Machine Learning Study Guide

**Decision Trees**

- Nodes and edges
- Find a hypothesis such that it approximates the target function
- DT can express any function of the input attributes
- DT overfit as they get deeper, prefer more compact trees
- If splitting on continuous feature, internal nodes can test the value against a threshold (more or less than 10)
- DT create hyperspace into parallel hyper-rectangles with each rectangle labeled with one label
- DT wants splits that result in nodes with attributes only from one class
- Entropy measures the skew of each "bin" from a split, basically the proportion of each class from the split
- Entropy is highest when the skew is closest to a 50/50 split, ranging in values from 0 to 1.
- We can weight entropy to calculate reductions better and consider them more
- Information Gain measures how important a given attribute is and helps decide the ordering of attributes in the node of a decision tree
- IG = entropy of parent minus the weighted entropy of the children
- From IG we use ID3
    - Recursive greedy algorithm to build the tree
    - Picks best variable split on data
    - Makes optimal choice at current step without considering future steps
    - Can run into trouble with multiple variables (XOR, nonlinear)
- RULE OF ML
    - You can't learn anything without inductive bias
    - The best hypothesis almost never achieves 100% accuracy on training data
- Hume's Problem of Induction
    - Let us know that h ~ f
    - Try h on new set of test examples (using same distribution space as training)
    - Learning curve = % correct on test set as a function of the training set size. (i.e. as you get more training data the model generalizes better and therefore performs better on the test data)
    - Learning curve depends on realizability (ability to generalize), if non-realizable then we may be missing attributes or have a restricted hypothesis class or redundant expressiveness (lots of irrelevant attributes)
- Overfitting
    - If most DT leaves have a single example each, most likely too specific
    - How to Stop:
        - Early Stopping
            - Build the tree with some heuristic that stops the growth before overfitting
            - EX:

            - Min number in node
            - Min decreases in impurity
            - Max Depth
        - Pruning
            - Build tree to allow overfitting, then cut back nodes to remove overfitting
        - Random Forests
            - Collection of trees where each is a weak classifier
            - Each tree has a random subset (with replacement) of training examples
            - Random subset of the features of the training data
            - Decision made by majority vote
- DT Pros and Cons
    - Pros
        - Easy to interpret
        - Computationally inexpensive
    - Cons
        - Tend to overfit
        - Can be unstable, sensitive to noise

**Measuring Distance**

- Metric
    - Function that is:
        - Reflexive: 0 iff x=y
        - Non-Negative
        - Symmetric
        - Triangle Ineq
- Norm
    - Positive vector
- A vector Norm
    - Function that assigns a strictly positive value to all vectors in a space
- A normal vector
    - Perpendicular to another object
- Every norm determines a metric
- Some metrics determine a norm IF the metric is on a vector space
- Distances
    - Euclidean
        - Straight line distance
        - P=2
    - Manhattan Distance
        - Grid measure
        - P=1

- Lp Norms
    - L1 norm = Manhattan distance (p=1)
    - L2 norm = Euclidean distance (p=2)
    - Hamming Distance (p=1_ and x,y exists between 0 and 1
- Weighted norms
    - Weighting dimensions
        - Turn circular clusters into eclipses with axes parallel to the dimensions of the vectors
        - Mahalanobis distance
            - Ellipsoid around the mean of a distribution
            - Axes don’t have to be parallel to the dimensions describing vector
            - Utilizes covariance to calculate (similar to Euclidean calculation)
            - Good for non-spherical symmetric distributions
            - Accounts for scaling of axes
            - Can reduce to Euclidean
- Distance between categorical variables
    - Hamming Distance
        - Number of bits different between binary vectors
    - Edit Distance for how different words are (levenshtein)
- Cosine Similarity
    - Cosine distance: angle in between two vectors (degrees)
    - A.B / Mag(A)Mag(B)
    - Acute angle = similar
    - 90 degrees means orthogonal (perpendicular)
    - Obtuse (close to 180) = opposite
- KL Divergence
    - Non-symmetric measure of difference between two probability distributions
    - Not a metric
    - Sum of the natural log of prob P divided by prob Q times prob P for a given point i
    - Also is shown as cross entropy – entropy

**KNN**

- Nearest Neighbor is:
    - Classification and regression
    - Lazy learner
    - Non-parametric
    - Memory-based
- Eager Learning
    - Learn model ahead of time from training data
    - Learn **h** from training data

    - EX: decision tree, linear regression, svm, neural nets
- Lazy Learner
    - Delay the learning process until a query example must be labeled
    - **H** is implicitly learned from the training data
    - Nearest neighbor, kNN, locally weighted regression
- Parametric Function
    - Model that summarizes data using a fixed set of parameters
    - Parameters learned during training
- For kNN
    - Learning = memorizing
    - Store training examples, at prediction time, compare query to previous examples
- What makes a memory learner?
    - Distance measure
        - kNN is typically Euclidean
    - Number of neighbors
        - k in kNN
    - Weighting function
        - Unused for nearest neighbors
    - How to fit with the neighbor
        - Same output as nearest neighbor
        - Majority vote of k nearest neighbors
        - Classification: mode of k nearest neighbors
        - Regression: Avg output of k nearest neighbors
- How to choose K?
    - If too small
        - Can fit to the noise (overfit)
    - If too large
        - Can make decision boundaries indistinct (underfit)
- Voronoi Diagram is 1-NN decision boundary
- Kernel Regression
    - Distance
        - Scaled Euclidean
    - Number of neighbors
        - All of them
    - Weighting Function
        - Nearby points to the query are weighted strongly, far points weakly
    - How to fit with the neighbor
        - Weighted average
    - Gets smoother lines
- Pros and Cons
    - Pros
        - No train time
        - Adapts to new training data
        - Robust to noise

        - Effective when training data is sufficient
    - Cons
        - Can be fooled by irrelevant or redundant features
        - Computationally expensive
        - Evaluation time grows with training size
        - Lots of storage needed

**Clustering**

- K-Means
    - Take distance from k means, closest points are with each mean
    - Take mean of each "cluster", then move means and recalculate distances
    - If labels don't change, stop
- Unpacking the math
    - Update Labels
        - For every point and every mean
            - Get distance
            - Label the point with index of mean closest to point
    - Update Means
        - For each mean
            - For all points closest to mean, sum values
            - Divided by number of points assigned to the mean
- Soft K-Means
    - Proportion/Weighting of how close each point is to each mean
    - Math
        - Update Labels
            - For each mean, raise *e* to the distance to the point
            - Divide by the sum of *e* to the distance between the point and all of the means
            - Puts the "distance" labeling between 0 and 1
            - *e* to the negative B multiplied by the Euclidean distance
        - Creates Gaussian distribution
        - As softmax approaches 1, the distance between the mean and the point approaches 0
        - As B increases, the distribution tightens, low distances get really big while high distances go to 0
        - The higher the B the closer to hard k-means we get
- K-Means iterative process is called Expectation Maximization (EM)

**Hypothesis Testing**

- Classifier (DT)
    - Count how often it is wrong
- Regressor (LR)
    - Distance between predicted and observed values
- Probability Mass Estimator
    - Pick distribution that maximizes likelihood of data
- Ranker (search engine)
    - Rank order?
- True Error
    - Of hypothesis with respect to the target function and full data distribution
    - Probability hypothesis will misclassify a random data point according to the distribution
- Sample Error
    - Of hypothesis with respect to target function and a data sample set
    - Proportion of examples in the sample that the hypothesis misclassifies
- Problems with estimating error:
    - BIAS
        - Having 100% accuracy in a sample, but misclassifications elsewhere
    - VARIANCE
        - Lots of positives in one sample, lots of negatives in another
    - Why not take bigger sample?
        - From one mean you can't tell how the sample error varies from the true error
- Bernoulli Trial
    - Experiment whose outcome is random
    - Set of random variables is independent and identically distributed if all variables are mutually independent and all under same probability distribution
- As the number of iterations goes to infinity, the normal distribution approximates the binomial distribution if you set STD and Mean correctly
- CENTRAL LIMIT THEOREM
    - For large numbers of iterations/data that are IID from same distribution, the sample average distribution is approximated by the normal distribution
    - It approaches a normal distribution regardless of the shape of the distribution governing the random samples
    - Normal distribution lets us estimate how close the true error is to the sample error
- Rule of thumb: have at least 30 IID trials
- Confidence Intervals: Estimating a value
    - Pick a parameter to estimate (true error)
    - Choose an estimator (sample error)
    - Determine the probability distribution governing (governed by binomial, approximated by normal when n>30)

    - Find the interval such that N% of the probability mass falls in that interval
    - Mass within 2 standard distributions from the mean
- Cross Validation
    - We know this
- T-tests
    - Commonly used
    - Assumes normally distributed data
    - One sample
        - Is a population mean different from the mean of a sample
        - 30 neural nets get 0.2 error rate, but ID3 gets 0.3 error rate, is the neural net rate a fluke?
    - Independent samples
        - Are means of two normally distributed populations equal
        - Does eating ice cream make you heavier? Weight 1000 of them, give each an ice cream cone, weigh again
    - Paired samples
        - Is 0 the mean difference between paired responses measured on same data
        - Take two things on same data, run the experiment at least 30 times using half data to train then half to test, then run experiment for other thing, compare errors
    - Pitfalls
        - Data not normal
        - Not enough sample points
        - Using a paired samples test when samples aren't paired
        - Using students independent when the variances of two sets are different (use Welch's instead)

**Gaussian Mixture Models**

- Discriminative vs Generative
    - Discriminative
        - Learn decision boundary between your sets
        - Decision Trees
    - Generative
        - Learn enough about sets to make new examples
        - GMMs
- Generative Models
    - Assume data was generated from a process we can model as a probability distribution
    - Learn the probability distribution
- GMMs
    - Start with unlabeled dataset
    - Learn underlying distribution

    - Use model to generate new data points
    - Uses Expectation Maximization
    - Goal is to find the best Gaussian
    - Find parameters that maximize the probability of observing the data
    - Use Log probabilities to prevent underflow
- Maximizing Log-Likelihood
    - To find best parameters, take partial derivative with respect to parameters and set to 0
    - Result is closed form
- We can underfit if one normal distribution is too simple to fit
- GMM part 2
    - Model distribution as a mix of Gaussians
    - Optimize the weight, standard deviation, and mean
    - No closed form
    - Solution is EM
        - Updates parameters iteratively
        - Always converges to local minimum
    - Steps
        - Initialize parameters
        - E Step: calculate likelihood a model with the parameters generated the data
        - M Step: Update parameters to increase the likelihood from E step
        - Repeat E and M until convergence at local optimum
    - Covariance Matrix
        - Allows us to work with multi-dimensional data
        - Describes the shape and orientation of an ellipse

**Linear Regression**

- Why LR?
    - Easily understood
    - Interpretable
    - Well studied
    - Computationally efficient
- Assumptions
    - Observed response (y) is the true function (mx) with additive Gaussian noise (b) with a 0 mean
    - Expected value of y is a linear combination of the k independent attributes/features
- Hypothesis Space
    - W0 + w1x1 + w2x2 + …
    - W0 is offset from origin
    - Goal is to learn a k+1 dimensional vector of weights that define a hyperplane minimizing error

- Error Criterion
    - Sum of squared residuals (RSS)
        - Aka Sum of Squared Errors (SSE)
        - Sum of difference between true y values and target function squared
- Multivariate Linear Regression
    - Goes past w0 + w1x1
- Closed Form Solution
    - W = (XTX)-1 XTy
    - RSS(w) = (y-Xw)T (y-Xw)
    - Presupposes XTX is invertible
    - If it isn't invertible:
        - Dimensionality Reduction
            - Make every column of X independent
            - Add small amount of random noise
            - Or dimensionality reduction to get rid of redundant columns

**Polynomial Regression**

- H(x) = w0 + w1x + w2x^2 + …
- More dimensions means more complexity
- Too high of a degree causes overfitting
- Bias and Variance
    - As complexity increases:
        - Bias decreases (better fit to data)
        - Variance increases (fit varies more with data)
    - Bias
        - Measures how much h(x) is wrong disregarding the effect of varying examples
    - Variance
        - Measures how much h(x) fluctuates around the expected value as the sample varies
    - Coefficient of determination
        - R^2
        - Indicates how well data points fit a line or curve
        - Closer to 1 the better
        - 1 – E(RSS)

**Gradient Descent/Regularization**

- If loss function and hypothesis are differentiable we can evaluate the gradient to find where the loss increases the fastest (so we can find the opposite direction)
- Pseudocode
    - O(t+1) = O(t) – nV(O(t)) L(X,Y;O(t))
    - O(t+1) = parameters at next step
    - O(t) = parameters at current step

    - n = learning rate
    - V = gradient of O(t)
    - L(…) = Loss function for X and Y for parameters of O(t)
- Design Choices (what we can change)
    - How O is initialized
        - 0
        - Random values
        - If bad
            - Convergence to local minimum
            - No way to determine if converged to global minimum
    - Convergence criteria
    - Choosing loss function
    - How much data is used at each step
    - Learning rate
- Knowing when to stop
    - When gradient is close to 0 (reaching minimum)
    - Stop after number of iterations
    - Stop when loss on validation step stops decreasing
- Loss functions
    - Required
        - L(…) >= 0
        - Decreases as performance improves
    - Required for GD
        - Differentiable with respect to O
    - Helpful for GD
        - Gradient of L is bounded between 0 and infinity
    - 0-1 Loss
        - Loss = 1 if y != h, else 0
        - Count of mislabeled items
        - Outputs step function
        - Not useful for GD
    - Squared Loss
        - L(…) = 1/2N * E(y-h(x))^2
        - If linear then:
            - H(x) = OT x
    - Hinge Loss
        - Loss only happens if data is on wrong side of the line
- Batches
    - Batch GD
        - Consider all data
        - Most accurate
        - Slow
        - Not possible if |D| > RAM
    - Stochastic GD

        - Consider one random training sample at each iteration
        - Noisy, inaccurate
        - FAST
        - Random shuffling is important
    - Mini-batch GD
        - Random subsets of the data
        - Most common
        - Balances speed and accuracy
        - Random shuffling ☺
        - Want batch size as big as possible
    - Different data = different loss
        - Probably won't know what changed
- Regularization
    - Overfitting occurs when model starts memorizing
    - Regularization is most common way to fight overfitting
    - Occam's Razor
        - Given two models with equal performance, the simpler model is preferred
    - Regularization is applied to any loss function
    - Amount of regularization is controlled by gamma
    - Penalizes model for being complex
    - Penalty assigned based on Lp Norm
    - How we measure determines what is penalized the most
    - L1
        - For L1 we are penalizing the absolute value of parameter weights
            - We end up with sparse vectors due to linear relationship
        - Form of feature selection
        - Only features with non-zero coefficients contribute to prediction
        - Gradient moves model parameters towards 0 at constant rate
        - Gradient bounded between -1 and 1
        - Function looks like a V, very absolute value like
    - For L2 we penalize small values less than large values, ensures we do not have large parameter values
- Multi-Class
    - One v One
        - Train binary classifiers between each pair of classes
    - One v All
        - Binary classifier for data that is the current label or not
        - Predict based on highest confidence score (i.e. regression output)

**Logistic Regression**

- Discriminative
    - Models posterior probabilities
- Maps logit scores to probability and a valid probability distribution over the two classes
- Differentiable error function (cross entropy)
- Works well when classes are linearly separable

**Perceptrons**

- Understand the difference between pre-activation values (dot product of input and weights plus bias)
- Z values go to activation function (step function) and outputs the activation or a values
- Primary contributions were the learning algorithm
- Perceptron Update Rule
    - A w = n(t – o) x
    - Aw is updated weight
    - n = learning rate
    - t = target value
    - o = perceptron output
    - x = input value
    - Activation Function = Not differentiable
    - Step Function = Non-Linear
- Combining Perceptrons
    - Original Perceptron algorithm only involved in one layer
    - Multiple layers can represent nonlinear decision boundaries
    - Loss is non-differentiable so no way of learning
- ADALINE
    - Remove step function to make linearly continuous
    - Sucks because then everything is stuck in linear
    - Differentiable though
- Perceptron v ADALINE
    - Predictive power of ADALINE much less than original perceptron
    - Both can only capture linear decision boundaries, but the addition of layers no longer adds expressiveness to our model with linear activation
- SIGMOID
    - Differentiable and non linear need I say more

**Neural Nets**

- Network of perceptrons
- Feed-Forward = Information only moves forward
- Dense = every neuron is connected to all neurons in next layer
- Input, hidden, output layers
- Layers create a composition of functions f3(f2(f1(x)))
- Universal Function Approximator

        - H(x) ~ f(x)
        - Given enough hidden units, a neural network can approximate any function with bounded error (assumes non linear activation function)
        - Arbitrary expressiveness in our decision boundary
        - In reality, limited by finite network and data
- Logits linearly separate classes in final latent space
- Multiclass uses softmax probabilities to determine class
- Activation Functions
    - Sigmoid
        - Standard
    - ReLU
        - Helps with vanishing gradients
- Cost Functions
    - Regression uses MSE
    - Binary Classification uses two-class cross entropy
    - Multi-Class uses multi-class cross-entropy
- Backpropagation
    - Error vector at l =
        - Weight matrix between l to l + 1 times error vector of next layer
        - Parameter multiplied to element-wise derivative of the activation function

**Language Models**

- Vocabulary = set of unique tokens a model knows
- Uses chain rule to consider each previous token to predict next
- Uses log probabilities to avoid underflow
- Allow us to predict sequence of words by modeling the probability of observing specific sequences of tokens
- Take sequence of words and predict how likely it would be observed
- Makes predictions by choosing next word that maximizes probability of observation
- Understand perplexity and that it is common way of evaluating how well an LM is performing
    - Exponential of cross entropy
    - Represents how surprised we are to see a certain token (lower is better)
- N-gram
    - Break up sequences into n tokens to understand longer context sets
- **Bengio Model**
    - Fixed context window like n-grams
    - Introduces embeddings
    - Why does it work?
        - Distributional Hypothesis
            - Similar roles (semantically and syntactically)
            - Words that occur in the same contexts tend to have similar meanings
    - 3 layers of computations, know what is happening at each layer

    - Fixed context window, one hot vectors multiplied by C value (embedding matrix), outputs word embedding, concatenate one after another, that is input to the model (x)
    - Once we have x, tanh is vanilla neural net that we use, also have skip connection that takes hidden state and raw word embeddings and multiplying by weight matrix which makes logits, softmax, probability distribution, determine which is best
    - The skip connection allows the model to remember the original word embedding context to ensure that the initial considerations of the word were not fully discarded in the neural net
    - Network parameters and word embeddings are learned at the same time
    - Pros and Cons
        - Pros
            - Neural Nets can outperform count based n-gram models
            - Learned word embeddings
            - Generalization to unseen contexts through continuous representations
        - Cons
            - Fixed context window
            - Cannot capture long range dependencies
            - Number of parameters grows with window size
- **Embedding Spaces *****
    - Understand distributional hypothesis
        - Words in similar context have similar meanings
    - Similar words will be closer together within a space
    - With LMs, we are taking embedding space and looking at output of last hidden layer, we get a position in the space, then measure the distance between that output and every vector in the space which allows for computing logits that go to softmax

**Transformers/ RNNs**

- RNNs
    - Process sequences one step at a time where each step the model takes current input and previous hidden state, then makes new hidden state which is passed forward
    - Pros and Cons
        - Pros
            - Handles arbitrary length sequences
            - Uses a recurrent hidden state ro summarize prior context
            - Shares weights across time which reduces parameters
        - Cons
            - Vanishing/exploding gradients make learning long range dependencies difficult

        - Hidden state mixes short term and long term information
        - Sequential nature = slow
        - Instability with longer sequences
- LSTMs
    - RNNs but now hidden layer things for short term and long term context