ALL IMPORTANT CHEAT SHEETS

DATA SCIENCE, MACHINE LEARNING, REINFORCEMENT LEARNING & DATA ENGINEERING

7/7/2020

COMPILED BY ABHISHEK PRASAD

Data Science Cheatsheet

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

Last Updated April 19, 2020

What is Data Science?

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

Two paradigms of data research.

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

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

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

Types of Data

Structured: Data that has predefined structures. e.g. tables, spreadsheets, or relational databases.
Unstructured Data: Data with no predefined structure, comes in any size or form, cannot be easily stored in tables. e.g. blobs of text, images, audio
Quantitative Data: Numerical. e.g. height, weight
Categorical Data: Data that can be labeled or divided into groups. e.g. race, sex, hair color.
Big Data: Massive datasets, or data that contains greater *variety* arriving in increasing *volumes* and with ever-higher *velocity* (3 Vs). Cannot fit in the memory of a single machine.

Data Sources/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

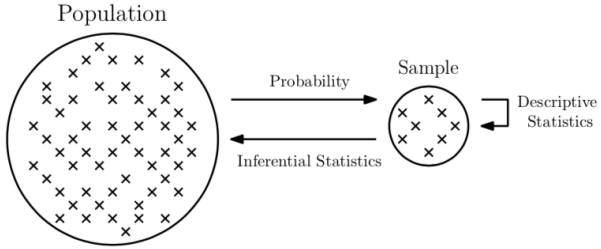
Raw Measures: data that hasn't been transformed yet
Rounding: sometimes precision is noise; round to nearest integer, decimal etc..
Scaling: log, z-score, minmax scale
Imputation: fill in missing values using mean, median, model output, etc..
Binning: transforming numeric features into categorical ones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

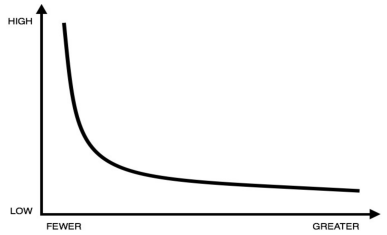
Classic Statistical Distributions

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.

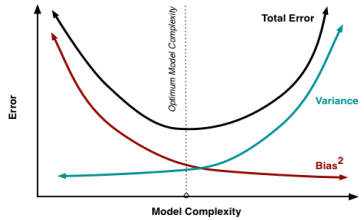
Modeling- Philosophies

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. Some models operate this way, such as boosting and bagging, which uses large number of weak classifiers to produce a strong one.

Modeling- Taxonomy

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

Parametric vs. Nonparametric

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

Supervised vs. Unsupervised

- **Supervised**: models that fit input variables $x_i = (x_1, x_2, ...x_n)$ 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: $\mathrm{F} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
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 squared differences between the actual ith value and the predicted ith value. RSS = $\sum_{i=1}^{n} e_i^2$, where $e_i = y_i - \hat{y_i}$

How to find best fit?

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

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

Gradient descent algorithm in two dimensions. Repeat until convergence.

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

For non-convex functions, gradient descent no longer guarantees an optimal 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

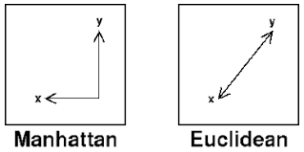
Interpreting examples as points in space provides a way to find natural groupings or clusters among data e.g. which stars are the closest to our sun? Networks can also be built from point sets (vertices) by connecting related points.

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

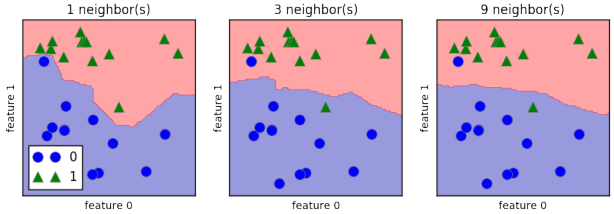
Kullback-Leibler Divergence: $KL(A||B) = \sum_{i=i}^{d} a_i log_2 \frac{a_i}{b_i}$
KL divergence measures the distances between probability distributions by measuring the uncertainty gained or uncertainty lost when replacing distribution A with distribution B. However, this is not a metric but forms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

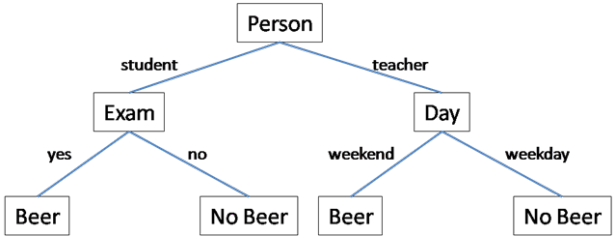
$$C(X) = argmax_{i \in classes(t)} \frac{P(X|C_i)P(C_i)}{P(X)}$$

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

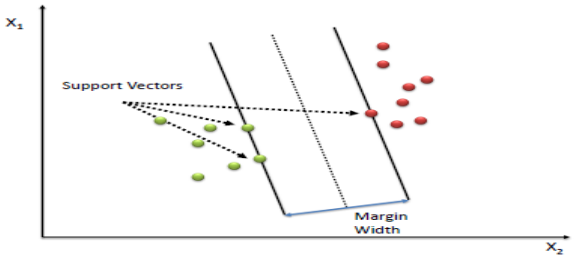
Random Forests: builds on bagging by decorrelating the trees. We do everything the same like in bagging, but when we build the trees, everytime we consider a split, a random sample of the p predictors is chosen as split candidates, not the full set (typically m $\approx \sqrt{p}$). When m = p, then we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

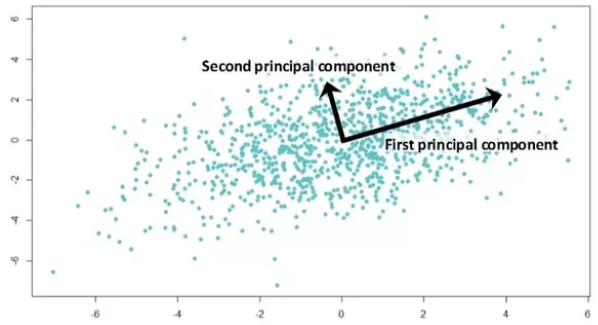
Work by constructing a hyperplane that separates points between two classes. The hyperplane is determined using the maximal margin hyperplane, which is the hyperplane that is the maximum distance from the training observations. This distance is called the margin. Points that fall on one side of the hyperplane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

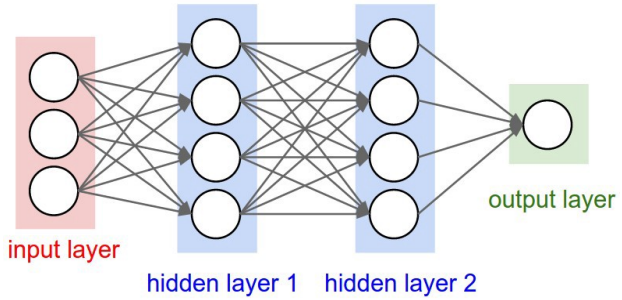
ML Terminology and Concepts

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

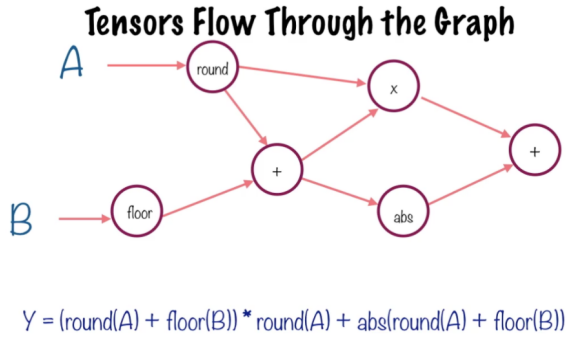
There are different kinds of NNs that are suitable for certain problems, which depend on the NN's architecture.

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)

Graph Theory

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

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

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

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

A graph can be represented:

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

Applications of graph theory include:

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

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

SQL Part I

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

Basic Queries

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

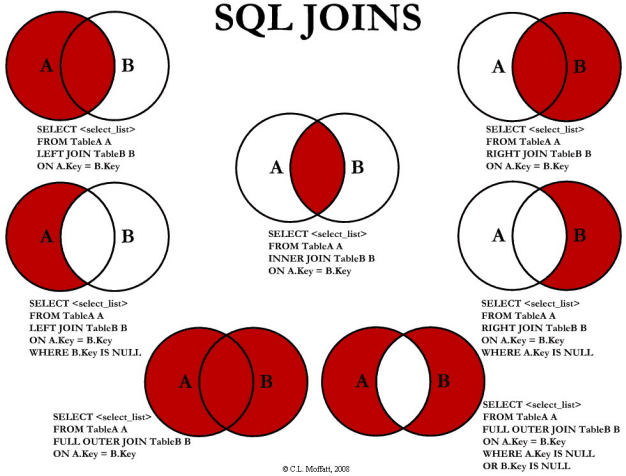
Useful Keywords for **SELECT**
DISTINCT- return unique results
BETWEEN a AND b- limit the range, the values can be numbers, text, or dates
LIKE- pattern search within the column text
IN (a, b, c) - check if the value is contained among given

Data Modification

- update specific data with the **WHERE** clause:
UPDATE table1 **SET** col1 = 1 **WHERE** col2 = 2
- insert values manually
INSERT INTO table1 (col1,col3) **VALUES** (val1,val3);
- by using the results of a query
INSERT INTO table1 (col1,col3) **SELECT** col,col2 FROM table2;

Joins

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



Python- Data Structures

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

Lists: or arrays, ordered sequences of objects, mutable

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

Tuples: like lists, but immutable

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

Dictionaries: hash tables, key-value pairs, unsorted

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

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

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

Collections Module

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

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

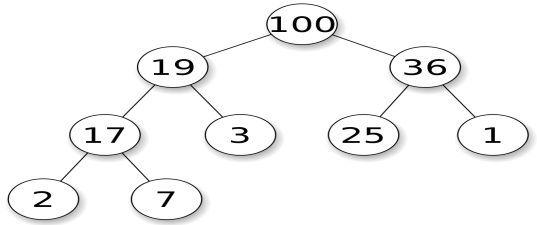
Counter: dict subclass, unordered collection where elements are stored as keys and counts stored as values

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

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/`)

MACHINE LEARNING CHEATSHEET

Summary of Machine Learning Algorithms descriptions, advantages and use cases. Inspired by the very good book and articles of *MachineLearningMastery,* with added math, and *ML Pros & Cons* of *HackingNote.* Design inspired by *The Probability Cheatsheet* of W. Chen. Written by Rémi Canard.

General

Definition

We want to learn a target function ***f*** that maps input variables ***X*** to output variable ***Y***, with an error ***e***:

$$Y = f(X) + e$$

Linear, Nonlinear

Different algorithms make different assumptions about the shape and structure of *f*, thus the need of testing several methods. Any algorithm can be either:

- **Parametric** (or **Linear**): simplify the mapping to a known linear combination form and learning its coefficients.

- **Non parametric** (or **Nonlinear**): free to learn any functional form from the training data, while maintaining some ability to generalize.

Linear algorithms are usually simpler, faster and requires less data, while Nonlinear can be are more flexible, more powerful and more performant.

Supervised, Unsupervised

Supervised learning methods learn to predict Y from X given that the data is labeled.

Unsupervised learning methods learn to find the inherent structure of the unlabeled data.

Bias-Variance trade-off

In supervised learning, the prediction error ***e*** is composed of the **bias**, the **variance** and the **irreducible** part.

Bias refers to **simplifying assumptions** made to learn the target function easily.

Variance refers to sensitivity of the model to changes in the training data.

The **goal of parameterization** is to achieve a **low bias** (underlying pattern not too simplified) and **low variance** (not sensitive to specificities of the training data) **tradeoff**.

Underfitting, Overfitting

In statistics, ***fit*** refers to how well the target function is approximated.

Underfitting refers to poor inductive learning from training data and poor generalization.

Overfitting refers to learning the training data detail and noise which leads to poor generalization. It can be **limited** by using resampling and defining a validation dataset.

Optimization

Almost every machine learning method has an optimization algorithm at its core.

Gradient Descent

Gradient Descent is used to **find the coefficients** of f that **minimizes a cost function** (for example MSE, SSR).

Procedure:

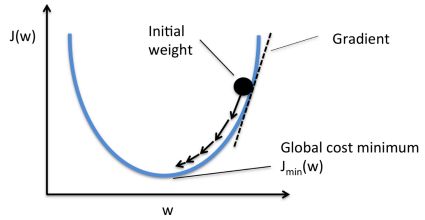
→ Initialization $\theta = 0$ (coefficients to 0 or random)

→ Calculate cost $J(\theta) = evaluate(f(coefficients))$

→ Gradient of cost $\frac{\partial}{\partial \theta j} J(\theta)$ we know the uphill direction

→ Update coeff $\theta j = \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$ we go downhill

The cost updating process is repeated until convergence (minimum found).



Batch Gradient Descend does summing/averaging of the cost over all the observations.

Stochastic Gradient Descent apply the procedure of parameter updating for each observation.

Tips:

- Change **learning rate** α ("size of jump" at each iteration)

- Plot *Cost vs Time* to assess learning rate performance

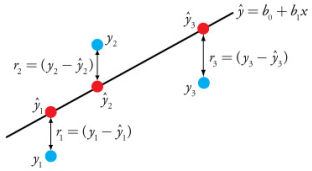
- Rescaling the input variables

- Reduce passes through training set with SGD

- Average over 10 or more updated to observe the learning trend while using SGD

Ordinary Least Squares

OLS is used to find the estimator $\hat{\beta}$ that **minimizes the sum of squared residuals:** $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 = y - X\hat{\beta}$



Using linear algebra such that we have $\hat{\beta} = (X^T X)^{-1} X^T y$

Maximum Likelihood Estimation

MLE is used to find the estimators that **minimizes the likelihood function:**

$\mathcal{L}(\theta|x) = f_\theta(x)$ density function of the data distribution

Linear Algorithms

All linear Algorithms assume a linear relationship between the input variables *X* and the output variable *Y*.

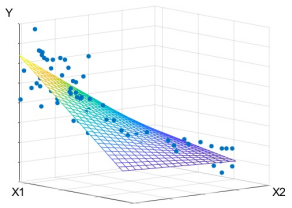
Linear Regression

Representation:

A LR model representation is a linear equation:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_i x_i$$

β_0 is usually called intercept or **bias** coefficient. The dimension of the hyperplane of the regression is its **complexity.**



Learning:

Learning a LR means estimating the coefficients from the training data. Common methods include **Gradient Descent** or **Ordinary Least Squares**.

Variations:

There are extensions of LR training called **regularization** methods, that aim to **reduce the complexity** of the model:

- **Lasso Regression**: where OLS is modified to minimize the sum of the coefficients (L1 regularization)

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

- **Ridge Regression**: where OLS is modified to minimize the squared sum of the coefficients (L2 regularization)

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}{\beta_j}^2 = RSS + \lambda\sum_{j=1}^{p}{\beta_j}^2$$

where $\lambda \geq 0$ is a tuning parameter to be determined.

Data preparation:

- Transform data for linear relationship (ex: log transform for exponential relationship)
- Remove noise such as outliers
- Rescale inputs using standardization or normalization

Advantages:

+ Good regression baseline considering simplicity
+ Lasso/Ridge can be used to avoid overfitting
+ Lasso/Ridge permit feature selection in case of collinearity

Usecase examples:

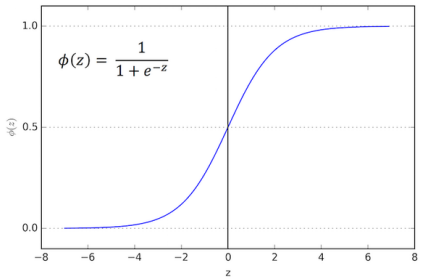
- Product sales prediction according to prices or promotions
- Call-center waiting-time prediction according to the number of complaints and the number of working agents

Logistic Regression

It is the go-to for **binary classification.**

Representation:

Logistic regression a linear method but predictions are transformed using the **logistic function** (or sigmoid):



ϕ is *S*-shaped and map real-valued number in (0,1).

The representation is an equation with binary output:

$$y = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}}{1 + e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}}$$

Which actually models the probability of default class:

$$p(X) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}}{1 + e^{\beta_0+\beta_1 x_1+\cdots+\beta_i x_i}} = p(Y = 1|X)$$

Learning:

Learning the Logistic regression coefficients is done using **maximum-likelihood estimation**, to predict values close to 1 for default class and close to 0 for the other class.

Data preparation:

- Probability transformation to binary for classification
- Remove noise such as outliers

Advantages:

+ Good classification baseline considering simplicity
+ Possibility to change cutoff for precision/recall tradeoff
+ Robust to noise/overfitting with L1/L2 regularization
+ Probability output can be used for ranking

Usecase examples:

- Customer scoring with probability of purchase
- Classification of loan defaults according to profile

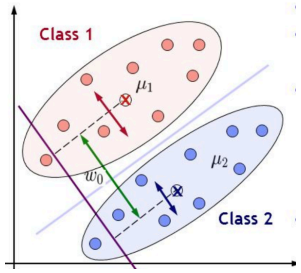
Linear Discriminant Analysis

For **multiclass classification**, LDA is the preferred linear technique.

Representation:

LDA representation consists of **statistical properties** calculated for **each class**: **means** and the **covariance matrix:**

$$\mu_k = \frac{1}{n_k}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma^2 = \frac{1}{n-K}\sum_{i=1}^{n}(x_i - \mu_k)^2$$



LDA assumes **Gaussian** data and attributes of **same σ^2**.

Predictions are made using **Bayes Theorem:**

$$P(Y = k|X = x) = \frac{P(k)\times P(x|k)}{\sum_{l=1}^{K} P(l)\times P(x|l)}$$

to obtain a discriminate function (latent variable) for each class *k*, estimating $P(x|k)$ with a Gaussian distribution:

$$D_k(x) = x \times \frac{\mu_k}{\sigma^2} - \frac{{\mu_k}^2}{2\sigma^2} + \ln(P(k))$$

The class with largest **discriminant value** is the **output class**.

Variations:

- **Quadratic DA**: Each class uses its own variance estimate
- **Regularized DA**: Regularization into the variance estimate

Data preparation:

- Review and modify univariate distributions to be Gaussian
- Standardize data to $\mu = 0, \sigma = 1$ to have same variance
- Remove noise such as outliers

Advantages:

+ Can be used for dimensionality reduction by keeping the latent variables as new variables

Usecase example:

- Prediction of customer churn

Nonlinear Algorithms

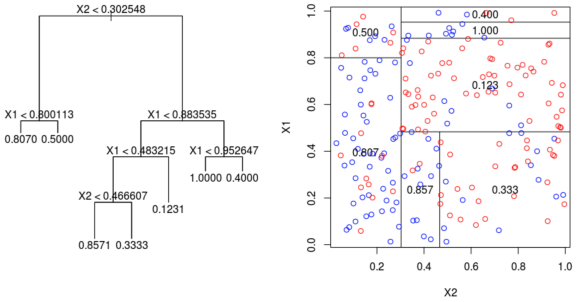
All Nonlinear Algorithms are non-parametric and more flexible. They are not sensible to outliers and do not require any shape of distribution.

Classification and Regression Trees

Also referred as CART or Decision Trees, this algorithm is the foundation of Random Forest and Boosted Trees.

Representation:

The model representation is a **binary tree**, where each **node** is an **input variable** *x* with a split point and each **leaf** contain an **output variable** *y* for prediction.



The model actually **split the input space** into (hyper) rectangles, and predictions are made according to the **area** observations *fall* into.

Learning:

Learning of a CART is done by a greedy approach called **recursive binary splitting** of the input space:

At each step, the best **predictor** X_j and the best **cutpoint** s are selected such that $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ **minimizes the cost**.

- For **regression** the cost is the **Sum of Squared Error:**

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

- For **classification** the cost function is the **Gini index:**

$$G = \sum_{i=1}^{n} p_k(1 - p_k)$$

The Gini index is **an indication of how *pure* are the leaves**, if all observations are the same type G=0 (perfect purity), while a 50-50 split for binary would be G=0.5 (worst purity).

The most common **Stopping Criterion** for splitting is a minimum of **training observations per node**.

The simplest form of pruning is **Reduced Error Pruning:** Starting at the leaves, each node is replaced with its most popular class. If the prediction accuracy is not affected, then the change is kept

Advantages:

+ Easy to interpret and no overfitting with pruning

+ Works for both regression and classification problems

+ Can take any type of variables without modifications, and do not require any data preparation

Usecase examples:

- Fraudulent transaction classification

- Predict human resource allocation in companies

Naive Bayes Classifier

Naive Bayes is a **classification** algorithm interested in selecting the **best hypothesis *h* given data *d*** assuming there is no interaction between features.

Representation:

The representation is the based on Bayes Theorem:

$$P(h|d) = \frac{P(d|h) \times P(h)}{P(d)}$$

with naïve hypothesis $P(h|d) = P(x_1|h) \times \ldots \times P(x_i|h)$

The prediction is the **Maximum A posteriori Hypothesis**:

$$MAP(h) = \max\big(P(h|d)\big) = \max\left(P(d|h) \times P(h)\right)$$

The denominator is not kept as it is only for normalization.

Learning:

Training is **fast** because only **probabilities** need to be calculated:

$$P(h) = \frac{instances_h}{all\ instances} \text{ and } P(x|h) = \frac{count(x \wedge h)}{instances_h}$$

Variations:

Gaussian Naive Bayes can extend to numerical attributes by assuming a Gaussian distribution.

Instead of $P(x|h)$ are calculated with $P(h)$ during **learning**:

$$\mu(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu(x))^2}$$

and MAP for **prediction** is calculated using Gaussian PDF

$$f(x|\mu(x), \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Data preparation:

- Change numerical inputs to categorical (binning) or near-Gaussian inputs (remove outliers, log & boxcox transform)

- Other distributions can be used instead of Gaussian

- Log-transform of the probabilities can avoid overflow

- Probabilities can be updated as data becomes available

Advantages:

+ Fast because of the calculations

+ If the naive assumptions works can converge quicker than other models. Can be used on smaller training data.

+ Good for few categories variables

Usecase examples:

- Article classification using binary word presence

- Email spam detection using a similar technique

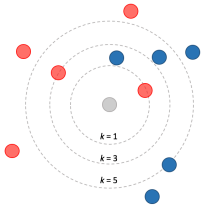
K-Nearest Neighbors

If you are similar to your neighbors, you are one of them.

Representation:

KNN uses the **entire training set**, **no training** is required.

Predictions are made by searching the ***k* similar instances**, according to a **distance**, and **summarizing the output**.



For **regression** the output can be the **mean**, while for **classification** the output can be the **most common class**.

Various distances can be used, for example:

- **Euclidean** Distance, good for **similar** type of variables

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

- **Manhattan** Distance, good for **different** type of variables

$$d(a,b) = \sum_{i=1}^{n}|a_i - b_i|$$

The **best value of *k*** must be found by **testing**, and the algorithm is **sensible** to the ***Curse of dimensionality***.

Data preparation:

- Rescale inputs using standardization or normalization

- Address missing data for distance calculations

- Dimensionality reduction or feature selection for *COD*

Advantages:

+ Effective if the training data is large

+ No learning phase

+ Robust to noisy data, no need to filter outliers

Usecase examples:

- Recommending products based on similar customers

- Anomaly detection in customer behavior

Support Vector Machines

SVM is a go-to for high performance with little tuning

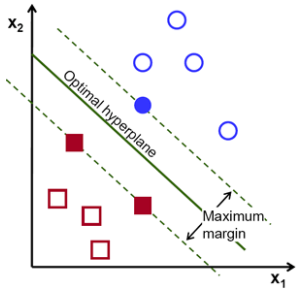
Representation:

In SVM, a **hyperplane** is selected to **separate the points** in the input variable space by their class, with the **largest margin**.

The closest datapoints (defining the margin) are called the **support vectors**.

But real **data cannot be perfectly separated**, that is why a ***C*** defines the amount of **violation** of the margin allowed.

The lower C, the more sensitive SVM is to training data.



The **prediction** function is the signed **distance** of the new input *x* to the separating hyperplane *w*:

$$f(x) = < w, x > + \rho = w^T x + \rho \quad \text{with } \rho \text{ the bias}$$

Which gives for **linear kernel**, with x_i the support vectors:

$$f(x) = \sum_{i=1}^{n} a_i \times (x \times x_i)) + \rho$$

Learning:

The hyperplane learning is done by transforming the problem using linear algebra, and minimizing:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\vec{w}.\vec{x_i} - b)\right)\right] + \lambda||\vec{w}||^2$$

Variations:

SVM is implemented using various kernels, which define the measure between new data and support vectors:

- **Linear** (dot-product): $K(x, x_i) = \sum(x \times x_i)$

- **Polynomial**: $K(x, x_i) = 1 + \sum(x \times x_i)^d$

- **Radial**: $K(x, x_i) = e^{-\gamma\sum((x - x_i)^2)}$

Data preparation:

- SVM assumes numeric inputs, may require dummy transformation of categorical features

Advantages:

+ Allow nonlinear separation with nonlinear Kernels

+ Works good in high dimensional space

+ Robust to multicollinearity and overfitting

Usecase examples:

- Face detection from images

- Target Audience Classification from tweets

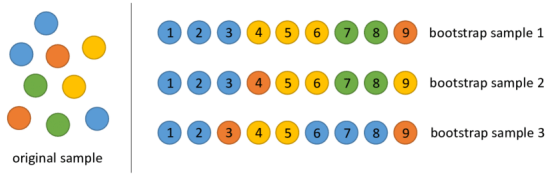
Ensemble Algorithms

Ensemble methods use multiple, simpler algorithms combined to obtain better performance.

Bagging and Random Forest

Random Forest is part of a bigger type of ensemble methods called **Bootstrap Aggregation** or **Bagging**. Bagging can **reduce the variance** of high-variance models.

It uses the **Bootstrap statistical procedure**: estimate a quantity from a sample by creating many random subsamples with replacement, and computing the mean of each subsample.



Representation:

For **bagged decision trees**, the steps would be:

- Create many subsamples of the training dataset

- Train a CART model on each sample

- Given a new dataset, calculate the average prediction

However, combining models works best if submodels are **weakly correlated** at best.

Random Forest is a tweaked version of bagged decision trees to **reduce tree correlation**.

Learning:

During learning, each **sub-tree** can only access a random **sample of features** when **selecting the split points**. The size of the feature sample at each split is a parameter *m*.

A good default is $\sqrt{p}$ for classification and $\frac{p}{3}$ for regression.

The **OOB** estimate is the performance of **each model on its Out-Of-Bag** (not selected) samples. It is a reliable estimate of test error.

Bagged method can provide **feature importance**, by calculating and averaging the **error function drop for individual variables** (depending of samples where a variable is selected or not).

Advantages:

In addition to the advantages of the CART algorithm

+ Robust to overfitting and missing variables

+ Can be parallelized for distributed computing

+ Performance as good as SVM but easier to interpret

Usecase examples:

- Predictive machine maintenance

- Optimizing line decision for credit cards

Boosting and AdaBoost

AdaBoost was the first successful boosting algorithm developed for binary classification.

Representation:

A boost classifier is of the form

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where each f_t is a **week learner correcting the errors** of the previous one.

Adaboost is commonly used with decision trees with one level (***decision stumps***).

Predictions are made using the weighted average of the weak classifiers.

Learning:

Each training set instance is initially weighted $w(x_i) = \frac{1}{n}$

One *decision stump* is prepared using the weighted samples, and a **misclassification rate** is calculated:

$$\epsilon = \frac{\sum_{i=1}^{n}(w_i \times p_{error\ i})}{\sum_{i=1}^{n} w}$$

Which is the weighted sum of the misclassification rates, where w is the training instance i weight and $p_{error\ i}$ its prediction error (1 or 0).

A **stage value** is computed from the misclassification rate:

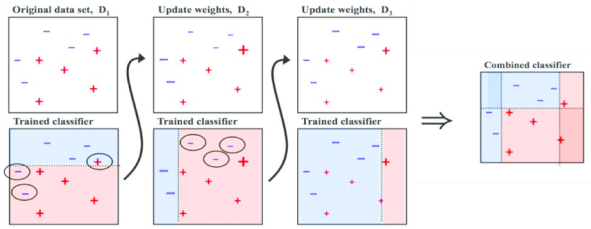
$$stage = \ln\left(\frac{1-\epsilon}{\epsilon}\right)$$

This stage value is used to **update the instances weights**:

$$w = w \times e^{stage \times \epsilon}$$

The **incorrectly** predicted instance are given **more** weight.

Weak models are added sequentially using the training weights, until no improvement can be made or the number of rounds has been attained.



Data preparation:

- Outliers should be removed for AdaBoost

Advantages:

+ High performance with no tuning (only number of rounds)

Interesting Resources

Machine Learning Mastery website

> https://machinelearningmastery.com/

Scikit-learn website, for python implementation

> http://scikit-learn.org/

W.Chen probability cheatsheet

> https://github.com/wzchen/probability_cheatsheet

HackingNote, for interesting, condensed insights

> https://www.hackingnote.com/

Seattle Data Guy blog, for business oriented articles

> https://www.theseattledataguy.com/

Explained visually, making hard ideas intuitive

> http://setosa.io/ev/

This Machine Learning Cheatsheet

> https://github.com/remicnrd/ml_cheatsheet

Super VIP Cheatsheet: Machine Learning

Afshine Amidi and Shervine Amidi

October 6, 2018

Contents

1 Supervised Learning

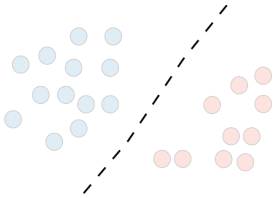
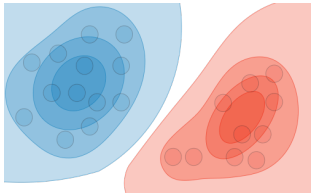
1.1 Introduction to Supervised Learning

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

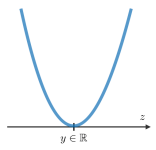
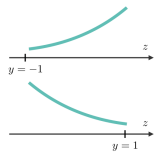
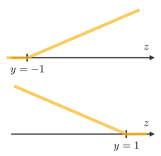
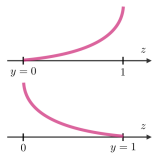
❒ **Type of model** – The different models are summed up in the table below:

	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

1.2 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:

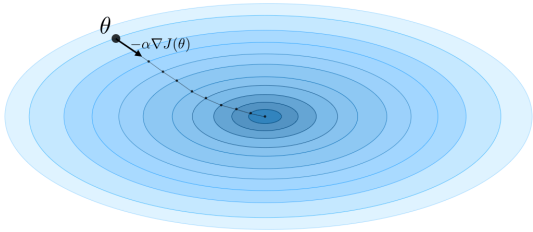
Least squared	Logistic	Hinge	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

❒ **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:

$$\boxed{J(\theta)=\sum_{i=1}^{m}L(h_\theta(x^{(i)}),y^{(i)})}$$

❒ **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:

$$\boxed{\theta\longleftarrow\theta-\alpha\nabla J(\theta)}$$



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:

$$\boxed{\theta^{\text{opt}}=\underset{\theta}{\arg\max}\ L(\theta)}$$

❒ **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:

$$\boxed{\theta\leftarrow\theta-\frac{\ell'(\theta)}{\ell''(\theta)}}$$

Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta\leftarrow\theta-\left(\nabla^2_\theta\ell(\theta)\right)^{-1}\nabla_\theta\ell(\theta)$$

1.3 Linear models

1.3.1 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:

$$\boxed{w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)}$$

1.3.2 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:

$$\boxed{\phi = p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^T x)} = g(\theta^T x)}$$

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:

$$\boxed{\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^{K} \exp(\theta_j^T x)}}$$

1.3.3 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) $\boxed{\eta = \theta^T x}$

Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

1.4 Support Vector Machines

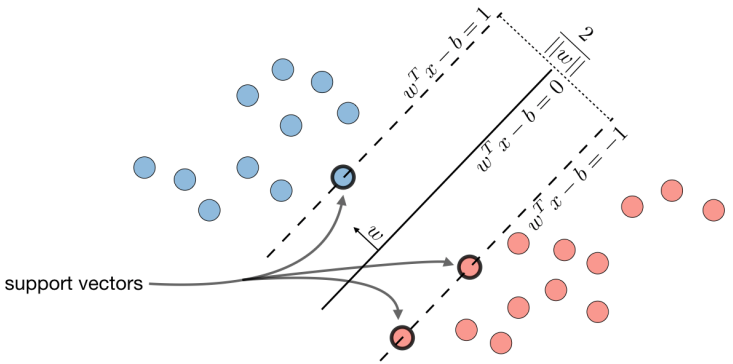
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

❒ **Optimal margin classifier** – The optimal margin classifier h is such that:

$$\boxed{h(x) = \text{sign}(w^T x - b)}$$

where $(w,b) \in \mathbb{R}^n \times \mathbb{R}$ is the solution of the following optimization problem:

$$\boxed{\min \frac{1}{2}||w||^2} \quad \text{such that} \quad \boxed{y^{(i)}(w^T x^{(i)} - b) \geqslant 1}$$



Remark: the line is defined as $\boxed{w^T x - b = 0}$.

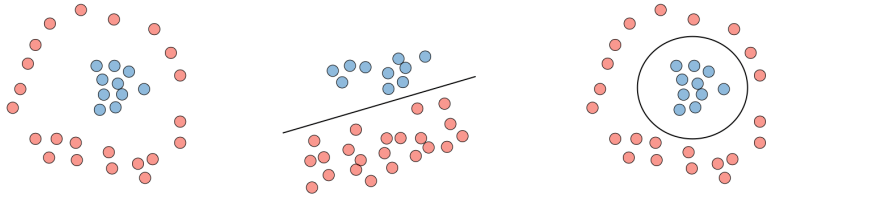
❒ **Hinge loss** – The hinge loss is used in the setting of SVMs and is defined as follows:

$$\boxed{L(z,y) = [1-yz]_+ = \max(0, 1-yz)}$$

❒ **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.

1.5 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.

1.5.1 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:

$\widehat{\phi}$	$\widehat{\mu_j}$ $(j=0,1)$	$\widehat{\Sigma}$
$\frac{1}{m}\sum_{i=1}^{m} 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}x^{(i)}}{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T$

1.5.2 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.

1.6 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.

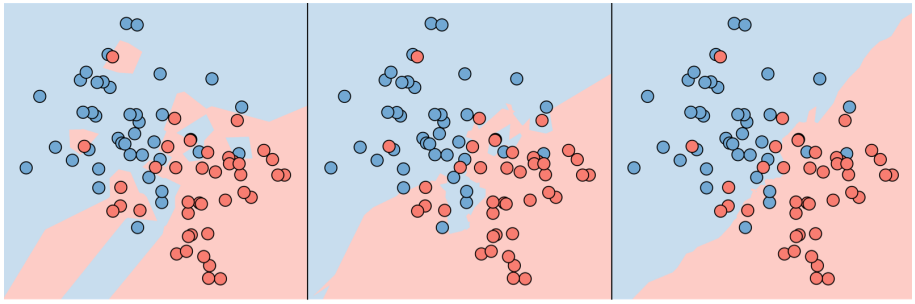
❒ **Boosting** – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

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

1.7 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.

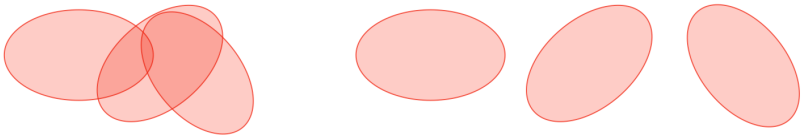


$k = 1$ $k = 3$ $k = 11$

1.8 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:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + O\left(\sqrt{\frac{d}{m}\log\left(\frac{m}{d}\right) + \frac{1}{m}\log\left(\frac{1}{\delta}\right)}\right)}$$

2 Unsupervised Learning

2.1 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:

$$\boxed{E[f(X)] \geqslant f(E[X])}$$

2.2 Clustering

2.2.1 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	$\mathcal{N}(0,I)$	$\mathcal{N}(\mu+\Lambda z,\psi)$	$\mu_j \in \mathbb{R}^n$

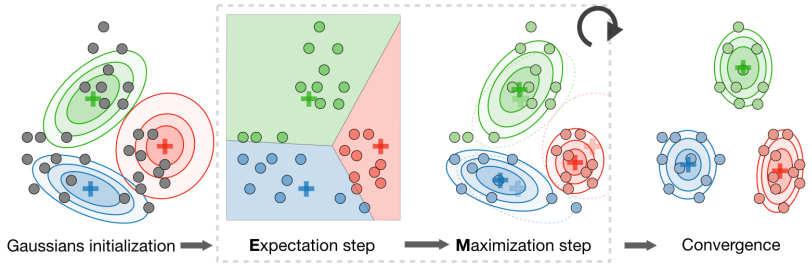
❒ **Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- 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:

$$\boxed{\theta_i = \underset{\theta}{\text{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right) dz^{(i)}}$$

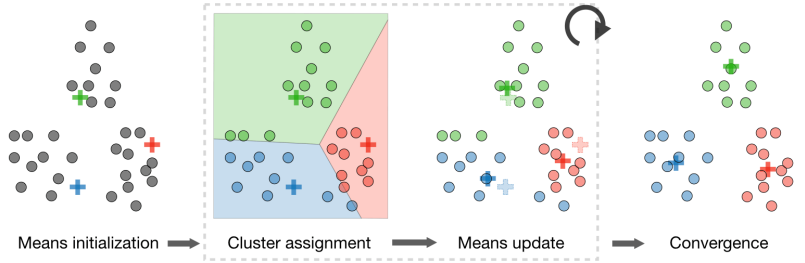


2.2.2 k-means clustering

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:

$$\boxed{J(c,\mu) = \sum_{i=1}^m ||x^{(i)} - \mu_{c^{(i)}}||^2}$$

2.2.3 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

2.2.4 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:

$$\boxed{s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N-k}{k-1}}$$

2.3 Dimension reduction

2.3.1 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:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

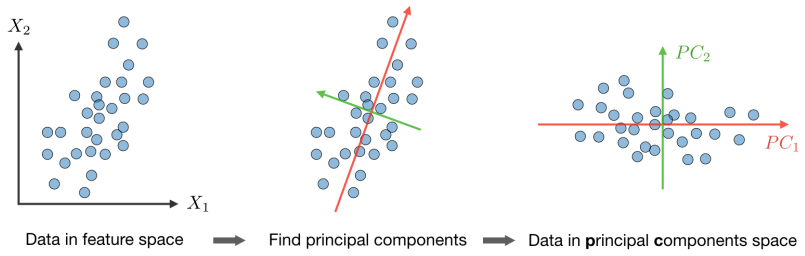
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

❒ **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.



2.3.2 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:

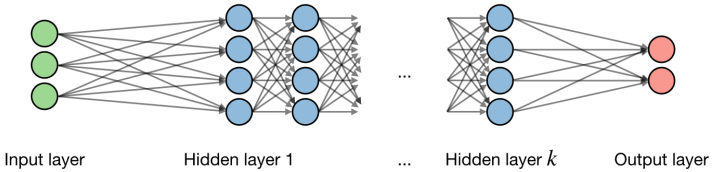
$$\boxed{W \longleftarrow W + \alpha\left(\begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)^T} + (W^T)^{-1}\right)}$$

3 Deep Learning

3.1 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:

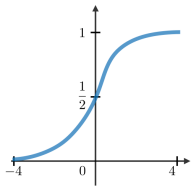
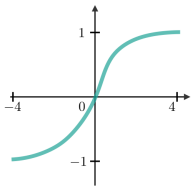
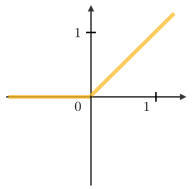
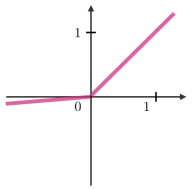


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

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
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ 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 $1-p$.

3.2 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.

3.3 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.

3.4 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:

$$\boxed{V^\pi(s) = E\Big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|s_0 = s,\pi\Big]}$$

❒ **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:

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]}$$

4 Machine Learning Tips and Tricks

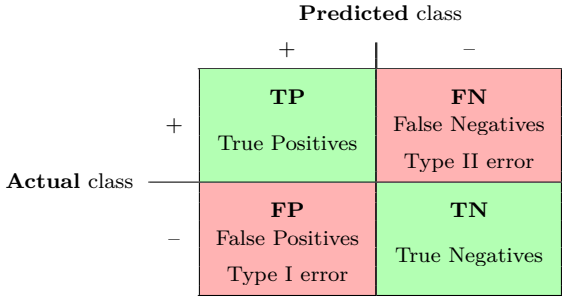
4.1 Metrics

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.

4.1.1 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:



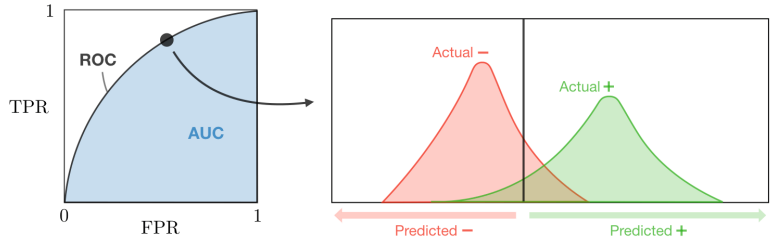
❒ **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

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:



4.1.2 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:

$$\boxed{R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}}$$

❒ **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
$\frac{\text{SS}_{\text{res}} + 2(n+1)\widehat{\sigma}^2}{m}$	$2\big[(n+2) - \log(L)\big]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

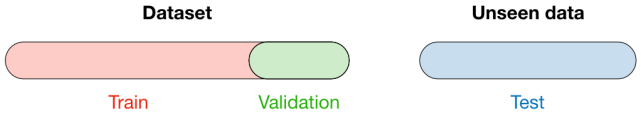
where L is the likelihood and $\widehat{\sigma}^2$ is an estimate of the variance associated with each response.

4.2 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

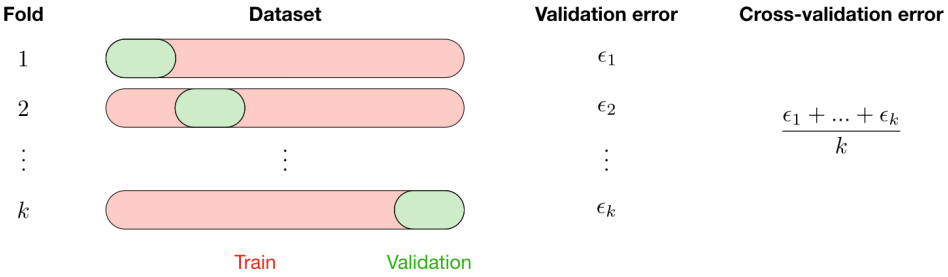
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



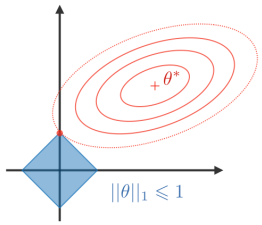
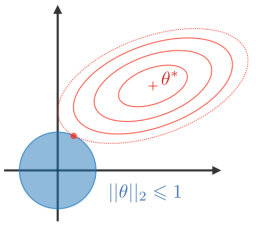
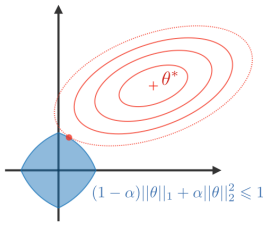
❒ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

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
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}$, $\alpha \in [0,1]$

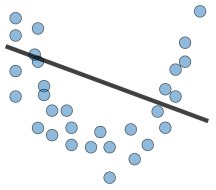
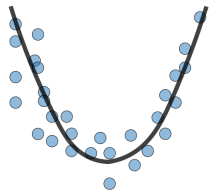
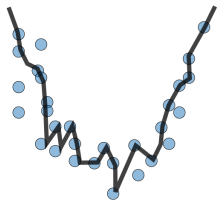
❒ **Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

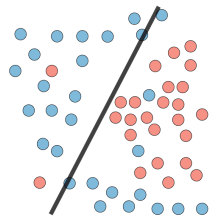
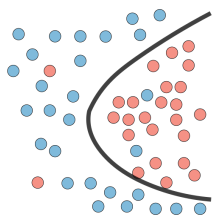
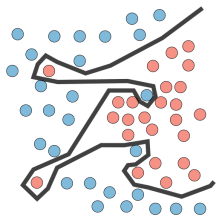
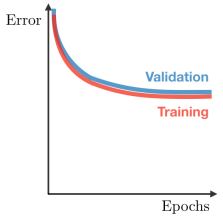
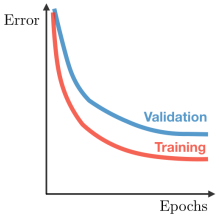
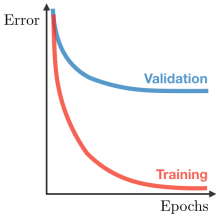
4.3 Diagnostics

❒ **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.

5 Refreshers

5.1 Probabilities and Statistics

5.1.1 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 $P(n,r) \geqslant C(n,r)$.

5.1.2 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:

$$\boxed{P(A \cap B) = P(A)P(B)}$$

5.1.3 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:

$$\boxed{P(|X - \mu| \geqslant k\sigma) \leqslant \frac{1}{k^2}}$$

5.1.4 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^2_{XY} or more commonly $\text{Cov}(X,Y)$, as follows:

$$\boxed{\text{Cov}(X,Y) \triangleq \sigma^2_{XY} = E[(X-\mu_X)(Y-\mu_Y)] = E[XY] - \mu_X\mu_Y}$$

❒ **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:

$$\boxed{\rho_{XY} = \frac{\sigma^2_{XY}}{\sigma_X\sigma_Y}}$$

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	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1-\frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

5.1.5 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:

$$\boxed{\overline{X} \underset{n\to+\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)}$$

5.2 Linear Algebra and Calculus

5.2.1 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 $diag(d_1,...,d_n)$.

5.2.2 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:

$$\boxed{xy^T = \begin{pmatrix} x_1y_1 & \cdots & x_1y_n \\ \vdots & & \vdots \\ x_my_1 & \cdots & x_my_n \end{pmatrix} \in \mathbb{R}^{m\times n}}$$

❒ **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 $\mathrm{tr}(A)$, is the sum of its diagonal entries:

$$\boxed{\mathrm{tr}(A) = \sum_{i=1}^{n} A_{i,i}}$$

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:

$$\boxed{\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} |A_{\backslash i,\backslash j}|}$$

Remark: A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

5.2.3 Matrix properties

❒ **Symmetric decomposition** – A given matrix A can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$\boxed{A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}}$$

❒ **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:

$$\boxed{A = A^T} \quad \text{and} \quad \boxed{\forall x \in \mathbb{R}^n, \quad x^T A x \geqslant 0}$$

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, $x^T Ax > 0$.

❒ **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:

$$\boxed{Az = \lambda z}$$

❒ **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:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

❒ **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:

$$\boxed{A = U\Sigma V^T}$$

5.2.4 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:

$$\boxed{\Big(\nabla_A f(A)\Big)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}}$$

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:

$$\boxed{\Big(\nabla_x^2 f(x)\Big)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}}$$

Remark: the hessian of f is only defined when f is a function that returns a scalar.

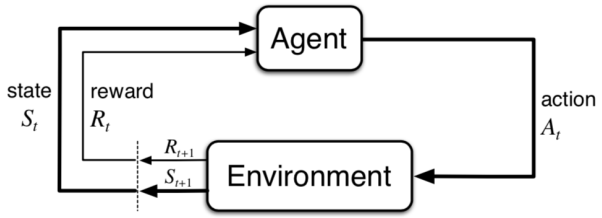
❒ **Gradient operations** – For matrices A,B,C, the following gradient properties are worth having in mind:

$$\boxed{\nabla_A \text{tr}(AB) = B^T} \qquad \boxed{\nabla_{A^T} f(A) = (\nabla_A f(A))^T}$$

$$\boxed{\nabla_A \text{tr}(ABA^TC) = CAB + C^TAB^T} \qquad \boxed{\nabla_A |A| = |A|(A^{-1})^T}$$

Reinforcement Learning Cheat Sheet

Agent-Environment Interface



The Agent at each step t receives a representation of the environment's *state*, $S_t \in S$ and it selects an action $A_t \in A(s)$. Then, as a consequence of its action the agent receives a *reward*, $R_{t+1} \in R \in \mathbb{R}$.

Policy

A *policy* is a mapping from a state to an action

$$\pi_t(s|a) \tag{1}$$

That is the probability of select an action $A_t = a$ if $S_t = s$.

Reward

The total *reward* is expressed as:

$$G_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{2}$$

Where γ is the *discount factor* and H is the *horizon*, that can be infinite.

Markov Decision Process

A **Markov Decision Process**, MPD, is a 5-tuple (S, A, P, R, γ) where:

$$\begin{aligned}
&\text{finite set of states:}\\
&s \in S\\
&\text{finite set of actions:}\\
&a \in A\\
&\text{state transition probabilities:}\\
&p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\}\\
&\text{expected reward for state-action-nexstate:}\\
&r(s',s,a) = \mathbb{E}[R_{t+1}|S_{t+1} = s', S_t = s, A_t = a]
\end{aligned} \tag{3}$$

Value Function

Value function describes *how good* is to be in a specific state s under a certain policy π. For MDP:

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{4}$$

Informally, is the expected return (expected cumulative discounted reward) when starting from s and following π

Value Iteration

We can avoid to wait until $V(s)$ has converged and instead do policy improvement and truncated policy evaluation step in one operation

Optimal

$$v_*(s) = \max_\pi v^\pi(s) \tag{5}$$

Action-Value (Q) Function

We can also denoted the expected reward for state, action pairs.

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right] \tag{6}$$

Optimal

The optimal value-action function:

$$q_*(s,a) = \max_\pi q^\pi(s,a) \tag{7}$$

Clearly, using this new notation we can redefine v^*, equation 5, using $q^*(s,a)$, equation 7:

$$v_*(s) = \max_{a \in A(s)} q_{\pi *}(s,a) \tag{8}$$

Intuitively, the above equation express the fact that the value of a state under the optimal policy **must be equal** to the expected return from the best action from that state.

Bellman Equation

An important recursive property emerges for both Value (4) and Q (6) functions if we expand them.

Value Function

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t|S_t = s\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s\right]\\
&= \underbrace{\sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)}_{\text{Sum of all probabilities } \forall \text{ possible } r}\\
&\quad \left[r + \gamma \underbrace{\mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]}_{\text{Expected reward from } s_{t+1}}\right]\\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]
\end{aligned} \tag{9}$$

Initialise $V(s) \in \mathbb{R}, \text{e.g} V(s) = 0$
$\Delta \leftarrow 0$
while $\Delta \geq \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end

Similarly, we can do the same for the Q function:

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]\\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s, A_t = a\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s'\right]\right]\\
&= \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_\pi(s')\right]
\end{aligned} \tag{10}$$

Dynamic Programming

Taking advantages of the subproblem structure of the V and Q function we can find the optimal policy by just *planning*

Policy Iteration

We can now find the optimal policy

1. Initialisation
$V(s) \in \mathbb{R}$, (e.g $V(s) = 0$) and $\pi(s) \in A$ for all $s \in S$,
$\Delta \leftarrow 0$
2. Policy Evaluation
while $\Delta \geq \theta$ *(a small positive number)* **do**
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
end
3. Policy Improvement
policy-stable $\leftarrow$ *true*
foreach $s \in S$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
  policy-stable $\leftarrow$ *old-action* $= \pi(s)$
end
if *policy-stable* return V $\approx v_*$ and $\pi \approx \pi_*$, else go to 2

Algorithm 1: Policy Iteration

Monte Carlo Methods

Monte Carlo (MC) is a *Model Free* method, It does not require complete knowledge of the environment. It is based on

averaging sample returns for each state-action pair. The following algorithm gives the basic implementation

Initialise for all $s \in S, a \in A(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $\pi(s) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
while *forever* **do**
  Choose $S_0 \in S$ and $A_0 \in A(S_0)$, all pairs have probability > 0
  Generate an episode starting at S_0, A_0 following π
  foreach *pair* s, a *appearing in the episode* **do**
    $G \leftarrow$ return following the first occurrence of s, a
    Append G to $Returns(s,a)$
    $Q(s,a) \leftarrow average(Returns(s,a))$
  end
  foreach s *in the episode* **do**
    $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\, Q(s,a)$
  end
end

Algorithm 3: Monte Carlo first-visit

For non-stationary problems, the Monte Carlo estimate for, e.g, V is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[G_t - V(S_t)\right] \tag{11}$$

Where α is the learning rate, how much we want to forget about past experiences.

Sarsa

Sarsa (State-action-reward-state-action) is a on-policy TD control. The update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

n-step Sarsa

Define the n-step Q-Return

$$q^{(n)} = R_{t+1} + \gamma Rt + 2 + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

n-step Sarsa update $Q(S,a)$ towards the n-step Q-return

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^{(n)} - Q(s_t, a_t)\right]$$

Forward View Sarsa(λ)

$$q_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

Forward-view Sarsa(λ):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[q_t^\lambda - Q(s_t, a_t)\right]$$

Initialise $Q(s,a)$ arbitrarily and
  $Q(terminal - state,) = 0$
foreach *episode* $\in$ *episodes* **do**
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  while s *is not terminal* **do**
    Take action a, observer r, s'
    Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', a') - Q(s,a)\right]$
    $s \leftarrow s'$
    $a \leftarrow a'$
  end
end

Algorithm 4: Sarsa(λ)

Temporal Difference - Q Learning

Temporal Difference (TD) methods learn directly from raw experience without a model of the environment's dynamics. TD substitutes the expected discounted reward G_t from the episode with an estimation:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1} - V(S_t)\right] \tag{12}$$

The following algorithm gives a generic implementation.

Initialise $Q(s,a)$ arbitrarily and
  $Q(terminal - state,) = 0$
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observer r, s'
    $Q(s,a) \leftarrow$
      $Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right]$
    $s \leftarrow s'$
  end
end

Algorithm 5: Q Learning

Deep Q Learning

Created by *DeepMind*, Deep Q Learning, DQL, substitutes the Q function with a deep neural network called *Q-network*. It also keep track of some observation in a *memory* in order to use them to train the network.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\underbrace{(r + \gamma \max_a Q(s', a'; \theta_{i-1})}_{\text{target}} - \underbrace{Q(s,a;\theta_i))}_{\text{prediction}}^2 \right] \tag{13}$$

Where θ are the weights of the network and $U(D)$ is the experience replay history.

Initialise replay memory D with capacity N
Initialise $Q(s,a)$ arbitrarily
foreach *episode* $\in$ *episodes* **do**
  while s *is not terminal* **do**
    With probability ϵ select a random action $a \in A(s)$
    otherwise select $a = \max_a Q(s,a;\theta)$
    Take action a, observer r, s'
    Store transition (s,a,r,s') in D
    Sample random minibatch of transitions (s_j, a_j, r_j, s'_j) from D
    Set $y_j \leftarrow$
      $\begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_a Q(s', a'; \theta) & \text{for non-terminal } s'_j \end{cases}$
    Perform gradient descent step on $(y_j - Q(s_j, a_j; \Theta))^2$
    $s \leftarrow s'$
  end
end

Algorithm 6: Deep Q Learning

Copyright © 2018 Francesco Saverio Zuppichini
https://github.com/FrancescoSaverioZuppichini/Reinforcement-Learning-Cheat-Sheet

UDACITY

REINFORCEMENT LEARNING

1. The Problem

S_t	state at time t
A_t	action at time t
R_t	reward at time t
γ	discount rate (where $0 \leq \gamma \leq 1$)
G_t	discounted return at time t ($\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$)
$\mathcal{S}$	set of all nonterminal states
$\mathcal{S}^+$	set of all states (including terminal states)
$\mathcal{A}$	set of all actions
$\mathcal{A}(s)$	set of all actions available in state s
$\mathcal{R}$	set of all rewards
$p(s', r\|s, a)$	probability of next state s' and reward r, given current state s and current action a ($\mathbb{P}(S_{t+1} = s', R_{t+1} = r\|S_t = s, A_t = a)$)

2. The Solution

π	policy *if deterministic*: $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$ *if stochastic*: $\pi(a\|s) = \mathbb{P}(A_t = a\|S_t = s)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
v_π	state-value function for policy π ($v_\pi(s) \doteq \mathbb{E}[G_t\|S_t = s]$ for all $s \in \mathcal{S}$)
q_π	action-value function for policy π ($q_\pi(s, a) \doteq \mathbb{E}[G_t\|S_t = s, A_t = a]$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
v_*	optimal state-value function ($v_*(s) \doteq \max_\pi v_\pi(s)$ for all $s \in \mathcal{S}$)
q_*	optimal action-value function ($q_*(s, a) \doteq \max_\pi q_\pi(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)

3. Bellman Equations

3.1. Bellman Expectation Equations.

$$\boxed{v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma v_\pi(s'))}$$

$$\boxed{q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(a'|s') q_\pi(s', a'))}$$

3.2. Bellman Optimality Equations.

$$\boxed{v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma v_*(s'))}$$

$$\boxed{q_*(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma \max_{a' \in \mathcal{A}(s')} q_*(s', a'))}$$

3.3. Useful Formulas for Deriving the Bellman Equations.

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) q_\pi(s, a)$$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_*(s, a)$$

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma v_\pi(s'))$$

$$q_*(s, a) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma v_*(s'))$$

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t|S_t=s,A_t=a] && (1)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} \mathbb{P}(S_{t+1}=s',R_{t+1}=r|S_t=s,A_t=a)\mathbb{E}_\pi[G_t|S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r] && (2)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\mathbb{E}_\pi[G_t|S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r] && (3)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\mathbb{E}_\pi[G_t|S_{t+1}=s',R_{t+1}=r] && (4)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)\mathbb{E}_\pi[R_{t+1}+\gamma G_{t+1}|S_{t+1}=s',R_{t+1}=r] && (5)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)(r+\gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']) && (6)\\
&= \sum_{s'\in\mathcal{S},r\in\mathcal{R}} p(s',r|s,a)(r+\gamma v_\pi(s')) && (7)
\end{aligned}
$$

The reasoning for the above is as follows:

- (1) by definition ($q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t=s,A_t=a]$)
- (2) Law of Total Expectation
- (3) by definition ($p(s',r|s,a) \doteq \mathbb{P}(S_{t+1}=s',R_{t+1}=r|S_t=s,A_t=a)$)
- (4) $\mathbb{E}_\pi[G_t|S_t=s,A_t=a,S_{t+1}=s',R_{t+1}=r] = \mathbb{E}_\pi[G_t|S_{t+1}=s',R_{t+1}=r]$
- (5) $G_t = R_{t+1}+\gamma G_{t+1}$
- (6) Linearity of Expectation
- (7) $v_\pi(s') = \mathbb{E}_\pi[G_{t+1}|S_{t+1}=s']$

Algorithm 1: Policy Evaluation

Input: MDP, policy π, small positive number θ
Output: $V \approx v_\pi$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
repeat
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;
return V

Algorithm 2: Estimation of Action Values

Input: MDP, state-value function V
Output: action-value function Q
for $s \in \mathcal{S}$ **do**
  for $a \in \mathcal{A}(s)$ **do**
    $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
end
return Q

Algorithm 3: Policy Improvement

Input: MDP, value function V
Output: policy π'
for $s \in \mathcal{S}$ **do**
  for $a \in \mathcal{A}(s)$ **do**
    $Q(s,a) \leftarrow \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
  $\pi'(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} Q(s, a)$
end
return π'

Algorithm 4: Policy Iteration

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize π arbitrarily (e.g., $\pi(a|s) = \frac{1}{|\mathcal{A}(s)|}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
policy-stable $\leftarrow$ *false*
repeat
  $V \leftarrow$ **Policy_Evaluation**(MDP, π, θ)
  $\pi' \leftarrow$ **Policy_Improvement**(MDP, V)
  if $\pi = \pi'$ **then**
    policy-stable $\leftarrow$ *true*
  end
  $\pi \leftarrow \pi'$
until *policy-stable* $=$ *true*;
return π

Algorithm 5: Truncated Policy Evaluation

Input: MDP, policy π, value function V, positive integer *max_iterations*
Output: $V \approx v_\pi$ (if *max_iterations* is large enough)
counter $\leftarrow 0$
while *counter* $<$ *max_iterations* **do**
  for $s \in \mathcal{S}$ **do**
    $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
  end
  counter $\leftarrow$ *counter* $+ 1$
end
return V

Algorithm 6: Truncated Policy Iteration

Input: MDP, positive integer $max\_iterations$, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
Initialize π arbitrarily (e.g., $\pi(a|s) = \frac{1}{|\mathcal{A}(s)|}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
repeat
  $\pi \leftarrow$ **Policy_Improvement**(MDP, V)
  $V_{old} \leftarrow V$
  $V \leftarrow$ **Truncated_Policy_Evaluation**(MDP, π, V, $max\_iterations$)
until $\max_{s \in \mathcal{S}} |V(s) - V_{old}(s)| < \theta$;
return π

Algorithm 7: Value Iteration

Input: MDP, small positive number θ
Output: policy $\pi \approx \pi_*$
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
repeat
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)(r + \gamma V(s'))$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end
until $\Delta < \theta$;
$\pi \leftarrow$ **Policy_Improvement**(MDP, V)
return π

Algorithm 8: First-Visit MC Prediction (*for state values*)

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$ if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize $returns\_sum(s) = 0$ for all $s \in \mathcal{S}$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T - 1$ **do**
    if S_t *is a first visit (with return* G_t*)* **then**
      $N(S_t) \leftarrow N(S_t) + 1$
      $returns\_sum(S_t) \leftarrow returns\_sum(S_t) + G_t$
  end
end
$V(s) \leftarrow returns\_sum(s)/N(s)$ for all $s \in \mathcal{S}$
return V

Algorithm 9: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer *num_episodes*
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize $N(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $returns\_sum(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T - 1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$
  end
end
$Q(s, a) \leftarrow returns\_sum(s, a)/N(s, a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
return Q

Algorithm 10: First-Visit GLIE MC Control

Input: positive integer *num_episodes*, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon$-greedy(Q)
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
  end
end
return π

Algorithm 11: First-Visit Constant-α (GLIE) MC Control

Input: positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon$-greedy(Q)
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$
  end
end
return π

Algorithm 12: TD(0)

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$ if *num_episodes* is large enough)
Initialize V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy π
    Take action A_t and observe R_{t+1}, S_{t+1}
    $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return V

Algorithm 13: Sarsa

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  Choose action A_0 using policy derived from Q (e.g., ϵ-greedy)
  $t \leftarrow 0$
  repeat
    Take action A_t and observe R_{t+1}, S_{t+1}
    Choose action A_{t+1} using policy derived from Q (e.g., ϵ-greedy)
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Algorithm 15: Expected Sarsa

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
    Take action A_t and observe R_{t+1}, S_{t+1}
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Google Data Engineering Cheatsheet

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

Last Updated August 4, 2018

What is Data Engineering?

Data engineering enables data-driven decision making by collecting, transforming, and visualizing data. A data engineer designs, builds, maintains, and troubleshoots data processing systems with a particular emphasis on the security, reliability, fault-tolerance, scalability, fidelity, and efficiency of such systems.

A data engineer also analyzes data to gain insight into business outcomes, builds statistical models to support decision-making, and creates machine learning models to automate and simplify key business processes.

Key Points

- Build/maintain data structures and databases
- Design data processing systems
- Analyze data and enable machine learning
- Design for reliability
- Visualize data and advocate policy
- Model business processes for analysis
- Design for security and compliance

Google Compute Platform (GCP)

GCP is a collection of Google computing resources, which are offered via *services*. Data engineering services include Compute, Storage, Big Data, and Machine Learning.

The 4 ways to interact with GCP include the console, command-line-interface (CLI), API, and mobile app.

The GCP resource hierarchy is organized as follows. All resources (VMs, storage buckets, etc) are organized into **projects**. These projects *may* be organized into **folders**, which can contain other folders. All folders and projects; can be brought together under an organization node. Project folders and organization nodes are where policies can be defined. Policies are inherited downstream and dictate who can access what resources. Every resource must belong to a project and every must have a billing account associated with it.

Advantages: Performance (fast solutions), Pricing (sub-hour billing, sustained use discounts, custom machine types), PaaS Solutions, Robust Infrastructure

Hadoop Overview

Hadoop

Data can no longer fit in memory on one machine (monolithic), so a new way of computing was devised using many computers to process the 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 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; the rest are data nodes. The **master node** manages the overall file system by storing the directory structure and metadata of the files. The **data nodes** physically store the data. Large files are broken up/distributed across multiple machines, which are replicated across 3 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.

Hadoop Ecosystem

An entire ecosystem of tools have emerged around Hadoop, which are based on interacting with HDFS.

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 backends (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)

IAM

Identity Access Management (IAM)

Access management service to manage different members of the platform- who has what access for which resource.

Each member has roles and permissions to allow them access to perform their duties on the platform. 3 member types: Google account (single person, gmail account), service account (non-person, application), and Google Group (multiple people). Roles are a set of specific permissions for members. Cannot assign permissions to user directly, must grant roles.

If you grant a member access on a higher hierarchy level, that member will have access to all levels below that hierarchy level as well. You cannot be restricted a lower level. The policy is a union of assigned and inherited policies.

Primitive Roles: Owner (full access to resources, manage roles), Editor (edit access to resources, change or add), Viewer (read access to resources)
Predefined Roles: finer-grained access control than primitive roles, predefined by Google Cloud
Custom Roles

Best Practice: use predefined roles when they exist (over primitive). Follow the principle of least privileged favors.

Stackdriver

GCP's monitoring, logging, and diagnostics solution. Provides insights to health, performance, and availability of applications.
Main Functions

- **Debugger**: inspect state of app in real time without stopping/slowing down e.g. code behavior
- **Error Reporting**: counts, analyzes, aggregates crashes in cloud services
- **Monitoring**: overview of performance, uptime and heath of cloud services (metrics, events, metadata)
- **Alerting**: create policies to notify you when health and uptime check results exceed a certain limit
- **Tracing**: tracks how requests propagate through applications/receive near real-time performance results, latency reports of VMs
- **Logging**: store, search, monitor and analyze log data and events from GCP

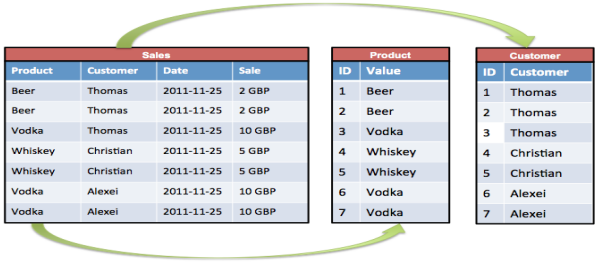
Key Concepts

OLAP vs. OLTP

Online Analytical Processing (OLAP): primary objective is data analysis. It is an online analysis and data retrieving process, characterized by a large volume of data and complex queries, uses data warehouses.
Online Transaction Processing (OLTP): primary objective is data processing, manages database modification, characterized by large numbers of short online transactions, simple queries, and traditional DBMS.

Row vs. Columnar Database

Row Format: stores data by row
Column Format: stores data tables by column rather than by row, which is suitable for analytical query processing and data warehouses



Sales

Product	Customer	Date	Sale
Beer	Thomas	2011-11-25	2 GBP
Beer	Thomas	2011-11-25	2 GBP
Vodka	Thomas	2011-11-25	10 GBP
Whiskey	Christian	2011-11-25	5 GBP
Whiskey	Christian	2011-11-25	5 GBP
Vodka	Alexei	2011-11-25	10 GBP
Vodka	Alexei	2011-11-25	10 GBP

Product

ID	Value
1	Beer
2	Beer
3	Vodka
4	Whiskey
5	Whiskey
6	Vodka
7	Vodka

Customer

ID	Customer
1	Thomas
2	Thomas
3	Thomas
4	Christian
5	Christian
6	Alexei
7	Alexei

IaaS, Paas, SaaS

IaaS: gives you the infrastructure pieces (VMs) but you have to maintain/join together the different infrastructure pieces for your application to work. Most flexible option.
PaaS: gives you all the infrastructure pieces already joined so you just have to deploy source code on the platform for your application to work. PaaS solutions are managed services/no-ops (highly available/reliable) and serverless/autoscaling (elastic). Less flexible than IaaS

Fully Managed, Hotspotting

Compute Choices

Google App Engine

Flexible, serverless platform for building highly available applications. Ideal when you want to focus on writing and developing code and do not want to manage servers, clusters, or infrastructures.
Use Cases: web sites, mobile app and gaming backends, RESTful APIs, IoT apps.

Google Kubernetes (Container) Engine

Logical infrastructure powered by Kubernetes, an open-source container orchestration system. Ideal for managing containers in production, increase velocity and operatability, and don't have OS dependencies.
Use Cases: containerized workloads, cloud-native distributed systems, hybrid applications.

Google Compute Engine (IaaS)

Virtual Machines (VMs) running in Google's global data center. Ideal for when you need complete control over your infrastructure and direct access to high-performance hardward or need OS-level changes.
Use Cases: any workload requiring a specific OS or OS configuration, currently deployed and on-premises software that you want to run in the cloud.

Summary: AppEngine is the PaaS option- serverless and ops free. ComputeEngine is the IaaS option- fully controllable down to OS level. Kubernetes Engine is in the middle- clusters of machines running Kuberenetes and hosting containers.

Additional Notes

You can also mix and match multiple compute options.
Preemptible Instances: instances that run at a much lower price but may be terminated at any time, self-terminate after 24 hours. ideal for interruptible workloads
Snapshots: used for backups of disks
Images: VM OS (Ubuntu, CentOS)

Storage

Persistent Disk Fully-managed block storage (SSDs) that is suitable for VMs/containers. Good for snapshots of data backups/sharing read-only data across VMs.

Cloud Storage Infinitely scalable, fully-managed and highly reliable object/blob storage. Good for data blobs: images, pictures, videos. Cannot query by content.

To use Cloud Storage, you create buckets to store data and the location can be specified. Bucket names are globally unique. There a 4 storage classes:

- **Multi-Regional**: frequent access from anywhere in the world. Use for "hot data"
- **Regional**: high local performance for region
- **Nearline**: storage for data accessed less than once a month (archival)
- **Coldline**: less than once a year (archival)

CloudSQL and Cloud Spanner- Relational DBs

Cloud SQL

Fully-managed relational database service (supports MySQL/PostgreSQL). Use for relational data: tables, rows and columns, and super structured data. SQL compatible and can update fields. Not scalable (small storage- GBs). Good for web frameworks and OLTP workloads (not OLAP). Can use **Cloud Storage Transfer Service** or **Transfer Appliance** to data into Cloud Storage (from AWS, local, another bucket). Use gsutil if copying files over from on-premise.

Cloud Spanner

Google-proprietary offering, more advanced than Cloud SQL. Mission-critical, relational database. Supports horizontal scaling. Combines benefits of of relational and non-relational databases.

- **Ideal**: relational, structured, and semi-structured data that requires high availability, strong consistency, and transactional reads and writes
- **Avoid**: data is not relational or structured, want an open source RDBMS, strong consistency and high availability is unnecessary

Cloud Spanner Data Model

A database can contain 1+ tables. Tables look like relational database tables. Data is strongly typed: must define a schema for each database and that schema must specify the data types of each column of each table.
Parent-Child Relationships: Can optionally define relationships between tables to physically co-locate their rows for efficient retrieval (data locality: physically storing 1+ rows of a table with a row from another table.

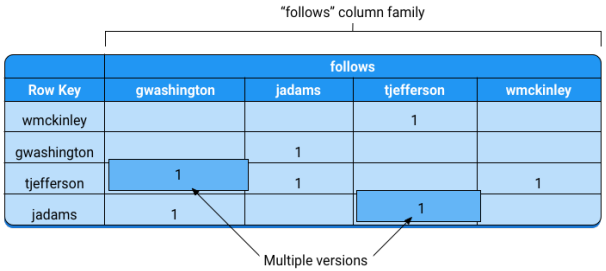
BigTable

Columnar database ideal applications that need high throughput, low latencies, and scalability (IoT, user analytics, time-series data, graph data) for non-structured key/value data (each value is < 10 MB). A single value in each row is indexed and this value is known as the row key. Does not support SQL queries. Zonal service. Not good for data less than 1TB of data or items greater than 10MB. Ideal at handling large amounts of data (TB/PB) for long periods of time.

Data Model

4-Dimensional: row key, column family (table name), column name, timestamp.

- Row key uniquely identifies an entity and columns contain individual values for each row.
- Similar columns are grouped into column families.
- Each column is identified by a combination of the column family and a column qualifier, which is a unique name within the column family.



Load Balancing Automatically manages splitting, merging, and rebalancing. The master process balances workload/data volume within clusters. The master splits busier/larger tablets in half and merges less-accessed/smaller tablets together, redistributing them between nodes.
Best write performance can be achieved by using row keys that do not follow a predictable order and grouping related rows so they are adjacent to one another, which results in more efficient multiple row reads at the same time.

Security Security can be managed at the project and instance level. Does not support table-level, row-level, column-level, or cell-level security restrictions.

Other Storage Options

- Need SQL for OLTP: CloudSQL/Cloud Spanner.
- Need interactive querying for OLAP: BigQuery.
- Need to store blobs larger than 10 MB: Cloud Storage.
- Need to store structured objects in a document database, with ACID and SQL-like queries: Cloud Datastore.

BigTable Part II

Designing Your Schema

Designing a Bigtable schema is different than designing a schema for a RDMS. Important considerations:

- Each table has only one index, the row key (4 KB)
- Rows are **sorted lexicographically by row key**.
- All operations are **atomic** (ACID) at row level.
- Both reads and writes should be distributed evenly
- Try to keep all info for an entity in a single row
- Related entities should be stored in adjacent rows
- Try to store **10 MB in a single cell** (max 100 MB) and **100 MB in a single row** (256 MB)
- Supports max of 1,000 tables in each instance.
- Choose row keys that don't follow predictable order
- Can use up to around 100 column families
- Column Qualifiers: can create as many as you need in each row, but should avoid splitting data across more column qualifiers than necessary (16 KB)
- **Tables are sparse**. Empty columns don't take up any space. Can create large number of columns, even if most columns are empty in most rows.
- Field promotion (shift a column as part of the row key) and salting (remainder of division of hash of timestamp plus row key) are ways to help design row keys.

For time-series data, use tall/narrow tables.
Denormalize- prefer multiple tall and narrow tables

BigQuery

Scalable, fully-managed Data Warehouse with extremely fast SQL queries. Allows querying for massive volumes of data at fast speeds. Good for OLAP workloads (petabyte-scale), Big Data exploration and processing, and reporting via Business Intelligence (BI) tools. Supports SQL querying for non-relational data. Relatively cheap to store, but costly for querying/processing. Good for analyzing historical data.

Data Model

Data tables are organized into units called datasets, which are sets of tables and views. A table must belong to dataset and a datatset must belong to a porject. Tables contain records with rows and columns (fields).
You can load data into BigQuery via two options: batch loading (free) and streaming (costly).

Security

BigQuery uses IAM to manage access to resources. The three types of resources in BigQuery are organizations, projects, and datasets. Security can be applied at the project and dataset level, but not table or view level.

Views

A view is a virtual table defined by a SQL query. When you create a view, you query it in the same way you query a table. **Authorized views allow you to share query results with particular users/groups without giving them access to underlying data.** When a user queries the view, the query results contain data only from the tables and fields specified in the query that defines the view.

Billing

Billing is based on **storage** (amount of data stored), **querying** (amount of data/number of bytes processed by query), and **streaming inserts**. Storage options are active and long-term (modified or not past 90 days). Query options are on-demand and flat-rate.
Query costs are based on how much data you read/process, so if you only read a section of a table (partition), your costs will be reduced. Any charges occurred are billed to the attached billing account. Exporting/importing/copying data is free.

BigQuery Part II

Partitioned tables

Special tables that are divided into partitions based on a column or partition key. Data is stored on different directories and specific queries will only run on slices of data, which improves query performance and reduces costs. Note that the partitions will not be of the same size. BigQuery automatically does this.
Each partitioned table can have up to 2,500 partitions (2500 days or a few years). The daily limit is 2,000 partition updates per table, per day. The rate limit: 50 partition updates every 10 seconds.

Two types of partitioned tables:

- **Ingestion Time**: Tables partitioned based on the data's ingestion (load) date or arrival date. Each partitioned table will have pseudocolumn _PARTITIONTIME, or time data was loaded into table. Pseudocolumns are reserved for the table and cannot be used by the user.
- **Partitioned Tables**: Tables that are partitioned based on a TIMESTAMP or DATE column.

Windowing: window functions increase the efficiency and reduce the complexity of queries that analyze partitions (windows) of a dataset by providing complex operations without the need for many intermediate calculations. They reduce the need for intermediate tables to store temporary data

Bucketing

Like partitioning, but each split/partition should be the same size and is based on the hash function of a column. Each bucket is a separate file, which makes for more efficient sampling and joining data.

Querying

After loading data into BigQuery, you can query using Standard SQL (preferred) or Legacy SQL (old). Query jobs are actions executed asynchronously to load, export, query, or copy data. Results can be saved to permanent (store) or temporary (cache) tables. 2 types of queries:

- **Interactive**: query is executed immediately, counts toward daily/concurrent usage (default)
- **Batch**: batches of queries are queued and the query starts when idle resources are available, only counts for daily and switches to interactive if idle for 24 hours

Wildcard Tables

Used if you want to union all similar tables with similar names. '*' (e.g. project.dataset.Table*)

Optimizing BigQuery

Controlling Costs

- Avoid SELECT * (full scan), select only columns needed (SELECT * EXCEPT)
- Sample data using preview options for free
- Preview queries to estimate costs (dryrun)
- Use max bytes billed to limit query costs
- Don't use LIMIT clause to limit costs (still full scan)
- Monitor costs using dashboards and audit logs
- Partition data by date
- Break query results into stages
- Use default table expiration to delete unneeded data
- Use streaming inserts wisely
- Set hard limit on bytes (members) processed per day

Query Performance

Generally, queries that do less work perform better.

Input Data/Data Sources

- Avoid SELECT *
- Prune partitioned queries (for time-partitioned table, use PARTITIONTIME pseudo column to filter partitions)
- Denormalize data (use nested and repeated fields)
- Use external data sources appropriately
- Avoid excessive wildcard tables

SQL Anti-Patterns

- Avoid self-joins., use window functions (perform calculations across many table rows related to current row)
- Partition/Skew: avoid unequally sized partitions, or when a value occurs more often than any other value
- Cross-Join: avoid joins that generate more outputs than inputs (pre-aggregate data or use window function)
Update/Insert Single Row/Column: avoid point-specific DML, instead batch updates and inserts

Managing Query Outputs

- Avoid repeated joins and using the same subqueries
- Writing large sets has performance/cost impacts. Use filters or LIMIT clause. 128MB limit for cached results
- Use LIMIT clause for large sorts (Resources Exceeded)

Optimizing Query Computation

- Avoid repeatedly transforming data via SQL queries
- Avoid JavaScript user-defined functions
- Use approximate aggregation functions (approx count)
- Order query operations to maximize performance. Use ORDER BY only in outermost query, push complex operations to end of the query.
- For queries that join data from multiple tables, optimize join patterns. Start with the largest table.

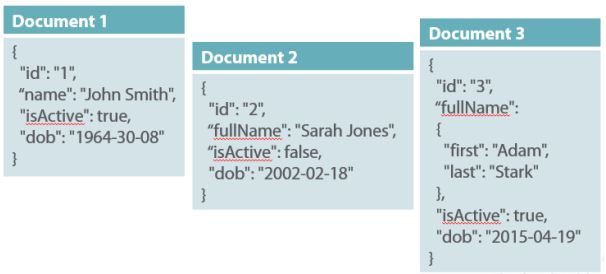
DataStore

NoSQL document database that automatically handles sharding and replication (highly available, scalable and durable). Supports ACID transactions, SQL-like queries. Query execution depends on size of returned result, not size of dataset. Ideal for "needle in a haystack"operation and applications that rely on highly available structured data at scale

Data Model

Data objects in Datastore are known as entities. An entity has one or more named properties, each of which can have one or more values. Each entity in has a key that uniquely identifies it. You can fetch an individual entity using the entity's key, or query one or more entities based on the entities' keys or property values.

Ideal for highly structured data at scale: product catalogs, customer experience based on users past activities/preferences, game states. Don't use if you need extremely low latency or analytics (complex joins, etc).



DataProc

Fully-managed cloud service for running Spark and Hadoop clusters. Provides access to Hadoop cluster on GCP and Hadoop-ecosystem tools (Pig, Hive, and Spark). Can be used to implement ETL warehouse solution.

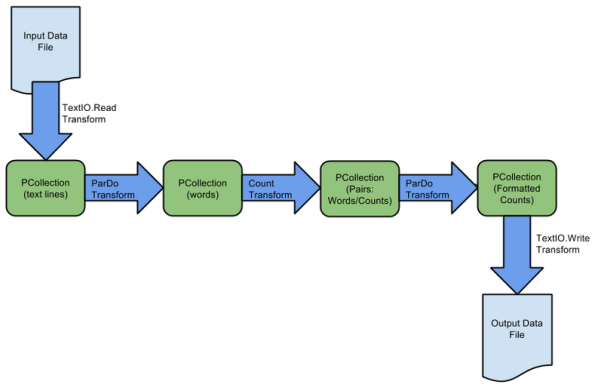
Preferred if migrating existing on-premise Hadoop or Spark infrastructure to GCP without redevelopment effort. Dataflow is preferred for a new development.

DataFlow

Managed service for developing and executing data processing patterns (ETL) (based on Apache Beam) for **streaming** and **batch** data. Preferred for new Hadoop or Spark infrastructure development. Usually site between front-end and back-end storage solutions.

Concepts

Pipeline: encapsulates series of computations that accepts input data from external sources, transforms data to provide some useful intelligence, and produce output
PCollections: abstraction that represents a potentially distributed, multi-element data set, that acts as the pipeline's data. PCollection objects represent input, intermediate, and output data. The edges of the pipeline.
Transforms: operations in pipeline. A transform takes a PCollection(s) as input, performs an operation that you specify on each element in that collection, and produces a new output PCollection. Uses the "what/where/when/how"model. Nodes in the pipeline. Composite transforms are multiple transforms: combining, mapping, shuffling, reducing, or statistical analysis.
Pipeline I/O: the source/sink, where the data flows in and out. Supports read and write transforms for a number of common data storage types, as well as custom.



Windowing

Windowing a PCollection divides the elements into windows based on the associated event time for each element. Especially useful for PCollections with unbounded size, since it allows operating on a sub-group (mini-batches).

Triggers

Allows specifying a trigger to control when (in processing time) results for the given window can be produced. If unspecified, the default behavior is to trigger first when the watermark passes the end of the window, and then trigger again every time there is late arriving data.

Pub/Sub

Asynchronous messaging service that decouples senders and receivers. Allows for secure and highly available communication between independently written applications. A publisher app creates and sends messages to a *topic*. Subscriber applications create a subscription to a topic to receive messages from it. Communication can be one-to-many (fan-out), many-to-one (fan-in), and many-to-many. Gaurantees at least once delivery before deletion from queue.

Scenarios

- Balancing workloads in network clusters- queue can efficiently distribute tasks
- Implementing asynchronous workflows
- Data streaming from various processes or devices
- Reliability improvement- in case zone failure
- Distributing event notifications
- Refreshing distributed caches
- Logging to multiple systems

Benefits/Features

Unified messaging, global presence, push- and pull-style subscriptions, replicated storage and guaranteed at-least-once message delivery, encryption of data at rest/transit, easy-to-use REST/JSON API

Data Model

Topic, **Subscription**, **Message** (combination of data and attributes that a publisher sends to a topic and is eventually delivered to subscribers), **Message Attribute** (key-value pair that a publisher can define for a message)

Message Flow

- Publisher creates a topic in the Cloud Pub/Sub service and sends messages to the topic.
- Messages are persisted in a message store until they are delivered and acknowledged by subscribers.
- The Pub/Sub service forwards messages from a topic to all of its subscriptions, individually. Each subscription receives messages either by pushing/pulling.
- The subscriber receives pending messages from its subscription and acknowledges message.
- When a message is acknowledged by the subscriber, it is removed from the subscription's message queue.

ML Engine

Managed infrastructure of GCP with the power and flexibility of TensorFlow. Can use it to train ML models at scale and host trained models to make predictions about new data in the cloud. Supported frameworks include Tensorflow, scikit-learn and XGBoost.

ML Workflow

Evaluate Problem: What is the problem and is ML the best approach? How will you measure model's success?
Choosing Development Environment: Supports Python 2 and 3 and supports TF, scikit-learn, XGBoost as frameworks.
Data Preparation and Exploration: Involves gathering, cleaning, transforming, exploring, splitting, and preprocessing data. Also includes feature engineering.
Model Training/Testing: Provide access to train/test data and train them in batches. Evaluate progress/results and adjust the model as needed. Export/save trained model (250 MB or smaller to deploy in ML Engine).
Hyperparameter Tuning: hyperparameters are variables that govern the training process itself, not related to the training data itself. Usually constant during training.
Prediction: host your trained ML models in the cloud and use the Cloud ML prediction service to infer target values for new data

ML APIs

Speech-to-Text: speech-to-text conversion
Text-to-Speech: text-to-speech conversion
Translation: dynamically translate between languages
Vision: derive insight (objects/text) from images
Natural Language: extract information (sentiment, intent, entity, and syntax) about the text: people, places..
Video Intelligence: extract metadata from videos

Cloud Datalab

Interactive tool (run on an instance) to explore, analyze, transform and visualize data and build machine learning models. Built on Jupyter. Datalab is free but may incus costs based on usages of other services.

Data Studio

Turns your data into informative dashboards and reports. Updates to data will automatiaclly update in dashbaord. **Query cache** remembers the queries (requests for data) issued by the components in a report (lightning bolt)-turn off for data that changes frequently, want to prioritize freshness over performance, or using a data source that incurs usage costs (e.g. BigQuery). **Prefetch cache** predicts data that could be requested by analyzing the dimensions, metrics, filters, and date range properties and controls on the report.

ML Concepts/Terminology

Features: input data 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
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
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
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

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.

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

Popular Architectures

Linear Classifier: takes input features and combines them with weights and biases to predict output value
DNNClassifier: deep neural net, contains intermediate layers of nodes that represent "hidden features" and activation functions to represent non-linearity
ConvNets: convolutional neural nets. 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 neural nets, 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 neural net, 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

Case Study: Flowlogistic I

Company Overview

Flowlogistic is a top logistics and supply chain provider. They help businesses throughout the world manage their resources and transport them to their final destination. The company has grown rapidly, expanding their offerings to include rail, truck, aircraft, and oceanic shipping.

Company Background

The company started as a regional trucking company, and then expanded into other logistics markets. Because they have not updated their infrastructure, managing and tracking orders and shipments has become a bottleneck. To improve operations, Flowlogistic developed proprietary technology for tracking shipments in real time at the parcel level. However, they are unable to deploy it because their technology stack, based on Apache Kafka, cannot support the processing volume. In addition, Flowlogistic wants to further analyze their orders and shipments to determine how best to deploy their resources.

Solution Concept

Flowlogistic wants to implement two concepts in the cloud:

- Use their proprietary technology in a real-time inventory-tracking system that indicates the location of their loads.
- Perform analytics on all their orders and shipment logs (structured and unstructured data) to determine how best to deploy resources, which customers to target, and which markets to expand into. They also want to use predictive analytics to learn earlier when a shipment will be delayed.

Business Requirements

-Build a reliable and reproducible environment with scaled parity of production
-Aggregate data in a centralized Data Lake for analysis
-Perform predictive analytics on future shipments
-Accurately track every shipment worldwide
-Improve business agility and speed of innovation through rapid provisioning of new resources
-Analyze/optimize architecture cloud performance
-Migrate fully to the cloud if all other requirements met

Technical Requirements

-Handle both streaming and batch data
-Migrate existing Hadoop workloads
-Ensure architecture is scalable and elastic to meet the changing demands of the company
-Use managed services whenever possible
-Encrypt data in flight and at rest
-Connect a VPN between the production data center and cloud environment

Case Study: Flowlogistic II

Existing Technical Environment

Flowlogistic architecture resides in a single data center:

- Databases:
  - 8 physical servers in 2 clusters
    - SQL Server: inventory, user/static data
  - 3 physical servers
    - Cassandra: metadata, tracking messages
  - 10 Kafka servers: tracking message aggregation and batch insert
- Application servers: customer front end, middleware for order/customs
  - 60 virtual machines across 20 physical servers
    - Tomcat: Java services
    - Nginx: static content
    - Batch servers
- Storage appliances
  - iSCSI for virtual machine (VM) hosts
  - Fibre Channel storage area network (FC SAN): SQL server storage
  - Network-attached storage (NAS): image storage, logs, backups
- 10 Apache Hadoop / Spark Servers
  - Core Data Lake
  - Data analysis workloads
- 20 miscellaneous servers
  - Jenkins, monitoring, bastion hosts, security scanners, billing software

CEO Statement

We have grown so quickly that our inability to upgrade our infrastructure is really hampering further growth/efficiency. We are efficient at moving shipments around the world, but we are inefficient at moving data around. We need to organize our information so we can more easily understand where our customers are and what they are shipping.

CTO Statement

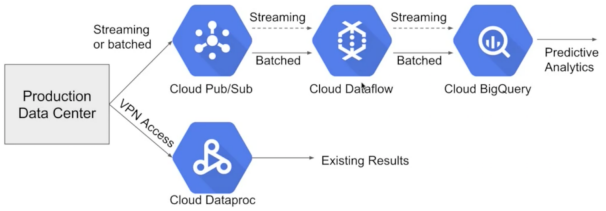
IT has never been a priority for us, so as our data has grown, we have not invested enough in our technology. I have a good staff to manage IT, but they are so busy managing our infrastructure that I cannot get them to do the things that really matter, such as organizing our data, building the analytics, and figuring out how to implement the CFO's tracking technology.

CFO Statement

Part of our competitive advantage is that we penalize ourselves for late shipments/deliveries. Knowing where our shipments are at all times has a direct correlation to our bottom line and profitability. Additionally, I don't want to commit capital to building out a server environment.

Flowlogistic Potential Solution

1. Cloud Dataproc handles the existing workloads and produces results as before (using a VPN).
2. At the same time, data is received through the data center, via either stream or batch, and sent to Pub/Sub.
3. Pub/Sub encrypts the data in transit and at rest.
4. Data is fed into Dataflow either as stream/batch data.
5. Dataflow processes the data and sends the cleaned data to BigQuery (again either as stream or batch).
6. Data can then be queried from BigQuery and predictive analysis can begin (using ML Engine, etc..)



Note: All services are fully managed., easily scalable, and can handle streaming/batch data. All technical requirements are met.

Case Study: MJTelco I

Company Overview

MJTelco is a startup that plans to build networks in rapidly growing, underserved markets around the world. The company has patents for innovative optical communications hardware. Based on these patents, they can create many reliable, high-speed backbone links with inexpensive hardware.

Company Background

Founded by experienced telecom executives, MJTelco uses technologies originally developed to overcome communications challenges in space. Fundamental to their operation, they need to create a distributed data infrastructure that drives real-time analysis and incorporates machine learning to continuously optimize their topologies. Because their hardware is inexpensive, they plan to overdeploy the network allowing them to account for the impact of dynamic regional politics on location availability and cost.

Their management and operations teams are situated all around the globe creating many-to-many relationship between data consumers and providers in their system. After careful consideration, they decided public cloud is the perfect environment to support their needs.

Solution Concept

MJTelco is running a successful proof-of-concept (PoC) project in its labs. They have two primary needs:

- Scale and harden their PoC to support significantly more data flows generated when they ramp to more than 50,000 installations.
- Refine their machine-learning cycles to verify and improve the dynamic models they use to control topology definitions. MJTelco will also use three separate operating environments – development/test, staging, and production – to meet the needs of running experiments, deploying new features, and serving production customers.

Business Requirements

-Scale up their production environment with minimal cost, instantiating resources when and where needed in an unpredictable, distributed telecom user community.
-Ensure security of their proprietary data to protect their leading-edge machine learning and analysis.
-Provide reliable and timely access to data for analysis from distributed research workers.
-Maintain isolated environments that support rapid iteration of their machine-learning models without affecting their customers.

Case Study: MJTelco II

Technical Requirements

-Ensure secure and efficient transport and storage of telemetry data.
-Rapidly scale instances to support between 10,000 and 100,000 data providers with multiple flows each.
-Allow analysis and presentation against data tables tracking up to 2 years of data storing approximately 100m records/day. Support rapid iteration of monitoring infrastructure focused on awareness of data pipeline problems both in telemetry flows and in production learning cycles.

CEO Statement

Our business model relies on our patents, analytics and dynamic machine learning. Our inexpensive hardware is organized to be highly reliable, which gives us cost advantages. We need to quickly stabilize our large distributed data pipelines to meet our reliability and capacity commitments.

CTO Statement

Our public cloud services must operate as advertised. We need resources that scale and keep our data secure. We also need environments in which our data scientists can carefully study and quickly adapt our models. Because we rely on automation to process our data, we also need our development and test environments to work as we iterate.

CFO Statement

This project is too large for us to maintain the hardware and software required for the data and analysis. Also, we cannot afford to staff an operations team to monitor so many data feeds, so we will rely on automation and infrastructure. Google Cloud's machine learning will allow our quantitative researchers to work on our high-value problems instead of problems with our data pipelines.

Choosing a Storage Option

Need	Open Source	GCP Solution
Compute, Block Storage	Persistent Disks, SSD	Persistent Disks, SSD
Media, Blob Storage	Filesystem, HDFS	Cloud Storage
SQL Interface on File Data	Hive	BigQuery
Document DB, NoSQL	CouchDB, MongoDB	DataStore
Fast Scanning NoSQL	HBase, Cassandra	BigTable
OLTP	RDBMS-MySQL	CloudSQL, Cloud Spanner
OLAP	Hive	BigQuery

Cloud Storage: unstructured data (blob)
CloudSQL: OLTP, SQL, structured and relational data, no need for horizontal scaling
Cloud Spanner: OLTP, SQL, structured and relational data, need for horizontal scaling, between RDBMS/big data
Cloud Datastore: NoSQL, document data, key-value structured but non-relational data (XML, HTML, query depends on size of result (not dataset), fast to read/slow to write
BigTable: NoSQL, key-value data, columnar, good for sparse data, sensitive to hot spotting, high throughput and scalability for non-structured key/value data, where each value is typically no larger than 10 MB

Solutions

Data Lifecycle

At each stage, GCP offers multiple services to manage your data.

1. **Ingest**: first stage is to pull in the raw data, such as streaming data from devices, on-premises batch data, application logs, or mobile-app user events and analytics
2. **Store**: after the data has been retrieved, it needs to be stored in a format that is durable and can be easily accessed
3. **Process and Analyze**: the data is transformed from raw form into actionable information
4. **Explore and Visualize**: convert the results of the analysis into a format that is easy to draw insights from and to share with colleagues and peers

Ingest

There are a number of approaches to collect raw data, based on the data's size, source, and latency.

- **Application**: data from application events, log files or user events, typically collected in a push model, where the application calls an API to send the data to storage (Stackdriver Logging, Pub/Sub, CloudSQL, Datastore, Bigtable, Spanner)
- **Streaming**: data consists of a continuous stream of small, asynchronous messages. Common uses include telemetry, or collecting data from geographically dispersed devices (IoT) and user events and analytics (Pub/Sub)
- **Batch**: large amounts of data are stored in a set of files that are transferred to storage in bulk. common use cases include scientific workloads, backups, migration (Storage, Transfer Service, Appliance)

Solutions- Part II

Storage

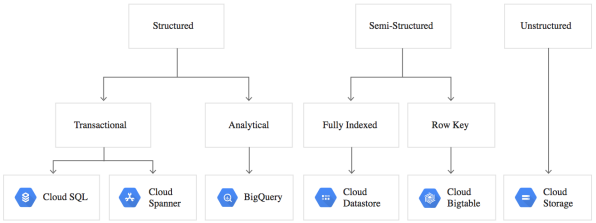
Cloud Storage: durable and highly-available object storage for structured and unstructured data

Cloud SQL: fully managed, cloud RDBMS that offers both MySQL and PostgreSQL engines with built-in support for replication, for low-latency, transactional, relational database workloads. supports RDBMS workloads up to 10 TB (storing financial transactions, user credentials, customer orders)

BigTable: managed, high-performance NoSQL database service designed for terabyte- to petabyte-scale workloads. suitable for large-scale, high-throughput workloads such as advertising technology or IoT data infrastructure. does not support multi-row transactions, SQL queries or joins, consider Cloud SQL or Cloud Datastore instead

Cloud Spanner: fully managed relational database service for mission-critical OLTP applications. horizontally scalable, and built for strong consistency, high availability, and global scale. supports schemas, ACID transactions, and SQL queries (use for retail and global supply chain, ad tech, financial services)

BigQuery: stores large quantities of data for query and analysis instead of transactional processing



Exploration and Visualization Data exploration and visualization to better understand the results of the processing and analysis.

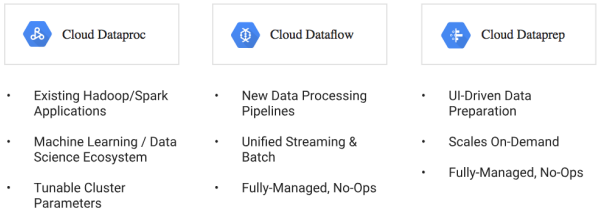
- **Cloud Datalab**: interactive web-based tool that you can use to explore, analyze and visualize data built on top of Jupyter notebooks. runs on a VM and automatically saved to persistent disks and can be stored in GC Source Repo (git repo)
- **Data Studio**: drag-and-drop report builder that you can use to visualize data into reports and dashboards that can then be shared with others, backed by live data, that can be shared and updated easily. data sources can be data files, Google Sheets, Cloud SQL, and BigQuery. supports **query** and **prefetch cache**: query remembers previous queries and if data not found, goes to prefetch cache, which predicts data that could be requested (disable if data changes frequently/using data source incurs charges)

Solutions- Part III

Process and Analyze

In order to derive business value and insights from data, you must transform and analyze it. This requires a processing framework that can either analyze the data directly or prepare the data for downstream analysis, as well as tools to analyze and understand processing results.

- **Processing**: data from source systems is cleansed, normalized, and processed across multiple machines, and stored in analytical systems
- **Analysis**: processed data is stored in systems that allow for ad-hoc querying and exploration
- **Understanding**: based on analytical results, data is used to train and test automated machine-learning models



Processing

- **Cloud Dataproc**: migrate your existing Hadoop or Spark deployments to a fully-managed service that automates cluster creation, simplifies configuration and management of your cluster, has built-in monitoring and utilization reports, and can be shutdown when not in use
- **Cloud Dataflow**: designed to simplify big data for both streaming and batch workloads, focus on filtering, aggregating, and transforming your data
- **Cloud Dataprep**: service for visually exploring, cleaning, and preparing data for analysis. can transform data of any size stored in CSV, JSON, or relational-table formats

Analyzing and Querying

- **BigQuery**: query using SQL, all data encrypted, user analysis, device and operational metrics, business intelligence
- **Task-Specific ML**: Vision, Speech, Natural Language, Translation, Video Intelligence
- **ML Engine**: managed platform you can use to run custom machine learning models at scale

streaming data: use Pub/Sub and Dataflow in combination