2016-01-15, CS7641 (Machine Learning)

Lectures

- Textbook: "Machine Learning" by Tom Mitchell (1997)
- "computationally-applied statistics"
- Three mini-courses in this class:
  - Supervised learning - labeled inputs/outputs
  - Unsupervised learning
  - Reinforcement learning

Though early AI was.

Primary problem in ML is generalization.

- "inductive bias" - ML is about induction, not deduction
- In supervised learning we are trying to approximate some function given labeled inputs/outputs
- In unsupervised learning we really just have a pile of inputs and we're trying to describe, summarize, categorize more concisely.
- Reinforcement learning (Michael & Charley specialize in this):
  - "Learning from delayed reward"
  - Feedback may come many steps after decision.
  - In some sense, it's harder - no one is telling you what to do, just giving vague/delayed good/bad signals
- All these can be formulated as some kind of optimization problem.
- "Data is king in machine learning"

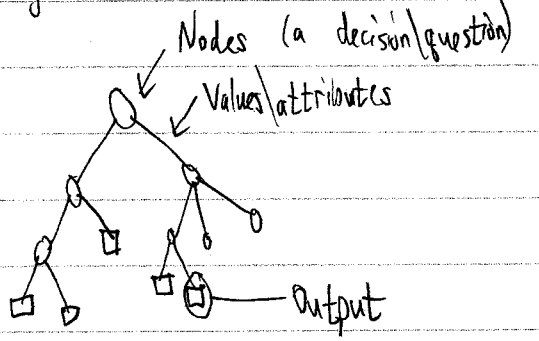
SL1

- Supervised learning, two main types:
  - Classification - Mapping input X to some discrete label
  - Regression - More about continuous-valued functions
- e.g. classification of a picture of a person as male or female

→ Discrete output (incl. binary)

→ Continuous output

- Classification vs. Regression is about output, not input.
- Classification learning
  - Instances - Vectors of values giving your input.
  - Concept - The function mapping input to output ('concept' in the sense that - for instance if we have a function that determines male/female, that function is expressing the concept of maleness/femaleness)
  - Target concept - The thing we're trying to find; the 'answer'
  - Hypothesis class - Set of all concepts we're willing to entertain
  - Sample - Training set; set of all inputs paired with correct output. We often need lots of this...
  - Candidate - The concept we think is the target concept.
  - Testing set - Looks like training set - but the result of applying the candidate to some input.
- Do not make your training & testing set the same! Testing should be on novel input!
- Decision trees
  - Representation vs. algorithm
  - It is a particular representation
    - Leaves of tree = answers



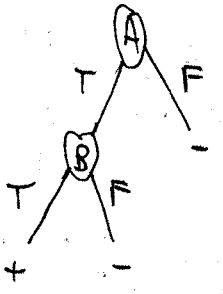
  - Think '20 Questions': Start with questions that narrow possibilities the most, and keep narrowing the possibilities. This depends on results of earlier questions

To build decision tree:

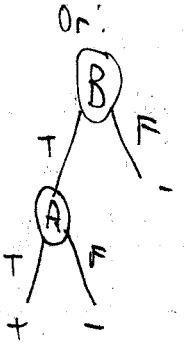
- Or: 1. Pick best attribute (best = splits the data)
  2. Ask question
  3. Follow answer path
  4. Go to 1 until answer is found

2016-01-15b, CS7641

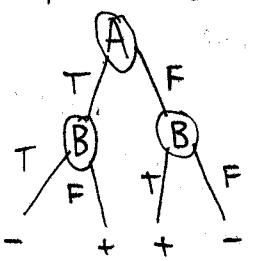
- e.g. Boolean AND
  - Automatically know result when either one is false
  - Flip T & F, + & −, and you get Boolean OR



Or:

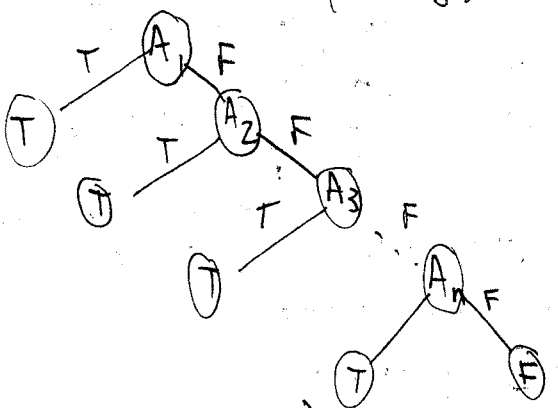


- Boolean XOR:



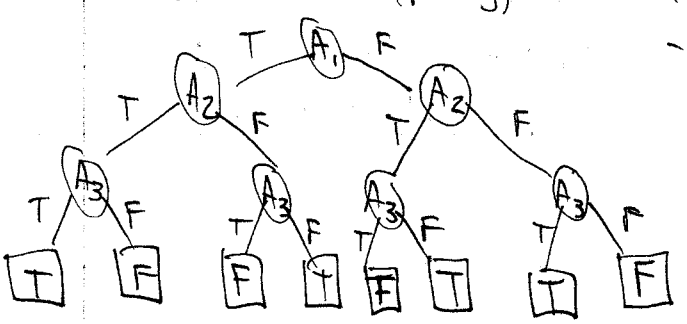
  - We must split on B in all cases
  - This is just another form of full truth table
  - We could write AND/OR thus, but don't need to.

- What about n-OR (or 'any')?



  - This requires n nodes.
  - This tree is then linear. ('Easy')

- And n-XOR (parity)?



  - That is, odd # true = true (odd parity)
  - Note the recursive structure (lots of embedded XOR)
  - Needs $2^n - 1$ nodes $\approx O(2^n)$ – Exponential # of nodes. ('Hard', 'Evil')

- How expressive is a decision tree?
  - Consider 'hard' problem like n-XOR. How many trees, for a given n, can express n-XOR?
    - You can branch on any of n inputs for 1st node, n-1 for 2nd, n-2 for 3rd... n! different trees?
  - But for an n-variable Boolean expression, 2^n rows in truth table - and 2^{2^n} possible outputs

This is a very large hypothesis space to search, so we must be clever in how we search.

- ID3 algorithm, loop: (top-down learning algorithm)
  - A ← best attribute
  - Assign A as decision attribute for Node
  - For each value of A, create descendant of Node
  - Sort training examples to leaves
  - If examples perfectly classified, stop, else, iterate over leaves.
- 'Best' attribute: One way is 'information gain', which is really a reduction in randomness: (from picking one attribute)

$$Gain(S,A) = Entropy(S) - \sum_v \frac{|S_v|}{|S|} Entropy(S_v)$$

S → collection of training examples; A → particular attribute; $\frac{|S_v|}{|S|} Entropy(S_v)$ → expected/average entropy over each value/leaf

We want max gain.

  - 'Entropy' here is a randomness measure
  - Actual formula: $-\sum_v p(v) \log p(v)$ (p(v) → probability of seeing value 'v')

We may compute these by counting up the labels in each node

  - If an attribute effectively splits up all our labels, that's an entropy of 0; it would be a 'best choice

2016-01-15c, CS764/

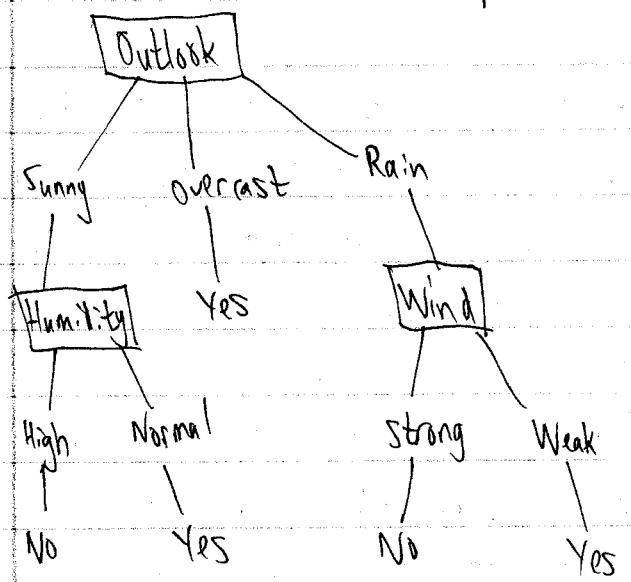
- ID3: Bias (how we restrict trees we're looking at)
  - Restriction bias: The hypothesis set we actually care about. (in our case: all decision trees, not other functions)
  - Preference bias: What hypotheses do we 'prefer' out of that set?
  - → Inductive bias
- So what is ID3's inductive bias?
  - Prefers 'good' splits at top
  - Prefers 'correct' trees to 'incorrect' ones
  - Prefers shorter trees to longer
- Now, what if we have continuous attributes? (e.g. age, weight, distance)
  - We could branch on specific values, or break it up into ranges
- When do we stop? When everything's classified correctly. But what if data's noisy and will never classify perfectly? Or, if we don't want to classify noisy data perfectly and have decision tree overfit?
  - A tree that's too big tends to overfit.
  - Cross-validation is one way (split up training set, and train on one part & validate on the other).
    - Or: Split up training set. Train on part. Every time it attempts to expand your tree, validate against a new part of the training set and only expand if error is below some threshold
      - Likely want to expand breadth-first
    - Or: Prune a tree later (check error in validation set if we collapse tree)

- Regression (vs. classification) - continuous outputs?

2016-01-19, CS7641

- Mitchell, Ch. 1 notes
  - p2, their definition of learning: "A computer program is said to learn from experience E with respect to some class of tasks T & performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."
  - A well-defined learning problem must specify E, T, & P!
  - This encompasses some things too we may not normally call 'learning', like a database system that lets users update data - and improves its performance at answering DB queries based on this.
  - Training — Direct Feedback - e.g. a system learning to play checkers, & being told specific board positions and the 'correct' move from there.
    Indirect Feedback - e.g. same system only has access to move sequences & final game outcome.
    - Indirect involves credit assignment - how much did each move contribute to a win or a loss?
      - Can be very difficult
    - To what degree does 'learner' control training examples?
    - How well does the training represent the distribution of examples that will determine P's measurement?

- Ch. 3 (Decision Tree Learning)
  - "In general, decision trees represent a <u>disjunction</u> of <u>conjunctions</u> of constraints on the attribute values of instances."
  - "Each path from the tree root to a leaf corresponds to a <u>conjunction</u> of attribute tests"
  - "The tree itself corresponds to a <u>disjunction</u> of these conjunctions."



$\equiv$ (Outlook = Sunny $\wedge$ Humidity = Normal)
$\vee$ (Outlook = Overcast)
$\vee$ (Outlook = Rain $\wedge$ Wind = Weak)

- p58: ID3 algorithm

2016-01-[illegible], CS7641

SL2

- Regression
  - Recall that supervised learning tries to predict outputs of new inputs, given examples of inputs & outputs.
  - Regression is concerned w/ mapping continous inputs & outputs.
  - "regression to the mean"
    → 'regression' as using a functional form to approximate a bunch of data points (yes, this has almost nothing to do with other 'regression')

"used wrong, but it's stuck". Reinforcement learning also is this.

  - Finding best constant function:

$$f(x) = c, \quad E(c) = \sum_{i=1}^{n} (y_i - c)^2$$ ← squared for various reasons, partly because it's well-behaved

$$d/dc\, E(c) = \sum_{i=1}^{n} 2(y_i - c)(-1) = 0 \rightarrow \text{Minimum}$$

$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} c = 0, \quad \sum_{i=1}^{n} y_i - nc = 0 \rightarrow c = \sum_{i=1}^{n} \frac{y_i}{n}$$ - which is the mean of y!

  - So here is one way mean comes up!
- That was 0th-order ($k=0$); $k=1$ is line, $k=2$ is parabola.
  $f(x) = c_0 + c_1 x + c_2 x^2$ for that.
  - N.B. If a parabola did not exactly fit and a line fit better, we'd just get $c_2 = 0$ in regression!
- We can use much higher-order too but this has its own undesired side effects. Also, diminishing returns with reducing error.

- Polynomial regression, $k=3$:

$$c_0 + c_1 x + c_2 x^2 + c_3 x^3 = y$$

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ & & \vdots & \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

X may not have nice inverse, but $X^T X$ will.

$$\rightarrow \quad Xw = Y$$

$$X^T X w = X^T Y$$

$$(X^T X)^{-1} X^T X w = (X^T X)^{-1} X^T Y$$

$$\boxed{w = (X^T X)^{-1} X^T Y}$$

(we explain why this works later)

- We can't just solve directly because training data has errors. We don't model f, but $f + \varepsilon$, where are errors from?
  - Sensor error
  - Malicious data
  - Transcription error
  - Unmodeled influence

- We don't want to fit the error when we fit the data.

- Statistician acronym, IID = Independent & Identically Distributed
- "Use a model that is complex enough to fit the data without causing problems on the test set."
  - But note that when we build the model we're not even supposed to look at the test set.
  - Thus: Make a stand-in for the test set by making a cross-validation set from training data.
  - When checking our model, look at its error with cross-validation!
    - It's common to see cross-validation error rise as the model fits the training set better. (i.e. overfitting).
    - You want to find the balance between underfitting & overfitting.

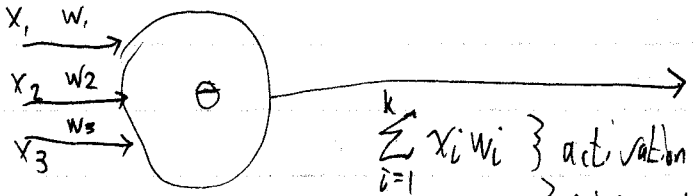
- To read:
  - Linear Algebra PDF
  - Mitchell Ch. 1 & 3

16
2016-01-~~11~~b, CS7641

- So that covered scalar input & continuous x.
  - But we'll also have vector input, $\vec{x}$, w/ more input features
  - Luckily the math generalizes.

SL3

- Neural Networks
  - Very abstracted version in ANN:



$\sum_{i=1}^{k} x_i w_i$ } activation
$\geq \theta$ } firing threshold

"Perceptron"

yes: $y = 1$
no: $y = 0$

- Perceptrons, in general, always compute linear functions. (For 2 inputs, it's a line. For 3, a plane.)
- Note also that for $w_1 = w_2 = 1/2$, $\theta = 3/4$, and $x_1 \in \{0, 1\}$, $x_2 \in \{0, 1\}$, this is just an AND gate.
- Likewise but $\theta = 1/4$ is an OR. (These aren't unique either.)
- $w_1 = -1$, $\theta = 0 \rightarrow$ NOT.
- Can also represent any Boolean by combining perceptrons.

- Perceptron Training
  - We've set weights by hand so far but that's not really too useful. We want to train them: Find weights, given examples, to map input → output.
  - perceptron rule (thresholded)
  - gradient descent/delta rule (unthresholded)

- Perceptron rule for single unit:

$w_i = w_i + \Delta w_i$ (iterative)

$\Delta w_i = \eta (y - \hat{y}) x_i$

$\hat{y} = \left(\sum_i w_i x_i \geq 0 \right)$

y: target

$\hat{y}$: output

η: learning rate

$y, \hat{y} \in \{0, 1\}$, so:

y	$\hat{y}$	$y - \hat{y}$
0	0	0
0	1	-1
1	0	1
1	1	0

- θ is just treated as another weight (think of subtracting θ from both sides) - we apply this bias.
- Δw_i formula basically says: If output is correct, don't change. If it's wrong, push it in the right direction (scaling by both η and the input's magnitude).
- If our data is linearly separable, perceptron rule always finds the line in finite iterations.
- We need some condition to stop iterating though...
  e.g. $\Delta w_i <$ some threshold

Text p89 (cites Minsky)

- Gradient descent. (Delta rule?)

$a = \sum_i x_i w_i$, $\hat{y} = \{a \geq 0\}$ but pretend output is not thresholded.

a = activation

$E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$ ($\frac{1}{2}$ is arbitrary but matters later)

D = data points

$\partial / \partial w_i E = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{\dots} (y - a)^2 = \sum_{\dots} (y - a) \frac{\partial}{\partial w_i} \left(-\sum_{i'} x_{i'} w_{i'} \right)$ ↓ definition of a, expanded

$= \cancel{\frac{1}{2}} \cancel{2} \sum_{(x,y) \in D} (y - a) \cancel{(-x_i)} = \sum_{(x,y) \in D} (y - a)(-x_i)$

See text, 4.4.3.2

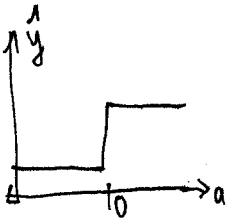
Note that $\partial / \partial w_i \, x_{i'} w_{i'} = 0$ except for $i' = i$, because the weights don't depend on each other

- Note that this is near-identical to perceptron rule.

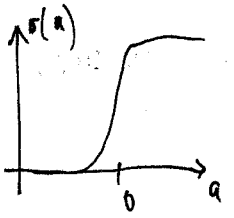
Is this stochastic gradient-descent rather than 'true' gradient-descent?

2016-01-17, CS7641

- So $\Delta w_i = \eta(y-a)x_i$ for gradient descent. Note however that we do <u>not</u> use thresholded $\hat{y}$ but a. This also only converges in the limit.
  - But $\hat{y}$ is non-differentiable so we cannot do gradient descent on it! ↳ discontinuous



  - However, we can make it a 'softer' threshold. Say...



$$\sigma(\alpha) = \frac{1}{1+e^{-a}}, \quad a \to -\infty,\ \sigma(a) \to 0. \quad a \to \infty,\ \sigma(a) \to 1.$$

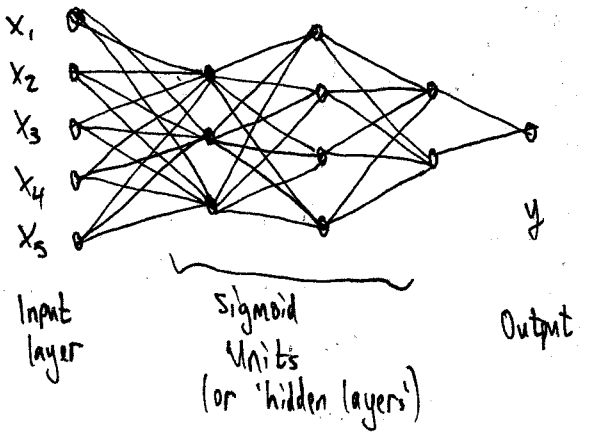
} Sigmoid (or: logistic function)

This is differentiable (and nicely):

$$d/da = \sigma(a)(1-\sigma(a))$$ - thanks to $\frac{d}{dx}e^{f(x)}$ being easy.

Other functions work (and may have good reasons to use) but this is pretty well-behaved. (tanh is mentioned)

- Now, for the actual <u>network</u> aspect of neural networks...



- One neat feature: Because of sigmoids, this entire network (output back to input) is differentiable! We can analyze the entire network's change in behavior relative to each individual weight.

See text p110, table 4.2
(backpropagation algorithm)

- Backpropagation just utilizes this; it's a computationally beneficial organization of the chain rule (all we're doing is derivatives).
  - Information flows from input to output, but error information flows from output to input - hence backpropagation. (sometimes, a.k.a. error back propagation).
- Also, we can use other functions here if differentiable and sort of like a threshold.
- But, we may end up with many local optima in overall error function.
- So, how do we avoid this?
  - In view of some, optimization = learning.
  - 4.5.2.1 'momentum terms' - Tend to push the change in the direction it was already going (hoping to avoid local minima)
  - higher-order derivatives - Look at how combinations of weights behave (Hamiltonians and such)
  - randomized optimization
  - Penalize 'complexity' (as in decision trees).
    - More nodes, more layers.
    - But also, larger weights complicate a network (perplexingly)

Larger weights can overfit

- Restriction bias of NN (i.e. what hypotheses do we consider):
  - Perceptrons represent half-spaces & Booleans

And other thresholds

  - Sigmoids don't leave much restriction.
  - Now, can we represent continous functions in NN?
    - Yes, with a large enough hidden layer.

Single hidden layer is all that's need

  - What about arbitrary functions (not continuous)?
    - Yes, with enough hidden layers. (2 of them to stitch together continuous ones)

Reading: Mitchell Ch. 4
Mitchell Ch. 8

2016-01-17b, CS7641

And when to stop training

- We can use cross-validation again here to decide on more layers, more nodes. → in training
- But how neural networks differ is that they're iterative, and without ever changing the structure, it may begin to overfit, perhaps as weights becoming larger & larger. Be mindful of that and cross-validate.

=Inductive bias?

- Preference bias (what NNs should we 'prefer'):
  - First: What should our initial weights be?
    (Typical: Small random values. Random, because we want some variation between runs to avoid local minima; small, because it's lower complexity).
  - Prefer low complexity; prefer simple explanations.
  - Occam's razor - don't multiply entities unnecessarily. (It gives lower generalization error for one thing)

2016-01-22

SL4

- Instance-based learning
  - A 'different' class of learning
    - Before, given $\langle x, y \rangle_1, \langle x, y \rangle_2, \cdots \langle x, y \rangle_n$ we assumed some $f(x)$ - and then we used $f(x)$ and <u>threw away</u> training data.
    - The different idea: Stick our training data into a database and try to match novel input against it, and return some input data.
    - Pros: 'Remembers', fast, simple (no 'interesting' learning goes on)

	Eu.	Manh.
1,6	$\sqrt{25}$	7
2,4	$\sqrt{8}$	4
3,7	$\sqrt{26}$	6
6,8	$\sqrt{40}$	8
7,1	$\sqrt{10}$	4
8,4	$\sqrt{20}$	6

- But: No generalization. Models noise exactly. Can't handle duplicate input.
- How might we work around this cleverly?
  - Nearest neighbor? (But what distance metric?) Multiple neighbors?
  - Or... k nearest neighbors! (or, similarity (inverse))
- k-NN:
  - Given: Training data $D = \{x_i, y_i\}$ — Domain knowledge → # of neighbors k
    Distance metric $d(q, x)$ — Query point q
  - $NN = \{i : d(q, x_i)$ over k is smallest$\}$
  - Return:
    - Classification: Let labels vote? (Plurality/mode of $y_i \in NN$) plus some tiebreaking decider
    - Regression: Mean of $y_i \in NN$ (weighted by similarity?)
  - But, this leaves a lot of details. d & k both matter substantially, and the voting/weighting/averaging matters. (They are 'domain knowledge')
  - Literature calls this a "lazy" learner (vs. an "eager" one like linear regression which involves most work done up-front)
  - k-NN and distance metric do make assumptions about data, but in general tend to work well
- Preference bias for k-NN:
  - Locality → Nearby points are similar (distance metric embeds, too, what 'nearby' means)
  - Smoothness → Assumes a sort of averaging in a neighborhood
  - More subtle: All features matter equally.
    - In practice some dimensions (for instance) may be more sensitive to change.

2016-01-22(b)

SL4 (cont)

"Curse of Dimensionality": As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially. (Not just in k-NN but in general)

From Bellman, the dynamic programming guy

- This runs against a common intuition in ML, which is to just keep adding more and more dimensions for every possibly-salient sensor/feature.
- To think about this 'curse' intuitively, think of growing a space in dimensions but still wanting data to cover the same 'area' overall.
  You want to 'cover' the space, and must in order to do learning.
- "Better off giving more data than more dimensions."
- How to pick k:
  - What if k=n & weighted average?
    - All points influence but not equally.
  - Or - why not use regression within neighborhood?
  → "Locally weighted regression"
    - In here one may use other things too, e.g. neural network.
    - Or, just linear regression.
    - This greatly expands hypothesis space.
- k-NN lets you build models around local things with the help of the above - which is a powerful technique.
  - Both k & distance/similarity are domain knowledge.

Note that average is just a special case of regression (0th-order)

SL5

- Ensemble Learning: Boosting
  - Think of... spam email. Instead of thinking of complicated schemes via what we've discussed so far, think of what simple rules are useful. Maybe:

Particularly, classifying email as $\{+, -\}$: spam / not spam

+	in body: "manly"	These are all 'evidence' but none suffice on their own
−	from spouse/friend	
+	short	
+	just URLs	
+	misspelled words like 'pr0n'	

  - This is sort of like a decision tree & sort of like a neural network, but sort of not.

Neural networks are sort of this

- Ensemble learning, in basic form:
  - Learn over subset of data → produce rule
  - Combine over many different subsets
- Why subsets? For one thing, can't develop simple rules over all data.
- But how do we (1) pick subsets, (2) combine?

Sounds sort of like RANSAC...

  - Suppose we do it uniformly & randomly, and applying learner, then combine w/ averaging. This has problems, though.
  - Picking subsets at random can still help with averaging out noise. (This approach: "Bagging" or "Bootstrap Aggregation")
    - Actually works well but has problems.
- Now: Suppose we pick a subset of examples that are "hard", or those that are the most difficult to classify properly.
  - Still, we'll do a weighted mean (but weights matter).
  - Now, how do we keep progress here from wrecking the 'easy' examples?

2016-01-22c

- First:
  - What is 'error'? # mismatches?
    We use: $\Pr_D[h(x) \neq c(x)]$
    - $\Pr_D$ → Probability over distribution D (whatever underlying D we have a sampling from)
    - $h(x)$ → hypothesis (specific one from our learner)
    - $c(x)$ → 'true' concept
  - That is: How likely is our hypothesis to be correct, given some random instance?
  - But note that we may have to take into account that our instances aren't equally likely to come up.
    - Some examples are more rare and perhaps more relevant for that reason.
  - "Weak learner": A learner that, no matter what the distribution over the data is, always performs better than chance. That is: $\forall_D \Pr_D[h(x) \neq c(x)] \leq 1/2 - \epsilon$
    - You always get <u>some</u> information from it.
    - This is actually a fairly strong condition.

Compare with "information gain"

- Finally, <u>boosting</u>:
  - Given training $\{(x_i, y_i)\}$, y_i in $\{-1, +1\}$
  - For $t = 1 \ldots T$:
    - Construct D_t
    - Find weak classifier $h_t(x)$ w/ small error $\epsilon_t = \Pr_{D_t}(h_t(x_i) \neq y_i)$ → $\epsilon_t < 1/2$ to be weak learner
  - Output H_{final}

Confusion: Are weak learner & weak classifier the same? If they must perform better than chance on all distributions then why do we refer to ones made for particular distributions? Did I misread?

- But: How do we build D_t, and how do we combine to H_{final}?
- Let $D_1(i) = 1/n$ - distribution starts out uniform (no instances are yet more valuable than others). And:

$$D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \quad \text{where} \quad \alpha_t = \frac{1}{2} \ln \frac{1-E_t}{E_t}, \quad \alpha_t > 0$$

AdaBoost?

→ Weight each instance based on error (adjust weight)

- Note that y_i & $h_t(x_i)$ both are -1 or 1
  Thus: $y_i h_t(x_i) = +1$ if they agree, -1 if not.
- Z_t is just a normalization constant to make D a distribution.
- This then weights higher the instances that it gets wrong.

- Then $H_{final}(x) = \text{sgn}\left(\sum_t \alpha_t h_t(x)\right)$ → weighted average & threshold.
- Because we've assumed a weak learner h_t, it's guaranteed to get at least some incorrect ones 'right'.
- This produces more expressive results than 'individual' hypothesis.

I need to read this.

- Text presents proof that boosting 'does well' converges over time
- The discussion here of 'why' has kind of lost me...

- Boosting = form of bagging?
- Boosting lets us use any learner that's a Weak learner.
- In practice: As your training error in boosting goes down, your testing error also does! (i.e. it doesn't overfit as readily).
  - More on that later.

2016-01-24, CS7641

- For reading: Ch. 4 (Mitchell)
- Artificial Neural Networks (ANNs)
  - Book says that for 'complex real-world sensor data' ANNs are among most effective methods known. (However, book is © 1997.)
  - Cites standard LeCun paper for handwriting recognition
  - Fastest neurons in brain switch in $\sim 10^{-3}$ s, which is quite slow compared to a computer. Yet, we can still recognize a person in 10^{-1} s, which led to speculation that in the brain it is highly parallel, yet more shallow.
  - One famous ANN example: ALVINN, which drove a car (→ sort of. steered...) on a highway using input from 30×32 pixels of camera
    - ALVINN uses a directed-acyclic-graph but other arrangements exist - acyclic/cyclic, directed/undirected. But, backpropagation assumes directed, though it can have cycles. Mostly though it's acyclic feed-forward.
- ANNs are suited to...
  - Noisy, complex sensor data. (cameras, microphones)
  - Problems w/ more symbolic reps. (e.g. decision tree learning)
- Backpropagation is most common learning technique, & suited to:
  - Instances with many attrib-value pairs.
  - Target function of discrete-valued, real-valued, or vector of either for output.
  - Training examples w/ errors.
  - Long training times, fast evaluation
  - No need for humans to understand target function

http://www.cs.cmu.edu/~tom/mlbook.html ?

- Perceptrons
  - Text differs slightly. They use:

$$o(x_1, x_2, \ldots x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

  - then assumes an $x_0 = 1$ so that:

$$o(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x}).$$

  - It represents hyperplane decision surface, $\vec{w} \cdot \vec{x} = 0$, with +1 on one side, -1 on other.
  - Examples must be "linearly separable" for this hyperplane to separate them. (XOR isn't linearly separable, while AND/OR/NAND/NOR all are.)
    - However, a 2-level deep network of perceptrons can represent any Boolean. (Since AND/OR/NAND/NOR can be combined to form others)
- ~~perceptron~~ Gradient descent / delta rule
  - "Linear unit" is an unthresholded perceptron:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

  - The delta training rule (see 2016-01-16b, 2nd page) is for how to train a linear unit.
- Hypothesis space search for backpropagation: n-dimensional Euclidean space of the n network weights. This is continous, not discrete (like decision tree learning)
- "Recurrent networks" - apply to time-series data and use a form of ~~recurrent~~ directed cycles to use output at time t as input at time t+1. (For instance: Stock trading)
- 'Cascade-correlation' algorithm - Adds hidden-layer units as needed

2016-01-26, CS7641

Mitchell text, Ch. 8 (p232) (p230)

- Instance-based learning
  - Can use "more" complex, symbolic representations for instances", though this also complicates 'neighboring' meaning
  - Simplest k-NN: Instances are in $\mathbb{R}^n$ & distance is normal Euclidean - $d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$ for arbitrary instance $x = \langle a_1(x), a_2(x), \ldots a_n(x) \rangle$ feature vector
  - Table 8.1 (p232) - k-NN for discrete-valued target function, $f: \mathbb{R}^n \to V$, $V = \{v_1, v_2, \ldots v_s\}$
  - 1-NN is basically Voronoi diagram?
  - Distance-weighted k-NN: $\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \, \delta(v, f(x_i))$,

    $w_i = \frac{1}{d(x_q, x_i)^2}$ and $\hat{f}(x_q) = f(x_i)$ if $d(x_q, x_i)^2 = 0$

    - And for real-valued: $\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i} w_i}$
    - We may safely increase k with very little harm to the results (as further neighbors have less effect) but this has performance drawbacks.
  - k-NN is highly-effective against noisy training data (if enough training data is present).
  - Drawback: k-NN uses <u>all</u> attributes when forming hypothesis, including irrelevant ones. One way to work with this is to stretch certain axes or even eliminate them, including using cross-validation to remove random axes & check the error to determine which are most relevant.

- Common terms:
  - Residual: error $\hat{f}(x) - f(x)$ in approximating target
  - Kernel function: Distance function used to determine weight, i.e. K such that $w_i = K(d(x_i, x_q))$.
- Locally-weighted regression:
  - "constructs explicit approximation to f over a local region surrounding x_q" (x_q = query point)
  - It uses nearby/distance-weighted training examples to form this approximation, $\hat{f}$. Thus, each new query instance has a different approximation
  - 8.5.1, locally-weighted linear regression
    - $\hat{f}(x) = w_0 + w_1 a_1(x) + \dots + w_n a_n(x)$
    - We can use gradient-descent but must modify to do a local, not global, approximation
    - Not copying formulas, see p 237
  - Note that usually for a <u>local</u> approximation these simpler regressions (constant, linear, quadratic) suffice - because of the locality. More complex approximations usually are not worth the added cost in fitting them.
- Radial Basis Functions
  - Related to distance-weighted regression & to ANNs

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

  - $d(x_u, x)$ → Distance
  - x → Instance from X
  - K_u → Kernel function. Must <u>decrease</u> as $d(x_u, x)$ increases.

Common K_u is Gaussian: $K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$

Note that variance σ_u^2 is in terms of u, and center of Gaussian is x_u.

2016-01-26b, CS7641

- See Figure 8.2 - this is equivalent to a 2-layer network.
- Training is typ. 2-stage:
  1. Number of hidden units (k) is determined; each hidden unit 'u' is defined by choosing x_u, σ_u^2 to define K_u.
  2. Weights w_u are trained to maximize fit of network to training data.
- Instance-based
  1. 'Lazy' learning (defer generalization to query time)
  2. Classify by analyzing 'similar' instances, ignoring 'different'
  3. Represent instances as points in $\mathbb{R}^n$

  Case-based reasoning

  X
  - CBR = case-based reasoning (see 8.5)
    - Typically a richer symbolic description, more complex querying
    - One system from '92: CADET
- Note that 'eager' learners must have already chosen their global approximation to the target function, and can't consider query instance when deciding how to generalize
  - If lazy & eager learners use same hypothesis space, then lazy learner tends to perform better. In effect, it has a 'richer' hypothesis space by being able to specialize to local neighborhoods.
- Radial basis functions sort of try to get benefit of multiple local approximations (as in lazy) but still are eager.

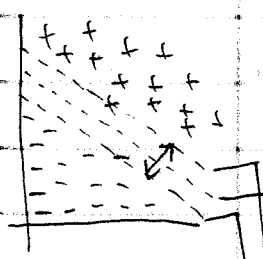
- Notes on paper:
"The Boosting Approach to Machine Learning: An Overview"
(Robert E. Schapire, 2001-12-19)
- "Boosting... is based on the observation that finding many rough rules of thumb can be a lot easier than finding a single, highly accurate prediction rule."
- My earlier notes appear to explain AdaBoost but instructors didn't call it that. → 2016-01-22c
- "boosting", because it 'boosts' learning algorithms that are just better than random guessing into arbitrarily accurate learners.
- See: LogitBoost
  - AdaBoost can be extended to produce not just a label but a probability of the label.
- AdaBoost.M1 is for Multiclass case (or AdaBoost.MH)
- This paper provides some good proofs, analysis, and extensions of AdaBoost.

- Notes on: AdaBoost (Matas & Šochman) slides
  - p14: Why α_t is what it is (they have $\frac{1}{2}\log\left(\frac{1+r_t}{1-r_t}\right)$, $r_t = \sum_{i=1}^{m} D_t(i) h_t(x_i) y_i$ - not sure if $= \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ as in notes)
- TCA = totally corrective step (Totally Corrective Adaboost?)
  - They don't really explain this past saying that it helps.

2016-01-26c, CS7641

SL-6

- Kernel Methods & SVM (Support Vector Machines)
  - This will circle back around to boosting...
  - 8C-L3 from CS4476 (see 2015-11-09c) also talks about SVMs albeit in probably less detail
- For our plane/hyperplane, $y = w^T x + b$ (any # of dimensions)
  - w, b → parameters of plane
  - y → classification label (+ = in class, − = out of class)



- Decision boundary: $w^T x + b = 0$
- Say that $y \in \{-1, +1\}$ for the +/− examples in the middle that are at 'margins'.
- Margins: $w^T x + b = 1$, $w^T x + b = -1$
- Also, we want to maximize distance between margins.
- Lets say for some point x_1 on 'positive' margin line, point x_2 on 'negative' margin line...
  $w^T x_1 + b = 1$, $w^T x_2 + b = -1$ → $w^T(x_1 - x_2) = 2$
- We want to maximize length of line between x_1 & x_2.
- $\frac{w^T(x_1 - x_2)}{\|w\|} = \frac{2}{\|w\|}$ — basically, projecton of $x_1 - x_2$ onto w?
- But w is the hyperplane's normal... and $(x_1 - x_2)$ is perpendicular. So, $m = \frac{w^T(x_1 - x_2)}{\|w\|}$ is actually $\|x_1 - x_2\|$, the length we want, and $m = 2/\|w\|$.
- Constraint of classifying correctly: $\forall i,\ y_i(w^T x_i + b) \geq 1$ (recall definition of y).
- Maximizing $2/\|w\|$ is sort of a pain, but minimizing $\frac{1}{2}\|w\|^2$ is equivalent and easier. (For whatever reason) ("Quadratic programming"?)

Also, say that line between x_1 & x_2 is perpendicular to hyperplane.

m = 'margin'

→ Goal: Maximize margin but still classify data correctly

We are finding optimal 'linear separator'.

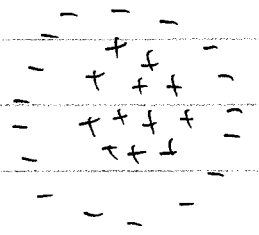
- Through a bunch of quadratic programming magic:
- $\min \frac{1}{2}\|w\|^2 \rightarrow W(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j$, maximize W

such that $\alpha_i \geq 0$, $\sum_i \alpha_i y_i = 0$

- $w = \sum_i \alpha_i y_i x_i$, solve for b, once we know α_i
- α_i are 'mostly zero, so only a few vectors here actually matter (those for nonzero α)... these are the support vectors.

Points far from decision boundary generally are not support vectors.

- Note that $x_i^T x_j$ in 'w' formula = dot product = projection of one on the other. It captures some notion of similarity.
- But what if points aren't linearly separable? e.g.



- Suppose we have function $\Phi(g) = \langle g_1^2, g_2^2, \sqrt{2}\, g_1 g_2 \rangle$ → point, 2D, $\langle g_1, g_2 \rangle$
  - This hasn't really added any 'new' info, though it maps to 3D
  - Now, see what $\Phi(x)^T \Phi(y)$ equals... $(x^T y)^2$. That is intentional; it also relates to a circle, while still representing similarity (as did $x^T y$).
  - This would effectively lift the +s up → to a new dimension and separate them from the −s based on that circle.
- This is the 'kernel trick'. We can use other functions to define similarity in higher-dimensional spaces, and Φ itself doesn't actually matter.

2016-01-26d, CS7641

SL-6 (cont.)

- We then used: $W(\alpha) = \sum \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \overbrace{K(x_i, x_j)}^{\text{Kernel}}$.

- Started with $K = x^T y$
  Then used $K = (x^T y)^2$ but note that we never actually had to transform points with Φ
- Common kernel: $K = (x^T y + c)^p$ (polynomial kernel)
- Another: $K = e^{-(\|x-y\|^2 / 2\sigma^2)}$ (radial basis kernel)

- Sigmoid-like: $K = \tanh(\alpha x^T y + \theta)$
- The kernel needs to capture some domain knowledge.
  - x can be discrete, or strings, or images - as long as your kernel expresses similarity.
- But must satisfy Mercer Condition; loosely, 'it acts like a distance function'. More on that later...
- SVMs are sort of "eager lazy learners" - they only use a certain vectors that are needed.
- Back to boosting; why doesn't it overfit how we'd expect?
  - Recall $H_{final}(x) = \text{sgn}(\sum_t \alpha_t h_t(x))$.

    $\frac{\sum_t \alpha_t h_t(x)}{\sum_t \alpha_t}$ will be in -1 to $+1$, and it will give some notion of confidence rather than error.
  - As you keep iterating, your error plateaus - but your confidence continues to rise. That is, the $-$ and $+$ are pushed closer & closer to -1 and $+1$.

& unrelated to in SVMs

- That should look familiar: It's making a bigger and bigger margin as you add more and more weak learning.
  And: Larger margins → less overfitting, as in SVMs.
- Boosting still overfits ...? (example is kind of wonky)
- Pink noise tends to make boosting overfit.
  uniform noise (not white noise, that's Gaussian noise)
- If underlying learner to boosting tends to overfit then boosting will overfit.

2016-01-28

- Notes on: "Support Vector & Kernel Machines" (Nello Cristianini), ICML 2001
- SVMs: COLT-92 (?), Boser, Guyon, Vapnik
  - They are a particular instance of kernel machines.
  - Two problems — Computational: Detect & exploit complex patterns in data (clustering, classifying, ranking, cleaning, etc.)
    Statistical: How to represent complex patterns, and not include spurious/unstable patterns (i.e. don't overfit)
- Informally, kernel methods limit patterns by introducing notion of similarity, and thus relevant features
- Kernel functions need to be inner products in some space (but most algorithms only exploit inner product, not the space)

2016-01-28 (6) CS7641

- Kernel-based learning algo → General-purpose learning machine
  → Problem-specific kernel function
- Great for "modularity" and "software engineering" (?)
- SVMs are Linear Learning Machines represented in dual fashion? (LLMs)
- But if data is non-linearly separable...
  - Use a neural network? This has its own issues.
  - Or: Map into a richer feature space where it *is* linearly separable.
    $x \rightarrow \phi(x)$ - May be higher-dimensional.
  - Problems: Computational issues (very large vectors)
    Generalization theory problem (curse of dimensionality)
  - "Implicit" mapping of kernels can solve this and even allow infinite-dimensional feature spaces.
  - In "dual representation" data is only in dot products,

$$f(x) = \sum \alpha_i y_i \underbrace{\langle \phi(x_i), \phi(x_j) \rangle}_{\text{Don't even need to know } \phi \text{ for this.}} + b$$

$$= K(x_1, x_2)$$

  - LLMs work in this feature space: simply rewrite in dual representation (?) & replace dot product w/ kernels.
  - Kernel matrix (a.k.a. Gram matrix):

$$K = \begin{bmatrix} K(1,1) & K(1,2) & \cdots & K(1,m) \\ K(2,1) & K(2,2) & \cdots & K(2,m) \\ \vdots & \vdots & \ddots & \vdots \\ K(m,1) & K(m,2) & \cdots & K(m,m) \end{bmatrix}$$

  - Contains *all information* for learning algorithm - both data & kernel.
  - It is Symmetric Positive Definite, and any S.P.D. matrix can be regarded as a kernel matrix (that is, an inner product in *some* space) (Mercer's theorem)

www.support-vector.net
www.kernel-machines.org
www.NeuroCOLT.org

- Eigenvalues expansion of Mercer's Kernels:
  $K(x_1, x_2) = \sum \lambda_i \phi_i(x_1) \phi_i(x_2)$
  → Eigen functions act as Features?
- Kernels are closed under some operations:
  - If K & K' are kernels, then...
  - $K + K'$ is a kernel
  - $c > 0$, cK is a kernel
  - $a, b > 0$, $aK + bK'$ is a kernel
- Kernels can work over general structures - Sets, sequences, trees
- A mostly diagonal kernel matrix would be a bad kernel - all points would be orthogonal.
  → If mapping in space w/ too many irrelevant features, kernel becomes this.
- p50 - How to find hyperplane?
- 'SVMs control capacity by increasing the margin, not by reducing the number of degrees of freedom'
- p54 gives a little more formal look
- p59 'the dual problem'?
- p64 - How to minimize margin when noise is present
- p73 - Other kernel-based algorithms exist
- Kernel alignment: $A(K_1, K_2) = \frac{\langle K_1, K_2 \rangle}{\sqrt{\langle K_1, K_1 \rangle \langle K_2, K_2 \rangle}}$ (similarity metric)

"Ultimate" kernel should be YY', that is, given only by labels vector?

- Combining kernels aligned to the <u>target</u>, not each other, increases alignment.
- 'Spectral machines', p79! See Courant-Hilbert Theorem.
- Can adapt kernels to labels or vice versa.

2016-01-28c, CS7641

- Notes on: "SVM & Kernel Methods" (Bernhard Schölkopf)
  - "Statistical learning theory" (Vapnik & Chervonenkis, '60s) → VC
    - Model: Unknown stochastic regularity produces data that we observe.
    - Learning: Extraction of that data's regularity.
      → notions of capacity of function classes that learning machine can implement.
  - First few pages is a lot of derivation
    - VC dimension? VC theory?
  - $\phi : \mathcal{X} \to \mathcal{H}$, $x \mapsto \phi(x)$] usu. $\dim(\mathcal{X}) \ll \dim(\mathcal{H})$
    ↳ dot product space
    - and then learn mapping from $\phi(x)$ to y.
    - crucial issue: "capacity, not dimensionality" ?
  - Mercer's theorem, p 21
    - Conditions for valid kernel...
    - Kernels must be symmetric
    - For any training points $x_1, \ldots x_m \in \mathcal{X}$, any $a_1, \ldots a_m \in \mathbb{R}$, and $K_{ij} := k(x_i, x_j)$, $\sum_{i,j} a_i a_j K_{ij} \geq 0$
  - These slides are over my head a bit...

- Notes on: "A Tutorial on Support Vector Machines for Pattern Recognition" (Christopher J.C. Burges)
  - The "capacity" of an SVM is the "ability to learn any training set without error".

- This paper assumes I understand Lagrange multipliers to solve problems with constraints. That's unfortunate.

- Too much capacity → No ability to generalize
- Too little capacity → Incorrect generalization
- This paper is a little too nitty-gritty for me to focus on right now.

2016-02-05

Mitchell, Ch.7

- Computational Learning Theory
- PAC: Probably Approximately Correct learning model
  - X = set of all possible <u>instances</u> over which target functions may be defined.
  - C = set of target concepts learner might need to learn
  - c in C is a subset of X, or $c: X \to \{0, 1\}$ where $c(x)$ for positive example = 1, and = 0 for negative.
  - Prob. distribution D produces instances at random from X.
    - D is generally unknown to learner, but must be stationary (not change over time)
  - One may produce training examples from D - create instance x and pass $(x, c(x))$ to learner.
  - Learner L considers set H of possible hypotheses, and (having observed training examples) produces $h \in H$, where h estimates c.
    - We only evaluate L's success over same D as it trained on.
  - $error_D(h)$ = true error of h w.r.t. target concept c, distr. D, is prob. that h <u>misclassifies</u> a random instance drawn according to D.
    $= \Pr_{x \in D}[c(x) \neq h(x)]$, or, 'set difference' of c & h.

    → If c & h agree on something with probability zero in distribution, this doesn't matter. (Or, if they disagree)

2016-02-05(b), CS7641

Mitchell, Ch.7 (cont)

- Note that true error isn't visible to learner. It can only use training examples. ('training error' = fraction of training examples misclassified)
- This refers to Ch.6 which I've not read yet...
- What we're interested in:
  - Error $\leq \epsilon$ and can be made arbitrarily small ('approximately correct')
  - Probability of failure $\leq \delta$ " " " ('probably')

$0 < \epsilon < \frac{1}{2}$
$0 < \delta < \frac{1}{2}$

- 'PAC-learnable' (p206) says basically this, with constraint of polynomial time in $1/\epsilon$, $1/\delta$, n, size(c). → encoding length of c in C
  - with # of instances
  - N.B. If L requires processing time, then 'C is PAC-learnable' ⇒ 'L must learn from polynomial # of training examples'. PAC-learnable constrains the number of traning examples, indirectly.
- This assumes: For every target concept in C, H contains a hypothesis with arbitrarily small error. This is quite difficult without C known in advance.
- 'Sample complexity' of the learning problem = growth in number of required examples with problem size. This is usually of the most interest because: The limiting factor in most practical settings is the amount of training data.
- A 'consistent' learner: Outputs hypotheses perfectly fitting training data, whenever possible. This is fairly reasonable.
- From Ch.2, Version space $VS_{H,D}$ = set of all hypotheses

$h \in H$ that correctly classify training examples D:

$$VS_{H,D} = \{h \in H \mid (\forall \langle x, c(x) \rangle \in D)\ (h(x) = c(x))\}$$

- Note that every consistent learner outputs a hypothesis in $VS_{H,D}$ ($VS_{H,D}$ contains every consistent learner in H - by definition).
  - If we can bound the examples needed for $VS_{H,D}$, we can bound the examples needed for any consistent learner.

p208

- $VS_{H,D}$ is ϵ-exhausted if $(\forall h \in VS_{H,D})\ error_D(h) < \epsilon$
  - That is: All hypotheses consistent w/ observed training examples (i.e. training error = 0) have true error $< \epsilon$.
  - Learner sees only training error. Only observer who knows target concept can tell apart instances of version space (to the learner they are identical).

Proof p209

- Theorem 7.1: For finite H, D is sequence of $m \geq 1$ independent randomly drawn examples of some target concept c, then for any $0 \leq \epsilon \leq 1$, probability that $VS_{H,D}$ is not ϵ-exhausted w.r.t. c $\leq |H| e^{-\epsilon m}$ ↰
  - That is - that's the probability that m training examples won't eliminate all 'bad' hypotheses.
- Suppose we want to put that probability below δ.

$$|H| e^{-\epsilon m} \leq \delta \longrightarrow m \geq \frac{1}{\epsilon}(\ln |H| + \ln(1/\delta))$$

→ With probability $(1-\delta)$, any consistent learner will be correct (within error ϵ), if it has that 'm' of training examples.

Note m is linear in $1/\epsilon$, logarithmic in $1/\delta$, logarithmic in size of hypothesis space.

- This is an overestimate, often.

2016-02-05c, CS7641

Ch. 7
Mitchell (cont.)

- If H does not contain target concept c, a zero-error hypothesis may not be possible. We can only ask the learner to output the hypothesis $\in H$ with minimum error over training examples.
- agnostic learner - makes no commitment about whether or not $C \subseteq H$; just finds hypothesis with min. training error.
- The prior bounds generalize to learners that produce hypothesis with nonzero error.
- That uses 'general Hoeffding bounds':

$$Pr[error_{\mathcal{D}}(h) > error_{D}(h) + \epsilon] \leq e^{-2m\epsilon^2}$$

$$\rightarrow m \geq \frac{1}{2\epsilon^2}\left(\ln|H| + \ln(1/\delta)\right)$$

'Best hypothesis' here may have nonzero error.

N.B.
$D \neq \mathcal{D}$.
$\mathcal{D}$ = training samples available to learner.
D = distribution over whole set of instances.

2016-02-06

- Basing bounds on $|H|$ leads to rather weak bounds, and the prior equations can't handle infinite hypothesis spaces either.
- $VC(H)$ - or VC dimension, or Vapnik-Chervonenkis dimension of H - is a better measure of H's complexity and handles infinite spaces too.
- VC measures by number of distinct instances from X that H can discriminate - not by how many distinct hypotheses.
- Consider $S \subseteq X$ (a subset of instances). Note that every $h \in H$ imposes a dichotomy on S: it partitions to two subsets, $\{x \in S | h(x) = 1\}$, $\{x \in S | h(x) = 0\}$.

- $2^{|S|}$ dichotomies exist (though H may not represent all of them).

See 7.4.1
- H <u>shatters</u> S if H can represent every possible dichotomy of S.
- That is, if H does not shatter S, then some concept can be defined (i.e. dichotomy of S) which H cannot represent.
- This relates closely to inductive bias of H.
  - Loosely: Unbiased H shatters instance space X.
- H shatters a larger subset of X → more expressive H.

7.4.2
- VC(H) = size of largest finite subset of X shattered by H. (If arbitarily large finite subsets of X, then $VC(H) = \infty$.)
  → Finite H, $VC(H) \leq \log_2|H|$.
  (Suppose $VC(H) = d$. Then H needs 2^d distinct hypotheses to shatter d instances. Hence, $2^d \leq |H|$, solve for d.)
- p215, 216 give a decent example for intuition.
  - N.B. <u>Any</u> set of instances, size d, that H shatters → $VC(H) \geq d$
    <u>No</u> set of size d that can be shattered by H → $VC(H) < d$
  - VC dim. of linear decision surface in 'r' dimensional space = r+1.
- Through mathematical magic, new bounds are:

7.4.3
$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8\,VC(H)\log_2\left(\frac{13}{\epsilon}\right)\right)$$
- Theorem 7.3 establishes a lower bound too:
  $\max\left[\frac{1}{\epsilon}\log\left(\frac{1}{\delta}\right), \frac{VC(C)-1}{32\epsilon}\right]$ provided $VC(C) \geq 2$, $0 < \epsilon < \frac{1}{8}$, $0 < \delta < \frac{1}{100}$
  - N.B. $VC(C)$, <u>not</u> $VC(H)$.
- 7.4.4 covers VC of directed acyclic graphs (and I ignore it)

- Another question aside from PAC: The <u>mistake bound</u> model of learning - How many mistakes can learner make before converging to correct answer?

2016-02-06(b), CS7641

Mitchell
Ch. 7 (cont.)

- 'mistake': The learner, before it receives the target value $c(x)$, must predict it. Predicting incorrectly here is the mistake.
- In practical settings where learning is on-the-fly, this is very relevant.
- 7.5.1 - deriving this for Find-S (from Ch. 2)

2016-02-08

SL7

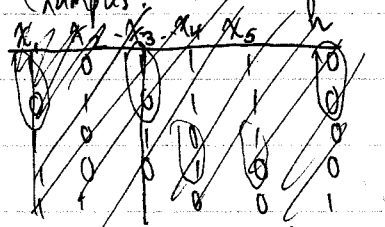
- Time & space matter in theory of computing.
- In computational learning theory, these both matter, but so (more so) does the number of samples required
- Inductive Learning
  1. Probability of successful training $(1-\delta)$
  2. Number of examples to train on
  3. Complexity of hypothesis class
  4. Accuracy to which target is approximated (ε)
  5. Manner in which training examples presented (Batch? or online?)
  6. " " " " " selected
     - Learner can ask questions of teacher (supplies x, gets $c(x)$)
     - Teacher gives examples to help learner (teacher supplies $x, c(x)$)
     - Fixed distribution (chosen by 'nature')
     - Evil distribution

- Consider teaching via 20 questions
  H: Set of possible people, X: Set of questions
  - Teacher chooses X vs. learner chooses X
    - If teacher chooses question then learner could get answer on just 1st question. (Though, a bit contrived.)
- Now suppose...
  X: $x_1, x_2 \dots x_k$ (k-bit input)
  h: x_1 and x_3 and $\overline{x_5}$
  H: Conjunctions of literals or negation

For example →

x_1	x_2	x_3	x_4	x_5	h
0	1	0	1	1	0
1	0	1	0	1	0
1	0	0	1	0	0
1	1	1	0	0	1
1	0	1	1	1	0

- But if we don't know h how would we find it from examples?



→ x_1 & x_3 change but h does not. Formula can't have x_1 or x_5 then - they're absent.
~~- But x_4 & x_2 also behave that way?~~

For a 'helpful' teacher

- Find what's irrelevant - which actually needs only 2 questions
- Find what's relevant - needs $\leq k$ questions for k variables left
  - But, 3^k hypotheses total (positive, negative, or absent)
- But what if the learner chooses questions, not the teacher?
  - Learner has 3^k hypotheses to choose from...
  - But, it ends up $\sim 2^k$ samples & questions. Each question doesn't really narrow down hypothesis space. (Assuming that learner can only supply $x_1 \dots x_k$ and receive 0 or 1.)
- But... with mistake bounds, learner guesses answer from input before getting correct answer

2016-02-08(b), CS7641

SL7 (cont)

- To make an algorithm:
  1. Assume each variable positive and negative (i.e. $x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge x_3 \wedge \neg x_3 \cdots x_K \wedge \neg x_K$)
  2. Given input, compute output from this ↗
  3. If guess from step 2 is wrong: Set all positive variables that were 0 to absent, and negative variables that were 1 to absent.
  4. Return to 2.

	Positive	Absent	Negative
x_1	X		X
x_2	X		X
x_3	X		X
x_4	X		X
x_5	X		X
⋮	⋮		⋮

- That is, we start with:
  - and remove from columns as needed, adding to 'absent'.
- This guarantees that it finds the correct formula with only k+1 mistakes - provided the teacher asks the right questions.
- Definitions of relevance:
  - Computational complexity - How much computational effort is needed for learner to converge?
  - Sample complexity - Relevant for batch training; how many training examples are needed for learner to create right hypothesis?
  - Mistake bounds - Relevant for online training; how many misclassifications can a learner make over an infinite run?

we don't focus much on this through ↗

- Version spaces
  - True hypothesis, $c \in H$ Training set, $S \subseteq X$, $c(x)\ \forall x \in S$
  - Candidate hypothesis, $h \in H$

That notation differs a little from text, I think

→ Thus, consistent with respect to some S (training set)

- Consistent learner produces $c(x) = h(x)$ for $x \in S$
- Version space: $VS(S) = \{h \mid h \in H \text{ consistent w.r.t. } S\}$
  → Hypothesis consistent with examples.
- True error/training error - consistent w/ text (see 2016-02-05 notes)
  - Likewise for PAC-learnable, ϵ-exhausted.
- Haussler theorem, bounding true error
  - Let $error_D(h_1, \ldots h_k \in H) > \epsilon$, high true error.
  - How much data do we need to 'knock out' these hypotheses?
  - $Pr_{x \in D}(h_i(x) = c(x)) \leq 1 - \epsilon$ (it's unlikely they will be right)
  - $Pr(h_i \text{ consistent with } c \text{ on } m \text{ examples}) \leq (1-\epsilon)^m$ (multiply probabilities)
  - $Pr(\text{at least one of } h_1 \ldots h_k \text{ consistent with } c \text{ on } m \text{ examples})$
    $\leq k(1-\epsilon)^m \leq |H|(1-\epsilon)^m \leq |H| e^{-\epsilon m} \leq \delta$ where δ = failure probability
  → $m \geq \frac{1}{\epsilon}(\ln|H| + \ln\frac{1}{\delta})$

Why Exhausted matters: if VS is this, then all hypotheses are "equally good" as far as learner cares, and can just pick one.

$-\epsilon \leq \ln(1-\epsilon)$ (it's not obvious by itself)

→ $(1-\epsilon)^m \leq e^{-\epsilon m}$

$|H| e^{-\epsilon m}$ = upper bound that VS is not ϵ-exhausted after m samples

N.B. This is independent of distribution D!
- They sort of 'cancel out' in training & true error

2016-02-10

SL-8

- VC Dimensions
  - That $|H|$ is troubling.
    - Linear separators are infinite-dimensional (even for $y = mx + b$)
    - So are ANNs, so are decision trees w/ continuous inputs
    - There are for the above an infinite number of hypotheses.
    - For k-NN this is sort of the case
  - Suppose... $X: \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
    $H: h(x) = x \geq \theta \leftarrow \theta \in \mathbb{R}$
    $|H| = \infty$

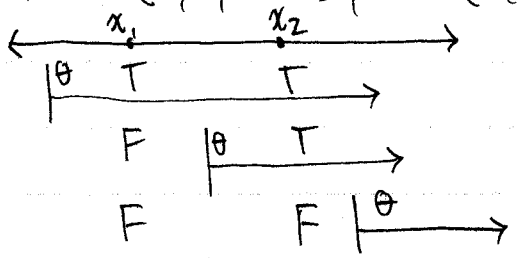
2016-02-10(b), CS7641

SL-8 (cont)

- Why though do we need infinite hypotheses if X has only 8 elements?
- "What is the largest set of inputs that the hypothesis class can label in all possible ways?"

"Shattering"

- So for $X = \{1, 2, \ldots 10\}$, $H = \{h(x) = x \geq \theta\}$...



- Choice of θ here can label x_1 & x_2 in class, or x_2 but not x_1, or neither x_1 nor x_2.
- But no θ can put x_1 in class, and not x_2.

- So though $|H| = \infty$ it cannot express even this!
- VC dimension relates to shattering. (What's largest subset H can shatter?)
- Example: $X = \mathbb{R}$

  $H = \{h(x) = x \in [a,b]\}$ parametrized by $a \in \mathbb{R}, b \in \mathbb{R}$

  - Shattering $\{x_1\}$: just pick $[a,b]$ to include x_1 and another to exclude → VC dimension ≥ 1
  - Shattering $\{x_1, x_2\}$, $x_2 > x_1$:

    If $a < b < x_1$: $x_1 -, x_2 -$

    $a < x_1 < b$: $x_1 +, x_2 -$

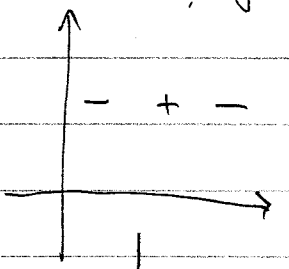
    $x_1 < a < x_2 < b$: $x_1 -, x_2 +$

    $a < x_1 < x_2 < b$: $x_1 +, x_2 +$

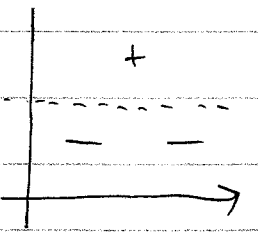
    → VC dim ≥ 2
  - Shattering $\{x_1, x_2, x_3\}$, $x_3 > x_2 > x_1$:
    - We cannot produce $x_1 +, x_2 -, x_3 +$, so H does not shatter this cardinality-3 subset (which is sufficiently general to cover all such subsets).

So, VC dim. is 2.

- Proving an upper bound is sometimes difficult here, because one must prove that no subset of size N can be shattered, whereas for lower bound a single example of size $M < N$ suffices.
  - Which matters e.g. for a linear separator in $\mathbb{R}^2$.

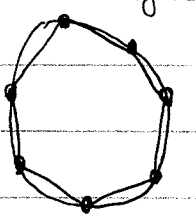


No linear separator can do this case of size 3. (for those points collinear).

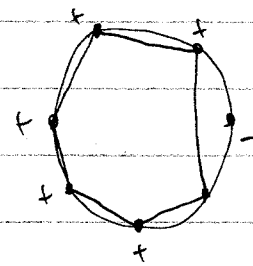


But can handle this case, and only that's needed for VC dim. ≥ 3.

- Suppose $X: \mathbb{R}^2$, H: points inside convex polygon (on edge = inside).
  - Consider any n points on unit circle.



- Connect all these points. That makes a convex polygon that includes all of them.
- Suppose we want to remove any point from the hypothesis:



- simply 'push' that edge in. The polygon is still convex (due to being in unit circle).
- We may apply this recursively with any combination of points. Thus, we can shatter <u>any</u> number of points.

- Thus, <u>VC dimension = ∞</u>.

- What is VC of a finite H?
  - We <u>must</u> use VC for infinite H. We <u>can</u> for finite H.

2016-02-10(c), CS7641

- Upper bound: (for finite $|H|$)
  $d = VC(h) \Rightarrow \exists 2^d$ distinct concepts (each with different H)
  $2^d \leq |H|$, $d \leq \log_2 |H|$
- Theorem: H is PAC-learnable iff VC dimension is finite!
  - VC dimension actually captures PAC-learnability.
- VC dimension also ends up equating the number of parameters in the hypothesis - but the true number of parameters (e.g. if you expressed it inefficiently and had redundant parameters).

Mitchell Ch.6

- Bayesian Learning
  - "Bayesian reasoning provides a probabilistic approach to inference."
- Features of Bayesian learning methods:
  - Each observed training example can increase/decrease estimated probability that hypothesis is correct (rather than eliminating it with an inconsistent example)
  - Prior knowledge can be combined with observed data.
  - Can accomodate hypotheses making probabilistic predictions (e.g. patient has 93% chance of recovery)
  - New instances can be classified by combining predictions of multiple hypotheses, weighted by their probabilities
- Difficulties:
  - Require initial knowledge of many probabilities.
  - Significant computational cost (linear w/ # of candidate hypotheses)

- Compare w/ CS6476, 7C-L1 ?

- "Best" hypothesis might mean "most probable" hypothesis - given data plus initial knowledge of prior probabilities of H.
  - Bayes theorem lets us compute this.
- $P(h)$ = prior probability of h, before observing training data.
  - Might reflect background knowledge, might just be evenly distributed if we don't know.
- $P(D)$ = prior probability of training data D being observed (given no knowledge of which hypothesis holds).
- $P(D|h)$ = probability of observing D given that h holds.
- We're most interested in $P(h|D)$, posterior probability of h = our confidence that h is right after seeing D.

Actually much more general than this.

Bayes theorem: $$P(h|D) = \frac{P(D|h)\,P(h)}{P(D)}$$

- Intuition: Increasing $P(h)$ or $P(D|h)$ → Increasing $P(h|D)$
  Increasing $P(D)$ → Decreasing $P(h|D)$
  (more likely D is, the less evidence for h it provides)

- MAP hypothesis = maximum a posteriori - Most probable $h \in H$ given observed data D. By Bayes theorem:

$$h_{MAP} \equiv \underset{h \in H}{\operatorname{argmax}}\, P(h|D) = \underset{h \in H}{\operatorname{argmax}}\, \frac{P(D|h)P(h)}{P(D)} = \underset{h \in H}{\operatorname{argmax}}\, P(D|h)P(h)$$

↳ dropped because it's independent of h

- If we assumed all of H was equally likely then $P(h)$ is also constant and only $P(D|h)$ matters.
- $P(D|h)$ = 'likelihood' of data and hypothesis maximizing it is 'maximum likelihood' (ML) hypothesis:

$$h_{ML} \equiv \underset{h \in H}{\operatorname{argmax}}\, P(D|h).$$

2016-02-10d, CS7641

Mitchell Ch. 6 (cont)

- Consider example (cancer diagnosis):
  - $P(\text{cancer}) = 0.008$, $P(\neg\text{cancer}) = 0.992$
    - Only 0.8% of people have this cancer.
  - $P(\oplus|\text{cancer}) = 0.98$, $P(\ominus|\text{cancer}) = 0.02$
    - Test returns $\oplus$ in only 98% of cases when disease is present.
  - $P(\oplus|\neg\text{cancer}) = 0.03$, $P(\ominus|\text{cancer}) = 0.97$
    - Test returns $\ominus$ in only 97% of cases when disease is not present.
  - So for a new patient with positive test result, what is chance it actually is cancer?

    $P(\oplus|\text{cancer})P(\text{cancer}) = 0.0078$ =

    $P(\oplus|\neg\text{cancer})P(\neg\text{cancer}) = 0.0298$

    → Normalized $P(\text{cancer}|\oplus) = 0.21$ - so likely not cancer.
    - We don't have $P(\oplus)$ but know that $P(\text{cancer}|\oplus) + P(\neg\text{cancer}|\oplus) = 1$.
  - Posterior prob. of cancer is much higher than prior (21% vs. 0.8%) - but most probable hypothesis is not cancer.
- See table 6.1, p159:

  $P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$, $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

  $A_1 \ldots A_n$ mutually exclusive, $\sum_{n=1}^{n} P(A_i) = 1 \Rightarrow P(B) = \sum_i P(B|A_i)P(A_i)$
- Interestingly, some algorithms already covered produce the same hypothesis as brute-force Bayesian despite not explicitly using probabilities.
- Brute force MAP learning algorithm:
  1. $\forall h \in H$ find posterior prob.: $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$
  2. Output $h_{MAP} = \underset{h \in H}{\text{argmax}}\ P(h|D)$, $h_{ML} = \text{argmax}\ P(D|h)$ if $P(h)_{\text{uniform}}$
  - That $\forall h \in H$ is an issue for similar reasons as why $|H|$ is.

- We must specify $P(h)$ & $P(D|h)$ here ($P(D)$ is determined by these), based on prior knowledge. Here we assume:

"uninformed" prior

$d_i = c(x_i)$ (no noise in D), $c \in H$, no a priori reason for any hypothesis to be more likely.

→ $P(h) = \frac{1}{|H|}$ for all $h \in H$ (N.B. That's prior probability of h)

and $P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \ \forall d_i \in D \\ 0 & \text{otherwise} \end{cases}$

- That is, if h holds, assume we observe all consistent training data.

$VS_{H,D}$ = version space, = $|VS_{H,D}|$ = # of hypotheses consistent with D

→ If h is inconsistent w/ D then $P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0$

If consistent: $P(h|D) = \frac{1 \cdot 1/|H|}{P(D)} = \frac{1 \cdot 1/|H|}{|VS_{H,D}|/|H|} = \frac{1}{|VS_{H,D}|}$

- See p161 for why $P(D) = |VS_{H,D}| / |H|$

Then: $P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$

Or: Every consistent hypothesis = a MAP hypothesis

- Intuitively: Absent any prior knowledge, all hypotheses that are consistent with D are equally likely, and inconsistent hypotheses have probability 0.

- So: Every consistent learner outputs a MAP hypothesis, assuming uniform prior prob. distr. over H & deterministic, noise-free training.
- Find-S for instance does this. It also outputs MAP hyp. over other $P(h)$ & $P(D|h)$ because it favors maximally specific hypotheses, which we may encode into our priors.
- This is similar to how we analyze inductive bias. (the set of assumptions sufficient to deductively justify inductive inference by learner)
- We model here instead by "probabilistic reasoning" and expressing those assumptions in $P(h)$ and $P(D|h)$ about what hypotheses a learner 'prefers'

2016-02-12, CS7641

Mitchell Ch. 6 (cont)

- Section 6.4: Any learning algorithm that minimizes squared error outputs a <u>maximum likelihood</u> hypothesis.
  - This relates particularly to the assumption of normally-distributed (Gaussian) noise, zero-mean. It differs for other noise distributions.
  - Max. likelihood hypothesis might not be MAP hypothesis (but it is if one assumes uniform priors over hypotheses)
  - N.B. This assumes noise <u>only in target values</u> - not in attributes describing instances. (e.g. if predicting person's weight based on age & height, it assumes noisy weight measurements but perfect measurements of age & height.)

See "Central Limit Theorem" for why. Also, normal dists. is just easier.

6.5

- We sometimes need nondeterministic/probabilistic functions, e.g. $f: X \to \{0,1\}$ where X = medical patients & their symptoms and $f(x)=1$ if they survive the disease & $f(x)=0$ otherwise, but with some level of unpredictability (e.g. with identical conditions maybe 92% survive and 8% die).
- Perhaps we want an ANN or other $\mathbb{R}$ approximator which outputs <u>probability</u> $f(x)=1$, i.e. $f': X \to [0,1]$ where $f'(x) = P(f(x)=1)$.
- End result: $h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} h(x_i)^{d_i} \left(1-h(x_i)\right)^{1-d_i}$

Generalization of binomial distribution

where d_i is observed 0 or 1 value for $f(x_i)$ from training data $D = \{\langle x_1, d_1\rangle \ldots \langle x_m, d_m\rangle\}$

- ln is monotonic so this is equivalent to:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} d_i \ln h(x_i) + (1-d_i) \ln (1-h(x_i))$$

- Note that this is just the negation of cross-entropy. (Ch. 3)

- Minimum Description Length
  - Already seen: $h_{MAP} = \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$

  Equivalent to: $h_{MAP} = \operatorname{argmin} \left(-\log_2 P(D|h) - \log_2 (P(h)) \right)$

  - From Shannon & Weaver (1949), if probability of message i is p_i, the optimal code (minimizing message length) assigns $-\log_2 p_i$ bits to message i.
    Let $L_C(i)$ = number of bits required to encode i w/ code C
  - Per this, we can see $-\log_2 P(h)$ from above as $L_{C_H}(h)$ where C_H is optimal code for H.
  - and $(-\log_2 P(D|h)) = L_{C_{D|h}}(D|h)$ where $C_{D|h}$ is optimal code for describing data D assuming sender & receiver know h.
  - Thus... $h_{MAP} = \underset{h}{\operatorname{argmin}} L_{C_H}(h) + L_{C_{D|h}}(D|h)$

    and $h_{MDL} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$

    $h_{MAP} = h_{MDL}$ if we choose 'optimal' C_1 & C_2.

  - So, somehow, this recommends choosing a hypothesis that minimizes description lengths. Or: Choose the shortest method for re-encoding training data, counting both hypothesis size & any data encoding given it.
  - The example on p173-174 treats this as a way to trade off errors & size, and in the process avoid overfitting. (eg. longer hypothesis might perfectly classify training data, shorter one might have some errors).

2016-02-12b, CS7641

Ch. 6 (cont.)

- Two questions.
  1: What is most probable hypothesis given the training data?
  2: What is most probable classification of new instance given the training data?
- We've approached #1. We could try to answer question #2 by simply applying that MAP hypothesis to new data. However, we can do better.
- Consider $P(h_1|D) = 0.4$, $P(h_2|D) = 0.3$, $P(h_3|D) = 0.3$.
  h_1 is then the MAP hypothesis (has highest posterior probability)
  - Consider new x for which $h_1(x) = 1$, $h_2(x) = 0$, $h_3(x) = 0$.
  - Based on this, probability of positive x is 0.4 (it's just h_1), while of negative x is 0.6 ($h_2 + h_3$). This is different from what $h_1(x)$, the MAP hypothesis, says.
- To generalize to Bayes optimal classification:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

  where V is set of possible classifications & $v_j \in V$
  - Anything classifying new instances per that is a Bayes optimal classifier/learner. No other classification method w/ same hypothesis space & priors can outperform one on average.
  - A curious property: It can predict hypotheses not in H, because it effectively uses linear combinations of multiple $h \in H$.
- Gibbs Algorithm is an alternative, faster but less optimal (Bayes optimal classifier is quite costly):
  - Choose $h \in H$ at random according to posterior prob. distr. over H.
  - Use that h to predict new x's classification.
  - Expected misclassification error is at most twice that of Bayes optimal classifier

Again because of that $|H|$ dependence.

- Naive Bayes classifier/learner
  - Highly practical! comparable to ANN & decision tree learning.
  - Applies to learning tasks where each instance x is described by conjunction of attribute values, and $f(x) \in V$ for finite set V.
  - Training examples each have tuple of attribs, $\langle a_1, a_2, \ldots a_n \rangle$, with some $v \in V$, we want to predict v for new instance & new tuple.
  - $v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2, \ldots a_n) \overset{\text{Bayes theorem}}{=} \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2, \ldots a_n | v_j) P(v_j)$
    - $P(v_j)$ is easy to estimate from training data (just count frequency)
    - $P(a_1, a_2 \ldots a_n | v_j)$ however is very difficult to estimate (unless we have a ridiculously large amount of training data).
    - The naive part says: Attrib values are conditionally independent.
      $P(a_1, a_2, \ldots a_n | v_j) = \prod_i P(a_i | v_j)$.
  - Thus, naive Bayes classifier: $v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$
    Then, we may much more readily estimate $P(a_i | v_j)$ from training data.
    - Note that no search of the hypothesis space is involved here!
    - If conditional independence assumption holds, this is identical to MAP classification too.

6.9.1 p178 for example

- We estimated probability with n_c/n. - n_c = # occurred, n = # possible.
  This is a poor estimate when n_c is small, and if $n_c = 0$ for some low probability it will produce a 0 probability that dominates product.
- To avoid this bias, use m-estimate: $\frac{n_c + mp}{n + m}$
  p = prior estimate of the probability (assume uniform if no information)
  m = equivalent sample size (how much to weight p)
- This adds 'virtual samples' to augment distribution, hence 'equivalent sample size'

2016-02-12c

Mitchell
Ch. 6 (cont.)

- Probabilistic approaches like naive Bayes classifier are some of the most effective for classifying text.
  - example in 6.10, p180
- Bayesian Belief Networks
  - Naive Bayes classifier works great when attribs $a_1, a_2, \ldots a_n$ are conditionally independent given target value v, and even when this doesn't completely hold.
  - Sometimes though this isn't appropriate.
  - Bayesian belief networks let this conditional independence apply to subsets of variables, rather than dispensing with it altogether.
  - Consider arbitrary set of random variables $Y_1 \ldots Y_n$, each $Y_i \in V(Y_i)$ (V is just set of possible values).

cross product = Cartesian product?

  - Joint space of Y is cross product $V(Y_1) \times V(Y_2) \ldots V(Y_n)$
    - i.e. each item of joint space = one possible assignment to tuple of vars $\langle Y_1 \ldots Y_n \rangle$
  - Joint probability distribution is that over this joint space, describing probability of each $\langle Y_1 \ldots Y_n \rangle$.
  - Bayesian belief network describes joint probability distr. for a set of variables.
  - Conditional independence:
    - Let X, Y, Z be discrete-valued random vars. X is conditionally independent of Y given Z if prob. distr. for X is independent of value of Y given a value for Z... or:

Don't ignore the x_i, y_j, z_k!

$$(\forall x_i, y_j, z_k)\, P(X = x_i \mid Y = y_j, Z = z_k) = P(X = x_i \mid Z = z_k) \qquad x_i \in V(X),\ y_j \in V(Y),\ z_k \in V(Z)$$

$$= P(X \mid Y, Z) = P(X \mid Z)$$

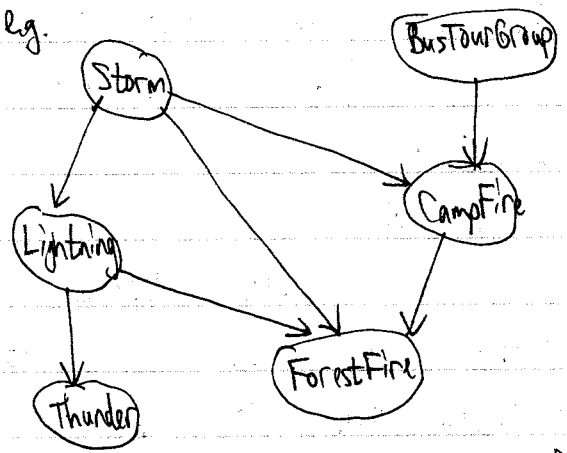
2016-02-12d,e moved to 2016-03-02
(I read all of Ch. 6, not realizing we'd cover the EM section later)

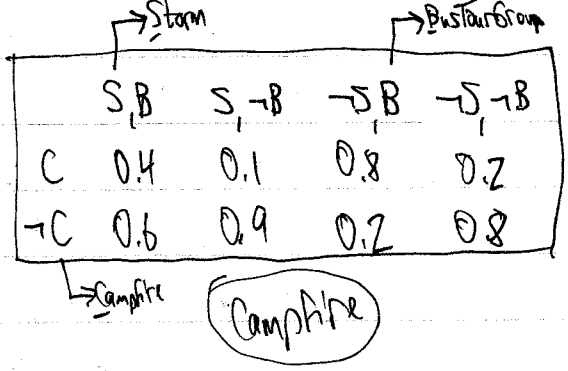
- Extends to sets too. Set $X_1 \dots X_\ell$ is independent of set $Y_1 \dots Y_n$ given set $Z_1 \dots Z_n$ if: (conditionally)

$$P(X_1 \dots X_\ell \mid Y_1 \dots Y_m, Z_1 \dots Z_n) = P(X_1 \dots X_\ell \mid Z_1 \dots Z_n)$$

- Note that naive Bayes classifier is just a form of this.
- Bayesian network = Bayesian belief network: Specifies conditional independence assumptions (via a DAG) & sets of local conditional probabilities

eg.





- Campfire is a descendant of Storm & BusTourGroup. It is conditionally dependent on those (with table giving those probabilities), and independent of its nondescendants, given its immediate parents.
- For any $\langle y_1, \dots y_n \rangle$, joint probability $P(y_1, \dots y_n) = \prod_{i=1}^{n} P(y_i \mid \text{Parents}(Y_i))$
  - DAG supplies Parents(Y_i), table gives $P(y_i \mid \text{Parents}(Y_i))$.
- Bayesian networks can express causal knowledge - like that lightning causes thunder (as above).
- Inference
  - Bayesian networks can give us a probability distribution for some target variable (given observed values of other variables).
  - In general, we may compute probability distribution for any subset of network variables, given any subset of the remaining variables.

But exact probabilities are NP-hard to compute.

2016-02-12f, CS7641

SL9

- Bayesian Learning
  - Goal: Learn <u>best</u> hypothesis given data & some domain knowledge
    - Best = most probable?
  - $Pr(a,b) = Pr(a|b)P(b)$
    But $Pr(a,b) = Pr(b,a) = Pr(b|a)P(a)$ → Bayes theorem
  - $Pr(D|h)$ is sort of a subtlety. We take h as a given but we sort of take $x_1, x_2, \ldots x_n$ as also given, but <u>not</u> its labels. We want the probability of D under that - which is basically just the probability of seeing the labels.
    This is much easier than $Pr(h|D)$ to consider, in many ways.
  - Our domain knowledge goes into the priors, $P(h)$.
  - In the 'cancer' example, the test is not particularly useful by itself, but is very useful in conjunction with other priors (e.g. drawing from a population known to have higher chances of that cancer).
  - Given $\{\langle x_i, d_i \rangle\}$
    $d_i = f(x_i) + \epsilon_i$ & want to find f
    $\epsilon_i \sim N(0, \sigma^2)$ ← i.i.d.
    $h_{ML} = \underset{h}{\operatorname{argmax}}\ P(h|D)$
    $= \underset{h}{\operatorname{argmax}}\ P(D|h)$ ← Assume uniform prior
    $= \underset{h}{\operatorname{argmax}} \prod_i P(d_i|h)$ ← Since ϵ is independent
    $= \underset{h}{\operatorname{argmax}} \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(d_i - h(x_i))^2/\sigma^2}$
    - Note that h is an approximation of f
    - If we have some d_i, the likelihood of it is completely determined by our model & how far away it is

$\lg = \log_2$ (so goes Isbell's notation)

- Can be simplified by removing some constant factors

- So, apply a logarithm. $(\Pi \to \Sigma,\ e^x \to x \ldots)$

$= \text{argmax} \sum_i -\frac{1}{2}\left(d_i - h(x_i)\right)^2 / \sigma^2$

$= \text{argmax} -\sum_i \left(d_i - h(x_i)\right)^2 = \text{argmin} \sum_i \left(d_i - h(x_i)\right)^2$

- This is just minimizing sum-of-squares!
  - So, this is the route to maximum likelihood hypothesis.
    We derive many methods from minimizing sum of squares
- But also: if you minimize sum-of-squares, you are assuming independently-distributed, zero-mean Gaussian noise

- One way to think of $\underbrace{-\lg\left(P(D|h)\right)}_{\text{length}(D|h)} + \underbrace{-\lg\left(P(h)\right)}_{\text{length}(h)}$:
  The first term is sort of 'error' or 'residual' - it is how much additional information one requires in order to 'express' the data from the hypothesis.
  So this is optimizing for the simplest hypothesis that expresses the data.
  A larger hypothesis might lower the error at the cost of a 'bigger' hypothesis; smaller one might make for higher error, but be 'shorter' to express.

- Voting $h \to$ Bayes optimal classifier (you cannot do better)
- Look carefully at whether we're looking for best hypothesis, or best classification. Usually we want the latter.
- Bayes' rule sort of lets us swap cause & effect, and work with probability of data given a hypothesis (weighted by priors) - rather than probability of hypothesis given data, which is more difficult.

2016-02-13, CS7641

SL 10

- Joint Distribution example:

Storm	Lightning	
T	T	0.25
T	F	0.40
F	T	0.05
F	F	0.30

$P(\neg storm) = 0.35$

$P(lightning|storm) = \frac{0.25}{0.65} = 0.384$

(~~Make Chi Stats?~~)

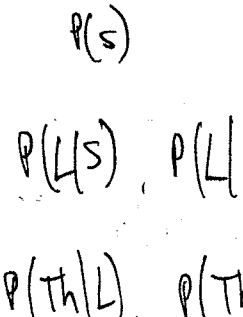
- Maybe we add thunder to this too...
- It's much more convenient to 'factor' these though.
- "Normal" independence: $Pr(X,Y) = Pr(X)\,Pr(Y)$

  Chain rule: $Pr(X,Y) = Pr(X|Y) \cdot Pr(Y)$

  $\Rightarrow Pr(X|Y) = Pr(X)$

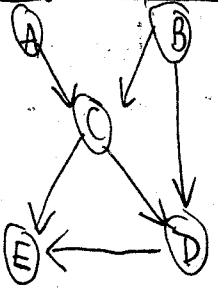
Compare w/ conditional independence - independence, given some knowledge!

- Independence: joint distribution between two variables is equal to their marginals
- Belief Networks = Bayes Nets = Bayesian Networks = Graphical Models



degree?

- This grows exponentially with vars
- 'Dependence' here does not refer directly to causality!



Sample:

$A \sim P(A)$

$B \sim P(B)$

$C \sim P(C|A,B)$

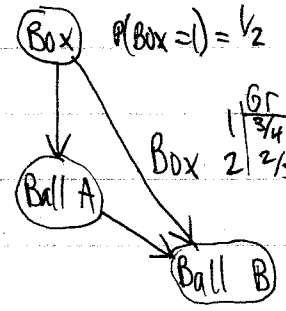
$D \sim P(D|B,C)$

$E \sim P(E|C,D)$

- Note that this is topological order. Dependencies go first. Also, must be directed & acyclic!

- This representation also lets you find all joint distributions (any combination of variables)
  - It just happens to be more compact. (in general)
  - CPTs, conditional probability tables
- Why sampling?
  - Distributions are good for:
    - Finding probability of producing some value
    - Producing values consistent with some process
  - ~~Sampling~~ Simulating of a complex process
  - Approximate inference (exact is hard, approximate is faster)
  - Visualization (or more generally intuition)
- Inferencing Rules:
  - Marginalization: $P(x) = \sum_y P(x,y)$
    Chain rule: $P(x,y) = P(x)\,P(y|x)$
    Bayes rule: $P(y|x) = P(x|y)\,P(y)\,/P(x)$
  - $Y \rightarrow X$ corresponds to $P(x,y) = P(y)\,P(x,y)$
- Example:
  - 2 boxes, box 1 has 4 balls (3 green, 1 orange), box 2 has 5 (3 blue, 2 green).
  - Pick a box at random, and a ball at random. (A)
  - Now, draw another ball from the same box. (B)
  - What is $P(B=\text{blue}|A=\text{green})$? $= P(B=\text{blue}|\text{Box}=1, A=\text{green})\,P(\text{Box}=1|A=\text{green})$ (use Bayes' rule) $+ P(B=\text{blue}|\text{Box}=2, A=\text{green})\,P(\text{Box}=2|A=\text{green})$

Note that A & B both depend on box.



$P(\text{Box}=1) = 1/2$

Ball A	Gr	Or	Bl
Box 1	3/4	1/4	0
Box 2	2/5	0	3/5

If Ball A = Gr:

Ball B	Gr	Or	Bl
Box 1	2/3	1/3	0
Box 2	1/4	0	3/4

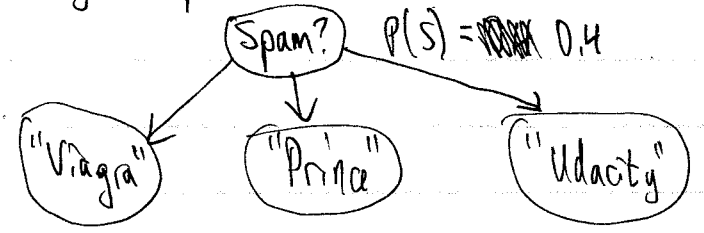
$= 15/23\,(0) + 8/23\,(3/4)$

$= \frac{6}{23}$

2016-02-13b, CS7641

SL10 (cont.)

- Naive Bayes: Special case



$P(V|S) = 0.3$ $P(P|S) = 0.2$ $P(U|S) = 0.0001$

$P(V|\neg S) = 0.001$ $P(P|\neg S) = 0.1$ $P(U|\neg S) = 0.1$

- This works to tell us probability of words, given something is spam
- $P(\text{spam?} \mid \text{Viagra, not prince, not udacity})$ - What is this?
  $= P(\text{Viagra, not prince, not udacity}\mid \text{spam?})\, P(\text{spam?}) / \ldots$
  $= P(\text{Viagra}\mid\text{spam?})\, P(\text{not prince}\mid\text{spam?})\, P(\text{not udacity}\mid\text{spam?})\, P(\text{spam?}) / \ldots$
- General solution:
  $P(V \mid a_1, a_2, \ldots a_n) = \Pi_i P(a_i|V) \cdot P(V) / Z$
  - MAP class $= \underset{V}{\text{argmax}}\, P(V) \Pi_i P(a_i|V)$
- Why naive Bayes is cool:
  - Inference is cheap! (it's normally not)
  - Few parameters
  - Estimate params. w/ labeled data (just count them)
  - Connects inference & classification!
  - Empirically successful
- But, naive Bayes models no interrelationship between attributes.
  - Once we're down to classifications, the exact probabilities don't really matter. (It does matter for inference)

- So, naive Bayes uses unrealistic assumptions, but because its results are concerned more with labels than with accurate probabilities, it still works well in practice.

Mitchell Ch. 9

- Genetic Algorithms / Randomized Optimization
  - GAs (Genetic Algorithms) provide a learning method which biological evolution inspired. They don't search hypotheses in a general-to-specific or simple-to-complex order, rather, they repeatedly mutate & recombine parts of best currently known hypotheses.
  - Some favorable parts:
    - Evolution in general is a successful, robust method of adaptation.
    - GAs can search hypothesis spaces with complex, interacting paths, where impact of each part is difficult to model.
    - GAs parallelize easily
  - A variant: genetic programming (this & GA are <u>evolutionary computation</u>)
- In GAs, one searches hypothesis space for "best" hypothesis as defined by some "fitness" - e.g. accuracy over training data, or number of chess games won against other people.
- The pool of hypotheses is the "population" - this is iteratively updated & evaluated with fitness function. New population is produced probabilistically, selecting most 'fit' individuals, and using some for crossover & mutation.
- Table 9.1 (p251) gives a rough algorithm

2016-02-13c, CS7641

Mitchell Ch. 9 (cont.)

- Parameters of note:
  - Fitness function (hypothesis → score)
  - Fitness threshold (when to stop)
  - p: Number of hypotheses in population
  - r: Fraction of population replaced by Crossover at each step
  - m: Mutation rate
- At 'Select' step, $Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$ for distribution of new population. → "fitness proportionate selection"

This sounds like particle filters from CS8470 are a form...

- Hypotheses in GAs often use bit strings to facilitate crossover & mutation. This could even encode if/then (for instance) by using a bit to specify a constraint in some attribute. Usually, we also want every bit string to represent a well-defined hypothesis.
- Typical GA operators:
  - Single-point crossover (string of bits exchanged for another hypothesis' bits)
  - Two-point crossover
  - Uniform crossover
  - Point mutation (single random bit-flip)
- 'Tournament selection' is sometimes used rather than fitness proportionate, or 'rank selection' (see p256)
- GABIL system, sec. 9.3
  - Hypotheses = disjunctive set of propositional rules ?

  e.g. IF $a_1 = T \wedge a_2 = F$ THEN $c = T$; if $a_2 = T$ then $c = F$

  →

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

- GABIL's crossover had to be modified because these are variable-length.
  - Its Fitness(h) = (correct(h))^2

2016-02-19

- GABIL also added some different mutation operators to help w/ variable-length bits (e.g. dropping conditions, adding conditions)

beam search?

- GAs employ 'randomized beam search', which is very different from other hypotheses searches we've done.
  - e.g. backpropagation for ANNs moves 'smoothly', and newer hypotheses are similar. A GA search might move very abruptly and produce 'offspring' hypotheses totally different from parent.
- They thus avoid issues of local minima, but do have issues with crowding: an individual that is more fit than others quickly reproduces, and it & very similar ones take over the population, reducing diversity & slowing progress.
  - Ways around this: tournament selection, rank selection, "fitness sharing" in which fitness of individual is reduced by nearby similar individuals, restricting recombination to less-similar individuals
- Population Evolution & Schema Theorem
- Schema theorem provides a characterization over time of a GA's population
  - schemas - patterns describing sets of bit strings w/ 0, 1, * (= don't care)
    - e.g. 0*10 schema → {0010, 0110}, and bit string 0010 can be seen as having 2^4 schemas - 00**, 0*10, ****, etc.
  - Let m(s,t) denote # of instances of schema s in population at time t (the tth generation)
  - Let f(h) = Fitness of bit string h, $\bar{f}(t)$ = average fitness of all individuals at time t

2016-02-15(b), CS7641

Mitchell Ch. 9 (cont.)

- Let n = # of individuals in population
- $h \in s \cap p_t$ ≡ h is both representative of schema s, and a member of the population at time t
- $\hat{u}(s,t)$ = average fitness of instances of schema s at time t
- Note that by eq. 9.1, $Pr(h) = f(h)/n\bar{f}(t)$
- Then $Pr(h \in s) = \sum_{h \in s \cap p_t} f(h)/n\bar{f}(t) = \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)$ by definition of $\hat{u}$

- Note that at the next step we pick n new individuals, and $Pr(h \in s)$ above is the probability that one is in s.

E = expected value

So: $E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \quad (= n \cdot Pr(h \in s)|_{t=t+1})$

→ $\hat{u}$ goes up, m goes up.
$\bar{f}$ goes up (entire population is more fit), m goes down.

- That is just selection step. If crossover & mutation also matter: $E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}$

p_c = prob. that single-point crossover will be applied to any individual
p_m = " " arbitrary bit is mutated
$o(s)$ = # of defined bits in s (0,1 = defined, * is not)
l = length of individual bit strings
$d(s)$ = distance between leftmost & rightmost defined bits in s

- See p261, 9.4.1, for details I don't feel like copying
- This inequality does not consider positive effects of crossover & mutation.

- Genetic Programming
  - Individuals in population are computer programs, not just bit strings
  - e.g. parse trees of the program; need to choose nodes for functions (descendants = args), & terminals
  - Crossover might randomly replace subtrees between two programs
  - Koza (1992) used this with a sort of stack machine to rearrange letters into a correct order
  - It has also been applied to circuit design (using SPICE simulation to determine fitness).
  - In general, results depend crucially on choice of representation & fitness function
- Lamarckian processes can improve GAs sometimes
  - Baldwin effect (talking of actual biology):
    - If species evolves in changing environment, this puts evolutionary pressure favoring individuals that can learn during lifetime.
    - Individuals that can <u>learn</u> many traits will rely less strongly on genetic code to 'hard-wire' traits, and this can make for a more diverse gene pool & support more rapid evolutional adaptation. (Thus: Learning can indirectly speed up overall evolution.)
    - Some attempts were made to add this to GAs
- Parallelizing GAs

coarse-grain reduces crowding

  - GAs parallelize well; the <u>coarse-grain</u> approach divides up the population into 'demes' and each group runs normally (with less frequent crossover between demes); <u>fine-grained</u> approaches might use one processor per individual.