Data Science Fundamentals

* Premise
  + Map reality on the ground: combines crowd-sourcing and machine learning to better understand the developing world
  + Fourth paradigm of science
    - Experimental
    - Theoretical
    - Simulation
    - ***Data intensive***
* Technology trend
  + 1980s: hardware
  + 1990s: software
  + 2000s: internet – online retailer and services
  + 2010s: data – collect and sell information
* Dark side of mathematical models
  + obscuring complex decisions
  + reinforcing historical trends
  + danger of targeting and personalization
* Data science
  + the application of data centric, computational and inferential thinking to under the world (science) and solve problems (engineering)
  + interdisciplinary
* Skills of data science
  + ***A screenshot of a cell phone screen with text

    Description automatically generated***
* Top tasks of data science
  + exploratory data analysis
  + data analysis to answer research questions
  + communicate findings
  + **data cleaning**
  + **data visualization**
  + identify business problems
  + **feature extraction (ML)**
  + **develop prototype models (ML)**
  + Implement models/algorithms
* Tools: SQL, R, Python, Python > R: data scientists, R > python: analysis. Highest paid: SCALA
* Principles CS in DS
  + Software design and debugging
  + abstraction and algorithm design
  + computational complexity
  + parallelism and locality
* Statistics in DS
  + Experimental design and sampling
  + probability and uncertainty
  + modeling
  + inference and prediction
* **Data science life cycle**
  + High-level description: frame questions & design experiment --> obtain and clean data --> summarize and visualize data --> inference and prediction
  + Ask question
  + Obtain data
  + Understand data
  + Understand world
  + Ask question / Report data products
  + ...
* Working with real data
  + messy, big, complicated

Problem formulation and experimental design

* Quantification
  + Advantage: simplify
  + Disadvantage: lose information
* Process of Data Science
  + Ask Question --> Obtain Data --> Understand Data --> Understand World --> Ask Question/Reports&Data-Products
* Domain Knowledge
* **Data Analytics (QPR-V)**
  + Question: impact everything
    - Domain question to answer
    - Polls: does not work because we cannot predict result
      * Gallup Poll: Simple random sampling, need to pick the right population. If difference between data is less than the margin of error, one poll is not enough.
    - Two types of questions
      * Hypothesis driven: whether or now?
      * Discovery-driven: what?
      * separate hypothesis generation and decision on hypothesis: discovery phase vs. validation phase
        + sample split: half for discovery, half for validation
    - Feasibility of question
      * Is the question feasible to answer using data?
      * Need to translate questions into a more precise form
        + Why didn’t polls work? – how to predict something? how to predict the whole population?
        + Did gallup poll have the recourses (energy, expertise, relevant data) to do the prediction?
  + Population: where do you get the data?
    - Population is the relevant group of people that the data-driven answer to the question will be applied to
    - Need to record the population
  + Representative data collection: data neutral. Is it a *right* sample?
    - Data neutral: data to answer the question should be representative of the relevant group
    - Qualitative
    - Quantitative
    - **Representative** sample
    - Uncertainty: we can’t ask all voters.
      * Who are going to vote?
      * How do they vote?
      * Are they going to change their minds?
    - Assumptions
      * Votes don’t change
      * People tell the truth
      * Undecided voters vote similarly as decided voters
      * Sample is representative of the population
    - How to make data neutral?
      * survey sampling: random sampling of different kinds. The simplest one is ***simple random sampling without replacement*** – putting all entities of the population in a hat and randomly drawing one by one
      * Gallup poll
        + simplest form: 1000 SRS samples from phones
        + 68 for trump, 71 for Clinton, **<2%** difference. Margin of error is about **3%.**
        + One gallup poll is not enough to predict such a close race **even if all assumptions hold and they do not**.
  + –
  + Vetting of validation of answers (minimize error)
    - Predicts on test data
    - Stability analysis
    - post-modeling EDA or visualization
    - domain knowledge verification
    - Longer time scale
      * down-stream consequences
      * further studies
* Danger zone
  + algorithms applied to domain problems without understanding of statistical concepts/issues such as population, representative data collection, and uncertainty
  + Analytical algorithms can’t automatically detect nonrepresentative or biased samples.
  + Information about undecided voters and people who do not respond can be obtained only through group operatives in their interaction with such people
* **Experiment design**
  + Data collection
    - ecological correlation does not imply correlation at person level
    - association (correlation) is not causation
    - confounding factors are always lurking in the back
      * confounding factor: a possible driver for both A and B. Does not mean A causes B.
    - What is the population?
    - Assumptions we make
    - Randomization
      * rely on pseudo-random number generators
      * reduce or combat confounding ≠0
      * ground probabilistic reasoning and calculation
* Three principles of experimental design
  + replication
  + randomization
  + blocking (reduce variability of using extra information, highly needed for election situation)

Perceptron and Metrics

Classification tasks

* + Given a set of **classes** and **instances**
    - **class**: label
      * classes must be disjoint (each instance belongs to only one class)
      * classification tasks are binary if there are only two classes
    - **instance:**  a set of features or attributes and their values
    - **input:** a set of labeled instances
  + generate a method or model that determines the class of a new instance
* Metrics for a classifier
  + **accuracy** = total test instances classified correctly / total number of instances
    - When you care about both FN and FP
    - Class distribution is similar
  + N-fold cross validation
    - suppose m labeled instances (divide into n subsets of equals size)
    - run classifier n times, with each of the fold as test sets, the rest n-1 folds for training. each run gives an accuracy result.
  + confusion matrix

|  |  |  |
| --- | --- | --- |
|  | Classified + | Classified - |
| Actual + | True positive | False negative |
| Actual - | False positive | True negative |

* + - TP: number of positive examples classified correctly
    - FN: number of positive examples classified incorrectly
    - FP: number of negative examples classified incorrectly
    - TN: number of negative examples classified correctly
  + **precision** = total correct positive / total tested positive
    - TP/(TP+FP)
    - % of selected items that are correct (in all classified positive classes, how many are actually positive)
    - Care about FP

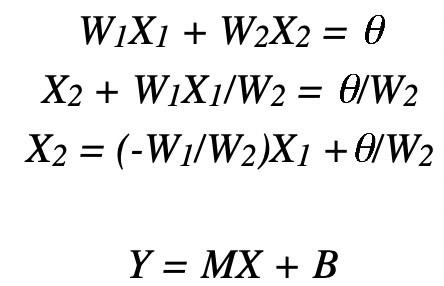
|  |  |  |
| --- | --- | --- |
|  | Classified + | Classified - |
| Actual + | True positive | False negative |
| Actual - | False positive | True negative |

* + **recall** = total correct positive / total actual positive
    - TP/(TP+FN)
    - % of corrected items that are selected (in all actually positive classes, how many are classified correctly)
    - Care about FN

|  |  |  |
| --- | --- | --- |
|  | Classified + | Classified - |
| Actual + | True positive | False negative |
| Actual - | False positive | True negative |

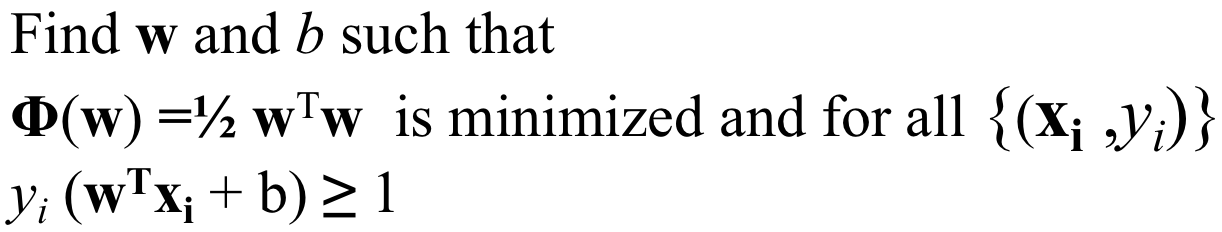
* + **f1 score** = 2\*(recall\*precision)/(recall + precision)
  + ROC curve
    - Plot true positive rate (recall) against false positive rate at various threshold settings
    - TPR = TP/(TP+FN)
    - FPR = FP/(FP+TN)
    - Lowering classification threshold classifies more items as positive, increasing both FP and TP.
  + AUC: Area under ROC curve is a measure of usefulness of a model.
    - = 1 -> good measure of separability
    - = 0 -> reciprocating results
    - = 0.5 -> no separation capacity
  + Other concerns
    - explainability of classifier results
    - costs of examples
      * costs of feature values
      * labeling
* Perceptron learning algorithm
  + simple and limited (single layer model)
  + perceptron node – threshold logic unit
    - calculate from weighing a set of features: A close up of a logo

      Description automatically generated
    - 
  + how to change weight:

    - c: learning rate
    - t: target
    - z: current output
    - principles
      * only change weights if there is an error
      * small c
      * scale by x\_i
  + **Algorithm**: create a perceptron node with n inputs
    - iteratively apply a pattern from the training set and apply the perceptron rule
    - each iteration through the training set is an epoch
    - continue training until total training set error stops improving
    - **perceptron convergence theorem: guaranteed to find a solution in finite time if a solution exists**
  + Linear separability
    - decision boundary: 
    - if no bias weight, then the hyperplane must go through the origin
  + How to handle multi-class output?
    - 1. create 1 perceptron for each output class, all other classes are negative examples
      * run all perceptrons on novel data and set the output to the class of the perceptron which outputs high
      * if there is a tie, choose the perceptron with the highest net value
    - 2. create 1 perceptron for each pair of output classes where the training set only contains examples from 2 classes
      * run all perceptrons on novel data and set the output to be the class with the most wins from perceptrons
      * tie --> net values
      * #models = #classes^2
* **Objective function**: accuracy/error/loss function
  + Used to judge the quality of a particular model (perceptron with a particular setting of weights)
    - accuracy = correct / total instances
    - error = 1-accuracy
  + Want to minimize a loss (cost, error) function
  + For real-valued outputs/targets
    - **L1 loss**: sum(|y-y’|)
    - **L2 loss**: sum(y-y’)^2
    - **total sum squared error**: sum L2 loss
    - 0 for match, 1 for a mismatch
  + **Mean squared error (MSE),** Sum squared error/#instances -> can **normalize** error for datasets with different sizes
  + **Root mean squared error (RMSE)** -> average distance from output to target in the same units/scale as features, more intuitive
  + For regression: MSE
  + For classification: likelihood loss, log loss
    - both MSE and log loss penalize heavily on very wrong label
  + Not always minimize loss function --> overfit
* **Gradient descent**: minimize(maximize) the objective function
  + Gradient: change in all weights with regard to the change in error
    - Higher gradient, learning faster
    - Lower gradient, slope zero, not learning
  + How does gradient descent work?
    - initialize w and b randomly
    - Adjust parameters in the steepest downside direction until it reaches the point where the cost function is as small as possible
  + batch update (BGD)
    - To the true gradient with the delta value, we need to sum errors over the entire training set and only update weights at the end of each epoch
    - advantage: computationally efficient, stable error gradient and stable convergence
    - disadvantage: does not always find the global minimum, memory expensive
  + stochastic update (SGD)
    - Update after every pattern, like perceptron algorithm (even though that means each change may not be exactly along the true gradient)
    - advantage: more efficient, frequent updates provide a detailed rate of improvement
    - disadvantage: more computationally expensive, noisy gradients -> error rate jumping
  + mini-batch gradient descent
    - split dataset into small batches and performs updates for each of those batches
* Learning rate of gradient descent
  + Large: overshoot the minima, keep bouncing along the ridges of the valley
  + Small: stuck in sub-optimal outcomes, slow computation
    - Local minima: gradient is zero but not the lowest
    - Batch GD solves the problem
  + Increase learning rate
* perceptron rule
  + guaranteed to converge to a separating hyperplane *if the problem is linearly separable.*
* delta rule
  + single layer: only ONE global minimum
  + converge to the best SSE *whether the problem is linear separable or not*
  + could have a higher misclassification rate than with the perceptron rule and a less intuitive decision surface
* stopping criteria
  + max # of iterations
  + no improvement or oscillate over epochs

Support Vector Machine

* Some methods find a separating hyperplane, but not the optimal one: perceptron
* **Support vector machine (SVM)**: finds **an optimal solution**
  + the classifier is a separating hyperplane
  + **Support vectors**: Examples closest to the hyperplane, most important training points that define the hyperplane and most difficult to classifiy.
  + **Margin** : width of separation between support vectors of classes. Maximized by SVMs.
  + maximize the distance between the hyperplane (large margin classifier) and the difficult points close to decision boundary
  + if there are no points near the decision surface then there are no very uncertain classification decision.
  + fatter separator --> less choice --> capacity of the model has been decreased
  + most successful *text* classification method
  + solving SVMs is a ***quadratic*** programming problem
* Maximize margin: formalization
  + w: decision hyperplane *normal* vector
  + x\_i: data point i
  + y\_i: class (-1, +1), not 0,1
  + classifier**,** b is bias
  + functional margin of x: yi(wxi+b)
    - can increase margin by scaling w,b
  + functional margin of dataset is **twice the minimum functional margin (w) for any point**
    - 2 comes from the whole width of the margin
  + Training points appear inside inner products
* Geometric margin
  + Distance from example to the separator is
  + **A close up of a map

    Description automatically generated**
  + If all data is at least 1 from hyperplane,
  + For support vectors, inequality -> equality
  + Margin is
  + Algorithm: find w and b such that 
* **When the data is not linearly separable....**
* **Soft margin classification**
  + add slack variable (distance from misclassified points to class margin) to allow noisy examples
  + allow errors: move points to where they belong at a cost
  + minimize errors
* **Non-linear SVM**
  + General idea: use **kernel** to map original feature space into higher-dimensional feature space where the training set is separable
  + **Kernel function:** inner product in some expanded feature space
    - Why?
      * make non-separable problem separable
      * map data into better representational space (higher)
    - Common kernels
      * linear
      * polynomial (1+x)^d
      * radial basis e^x
        + Why not always RBF? Overfitting, expensive to compute
    - not useful in text classification
* Metrics for per class evaluation
  + recall: fraction of instances in class i classified correctly
  + precision: fraction of instances assigned class i that are actually in class i
  + accuracy: fraction of instances classified correctly
* Micro vs. Macro averaging
  + Micro: collect decisions for all classes, compute contingency table, evaluate (dominated by score on common classes)
  + Macro: compute performance for each class, then average
* Confusion matrix
  + entry (i,j) => instances actually in i are classified as j
  + in perfect classification, only the diagonal has non-zero entries
* **Real world scenarios: amount of training data**
  + None training data
    - handwritten rule (if statements)
    - 2 day per class, expensive
  + Very little training data:
    - Use **naïve bayes**: high bias algorithm with supervised classification
    - Semi-supervised learning: bootstrapping and EM with
    - Get more labeled data
  + A reaonable amount of data: good
    - **SVM, regularized logistic regression**
    - user-interpretable decision trees
  + A huge amount and growing training data
    - expensive methods like kNN and SVM are impractical
    - Use **Naïve Bayes**
* Accuracy as a function of data size
* **Logistic regression**
  + Binary classifier
  + Output the probability of a label between 0 and 1, can set a threshold to determine class
  + Linear regression – output actual value
  + Models data using the sigmoid or logistic function
  + Better capture outliers without skewing the model

KNN

* Introduction
  + instance-based learning (“lazy learning”)
    - no transformation of training data**, no model**
    - use a similar, related instances from memory to classify the new query instance
    - never form an explicit general hypothesis regarding the target function
  + kNN: simplest instance-based learning algorithm
    - define neighbors in distance
    - k: #neighbors considered
* Voronoi graph
  + all possible points within a sample’s Voronoi cell are the nearest neighboring points for that sample.
  + for any sample, the nearest sample is determined by the closest Voronoi cell edge
* Basic idea of KNN
  + classification rule: assign to a test sample the majority category label of its k nearest training samples
* Algorithm
  + For each training instance
    - Kronecker function: f(q) = argmax delta(v, f(xi)) (delta(a,b)=1 if a=b, 0 otherwise, xi are k neighbors near q)
    - Intuitively, assign the majority class among its k nearest neighbors
* Scale effects
  + **KNN is subject to scale effect**, bias the performance of the classifier
  + Standardization: transform raw feature values into z-scores
  + Normalization: normalized = (x-min(x))/(max(x)-min(x))
* Remarks
  + Noise tolerant
  + subject to curse of dimensionality (presence of many irrelevant attributes)
  + need adequate distance measure
  + relies on efficient indexing
* How is KNN incremental?
  + all training instances are stored (violate strict definition of incremental learning)
  + model consists of the set of training instances
  + adding new training instance only affects the computation of neighbors, which is done at execution time
* Advantages
  + No training needed. Immediate.
  + Useful for understanding structure of data
  + Robust
  + Work well on non-linear data
  + Datasets with multiple clusters do well
  + Work best in small datasets with many features and nonlinear
* Disadvantages
  + Memory expensive to store all data
  + Does not scale to product deployment
  + Heavily impacted by scaling issues
* How to choose parameters?
  + High value k -> smoothen decision boundary, avoid overfitting and improve generalization, low efficacy
  + k is usually *odd* to avoid ties
  + Weighing greater for closer datapoints
  + Euclidean and manhattan distance
* Balance dataset
  + Accuracy is a bad measure with imbalanced data
  + downsample
  + augment
  + weight balancing: use loss function to prioritize certain errors

Naïve Bayes

* Text classification: definition
  + input a document d and a fix set of classes
  + output a predicted class
* Classification methods: hand-coded rules (expensive maintenance, accuracy can be high)
* Classification methods: supervised machine learning
  + input a document, a fixed set of classes, and a training set of labeled documents
  + output a learned classifier
* Naïve Bayes: intuition
  + For a document d and a class c
  + Multinomial naïve bayes independence assumptions
    - Bag of words: position doesn’t matter
    - Conditional independence: feature probabilities P(x|c) are independent given in the class c
  + Learning NB model
    - Maximum likelihood estimates:
      * P(C) = #c class doc / all document.
      * P(w|C) = #w appeared in c/# all words in c
      * Problem: what if zero? --> laplace: add 1 to denominator and numerator
    - Assigning each word: P(word|C)
    - Assigned each sentence: P(sent|C) = product of P(word|C)
* How to choose a class?
  + Step 1: Calculate P(class)
  + Step 2: Calculate conditional probabilities
* Remark
  + fast, low storage requirements
  + **Robust to irrelevant features**
  + Very good in domains with many equally important features
  + Optimal if the independence assumptions hold
* More than two classes? – Sets of binary classifier
  + For each class c
    - build a classifier to distinguish c from all other classes
  + Given test doc d
    - evaluate d with every classifier
    - **d belongs to any class for which the classifier returns true (any-of)**
    - **OR**
    - **d belongs to one class with maximum score (one-of)**
* Development test sets and cross-validation
  + Unseen test set
    - avoid overfitting
    - more conservative estimate of performance’
  + Cross validation over multiple splits
    - handle sampling errors
    - pool results over each split
    - compute pooled dev set performance
* Log space: prevent underflow
  + Multiplying probabilities -> underflow -> log(xy)=logx+logy
  + model is now just max of sum of weights

Random Forest

* Decision Trees
  + non-linear classifier, easy
  + susceptible to overfitting but can be avoided
* Anatomy of a decision tree
  + Node: test on one attribute
  + Leaf: decision
  + Data gets smaller going down the decision tree
* Representation of a decision tree
  + A close up of a clock

    Description automatically generated
  + Y = ((A and B) or ((not A) and C))
    - A and B -> true
    - not A and C -> true
  + How to choose the split from representation?
    - Prefer the one that separates the training as much as possible
* **Information gain** to decide on the attribute to split
  + I(E) = -log\_2(P(x))
    - Coin flip: I = -log(1/2)
    - Dice=6: I = -log(1/6)
  + IG(X,Y) = H(X) – H(X|Y) --> information before split – information after split
    - reduction in uncertainty by knowing Y
* **Entropy**: expected amount of information
  + H(X) = E(I(X)) = -sum(p(x)\*log(p(x)))
    - 8 equally likely outcomes: H(x) = -sum(1/8\*log(1/8)) = 3
  + H(Y|X) = sum(p(x)\*H(Y|X=x))
  + Entropy measures purity
* Bagging (bootstrap aggregation)
  + reduce variance of an estimated prediction function
  + Basic idea
    - randomly draw datasets with replacement from the training data
    - each sample is the same size as the original training set
    - construct decision trees on each bootstrap sample
    - take the majority vote
* Random forest classifier
  + an extension to bagging which uses *de-correlated* trees
  + create bootstrap samples from training data
  + construct decision for each sample
  + at each node in choosing the split feature, **choose only among m < M features**
  + take majority vote

Deep learning

* What is deep learning?
  + ML: human-designed features
  + DL: learning representations of data
    - effective at learning patterns by using hierarchy of multiple layers
  + Why is DL useful?
    - Easy to adapt, flexible, universal
    - Can learn both supervised and unsupervised
    - End-to-end joint system
    - Large amount of training data
* Neural network
  + Sample labeled data -> forward it through network, get predictions -> back propagate the errors -> update network weights
    - optimize objective/cost function
    - generate error signal that measures difference -> change weights
    - subtract a fraction of the gradient moves you towards the local minimum of the cost function (gradient descent)
  + Activation function
    - Purpose
      * non-linearities needed to learn complex representations of data
      * more layers and neurons -> more complex functions
    - Sigmoid
      * Takes a real-valued number -> range between 0 to 1
      * saturate and **kill** gradients, thus NN will barely learn -> no signal flow to weights
    - Tanh
      * Takes a real-valued number -> range between -1 to 1
      * Output is zero-centered
      * Scaled sigmod
      * saturate and **kill** gradients,
    - ReLU
      * f(x) = max(0,x)
      * Faster, linear, non-saturating, no gradient vanishing, cheaper, expressive
  + Overfitting
    - Model fits training data and its noise but fail to generalize to test data
    - Regularization to solve overfitting
    - High variance
  + **Regularization**
    - Dropout
      * randomly drop units during training (set a portion of outputs of nodes to zero)
      * hyper-parameter
    - L2 = weight decay
      * introduce regularization term that penalizes big weights, added to the objective function
      * weight decay value determines how dominant regularization is during gradient computation
      * bigger penalty for bigger weights
    - Early-stopping
      * validation error to decide when to stop training
      * stop when not improving after n subsequent epochs (n is patience)
* Convolutional neural networks
  + Main CNN idea for text: compute n-grams and group them afterwards
  + input matrix + convolutional filter -> feature map -> max pooling
* Recurrent neural networks
  + Main RNN idea for text: condition on all previous words, use **same set of weights** at all time steps
  + Vanishing gradient
* Bidirectional RNNs
  + Main bRNN idea for text: incorporate left and right context, output may not only depend on previous elements in the sequence but also future elements
  + Two RNNs, output based on hidden state of both RNNs
* Gated Recurrent Units (GRUs)
  + Keep around memory to capture long dependencies
  + Allow error messages to flow at different strengths depending on the inputs
  + Standard RNN computes hidden layer at next time step directly
  + GRU computes **an updated gate** z based on current **input word vector AND hidden state**
  + GRU computes **a reset gate** r with different weights
  + if z close to 1, copy information in that unit for many steps
  + if r close to 0, ignore previous hidden state (drop information that is irrelevant in the future)
* Attention

Reinforcement learning

* Do you know your environment?
  + Yes -> policy/value iteration, dp
  + No -> reinforcement learning
* Types of RL
  + search-based: evolution directly on a policy, genetic algorithms
  + model-based: build a model of environment -> DP, memery-intensive learning
  + model-free: learn a policy without a model. Limited episodic memory
    - actor-critic learning: temporal difference version of policy iteration
    - q-learning: td version of value iteration. most widely used.
* Solves a problem where decision making is sequential, goal is long-term
  + game playing, robotics, resource management, logistics
* Modeled in terms of states, actions, and rewards
  + **state**: given condition an agent evaluates its position in
  + **action**: the agent at a state can choose from a set of actions which may fetch different rewards or penalties
  + agent maximize these rewards to behave optimally at any state
* Q-learning

Ensembles

* Learn a good **set** of models
* Ensembles of neural networks (or any supervised)
  + 5-10 accuracy gains
  + combine classifiers of various types
* Combining multiple models: 3 ideas
  + simple
  + weighted
  + training a combing function (prone to overfitting)
* Produce uncorrelated members of an ensemble
  + **Bagging**: k times randomly choose with replacement N samples from a training set of size NsaZDS
  + **Adaboosting**: reweight examples each cycle (increase if wrong)
  + Directly optimize accuracy + diversity
  + Different number of hidden units in a NN, different k in kNN, tie-breaking scheme, example ordering...
* Variance/diversity creating methods
  + Train with different associated tasks (multi-task learning)
  + Use different input features
  + Assign each category an error correcting code, train on each bit separately, want large hamming distance between rows and columns
* Random forest
* Adaboosting
* dealing with weighted examples
* Boosting and overfitting
* Bagging vs. Boosting
  + Bagging > single D-tree and ANN
  + Boosting > Bagging
  + Boosting: too much emphasis on outliers
* Always consider bagging or boosting – easy to use and highly effect8ive
* reduce comprehensibility of models
* increase in runtime, but cycles cheaper than examples

Clustering

* Incentives
  + real estate agent: 5-6%, push you to sell.
  + car seats increase safety. How to gather data for 2-4? Need machine learning model that takes into account of kid’s height and weight
  + wrestlers with a 7-7 record, 50% more 7-7, 80% more 8-6, 73% more 9-5
* Supervised learning
  + regression
  + classification
* Unsupervised learning
  + no labels
  + clustering is a method of unsupervised learning
* Clustering
  + Key idea: instead of assigning label and grouping, first group and then assign a label
  + taxonomy in biology is clustering; A branch of science that encompasses the description, identification, nomenclature and classification of organisms.
  + A form of information reduction/organization
  + To store in human finite memory or computers finite memory and facilitate understanding
  + to communicate between people for more effective understand
* **K-means**
  + initialize k-means randomly
  + categorize each item to closest mean
  + update mean’s coordinates
  + after one epoch, recalculate the distance between every point
  + reassign the point if needed, also update the centroid
  + converge
* **Metrics**
  + Homogeneity score (precision): how uniform a cluster is (1: all data points in a cluster belong to the same class)
  + Completeness score (recall): how comprehensive a cluster is in capturing elements from a single class (1: all data points in a given class are in the same cluster)
  + V-measure (f1) : harmonic average of h and c
  + Adjusted rand index (accuracy): similarity between two clusterings (between 1 and -1), adjusted corrected random chance clusterings
  + Adjusted mutual information: also accuracy, mutual information between distributions
* Ames Data Housing, 1460 observations and 79 explanatory variables
  + QPR-V
* How do we computer cluster
  + MST simple algrothm: sort distance between 2 points from min to max,
  + feature based clustering
  + K-means
    - algorithm
    - guaranteed convergence
  + Expectation-maximization
  + spectral methods(PCA)
  + Hierarchical clustering: feature based or not
* Principal component analysis (PCA)
  + dimensionality reduction
* Summary
  + Clustering: information organization
  + New name: PQRS
  + K-means clustering
    - sensitive to outliers
    - using medians corresponds to absolute loss function,
    - robust
  + we do not always have labels to compare
* Midterm
  + perceptron
  + KNN
  + project
  + homework 3 movie
  + Shocking headline
    - Coffee is good/bad?
    - Small sample size
    - Biased sample, focus on one perspective of data --> inconclusive
    - 60-70% are wrong and self-serving and does not provide enough information
    - Need to be critical
    - Simple solution!
  + Case: Death of labor: 10% in one clinic, 3% in the other clinic
    - puerperal fever – doctors don’t wash hands
    - washing hands have no relationship with infection
    - need educational ads
  + Case: Global warming
    - Idea: create sulfur dioxide to pollute air and cool temperature
    - Not true, why?
    - How to collect data?
    - For certain types, electric cars are good, some are bad.
  + Probability, random variables, and distributions
    - Deck of cards
    - conditional probability: bayes theorem
      * P(A|B) = P(A and B)/P(B)
    - independence: P(A|B) = P(A)
  + Design
    - divide into geographic regions
    - do clustering on income of people
    - hierarchical clustering: on gender, on kmeans
  + Decide which ML algorithm to use and why?
    - Linear separable: not perceptron, SVM
    - Not linear separable, circle: NO linear classifier, clustering
    - Not labeled: unsupervised learning. Use K-Means to cluster and know information