t>}+Ua^{<t-1>}+b)$$

where W,U,b are coefficients specific to the gate and σ is the sigmoid function. The main ones are summed up in the table below:

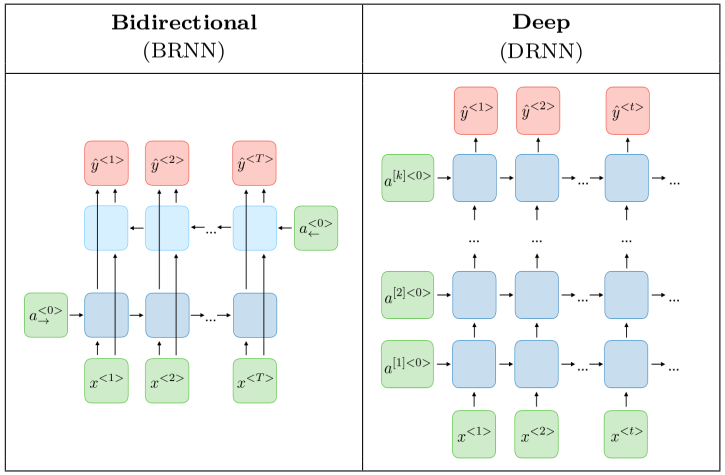
Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

❒ **GRU/LSTM** – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r\star a^{<t-1>},x^{<t>}]+b_c)$	$\tanh(W_c[\Gamma_r\star a^{<t-1>},x^{<t>}]+b_c)$
$c^{<t>}$	$\Gamma_u\star\tilde{c}^{<t>}+(1-\Gamma_u)\star c^{<t-1>}$	$\Gamma_u\star\tilde{c}^{<t>}+\Gamma_f\star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o\star c^{<t>}$
Dependencies		

Remark: the sign $\star$ denotes the element-wise multiplication between two vectors.

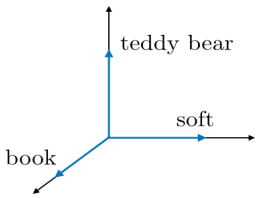
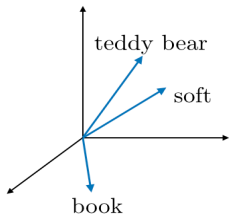
❒ **Variants of RNNs** – The table below sums up the other commonly used RNN architectures:



Learning word representation

In this section, we note V the vocabulary and $|V|$ its size.

❒ **Representation techniques** – The two main ways of representing words are summed up in the table below:

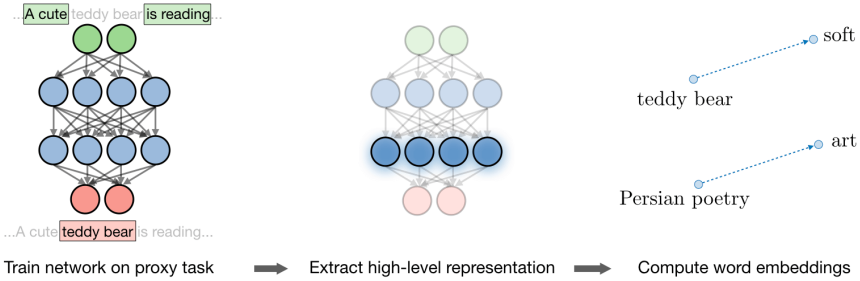
1-hot representation	Word embedding
	
- Noted o_w - Naive approach, no similarity information	- Noted e_w - Takes into account words similarity

❒ **Embedding matrix** – For a given word w, the embedding matrix E is a matrix that maps its 1-hot representation o_w to its embedding e_w as follows:

$$\boxed{e_w = Eo_w}$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

❒ **Word2vec** – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



❒ **Skip-gram** – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word t happening with a context word c. By noting θ_t a parameter associated with t, the probability $P(t|c)$ is given by:

$$\boxed{P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}}$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

❒ **Negative sampling** – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of k negative examples and 1 positive example. Given a context word c and a target word t, the prediction is expressed by:

$$\boxed{P(y = 1|c,t) = \sigma(\theta_t^T e_c)}$$

Remark: this method is less computationally expensive than the skip-gram model.

❒ **GloVe** – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurence matrix X where each $X_{i,j}$ denotes the number of times that a target i occurred with a context j. Its cost function J is as follows:

$$\boxed{J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2}$$

here f is a weighting function such that $X_{i,j} = 0 \Longrightarrow f(X_{i,j}) = 0$.

Given the symmetry that e and θ play in this model, the final word embedding $e_w^{(\text{final})}$ is given by:

$$\boxed{e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}}$$

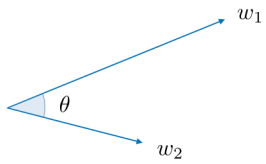
Remark: the individual components of the learned word embeddings are not necessarily interpretable.

Comparing words

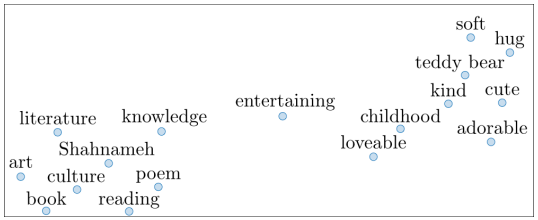
❒ **Cosine similarity** – The cosine similarity between words w_1 and w_2 is expressed as follows:

$$\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)$$

Remark: θ is the angle between words w_1 and w_2.



❒ **t-SNE** – t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



Language model

❒ **Overview** – A language model aims at estimating the probability of a sentence $P(y)$.

❒ **n-gram model** – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

❒ **Perplexity** – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words T. The perplexity is such that the lower, the better and is defined as follows:

$$\text{PP} = \prod_{t=1}^{T}\left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \widehat{y}_j^{(t)}}\right)^{\frac{1}{T}}$$

Remark: PP is commonly used in t-SNE.

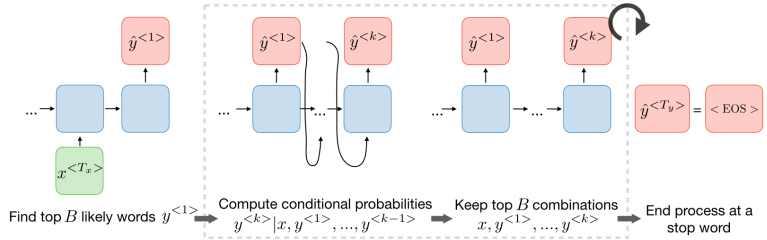
Machine translation

❒ **Overview** – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence y such that:

$$y = \underset{y^{<1>},\ldots,y^{<T_y>}}{\arg\max}\ P(y^{<1>},\ldots,y^{<T_y>}|x)$$

❒ **Beam search** – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x.

- Step 1: Find top B likely words $y^{<1>}$
- Step 2: Compute conditional probabilities $y^{<k>}|x,y^{<1>},\ldots,y^{<k-1>}$
- Step 3: Keep top B combinations $x,y^{<1>},\ldots,y^{<k>}$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

❒ **Beam width** – The beam width B is a parameter for beam search. Large values of B yield to better result but with slower performance and increased memory. Small values of B lead to worse results but is less computationally intensive. A standard value for B is around 10.

❒ **Length normalization** – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective} = \frac{1}{T_y^{\alpha}}\sum_{t=1}^{T_y}\log\left[p(y^{<t>}|x,y^{<1>},\ldots,y^{<t-1>})\right]$$

Remark: the parameter α can be seen as a softener, and its value is usually between 0.5 and 1.

❒ **Error analysis** – When obtaining a predicted translation $\widehat{y}$ that is bad, one can wonder why we did not get a good translation y^* by performing the following error analysis:

Case	$P(y^*\|x) > P(\widehat{y}\|x)$	$P(y^*\|x) \leqslant P(\widehat{y}\|x)$
Root cause	Beam search faulty	RNN faulty
Remedies	Increase beam width	- Try different architecture - Regularize - Get more data

❒ **Bleu score** – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp\left(\frac{1}{n}\sum_{k=1}^{n} p_k\right)$$

where p_n is the bleu score on n-gram only defined as follows:

$$p_n = \frac{\sum_{\text{n-gram}\in\widehat{y}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram}\in\widehat{y}} \text{count}(\text{n-gram})}$$

Remark: a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.

Attention

❒ **Attention model** – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting $\alpha^{<t,t'>}$ the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ and $c^{<t>}$ the context at time t, we have:

$$\boxed{c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>}} \quad \text{with} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

Remark: the attention scores are commonly used in image captioning and machine translation.

A cute teddy bear is reading Persian literature

A cute teddy bear is reading Persian literature

❒ **Attention weight** – The amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ is given by $\alpha^{<t,t'>}$ computed as follows:

$$\boxed{\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}}$$

Remark: computation complexity is quadratic with respect to T_x.

⋆ ⋆ ⋆

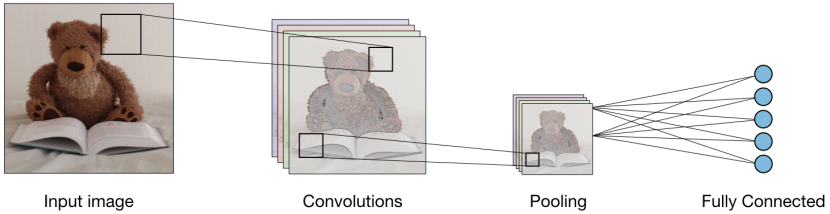
VIP Cheatsheet: Convolutional Neural Networks

Afshine Amidi and Shervine Amidi

November 26, 2018

Overview

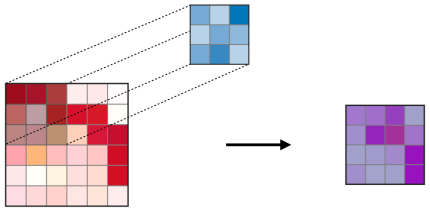
❒ **Architecture of a traditional CNN** – Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters that are described in the next sections.

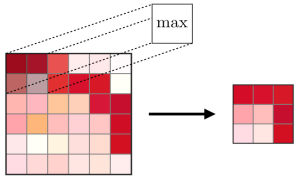
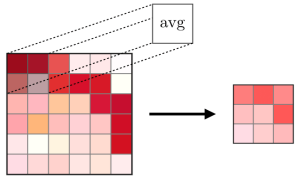
Types of layer

❒ **Convolutional layer (CONV)** – The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called *feature map* or *activation map*.

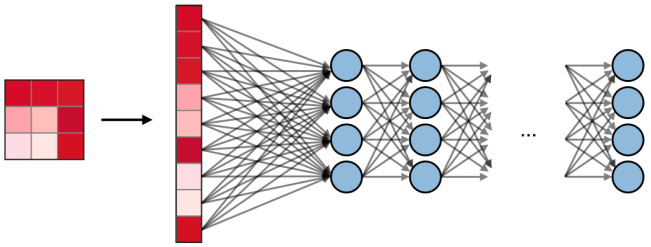


Remark: the convolution step can be generalized to the 1D and 3D cases as well.

❒ **Pooling (POOL)** – The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

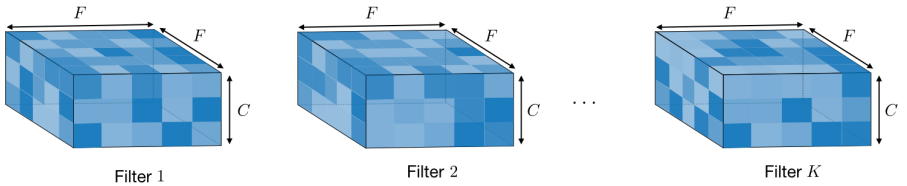
❒ **Fully Connected (FC)** – The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



Filter hyperparameters

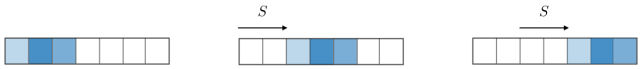
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

❒ **Dimensions of a filter** – A filter of size $F \times F$ applied to an input containing C channels is a $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called activation map) of size $O \times O \times 1$.

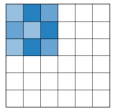
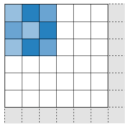
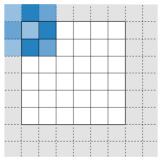


Remark: the application of K filters of size $F \times F$ results in an output feature map of size $O \times O \times K$.

❒ **Stride** – For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



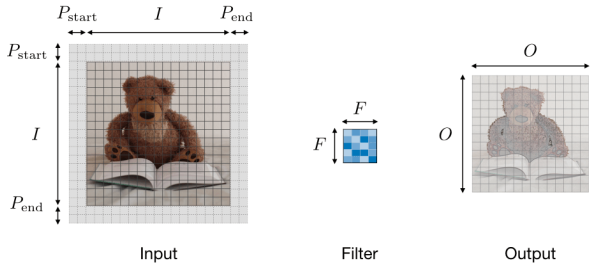
❒ **Zero-padding** – Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in [\![0, F-1]\!]$ $P_{\text{end}} = F - 1$
Illustration			
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ - Output size is mathematically convenient - Also called 'half' padding	- Maximum padding such that end convolutions are applied on the limits of the input - Filter 'sees' the input end-to-end

Tuning hyperparameters

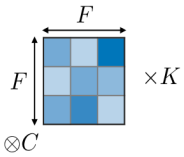
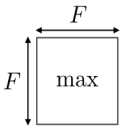
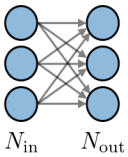
❒ **Parameter compatibility in convolution layer** – By noting I the length of the input volume size, F the length of the filter, P the amount of zero padding, S the stride, then the output size O of the feature map along that dimension is given by:

$$\boxed{O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1}$$



Remark: often times, $P_{start} = P_{end} \triangleq P$, in which case we can replace $P_{start} + P_{end}$ by $2P$ in the formula above.

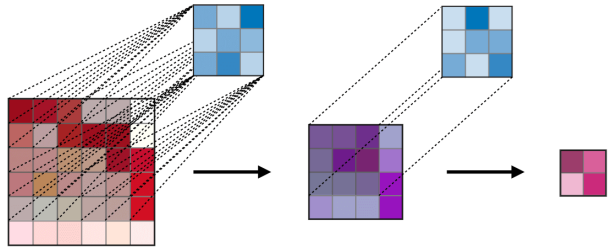
❒ **Understanding the complexity of the model** – In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	- One bias parameter per filter - In most cases, $S < F$ - A common choice for K is $2C$	- Pooling operation done channel-wise - In most cases, $S = F$	- Input is flattened - One bias parameter per neuron - The number of FC neurons is free of structural constraints

❒ **Receptive field** – The receptive field at layer k is the area denoted $R_k \times R_k$ of the input that each pixel of the k-th activation map can 'see'. By calling F_j the filter size of layer j and S_i the stride value of layer i and with the convention $S_0 = 1$, the receptive field at layer k can be computed with the formula:

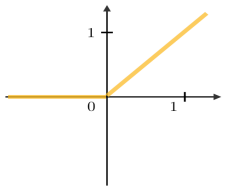
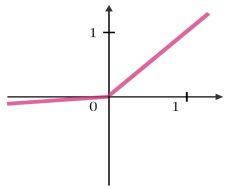
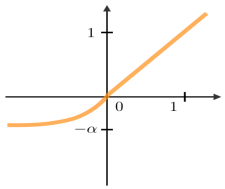
$$\boxed{R_k = 1 + \sum_{j=1}^{k} (F_j - 1) \prod_{i=0}^{j-1} S_i}$$

In the example below, we have $F_1 = F_2 = 3$ and $S_1 = S_2 = 1$, which gives $R_2 = 1 + 2 \cdot 1 + 2 \cdot 1 = 5$.



Commonly used activation functions

❒ **Rectified Linear Unit** – The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

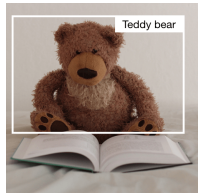
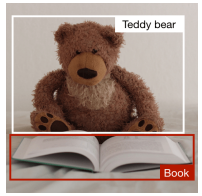
ReLU	Leaky ReLU	ELU
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

❒ **Softmax** – The softmax step can be seen as a generalized logistic function that takes as input a vector of scores $x \in \mathbb{R}^n$ and outputs a vector of output probability $p \in \mathbb{R}^n$ through a softmax function at the end of the architecture. It is defined as follows:

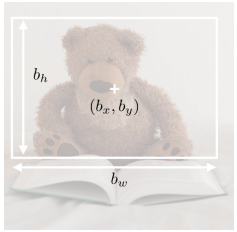
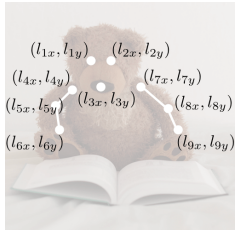
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Object detection

❒ **Types of models** – There are 3 main types of object recognition algorithms, for which the nature of what is predicted is different. They are described in the table below:

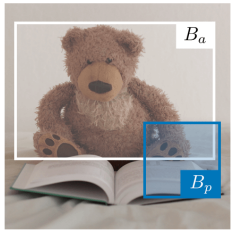
Image classification	Classification w. localization	Detection
		
- Classifies a picture - Predicts probability of object	- Detects object in a picture - Predicts probability of object and where it is located	- Detects up to several objects in a picture - Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

❒ **Detection** – In the context of object detection, different methods are used depending on whether we just want to locate the object or detect a more complex shape in the image. The two main ones are summed up in the table below:

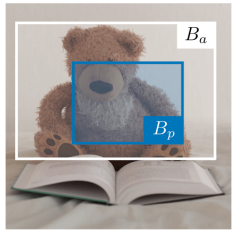
Bounding box detection	Landmark detection
Detects the part of the image where the object is located	- Detects a shape or characteristics of an object (e.g. eyes) - More granular
	
Box of center (b_x,b_y), height b_h and width b_w	Reference points (l_{1x},l_{1y}), ...,(l_{nx},l_{ny})

❒ **Intersection over Union** – Intersection over Union, also known as IoU, is a function that quantifies how correctly positioned a predicted bounding box B_p is over the actual bounding box B_a. It is defined as:

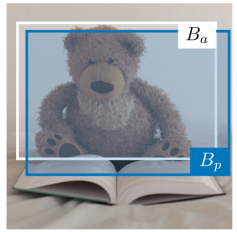
$$\text{IoU}(B_p,B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$



$\text{IoU}(B_p, B_a) = 0.1$



$\text{IoU}(B_p, B_a) = 0.5$



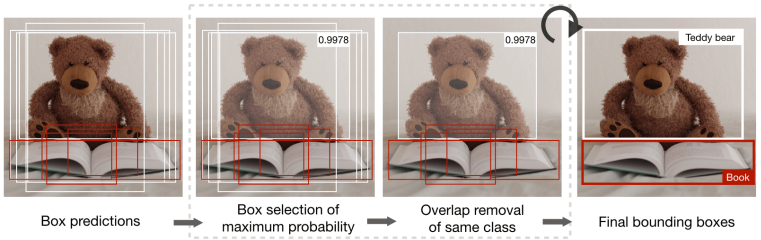
$\text{IoU}(B_p, B_a) = 0.9$

Remark: we always have $IoU \in [0,1]$. By convention, a predicted bounding box B_p is considered as being reasonably good if $IoU(B_p,B_a) \geqslant 0.5$.

❒ **Anchor boxes** – Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometrical form.

❒ **Non-max suppression** – The non-max suppression technique aims at removing duplicate overlapping bounding boxes of a same object by selecting the most representative ones. After having removed all boxes having a probability prediction lower than 0.6, the following steps are repeated while there are boxes remaining:

- Step 1: Pick the box with the largest prediction probability.
- Step 2: Discard any box having an IoU $\geqslant 0.5$ with the previous box.



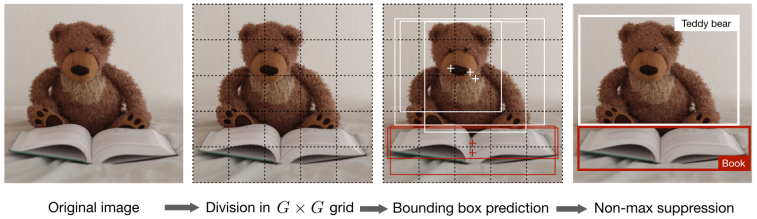
❒ **YOLO** – You Only Look Once (YOLO) is an object detection algorithm that performs the following steps:

- Step 1: Divide the input image into a $G \times G$ grid.
- Step 2: For each grid cell, run a CNN that predicts y of the following form:

$$y = \Big[\underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, ..., c_p}_{\text{repeated } k \text{ times}}, ...\Big]^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

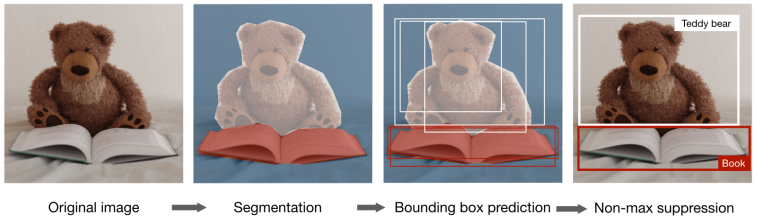
where p_c is the probability of detecting an object, b_x, b_y, b_h, b_w are the properties of the detected bouding box, $c_1, ..., c_p$ is a one-hot representation of which of the p classes were detected, and k is the number of anchor boxes.

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



Remark: when $p_c = 0$, then the network does not detect any object. In that case, the corresponding predictions $b_x, ..., c_p$ have to be ignored.

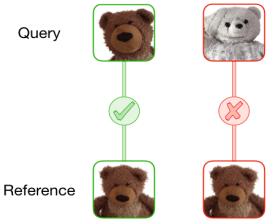
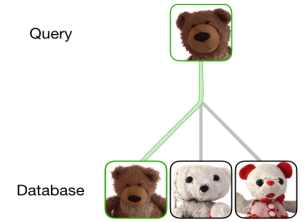
❒ **R-CNN** – Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.



Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

Face verification and recognition

❒ **Types of models** – Two main types of model are summed up in table below:

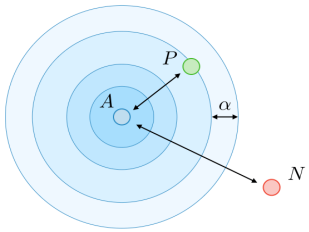
Face verification	Face recognition
- Is this the correct person? - One-to-one lookup	- Is this one of the K persons in the database? - One-to-many lookup
	

❒ **One Shot Learning** – One Shot Learning is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are. The similarity function applied to two images is often noted $d(\text{image 1}, \text{image 2})$.

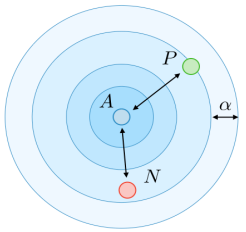
❒ **Siamese Network** – Siamese Networks aim at learning how to encode images to then quantify how different two images are. For a given input image $x^{(i)}$, the encoded output is often noted as $f(x^{(i)})$.

❒ **Triplet loss** – The triplet loss ℓ is a loss function computed on the embedding representation of a triplet of images A (anchor), P (positive) and N (negative). The anchor and the positive example belong to a same class, while the negative example to another one. By calling $\alpha \in \mathbb{R}^+$ the margin parameter, this loss is defined as follows:

$$\ell(A,P,N) = \max\left(d(A,P) - d(A,N) + \alpha, 0\right)$$



$\ell(A,P,N) = 0$



$\ell(A,P,N) > 0$

Neural style transfer

❒ **Motivation** – The goal of neural style transfer is to generate an image G based on a given content C and a given style S.

❒ **Activation** – In a given layer l, the activation is noted $a^{[l]}$ and is of dimensions $n_H \times n_w \times n_c$

❒ **Content cost function** – The content cost function $J_{\text{content}}(C,G)$ is used to determine how the generated image G differs from the original content image C. It is defined as follows:

$$\boxed{J_{\text{content}}(C,G) = \frac{1}{2}||a^{[l](C)} - a^{[l](G)}||^2}$$

❒ **Style matrix** – The style matrix $G^{[l]}$ of a given layer l is a Gram matrix where each of its elements $G^{[l]}_{kk'}$ quantifies how correlated the channels k and k' are. It is defined with respect to activations $a^{[l]}$ as follows:

$$\boxed{G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l]}_{ijk} a^{[l]}_{ijk'}}$$

Remark: the style matrix for the style image and the generated image are noted $G^{[l](S)}$ and $G^{[l](G)}$ respectively.

❒ **Style cost function** – The style cost function $J_{\text{style}}(S,G)$ is used to determine how the generated image G differs from the style S. It is defined as follows:

$$\boxed{J^{[l]}_{\text{style}}(S,G) = \frac{1}{(2n_H n_w n_c)^2}||G^{[l](S)} - G^{[l](G)}||^2_F = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2}$$

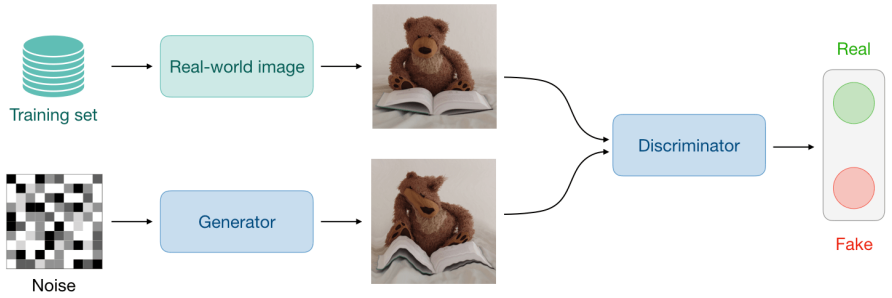
❒ **Overall cost function** – The overall cost function is defined as being a combination of the content and style cost functions, weighted by parameters α,β, as follows:

$$\boxed{J(G) = \alpha J_{\text{content}}(C,G) + \beta J_{\text{style}}(S,G)}$$

Remark: a higher value of α will make the model care more about the content while a higher value of β will make it care more about the style.

Architectures using computational tricks

❒ **Generative Adversarial Network** – Generative adversarial networks, also known as GANs, are composed of a generative and a discriminative model, where the generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true image.



Remark: use cases using variants of GANs include text to image, music generation and synthesis.

❒ **ResNet** – The Residual Network architecture (also called ResNet) uses residual blocks with a high number of layers meant to decrease the training error. The residual block has the following characterizing equation:

$$\boxed{a^{[l+2]} = g(a^{[l]} + z^{[l+2]})}$$

❒ **Inception Network** – This architecture uses inception modules and aims at giving a try at different convolutions in order to increase its performance. In particular, it uses the 1×1 convolution trick to lower the burden of computation.

⋆ ⋆ ⋆

VIP Cheatsheet: Tips and Tricks

Afshine Amidi and Shervine Amidi

November 26, 2018

Data processing

❒ **Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Training a neural network

❒ **Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

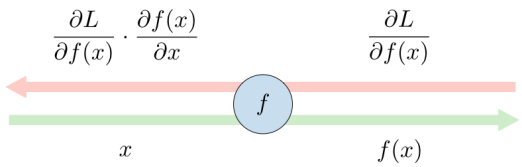
❒ **Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

❒ **Loss function** – In order to quantify how a given model performs, the loss function L is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z.

❒ **Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.

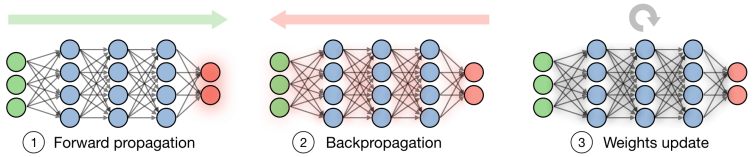


Using this method, each weight is updated with the rule:

$$\boxed{w \longleftarrow w - \alpha\frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

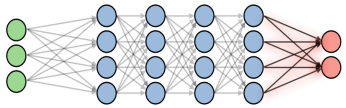
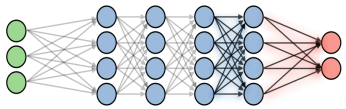
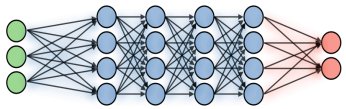
- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.



Parameter tuning

❒ **Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

❒ **Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

❒ **Learning rate** – The learning rate, often noted α or sometimes η, indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

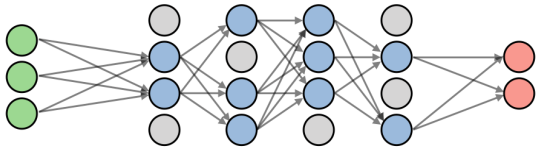
❒ **Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of w	Update of b
Momentum	- Dampens oscillations - Improvement to SGD - 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Speeds up learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Most popular method - 4 parameters to tune	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

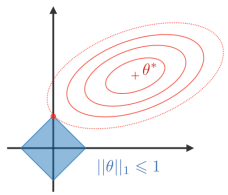
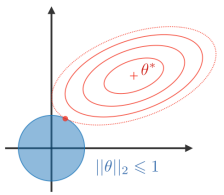
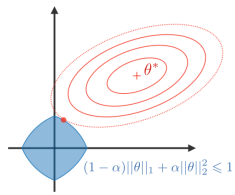
Regularization

❒ **Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

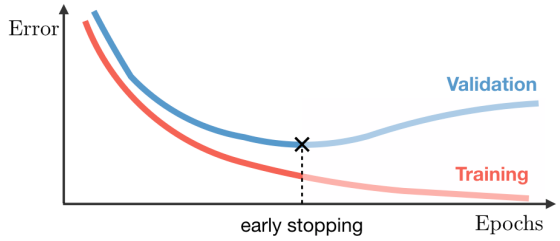


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1-p$.

❒ **Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda \|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda \|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda \left[(1-\alpha) \|\|\theta\|\|_1 + \alpha \|\|\theta\|\|_2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

❒ **Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



Good practices

❒ **Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

❒ **Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

	Numerical gradient	**Analytical gradient**
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	- Expensive; loss has to be computed two times per dimension - Used to verify correctness of analytical implementation -Trade-off in choosing h not too small (numerical instability) nor too large (poor gradient approx.)	- 'Exact' result - Direct computation - Used in the final implementation

⋆ ⋆ ⋆

DASK FOR PARALLEL COMPUTING CHEAT SHEET

See full Dask documentation at: **http://dask.pydata.org/**

These instructions use the conda environment manager. Get yours at **http://bit.ly/getconda**

DASK QUICK INSTALL	
Install Dask with conda	`conda install dask`
Install Dask with pip	`pip install dask[complete]`
DASK COLLECTIONS	**EASY TO USE BIG DATA COLLECTIONS**
DASK DATAFRAMES	PARALLEL PANDAS DATAFRAMES FOR LARGE DATA
Import	`import dask.dataframe as dd`
Read CSV data	`df = dd.read_csv('my-data.*.csv')`
Read Parquet data	`df = dd.read_parquet('my-data.parquet')`
Filter and manipulate data with Pandas syntax	`df['z'] = df.x + df.y`
Standard groupby aggregations, joins, etc.	`result = df.groupby(df.z).y.mean()`
Compute result as a Pandas dataframe	`out = result.compute()`
Or store to CSV, Parquet, or other formats	`result.to_parquet('my-output.parquet')`

EXAMPLE	`df = dd.read_csv('filenames.*.csv')`
	`df.groupby(df.timestamp.day)\`
	`  .value.mean().compute()`

DASK ARRAYS	PARALLEL NUMPY ARRAYS FOR LARGE DATA
Import	`import dask.array as da`
Create from any array-like object	`import h5py` `dataset = h5py.File('my-data.hdf5')['/group/dataset']`
Including HFD5, NetCDF, or other on-disk formats.	`x = da.from_array(dataset, chunks=(1000, 1000))`
Alternatively generate an array from a random distribution.	`da.random.uniform(shape=(1e4, 1e4), chunks=(100, 100))`
Perform operations with NumPy syntax	`y = x.dot(x.T - 1) - x.mean(axis=0)`
Compute result as a NumPy array	`result = y.compute()`
Or store to HDF5, NetCDF or other on-disk format	`out = f.create_dataset(...)` `x.store(out)`

EXAMPLE	`with h5py.File('my-data.hdf5') as f:`
	`  x = da.from_array(f['/path'], chunks=(1000, 1000))`
	`  x -= x.mean(axis=0)`
	`  out = f.create_dataset(...)`
	`  x.store(out)`

DASK BAGS	PARELLEL LISTS FOR UNSTRUCTURED DATA
Import	`import dask.bag as db`
Create Dask Bag from a sequence	`b = db.from_sequence(seq, npartitions)`
Or read from text formats	`b = db.read_text('my-data.*.json')`

Map and filter results	`import json`
	`records = b.map(json.loads)`
	`  .filter(lambda d: d["name"] == "Alice")`

Compute aggregations like mean, count, sum	`records.pluck('key-name').mean().compute()`
Or store results back to text formats	`records.to_textfiles('output.*.json')`

EXAMPLE	`db.read_text('s3://bucket/my-data.*.json')`
	`  .map(json.loads)`
	`  .filter(lambda d: d["name"] == "Alice")`
	`  .to_textfiles('s3://bucket/output.*.json')`

ADVANCED

Read from distributed file systems or cloud storage
Prepend prefixes like hdfs://, s3://, or gcs:// to paths

```
df = dd.read_parquet('s3://bucket/myfile.parquet')
b = db.read_text('hdfs:///path/to/my-data.*.json')
```

Persist lazy computations in memory

```
df = df.persist()
```

Compute multiple outputs at once

```
dask.compute(x.min(), x.max())
```

CUSTOM COMPUTATIONS — FOR CUSTOM CODE AND COMPLEX ALGORITHMS

DASK DELAYED — LAZY PARALLELISM FOR CUSTOM CODE

Import

```
import dask
```

Wrap custom functions with the @dask.delayed annotation

```
@dask.delayed
def load(filename):
    ...
```

Delayed functions operate lazily, producing a task graph rather than executing immediately

```
@dask.delayed
def process(data):
    ...
```

Passing delayed results to other delayed functions creates dependencies between tasks

```
load = dask.delayed(load)
process = dask.delayed(process)
```

Call functions in normal code

```
data = [load(fn) for fn in filenames]
results = [process(d) for d in data]
```

Compute results to execute in parallel

```
dask.compute(results)
```

CONCURRENT.FUTURES — ASYNCHRONOUS REAL-TIME PARALLELISM

Import

```
from dask.distributed import Client
```

Start local Dask Client

```
client = Client()
```

Submit individual task asynchronously

```
future = client.submit(func, *args, **kwargs)
```

Block and gather individual result

```
result = future.result()
```

Process results as they arrive

```
for future in as_completed(futures):
    ...
```

EXAMPLE

```
L = [client.submit(read, fn) for fn in filenames]
L = [client.submit(process, future) for future in L]
future = client.submit(sum, L)
result = future.result()
```

SET UP CLUSTER — HOW TO LAUNCH ON A CLUSTER

MANUALLY

Start scheduler on one machine

```
$ dask-scheduler
Scheduler started at SCHEDULER_ADDRESS:8786
```

Start workers on other machines
Provide address of the running scheduler

```
host1$ dask-worker SCHEDULER_ADDRESS:8786
host2$ dask-worker SCHEDULER_ADDRESS:8786
```

Start Client from Python process

```
from dask.distributed import Client
client = Client('SCHEDULER_ADDRESS:8786')
```

ON A SINGLE MACHINE

Call Client() with no arguments for easy setup on a single host

```
client = Client()
```

CLOUD DEPLOYMENT

See dask-kubernetes project for Google Cloud

```
pip install dask-kubernetes
```

See dask-ec2 project for Amazon EC2

```
pip install dask-ec2
```

MORE RESOURCES

User Documentation	dask.pydata.org
Technical documentation for distributed scheduler	distributed.readthedocs.org
Report a bug	github.com/dask/dask/issues

Python For Data Science *Cheat Sheet*

PySpark - SQL Basics

Learn Python for data science Interactively at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
```

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +--------+---+
  |    name|age|
  +--------+---+
  |    Mine| 28|
  |   Filip| 29|
  |Jonathan| 30|
  +--------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName |lastName|         phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
               "firstName",
               "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)
      .show()
>>> df.select(df['age'] > 24).show()
```

When

```
>>> df.select("firstName",
               F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]
                .collect()
```

Like

```
>>> df.select("firstName",
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName",
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th")) \
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \
                       .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \
      .show()
```

Show all entries in `firstName` column

Show all entries in `firstName`, `age` and `type`

Show all entries in `firstName` and `age`, add `1` to the entries of `age`

Show all entries where `age` >24

Show `firstName` and 0 or 1 depending on `age` >30

Show `firstName` if in the given options

Show `firstName`, and `lastName` is `TRUE` if `lastName` is like `Smith`

Show `firstName`, and `TRUE` if `lastName` starts with `Sm`

Show last names ending in `th`

Return substrings of `firstName`

Show `age`: values are `TRUE` if between 22 and 24

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age")\
      .count() \
      .show()
```

Group by `age`, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of `age`, only keep those records of which the values are >24

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
      .replace(10, 20) \
      .show()
```

Replace null values
Return new `df` omitting rows with null values
Return new `df` replacing one value with another

Repartitioning

```
>>> df.repartition(10)\
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

`df` with 10 partitions

`df` with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert `df` into an RDD
Convert `df` into a RDD of string
Return the contents of `df` as Pandas `DataFrame`

Write & Save to Files

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

Inspect Data

```
>>> df.dtypes
>>> df.show()
>>> df.head()
>>> df.first()
>>> df.take(2)
>>> df.schema
```

Return `df` column names and data types
Display the content of `df`
Return first `n` rows
Return first row
Return the first `n` rows
Return the schema of `df`

```
>>> df.describe().show()
>>> df.columns
>>> df.count()
>>> df.distinct().count()
>>> df.printSchema()
>>> df.explain()
```

Compute summary statistics
Return the columns of `df`
Count the number of rows in `df`
Count the number of distinct rows in `df`
Print the schema of `df`
Print the (logical and physical) plans

Python For Data Science *Cheat Sheet*

PySpark - RDD Basics

Learn Python for data science Interactively at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version                  Retrieve SparkContext version
>>> sc.pythonVer                Retrieve Python version
>>> sc.master                   Master URL to connect to
>>> str(sc.sparkHome)           Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())         Retrieve name of the Spark User running SparkContext
>>> sc.appName                  Return application name
>>> sc.applicationId            Retrieve application ID
>>> sc.defaultParallelism       Return default level of parallelism
>>> sc.defaultMinPartitions     Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()      List the number of partitions
>>> rdd.count()                 Count RDD instances
 3
>>> rdd.countByKey()            Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()          Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()          Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()                  Sum of RDD elements
 4950
>>> sc.parallelize([]).isEmpty()  Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max()                  Maximum value of RDD elements
  99
>>> rdd3.min()                  Minimum value of RDD elements
  0
>>> rdd3.mean()                 Mean value of RDD elements
  49.5
>>> rdd3.stdev()                Standard deviation of RDD elements
  28.866070047722118
>>> rdd3.variance()             Compute variance of RDD elements
  833.25
>>> rdd3.histogram(3)           Compute histogram by bins
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                Summary statistics (count, mean, stdev, max & min)
```

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element and flatten the result

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)   Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Selecting Data

Getting

```
>>> rdd.collect()               Return a list with all RDD elements
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                 Take first 2 RDD elements
  [('a', 7), ('a', 2)]
>>> rdd.first()                 Take first RDD element
  ('a', 7)
>>> rdd.top(2)                  Take top 2 RDD elements
  [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()   Return sampled subset of rdd3
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x)   Filter the RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()   Return distinct RDD values
  ['a',2,'b',7]
>>> rdd.keys().collect()        Return (key,value) RDD's keys
  ['a', 'a', 'b']
```

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)              Apply a function to all RDD elements
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)   Merge the rdd values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)   Merge the rdd values
  ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)   Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()            Group rdd by key
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)   Aggregate RDD elements of each partition and then the results
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)   Aggregate values of each RDD key
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)            Aggregate the elements of each partition, and then the results
  4950
>>> rdd.foldByKey(0, add)       Merge the values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)   Create tuples of RDD elements by applying a function
       .collect()
```

Mathematical Operations

```
>>> rdd.subtract(rdd2)          Return each rdd value not contained in rdd2
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)     Return each (key,value) pair of rdd2 with no matching key in rdd
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()   Return the Cartesian product of rdd and rdd2
```

Sort

```
>>> rdd2.sortBy(lambda x: x[1])   Sort RDD by given function
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()            Sort (key, value) RDD by key
       .collect()
  [('a',2),('b',1),('d',1)]
```

Repartitioning

```
>>> rdd.repartition(4)          New RDD with 4 partitions
>>> rdd.coalesce(1)             Decrease the number of partitions in the RDD to 1
```

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

NLP Cheat Sheet

Bag of Words Methods

Information Retrieval

Cosine similarity metric

Define a similarity metric between topic vectors. It is the cosine of the angle between the term vectors.

$$sim(A,B) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}}$$

Unusual words determine the topic much more than common words. Weight each term frequency tf_i by its inverse document frequency idf.

$$idf_i = \log(\frac{N}{n_i})$$

where N=size of collection and n_i = number of documents containing term i

$$w_i = tf_i \times idf_i$$

$$sim(A,B) = \frac{\sum_{w \in A,B} tf_{w,A} \times tf_{w,B} \times idf_w^2}{\sqrt{\sum_{w_1 \in A} (tf_{w_1,A} idf_{w_1})^2} \times \sqrt{\sum_{w_2 \in B} (tf_{w_2,A} idf_{w_2})^2}}$$

Sentiment analysis (opinion mining)

Opinion mining is the task of judging whether a document expresses a positive or a negative opinion (or no opinion) regarding a particular object or topic.

The simplest strategy uses a bag-of-words model. We create lists of 'positive' and 'negative' words and judge a document based on whether it has a preponderance of positive or negative words (and judge it neutral or 'objective' if it has few words of either category.

Creating such lists is not easy; some of the words are likely to be quite different for different kinds of topics. As an alternative, we will take a collection of documents on some topic and rate them (by hand) as positive or negative. We will then use this collection to train a classifier model.

$c = argmax_{c \in C} P(c) \prod_{i \in positions} P(w_i|c)$

$P(c) = \frac{count(\text{docs labeled t})}{N}$

$P(w_i|c) = \frac{count(\text{docs labeled t containing } w_i)}{count(\text{docs labeled t})}$

Smoothing: $P(w_i|c) = \frac{count(w_i,c)+1}{(\sum_{w \in V} count(w,c))+|V|}$

The bag-of-words strategy with NB model does quite well for short reviews, but fails for:

- ambiguous terms
- negation
- comparative reviews
- revealing aspects of an opinion

Context-free grammar

A context-free grammar consists of:

- a set of non-terminal symbols $A, B, C, \cdots \in N$
- a set of terminal symbols $a, b, c, \cdots \in T$
- a start symbol $S \in N$
- a set of productions P of the form $N \to (NUT)*$

CFG can be used to generate sentences, recognize sentences or parse sentences.

Parsing

The CKY algorithm

The grammar has to be in **Chomsky normal form (CNF)**. Every production must have the form $A \to BC$ or $A \to a$.

- Use a parse table $PT[i,j]$
  - each entry is a list of grammar symbols
  - $PT[i,j]$ includes x if the words from i up to (but not including) j can be analyzed as an x
- Two steps
  - if there is a rule $A \to a$ and word i is an a, put an A in $PT[i,i+1]$
  - if there is a rule $A \to BC$, $PT[i,j]$ includes A and $PT[j,k]$ includes B, add C to $PT[i,k]$

Time complexity = $O(n^3)$
Space complexity = $O(n^2)$

POS tagging

Accuracy

for part-of-speech tagging, accuracy is a simple and reasonable metric.

$accuracy = \frac{\text{tokens with correct tag}}{totaltokens}$

Precision and recall

Instead of counting the tags themselves, we count the names defined by these tags: key = number of names in key response = number of names in system response correct = number of names in response which exactly match (in type and extent) a name in the key

$precision = correct/response$
$recall = correct/key$
Also
$precision$ = True Positives/(True Positives + False Positives)
$recall$ = True Positives/(True Positives + False Negatives)

F-measure: the harmonic mean of recall and precision.
$F = 2 \times \frac{precision \times recall}{precision + recall}$

HMMs and the Viterbi decoder

HMM is a Weighted Finite-state Automaton (WFSA)

- Each transition arc is associated with a probability
- The sum of all arcs outgoing from a single node is 1

Viterbi decoder's time complexity is $O(s^2 n)$ and the space complexity is $O(sn)$ for s states and n words.
$viterbi[s,t] = max_{s'}(viterbi[s',t-1] \times p(s,s') \times P(token[t]|s))$
for each s, t: record which $s, t-1$ contributed the maximum.

Chunkers and name taggers

BIO (IOB) tags

$I \to$ inside a baseNP
$O \to$ outside a baseNP
$B \to$ start of a baseNP which follows another baseNP

Maximum entropy

MaxEnt is typically used for a multi-class classifier. We are given a set of training data, where each datum is labeled with a set of features and a class (tag). Each feature-class pair constitutes an indicator function. We train a classifier using this data, computing the s. We can then classify new data by selecting the class (tag) which maximizes $p(h,t)$.

MaxEnt models can be trained by providing a set of K features in the form of binary-valued indicator functions $f_i(h,t)$.

We will use a log-linear model, where
$p(h,t) = (1/Z) \prod_{i=1}^{K} \alpha_i^{f_i(h,t)}$
where α_i is the weight for feature i, and Z is a normalizing constant.

Maximum Entropy Markov Models (MEMM)

Maximum entropy modeling can be combined with a Markov model, so that, for each state, the probability of a transition to that state is computed by a Max Ent model, based on the prior state and arbitrary features of the input sequence. The result is a Maximum Entropy Markov Model (MEMM). Decoding (selecting the best tag sequence) can be done deterministically left-to-right or (better) using Viterbi decoding like an HMM.

HMM vs MEMM

HMM is a generative model, while MEMM is a discriminative model. MEMM can incorporate long distance features.

Feature engineering

When using a package such as MaxEnt, the computational linguist's job becomes one of feature engineering – identifying the features which will be most predictive of the tags we are trying to assign. Simple features are the current token (or the previous token, or the next token) having a particular value, or being on a particular list (such as a list of common titles or common first names).

Lexical semantics and word sense disambiguation

Terminology

- multiple senses of a word
- polysemy (and homonymy for totally unrelated senses ("bank"))
- metonomy for certain types of regular, productive polysemy ("the White House", "Washington")
- zeugma (conjunction combining distinct senses) as test for polysemy ("serve")
- synonymy: when two words mean (more-or-less) the same thing
- hyponymy: X is the hyponym of Y if X denotes a more specific subclass of Y (X is the hyponym, Y is the hypernym)
- synsets: synonym sets

Word Sense Disambiguation

process of identifying the sense of a word in context
Simple supervised WSD algorithm: naive Bayes:
selected sense s' = $argmax(senses)P(s|F) =$
$argmax(s)P(s)\prod_i P(f[i]|s)$ where $F = \{f1, f2, \cdots\}$ is the set of context features
Training a naive Bayes classifier consists of estimating each of these probabilities.
$P(s_i) = \frac{count(s_i, w_j)}{count(w_j)}$
Number of times the sense s_i occurs in w_j and divide by the total count of the target word w_j.
$P(f_j|s) = \frac{count(f_j, s)}{count(s)}$

Word Embeddings

Word embeddings are low-dimensional (50-200 element) real vectors that encode information on the contexts in which a word appears. Each component of the vector combines information on multiple contexts.
The embeddings can be produced by dimensionality reduction on the context matrix.

Information Content

$P(c)$: for each concept (synset), $P(c)$ = probability that a word in a corpus is an instance of the concept (matches the synset c or one of its hyponyms)
Information content of a concept $IC(c) = -\log P(c)$

Probabilistic CFG

Associate a probability with each production

- sum of probabilities of productions expanding a symbol = 1
- use MLE

$P(A \to \alpha) = count(A \to \alpha)/\sum_\beta count(A|\beta)$
Probability of parse = product of prob. of productions

Learning

Supervised learning: All training data is labeled
Semi-Supervised learning

- Part of training data is labeled
- Make use of redundancies to learn labels of additional data, then train model
- Co-training
- Reduces amount of data which must be hand-labeled to achieve a given level of performance

Active learning:

- Start with partially labeled data
- System selects additional 'informative' examples for user to label

Unsupervised learning: driven entirely by an unlabeled corpus
Distant supervision: given a large amount of tabular information, we use a heuristic procedure to (partially) label a text corpus. Distant supervision works roughly like a half iteration of bootstrapping, but with a large set of seeds. Given a data base with a large set of instances of a relation and a large raw text corpus, we use the instances to label the corpus and then train a classifier over the corpus.
Confidence of pattern P: $Conf(P) = \frac{P.positive}{P.positive+P.negative}$
where P.positive = number of positive matches to pattern P and P.negative = number of negative matches to pattern P

Reference resolution

Identify all phrases which refer to the same real-word entity.

Terminology

Referent: real world object referred to
Referring expression [mention]: a phrase referring to that object
Discourse entities: the set of objects referred to by a text
Coreference: two expressions referring to the same thing.
Antecedent: prior expression
Anaphor: following expression
Types of referring expressions: definite pronouns (he, she, it), indefinite pronouns (one), definite NPs (the car), indefinite NPs (a car), names.
Wikification: entity linking, so map each document-level entity to an entry in a standard database as Wikipedia

Complications

- inferrables: sometimes the relation between anaphor and antecedent is not one of identity
- zero Anaphora: many languages allow subject omission, and some allow omission of other arguments
- Cataphora: pronoun referring to a following mention
- Bridging Anaphora: reference to related object
- Non-NP Anaphora: pronouns can also refer to events or propositions

Hobbs search

In selecting which sentences to search, Hobbs starts with the sentence containing the anaphor and then selects prior sentences from right to left, examining first the immediately preceding sentence. However, within an individual sentence he searches from left to right; this is the way he captures the preference for antecedents in subject position. The resulting search order defines a Hobbs distance: if in searching for antecedents of X the $d-th$ candidate we consider is Y, we say that the Hobbs distance from X to Y is d.

Resolving pronoun reference

Constraints:

- animacy
- gender
- number

Preferences:

- recent: at most 3 sentences back
- salient: mentioned several times recently
- subjects:

Selectional preferences: prefer antecedent that is more likely to occur in context of pronoun.
Ge, Hale, and Charniak formula:
$P = P1$(correct antecedent is at Hobbs distance d) $\times$
$P2$(pronoun|head of antecedent) $\times$
$P3$(antecedent|mention count) $\times$
$P4$(head of antecedent|context of pronoun)

Models

- mention-pair model: train binary classifier
  - scan mention in text order
  - link each mention to the closest antecedent classified
  - link each mention to antecedent most confidently labeled
  - cluster mentions
- entity-mention model: uses information from all mentions gathered so far

Question Answering

Mean Reciprocal Rank

Typically for evaluations QA systems generate a ranked set of answers to each query and are scored based on the rank of the first correct answer: mean reciprocal rank

$$MRR = (\frac{1}{N})\sum_i \frac{1}{rank_i}$$

Each question is then scored according to the reciprocal of the rank of the first correct answer. The score of a system is then the average of the score for each question in the set.

Machine translation

The Noisy Channel Model

Suppose we are translating French/Foreign sentences **F** to English **E** We want to determine the English sentence most likely to have generated it:

$$\begin{aligned} E' &= argmax_E P(E|F) \\ &= argmax_E \frac{P(F|E)P(E)}{P(F)} \\ &= argmax_E P(F|E)P(E) \end{aligned}$$

It combines **translation model** $P(F|E)$ and **language model** $P(E)$. $P(F|E)$ is learned from parallel corpus, $P(E)$ can be learned from large monolingual corpus.

The Lexical Translation Model

IBM Model 1

Translation is based on alignment. Mapping a target word at position i to a source word at position j. Assumptions:

- Each target word is generated by exactly one source word
- A target word can be generated by a NULL word; multiple target words can be generated by the same source word
- All alignments are equally likely (a very crude model)

Translation model generates a sentence of F (given E) in 3 steps:

- pick a length for F
- pick an alignment of F (length J) and E (length I+1) $P(A|E) = \frac{\epsilon}{(I+1)^J}$
- pick the ith word of F based on English word e_j with which it aligns using distribution $t(f_i|e_j)$ $P(F|E,A) = \prod_j t(f_j, e_{a_j})$

EM training

Typically we are given sentence aligned but not word aligned corpora, so we cannot compute these probabilities by direct counting and instead use an iterative procedure called "EM".

$$P(F, A|E) = P(F|E, A) \times P(A|E) = \frac{\epsilon}{(I+1)^J} \prod_j t(f_j, e_{a_j}) \tag{1}$$

$$P(A|F, E) = \frac{P(F, A|E)}{\sum_A P(F, A|E)} \tag{2}$$

$$P(F|A) = \sum_A P(F, A|E)/P(A|F, E) = \frac{P(F, A|E)}{\sum_A P(A|F, E)}$$

The heart of the translation model is the word translation probabilities $t(f_j|e_i)$.

Algorithm:

- EM begins by assuming all word translations $t(f_i|e_j)$ are equally likely
- compute probabilities of alignments given word translation probabilities (E-step) following (**??**) and (**??**)
- compute counts of aligned word pairs, weighted by alignment probabilities (E-step)
- recompute MLE word translation probabilities from these counts (M-step)
- repeat

Text Analysis with NLTK Cheatsheet

```
>>> import nltk
>>> nltk.download()        This step will bring up a window in which you can download 'All Corpora'
>>> from nltk.book import *
```

Basics

tokens	>>> text1[0:100] - *first 101 tokens*
	>>> text2[5] - *fifth token*
concordance	>>> text3.concordance('begat') - *basic keyword-in-context*
	>>> text1.concordance('sea', lines=100) - *show other than default 25 lines*
	>>> text1.concordance('sea', lines=all) - *show all results*
	>>> text1.concordance('sea', 10, lines=all) - *change left and right context width to 10 characters and show all results*
similar	>>> text3.similar('silence') - *finds all words that share a common context*
common_contexts	>>>text1.common_contexts(['sea','ocean'])

Counting

Count a string	>>>len('this is a string of text') – *number of characters*
Count a list of tokens	>>>len(text1) –*number of tokens*
Make and count a list of unique tokens	*>>>len(set(text1)) – notice that set return a list of unique tokens*
Count occurrences	>>> text1.count('heaven') – *how many times does a word occur?*
Frequency	>>>fd = nltk.FreqDist(text1) – *creates a new data object that contains information about word frequency*
	>>>fd['the'] – *how many occurences of the word 'the'*
	>>>fd.keys() – *show the keys in the data object*
	>>>fd.values() – *show the values in the data object*
	>>>fd.items() – *show everything*
	>>>fd.keys()[0:50] – *just show a portion of the info.*
Frequency plots	>>>fd.plot(50,cumulative=False) – *generate a chart of the 50 most frequent words*
Other FreqDist functions	>>>fd.hapaxes()
	>>>fd.freq('the')
Get word lengths	>>>lengths = [len(w) for w in text1]
And do FreqDist	>>> fd = nltk.FreqDist(lengths)
FreqDist as a table	>>>fd.tabulate()

Normalizing

De-punctuate	>>>[w for w in text1 if w.isalpha()] – *not so much getting rid of punctuation, but keeping alphabetic characters*
De-uppercaseify (?)	>>>[w.lower() for w in text] – *make each word in the tokenized list lowercase*
	>>>[w.lower() for w in text if w.isalpha()] – *all in one go*
Sort	>>>sorted(text1) – *careful with this!*
Unique words	>>>set(text1) – *set is oddly named, but very powerful. Leaves you with a list of only one of each word.*
Exclude stopwords	*Make your own list of word to be excluded:*
	>>>stopwords = ['the','it','she','he']
	>>>mynewtext = [w for w in text1 if w not in stopwords]
	Or you can also use predefined stopword lists from NLTK:
	>>>from nltk.corpus import stopwords
	>>>stopwords = stopwords.words('english')
	>>> mynewtext = [w for w in text1 if w not in stopwords]

Searching

Dispersion plot	>>>text4.dispersion_plot([‘American’,’Liberty’,’Government’])
Find word that end with…	>>>[w for w in text4 if w.endswith(‘ness’)]
Find words that start with…	>>>[w for w in text4 if w.startsswith(‘ness’)]
Find words that contain…	>>>[w for w in text4 if ‘ee’ in w]
Combine them together:	>>>[w for w in text4 if ‘ee’ in w and w.endswith(‘ing’)]
Regular expressions	*‘Regular expressions’ is a syntax for describing sequences of characters usually used to construct search queries. The Python ‘re’ module must first be imported:* >>>import re >>>[w for w in text1 if re.search('^ab',w)] – *‘Regular expressions’ is too big of a topic to cover here. Google it!*

Chunking

	Collocations are good for getting a quick glimpse of what a text is about
Collocations	>>> text4.collocations() - *multi-word expressions that commonly co-occur. Notice that is not necessarily related to the frequency of the words.* *>>>text4.collocations(num=100) – alter the number of phrases returned* *Bigrams, Trigrams, and n-grams are useful for comparing texts, particularly for plagiarism detection and collation*
Bi-grams	>>>nltk.bigrams(text4) – *returns every string of two words*
Tri-grams	>>>nltk.trigrams(text4) – *return every string of three words*
n-grams	>>>nltk.ngrams(text4, 5)

Tagging

part-of-speech tagging	>>>mytext = nltk.word_tokenize(“This is my sentence”) >>> nltk.pos_tag(mytext)

Working with your own texts:

Open a file for reading	>>>file = open(‘myfile.txt’) – *make sure you are in the correct directory before starting Python*
Read the file	>>>t = file.read();
Tokenize the text	>>>tokens = nltk.word_tokenize(t)
Convert to NLTK Text object	>>>text = nltk.Text(tokens)

Quitting Python

Quit	>>>quit()

Part-of-Speech Codes

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Resources

Python for Humanists 1: Why Learn Python?
http://www.rogerwhitson.net/?p=1260

'Natural Language Processing with Python' book online
http://www.nltk.org/book/

Commands for altering lists – useful in creating stopword lists

list.append(x) - Add an item to the end of the list
list.insert(i, x) - Insert an item, i, at position, x.
list.remove(x) - Remove item whose value is x.
list.pop(x) - Remove item numer x from the list.

Cheatography

Natural Language Processing with Python & nltk Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15485/

Handling Text

`text='Some words'`	assign string
`list(text)`	Split text into character tokens
`set(text)`	Unique tokens
`len(text)`	Number of characters

Accessing corpora and lexical resources

`from nltk.corpus import brown`	import CorpusReader object
`brown.words(text_id)`	Returns pretokenised document as list of words
`brown.fileids()`	Lists docs in Brown corpus
`brown.categories()`	Lists categories in Brown corpus

Tokenization

`text.split(" ")`	Split by space
`nltk.word_tokenizer(text)`	nltk in-built word tokenizer
`nltk.sent_tokenize(doc)`	nltk in-built sentence tokenizer

Lemmatization & Stemming

`input="List listed lists listing listings"`	Different suffixes
`words=input.lower().split(' ')`	Normalize (lowercase) words
`porter=nltk.PorterStemmer`	Initialise Stemmer
`[porter.stem(t) for t in words]`	Create list of stems
`WNL=nltk.WordNetLemmatizer()`	Initialise WordNet lemmatizer
`[WNL.lemmatize(t) for t in words]`	Use the lemmatizer

Part of Speech (POS) Tagging

`nltk.help.upenn_tagset('MD')`	Lookup definition for a POS tag
`nltk.pos_tag(words)`	nltk in-built POS tagger
	<use an alternative tagger to illustrate ambiguity>

Sentence Parsing

`g=nltk.data.load('grammar.cfg')`	Load a grammar from a file
`g=nltk.CFG.fromstring("""..."")`	Manually define grammar
`parser=nltk.ChartParser(g)`	Create a parser out of the grammar
`trees=parser.parse_all(text)`	
`for tree in trees: ... print tree`	
`from nltk.corpus import treebank`	
`treebank.parsed_sents('wsj_0001.mrg')`	Treebank parsed sentences

Text Classification

`from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer`	
`vect=CountVectorizer().fit(X_train)`	Fit bag of words model to data
`vect.get_feature_names()`	Get features
`vect.transform(X_train)`	Convert to doc-term matrix

Entity Recognition (Chunking/Chinking)

`g="NP: {<DT>?<JJ>*<NN>}"`	Regex chunk grammar
`cp=nltk.RegexpParser(g)`	Parse grammar
`ch=cp.parse(pos_sent)`	Parse tagged sent. using grammar
`print(ch)`	Show chunks
`ch.draw()`	Show chunks in IOB tree
`cp.evaluate(test_sents)`	Evaluate against test doc
`sents=nltk.corpus.treebank.tagged_sents()`	
`print(nltk.ne_chunk(sent))`	Print chunk tree

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 28th May, 2018.
Last updated 29th May, 2018.

Cheatography

Natural Language Processing with Python & nltk Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15485/

RegEx with Pandas & Named Groups

```
df=pd.DataFrame(time_sents, columns=['text'])
df['text'].str.split().str.len()
df['text'].str.contains('word')
df['text'].str.count(r'\d')
df['text'].str.findall(r'\d')
df['text'].str.replace(r'\w+day\b', '???')
df['text'].str.replace(r'(\w)', lambda x: x.groups()
[0][:3])
df['text'].str.extract(r'(\d?\d):(\d\d)')
df['text'].str.extractall(r'((\d?\d):(\d\d) ?
([ap]m))')
df['text'].str.extractall(r'(?P<digits>\d)')
```

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 28th May, 2018.
Last updated 29th May, 2018.

Introduction

According to industry estimates, only 21% of the available data is present in structured form. Data is being generated as we speak, as we tweet, as we send messages on Whatsapp and in various other activities. Majority of this data exists in the textual form, which is highly unstructured in nature.

Few notorious examples include – tweets / posts on social media, user to user chat conversations, news, blogs and articles, product or services reviews and patient records in the healthcare sector. A few more recent ones includes chatbots and other voice driven bots.

Despite having high dimension data, the information present in it is not directly accessible unless it is processed (read and understood) manually or analyzed by an automated system.

In order to produce significant and actionable insights from text data, it is important to get acquainted with the techniques and principles of Natural Language Processing (NLP).

So, if you plan to create chatbots this year, or you want to use the power of unstructured text, this guide is the right starting point. This guide unearths the concepts of natural language processing, its techniques and implementation. The aim of the article is to teach the concepts of natural language processing and apply it on real data set. Moreover, we also have a video based **course on NLP** with 3 real life projects.

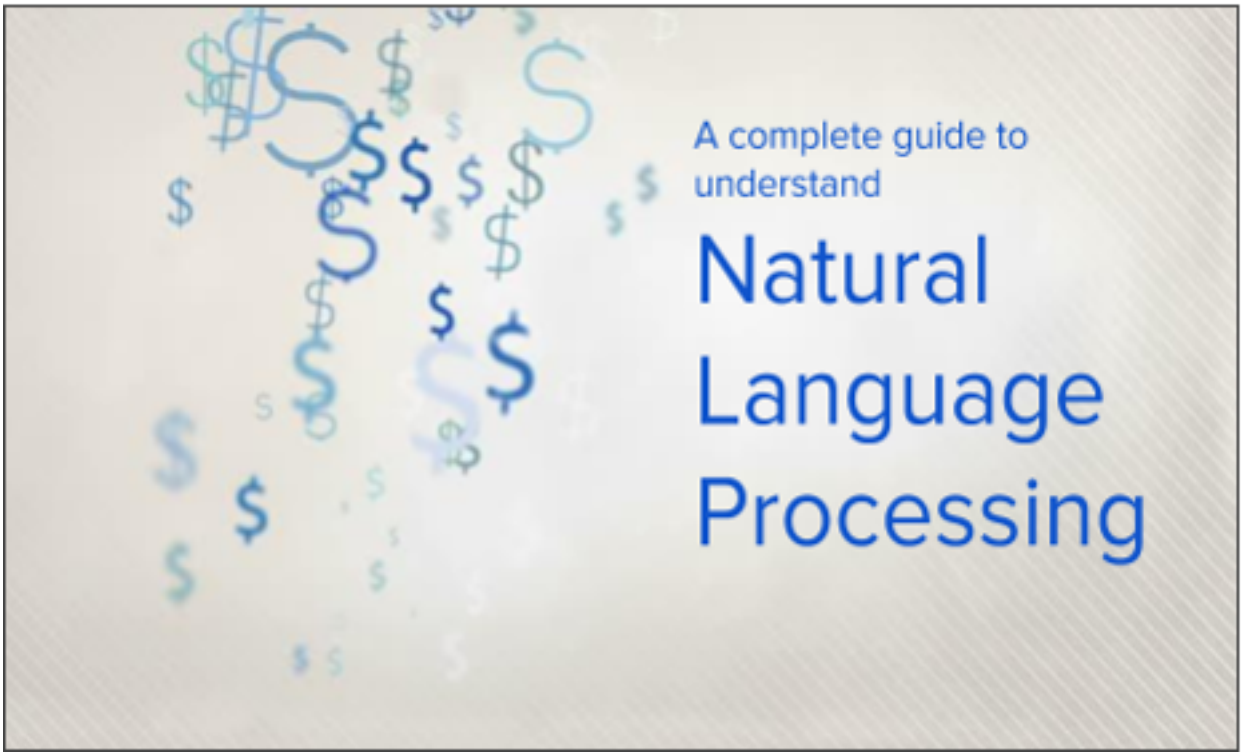


Table of Contents

1. Introduction to Natural Language Processing

NLP is a branch of data science that consists of systematic processes for analyzing, understanding, and deriving information from the text data in a smart and efficient manner. By utilizing NLP and its components, one can organize the massive chunks of text data, perform numerous automated tasks and solve a wide range of problems such as – automatic summarization, machine translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation etc.

Before moving further, I would like to explain some terms that are used in the article:

- Tokenization – process of converting a text into tokens
- Tokens – words or entities present in the text
- Text object – a sentence or a phrase or a word or an article

Steps to install NLTK and its data:

Install Pip: run in terminal:

```
    sudo easy_install pip
```

Install NLTK: run in terminal :

```
    sudo pip install -U nltk
```

Download NLTK data: run python shell (in terminal) and write the following code:

```
```


```
import nltk  nltk.download() ```
```

Follow the instructions on screen and download the desired package or collection. Other libraries can be directly installed using pip.

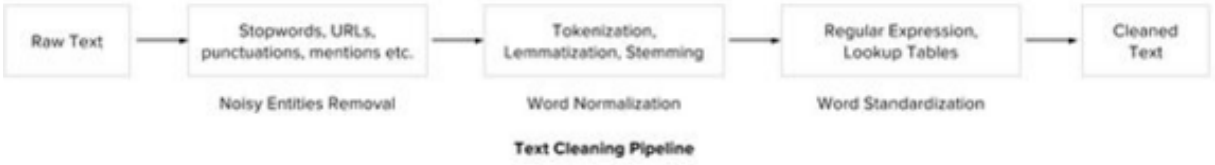
2. Text Preprocessing

Since, text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.

It is predominantly comprised of three steps:

- Noise Removal
- Lexicon Normalization
- Object Standardization

The following image shows the architecture of text preprocessing pipeline.



2.1 Noise Removal

Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.

For example – language stopwords (commonly used words of a language – is, am, the, of, in etc), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noisy entities, and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

Following is the python code for the same purpose.

```
# Sample code to remove noisy words from a text



noise_list = ["is", "a", "this", "..."]

def _remove_noise(input_text):

    words = input_text.split()

    noise_free_words = [word for word in words if word not in noise_list]

    noise_free_text = " ".join(noise_free_words)

    return noise_free_text



_remove_noise("this is a sample text")

>>> "sample text"
```

Another approach is to use the regular expressions while dealing with special patterns of noise. We have explained regular expressions in detail in one of our previous article. Following python code removes a regex pattern from the input text:

```
# Sample code to remove a regex pattern

import re

def _remove_regex(input_text, regex_pattern):

    urls = re.finditer(regex_pattern, input_text)

    for i in urls:

        input_text = re.sub(i.group().strip(), '', input_text)

    return input_text



regex_pattern = "#[\w]*"

_remove_regex("remove this #hashtag from analytics vidhya", regex_pattern)

>>> "remove this  from analytics vidhya"



```

2.2 Lexicon Normalization

Another type of textual noise is about the multiple representations exhibited by single word.

For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma). Normalization is a pivotal step for feature engineering with text as it converts the high dimensional features (N different features) to the low dimensional space (1 feature), which is an ideal ask for any ML model.

The most common lexicon normalization practices are :

- **Stemming:** Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.
- **Lemmatization:** Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

Below is the sample code that performs lemmatization and stemming using python’s popular library – NLTK.

```

from nltk.stem.wordnet import WordNetLemmatizer

lem = WordNetLemmatizer()



from nltk.stem.porter import PorterStemmer

stem = PorterStemmer()



word = "multiplying"

lem.lemmatize(word, "v")

>> "multiply"

stem.stem(word)

>> "multipli"



```

2.3 Object Standardization

Text data often contains words or phrases which are not present in any standard lexical dictionaries. These pieces are not recognized by search engines and models.

Some of the examples are – acronyms, hashtags with attached words, and colloquial slangs. With the help of regular expressions and manually prepared data dictionaries, this type of noise can be fixed, the code below uses a dictionary lookup method to replace social media slangs from a text.

```
lookup_dict = {'rt':'Retweet', 'dm':'direct message', "awsm" : "awesome", "lu
v" :"love", "..."}

def _lookup_words(input_text):

    words = input_text.split()

    new_words = []

    for word in words:

        if word.lower() in lookup_dict:

            word = lookup_dict[word.lower()]

        new_words.append(word) new_text = " ".join(new_words)

        return new_text



_lookup_words("RT this is a retweeted tweet by Shivam Bansal")

>> "Retweet this is a retweeted tweet by Shivam Bansal"



```

Apart from three steps discussed so far, other types of text preprocessing includes encoding-decoding noise, grammar checker, and spelling correction etc. The detailed article about preprocessing and its methods is given in one of my previous article**.**

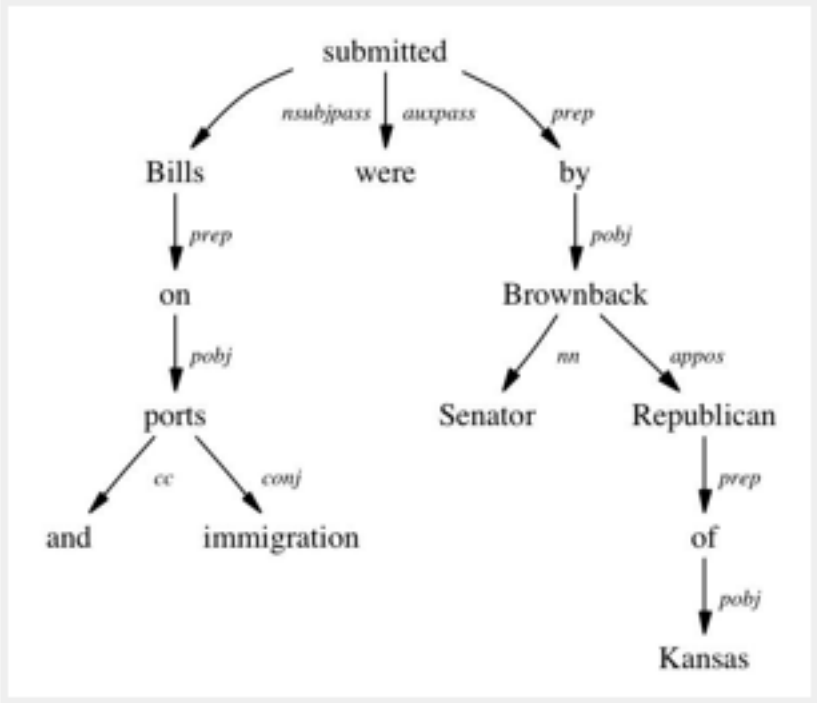
3.Text to Features (Feature Engineering on text data)

To analyse a preprocessed data, it needs to be converted into features. Depending upon the usage, text features can be constructed using assorted techniques – Syntactical Parsing, Entities / N-grams / word-based features, Statistical features, and word embeddings. Read on to understand these techniques in detail.

3.1 Syntactic Parsing

Syntactical parsing invol ves the analysis of words in the sentence for grammar and their arrangement in a manner that shows the relationships among the words. Dependency Grammar and Part of Speech tags are the important attributes of text syntactics.

Dependency Trees – Sentences are composed of some words sewed together. The relationship among the words in a sentence is determined by the basic dependency grammar. Dependency grammar is a class of syntactic text analysis that deals with (labeled) asymmetrical binary relations between two lexical items (words). Every relation can be represented in the form of a triplet (relation, governor, dependent). For example: consider the sentence – "*Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas."* The relationship among the words can be observed in the form of a tree representation as shown:



The tree shows that “submitted” is the root word of this sentence, and is linked by two sub-trees (subject and object subtrees). Each subtree is a itself a dependency tree with relations such as – (“Bills” <-> “ports” <by> “proposition” relation), (“ports” <-> “immigration” <by> “conjugation” relation).

This type of tree, when parsed recursively in top-down manner gives grammar relation triplets as output which can be used as features for many nlp problems like entity wise sentiment analysis, actor & entity identification, and text classification. The python wrapper StanfordCoreNLP (by Stanford NLP Group, only commercial license) and NLTK dependency grammars can be used to generate dependency trees.

Part of speech tagging – Apart from the grammar relations, every word in a sentence is also associated with a part of speech (pos) tag (nouns, verbs, adjectives, adverbs etc). The pos tags defines the usage and function of a word in the sentence. H ere is a list of all possible pos-tags defined by Pennsylvania university. Following code using NLTK performs pos tagging annotation on input text. (it provides several implementations, the default one is perceptron tagger)

```
```

from nltk import word_tokenize, pos_tag

text = "I am learning Natural Language Processing on Analytics Vidhya"

tokens = word_tokenize(text)

print pos_tag(tokens)

>>> [('I', 'PRP'), ('am', 'VBP'), ('learning', 'VBG'), ('Natural', 'NNP'),('L
anguage', 'NNP'),

('Processing', 'NNP'), ('on', 'IN'), ('Analytics', 'NNP'),('Vidhya', 'NNP')]

```
```

Part of Speech tagging is used for many important purposes in NLP:

A.Word sense disambiguation: Some language words have multiple meanings according to their usage. For example, in the two sentences below:

I. “Please book my flight for Delhi”

II. “I am going to read this book in the flight”

“Book” is used with different context, however the part of speech tag for both of the cases are different. In sentence I, the word “book” is used as ***v erb***, while in II it is used as ***no un***. (Lesk Algorithm is also us ed for similar purposes)

B.Improving word-based features: A learning model could learn different contexts of a word when used word as the features, however if the part of speech tag is linked with them, the context is preserved, thus making strong features. For example:

Sentence -"book my flight, I will read this book"

Tokens – ("book", 2), ("my", 1), ("flight", 1), ("I", 1), ("will", 1), ("read", 1), ("this", 1)

Tokens with POS – ("book_VB", 1), ("my_PRP$", 1), ("flight_NN", 1), ("I_PRP", 1), ("will_MD", 1), ("read_VB", 1), ("this_DT", 1), ("book_NN", 1)

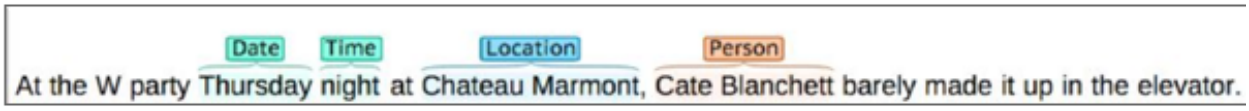
C. Normalization and Lemmatization: POS tags are the basis of lemmatization process for converting a word to its base form (lemma).

D.Efficient stopword removal : P OS tags are also useful in efficient removal of stopwords.

For example, there are some tags which always define the low frequency / less important words of a language. For example: (**IN** – "within", "upon", "except"), (**CD** – "one","two", "hundred"), (**MD** – "may", "mu st" etc)

3.2 Entity Extraction (Entities as features)

Entities are defined as the most important chunks of a sentence – noun phrases, verb phrases or both. Entity Detection algorithms are generally ensemble models of rule based parsing, dictionary lookups, pos tagging and dependency parsing. The applicability of entity detection can be seen in the automated chat bots, content analyzers and consumer insights.



Topic Modelling & Named Entity Recognition are the two key entity detection methods in NLP.

A. Named Entity Recognition (NER)

The process of detecting the named entities such as person names, location names, company names etc from the text is called as NER. For example :

Sentence – Sergey Brin, the manager of Google Inc. is walking in the streets of New York.

Named Entities – (“person” : “Sergey Brin”), (“org” : “Google Inc.”), (“location” : “New York”)

A typical NER model consists of three blocks:

Noun phrase identification: This step deals with extracting all the noun phrases from a text using dependency parsing and part of speech tagging.

Phrase classification: This is the classification step in which all the extracted noun phrases are classified into respective categories (locations, names etc). Google Maps API provides a good path to disambiguate locations, Then, the open databases from dbpedia, wikipedia can be used to identify person names or company names. Apart from this, one can curate the lookup tables and dictionaries by combining information from different sources.

Entity disambiguation: Sometimes it is possible that entities are misclassified, hence creating a validation layer on top of the results is useful. Use of knowledge graphs can be exploited for this purposes. The popular knowledge graphs are – Google Knowledge Graph, IBM Watson and Wikipedia.

B. Topic Modeling

Topic modeling is a process of automatically identifying the topics present in a text corpus, it derives the hidden patterns among the words in the corpus in an unsupervised manner. Topics are defined as “a repeating pattern of co-occurring terms in a corpus”. A good topic model results in – “health”, “doctor”, “patient”, “hospital” for a topic – Healthcare, and “farm”, “crops”, “wheat” for a topic – “Farming”.

Latent Dirichlet Allocation (LDA) is the most popular topic modelling technique, Following is the code to implement topic modeling using LDA in python. For a detailed explanation about its working and implementation, check the complete article here.

```
```

doc1 = "Sugar is bad to consume. My sister likes to have sugar, but not my fa
ther."

doc2 = "My father spends a lot of time driving my sister around to dance prac
tice."

doc3 = "Doctors suggest that driving may cause increased stress and blood pre
ssure."

doc_complete = [doc1, doc2, doc3]

doc_clean = [doc.split() for doc in doc_complete]

import gensim from gensim

import corpora

Creating the term dictionary of our corpus, where every unique term is assi
gned an index.

dictionary = corpora.Dictionary(doc_clean)
```
```

```
# Converting list of documents (corpus) into Document Term Matrix using dicti
onary prepared above.

doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]



# Creating the object for LDA model using gensim library

Lda = gensim.models.ldamodel.LdaModel



# Running and Training LDA model on the document term matrix

ldamodel = Lda(doc_term_matrix, num_topics=3, id2word = dictionary, passes=50
)



# Results

print(ldamodel.print_topics())



```

C. N-Grams as Features

A combination of N words together are called N-Grams. N grams (N > 1) are generally more informative as compared to words (Unigrams) as features. Also, bigrams (N = 2) are considered as the most important features of all the others. The following code generates bigram of a text.

```
```

```
def generate_ngrams(text, n):

    words = text.split()

    output = []

    for i in range(len(words)-n+1):

        output.append(words[i:i+n])

    return output



>>> generate_ngrams('this is a sample text', 2)

# [['this', 'is'], ['is', 'a'], ['a', 'sample'], , ['sample', 'text']]
```

```
```

3.3 Statistical Features

Text data can also be quantified directly into numbers using several techniques described in this section:

A. Term Frequency – Inverse Document Frequency (TF – IDF)

TF-IDF is a weighted model commonly used for information retrieval problems. It aims to convert the text documents into vector models on the basis of occurrence of words in the documents without taking considering the exact ordering. For Example – let say there is a dataset of N text documents, In any document "D", TF and IDF will be defined as –

Term Frequency (TF) – TF for a term "t" is defined as the count of a term "t" in a document "D"

Inverse Document Frequency (IDF) – IDF for a term is defined as logarithm of ratio of total documents available in the corpus and number of documents containing the term T.

TF . IDF – TF IDF formula gives the relative importance of a term in a corpus (list of documents), given by the following formula below. Following is the code using python's scikit learn package to convert a text into tf idf vectors:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

tf_{ij} = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

```
from sklearn.feature_extraction.text import TfidfVectorizer

obj = TfidfVectorizer()

corpus = ['This is sample document.', 'another random document.', 'third samp
le document text']

X = obj.fit_transform(corpus)

print X

>>>

(0, 1) 0.345205016865

(0, 4) ... 0.444514311537

(2, 1) 0.345205016865

(2, 4) 0.444514311537
```

The model creates a vocabulary dictionary and assigns an index to each word. Each row in the output contains a tuple (i,j) and a tf-idf value of word at index j in document i.

B. Count / Density / Readability Features

Count or Density based features can also be used in models and analysis. These features might seem trivial but shows a great impact in learning models. Some of the features are: Word Count, Sentence Count, Punctuation Counts and Industry specific word counts. Other types of measures include readability measures such as syllable counts, smog index and flesch reading ease. Refer to Textstat library to create such features.

3.4 Word Embedding (text vectors)

Word embedding is the modern way of representing words as vectors. The aim of word embedding is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are widely used in deep learning models such as Convolutional Neural Networks and Recurrent Neural Networks.

Word2Vec and GloVe are the two popular models to create word embedding of a text. These models takes a text corpus as input and produces the word vectors as output.

Word2Vec model is composed of preprocessing module, a shallow neural network model called Continuous Bag of Words and another shallow neural network model called skip-gram. These models are widely used for all other nlp problems. It first constructs a vocabulary from the training corpus and then learns word embedding representations. Following code using gensim package prepares the word embedding as the vectors.

```
from gensim.models import Word2Vec

sentences = [['data', 'science'], ['vidhya', 'science', 'data', 'analytics'],
['machine', 'learning'], ['deep', 'learning']]



# train the model on your corpus

model = Word2Vec(sentences, min_count = 1)



print model.similarity('data', 'science')

>>> 0.11222489293

print model['learning']

>>> array([ 0.00459356  0.00303564 -0.00467622  0.00209638, ...])



```

They can be used as feature vectors for ML model, used to measure text similarity using cosine similarity techniques, words clustering and text classification techniques.

4. Important tasks of NLP

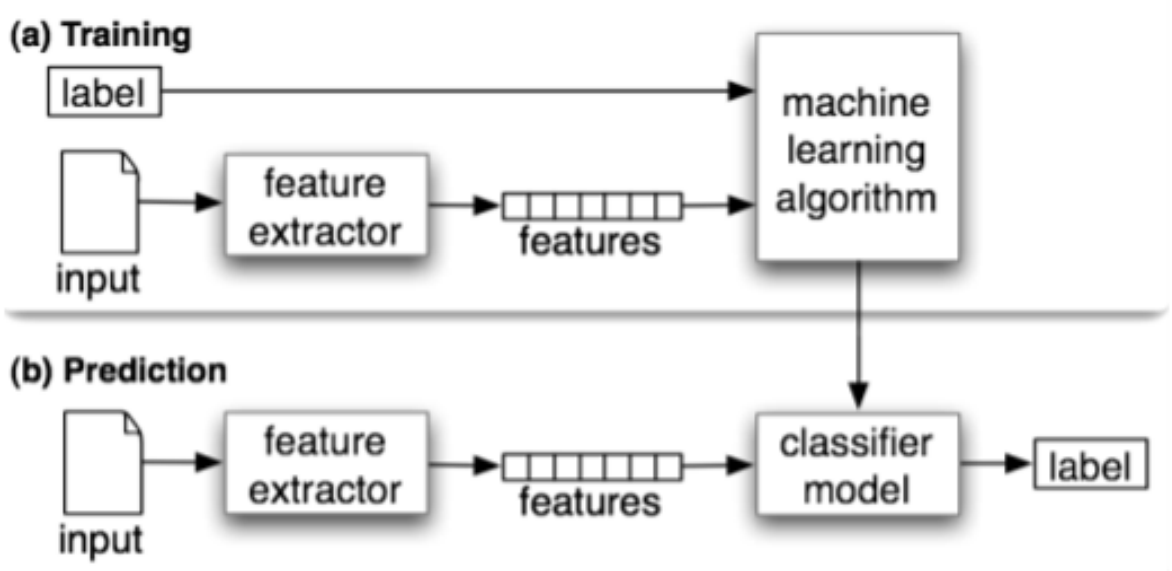
This section talks about different use cases and problems in the field of natural language processing.

4.1 Text Classification

Text classification is one of the classical problem of NLP. Notorious examples include – Email Spam Identification, topic classification of news, sentiment classification and organization of web pages by search engines.

Text classification, in common words is defined as a technique to systematically classify a text object (document or sentence) in one of the fixed category. It is really helpful when the amount of data is too large, especially for organizing, information filtering, and storage purposes.

A typical natural language classifier consists of two parts: (a) Training (b) Prediction as shown in image below. Firstly the text input is processes and features are created. The machine learning models then learn these features and is used for predicting against the new text.



Here is a code that uses naive bayes classifier using text blob library (built on top of nltk).

```
```

```
from textblob.classifiers import NaiveBayesClassifier as NBC

from textblob import TextBlob

training_corpus = [

                   ('I am exhausted of this work.', 'Class_B'),

                   ("I can't cooperate with this", 'Class_B'),

                   ('He is my badest enemy!', 'Class_B'),

                   ('My management is poor.', 'Class_B'),

                   ('I love this burger.', 'Class_A'),

                   ('This is an brilliant place!', 'Class_A'),

                   ('I feel very good about these dates.', 'Class_A'),

                   ('This is my best work.', 'Class_A'),

                   ("What an awesome view", 'Class_A'),

                   ('I do not like this dish', 'Class_B')]
```

```
test_corpus = [

                ("I am not feeling well today.", 'Class_B'),

                ("I feel brilliant!", 'Class_A'),

                ('Gary is a friend of mine.', 'Class_A'),

                ("I can't believe I'm doing this.", 'Class_B'),

                ('The date was good.', 'Class_A'), ('I do not enjoy my job',
'Class_B')]



model = NBC(training_corpus)

print(model.classify("Their codes are amazing."))

>>> "Class_A"

print(model.classify("I don't like their computer."))

>>> "Class_B"

print(model.accuracy(test_corpus))

>>> 0.83

```

Scikit.Learn also provides a pipeline framework for text classification:

```
```

```
from sklearn.feature_extraction.text
```

```
import TfidfVectorizer from sklearn.metrics
```

```
import classification_report
```

```
from sklearn import svm
```

```
# preparing data for SVM model (using the same training_corpus, test_corpus f
rom naive bayes example)
```

```
train_data = []
```

```
train_labels = []
```

```
for row in training_corpus:
```

```
    train_data.append(row[0])
```

```
    train_labels.append(row[1])
```

```
test_data = []
```

```
test_labels = []

for row in test_corpus:

    test_data.append(row[0])

    test_labels.append(row[1])



# Create feature vectors

vectorizer = TfidfVectorizer(min_df=4, max_df=0.9)

# Train the feature vectors

train_vectors = vectorizer.fit_transform(train_data)

# Apply model on test data

test_vectors = vectorizer.transform(test_data)



# Perform classification with SVM, kernel=linear

model = svm.SVC(kernel='linear')

model.fit(train_vectors, train_labels)
```

```
prediction = model.predict(test_vectors)

>>> ['Class_A' 'Class_A' 'Class_B' 'Class_B' 'Class_A' 'Class_A']



print (classification_report(test_labels, prediction))

```

The text classification model are heavily dependent upon the quality and quantity of features, while applying any machine learning model it is always a good practice to include more and more training data. H ere are some tips that I wrote about improving the text classification accuracy in one of my previous article.

4.2 Text Matching / Similarity

One of the important areas of NLP is the matching of text objects to find similarities. Important applications of text matching includes automatic spelling correction, data de-duplication and genome analysis etc.

A number of text matching techniques are available depending upon the requirement. This section describes the important techniques in detail.

A. Levenshtein Distance – The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. Following is the implementation for efficient memory computations.

```
def levenshtein(s1,s2):

    if len(s1) > len(s2):

        s1,s2 = s2,s1

    distances = range(len(s1) + 1)

    for index2,char2 in enumerate(s2):

        newDistances = [index2+1]

        for index1,char1 in enumerate(s1):

            if char1 == char2:

                newDistances.append(distances[index1])

            else:

                 newDistances.append(1 + min((distances[index1], distances[in
dex1+1], newDistances[-1])))

        distances = newDistances

    return distances[-1]

print(levenshtein("analyze","analyse"))
```

B. Phonetic Matching – A Phonetic matching algorithm takes a keyword as input (person's name, location name etc) and produces a character string that identifies a set of words that are (roughly) phonetically similar. It is very useful for searching large text corpuses, correcting spelling errors and matching relevant names. Soundex and Metaphone are two main phonetic algorithms used for this purpose. Python's module Fuzzy is used to compute soundex strings for different words, for example –

```
```

```
import fuzzy
```

```
soundex = fuzzy.Soundex(4)
```

```
print soundex('ankit')
```

```
>>> “A523”
```

```
print soundex('aunkit')
```

```
>>> “A523”
```

```
```

C. Flexible String Matching – A complete text matching system includes different algorithms pipelined together to compute variety of text variations. Regular expressions are really helpful for this purposes as well. Another common techniques include – exact string matching, lemmatized matching, and compact matching (takes care of spaces, punctuation's, slangs etc).

D. Cosine Similarity – W hen the text is represented as vector notation, a general cosine similarity can also be applied in order to measure vectorized similarity. Following code converts a text to vectors (using term frequency) and applies cosine similarity to provide closeness among two text.

```
```

```
import math

from collections import Counter

def get_cosine(vec1, vec2):

    common = set(vec1.keys()) & set(vec2.keys())

    numerator = sum([vec1[x] * vec2[x] for x in common])




    sum1 = sum([vec1[x]**2 for x in vec1.keys()])

    sum2 = sum([vec2[x]**2 for x in vec2.keys()])

    denominator = math.sqrt(sum1) * math.sqrt(sum2)




    if not denominator:

        return 0.0
```

```
    else:

        return float(numerator) / denominator



def text_to_vector(text):

    words = text.split()

    return Counter(words)



text1 = 'This is an article on analytics vidhya'

text2 = 'article on analytics vidhya is about natural language processing'



vector1 = text_to_vector(text1)

vector2 = text_to_vector(text2)

cosine = get_cosine(vector1, vector2)

>>> 0.62

```

4.3 Coreference Resolution

Coreference Resolution is a process of finding relational links among the words (or phrases) within the sentences. Consider an example sentence: " Donald went to John's office to see the new table. He looked at it for an hour."

Humans can quickly figure out that "he" denotes Donald (and not John), and that "it" denotes the table (and not John's office). Coreference Resolution is the component of NLP that does this job automatically. It is used in document summarization, question answering, and information extraction. Stanford CoreNLP provides a python wrapper for commercial purposes.

4.4 Other NLP problems / tasks

- **Text Summarization** – Given a text article or paragraph, summarize it automatically to produce most important and relevant sentences in order.
- **Machine Translation** – Automatically translate text from one human language to another by taking care of grammar, semantics and information about the real world, etc.
- **Natural Language Generation and Understanding** – Convert information from computer databases or semantic intents into readable human language is called language generation. Converting chunks of text into more logical structures that are easier for computer programs to manipulate is called language understanding.
- **Optical Character Recognition** – Given an image representing printed text, determine the corresponding text.
- **Document to Information** – This involves parsing of textual data present in documents (websites, files, pdfs and images) to analyzable and clean format.

5. Important Libraries for NLP (python)

- Scikit-learn: Machine learning in Python
- Natural Language Toolkit (NLTK): The complete toolkit for all NLP techniques.
- Pattern – A web mining module for the with tools for NLP and machine learning.
- TextBlob – Easy to use nl p tools API, built on top of NLTK and Pattern.
- spaCy – Industrial strength N LP with Python and Cython.
- Gensim – Topic Modelling for Humans
- Stanford Core NLP – NLP services and packages by Stanford NLP Group.