1. Introductory Classical Statistics: t-test, Wilcoxon, Correlation

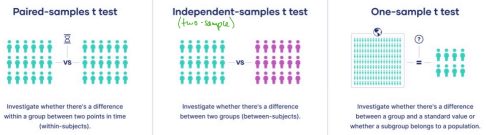
$$\text{test statistic} = \frac{\text{Observed data - what we expect if } H_0 \text{ is true}}{\text{Avg Variation}}$$

t-test

- a test used to compare means of 2 groups to see if treatment group has stat sig difference ("Pair wise comparison")

- Assumes: 1. Independent events 2. Normal Dist. 3. Similar variance amongst both groups
  - If Assumptions fail, use Wilcoxon Signed-Rank test

- Types



One-tailed: $H_0: \mu_1 < \mu_2$ — Pop 1 is less than Pop 2
two-tailed: $H_0: \mu_1 \neq \mu_2$ — Pops are diff

$$t = \frac{\mu_1 - \mu_2}{\sqrt{s^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

μ_i - mean
s - standard error of both groups
n_i - # of observations

Calculated difference in samples [represented in units of stand. error] / [relative to the data's variation]

- Code

```
t.test(variable ~ category var, data)
t.test(length ~ Species, flower.data)
```

Wilcoxon

- compares Paired-Samples (see above) to determine if there's a stat. sig. diff.
- for nonparametric data: isn't quantitative but CAN be ranked (eg. customer satisfaction)
- Assumes: 1. data is Paired-Sample 2. continuous data (since time) 3. Randomly sampled data

The **Wilcoxon Signed-Rank Test** is a statistical test used to determine if 2 measurements from a single group are significantly different from each other on your variable of interest. Your variable of interest should be continuous and your group randomly sampled to meet the assumptions of this test.

- types:
  1. Rank-sum: tests H_0: samples have same distribution i.e. no diff in samples
  2. Signed-Rank: nonparametric version of t-test tests H_0: Sample 1 is less/greater than Sample 2 Assumes skewed dist.

- code:

```
wilcox.test(sample1.vector, sample2.vector, alternative: "greater")
```

Correlation tests

- measures strength & direction of association btw two vars
- Types: Pearson, Kendall, Spearman

$$0 < r < 1$$

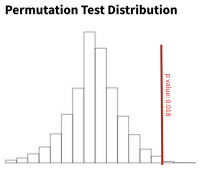
weak → strong

2. Resampling Methods: Randomization tests, Bootstrap, Cross-Validation

- the process of creating new samples based on 1 observed sample

Randomization tests

- aka Permutation tests
- nonparametric w/ very few assumptions => helpful when much is unknown about Pop.
- Randomly sample (without Replacement) OG data set into samples of different permutations, then peform test-stat to see if diff. exist (P-value)
  All samples are the same dim as OG



Bootstrap

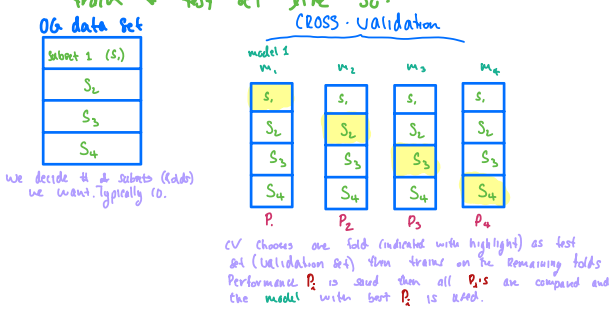
- Randomly select observations from OG data set (with Replacement) s.t. your new sample dim = OG dim. Call this Sample 1 (S_1)
  Find the μ_1: mean (S_1) doesn't have to be mean, can be median or some other stat.
  Repeat this for many S_i and μ_i.
  Add μ_i's to histogram to estimate what our data would look like if we Repeated OG experiment/sample many times
- Basically its a way to simulate what would happen if we Repeated experiment 1,000's of times
- Code:

```
library (boot)
mean.fn ← function (vector, i) {
    mean (vector [i])
}                                  # any stat

boot (data, mean.fn, R = 1000)
```

CROSS-Validation

- Automatically splits OG set into differently sized subsets for each train & test set like so:



P_i is proportion of correctly predicted observations

- types: Validation-set, Leave one out CV, k-fold, Repeated CV
- Code:

```
library (caret)
train_control ← trainControl (method = "LOOCV")
model ← train (y ~ ., data,
               method = "lm",
               trControl = train_control)
```

k-fold

```
method: "CV"
number = 10
```

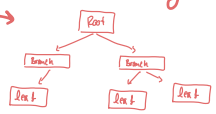
Repeated k-fold

```
method: "repeatedcv"
number = 10
repeats = 3
```

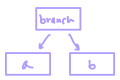
3. Decision Trees, Random Forest, Boosted Trees

Decision Trees

- non parametric supervised learning method used for classification + regression based on how a previous set of Q's were answered.



- types - classification (into categories) + Regression (predicts numeric values)
- leaves are PURE if a or b is empty

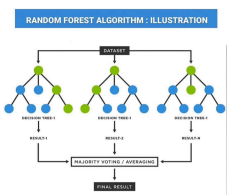


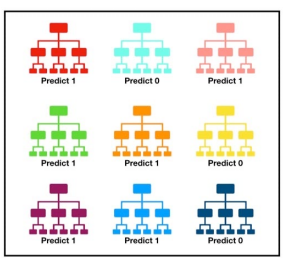
- Quantify impurity to train model: Gini Impurity, Entropy, Information Gain
- Output of leaf is the category with most votes
- Code: Sklearn
- main Disadvantage: tends to overfit training data

solution to this ↓

Random Forest

- a set of many decision trees: each outputting a class prediction → the class that is predicted the most is our model's prediction
  Called: Bagging (bootstrap aggregating)





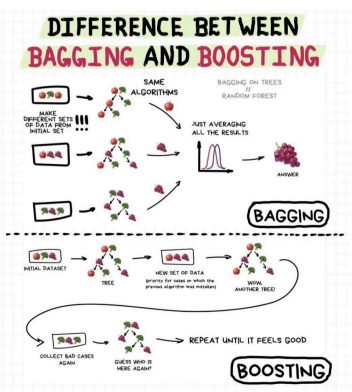
Tally: Six 1s and Three 0s
Prediction: 1

- Fundamental idea: a large # of uncorrelated models (trees) operating as a committee will outperform any individual model
  ★ The truth will out!
- bagging: trees are Independent and are computed in Parallel. Thus, Forest is an amalgamation of a bunch of Ind tree models
- Main Drawback: if one tree has a discrepancy then whole forest has discrepancy. They are all interdependent "Parallel Learning"

solution to this ↓

Boosted Trees

- Boosting: trees are created sequentially one after another. Each tree is created by previous tree, taking the errors of its predecessor into account



- code: extreme gradient boosting (XGBoost)
  Adaptive boosting (AdaBoost)

4. Generalized Additive Models (GAM)

- An alternative to linear models when data is non-linear that's easy to interpret and more flexible/complex than linear

Recall: linear models

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \quad \text{where} \quad \begin{cases} \beta_n \text{ - weights} \\ x_n \text{ - variables} \end{cases}$$

- GAM's replace β_n's with smooth non-linear functions called Splines (s_n)

$$y = s_0 x_0 + s_1 x_1 + \cdots + s_n x_n \quad \text{, where} \quad s_n = \sum_{k=1}^{k} \beta_k \cdot b_k(x)$$

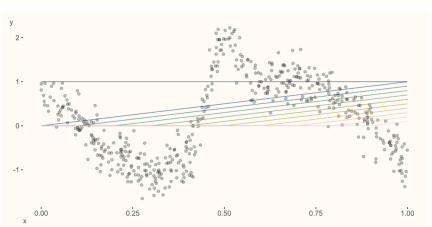
β_k - weights
$b_k(x)$ - "basis expansion" x^0, x^1, x^2, etc.

Splines - hard to define + understand. Essentially a polynomial func. with small range

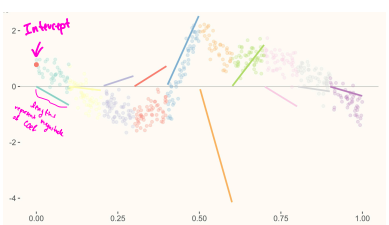
- The final model is just the sum of all the splines
- Wiggliness - how wiggly our model's line of best fit is. The more splines the more wiggly (overfitting)
  So there's strategy in deciding # of splines used
  ↓
  multiply s_k's by λ penalty term then use CV to optimize
  ★ Start w/ high # of splines then CV with λ to optimize
- Any Distribution goes: Normal, Poisson, Binomial
- Code:

```
library(mgcv)
model1 <- gam(y ~ x0 + s(x1) + s(x2), data)
model2 <- gam(y ~ x0 + s(x1)        , data)
```

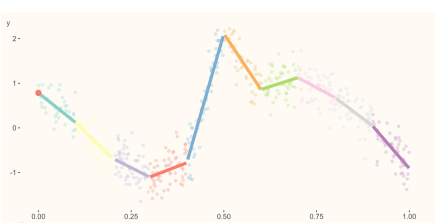
Set Spline degree = 1 (linear)



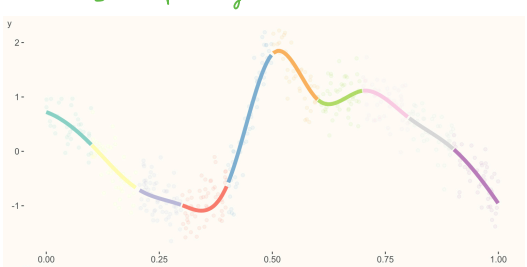
Multiply b_k's ("basis") by their corresponding Regression coeff



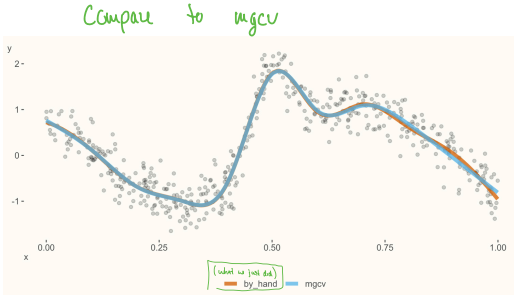
Sum of basis functions



Set Spline deg = 3 (cubic)

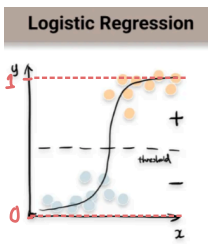


Compare to mgcv



5. Logistic Regression + Classification Models

Logistic Regression



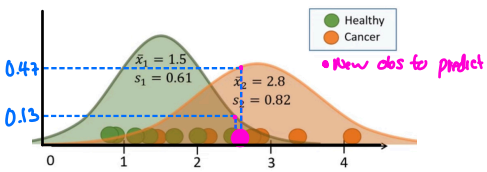
- Sigmoidal function: maps any Real # to value btw 0-1
  $x: \mathbb{R} \to (0,1)$

$$f(x) = \frac{1}{1+e^{-x}}$$

- helpful bc $0 < y < 1$, y - predicted output
- ex: x - emails
  y - prob that x is spam
  $\Rightarrow f(x) = \begin{cases} \text{Spam}, & \text{if } y \geq 0.5 \\ \text{Not Spam}, & \text{if } y < 0.5 \end{cases}$

- Code: `logReg <- glm(y ~ ., data, family = "binomial")`

Naive Bayes



• new obs to predict

1. Find mean ($\bar{x}$) and st dev (s) of each class
2. Plug into Gaussian function to plot Normal Dist's

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

3. Predict new observations' classes based on height of each curve (height Reps. prob. that the obs. is in that class) See: Pink
   $\text{height}_{\text{cancer}} = 0.47 \Rightarrow$ more likely so this class
   $\text{height}_{\text{healthy}} = 0.13$

- Based on Bayes Thm: $P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$, A - outcome (classification), B - Predictors: $B = x_1, x_2, \ldots, x_n$

$$\text{Posterior} = \frac{\text{Prior} \cdot \text{likelihood}}{\text{evidence}}$$

- "Naive" bc predictors are Independent or "naive" towards one another. hey Sam = Sam hey; $x_1 + x_2 = x_2 + x_1$

"Gaussian" because this is a normal distribution

This is our prior belief

$$P(\text{class}|\text{data}) = \frac{P(\text{data}|\text{class}) \times P(\text{class})}{P(\text{data})}$$

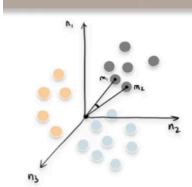
We don't calculate this in naive bayes classifiers

- All predictors have equal effect on outcome. Continuous vars

$$P(y|(x_1, x_2, x_3, \ldots, x_n)) = \frac{P(x_1|y)\,P(y) \cdot P(x_2|y)\,P(y) \cdots P(x_n|y)\,P(y)}{P(x_1) \cdot P(x_2) \cdots P(x_n)} = \prod_{k=1}^{n} \frac{P(x_k|y)\,P(y)}{P(x_k)}$$

Code:
```
library(caret)
nb.fit <- naiveBayes(classVar ~ ., trainingSet)
nb.pred <- predict(nb.fit, testSet)
confusion matrix...
```

k-Nearest Neighbors



- nonparametric supervised classifier that uses proximity of training data to make predictions about the grouping of an individual point
- k - labels describe the # of training observations within the neighborhood (for each class) ex:

4 Purple
3 blue
2 Red
k neighborhood

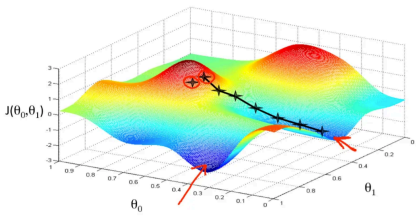
- Regression: returns mean of k-labels
- Classification: "mode"
- Pros: simple, versatile
- Cons: costly as predictors inc

Code:
```
library(class)
knn.fit <- knn(train = trainingSet
               test = testSet
               cl = classSet
               k = neighborhood)
```

confusion matrix...

Stochastic Gradient Descent

- Procedure to find minimum point of a cost function. Stochastic = Random
- Blindfolded on a MTN summit -> feel around with your feet to get Down.
- Iterative step algorithm. At each step, we choose a Random direction then calc step size
- Learning Rate - parameter that controls model's flexibility
  as learning rate INC steps INC -> too large and your model may jump over min. point



Step size = gradient · learning rate
new params = old params - step size
calc. gradient with partial derivatives

Code: sgd(y ~ ., data, model = "lm")
lin Reg is used here but can be other models

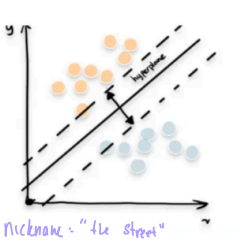
Normal GD considers all observations at each step. Stochastic randomly picks 1.

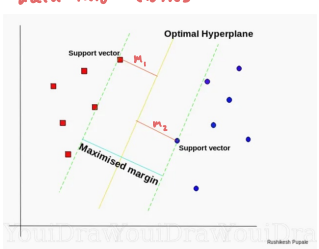
Normal GD Con: costly => a observations, b predictors
a^b computations per step
1000 steps is typical so => $1000 \cdot a^b$

* mini-batch GD is a solution to this as it takes a subset of a + Stochastic GD since it chooses a randomly

Support Vector Machines

- Algorithm creates a line/Hyperplane to separate data into classes





- Support Vector - a point within each class that is closest to estimated Hyperplane
- Margin: $M_1 + M_2$
- When margin is maximized optimal Hyperplane is found derive widest possible street

Aha moment: models are ALWAYS linearly separable when observed from the perspective of it's highest Dimension

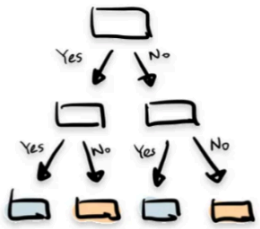
- mapping to higher dimensions is the toughest part of this AND computationally expensive
  solution: kernal trick => uses vector dot products instead of mapping to HDims
- Pro: handles higher dims well, versatile
- Con: Sensitive to kernals so if n<p SVM performs poorly,
  non-Probabalistic /not for Regression - but - can measure effectiveness of classification by looking at distance from new point to hyperplane

- soft vs. hard margin classification: allowing vs. not-allowing data within margin
  * if soft then use cost function to penalize (decentivize) data from being inside

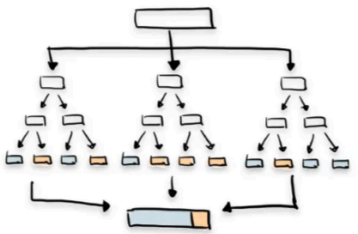
- Code: library(e1071)
  svm.fit(y ~ ., data, kernal = "linear", cost = 10, scale = False)
  - kernal: polynomial, Radial, Sigmoid
  - cost: penalized data within margin
  - scale: Scale data bf fitting?

Decision TREE & Random Forests

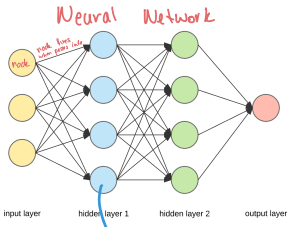
Decision Tree



Random Forest



6. Neural Networks



- If network > 3 layers including input/output then considered Deep learning Network
- The more layers the greater ability to recognise complexity
- Each node connection has weight representing how helpful node is in correctly classifying info
- Each layer has a weight threshold that must be satisfied for info to be passed on to next layer
- Initially weights + thresholds are randomly set
  - ↳ then with Forward + Backward Propagation they're updated + optimized
- Corrective Feedback Loop: gives more weight to nodes that guess correctly, less to mistaken nodes

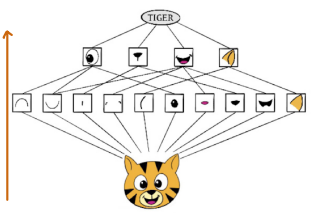
machine learning

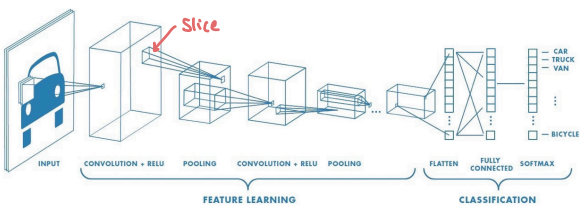
Each node has an Activation Function to compute weight. Pop functions: Soft Plus, Rectified Linear, Sigmoid

- When initializing a NN we must decide: # of hidden layers, # of nodes within each layer, Activ. func.

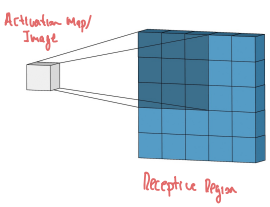
Convolutional NN

Gives computers Sight



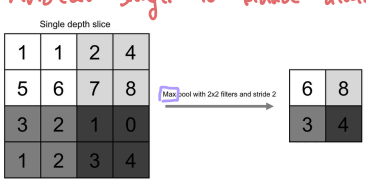


- Best for Image Analysis or any data with grid-like topology
- Picks out patterns + makes sense of them. Layers are arranged s.t. easier patterns (lines, curves, etc) are picked out 1st then more complex (faces, objects)
- 3 Layers
  1. Convolution layer: main computational layer
     - performs dot prod of 2 matrices
       - ↳ one is set of learnable parameters (kernals)
       - other is the nxn pixel set from input
     - Response map - matrix of dot products



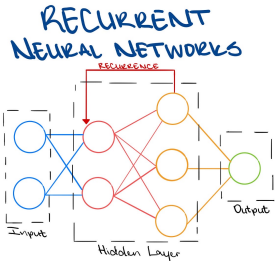
- Kernal Slides across height + width of Receptive Region, computing the dot prod each time and storing it in a smaller matrix image called the Activation map
  - Reduces memory ∴ "Sparse interaction"

2. Pooling layer: Computes + Stores a summary statistic for each Activation map
   - ex: mean, L2 norm, maximum of elements within nxn map
   - "Pooling function/operation"
   - Another layer to Reduce dimensions of input



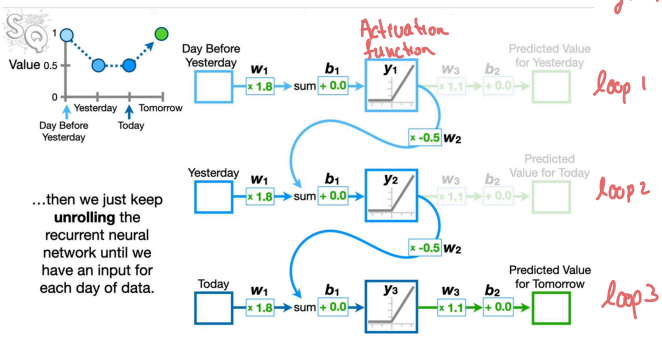
3. Fully connected Layer: - brings everything back together
   - All neurons are connected/mapped to all those immediately preceding + succeeding

Recurrent NN



ex: Stocks for 2 companies that started at diff dates

- Works well if inputs have diff dimensions
- Uses Feedback loops
- Sequencial data ex: time
- Each loop is for each previous time stamp / sequence in input data
- not used often bc as loops inc model gets hard to train "The Vanishing/Exploading Gradient Problem"
  ★ costly!!

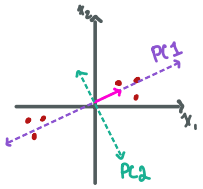


Long Short-term memory

7. UnSupervised Learning: Clustering & Principal Component Analysis

PCA

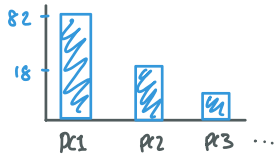
- Clustering method to Represent multi-dimensional data ($p > 3$) in 2 dimensions AND can determine which variable is most valuable
- Procedure:
  1. center data about origin & Project higher dim data ($d > 2$) on 2D plane
  2. Fit line of best fit by MAXIMIZING Sum of Squares of data's projections
     - PC1 is linear combo of this line: $\vec{u} = 0.97x_1 + 0.24x_2$
       - unit vector is eigenvector
       - mean(Sum of Sq) is eigen value for PC1
       - $\sqrt{\text{Sum of Sq}}$ is Single Value for PC1



eigen vector: $\langle x_1, x_2 \rangle = \langle 0.97, 0.24 \rangle$

  3. PC2 is line $\perp$ PC1
  4. Rotate Everything s.t. PC1 lies horizontally
  5. Use projected training obs. to compute eigenvalues for each PC which Reps the % of variation each PC accounts for
     - Eigenvalues are measures of variation
  6. Repeat 1-5 until you go thru all vars/dimensions
     - ★ Thus, # of PC's = # of vars OR # of obs (whichever is smaller)

6.5 Scree Plots: bar graph Representing % of variation caused by PC1 + PC2



- PC1 var accounted for > PC2 > PC3 > ...

Code: `pca ← prcomp(data.matrix, scale=TRUE)`

↳ Returns 3 things: x (↓ PC's), Sdev (standard dev), Rotation (loading scores)

Clustering

K-means

Procedure:

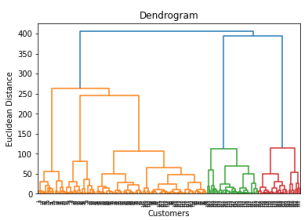
1. Choose # of clusters / centers (C) you want ★ CV helps optimize this
2. Randomly select C obs. to establish initial clusters
3. Assign all other obs. to their closest C cluster
4. Calc mean of each C
5. Assess quality of clustering by adding up var within each C: "total var"
6. Keep track of total vars then Repeat 1-5 a few more times
7. Compare total var values from 1-6 iterations and choose best one

(Clustering Process)

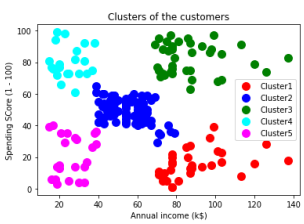
Code: `kmeans(x= data.vector, centers= 3)`

Hierarchal

Hierarchical



K-means

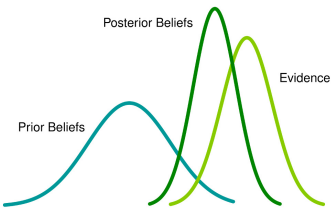


	K-means	Hierarchical
Time Complexity	O(n*k*t)	O(n^3)
Space Complexity	O(n(d + k))	O(n^2)
Hyperparameters Tuning	Must specify the number of clusters (k) and retrain model for each k	Can dynamically update k value without retraining model
Data Structure	Better performance when dealing with convex clusters	Generate better result when dealing with non-convex clusters
Variations	Many variations (e.g., K-median, K-medoid…) with different distance matrices	Two approaches: Agglomerative approach and Divisive approach
Optimization	K-means++ introduces smarter initialization of centroids, making convergence faster	Top-down approach reduces time complexity to O(n^2)
Result Robustness	Result may be different on different runs	Same parameters generate the same result every time

8. Bayesian Analysis

Bayesian Statistics

- A theory in the field of statistics based on Bayesian Interpretation of Probability where prob expresses a degree of belief in an event based on prior knowledge about the event (ie. results of prev. experiment)
- This is different from other Interpretations of Probability such as The Frequentist Interpretation that view prob as the limit of the Relative Frequency of an event after many trials.



$$\text{Posterior} = \frac{\text{Prior} \cdot \text{likelihood}}{\text{evidence}}$$

- All parameters assumed to be Random Quantities
- A param is summarized by an entire Dist. of values instead of just one value as is in Frequentist Analysis
- Posterior dist. is the goal / "heart" of Bayesian

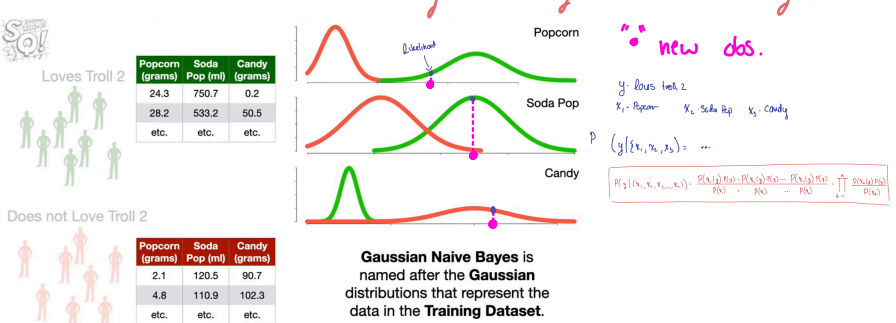
Gaussian Naive Bayes

- for continuous data & data w/ Normal Dist
- Good at handling continuous values

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

Multinomial Naive Bayes

- text classification: cares about # of occurrences of the word
- Good at handling Discrete values. Can classify non-numeric data
- main advantage: significantly Reduces Complexity
  - ↳ classification using small training sets without needing to be retrained



Bernoulli Naive Bayes

- text classification: cares whether the word happened or not
- Good at handling binary/boolean values

9. Bayesian Ext. Markov Chain Monte Carlo (mcmc)

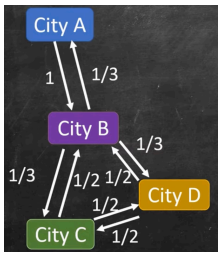
- A method to approximate Posterior Dist. of a parameter if not able to do so Analytically
- Applies Law of Large #'s (Central Limit Theorem) to Dependent Results

Markov Chain a sequence of events where the probs of the future only depend on the present

- Markov Property: given the present, the future is Independent of the past Memorylessness
- The posterior probability relies solely on present probability and Randomness of data generation

The next step is a Random choice based on current step

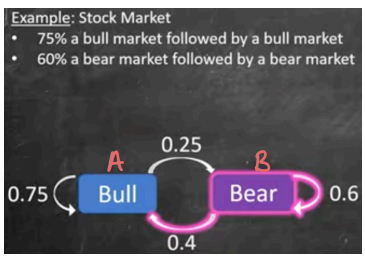
- Non-markov has memory



- Traveling example
- Transition Diagram: same idea as tree diagram
- Transition Matrix: $P = \begin{matrix} & A & B \\ A & 0.75 & 0.4 \\ B & 0.25 & 0.6 \end{matrix}$

$S_n = P^n S_0$ where P- transition matrix, S_0- initial state, S_n- n^{th} state

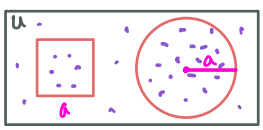
State: Bull vs. Bear, city A, B, C, D,



- A representation of how a var walks around a graph Nicknamed "Random Walk"
- How a Random var changes from one state to the next

Monte Carlo

- Simulations evolving Randomnly
- Fundamental idea that allows us to say: Random samples can represent Populations

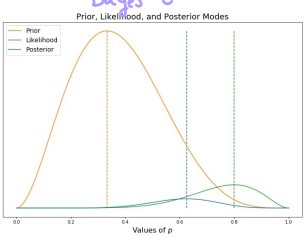


Randomly dropping marbles in U. Proportions of marbles in circle vs. square match area Ratio: $\pi a^2 / a^2 = \pi$

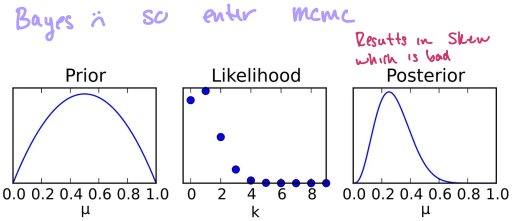
MCMC

- Allows us to leverage computers to do Bayesian Stats
- Bayesian Stats requires prior Dist. + likelihood Dist to be Normal

Bayes ☺



Bayes ☹ so enter mcmc



MCMC is used when prior + likelihood aren't Normally Dist.

Procedure

1. Monte Carlo : Generate Random Sample
   ex: generate a series θ_t of random #s from normal dist. $\theta_t \sim N(\mu, \sigma)$
2. Markov Chain : Determine Transition diagram / matrix
   - $\theta_t \sim N(\theta_{t-1}, \sigma)$ so plugging in the prior term θ_{t-1} in as the new term's Dist. mean
3. Acceptance - Rejection Sampling : Decide to keep/discard new obs. generated from 1 & 2
   - Algorithms
     1. Metropolis-Hastings
        - We are at point x.
        - We make a guess for the next step. We will call this x*
        - We then compute the ratio of the probability of x*/x. This is calculated using the product of the likelihood and prior distributions.
        - If the ratio of p(x*)/p(x) (also called the acceptance probability) is greater than 1 we accept x* as the new position.
        - Even if the acceptance probability is less than 1, we don't automatically reject x*. We flip a coin by selecting a random number, from a Uniform(0,1) distribution. If the number is smaller than the acceptance probability we accept x* if it is higher we reject x* and start the process over again.
     2. No U-turn

Summary Procedure

- We randomly generate numbers: This is the Monte Carlo part
- We allow the numbers we generated to influence the next generated number: This is the Markov chain
- We then decide if the new numbers generated are “moving in the right direction”: The Acceptance-rejection algorithm
- We then check for convergence: We determine when our data has converged to a reasonable distribution. The randomly generated values after the convergence point become our posterior distribution