Lecture 1: The Learning Problem (08/31/18)

- Ex: Predicting how a viewer will rate movie; 10% improvement = 1 million dollar price
    - Essence of machine learning:
        - A pattern exists
        - We cannot pin it down mathematically
        - We have data available
    - One approach
        - Viewer = vector of factors
            - Likes comedy? Likes action? Prefers blockbusters? Likes certain actor?
        - Movie = same vector of factors
        - Match movie and factors, add contributions from each factor, predict rating
        - Not really ML though!
    - Learning approach
        - Starts from rating
        - Initialize viewer and movie randomly
        - Nudge vectors toward the rating
        - Repeat for large num of ratings
- Components of learning
    - Ex: Credit approval
        - Applicant information: age, gender, annual salary, …
        - Input: x (customer application)
        - Output: y (good/bad customer?)
        - Target function: f: X->Y (ideal credit approval formula)
        - Data: (x1, y1), (x2, y2),…,(xN,yN)
        - Hypothesis: g: X->Y
        - Set of candidate formulas (Hypothesis set H)
            - Why not just pick from anything?
                - No downside for including hypothesis; can just make H set of all possible hypotheses
                - Having it will make theory go through if successful
        - Learning algorithm and hypothesis set are the tools we have control over
            - Hypothesis set H = {h}, g in H
            - Learning algorithm
            - Together = learning model
        - X = (x1, …, xd)
        - Approve credit if sum{i = 1 to d} w_i*x_i > threshold; deny otherwise
        - h(x) = sign((sum{i=1 to d} w_i*x_i)-threshold)
        - Assume linear separable data
        - Choose w0 = -threshold
            - Now, we can sum from i = 0 to d
            - h(x) = sign(w^T*x)
        - Perceptron Learning Algorithm
            - Pick a misclassified point h(x) != y_n

            - Update the weight vector w <- w + y_n*x_n
            - Maybe 1 change ruins classification of other points??
            - Just keep picking misclassified points and run PLA iteration until everything is classified correctly
- Basic premise of learning
    - Using a set of observations to uncover an underlying process
- Types of learning
    - Supervised vs. Unsupervised vs. Reinforcement Learning
    - Supervised
        - Outputs of inputs explicitly given
    - Unsupervised
        - Only inputs given (i.e. clustering)
        - High-level representations of data
    - Reinforcement
        - Given input, some output, and grade for this output
        - Important in playing games
- Is it too good to be true?
    - Model can fit all data points but be incorrect outside of data!
- Q&A
    - Linear separability
        - Very simple assumption, usually not true in practice; can also make points linearly separable; should assume that it is not linearly separable
        - Simple perceptron could never converge if data is not
    - Perceptron converges more and more slowly as number of dimensions grows
    - Cannot know if there is a pattern usually
        - Tell by running learning algorithm
        - Should not look at data usually
    - Pick optimization method based on problem
    - Hypothesis set can be both continuous or discrete
    - Amount of data available usually not under control
    - Perceptron are good for generalize but bad computationally
    - Model selection at end of lectures

Lecture 2: Is Learning Feasible? (09/01/18)

- Review
    - No pattern: we can try learning and it will fail
    - If we have the mathematical target function, we can do it but it's not the best way
    - We must have data!
- Ex: Consider bin with red and green marbles
    - P[red] = mu, P[green] = 1-mu
    - Mu unknown
    - Pick N marbles independently; fraction of red marbles in samples = nu
    - Does nu say anything about mu
        - No! Nu does not say anything about mu; sample can be mostly green while bin is mostly red
        - In a big sample (large N), mu is probably close to mu (within eps)
        - Possible vs. probable
        - Formally, **P[|nu - mu| > eps] <= 2*e^{-2*eps^2*N} (Hoeffding's Inequality)**
        - In other words, the statement "mu=nu" is P.A.C. (probably, approximately, correct)
        - Part of law of large numbers
        - Bound does not depend on mu!! Very advantageous since mu is unknown
        - Tradeoff between N and eps; smaller eps requires larger N to preserve bound
        - Nu approx. mu $\rightarrow$ mu approx. nu
- Connection to learning
    - Bin: Unknown is number mu
    - Learning: The unknown is a function f: X -> Y
    - Connection
        - Each marble is a point x in X
        - Bin is X
        - Green: Hypothesis got it right h(x) = f(x)
        - Red: Hypothesis got it wrong h(x) != f(x)
        - Probability distribution P on X
            - Use to generate x_1 up to x_N independently
        - NOT done yet though!! h is fixed
            - For this h, nu generalizes to mu
            - This is verification of h, not learning!
            - No guarantee nu will be small
            - Need to choose hypothesis from multiple hs
    - Next try
        - Multiple bins
        - Generalize bin model to more than one hypothesis h1, h2, h3, h4,…, hM
        - Both mu and nu depend on which hypothesis h
            - Nu is 'in sample' denoted by E_in(h)
            - Mu is 'out of sample' denoted by E_out(h)

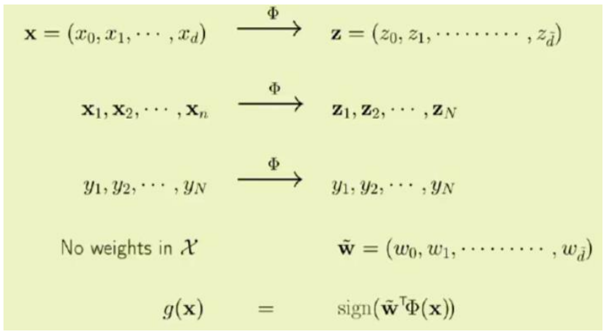
            - Heoffding inequality becomes P[| E_in(h) – E_out(h)| > eps] < 2*e^{-2*eps^2*N}
        - Still not good enough! Hoeffding does NOT apply to multiple bins
            - Coin analogy: If you toss a fair coin 10 times, what is probability that you will get 10 heads? 1/(2^10) appox. 0.1%
            - If you toss 1000 fair coins 10 times each, what is probability that some coin will get 10 heads? ~63%
            - From coins to learning: coins == bins; all green = not a perfect hypothesis
            - P[| E_in(g) - E_out(g)| > eps] <= P[| E_in(h1) - E_out(h1)| > eps **or** | E_in(h2) – E_out(h2)| > eps **or** …] <= sum{m=1 to N}(P[|E_in(hm)-E_out(hm)| > eps]) <= sum{m=1 to N}(2*e^{-2*eps^2*N})
            - $$\mathbb{P}[\,|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon\,] \leq \sum_{m=1} \mathbb{P}[|E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \epsilon] \leq \sum_{m=1}^{M} 2e^{-2\epsilon^2 N}$$
            - $$\mathbb{P}[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$$
            - The M is not good since this increases probability of something bad happening
- Q&A
    - If Hoeffding gives trivial inequality, then perhaps eps too stringent? Reassess parameters
    - M is usually infinite!! This is only first step, so we deal with this later on.
    - nu = mu vs. mu = nu; mu should determine nu, so technically nu is close to mu. What we’re doing is we are doing is using the fact that nu is close to mu to get that mu should also be close to nu (we can use nu to estimate mu)
    - Hoeffding shows that verification is feasible, modified Hoeffding shows that learning is feasible

Lecture 3: The Linear Model I (09/01/18)

- Review
  - Learning feasible in probabilistic sense
  - More hypotheses means looser bound (higher probability of bad event)
- Real data set
  - From zip codes in postal office, 16x16 images with numbers
  - Input representation
    - **x** = (x0, x1, x2, x3, x4, …, x256)
    - Linear model (w0, w1, w2, …, w256) → too many!
  - Features: Extract useful information
    - Intensity and symmetry **x** = (x0, x1, x2)
      - Linear model (w0, w1, w2)
    - x1: Intensity
    - x2: Symmetry
- What PLA does: evolution of E_in and E_out
  - Data not linearly separable, so PLA E_in and E_out fluctuates
  - Force PLA to stop at 1000th iteration
  - Hope that E_in approximates E_out
    - PLA tends to have good generalization as a simple model
- The 'pocket' algorithm
  - Keep the best candidate throughout PLA iterations and report that as final hypothesis (put best solution in pocket)
- Linear regression
  - Ex: Credit again
    - Classification: credit approval (yes/no)
    - Regression: credit line (dollar amount)
    - Input: **x** = applicant data (i.e. age, annual salary, …)
    - Linear regression output: h(x) = sum{i=0 to d}(w_i*x_i) = w^T*x
    - $$h(\mathbf{x}) = \sum_{i=0}^{d} w_i\, x_i = \mathbf{w}^{\mathsf{T}}\mathbf{x}$$
    - Don't need to threshold it like in classification
    - Data set: credit officers decide on credit lines
      - (x1, y1), (x2, y2), …, (xN, yN)
      - yn in R is credit line for customer xn
    - How to measure error:
      - How well does h(x) = w^T*x approximate f(x)
      - In linear regression, we use squared error (h(x)-f(x))^2
      - In-sample error: E_in(h) = (h(xn)-yn)^2
      - E_in(w) = 1/N*sum{n=1 to N} (w^T*xn-yn)^2 = 1/N*|Xw-y|^2
      - $$\begin{aligned} E_{\text{in}}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n)^2 \\ &= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2 \end{aligned}$$

      - Minimizing E_in
        - E_in(w) = 1/N*|X*w-y|^2
        - grad E_in(w) =2/N*X^T(X*w-y) = 0 vector
        - X^T*X*w =X^T*y
        - W = X^dag*y where X^dag = (X^T*X)^-1*X^T
        - X^dag is the pseudo-inverse of X
        - $$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$
          $$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\mathrm{X}^{\intercal}(\mathrm{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$
          $$\mathrm{X}^{\intercal}\mathrm{X}\mathbf{w} = \mathrm{X}^{\intercal}\mathbf{y}$$
          $$\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\intercal}\mathrm{X})^{-1}\mathrm{X}^{\intercal}$$
    - Linear regression algorithm
      - Construct the matrix X and vector y from data set as follows
      - Computer pseudo-inverse X^dag
      - Return w = X^dag*y
    - Linear regression for classification
      - LR learns a real valued function y = f(x) in R
      - Binary-valued functions are also real-valued!
      - Use LR to get w where w^T*xn approx. yn = +/- 1
      - In this case, sign(w^T*xn) is likely to agree with yn = +/- 1
      - Maybe good initial weights for classification!
- Nonlinear transformation
  - Linear is limited
  - Ex: Credit line
    - Credit line is affected by years in residence but not in a linear way!
    - Nonlinear [[xi < 1]] and [[xi > 5]] (two binary variables) are better
    - Can we do that with linear models?
      - Algorithms work because of linearity of weights, since xs are constant!
  - Transform data nonlinearly given (x1, x2)
    - (x1, x2) ->phi-> (x1^2, x2^2)
    - Weights still linear!!!!!! Therefore, we can still apply linear regression/PCA
    - There is a catch to be discussed later
- Q&A
  - Why is w0 included? Lines does not always pass through origin; w0 allows lines not going through origin; essentially a bias
  - E_in can be assessed directly (pick best in-sample error), E_out can be estimated with methods discussed later
  - Polynomial fit is a nonlinear transformation

Lecture 4: Error and Noise (09/01/18)

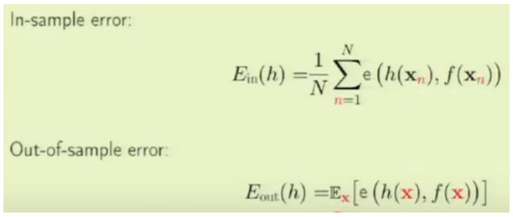
- Review
  - Linear models use the 'signal' w^T*x
    - Classification: h(x) = sign(signal)
    - Regression: h(x) = signal
  - Linear regression algorithm
    - w = (X^T*X)^-1*X^T*y
  - Nonlinear transformation
    - w^T*x is linear in w, not necessarily x since x is considered constant when training
- Continued nonlinear transformation
  - Need to convert regression in Z space back to X space
  - g(x) = g~(phi(x)) = sign(w~^T*phi(x))
  - **x** = (x0, x1, …, xd) → **z** = (z0, z1, …, zd')
  - **x1, x2, …, xn → z1, z2, …, zn**
  - y1, y2, …, yn → y1, y2, …, yn
  - No weights in X, w~ = (w0, w1, …, wd')
  - g(x) = sign(w~^T*phi(x))
  - 

    $$\mathbf{x} = (x_0, x_1, \cdots, x_d) \xrightarrow{\Phi} \mathbf{z} = (z_0, z_1, \cdots\cdots\cdots, z_{\tilde{d}})$$

    $$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \xrightarrow{\Phi} \mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_N$$

    $$y_1, y_2, \cdots, y_N \xrightarrow{\Phi} y_1, y_2, \cdots, y_N$$

    No weights in $\mathcal{X}$ $\quad \tilde{\mathbf{w}} = (w_0, w_1, \cdots\cdots\cdots, w_{\tilde{d}})$

    $$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^{\mathsf{T}}\Phi(\mathbf{x}))$$
- Error measures
  - What does "h approx. f" mean?
  - Error measure: E(h, f)
  - Almost always pointwise definition: e(h(x), f(x))
    - Ex: squared error e(h(x), f(x)) = (h(x)-f(x))^2
    - Ex: binary error e(h(x), f(x)) = [[h(x) != f(x)]]
  - Pointwise to overall: average of pointwise errors
  - In-sample error: E_in(h) = 1/N*sum{n=1 to N}e(h(xn), f(xn))
  - Out-of-sample error: E_out(h) = E_x[e(h(x), f(x))]
  - 

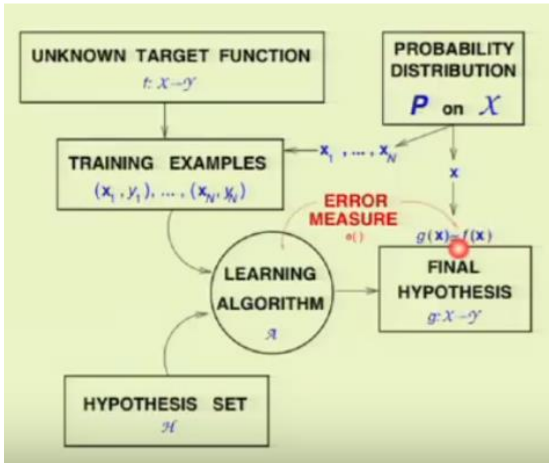
    In-sample error:

    $$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N} \mathrm{e}\left(h(\mathbf{x}_n), f(\mathbf{x}_n)\right)$$

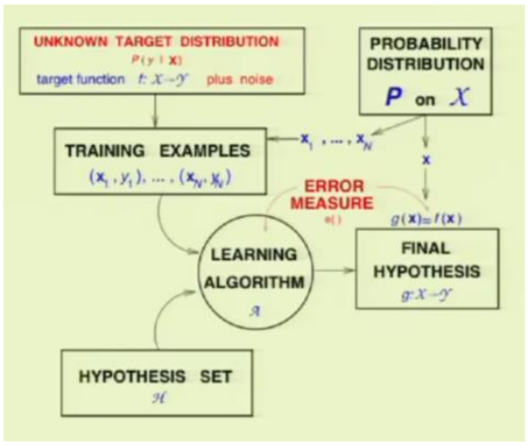
    Out-of-sample error:

    $$E_{\text{out}}(h) = \mathbb{E}_{\mathbf{x}}\left[\mathrm{e}\left(h(\mathbf{x}), f(\mathbf{x})\right)\right]$$
  - Apply error measure to final hypothesis
- How to choose error measure?
  - Ex: Fingerprint verification

- Two types of error: false accept, false reject
- How do we penalize each type?
- Ex: Use in supermarkets for discounts
    - False reject is costly; risk losing customer
    - False accept is minor; gave away a discount and intruder left fingerprint
    - Penalize false accepts only by 1, false rejects by 10
- Ex: Use in CIA for security
    - False accept is a disaster; national security at stake
    - False reject can be tolerated; just try again: you are an employee
    - Penalize false accepts by 1000, false rejects by 1
- tl;dr: Error measure should be specified by user!
    - Not always possible
    - Alternatives
        - Plausible measures: squared error === gaussian noise
        - Friendly measures: closed-form solution, convex optimization

- New diagram
- 

- Noisy targets
    - Target function is not always a function
    - Possible that identical inputs give different results
    - Target 'distribution'
        - Instead of y = f(x) use target distribution P(y | **x**)
        - (**x**, y) now generated by the joint distribution P(**x**)P(y | **x**)
        - Noisy target = deterministic target f(x) = E(y | **x**) plus noise y-f(**x**)
        - Deterministic target is a special case of noisy target P(y | **x**) is zero except for y = f(**x**)
    - FINAL DIAGRAM

- 

- Distinction between P(y | **x**) and P(**x**)
  - Both convey probabilistic aspects of **x** and y
  - The target distribution P(y | **x**) is what we are trying to learn
  - The input distribution P(**x**) quantifies relative importance of **x**
  - Merge P(**x**)P(y | **x**) as P(**x**, y) mixes the two concepts

- Preamble to theory
  - We know:
    - Learning is feasible: It is likely that E_out(g) approx. E_in(g)
    - Is this learning? We need g approx. f, which means E_out(g) approx. 0
  - The 2 questions of learning
    - E_out(g) approx. 0 is achieved through E_out(g) approx. E_in(g) and E_in(g) approx. 0
    - 1. Can we make sure that E_out(g) is close enough to E_in(g)?
    - 2. Can we make E_in(g) small enough?
  - What the theory will achieve
    - Characterize feasibility of learning for infinite M
    - Characterize the tradeoff of increasing model complexity
- Q&A
  - Emphasis that P(**x**) will give to certain inputs will affect decisions of algorithm
  - Poor generalization measured by E_out-E_in, the generalization error

Lecture 5: Training vs. Testing (09/01/18)

- Review
    - Error measures e(h(**x**), f(**x**)) specified by user
    - In-sample E_in(h) is average of pointwise error measures
    - Out-of-sample E_out(h) is expected value of pointwise error measures
    - Noisy targets come from possibility of non-deterministic targets
    - Thus, consider y ~ P(y | **x**)
    - P(**x**, y) = P(**x**)P(y | **x**) generates the input points
    - E_out(h) is now E_{x,y}[e(h(**x**), y)]
- From training to testing
    - Testing: P[|E_in-E_out| > eps] <= 2*e^{-2*eps^2*N}
    - Training: P[|E_in-E_out| > eps] <= 2*M*e^{-2*eps^2*N}
    - Goal: replace M with something else to deal with infinites
        - M came from union bound of bad events Bm P[B1 or B2 or…or Bm]
        - We can improve on M because bad events are very overlapping!
        - We want the following to hold
        - $|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| \approx |E_{\text{in}}(h_2) - E_{\text{out}}(h_2)|$
    - Instead of the whole input space, we consider only finite set of input points and count the number of 'hypotheses', or dichotomies
    - Dichotomies == mini-hypotheses
        - A hypothesis h: X -> {-1, +1}
        - A dichotomy h: {**x1, x2,…,xN**} -> {-1, +1}
        - Number of hypotheses |H| can be infinite
        - Number of dichotomies |H(**x1, x2,…,xN**)|
        - Candidate for replacing M: 2^N
    - Growth function counts the most dichotomies on any N points
        - m_H(N) = max_{**x1,…,xN** in X} |H(**x1,…,xN**)|
        - Function satisfies m_H(N) <= 2^N
        - The growth function counts the <u>most</u> dichotomies on any N points

          $$m_{\mathcal{H}}(N) = \max_{\mathbf{x}_1,\cdots,\mathbf{x}_N \in \mathcal{X}} |\mathcal{H}(\mathbf{x}_1,\cdots,\mathbf{x}_N)|$$

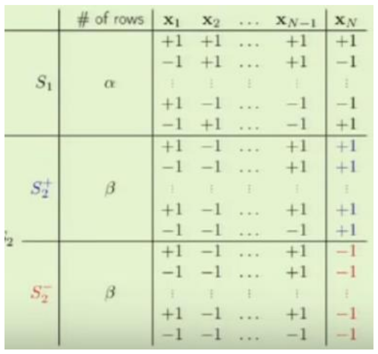
          The growth function satisfies:

          $$m_{\mathcal{H}}(N) \leq 2^N$$
    - Find growth function for perceptrons
        - N=3 → m_H(3) = 8 (find MAX value! Collinear does not matter)
        - N=4 → m_H(4) = 14
- Illustrative examples
    - Ex 1. Positive rays: All points >=a map to +1, all points <a map to -1
        - Generate some x1, x2, …, xN
        - m_H(N) = N+1
    - Ex 2. Positive intervals: All points x s.t. a <= x <= b map to 1, all else map to -1
        - m_H(N) = (N+1)*N/2+1 = N+1 choose 2+1 (need to consider a = b)

- Ex 3. Convex sets: H is set of h: R^2 -> {-1, +1}, any two points within region form line segment contained within region
  - h(x) = +1 is convex
  - Place all points on circle to MAXIMIZE number of convex regions
  - m_H(N) = 2^N
  - All possible dichotomies ➔ hypothesis set shattered the points
- Replace M with m_H(N)
  - P[|E_in-E_out| > eps] <= 2*M*e^{-2*eps^2*N}
    - m_H(N) is polynomial; eventually exponential dominates m_H → Good!

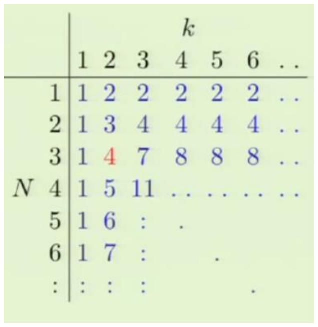
- Key notion: break point
  - Point at which we fail to get all possible dichotomies
  - If no data set of size k can be shattered by H, then k is a break point for H.
    - m_H(k) < 2^k
    - For 2D perceptrons, k = 4
    - Any larger k are also break points
  - Positive rays: break point k = 2 (2+1 < 2^2)
  - Positive intervals: break point k = 3 (3 choose 2 + 1 < 8)
  - Convex sets: break point k = infty
  - No break point → m_H(N) = 2^N
  - Any break point → m_H(N) is **polynomial** in N
- Puzzle
  - 3 points x1, x2, x3. Break point = 2.
  - Given this, how many dichotomies on 3 points? 4; lost half
- Q&A
  - Real valued functions addressed later
  - Shattering is good for fitting the data, bad for generalization

Lecture 6: Theory of Generalization (09/01/18)

- Review
    - Dichotomies: only consider data points, not entire input space
    - Growth function m_H(N) = maximum H(x1,…,xN) for all x1,…,xN in X
    - Break point: the point at which we cannot get all possible hypotheses/patterns
    - Every point > break point is also a break point
- Proof that m_H(N) is polynomial
    - Bounding m_H(N)
        - To show: m_H(N) is polynomial
        - We show: m_H(N) <= … <= … <= a polynomial
        - Key quantity: B(N, k): Maximum number of dichotomies on N points, with break point k
    - Recursive bound on B(N, k)
        - Table showing maximum number of dichotomies on N points, specially organized
        - 

	# of rows	$\mathbf{x}_1$	$\mathbf{x}_2$	…	$\mathbf{x}_{N-1}$	$\mathbf{x}_N$
		+1	+1	…	+1	+1
		−1	+1	…	+1	−1
S_1	α	⋮	⋮	⋮	⋮	⋮
		+1	−1	…	−1	−1
		−1	+1	…	−1	+1
		+1	−1	…	+1	+1
		−1	−1	…	+1	+1
S_2^+	β	⋮	⋮	⋮	⋮	⋮
		+1	−1	…	+1	+1
$_2$		−1	−1	…	−1	+1
		+1	−1	…	+1	−1
		−1	−1	…	+1	−1
S_2^-	β	⋮	⋮	⋮	⋮	⋮
		+1	−1	…	+1	−1
		−1	−1	…	−1	−1

        - B(N, k) = alpha + 2*beta
        - Estimating alpha and beta
            - Focus on x1 to x{N-1} columns
            - alpha + beta <= B(N-1, k)
        - Estimating beta by itself
            - Focus on S2 = S2+ U S2- rows
            - Argue that k-1 is break point for S2+ from x1 to x{N-1}
            - Since we can add both +/- 1 at the end, we must have 1 more column that has all possible patterns after adding xN
            - beta <= B(N-1, k-1)
        - Together, B(N, k) <= B(N-1, k) + B(N-1, k-1)
    - Numerical computation of B(N, k) bound
        - B(N, k) <= B(N-1, k) + B(N-1, k-1)
        - Base cases: B(N, 1) = 1, B(1, k) = 2 (if k != 1)
        - Table of first few values

- 

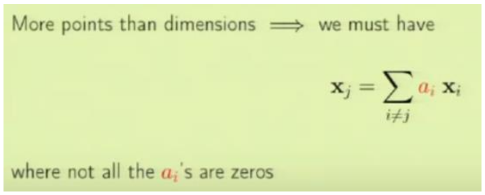
		k						
		1	2	3	4	5	6	..
	1	1	2	2	2	2	2	..
	2	1	3	4	4	4	4	..
	3	1	4	7	8	8	8	..
N	4	1	5	11	..	..	..	..
	5	1	6	:	.			
	6	1	7	:		.		
	:	:	:	:			.	

  - Analytical solution for B(N, k) bound
    - B(N, k) <= B(N-1, k) + B(N-1, k-1)
    - Theorem: B(N, k) <= sum[i=1 to k-1](N choose i)
    - $$B(N,k) \le \sum_{i=0}^{k-1} \binom{N}{i}$$
    - Boundary conditions: easy
    - Induction
      - Assume that formula holds for (N-1, k) and (N-1, k-1)
      - $$\begin{aligned}\sum_{i=0}^{k-1}\binom{N}{i} &= \sum_{i=0}^{k-1}\binom{N-1}{i} + \sum_{i=0}^{k-2}\binom{N-1}{i} \; ? \\ &= 1 + \sum_{i=1}^{k-1}\binom{N-1}{i} + \sum_{i=1}^{k-1}\binom{N-1}{i-1} \\ &= 1 + \sum_{i=1}^{k-1}\left[\binom{N-1}{i} + \binom{N-1}{i-1}\right] \\ &= 1 + \sum_{i=1}^{k-1}\binom{N}{i} = \sum_{i=0}^{k-1}\binom{N}{i}\end{aligned}$$
  - It is polynomial!
    - For given H, break point k is fixed
    - m_H(N) <= B(N, k) <= sum{i=0 to k-1}(N choose i) = O(N^{k-1})
  - Three examples
    - H is positive rays: (break point k = 2)
      - m_H(N) = N+1 <= N+1
    - H is positive intervals: (break point k = 3)
      - m_H(N) = ½*N^2 + ½*N + 1 <= ½*N^2 + ½*N + 1
    - H is 2D perceptrons (break point k = 4)
      - m_H(N) = ? <= 1/6*N^3 +5/6*N+1
- Proof that m_H(N) can replace M
  - What we want
    - Instead of M, use m_H(N) in Heoffding inequality
  - Pictorial proof
    - How does m_H(N) relate to overlaps
      - All hypotheses which result in the same dichotomy are redundant since we only know how hypothesis reacts to the points we have; dichotomies capture this redundancy

        - What to do about E_out?
            - Pick two samples: E_in(h), E'_in(h); these two track each other
        - Putting it together
            - **Vapnik-Cervonenkis Inequality**
            - **P[|E_in(g)-E_out(g)| > eps] <= 4*m_H(2*N)*e^{-1/8*eps^2*N}**
            - $\mathbb{P}[\,|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon\,] \ \leq \ 4 \ m_{\mathcal{H}}(2N) \ e^{-\frac{1}{8}\epsilon^2 N}$
- Q&A
    - If k = infty, no guarantee of learning using this analysis
    - Positive and negative rays problem has loose bound
    - Basically, we've shown that E_in and E_out are close given that there exists a break point

Lecture 7: The VC Dimension (09/02/18)

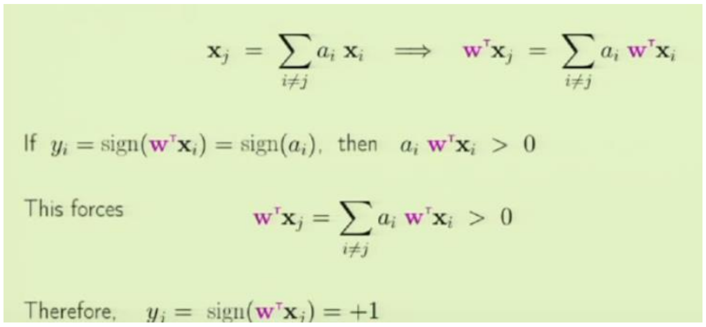
- Review:
  - m_H(N) is polynomial if H has a break point k
  - The VC inequality P[Bad event] = 4*m_H(2N)*e^{-1/8*eps^2*N} addresses overlaps in bad regions
- Definition
  - The VC dimension of a hypothesis set H, denoted by d_{VC}(H), is the largest value of N for which m_H(N) = 2^N "the most points H can shatter"
  - N <= d_{VC}(H) → H can shatter N points
  - k > d_{VC}(H) → k is a break point for H
  - Growth function:
    - In terms of k:
      - m_H(N) <= sum{i=0 to k-1}(N choose i)
    - In terms of VC dimension d_{VC};
      - m_H(N) <= sum{i=0 to dVC}(N choose i)
      - Maximum power is N^{dVC}
  - Ex: Positive rays: dVC = 1
  - Ex: 2D perceptrons: dVC = 3
  - Ex: Convex sets: dVC = infty
  - Relation to learning:
    - If dVC(H) is finite, then g in H will generalize
    - Independent of learning algorithm!
    - Independent of input distribution!
      - We are already picking the dist. of points which yields largest num of dichotomies
    - Independent of target function; we do not care what it is, it only generates the examples we use
- VC dimension of perceptrons
  - For d = 2, dVC = 3
  - In general, dVC = d+1
  - Prove two directions: dVC <= d+1, dVC >= d+1
  - Direction 1
    - Set of N=d+1 points in R^d shattered by perceptron
    - X = square matrix of d+1 d+1 dimensional points
    - $$\mathrm{X} = \begin{bmatrix} -\mathbf{x}_1^{\intercal}- \\ -\mathbf{x}_2^{\intercal}- \\ -\mathbf{x}_3^{\intercal}- \\ \vdots \\ -\mathbf{x}_{d+1}^{\intercal}- \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & & 0 \\ & \vdots & & \ddots & 0 \\ 1 & 0 & \dots & 0 & 1 \end{bmatrix}$$
    - X is invertible (det(X) = 1)
    - For any $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{d+1} \end{bmatrix} = \begin{bmatrix} \pm1 \\ \pm1 \\ \vdots \\ \pm1 \end{bmatrix}$, can we find a vector $\mathbf{w}$ satisfying
      $$\operatorname{sign}(\mathrm{X}\mathbf{w}) = \mathbf{y}$$

        - Solving for Xw = y, we get w = X^-1*y
        - Therefore, we can shatter these d+1 points
        - This implies that dVC >= d+1
    - Now, show dVC <= d+1
        - Need to show that we cannot shatter any set of d+2 points
        - Take any d+2 points
        - For any d+2 points, x1,…,x{d+1}, x{d+2}, we have more points than dimensions since each point is d+1 dimensional
        - More vectors than dimensions implies linear dependence
            - 

More points than dimensions $\Longrightarrow$ we must have

$$\mathbf{x}_j = \sum_{i \neq j} a_i \, \mathbf{x}_i$$

where not all the a_i's are zeros

        - Consider the following dichotomy: xi's with non-zero ai get yi=sign(ai) and xj gets yj=-1
        - Prove that no perceptron can implement such dichotomy
        - Linear dependence →
            - 

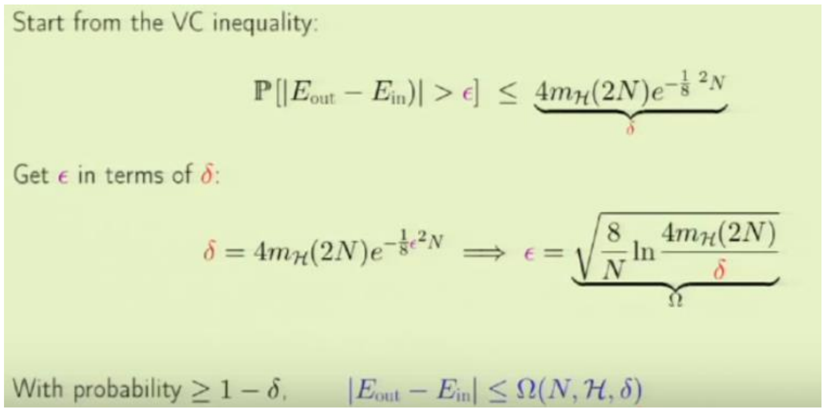
$$\mathbf{x}_j = \sum_{i \neq j} a_i \, \mathbf{x}_i \quad \Longrightarrow \quad \mathbf{w}^{\mathsf{T}}\mathbf{x}_j = \sum_{i \neq j} a_i \, \mathbf{w}^{\mathsf{T}}\mathbf{x}_i$$

If $y_i = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i) = \text{sign}(a_i)$, then $a_i \, \mathbf{w}^{\mathsf{T}}\mathbf{x}_i > 0$

This forces $$\mathbf{w}^{\mathsf{T}}\mathbf{x}_j = \sum_{i \neq j} a_i \, \mathbf{w}^{\mathsf{T}}\mathbf{x}_i > 0$$

Therefore, $y_j = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_j) = +1$

        - Therefore, we cannot shatter this set given ANY d+2 points
    - Therefore, dVC = d+1
        - What is d+1 in the perceptron? It is the number of parameters w0,w1,…,wd
- Interpreting the VC dimension
    - 1. Degrees of freedom
        - Parameters create degrees of freedom
        - Number of parameters: analog degrees of freedom
        - dVC: equivalent 'binary' degrees of freedom
        - Ex. Positive rays (dVC=1)
            - Choice of a is 1 degree of freedom
        - Ex. Positive intervals (dVC=2)
            - Choice of a and b is 2 degrees of freedom
        - NOT JUST PARAMETERS; parameters may not contribute degrees of freedom
        - Take chain of 4 1D perceptrons
            - 8 parameters
            - Only 2 degrees of freedom due to redundancy
        - dVC measures the **effective** number of parameters
    - 2. Number of data points needed

        - Existence of finite dVC means it is possible to learn
        - Two small quantities in the VC inequality: eps and RHS = delta
        - If we want certain eps and delta, how does N depend on dVC?
        - Let's look at N^dVC*e^{-N}
            - Plot
            - Fix N^d*e^{-N} = small value. How does N change with d?
            - We only get bound information in theory!
                - Practical observation: bigger VC dimension usually requires proportionally more examples
        - **Rule of thumb: N >= 10*dVC**
- Generalization bounds
    - Start from the VC inequality
    - Get eps in terms of delta
    - 

      Start from the VC inequality:

      $$\mathbb{P}[|E_{\text{out}} - E_{\text{in}})| > \epsilon] \leq \underbrace{4m_{\mathcal{H}}(2N)e^{-\frac{1}{8}\epsilon^2 N}}_{\delta}$$

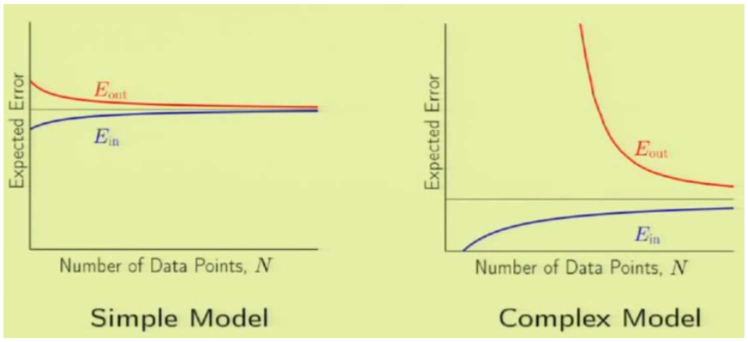
      Get ϵ in terms of δ:

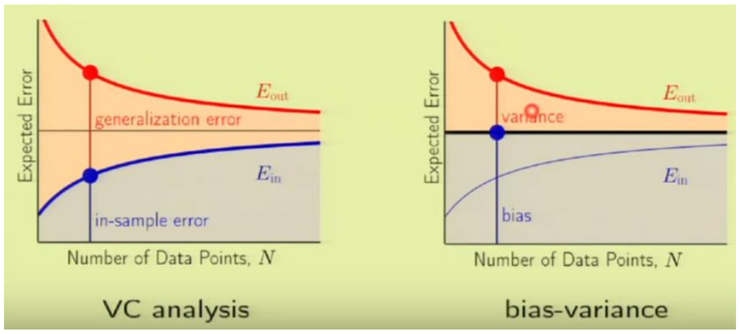
      $$\delta = 4m_{\mathcal{H}}(2N)e^{-\frac{1}{8}\epsilon^2 N} \implies \epsilon = \underbrace{\sqrt{\frac{8}{N}\ln\frac{4m_{\mathcal{H}}(2N)}{\delta}}}_{\Omega}$$

      With probability $\geq 1-\delta$, $\quad |E_{\text{out}} - E_{\text{in}}| \leq \Omega(N, \mathcal{H}, \delta)$
    - Take away absolute value
        - We are deliberately minimizing E_in in hopes of minimizing E_out, so should be positive
        - With probability >= 1-delta, E_out <= E_in + omega
            - E_in is lower and omega is higher with bigger hypothesis set
- Q&A
    - dVC usually not exactly known; instead, get some kind of estimate or bound

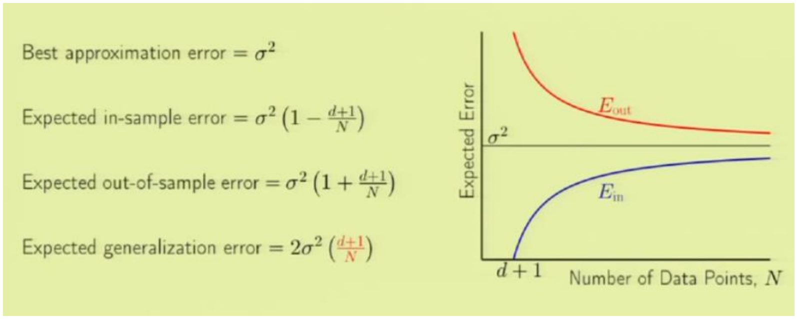
Lecture 8: Bias-Variance Tradeoff (09/02/18)

- Review
    - VC dimension dVC(H) = most points H can shatter
    - Scope of VC analysis includes training examples, final hypothesis, and hypothesis set
    - Rule of thumb: N >= 10*dVC
    - Generalization bound: E_out <= E_in + omega
- Bias and Variance
    - Approximation-generalization tradeoff
        - Small E_out: good approximation of f out of sample
        - More complex H → better chance of approximating f
        - Less complex H → better chance of generalizing out of sample
        - Ideal H = {f}
        - Quantifying the tradeoff
            - VC analysis: E_out <= E_in + omega
            - Bias-variance analysis: decomposing E_out into
                - 1. How well H can approximate f
                - 2. How well we can zoom in on a good h in H
            - Applies to real-valued targets and uses squared error
        - Start with E_out
            - E_out($g^{(D)}$) = $E_x[(g^{(D)}(x)-f(x))^2]$
            - E_D[E_out($g^{(D)}$)] = $E_D[E_x[(g^{(D)}(x)-f(x))^2]]$
            - E_D[E_out($g^{(D)}$)] = $E_x[E_D[(g^{(D)}(x)-f(x))^2]]$
            - Focus on inside quantity (E_D quantity)
        - Evaluate $E_D[(g^{(D)}(x)-f(x))^2]$
            - We define the 'average' hypothesis g~(x):
                - g~(x) = $E_D[g^{(D)}(x)]$
            - Imagine many data sets D1, D2,…,DK
                - g~(x) approx. 1/K*sum{k=1 to K}$g^{(DK)}$(x)
            - Using g~(x)
                - $E_D[(g^{(D)}(x)-f(x))^2]$ = $E_D[(g^{(D)}(x)$-g~(x)+g~(x)-f(x)$)^2]$
                - = …
                - = $E_D[(g^{(D)}(x)$-g~(x)$)^2]$ + (g~(x)-f(x)$)^2$
                - Measure distance from your fcn. to best hypothesis and distance of best hypothesis from target function
            - $$\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x})-f(\mathbf{x})\right)^2\right]=\underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x})-\bar{g}(\mathbf{x})\right)^2\right]}_{\mathrm{var}(\mathbf{x})}+\underbrace{\left(\bar{g}(\mathbf{x})-f(\mathbf{x})\right)^2}_{\mathrm{bias}(\mathbf{x})}$$
            - $$\mathbb{E}_{\mathcal{D}}\left[E_{\mathrm{out}}(g^{(\mathcal{D})})\right]=\mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x})-f(\mathbf{x})\right)^2\right]\right]$$
            - = E_x[bias(x)+var(x)] = bias + var
        - The tradeoff
            - Small hypothesis set → larger bias, smaller variance
            - Large hypothesis set → smaller bias, larger variance

        - Ex. f(x) = sin(pi*x), f: [-1, 1] -> R
        - Only give two training examples!
        - Two hypothesis sets
            - H0: h(x) = b
            - H1: h(x) = ax+b
        - E_out(H0) > E_out(H1) w/o considering learning
        - Learning
            - H0: midpoint
                - g~(x) = 0 with some variance
                - bias = 0.50, var = 0.25
            - H1: fit the line
                - g~(x) has better error, but much larger variance
                - bias = 0.21, var = 1.69
            - Which one is better for learning?
                - E[E_out] = 0.75 vs. 1.9
        - Match model complexity to data resources, not to the target complexity
- Learning Curves
    - Expected E_out and E_in
    - Data set D of size N
    - How do these vary with N?
    - 

    - Does not account for overfitting
    - VC vs. bias-variance analysis
    - 

    - Linear regression case
        - Noisy target y = w*^T*x+noise
        - Data set D = {(x1, y1),…,(xN,yN)}
        - Solution: w = (X^T*X)^-1*X^T*y
        - In-sample error vector = Xw-y

    - Out-of-sample error vector = Xw-y’ (y’ = y with different noise)
    - Best approx. error = sigma^2 due to noise
    - 

      Best approximation error $= \sigma^2$

      Expected in-sample error $= \sigma^2\left(1 - \frac{d+1}{N}\right)$

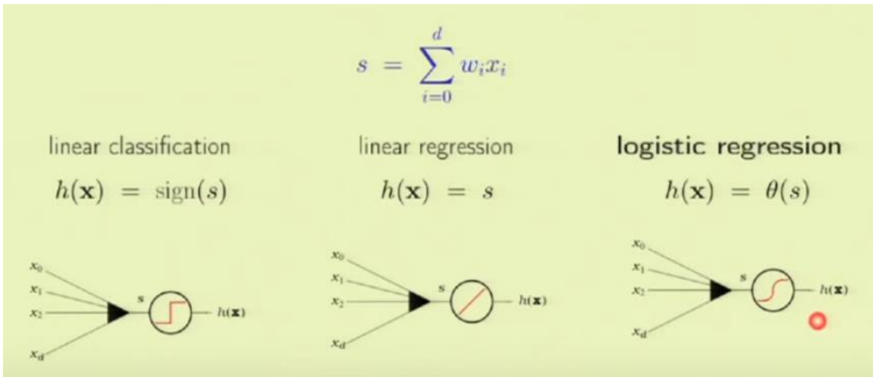
      Expected out-of-sample error $= \sigma^2\left(1 + \frac{d+1}{N}\right)$

      Expected generalization error $= 2\sigma^2\left(\frac{d+1}{N}\right)$

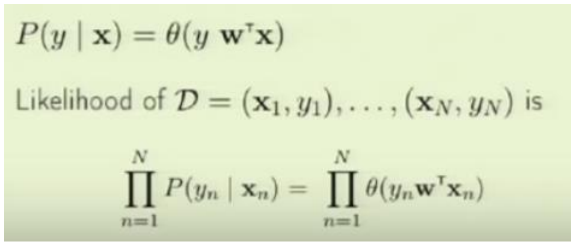
- Q&A
  - Complex models are better at minimizing error, but take more points to train and have larger generalization error
  - Clean formulas only valid for linear regression due to use of mean square error
  - Bootstrapping: generate large number of datasets and get the average to obtain approx. value of g~(x)
  - Bias-variance-covariance dilemma

Lecture 9: The Linear Model II (09/02/18)

- Review
  - Bias and variance: expected value of E_out w.r.t. D = bias + var
    - $g^{(D)}(x) \rightarrow g\sim(x) \rightarrow f(x)$
  - Learning curves: How E_in and E_out vary with N
    - Bias-Variance vs. VC interpretation
  - N needed is proportional to "VC dimension"
- Where we are
  - Linear classification [perceptron or pocket]
  - Linear regression [pseudo-inverse]
  - New: Logistic regression
  - Nonlinear transforms [almost done; address generalization issues]
- Generalization of nonlinear transforms
  - **x** = (x0, x1, …, xd) -> phi -> **z** = (z0, z1, …, z{d~})
  - Each zi = phi_i(x) → **z = phi(x)**
  - Example: **z** = $(1, x1, x2, x1x2, x1^2, x2^2)$
  - Final g(**x**) in X space:
    - Sign(**$w\sim^T$\*phi(x)**) or **$w\sim^T$\*phi(x)**
  - Price we pay:
    - **w~** has more dimensions/parameters
    - dVC for original is d+1
    - dVC for transform is <= d~+1
    - Two non-separable cases
      - First case: Mostly separable, with a few points not
        - Use a linear model in X; accept E_in > 0
        - OR insist on E_in = 0; go to high-dimensional Z
          - Go to fourth-order features
          - Not easily generalized!
      - Second case: Inherently non-linear points
        - Use second-order features
          - Why not **z** = $(1, x1^2, x2^2)$ instead of all second-order possibilities?
          - Or better yet **z** = $(1, x1^2+x2^2)$
          - Or even **z** = $(x1^2 + x2^2 - 0.6)$
        - Looking at the data means that you have acted as a learning algorithm in some capacity before passing it on to the model
          - Can be hazardous to E_out
          - VC inequality needs to be adjusted accordingly
          - Formally called data snooping
- Logistic regression
  - The model
    - Apply non-linearity to s between sign and identity function

- 

- Logistic function theta returns value between 0 and 1; usually outputs probabilities
- **theta(s) = $e^s/(1+e^s)$**
- Soft threshold: uncertainty
- Also called sigmoid (flattened out 's')
- Probability interpretation
    - h(**x**) = theta(s) is interpreted as a probability
    - Ex. Prediction of heart attacks
        - Input **x**: cholesterol level, age, weight, etc.
        - theta(s): probability of a heart attack
        - Signal s = $\mathbf{w}^T\mathbf{x}$ ("risk score")
    - This is genuine probability! Even during learning.
        - Data (**x**, y) with binary y, generated by a noisy target:
            - P(y | **x**) = {f(**x**) if y=+1; 1-f(**x**) if y=-1}
            - Target f: R^d -> [0, 1] is the probability
            - Learn g(**x**) = theta($\mathbf{w}^T\mathbf{x}$) approx. f(**x**)

- Error measure
    - For each (**x**, y), y is generated by probability f(**x**)
    - Plausible error measure based on likelihood:
        - If h = f, how likely to get y from **x**?
        - Same probability dist. as before, except with h instead of f
    - Formula for likelihood:
        - Substitute h(**x**) = theta($\mathbf{w}^T\mathbf{x}$), noting theta(-s) = 1-theta(s)
        - 

$$P(y \mid \mathbf{x}) = \theta(y\ \mathbf{w}^{\mathsf{T}}\mathbf{x})$$

Likelihood of $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ is

$$\prod_{n=1}^{N} P(y_n \mid \mathbf{x}_n) = \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n)$$

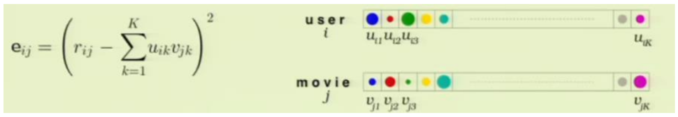
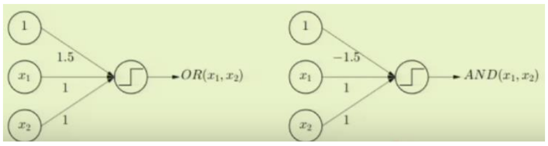
    - Maximizing the likelihood
        - Maximize the ln of the likelihood gives same result
        - Maximize 1/N* the result gives same result
        - Minimize -1* the result gives same result
        - Theta(s) = $1/(1+e^{-s})$

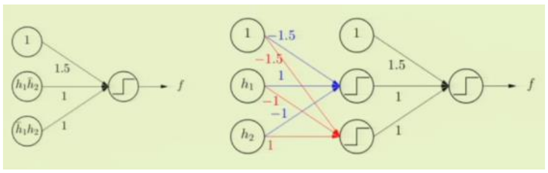
- Minimize $$-\frac{1}{N}\ln\left(\prod_{n=1}^{N}\theta(y_n\,\mathbf{w}^{\mathsf{T}}\mathbf{x}_n)\right)$$
  $$=\frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{1}{\theta(y_n\,\mathbf{w}^{\mathsf{T}}\mathbf{x}_n)}\right) \qquad \left[\theta(s)=\frac{1}{1+e^{-s}}\right]$$
  $$E_{\text{in}}(\mathbf{w})=\frac{1}{N}\sum_{n=1}^{N}\ln\left(1+e^{-y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)$$
- $$E_{\text{in}}(\mathbf{w})=\frac{1}{N}\sum_{n=1}^{N}\underbrace{\ln\left(1+e^{-y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right)}_{\mathrm{e}\left(h(\mathbf{x}_n),y_n\right)}$$
- Thus, we can minimize this quantity, the in-sample error
- Error measure called “cross-entropy” error

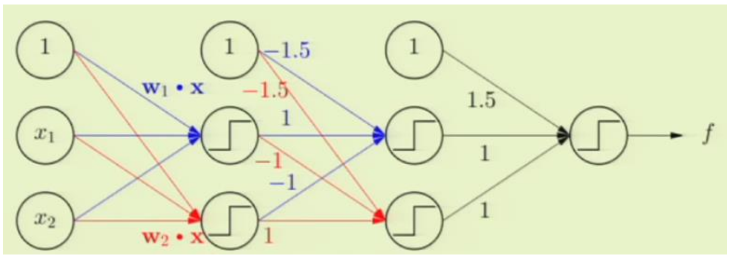
- Learning algorithm
  - How to minimize E_in
  - Compare to linear regression (squared error); could find closed solution
  - We cannot find closed-form solution here; instead use iterative approach
    - Gradient Descent
  - Gradient descent
    - General method for nonlinear optimization
    - Start at **w**(0); take a step along steepest slope
      - We only have local information! Not global information.
    - Fixed step size: **w**(1) = **w**(0) + eta***v^**
    - What is direction of **v^**?
      - $$\Delta E_{\text{in}}=E_{\text{in}}(\mathbf{w}(0)+\eta\hat{\mathbf{v}})-E_{\text{in}}(\mathbf{w}(0))$$
        $$=\eta\nabla E_{\text{in}}(\mathbf{w}(0))^{\mathsf{T}}\hat{\mathbf{v}}+O(\eta^2)$$
        $$\geq-\eta\|\nabla E_{\text{in}}(\mathbf{w}(0))\|$$
        Since $\hat{\mathbf{v}}$ is a unit vector,
        $$\hat{\mathbf{v}}=-\frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$
    - Fixed-size step?
      - Use a variable eta; start with large eta and end with small eta
      - Observation: eta should increase with the slope
      - Instead of fixed step, we have a **learning rate eta**
      - Instead of
        $$\Delta\mathbf{w}=\eta\,\hat{\mathbf{v}}$$
        $$=-\eta\frac{\nabla E_{\text{in}}(\mathbf{w}(0))}{\|\nabla E_{\text{in}}(\mathbf{w}(0))\|}$$
        Have
        $$\Delta\mathbf{w}=-\eta\;\nabla E_{\text{in}}(\mathbf{w}(0))$$

            - Logistic regression algorithm
                - Initialize weights at t = 0 to **w**(0)
                - for t = 0,1,2,… compute the gradient, update weights, iterate to next step until it Is time to stop
                - Return final weights **w**
- Summary of linear models
    - Ex: Credit analysis
        - Perceptron: approve or deny, classification error (PLA, pocket,…)
        - Linear regression: decide credit line, squared error (Pseudo-inverse)
        - Logistic regression: probability of default, cross-entropy error (Gradient descent)
- Q&A
    - Initialization of weights can be random or just 0 (most conservative)
    - Termination of gradient descent: can stop when steps become too small, have target error, limit number of iterations
    - Local vs. global minimum
        - Gradient descent will get to closest local minimum
        - Can just randomly initialize weights many times and go through algorithm
            - Usually finds local minimum close to global minimum
        - Finding global minimum is NP-hard
    - Logistic regression works for multi-class situations

Lecture 10: Neural Networks (09/02/18)

- Review
  - Logistic regression: same as linear regression except with soft threshold "sigmoid" function
  - Legitimate probability output
  - Likelihood measure based on cross-entropy error
  - Gradient descent can be used to minimize error iteratively
- Stochastic gradient descent
  - GD minimizes E_in(w) = 1/N*sum{n=1 to N}[e(h(xn), yn)] by iterative steps along direction of negative gradient
    - Considers all points
    - Label as "batch" GD
  - Pick one (**x**n, yn) at a time. Apply GD to e(h(**x**n), yn)
  - There is some average direction equal to -grad(E_in) (batch GD)
    - Randomized version of GD
  - Benefits
    - Much cheaper computation since we only consider 1 input point
    - Randomization is good for optimization since it can deal with shallow local minima; can possibly escape from them
      - Also good for flat regions
    - Simple
    - Rule of thumb: eta = 0.1 works
  - SGD in action
    - Ex. Movie ratings
      - Consider user and movie vectors
      - 
$e_{ij} = \left(r_{ij} - \sum_{k=1}^{K} u_{ik}v_{jk}\right)^2$
user i
$u_{i1}\, u_{i2}\, u_{i3}$
u_{iK}
movie j
$v_{j1}\, v_{j2}\, v_{j3}$
v_{jK}
- Neural network model
  - Biological inspiration
    - Biological function → Biological structure
    - Maybe we could use a network of perceptrons
    - Imitation has a limit (i.e. birds and planes)
  - Combining perceptrons
    - Use perceptrons to implement ORs and ANDs
    - 

    - Creating layers
    - 

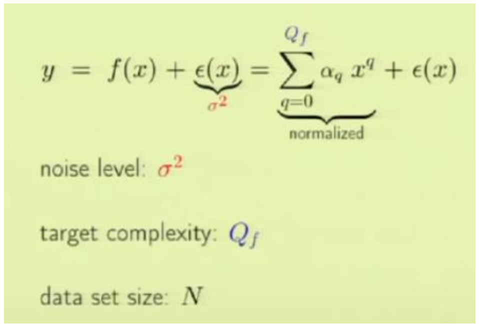
        - Multilayer perceptron
        - 

        - 3 layers, feed forward
    - A powerful model, but 2 red flags
        - Generalization
            - We can, however, directly compute VC dimension, so we just have to match number of examples
        - Optimization
            - Even for single perceptron, non-separable data causes lots of problems
    - Neural networks
        - Use soft thresholds, each layer has non-linearity
        - Intermediate values called hidden layers, 1 <= l <= L
        - Use tanh nonlinearlity
            - tanh(s) = $(e^s - e^{-s})/(e^s + e^{-s})$
            - Looks linear near beginning
        - $$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$
        - $$x_j^{(l)} = \theta(s_j^{(l)}) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right)$$
        - Apply **x** to layer 0 → → $x_1^{(L)}$ = h(**x**)
- Backpropagation algorithm
    - Apply SGD
        - All the weights determine h(**x**)
        - Error on example (**x**n, yn) is e(h(**x**n), yn) = e(w)
        - To implement SGD, just need grad of this quantity
        - $$\nabla \mathrm{e}(\mathbf{w}): \quad \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} \quad \text{for all } i, j, l$$
    - Computing the gradient
        - Can evaluate each one-by-one: analytically or numerically
        - Better if we can get ALL of them in one step
            - $$\frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, w_{ij}^{(l)}} = \frac{\partial\, \mathrm{e}(\mathbf{w})}{\partial\, s_j^{(l)}} \times \frac{\partial\, s_j^{(l)}}{\partial\, w_{ij}^{(l)}}$$

        - We have $\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}} = x_i^{(l-1)}$ We only need: $\frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}} = \delta_j^{(l)}$
    - Start with delta at output
        - For final layer l = L, j = 1: e(w) = e(h(**x**n), yn)m = e($x_1^{(L)}$, yn) = ($x_1^{(L)}$-yn)^2
        - $x_1^{(L)}$ = theta($s_1^{(L)}$)
        - theta'(s) = 1-theta^2(s)
    - Back propagation of delta
        - $$\begin{aligned} \delta_i^{(l-1)} &= \frac{\partial e(\mathbf{w})}{\partial s_i^{(l-1)}} \\ &= \sum_{j=1}^{d^{(l)}} \frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}} \times \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \times \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}} \\ &= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \times w_{ij}^{(l)} \times \theta'(s_i^{(l-1)}) \end{aligned}$$
          $$\delta_i^{(l-1)} = (1 - (x_i^{(l-1)})^2) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)}$$
        - Algorithm
            - Initialize all weights $w_{ij}^{(l)}$ AT RANDOM
            - For t = 0,1,2,…
                - Pick n in {1,2,…,N}
                - Forward: compute all $x_j^{(l)}$
                - Backward: compute all delta$_j^{(l)}$
                - Update the weights
                    - $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)}$ – eta*$x_i^{(l-1)}$delta$_j^{(l)}$
                - Iterate to next step until it is time to stop
            - Return final weights
- Final remark: hidden layers
    - Hidden layers apply nonlinear transform → creating higher-level features
    - These are **learned** features
        - VC dimension ~number of weights
    - Interpretation?
        - Very common for ML models to have difficult interpretations
        - What is the hidden layer doing?
            - Not easy to answer
- Q&A
    - Momentum is enhancement to batch GD by adding some part of 2nd order term
    - Conjugate gradient descent for 2nd order terms
    - Very little in terms of neural network research because it is mostly solved
        - Used widely in industry
    - Model selection

    - How many weights can I afford? Based on VC dimension.
    - How to distribute weights? Harder to answer.
- Why tanh? Want soft threshold within -1 to 1 that is easily differentiable.
- Final weights depend on initial weights and order of inputs
- Number of weights is approximately VC dimension
- Data snooping should be avoided or accounted for
- In-sample training is looking at the data, but this is okay as long as we have already accounted for the VC contribution of the weight space
- Early stopping prevents overfitting
- SGD: can choose random permutation for each epoch, can even do sequential version for each epoch

Lecture 11: Overfitting (09/02/18)

- Review
  - Multilayer perceptrons use logical combinations of perceptrons
  - Neural networks use tanh nonlinearity and SGD
  - Backpropagation makes networks less computationally intensive
- What is overfitting?
  - Ex. Simple target function
    - Simple 2nd degree target function, generate 5 data points with noise
    - 4th order polynomial fit – perfect in-sample fit: E_in = 0, E_out is huge
  - Overfitting vs. bad generalization
    - Overfitting happens when E_in decreases but E_out increases
    - Early stopping can be used in this case
  - Overfitting is fitting the data more than is warranted
    - Culprit: fitting the noise – harmful
  - Ex. Case study
    - 10th order polynomial target, generate 15 data points with noise
    - 50th order target, generate 15 data points **without** noise
    - Try 2nd order fit and 10th order fits
      - For 10th order target, very large E_out from increasing order of fit
      - For 50th order target, still very large E_out from increasing order of fit
        - Has a different type of noise!! Therefore still overfits.
    - Irony of two learners
      - Given that the target is 10th order, trying to fit with 10th order FAILS
      - Given that the target is 10th order, trying to fit with 2nd order WINS
    - Overfitting can also occur if not enough data points
    - Even without noise
      - Is there really no noise??
  - Ex. Detailed Experiment
    - Impact of noise level and target complexity
    - y = f(x) + eps(x) = f(x) + sigma^2
    - 

      $$y = f(x) + \underbrace{\epsilon(x)}_{\sigma^2} = \underbrace{\sum_{q=0}^{Q_f} \alpha_q\, x^q}_{\text{normalized}} + \epsilon(x)$$

      noise level: σ^2

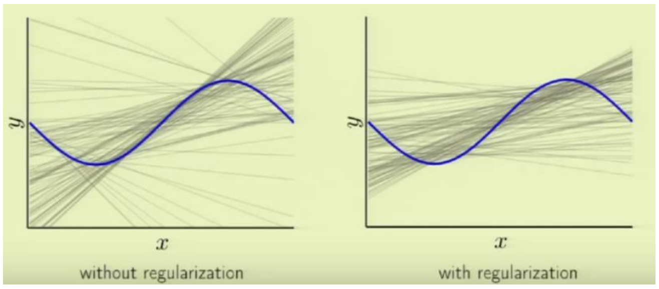
      target complexity: Q_f

      data set size: N
    - Overfit measure
      - Compare out-of-sample errors of g_2 in H_2 and g_{10} in H_{10}
      - Use E_out(g_{10}) – E_out(g_2)
- The role of noise
  - Increasing number of points decreases overfitting

- Increasing noise increases overfitting
- Increasing target complexity increases overfitting
- **Stochastic noise** is the noise in the target function
- **Deterministic noise** is the noise associated with target complexity

- Deterministic noise
  - The part of f that H cannot capture: f(x)-h’(x)
  - Many differences with stochastic noise:
    - 1. Depends on H
    - Fixed for a given **x**
  - Impact on overfitting
    - Finite N: H tries to fit the noise
  - Noise and bias-variance
    - What if f is a noisy target?
      - y = f(**x**) + eps(**x**), E[e(**x**)] = 0
    - var+bias+noise term which corresponds with stochastic noise
    - Bias corresponds to deterministic noise
- Dealing with overfitting
  - Regularization: putting the brakes
    - Restrained fit
  - Validation: checking the bottom line
- Q&A
  - Floating point error is a source of error, but so insignificant that it is not considered
  - Overfitting happens in the var part of bias-variance model
  - Tradeoff between validation set and training set size

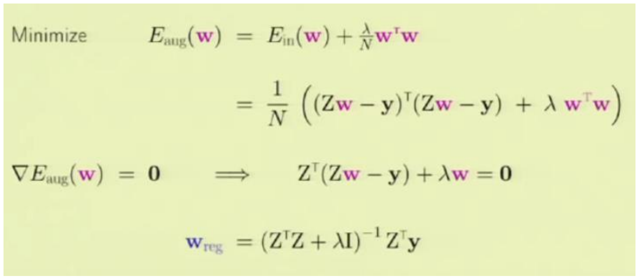
Lecture 12: Regularization (09/03/18)

- Review
    - Overfitting: fitting the data more than is warranted
        - VC allows it; doesn't predict it
    - Fitting the noise, stochastic/deterministic
    - Deterministic noise: function of limitations of our model
        - Get harmful noise
- Informal
    - Two approaches
        - Mathematical: ill-posed problems in function approximation
        - Heuristic
    - Ex. Sinusoid fit using two points and a line
        - Restrict lines in terms of offset and slope
        - 

        - A little more bias, but a lot less variance
- Formal
    - The polynomial model
        - Use Legendre polynomials
        - H_Q: polynomials of order Q
        - $$\mathbf{z} = \begin{bmatrix} 1 \\ L_1(x) \\ \vdots \\ L_Q(x) \end{bmatrix} \qquad \mathcal{H}_Q = \left\{ \sum_{q=0}^{Q} w_q \, L_q(x) \right\}$$
    - Unconstrained solution
        - Given (x1,y1),…,(xN,yN) → (z1,y1),…,(zN,yN)
        - Minimize $E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (\mathbf{w}^{\mathsf{T}}\mathbf{z}_n - y_n)^2$

          Minimize $\frac{1}{N} (\mathrm{Z}\mathbf{w} - \mathbf{y})^{\mathsf{T}}(\mathrm{Z}\mathbf{w} - \mathbf{y})$

          $\mathbf{w}_{\text{lin}} = (\mathrm{Z}^{\mathsf{T}}\mathrm{Z})^{-1}\mathrm{Z}^{\mathsf{T}}\mathbf{y}$
    - Constrained solution
        - Hard constraint: H_2 is constrained version of H_{10} with $w_q = 0$ for $q > 2$
        - Softer version: Total magnitude squared of weights <= C
            - Smaller VC dimension → Better generalization
            - "soft-order" constraint

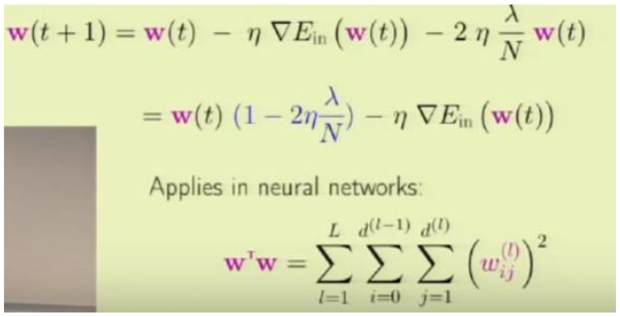
        - Minimize in-sample error given this constraint ($w^T w <= C$)
        - Solution: w_{reg} instead of w_{lin}
        - If w_{lin} already satisfies constraint, just return it
        - Otherwise, best solution at $w^T w = C$
            - grad(E_in(w_{reg})) proportional to $-w_{reg} = -2$*lambda/N*w_{reg}
            - grad(E_in(w_{reg})) + 2*lambda/N*w_{reg} = 0
            - Minimize E_in(w) + lambda/N*$w^T w$
                - C up → lambda down
        - Augmented error
            - Minimizing $E_{aug}(w) = E_{in}(w)$ + lambda/N*$w^T w$
                - Solves minimizing subject to constraint from VC formulation
            - 

Minimize $$E_{\rm aug}(\mathbf{w}) = E_{\rm in}(\mathbf{w}) + \tfrac{\lambda}{N}\mathbf{w}^{\rm T}\mathbf{w} = \frac{1}{N}\left((\mathrm{Z}\mathbf{w}-\mathbf{y})^{\rm T}(\mathrm{Z}\mathbf{w}-\mathbf{y}) + \lambda\, \mathbf{w}^{\rm T}\mathbf{w}\right)$$

$$\nabla E_{\rm aug}(\mathbf{w}) = \mathbf{0} \quad\Longrightarrow\quad \mathrm{Z}^{\rm T}(\mathrm{Z}\mathbf{w}-\mathbf{y}) + \lambda\mathbf{w} = \mathbf{0}$$

$$\mathbf{w}_{\rm reg} = (\mathrm{Z}^{\rm T}\mathrm{Z} + \lambda\mathrm{I})^{-1}\mathrm{Z}^{\rm T}\mathbf{y}$$

            - As lambda increases: overfitting → best lambda → underfitting

- Weight decay
    - Minimizing the previous error function is called weight decay
    - Gradient descent:
    - 

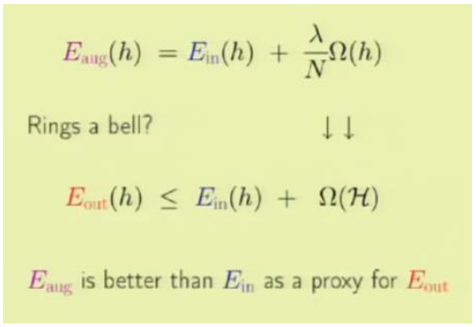
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta\,\nabla E_{\rm in}\left(\mathbf{w}(t)\right) - 2\,\eta\,\frac{\lambda}{N}\,\mathbf{w}(t)$$

$$= \mathbf{w}(t)\,\left(1 - 2\eta\frac{\lambda}{N}\right) - \eta\,\nabla E_{\rm in}\left(\mathbf{w}(t)\right)$$

Applies in neural networks:

$$\mathbf{w}^{\rm T}\mathbf{w} = \sum_{l=1}^{L}\sum_{i=0}^{d^{(l-1)}}\sum_{j=1}^{d^{(l)}}\left(w_{ij}^{(l)}\right)^2$$

    - Variations
        - Emphasis of certain weights
            - Add some coefficient gamma to weights
            - Neural networks: different layers get different gamma's
            - **Tikhonov regularizer: w^T*Gamma^T*Gamma*w**
        - Even weight growth!
            - Bad! Just results in lambda = 0.
            - Practical rule
                - Stochastic noise is 'high-frequency'
                - Deterministic noise is also non-smooth
                - → **Constrain learning towards smoother hypotheses**
                    - We want to punish noise more than actual signal
    - General form of augmented error

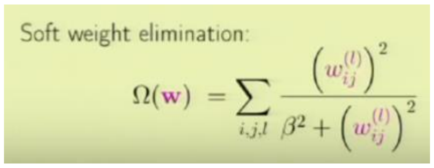
- 

  $$E_{\text{aug}}(h) = E_{\text{in}}(h) + \frac{\lambda}{N}\Omega(h)$$

  Rings a bell? $\downarrow\downarrow$

  $$E_{\text{out}}(h) \leq E_{\text{in}}(h) + \Omega(\mathcal{H})$$

  E_{aug} is better than E_{in} as a proxy for E_{out}

- Choosing a regularizer
  - Perfect regularizer Omega
    - Constraint in the direction of the target function
    - Guiding principle: Direction of smoother or simpler
    - What if we choose a bad Omega?
      - We still have lambda!
  - Must put some regularizer!!
  - Neural-network regularizers
    - Weight decay: From linear to logical
      - Small weights result in essentially a large linear function
    - Weight elimination
      - Fewer weights → smaller VC dimension
      - Soft weight elimination
  - 

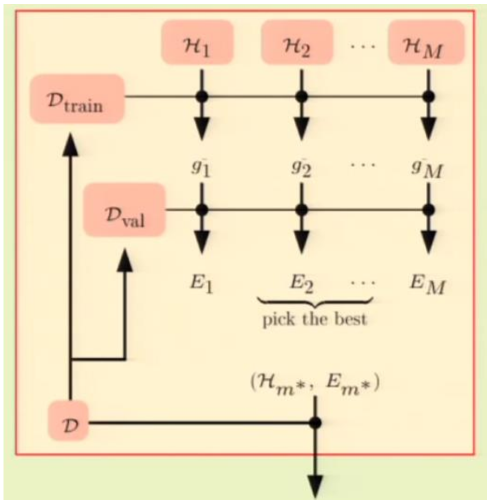
    Soft weight elimination:

    $$\Omega(\mathbf{w}) = \sum_{i,j,l} \frac{\left(w_{ij}^{(l)}\right)^2}{\beta^2 + \left(w_{ij}^{(l)}\right)^2}$$

  - Early stopping
    - Regularization through the optimizer
    - Stop using validation
  - Optimal lambda
    - More noise (either kind) requires larger lambda to optimize
- Q&A
  - Not data snooping because we're using a different part of data for validation
  - Can use multiple regularization techniques in one model
  - Regularization makes it so that we can maybe afford a higher dimension hypothesis with a good regularizer

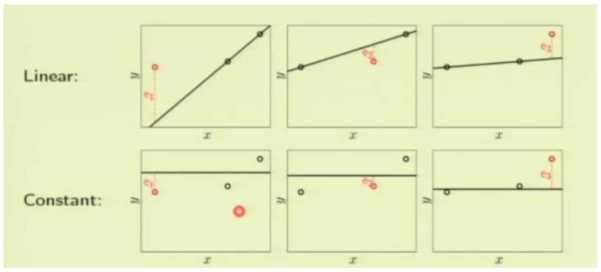
Lecture 13: Validation (09/03/18)

- Review
  - Regularization
    - Constrained → Unconstrained
    - Results in augmented error
  - Choose regularizer
    - Choose lambda in principled way using validation
    - Choose Omega heuristically to prefer smooth, simple h
- The validation set
  - Validation vs. regularization
    - E_out(h) = E_in(h) + overfit penalty
    - Regularization tries to estimate overfit penalty
    - Validation estimates E_out(h)
  - On out-of-sample point (**x**, y), the error is e(h(**x**), y)
    - Squared, binary,… error
    - E[e(h(**x**), y)] = E_out(h)
    - Var[e(h(**x**), y)] = sigma^2
  - From point to set
    - Use a validation set (**x1**, y1),…,(**xK**, yK)
    - Error is E_{val}(h) = 1/K*sum{k=1 to K}[e(h(**xk**), yk)]
    - Expected value is E_out(h)
    - Variance = 1/K^2*sum{k=1 to K}[var(e(h(**xk**), yk))] = sigma^2/K
      - Covariance between points is 0 since independent points
    - E_{val}(h) = E_{out}(h) +/- O(1/sqrt(K))
  - Every time we add a point to validation, we take it away from training set
    - Small K → bad estimate
    - Large K → ?
  - Put K back into N
    - The estimated Eout does not apply for actual final hypothesis, only reduced final hypothesis
    - Large K → bad estimate!
  - Rule of thumb: **K = N/5**
  - Why validate?
    - D_{val} is used to make learning choices
    - If an estimate of E_out affects learning
      - The set is no longer a test set!
  - What is the difference?
    - Test set is unbiased, validation set has optimistic bias
    - Two hypotheses h1, h2 with E_out(h1) = E_out(h2) = 0.5
    - Error estimates e1 and e2 uniform on [0, 1]
    - Pick h in {h1, h2} with e = min(e1, e2)
    - E[e] < 0.5 → Optimistic bias!
- Model selection
  - Use D_{val} more than once

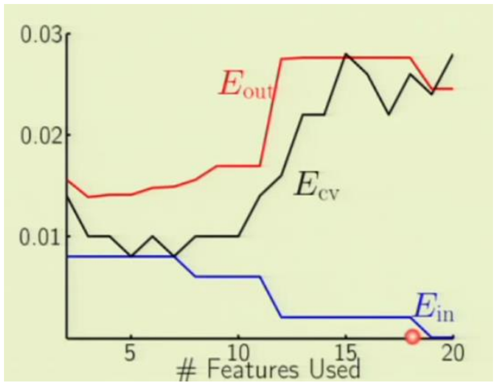
- M models H1,…,HM
- Use D_{train} to learn g_m^-
- Evaluate g_m^- using D_{val}
- 

- $E_{val}(g_{m^*}^-)$ is a biased estimate of $E_{out}(g_{m^*}^-)$
- Bigger K makes estimate more reliable, but reduces training set size
- How much bias?
  - Validation is used for "training" on the finalist models
  - $H_{val} = \{g_1^-, g_2^-,…,g_M^-\}$
  - Back to Hoeffding and VC!
  - E_out($g_{m^*}^-$) <= $E_{val}(g_{m^*}^-)$ + O(sqrt(ln(M)/K))
- Data combination
  - Error estimates: E_in, E_test, E_val
  - Contamination: optimistic bias in estimating E_out
    - Training set: totally contaminated
    - Validation set: slightly contaminated
    - Test set: total 'clean'

- Cross validation
  - Need small K so that g^- is close to g
  - Need large K to have reliable estimate of E_out from E_val
  - Can we have K both small and large?
  - Leave one out
    - N-1 points for training and 1 point for validation
    - Final hypothesis learned from D_n is g_n^-
    - $e_n = E_{val}(g_n^-) = e(g_n^-(xn), yn)$
    - Cross validation error
      - E_{CV} = 1/N*sum{n=1 to N}[e_n]
      - Average of all errors
      - N points in validation set while using N-1 points to train
      - Points not independent!!
        - However, effective number still very close to N

- 

- Constant model wins here in terms of CV error

- Cross validation in action
  - 

  - Better at generalization using validation
- Usually leave more than one out
  - Take data set and break into many folds (i.e. 10 folds)
  - Each time we use one fold for validation and rest for training
  - Therefore, we will have N/K training sessions on N-K points each

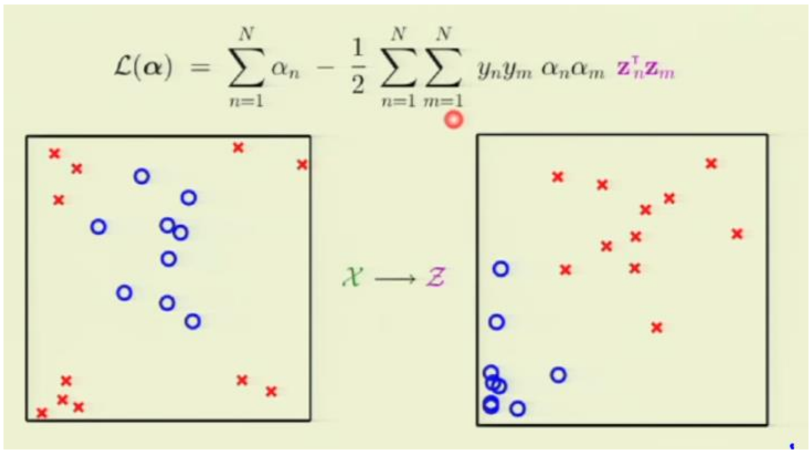
- Q&A
  - Why does model choice from validation not count as data snooping?
    - Because it is accounted for
  - Should use both regularization and validation
  - Validation does not require assumptions, unlike some other methods for model selection
  - Both validation and cross validation have bias for the same reasons
    - Cross validation allows us to use a lot of examples to validate and train, lowering the error bar of E_out estimate

Lecture 14: Support Vector Machines (09/03/18)

- Review
  - Validation: $E_{val}(g^{-})$ estimates $E_{out}(g)$
  - Data contamination: D_{val} is slightly contaminated since it is used to make decisions
    - Optimistic bias
  - Cross validation allows for large training set and large validation set simultaneously
- Arguably most successful classification technique in ML
- Maximizing the margin
  - Better linear separation
    - We can get different separating lines for the same dataset!
    - Add a margin around line to nearest point
    - We would prefer larger margins even though in-sample error is the same
    - Why is bigger margin better?
      - Chances are, new points will still be on the correct side of the line even with noise
    - Which w maximizes the margin?
  - Growth function: all dichotomies with any line
    - Dichotomies with fat margin
    - Fat margins imply fewer dichotomies possible
      - They have a smaller VC dimension
  - Find w with large margin
    - Let x_n be the nearest data point to the line $w^Tx = 0$
      - 2 preliminary technicities
        - Normalize w: require $|w^Tx_n| = 1$
        - Pull out w_0: $w = (w_1,\ldots,w_d)$ apart from b
          - The plane is now $w^Tx + b = 0$
    - Computing the distance
      - The distance between x_n and the plane $w^Tx + b = 0$ where $|w^Tx_n + b| = 1$
      - The vector w is perp. To the plane in the X space
        - Take x' and x'' on the plane
        - $w^Tx' + b = 0$ and $w^Tx'' + b = 0$
        - $\rightarrow w^T(x'-x'') = 0 \rightarrow w^T$ must be orthogonal to x'-x''
      - Take any point x on the plane
      - Find projection of x_n-x on w
        - $\hat{w} = w/|w| \rightarrow$ distance $= |\hat{w}^T(x_n-x)|$
      - $$\text{distance} = \frac{1}{\|\mathbf{w}\|}\left|\mathbf{w}^\intercal\mathbf{x}_n - \mathbf{w}^\intercal\mathbf{x}\right| = \frac{1}{\|\mathbf{w}\|}\left|\mathbf{w}^\intercal\mathbf{x}_n + b - \mathbf{w}^\intercal\mathbf{x} - b\right| = \frac{1}{\|\mathbf{w}\|}$$
    - The optimization problem
      - Maximize $1/|w|$, the margin, subject to $\min\{n=1,2,\ldots,N\}[|w^Tx_n + b| = 1]$
      - Find equivalent problem
        - Notice $|w^Tx_n + b| = y_n(w^Tx_n + b)$ since signal agrees with label
        - Minimize $½*w^Tw$ subject to $y_n(w^Tx_n + b) >= 1$ for $n = 1,2,\ldots,N$

- Minimum must be 1 because we can scale down w and b accordingly

- The solution
  - Constrained optimization
    - Lagrange? Inequality constraints → KKT
  - Relation between regularization and SVM
    - Regularization optimized E_out with constraint on $w^T w$
    - SVM optimized the other way around
  - Lagrange formulation
    - First, take inequality constraint and put it in 0 form
      - $$\text{Minimize} \quad \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^{\intercal}\mathbf{w} - \sum_{n=1}^{N} \alpha_n (y_n(\mathbf{w}^{\intercal}\mathbf{x}_n + b) - 1)$$
        $$\text{w.r.t. } \mathbf{w} \text{ and } b \text{ and maximize w.r.t. each } \alpha_n \ge 0$$
    - $$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = \mathbf{0}$$
      $$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0$$
    - Substituting in the Lagrangian
      - $$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^{\intercal}\mathbf{w} - \sum_{n=1}^{N} \alpha_n (y_n(\mathbf{w}^{\intercal}\mathbf{x}_n + b) - 1)$$
    - we get $$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\intercal}\mathbf{x}_m$$
      Maximize w.r.t. to $\boldsymbol{\alpha}$ subject to $\alpha_n \ge 0$ for $n = 1, \cdots, N$ and $\sum_{n=1}^{N} \alpha_n y_n = 0$
  - Quadratic programming
    - $$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^{\intercal}\mathbf{x}_m - \sum_{n=1}^{N} \alpha_n$$
    - Subject to: y^T*alpha = 0, 0 <= alpha <= infty
    - Word of warning: matrix is N^2
      - Slow for quadratic programming if N too large
    - QP hands us alpha = alpha1, …, alphaN
      - → w = sum{n=1 to N}[alpha$_n$y$_n$x$_n$]
    - KKT condition: For n = 1,…,N
      - Alpha$_n$(y$_n$(w^{T}x$_n$+b)-1) =0
      - Either slack or lagrange multiplier is 0
      - For all interior points, alpha will be 0
        - Slack is only 0 for nearest points on each side
      - alpha$_n$ > 0 → **x$_n$ is a support vector**
  - Support vectors
    - Closest **x**$_n$'s to the plane: achieve the margin

        - w = sum{n=1 to N}[alpha$_n$y$_n$x$_n$] (from before)
        - w = sum{**x**$_n$ is SV}[alpha$_n$y$_n$x$_n$] (from before)
            - End up with fewer parameters than initially
        - Solve for b using any SV
            - $y_n(w^T x_n+b)=1$
- Nonlinear transforms
    - Can handle non-separable case by transforming from x to z space
    - Can go to large space without paying the price
    - 

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m\, \alpha_n \alpha_m\, \mathbf{z}_n^{\mathsf{T}}\mathbf{z}_m$$

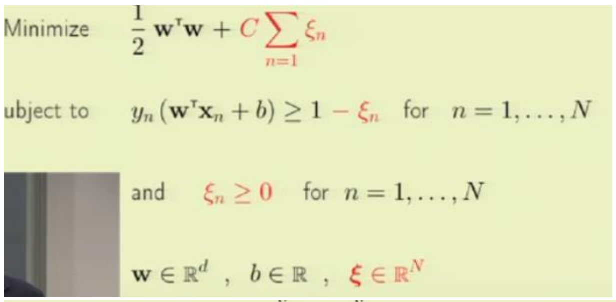
    - "Support vectors" in X space
        - Support vectors live in Z space
        - In X space, "pre-images" of support vectors
            - The margin is maintained in Z space
        - Generalization behavior goes with number of SVs
        - Generalization result: **E_out <= (# of SVs)/(N-1)**
            - Actual result is expected value of these
- Q&A
    - Why can we normalize w?
        - Scale of w does not matter for defining the plane
        - Does not result in loss of generality
        - Not absolutely necessary, just makes calculations easier
    - What if points are not linearly separable?
        - Nonlinear transformation
        - Soft-margin SVM; allows some errors
    - Why does number of SVs approximate VC dimension?
        - Intuition: number of parameters is approx. the VC dimension in many cases
        - We have as many alphas as data points
            - However, MOST alphas are 0
            - Effective number of parameters is approx. number of nonzero parameters
            - Therefore, we approximate VC dimension as number of SVs
    - Is there any advantage of other norms for margins?

        - Easiest to use Euclidean space for margin; if others are more practical, then use those
    - No clear way to further prune number of SVs
    - Noisy data sets will ruin SVMs as much as other models
        - Can just use soft-margin SVM to mitigate this

Lecture 15: Kernel Methods

- Review
  - The margin; maximizing the margin is a dual problem
    - More than 1 line works to separate the data, get one with largest margin to have an advantage in generalization
  - Quadratic programming to solve Lagrangian
  - Support vectors have Lagrange alpha$_n$ > 0
    - E[E$_{out}$] <= (E[# of SVs])/(N-1)
    - # of SVs is an in-sample quantity which can be used to check on out-of-sample error!
  - Nonlinear transform
    - Complex h, but simple H
    - Allow very sophisticated models without paying full price
- The kernel trick
  - What do we need from the Z space?
    - Want to get inner products of z vectors without ever actually knowing Z space
    - Only need inner products for both w and b
  - Given two points x and x' in X, we need z^{T}z'
    - Let z^{T}z' = K(x, x') (the kernel) "inner product" of x and x'
    - Ex. x = (x1, x2) → 2nd order transform
      - z = phi(x) = (1, x1, x2, x1^2, x2^2, x1x2)
      - K(x, x') = z^{T}z' = 1 + x1x1' + x2x2' + x1^{2}x1'^{2} + x2^{2}x2'^{2} + x1x1'x2x2'
  - The trick: can we compute K(x, x') without transforming x and x'?
    - Ex. Consider K(x, x') = (1+x^{T}x')^2 = (1 + x1x1' + x2x2')^2
      - = 1 + 2*x1x1' + 2*x2x2' + x1^{2}x1'^{2} + x2^{2}x2'^{2} + 2*x1x1'x2x2'
      - This is an inner product!
        - (1, x1^2, x2^2, sqrt(2)x1, sqrt(2)x2, sqrt(2)x1x2)
  - Polynomial kernel
    - X = R^d and phi: X -> Z is polynomial of order Q
    - The "equivalent" kernel K(x, x') = (1+x^{T}x')^Q
      - (1+x1x1'+x2x2'+…+xdxd')^Q
      - Can adjust scale:
        - K(x, x') = (ax^{T}x' + b)^Q
  - We only need Z to exist!
    - If K(x, x') is an inner product in some space Z, we are good
    - Ex. K(x, x') = exp(-gamma*|x-x'|^2)
      - Infinite-dimensional Z: take simple case
        - K(x, x') = exp(-(x-x')^2)
        - Taylor expansion
          - exp(-x^2)exp(-x'^{2})exp(2xx')
          - $$= \exp\left(-x^2\right)\exp\left(-x'^2\right)\underbrace{\sum_{k=0}^{\infty}\frac{2^k(x)^k(x')^k}{k!}}_{\exp(2xx')}$$

                    - Divide into some inner product on x and x'
    - Kernel method is very easy to compute
        - Overkill to go to infinite space?
            - Count the number of SVs
    - Kernel formulation of SVM
        - $$\underbrace{\begin{bmatrix} y_1y_1\,\mathbf{x}_1^{\mathsf{T}}\mathbf{x}_1 & y_1y_2\,\mathbf{x}_1^{\mathsf{T}}\mathbf{x}_2 & \dots & y_1y_N\,\mathbf{x}_1^{\mathsf{T}}\mathbf{x}_N \\ y_2y_1\,\mathbf{x}_2^{\mathsf{T}}\mathbf{x}_1 & y_2y_2\,\mathbf{x}_2^{\mathsf{T}}\mathbf{x}_2 & \dots & y_2y_N\,\mathbf{x}_2^{\mathsf{T}}\mathbf{x}_N \\ \dots & \dots & \dots & \dots \\ y_Ny_1\,\mathbf{x}_N^{\mathsf{T}}\mathbf{x}_1 & y_Ny_2\,\mathbf{x}_N^{\mathsf{T}}\mathbf{x}_2 & \dots & y_Ny_N\,\mathbf{x}_N^{\mathsf{T}}\mathbf{x}_N \end{bmatrix}}_{\text{quadratic coefficients}}$$
        - $$\underbrace{\begin{bmatrix} y_1y_1\,K(\mathbf{x}_1,\mathbf{x}_1) & y_1y_2\,K(\mathbf{x}_1,\mathbf{x}_2) & \dots & y_1y_N\,K(\mathbf{x}_1,\mathbf{x}_N) \\ y_2y_1\,K(\mathbf{x}_2,\mathbf{x}_1) & y_2y_2\,K(\mathbf{x}_2,\mathbf{x}_2) & \dots & y_2y_N\,K(\mathbf{x}_2,\mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ y_Ny_1\,K(\mathbf{x}_N,\mathbf{x}_1) & y_Ny_2\,K(\mathbf{x}_N,\mathbf{x}_2) & \dots & y_Ny_N\,K(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}}_{\text{quadratic coefficients}}$$
        - The final hypothesis
            - Express g(x) = sign(w^Tz+b) in terms of K(-, -)
            - $$\mathbf{w} = \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n \mathbf{z}_n \quad \Longrightarrow \quad g(\mathbf{x}) = \operatorname{sign}\left(\sum_{\alpha_n>0} \alpha_n y_n K(\mathbf{x}_n,\mathbf{x}) + b\right)$$
            - $$\text{where } \; b = y_m - \sum_{\alpha_n>0} \alpha_n y_n K(\mathbf{x}_n,\mathbf{x}_m)$$
        - How do we know that Z exists for a given K(x, x')?
            - Three approaches
                - 1. By construction
                - 2. Math properties (Mercer's condition)
                - 3. Who cares?
            - Design your own kernel (Mercer's condition)
                - K(x, x') is a valid kernel iff
                    - 1. It is symmetric
                    - 2. The matrix: {K(xi, xj) for i from 1 to N for j from 1 to N} is positive semi-definite (>= 0) for any x1,…,xN
- Soft-margin SVM
    - Two types of non-separable: slightly vs. seriously
        - Kernels deal with seriously non-separable
        - Soft-margin deals with slightly non-separable
    - Error measure
        - Margin violation: $y_n(w^Tx_n+b) >= 1$ fails
        - Quantify: $y_n(w^Tx_n+b) >= 1$ – slack
            - Slack >= 0
        - Total violation: sum of slacks
    - The new optimization

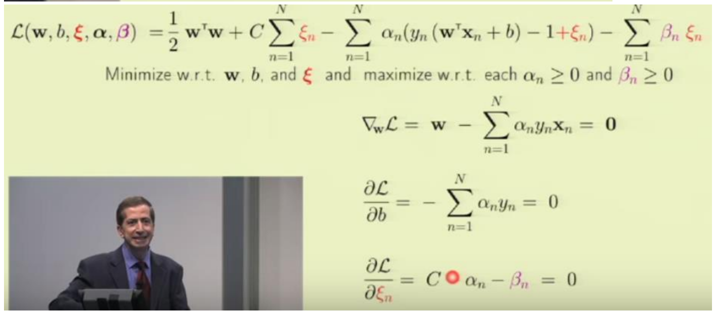
    - 

      Minimize $\frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{n=1}\xi_n$

      ubject to $y_n\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b\right) \geq 1 - \xi_n$ for $n = 1, \dots, N$

      and $\xi_n \geq 0$ for $n = 1, \dots, N$

      $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, $\boldsymbol{\xi} \in \mathbb{R}^N$

    - 

      $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n(y_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) - 1 + \xi_n) - \sum_{n=1}^{N}\beta_n\xi_n$

      Minimize w.r.t. $\mathbf{w}$, b, and $\boldsymbol{\xi}$ and maximize w.r.t. each $\alpha_n \geq 0$ and $\beta_n \geq 0$

      $$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n = \mathbf{0}$$

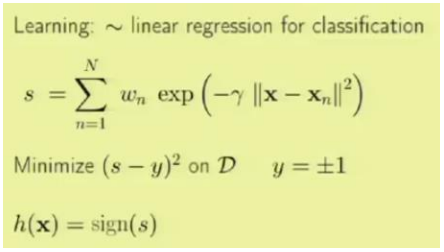
      $$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{n=1}^{N}\alpha_n y_n = 0$$

      $$\frac{\partial \mathcal{L}}{\partial \xi_n} = C - \alpha_n - \beta_n = 0$$

    - Everything cancels wow!
      - Except with added condition that alpha$_n$ <= C
  - Same results as before!!!
  - Types of support vectors
    - Margin support vectors (0 < alpha$_n$ < C)
    - Non-margin support vectors (alpha$_n$ = C)
  - Find C using cross-validation
- Technical observations
  - 1. Hard margin: What if data is not linearly separable?
    - “primal” → “dual” breaks down if not the case
    - Solution only works if data is linearly separable
      - Just try the SVM to see if it works
  - 2. Z: What if there is w0?
    - All goes to b and w0 → 0
    - Charged for size of w and not b
- Q&A
  - How to generalize SVM to regression cases?
    - Very technical
    - Major success of SVM is in classification, not regression
  - Safe but not certain to say to transforming to infinite dimensional plane creates linear separability
    - Never seen it happen with real dataset
  - If matrix is not positive semi-definite, quadratic programming tends to complain
    - Actually, complaints are very common
  - Is it possible to combine kernels?
    - As long as there is still a Z space this is fine
  - This analysis uses Euclidean norm
  - Scale of problems that can be solved by SVMs/Quad. Programming
    - Depends on MATLAB vs. other software

- 10,000 points pretty formidable, <1,000 should be okay
- Some packages have optimization/heuristic methods

Lecture 16: Radial Basis Functions (09/05/18)

- Review
  - Kernel methods: generalization of basic SVM to accommodate possibly infinite Z spaces
    - K(x, x') = $z^T z'$ for some Z space
    - K(x, x') = exp(-gamma*|x-x'|^2)
      - RBF Kernel
  - Soft-margin SVM
    - Minimize ½*$w^T w$ + C*sum{n=1 to N}[xi$_n$]
    - Same as hard margin except with 0 <= alpha$_n$ <= C
      - Large C means less tolerant of violations
      - Small C means more tolerant of violations
    - Pick C using cross-validation
- General RBF
  - Each (x_n, y_n) in D influences h(x) based on |x-x_n|
  - Standard form:
    - h(x) = sum{n=1 to N}[$w_n$exp(-gamma*|x-x_n|^2)]
  - Radial because of the distance |x-x_n|
  - Basis function because of exp function above
  - Learning algorithm
    - Finding w_1,…,w_n from above standard form that minimize some sort of error
      - Based on D = (x_1,y_1),…,(x_N,y_N)
      - E_in = 0: We have n data points and n parameters → should be easy to get parameters that result in 0 error
    - h(x_n) = y_n for n = 1,…,N:
      - sum{m=1 to N}[w_m*exp(-gamma*|x_n-x_m|^2)] = y_n
    - N equations, N unknowns
      - Exact interpolation
    - Now consider gamma
      - Gamma is small → Gaussian is wide
      - Gamma is large → Gaussian is narrow
        - Interpolation is very poor; multiple peaks from sum of contributions
      - Revisit in end of lecture
  - RBF for classification
    - Take sign of the h function defined previously
    - 

      Learning: ~ linear regression for classification

      $$s = \sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)$$

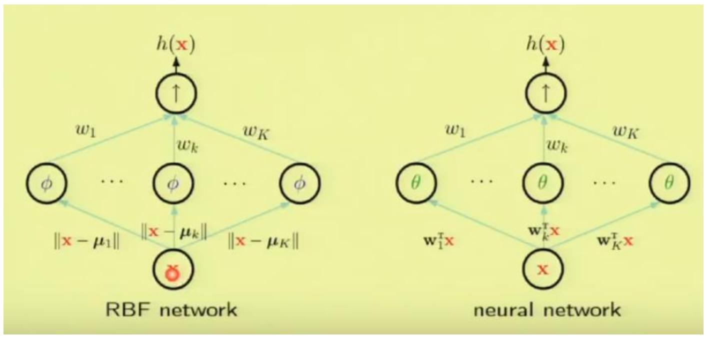
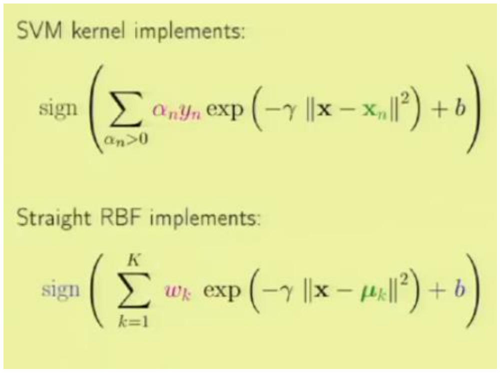
      Minimize $(s-y)^2$ on $\mathcal{D}$ $\quad y = \pm 1$

      $h(\mathbf{x}) = \text{sign}(s)$
  - Modify exact interpolation model
    - N parameters w_0,…,w_N based on N data points

- Hopeless to generalize?
- Instead use K << N centers: mu$_1$,…,mu$_K$ instead of x$_1$,…,x$_N$
- h(x) = sum{k=1 to K}w$_k$*exp(-gamma*|x-mu$_k$|^2)
- How to choose centers mu$_k$?
    - Minimize the distance between x$_n$ and the closest center mu$_k$
    - K-means clustering
        - Split x$_1$,…,x$_N$ into clusters S$_1$,…,S$_K$
        - Minimize sum{k=1 to K}[sum{x$_n$ in S$_k$}[|x$_n$-mu$_k$|^2]]
            - Unsupervised learning!
            - NP-hard!!
        - An iterative algorithm: Lloyd's algorithm
            - Iteratively minimize the sum quantity from before w.r.t. mu$_k$, S$_k$
            - Fixes S$_k$ and tries to optimize mu$_k$
                - mu$_k$ <- 1/|S$_k$|*sum{x$_n$ in S$_k$}x$_n$
            - Fixes mu$_k$ and tries to optimize S$_k$
                - S$_k$ <- {x$_n$: |x$_n$-mu$_k$| <= all |x$_n$-mu$_l$|}
            - Very fast convergence and very good results
                - Convergence → There are a finite number of possible values for the centers since there are a finite number of subsets of points
                - Convergence to local minimum
    - K-means vs. SVM 9 points
        - 2 solutions with RBF kernels that produce very different results
- How to choose the weights w$_k$?
    - N equations in K < N unknowns
    - Instead of equal, we get approx.
    - $$\sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2\right) \approx y_n \qquad N \text{ equations in } K < N \text{ unknowns}$$
      $$\underbrace{\begin{bmatrix} \exp(-\gamma\|\mathbf{x}_1-\boldsymbol{\mu}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_1-\boldsymbol{\mu}_K\|^2) \\ \exp(-\gamma\|\mathbf{x}_2-\boldsymbol{\mu}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_2-\boldsymbol{\mu}_K\|^2) \\ \vdots & \vdots & \vdots \\ \exp(-\gamma\|\mathbf{x}_N-\boldsymbol{\mu}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_N-\boldsymbol{\mu}_K\|^2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}}_{\mathbf{w}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{y}}$$
    - If $\Phi^{\mathsf{T}}\Phi$ is invertible, $\boxed{\mathbf{w} = (\Phi^{\mathsf{T}}\Phi)^{-1}\Phi^{\mathsf{T}}\mathbf{y}}$
- Choosing gamma
    - Treat gamma as a parameter to be learned
    - Could use gradient descent?
    - Iterative approach: Expectation Maximization (EM) algorithm
        - 1. Fix gamma. Solve for w$_1$,…,w$_K$
        - 2. Fix w$_1$,…,w$_K$. Minimize error w.r.t. gamma
    - Can use different gamma$_k$ for each center mu$_k$

- RBF and nearest-neighbors

- Adopt the y value of a nearby point
- Similar effect by basis function (cylinder)
- To make less abrupt, take k-NNs
- For basis function, use a gaussian instead of cylinder
- Similarity based method

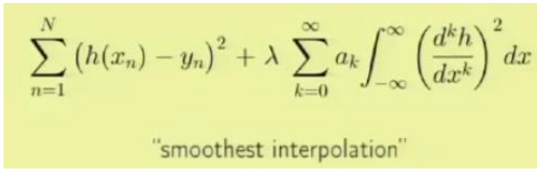
- RBF and neural networks
  - 
$h(\mathbf{x})$
w_1
w_k
w_K
ϕ
$\|\mathbf{x}-\boldsymbol{\mu}_1\|$
$\|\mathbf{x}-\boldsymbol{\mu}_k\|$
$\|\mathbf{x}-\boldsymbol{\mu}_K\|$
$\mathbf{x}$
RBF network
$h(\mathbf{x})$
w_1
w_k
w_K
θ
$\mathbf{w}_1^\mathsf{T}\mathbf{x}$
$\mathbf{w}_k^\mathsf{T}\mathbf{x}$
$\mathbf{w}_K^\mathsf{T}\mathbf{x}$
$\mathbf{x}$
neural network
  - For RBF, if |x-mu| is huge, we know the associated feature will have 0 contribution
  - For NNs, all features have representation
  - RBFs only worry about local parts of space
  - NNs use learned features, which is why result is non-linear
  - RBFs do not, which is why result is almost linear
  - Both 2-layer networks
  - Can implement NNs with SVMs
- RBF and kernel methods
  - RBF vs. its SVM kernel
    - RBF: unsupervised learning →pseudo-inverse → linear regression
    - SVM: maximize margin → equate with kernel → quad. Programming
  - 

SVM kernel implements:

$$\text{sign}\left(\sum_{\alpha_n>0} \alpha_n y_n \exp\left(-\gamma \|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

Straight RBF implements:

$$\text{sign}\left(\sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x}-\boldsymbol{\mu}_k\|^2\right) + b\right)$$

- RBF and regularization
  - Can derive RBFs entirely based on regularization
  - Add smoothness constraint to standard error
  - 

$$\sum_{n=1}^{N} \left(h(x_n) - y_n\right)^2 + \lambda \sum_{k=0}^{\infty} a_k \int_{-\infty}^{\infty} \left(\frac{d^k h}{dx^k}\right)^2 dx$$

"smoothest interpolation"

- Q&A
  - How does SVM simulate 2-level NN?

        - Sigmoid of $w^T x$ could be a valid kernel
    - How to choose number of centers?
        - No conclusive answer
        - If adding another center dramatically lowers error, it was probably worth it
    - Do RBFs work in high dimensions?
        - Yes; difficulties are not specific to RBFs though
    - How to choose gamma?
        - Iterative method where we fix one of w_n or gamma_n and solve/minimize error w.r.t. the other
    - RBF motivation
        - Assume smoothness of data
        - Assume noise in value x; cannot measure input exactly
            - Arrive at same result with slightly different inputs
    - Can RBFs be better than SVMs?
        - Perhaps unsupervised learning can let you get more ahead
        - SVMs more common

Lecture 17: Three Learning Principles (09/05/18)

- Review
  - Radial basis functions
    - Choose mu_k's: Lloyd's algorithm
    - Choose w_k's: Pseudo-inverse
  - Related to NNs, neural nets, SVMs, regularization, and unsupervised learning
    - NNs: influences regions around nearest points
      - RBFs are a soft version
    - Neural nets: Think of RBFs as activation in hidden layer
      - RBF takes care of 1 region at a time
    - SVM: Kernel = RBF
      - RBF clustered using unsupervised learning
      - SVM use support vectors which are from supervised learning
    - Regularization: RBFs derived from augmented error which account for smoothness of model using function of derivatives
    - Unsupervised learning: used to cluster points to find centers
- Occam's Razor
  - Recurring theme that simpler hypotheses are better
  - "Quote" by Einstein: An explanation of the data should be made as simple as possible, but no simpler
  - The razor: symbolic of a principle set by William of Occam
  - Statement: **The simplest model that fits the data is also the most plausible**
  - Two questions:
    - 1. What does it mean for a model to be simple?
    - 2. How do we know that simpler is better?
  - What does simple mean?
    - Measure of complexity - two types: complexity of h and complexity of H
      - Complexity of h: MDL (min. description length), order of polynomial
      - Complexity of H: Entropy, VC dimension
    - When we think of simple, it's in terms of h
    - Proofs use simple in terms of H
    - Link between complexity of h and H is counting
      - l bits specify h $\rightarrow$ h is one of 2^l elements in a set H
      - Real-valued parameters?
        - Ex. 17th order polynomial – complex and one of "many"
      - Exceptions? Looks complex but is one of few.
        - SVM
  - Puzzle 1: Football oracle
    - Letter predicting game outcome: ends up being correct
    - More letters: still correct for 5 more weeks
    - Letter says "want more?" $50 charge
    - Should you pay?
      - NO. He is actually sending letter to 32 different people!

        - Assured that at least 1 person will get correct results!
  - Why is simpler better?
    - Better does not mean more elegant! It means better out-of-sample performance
    - Basic argument: (formal proof under diff. idealized conditions)
      - Fewer simple hypotheses than complex ones $m_H(N)$
      - → less likely to fit a given data set $m_H(N)/2^N$
      - → more significant when it happens
      - The postal scam: $m_H(N) = 1$ vs. 2^N
  - A fit that means nothing
    - Conductivity linear in temperature?
      - A linear with 2 pts
      - B linear with 3 pts
      - B provides more evidence than A
      - A provides no evidence because any two points can be fitted with a line
        - Linear too complex for size of data set in A
- Sampling Bias
  - Puzzle 2: Presidential election
    - In 1948, Truman ran against Dewey in close elections
    - A newspaper ran a phone poll of how people voted
    - Dewey won the poll decisively – newspaper declared the result
    - Truman won!
    - It's not delta's fault: $P[|E_{in}-E_{out}| > eps] <= deltas$
    - The bias:
      - In 1948, phones were expensive
      - Richer people favored Dewey more than general population
  - **If the data is sampled in a biased way, learning will produce a similarly biased outcome**
  - Ex. Normal period in market, testing live trading in real market
  - Sampling bias makes you vulnerable
  - Matching the distributions
    - Methods to match training and testing distributions
    - Weigh points in training such that it matches the testing set
    - Doesn't work if region has P=0 in training but P>0 in testing
  - Puzzle 3: Credit approval
    - Historical records of customers
    - Input: information on credit application
    - Target: profitable for bank?
    - Bias? Using data of customers that were approved!
    - Sampling bias not too terrible
      - Probably have all the "support vectors" in data set already
- Data Snooping
  - Principle: **If a data set has affected any step in the learning process, its ability to assess the outcome has been compromised.**

- Most common trap for practitioners
  - Seems like better performance when we do this
  - Many ways to slip
- Looking at the data
  - Remember nonlinear transform?
    - z = (1, x1, x2, x1x2, $x1^2x2^2$)
    - Let's try just z = (1, $x1^2$, $x2^2$) or even z = (1, $x1^2+x2^2$)
      - Seems better from looking at data
      - You are acting as a learning algorithm but not charging for it
  - Snooping involves D, not other information
- Puzzle 4: Financial forecasting
  - Predict US Dollar vs. British Pound
  - 8 years of points
  - Find change in rates for 20 days before and use to predict today's change in rate
  - Normalize data, split randomly: D_{train}, D_{test}
  - Train on D_{train} only, test g on D_{test}
  - Model failed!
  - Data is still being snooped!
    - When we normalize! We normalized before splitting; this means that we have 'seen' the test set already
    - Should split first and apply same normalization in train as test set
- Reuse of a data set
  - Trying one model after the other on the same data set, you will eventually 'succeed'
  - If you torture the data long enough, it will confess
  - VC dimension of the **total** learning model
    - May include what **others** tried!!!
  - Key problem: matching a particular data set too well
- Two remedies
  - 1. Avoid data snooping
    - Strict discipline
  - 2. Account for data snooping
    - How much data contamination?
- Puzzle 5: Bias via snooping
  - Testing long-term performance of "buy and hold" in stocks
  - Use 50 years' worth of data
  - All currently traded companies in S&P500
  - Assume you strictly followed buy and hold
  - Would have made great profit!
  - Looking at currently traded stocks: excludes the stocks that have crashed
    - Sampling bias caused by 'snooping'

- Q&A
  - Input data processing is important, but not covered in lectures

- Using failure of previous models to choose new model and not accounting for VC dimension increase is data snooping
- How to deal with sampling bias?
  - If we know the distributions, give importance to data points to get a more similar distribution
- Any counterexamples to Occam’s Razor?
  - Occam’s Razor holds in practical cases

Lecture 18: Epilogue (09/06/18)

- Review
    - Occam's Razor
        - Complexity of h <-> complexity of H
        - Unlikely event <-> significant if it happens
    - Sampling bias
        - Can compensate for sampling bias in some sense
    - Data snooping
        - Just using any data in test set makes it not a test set
- The map of machine learning
    - Theory, techniques, paradigms
    - Paradigms
        - Supervised
        - Unsupervised
        - Reinforcement
        - Active: query about the value actively instead of having a set dataset
        - Online: streamed dataset
    - Theory
        - VC
        - Bias-variance
        - Complexity: treats ML as a branch of asymptotic complexity
        - Bayesian: treats ML as a branch of probability
    - Techniques
        - Models
            - Linear
            - Neural networks
            - SVM
            - Nearest neighbors
            - RBF
            - Gaussian processes: similar to Bayesian
            - SVD: singular value decomposition
            - Graphical models: target is a joint probability distribution
        - Methods
            - Regularization
            - Validation
            - Aggregation
            - Input processing
- Bayesian learning
    - Full probabilistic approach to learning
    - P(D|h=f) decides which h (likelihood)
        - How about P(h=f|D)?
    - The prior
        - P(h=f|D) requires an additional probability distribution

        - P(h=f|D) = P(D|h=f)*P(h=f)/P(D) prop. P(D|h=f)*P(h=f)
        - P(h=f) is the prior
        - P(h=f|D) is the posterior
        - Given the prior, we have the full distribution
        - Prior is an assumption!!
    - Example of a prior
        - Consider a perceptron: h is determined by weights w
        - Possible prior on w: make as benign as possible
            - Each w_i is independent, uniform over [-1, 1]
        - Given D, we can compute P(D|h=f)
        - Putting them together, we get P(h=f|D)
    - A prior is an assumption
        - Even the most “neutral” prior
            - X is an unknown number between -1 and 1
            - Does this mean X is random?
                - Model using Uniform(-1, 1)
                - Not saying that we do not know X! We’re saying that it’s coming from a uniform distribution!!
                - Not the same problem
            - The true equivalent would be:
                - X is random and is the delta function centered at unknown a
    - If we knew the prior
        - We could compute the posterior P(h=f|D) for very h in H
        - → We can find the most profitable h given the data
        - → We can derive E(h(x)) for every x
        - → We can derive everything in a principled way
    - When you are following Bayesian learning ideology, it’s as if we first rob a bank, then we live righteously ever after
    - When is Bayesian learning justified?
        - The prior is valid; trumps all other methods
        - The prior is irrelevant; just a computational catalyst
            - Eventually, size of data set makes prior irrelevant
- Aggregation methods
    - Combining different solutions h_1, h_2, …, h_T that we trained on D
    - Combining multiple features to make more informed decision
    - Regression: take a (weighted) average
    - Classification: take a (weighted) vote
    - Also called ensemble learning and boosting
    - Different from 2-layer learning
        - In a 2-layer model, all units learn jointly
        - In aggregation, they learn independently
    - Two types of aggregation
        - 1. After the fact: combines existing solutions; “blending”
        - 2. Before the fact: creates solutions to be combined

        - Ex. Bagging – resampling D
- Decorrelation – boosting
    - Create $h_1,\ldots,h_t,\ldots$ sequentially: Make h_t decorrelated with previous h's
    - Emphasize points in D that were misclassified, choose weight of each hypothesis
    - Adaptive boosting is common example of this
- Blending – after the fact
    - For regression, $h_1, h_2, \ldots, h_T \rightarrow g(x) = \text{sum}\{t=1 \text{ to } T\}[\text{alpha}_t * h_t(x)]$
    - Principled choice of alpha values: minimize the error on an aggregation data set
        - Points that were not used by any of the hypotheses' training
        - Pseudo-inverse
        - Some alphas can come out negative
            - Could be that you are so correlated with other solutions that the noise from your solution is being removed
        - Which is best possible solution?
            - Take out each solution individually and see effect on performance
            - If there are identical solutions, then taking one out will have NO effect; penalizes common solutions