Contents

Machine Learning
CS 189

Contents

Machine Learning	**Fall 2016**

Classification: August 30

Table of Contents Local *Scribe: Brandon McKinzie*

- **Goal:** Want to prove that w is normal to decision boundary.
  - Starting axiom: Any vector x along the decision boundary satisfies, by definition,

$$w \cdot x + \beta = 0 \tag{1}$$

  - Let x and x' be two such vectors that lie on the decision boundary. Then the vector $x' - x$ points from x to x' and is parallel to the decision boundary. If w really is normal to the decision boundary line, then

$$\begin{aligned} w \cdot (x' - x) &= 0 \\ &= w \cdot x' - w \cdot x \\ &= (w \cdot x' + \beta) - (w \cdot x + \beta) \\ &= 0 + 0 \end{aligned} \tag{2}$$

  - Euclidean distance of x to decision boundary:

$$\tau = -\frac{(w \cdot x + \beta)}{||w||} = -\frac{f(x)}{||w||} \tag{3}$$

  - The **margin** is can be found as the minimum over all training data τ:

$$M = \min_{i \in 1 \cdots n} \frac{|f(x_i)|}{||w||} \tag{4}$$

Machine Learning **Fall 2016**

Gradient Descent: September 1

Table of Contents Local *Scribe: Brandon McKinzie*

- **Optimization:** maximize goals/minimize cost, subject to constraints. However, can model a lot while ignoring constraints.
- Main optimization algorithm is **stochastic gradient descent**.
- The **SVM**[1] is just another cost function. Want to minimize[2]

$$C \sum_{i=1}^{n} \Big(1 - y_i(w \cdot x_i + \beta)\Big)_+ + ||w||^2 \tag{5}$$

with respect to the **decision variables** (w, β); Note that C is a **hyperparameter**.

[1] so-called because you could represent the decision boundary as a set of vectors pointing to the hyperplane.

[2] Notation: $(z)_+ = max(z, 0)$.

Machine Learning **Fall 2016**

Stochastic Gradient Descent: September 6

Table of Contents Local *Scribe: Brandon McKinzie*

- Review: minimize cost function $f(w)$ over w. Take gradient; set to zero to solve for w.
- If can't solve analytically, then Gradient Descent:

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) \tag{6}$$

- For convex f, can always find solution. Guaranteed global minimum.
- Cost functions of form: minimize $\sum \text{loss}(w_i(x_i, y_i)) + \text{penalty }(w)$.
- SVM example:

$$\min C/n \sum (1 - y_i w^T x_i)_+ + ||w||^2$$

. Add squared norm because better margins and better classifications. Also, because algorithms converge faster. C is the **regularization parameter**. "Do I fit the data, or make w simple?". Doesn't change optimal set, just changes the "Cost" (wat).

- Want algorithm constant in number of data points n[3].
- Unbiased estimate of the gradient:
  - Want expected vlaue of g to be gradient of cost function.
  - Sample i uniformly at random from $\{1, \ldots, n\}$.
  - Then set g to gradient of loss at ith data point.
- SGD:
  - initialize w_0, $k = 0$.
  - (Repeat) sample i at uniform. Do weight update on the loss for i term. Until converged.
  - Follow the expected value of the gradient (rather than the true gradient) until converge. Following a noisy version. As long as variance is bounded, direction will be more or less correct. For large number of data points n, will be pretty good.

[3] Regular GD is linear in n

- Numerical example:
  - $f(w) = 1/2n \sum (w - y_i)^2$. Assumes x always 1.
  - Solve $\nabla f(w) = 0 = 1/n \sum (w - y_i) = 0$.
  - Optimal $w = 1/n \sum y_i$. The empirical mean.
  - Init $w_1 = 0$. Set $\alpha_k = 1/k$. Where k is kth update reference.
  - $w_2 = w_1 - \alpha_k \nabla \text{loss}\,() = y_1$. Where loss the grad of f.
  - $w_3 = w_2 - \alpha_2(w_2 - y_2) = y_1 - \frac{1}{2}(y_1 - y_2) = \frac{y_1+y_2}{2}$.
  - $w_4 = \cdots =$. idk
  - Lesson: order we passed through data didn't matter. One pass over all data points leads to optimal w. Why advocate randomness then? He uses sum of trig function example to illustrate how SGD can struggle if done in order, but converge much quicker when randomly sampled.
- Illustrates "**region of confusion**". Coined by Bertsekas. Different convex functions along w. Rapid decrease in error early on iterations means we are far outside this region. Constant α means you'll jiggle around later iterations. That is why you do diminishing α; helps in region of confusion.
- Most important rules of SGD: (buzzwords)
  - shuffle! Can speed up by as much as 20x.
  - diminishing stepsize (α learning rate decay). After n steps, set $\alpha = \beta \cdot \alpha$.
  - **momentum**. $w_{k+1} = w_k - \alpha \nabla l_i(w_k) + \beta_k(w_k - w_{k-1})$. Momentum is in final term. Typical value is 0.9.
- Notation: $e(z) = (z < 1)$. Evaluates to 1 or 0.

Machine Learning **Fall 2016**

Risk Minimization & Optimization Abstractions: September 8

Table of Contents Local *Scribe: Brandon McKinzie*

- Where do these optimization problems come from?
  - General framework: minimizing an average loss + λ penalty.
  - Loss: measures data fidelity.
  - Penalty: Controls model complexity.
  - Features/representation: How to write (x_i, y_i).
  - Algorithms: ∇ cost(w) = 0.
  - **Risk**: Integral of loss over probability(x, y).
  - Empirical risk: Sample average. Converges to true Risk with more points; variance decreases.
- Begins discussion of splitting up data.
  - Let some large portion be the **Training set** and the small remaining data points be the **Validation set**.

$$R_T = \frac{1}{n_T} \sum_{train} loss(w, (x_i, y_i)) \tag{7}$$

$$R_V = \frac{1}{n_V} \sum_{val} loss(w, (x_i, y_i)) \tag{8}$$

  - By law of large numbers, can say the R_V will go like $\frac{1}{n_V}$. Looking lots of times at validation set becomes a problem with $n_V \approx 10^4$.
- Classification example:
  - **Hinge** loss: $(1 - yw^T x)_+$. Means you're solving a SVM.
  - Least-Squares: $(1 - yw^T x)^2$. Bayes classifiers.
  - In practice, hinge and LS perform basically the same.
  - Logistic loss useful for MLE.
- Most important theorem in machine learning: Relates risk with empirical risk:

$$R[w] = \frac{R[w] - R_T[w]}{\text{generalization err}} + \frac{R_T[w]}{\text{train err}} \tag{9}$$

$$\approx R_V[w] - R_T[w] + R_T[w] \tag{10}$$

- One vs. all classifation MNIST example: Have a classifier for each digit that treats as (their digit) vs. (everything else). Choose classifier with highest margin when classifying digit.
- **Maximum Likelihood**:
  - Have $p(x, y; w)$. Pick the w that makes data set have highest probability.
  - Assumes data points come independently.
  - Can get same result by minimizing the negative log avg.
- More than most things (like loss functions) are choosing the **features**. **LIFT THE D**. Conic sections because why not.
- N-grams. **Bag of words**.
  - x_i = number occurrences of term i. Count number of times each word appears in some document.
  - The two-gram is the *lifted* version. x_{ij} = number occurrences of terms i, j in same context. Count number of terms two words, e.g. appear in the same sentence, or next to each other. Like a quadratic model.
- Histograms
  - $\hat{x}_{ij} = 1$ if $x_i \in bin(j)$ else 0.
  - e.g. histograms of image gradients in the notes.
- "If you have too many features, then you have to have a penalty." - D.J. Khaled. i.e. if $d > n$, must use pen(w). Never need to have $d > n$, becaues of **Kernel trick**.

Machine Learning **Fall 2016**

Decision Theory: September 13

Table of Contents Local *Scribe: Brandon McKinzie*

- **Decision Theory:**
  - Given feature distributions conditioned on class values (e.g. ±1).
  - Goes over Bayes rule like a pleb.
  - Classification depends on loss function.
- Loss function can be **asymmetric**. e.g. would rather misclassify as cancer — normal rather than misclassify normal — cancer. So to minimize expected loss, may prefer to be wrong more on some misclassification than another.
- Can minimize on probability of error.

$$\min\left[Pr(\text{error}) = \int Pr(\text{error}|x)Pr(x)dx\right] \tag{11}$$

  The area (under) of overlap between conditionals (think hw problem w/Gaussians) is Pr(error). If K classes, similarly, classify as the maximum conditional, and error is 1 - Pr$()_{max}$.
- **Modified rule**:

$$\min \sum_k L_{kj} P(c_k|x) \tag{12}$$

  where L_{kj} is loss where true class is k but classify as j. "by integrating over x, we can compute the expected loss."
- Loss function in regression pretty clear (e.g. least squared loss). Classification is less so. Can use the **"Doubt option"**. Classifier humility (lolololol)[4]. In some range of inputs near the decision boundary, just say "idk".
- Good classifiers have expected loss closer to Bayes risk, given certain choice of features.
- **NOTE:** Jitendra praises to the lord Gauss. Note 2: Jitendra makes another god joke.
- Three ways of building classifiers:
  - **Generative:** Model the class conditional distribution $P(\tilde{x}|c_k)$. Model priors $P(c_k)$. Use bayes duh. Want to understand distributions under which data were *generated*. Physicists can get away with this shit. They have "models" [5]

[4] Wanna see my posterior
[5] fucking magnets how do they work?

- **Discriminative:** "Fuck it" method. Model $P(c_k|\widetilde{x})$ directly.
- Find decision boundaries.

- Posterior for Gaussian class-conditional densities :
  - $P(x|c_1)$ and $P(x|c_2) \sim \mathcal{N}(\mu_i, \sigma^2)$.
  - Univariate gaussian example *like a bitch*.
  - Posterior probabilities $P(c_i|x)$ turn out to be logistic σ functions.

Machine Learning **Fall 2016**

Multivariate Gaussians and Random Vectors: September 15

Table of Contents Local *Scribe: Brandon McKinzie*

⇒ Recht's Decision notation: $P(x|H_0)$. **RECHT CAN'T GET THE MIC ON HERE WE GO**

⇒ Want to discuss case of x being non-scalar. ***Random Vectors.***

- Def: vector x with probability density $p(x) : \mathbb{R}^n \to \mathbb{R}$.
- Example density is the **multivariate gaussian**.
- Usually want to know $Pr(x \in A) = \int_A p(x) dx_1 \dots dx_n$, the prob that x lives in set A.
- Properties of random vectors: **mean and covariance**.

⇒ Let $f : \mathbb{R}^n \to \mathbb{R}^m$. Expected value of f:

$$\mathbb{E}[f(x)] = \int \int f(x)p(x)dx_1 \dots dx_n \tag{13}$$

⇒ **Covariance Matrix**. A matrix. $\Lambda = \Lambda^T$.

$$\Lambda = \mathbb{E}[(x-\mu)(x-\mu)^T] = \mathbb{E}[xx^T] - \mu_x \mu_x^T \tag{14}$$

$$var(x_1) = \mathbb{E}[(x_1 - \mu_{x_1})2] = \Lambda_{11} \tag{15}$$

⇒ Let $v \in \mathbb{R}^n$. Then $var(v^T x) = v^T \Lambda v \geq 0 \Rightarrow \Lambda_x$ is ***positive semidefinite.***

- Suppose A is some square matrix, λ is an eigval of A with corresponding eigvec x if $Ax = \lambda x$. Larger eigenvalues tell about how much variance in a given direction.
- Eigenvectors are, like, eigen*directions*, man.
- **Spectral Theorem**: If $A = A^T$, then $\exists S = v_1, \dots, v_n \in \mathbb{R}^n$ such that $v_i^T v_j = 0$, $i \neq j$, and $Av_i = \lambda_i v_i$ and $\lambda_i \in \mathbb{R}$.
- Because matrix of eigenvectors has vectors linearly independent, invertible.
- A p.d $\Rightarrow B^T AB$ is p.s.d. $\forall B$.
- If A is p.s.d., then $f(x) = x^T Ax$ is *convex.*

⇒ **Multivariate Gaussian**

$$p(x) = \frac{1}{det(2\pi\Lambda_x)^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_x)^T\Lambda_x^{-1}(x-\mu_x)\right] \tag{16}$$

If $\mu_x \in \mathbb{R}^n,\ \Lambda_x$ p.d. $\Rightarrow$ p(x) is a density.

⇒ What happens if covariance is diagonal? Then the the vars are indepenedent.

- Λ_x diagonal,
- x Gaussian
- $\Rightarrow x_1, \ldots, x_n$ independent.

Machine Learning	**Fall 2016**

Maximum Likelihood: September 20

Scribe: Brandon McKinzie

- **Estimation**: hypothesis testing on the continuum.
- **Maximum Likelihood:** Pick the model so that $p(\text{data}|\text{model})$, the likelihood function, is maximized.
- Treat model as random var. Then maximize p(model—data), "**maximum a posteriori**". Assume uniform priors over all models. Flaw is assuming model is a random variable.
- ML examples:
  - *Biased Coin.* Flipping a coid.

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} = P(x|p) \tag{17}$$

    See $n = 10, 8$ heads. Choose estimate $\hat{p} = \frac{4}{5}$. Binomial is not concave, when you take the log it becomes concave.

- Gaussians.

$$x1, \ldots, x_n \sim \mathcal{N}(\mu, \sigma^2) \tag{18}$$

  independent samples.

$$P(\{x_i\}|\mu, \sigma^2) = \prod P(x_i|\mu, \sigma^2) \tag{19}$$

  where each term in prod is standard Gaussian PDF. Next, take the log.

$$\log P(\{x_i\}|\mu, \sigma^2) = \sum -\frac{x_i - \mu}{2\sigma^2} - \log \sigma - 1/2 \log 2\pi \tag{20}$$

  Ideally, best estimates for mean and variance:

$$\hat{\mu} = \frac{1}{n} \sum_i x_i \tag{21}$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_i \left(x_i - \hat{\mu}\right)^2 \tag{22}$$

- Multivariate Gaussian:

$$P(x|\mu, \Lambda) = \frac{1}{\det 2\pi\Lambda^{1/2}} \exp(-\frac{1}{2}(x-\mu)^T \Lambda^{-1} (x-\mu) \tag{23}$$

Machine Learning **Fall 2016**

LDA & QDA: September 22

Table of Contents Local *Scribe: Brandon McKinzie*

- How do we build classifiers?
  - ERM. Minimize
  $$\frac{1}{n}\sum(\text{loss})+\lambda\text{pen} \tag{24}$$
  Equivalent to discriminative
  - Generative models. Fit model to data, use that model to classify. For each class C, fit $p(x|y=C)$. Estimate $p(y=C)$. To minimize $Pr(err)$, pick y that maximizes $P(y|x)$ via Bayes rule.
  - Discriminative. Fit $p(y|x)$. Function fitting. Fitting each data point. For cost function, want to maximize $\prod P(y_i|x_i) \equiv \max 1/n \sum \log p(y_i|x_i)$. *Equivalent to ERM.*
- Generative example: What is a good model of x given y. Fit blobs of data given their labels.
  - Let
  $$p(x|y=C)=\mathcal{N}(\mu_c,\Lambda_c) \tag{25}$$
  where $\hat{\mu}_c = 1/n_c \sum_{i\in I_C} X_i$, and $\Lambda_c = 1/n_c \sum(x_i-\hat{\mu}_c)(x_i-\hat{\mu}_c)^T$. Only have one index i because it is a dxd sum of matrices.
  - Decision rule:
  $$\arg\max_c -1/2(x-\hat{\mu}_c)^T\hat{\Lambda}^{-1}(x-\hat{\mu}_c)-1/2\log\det\hat{\Lambda}_c-\log\hat{\pi}_c \tag{26}$$
  where last two terms are apparently constant. First term is a quadratic. $Q_c(x)$ denotes the whole thing. Decision boundary is set $\{x : Q_{c=-1}(x)=Q_{c=1}(x)\}$.
  - Need to make sure $n >> d^2$ in order to avoid overfitting.
  - called **Quadratic Discriminant Analysis**.
- **Linear Discriminant Analysis**. Assume Λ_c same for every class. They all have the same covariance matrix.

- How to find Λ?

$$\hat{\Lambda}_c = \sum_C \frac{n_c}{n} \frac{1}{n} \sum_{i \in C} (x_i - \hat{\mu}_c)(x_i - \hat{\mu}_c)^T \tag{27}$$

$$= \frac{1}{n} \sum_i^n (x_i - \hat{\mu}_{y_i})(x_i - \hat{\mu}_{y_i})^T \tag{28}$$

- Extremely similar to QDA, have (sort of; don't rely on this)

$$\arg\max_c -(x - \hat{\mu}_c)^T \hat{\Lambda}^{-1} (x - \hat{\mu}_c) - 1/2 \log\det\hat{\Lambda} - \log\hat{\pi}_c \tag{29}$$

- end up with linear boundaries.

* Did something called **method of centroids**
* ppl like LDA bc fuck optimization

Machine Learning **Fall 2016**

Regression: September 27

Table of Contents Local *Scribe: Brandon McKinzie*

- Model $p(y|x) \sim \mathcal{N}(w^T x, \sigma^2)$, where $y = w^T x + \epsilon$. Epsilon is noise causing data points to fluctuate about hyperplane. Assume noise is gaussian with zero mean, some variance. Variance of y is the variance of the noise ϵ.
- Unknown we want to estimate: w. Estimate by using maximum likelihood. **Q:** says this maximizes $p(y|x)$. Figure out how this is same as maximizing $p(x|y)$...
- $P(data|\theta) = p(y_1, \ldots, y_n | x_1, \ldots, x_n, \theta) = \prod p(y_i | x_i, \theta)$.
- "Hit that bitch with a log so she split." - Jitendra
- Use matrices so you can express as

$$\sum (y_i - w^T x_i)^2 = ||y - Aw||^2 \tag{30}$$

  where A is typically denoted as the designed matrix X.
- Take gradient of loss like usual...

$$\nabla_W \mathcal{L} = -A^T y + A^T A w \tag{31}$$

  where, if we differentiate again, yields the hessian $H = A^T A$.
- Implicitly want $y \approx Aw$ here. Re-interpret A as a bunch of columns now (rather than a bunch of rows).

$$A = \begin{bmatrix} a_1 & \cdots & a_d \end{bmatrix} \tag{32}$$

  and so

$$||y - Aw||^2 = ||y - (w_1 a_1 + \cdots + w_d a_d)||^2 \tag{33}$$

- **column space** of A refers to this type of linear combination of columns a_i.
- References 3.2 figure in ESL. Error is vertical component of y in figure. This "error vector" is perpendicular to the subspace spanned by the x's.
- y - Aw is perpendicular to each and every column of A. $A^T(y - Aw) = \mathbf{0}$.

Machine Learning **Fall 2016**

Bias-Variance Tradeoff: October 4

 Scribe: Brandon McKinzie

- Fitting the data to model is called **bias**. Bias summarizes the fact that the model is wrong, but we want to know how wrong.
- **Variance** is robustness to changes in data.
- Good model has both low bias and low variance.
- Example:
  - Sample one point $x \sim \mathcal{N}(\mu, \sigma^2 I_d)$.
  - What is most likely estimate for $\hat{\mu}$? Just x (only have one point).
  - Since, the expected value of x is (by definition) μ, we have that

$$\mathbb{E}[\hat{\mu} - \mu] = 0 \tag{34}$$

  - How about squared error

$$\begin{aligned} \mathbb{E}[||\hat{\mu} - \mu||^2] &= \mathbb{E}[(\hat{\mu} - \mu)^(\hat{\mu} - \mu)] & (35) \\ &= \mathbb{E}[Tr(x - \mu)(x - \mu)^T] & (36) \\ &= Tr(\Lambda) & (37) \end{aligned}$$

  which uses the **cyclic property of the trace**: if dot product is scalar, then it is equal to trace of outer product[6].
  - What is the trace of the covariance matrix Λ? Here (only) it is $d\sigma^2$.
  - What if I'm bored and I define $\hat{\mu} = \alpha x$, where $0 < \alpha < 1$? Then

$$\begin{aligned} \mathbb{E}[\hat{\mu}] &= \alpha\mu & (38) \\ \mathbb{E}[\hat{\mu} - \mu] &= (\alpha - 1)\mu & (39) \end{aligned}$$

  which isn't zero (woaAHhhh!)
  - Variance won't go down.

$$\mathbb{E}[||\hat{\mu} - \mu||^2] = \mathbb{E}[||\hat{\mu} - \mathbb{E}[\hat{\mu}] + \mathbb{E}[\hat{\mu} - \mu||^2]] \tag{40}$$

[6] Oh, it is just the fact that Tr(AB) = Tr(BA). Moving on...

- Me making sense of **Newton's Method** (as defined in this lecture):
  - Slow for high dimensional probs; Better than gradient descent though.
  - Gradient descent models func with first order taylor approx. Newton's method uses *second order*.

$$f(x) \approx f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) \tag{41}$$

    where grad-squared is the **Hessian**.
  - ***Actual derivation by Wikipedia***:

$$\begin{align} \textsc{Let } f(\alpha) &= 0 \tag{42} \\ f(\alpha) &= f(x_n) + f'(x_n)(\alpha - x_n) + R_1 \tag{43} \\ \textsc{where } R_1 &= \frac{1}{2} f''(\xi_n)(\alpha - x_n)^2 \tag{44} \\ \frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) &= -\frac{f''(\xi_n)}{2f'(x_n)}(\alpha - x_n)^2 \tag{45} \\ & \tag{46} \end{align}$$

    and all the x_n represent the nth approximation of some root of $f(x)$.

- Oh I get it now:
  - **Gradient descent**: Find optimal w iteratively by assuming first-order taylor expansion of $\nabla J(w*)$:

$$\begin{align} \nabla J(w^*) &\approx \nabla J(w_k) \tag{47} \\ & \tag{48} \end{align}$$

    where w_k is the current best guess for the minimum of J. If this gradient is zero, we are done. If it is not, then we continue to iterate closer and closer via the update

$$w_{k+1} = w_k - \eta \nabla J(w_k) \tag{49}$$

    until our first-order approximation (appears) valid.
  - **Newton's method** goes a step further and expands to second order:

$$\begin{align} \nabla J(w*) &\approx \nabla J(w_k) + \nabla J(w_k)^2 (w^* - w_k) \tag{50} \\ &= \nabla J(w_k) + \mathbf{H}(w^* - w_k) \tag{51} \end{align}$$

where[7], implicit in all these optimization algorithms, is the hope that $w_{k+1} \approx w^*$, and so we can set this derivative to 0 to "solve" for $w_{k+1} = w^*$ as

$$(w_{k+1} - w_k) = -\mathbf{H}^{-1}\nabla J(w_k) \tag{52}$$

$$w_{k+1} = w_k - \mathbf{H}^{-1}\nabla J(w_k) \tag{53}$$

where equatoin 53 is **Newton's Update**. It is computationally better to compute e, where

$$\mathbf{H}e = -\nabla J(w_k) \longrightarrow e = -\mathbf{H}^{-1}\nabla J(w_k) \tag{54}$$

[7] Remember that we are dealing with matrices now, so keep the order of $\mathbf{H}$ before $(w - w_k)$ even if you don't like it.

Machine Learning **Fall 2016**

Regularization: October 6

Scribe: Brandon McKinzie

- bias = $\mathbb{E}[f(x) - y]$
- Risk = $\mathbb{E}[\text{loss}(f(x), y)]$
- Variance = $\mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]$
- Regularization: Minimize empirical loss + penalty term.

Machine Learning **Fall 2016**

Neural Networks: October 20

Table of Contents Local *Scribe: Brandon McKinzie*

Basics/Terminology. Outputs can be computed for, say, a basic neuron to output 2 as $S_2 = \sum_i w_{2i} x_i$. We can also feed this through **activations functions** g such as the logistic or RELU. Why shouldn't we connect linear layers to linear layers? *Because that is equivalent to one linear layer.* If we want to stack (multilayer) need some nonlinearity. Want to find good weights so that output can perform classification/regression.

Learning and Training. Goal: Find w such that O_i is as close as possible to y_i (the labeled/desired output). Approach:

→ Define loss function $\mathcal{L}(w)$.
→ Compute $\nabla_w \mathcal{L}$.
→ Update $w_{new} \leftarrow w_{old} - \eta \nabla_w \mathcal{L}$.

and so training is all about *computing the gradient*. Amounts to computing partial derivatives like $\frac{\partial \mathcal{L}}{\partial w_{jk}}$. Approach for **training a 2-layer neural network**:

→ Compute $\nabla_w \mathcal{L}$ for all weights from input to hidden, hidden output.
→ Use SGD. Loss function **no longer convex** so can only find local minima.
→ Naive gradient computation is quadratic in num. weights. **Backpropagation** is a trick to compute it in linear time.

Computing gradients [for a two layer net]. The value of the ith output neuron can be computed as

$$O_i = g\Big(\sum_j W_{ij}\ g\big(\sum_k W_{jk} x_k\big)\Big) \tag{55}$$

where let's focus on the weight W_{12}. *Simple idea:*

- Consider some situation where we have value for output O_i as well as another value O_i' which is the same as O_i except one of the weights is slightly changed:

$$O_i = g(\dots, w_{jk}, \dots, x) \tag{56}$$
$$O_i' = g(\dots, w_{jk} + \Delta w_{jk}, \dots, x) \tag{57}$$

- Then we can compute numerical approx to derivative for <u>one</u> of the weights:

$$\frac{\mathcal{L}(O_i') - \mathcal{L}(O_i)}{\Delta w_{jk}} \tag{58}$$

a process typically called the **forward pass**[8] This is $\mathcal{O}(n)$ if there are n weights in the network. But since this is just the derivative for <u>one</u> of the weights, the total cost over all weights is $\mathcal{O}(n^2)$.

- This is why we need backprop: to lower complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Backpropagation. Big picture: A lot of these computations [gradients] seem to be shared. Want to find some way of avoiding computing quantities more than once.

→ **Idea:** Want to compute some quantity δ^i at output layer for each of the i output neurons. Then, find the δ^{i-1} for the layer below, repeat until reach *back* to input layer [hence name backprop]. Key idea is the **chain rule**.

→ Notation:[9]

$$x_j^{(l)} = g\left(\sum_i w_{ij}^{(l)} x_i^{(l-1)}\right) \equiv g\left(S_j^{(l)}\right)$$

where now w_{ij} is *from i to j*. **We will also denote e for 'error'.**

→ Define partial derivative of error with respect to the linear combination input to neuron j as

$$\delta_j^{(l)} \triangleq \frac{\partial e}{\partial S_j^{(l)}} \tag{59}$$

which carry the information we want about the partial derivatives along the way.

→ Consider simple case of

$$x_i^{(l-1)} \to w_{ij}^{(l)} \to x_j^{(l)}$$

and we want to calculate

$$\frac{\partial e}{\partial w_{ij}^{(l)}} = \frac{\partial e}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{60}$$

$$= \delta_j^{(l)} \frac{\partial S_j^{(l)}}{\partial w_{ij}^{(l)}} \tag{61}$$

[8] Not sure why he says this. See pg 396 of ESL. Forward Pass: the current weights are fixed and the predicted values $\hat{f}_k(x_i)$.

[9] Note: he really screws this up.

→ "Inductive step" for calculating δ with chain rule [for **regression problem using squared error loss of** $e = \frac{1}{2}\Big(g(S_i^{(l)}) - y\Big)^2$ **corresponds to a given example**]:

$$\delta_i^{(l)} = \frac{\partial e}{\partial S_i^{(l)}} \tag{62}$$

$$= \frac{1}{2}\left[2\,\left(g(S_i^{(l)}) - y\right)\, g'(S_i^{(l)})\right] \tag{63}$$

Don't confuse the above expression for sigmoid deriv. It is not assuming anything about the functional form of g.

Machine Learning **Fall 2016**

Neural Networks II: October 25

Table of Contents Local *Scribe: Brandon McKinzie*

Backprop Review. Cross-entropy loss derivation uses the following. Note that we are defining all $y_i = 0$ or $y_i = 1$.

$$O_i^{y_i}(1-O_i)^{1-y_i} \quad \rightarrow \quad y_i \ln O_i + (1-y_i)\ln(1-O_i) \tag{64}$$

where the expression on the RHS is the log (likelihood) of the LHS. We want to take partial derivatives of loss function with respect to weights. The δ terms represent layer-specific derivatives of error with respect to the S values (the summed input). See previous lecture note for more details on this. Note that, in order to get the values of the error in the first place, need to first perform the **forward pass**.

Clarifying the notation. In the last lecture, we barely scratched the surface of actually calculating $\delta_i^{(l-1)}$, the partial of the error with respect to $S_i^{(l-1)}$. Recall that the subscript on $S_i^{(l-1)}$ means the weighted sum *into* the ith neuron at layer. Specifically

$$S_j^{(l-1)} = \sum_i w_{ij}^{(l)} x_i^{(l-2)} \quad \rightarrow \quad x_j^{(l-1)} \tag{65}$$

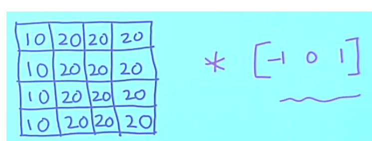
Calculating the δ terms. **Setup**: Only consider the following portion of the network: A summation value $S_i^{(l-1)}$ is fed into a single neuron $x_i^{(l-1)}$ at the $l-1$ layer. From this neuron, $g(S_i^{(l-1)})$ is fed to the neurons at the layer above (l) by connection weights w. We calculate the partial derivative of the error *corresponding to these particular weights* with respect to the summation fed to $x_i^{(l-1)}$ as

$$\begin{aligned}
\delta_i^{(l-1)} &= \frac{\partial err(w)}{\partial S_i^{(l-1)}} &(66)\\
&= \sum_j \frac{\partial err(w)}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial x_i^{(l-1)}} \frac{\partial x_i^{(l-1)}}{\partial S_i^{(l-1)}} &(67)\\
&= \sum_j \delta_j^{(l)} w_{ij}^{(l)} g'(S_i^{(l-1)}) &(68)
\end{aligned}$$

where we've already calculated all δ value in the layers above (i.e. we are somewhere along the backward pass).

[Inspiration for] Convolutional Neural Networks. [1 hr into lec]. Reviews biology of brain/neuron/eye. Rods and cones are the eye's pixels. Think of as 2D sheet of inputs. Such sheets can be thought of as 1D layers. Bipolar cell gets direct input (center input) from two photo-receptors, and gets indirect input (surround input) from horizontal cell. "Disc where you're getting indirect input from horizontal cell." Weights between neurons can be positive (excitatory) or negative (inhibitory). Assume center input is excitatory, surround input in inhibitory.

- Very small spot of light means neuron fires, as you increase size of spot, inhibition from surround cells kick in, and its output is diminished. Uses example of ON/OFF cells in retinal ganglia. Neurons can only individually communicate positive values, but multiple neurons can "encode" negative values.
- **Receptive Fields**. The receptive field of a receptor is simply the area of the visual field from which light strikes that receptor. For any other cell in the visual system, the receptive field is determined by which receptors connect to the cell in question.
- Relation to **Convolution**. Consider convolving an image with a filter.



Each output unit gets the weighted sum of image pixels. The [-1, 0, 1] is a "weighting mask."

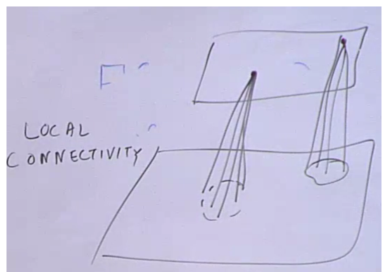
Machine Learning **Fall 2016**

Convolutional Neural Networks: October 27

Table of Contents Local *Scribe: Brandon McKinzie*

[started at 11:09]

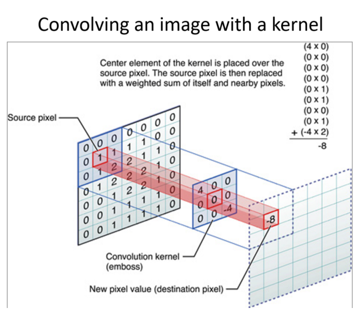
Review. Neurons in input layer arranged in rectangular array, as well as the neurons in next layer. There is **local connectivity** of neurons between layers (not fully connected).



Also recall that the receptive field of retinal ganglion cells can be modeled as a **Difference of Gaussians**:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{r^2}{2\sigma^2}} \tag{69}$$

Convolution. The 3x3 convolution kernel[10] in example below is like the w_{ij}. It does weighted combinations of input squares to map to the output squares. You keep the *same* w_{ij} kernel, which is related to **shift invariance**. Relation to brain: think of the difference of Gaussians in retinal ganglion as the mask (the w_{ij}).



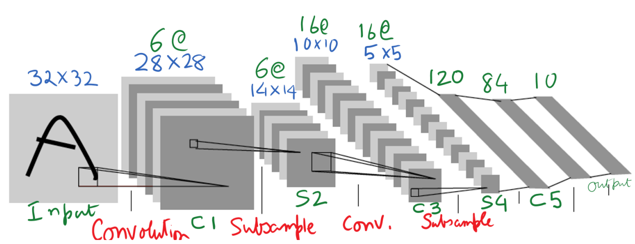
[10]Interchangeable terminology: conv kernel/filter/mask

- This makes the training problem easier, since it greatly reduces the number of parameters (the w_{ij}).
- Now, we increase complexity by increasing *depth* (more layers) rather than increasing the different parameters like fully-connected layers do.
- Long discussion of biological relevance. **skip to 50:00**.

Convolutional Neural Networks. Consider Input-¿middle-¿output rectangular layers. Neurons A and B in middle layer "like" specific orientations of the inputs. Have neuron C that does $\max(A, B)$. [General case: Have some group of neurons in middle layer that use the same mask (by definition) and some neuron in the next layer that takes a max of these neurons in its local neighborhood.]

- This has a slight **shift-invariance**, accomplished by the max operation. **[56:00]** This is called **max pooling**.
- Max pooling allows output layer to have less neurons than previous layers. This is related to **subsampling**.
- Note that when we stack layers, make sure they introduce nonlinearities.

Introduce Yann Lecun (1989). Here we go over his architecture.



- 32x32 grid of pixels (digits).
- Convolution with 5x5 filters (masks) $\rightarrow$ 28x28 grid. Six such masks. Each masks give 5^2 parameters + 1 for bias.
- Stopped at **[1 hour]**.

Shewchuck Notes

Contents

Shewchuck Notes **Fall 2016**

Perceptron Learning:

Table of Contents Local *Scribe: Brandon McKinzie*

Perceptron Algorithm

- Consider n sample points $X_1, \ldots, X_n$.
- For each sample point, let

$$y_i = \begin{cases} 1 & X_i \in \text{class C} \\ -1 & X_i \notin \text{class C} \end{cases}$$

- **Goal**: Find weights w that satisfy the constraint

$$y_i X_i \cdot w \geq 0 \tag{70}$$

- In order to minimize the number of constraint violations, need a way to quantify how "good" we are doing. Do this with the **loss function**

$$L(z,\ y_i) = \begin{cases} 0 & y_i z \geq 0 \\ -y_i z & \text{otherwise} \end{cases} \tag{71}$$

  Notice that this can only be ≥ 0 by definition. The larger L is, the worse you are as a human being.

- The **Risk/Objective/Cost** function is a sum total of your losses.

$$R(w) = \sum_{i=1}^{n} L(X_i \cdot w, y_i) = \sum_{i \in V} (-y_i\ X_i \cdot w) \tag{72}$$

  where $(\forall i \in V)(y_i X_i \cdot w < 0)$.

- **Goal**: Find w that minimizes $R(w)$.

Hyperplanes with Perceptron

- Notice the different between the two (purple) Goals stated in the previous subsection. We went from constraining w to certain **hyperplanes** in x-space ($y_i X_i \cdot w \geq 0$) to constraining w to certain **points** in w-space ($\min_w R(w)$).

- Figure **??** illustrates how the data points constrain the possible values for w in our optimization problem. For each sample point x, the constraints can be stated as
  - x in the "positive" class $\Rightarrow$ x and w must be on the **same** side of the hyperplane that x transforms into[11].
  - x in the "negative" class $\Rightarrow$ x and w must be on the **opposite** side of x's hyperplane.

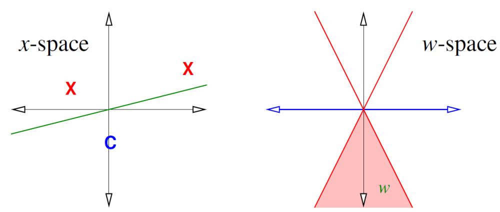


Figure 1: Illustration of how three sample points in x-space (left) can constrain the possible values for w in w space (right).

Algorithm: Gradient Descent

- GD on our risk function R is an example of an **optimization algorithm**. We want to *minimize* our risk, so we take successive steps in the *opposite* direction of $\nabla R(w)$.[12]

$$\nabla R(w) = \nabla \sum_{i \in V} (-y_i \; X_i \cdot w) \tag{73}$$

$$= -\sum_{i \in V} (y_i \; X_i) \tag{74}$$

- **Algorithm:**
  - $w \leftarrow$ arbitrary nonzero (e.g. any $y_i X_i$)
  - while $R(w) > 0$
    - $V \leftarrow$ all i for which $y_i X_i \cdot w < 0$
    - $w \leftarrow w + \epsilon \sum_{i \in V} (y_i \; X_i)$

  where ϵ is the **learning rate/step size**. Each step is $O(nd)$ time.

---

[11] x transforms into a hyperplane in w space defined as all w that satisfy $x \cdot w = 0$.
[12] Recall that the gradient points in direction of steepest ascent.

Algorithm: Stochastic GD

- Procedure is simply GD on one data point only per step, i.e. no summation symbol. Called the **perceptron algorithm**.
- **Algorithm:**
  - while some $y_i X_i \cdot w < 0$
    - $w \leftarrow w + \epsilon y_i X_i$
  - return w.
- **Perceptron Convergence:** If data is linearly separable, perfect linear classifier will be found in at most $O(R^2/\gamma^2)$ iterations, where
  - $R = \max_i |X_i|$ is radius of the data
  - γ is the maximum margin.

Maximum Margin Classifiers

- **Margin**: (of a linear classifier) the distance from the decision boundary to the nearest sample point.
- **Goal**: Make the margin as large as possible.
  - Recall that the margin is defined as $|\tau_{min}|$, the magnitude of the smallest euclidean distance from a sample point to the decision boundary, where for some x_i,

$$\tau_i = \frac{|f(x_i)|}{||w||}$$

    and our goal is to maximize the value of the smallest τ in the dataset.
  - Enforce the (seemingly arbitrary?) constraints that $|f(x_i)| \geq 1$, or equivalently

$$y_i(w \cdot x_i + \alpha) \geq 1 \tag{75}$$

    which can also be stated as requiring all $\tau_i \geq 1/||w||$.
  - **Optimize:** Find w and α that minimize $||w||^2$, subject to $y_i(w \cdot x_i + \alpha) \geq 1$ for all $i \in [1, n]$. "Called a **quadratic program** in d+1 dimensions and n constraints. It has **one unique solution.**"
- The solution is a **maximum margin classifier** aka a **hard SVM**.

Soft-Margin SVMs:

- Hard-margin SVMs fail if not linearly separable.
- **Idea:** Allow some points to violate the margin, with **slack variables** ξ

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \tag{76}$$

  where $\xi_i \geq 0$. Note that each sample point is assigned a value of ξ_i, which is only nonzero iff x_i violates the margin.

- To prevent abuse of slack, add a **loss term** to our objective function[13].
  - Find w, α, and ξ_i that minimize our objective function,

$$|w|^2 + C\sum_{i=1}^{n} \xi_i \tag{77}$$

    subject to

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \quad \text{for all } i \in [1, n] \tag{78}$$
$$\xi_i \geq 0 \quad \text{for all } i \in [1, n] \tag{79}$$

    a quadratic program in d + n + 1 dimensions and 2n constraints. The relative size of C, the **regularization hyperparameter** determines whether you are more concerned with getting a large margin (small C) or keeping the slack variables as small as possible (large C).

[13] Before this, looks like our objective function was just $|w|^2$ since that is what we wanted to minimize (subject to constraints).

Shewchuck Chapter 6 **Fall 2016**

Decision Theory:

Table of Contents Local *Scribe: Brandon McKinzie*

- For when "a sample point in feature space doesn't have just one class". Solution is to classify with probabilities.
- **Important terminology:**
  - **Loss Function** $L(z, y)$: Specifies badness of classifying as z when the true class is y. Can be **asymmetrical**. We are typically used to the *0-1 loss function* which is symmetric: 1 if incorrect, 0 if correct.
  - **Decision rule (classifier)** $r : \mathbb{R}^d \to \pm 1$. Maps feature vector x to a class (1 if in class, -1 if not in class for binary case).
  - **Risk**: Expected loss over *all* values of x, y:

$$R(r) = \mathbb{E}\Big[L(r(X), Y)\Big] \tag{80}$$
$$= \sum_y P(Y = y) \sum_x P(X = x|Y = y) L(r(x), y) \tag{81}$$

    In ESL Chapter 2.4, this is denoted as the **expected prediction error**.
  - **Bayes decision rule/classifier** $r*$: Defined as the decision rule $r = r*$ that minimizes $R(r)$. If we assume $L(z, y) = 0$ when $z = y$, then

$$r^*(x) = \begin{cases} 1 & L(-1,1)P(1|x) > L(1,1)P(-1|x) \\ -1 & \text{otherwise} \end{cases} \tag{82}$$

    which has *optimal risk*, also called the **Bayes risk** $R(r^*)$.
- Three ways to build classifiers:
  - **Generative models (LDA)**: Assume sample points come frome class-conditioned probability distributions $P(x|c)$, different for each class. Guess the form of these dists. For each class C, fit (guessed) distributions to points labeled as class C. Also need to estimate (basically make up?) $P(C)$. Use bayes rule and classify on $\max_C P(Y = C|X = x)$. **Advantage**: Can diagnose outliers (small P(x)). Can know the probability that prediction is wrong. **Real definition**: A full probabilistic model of all variables.

- **Discriminative models**. Model $P(Y|X)$ directly. (I guess this means don't bother with modelling all the other stuff like X—Y, just go for it bruh.) **Advantage:** Can know probability of prediction being wrong. **Real definition**: A model only for the target variables.
- **Decision boundary finding**: e.g. SVMs. Model $r(x)$ directly. **Advantage**: Easier; always works if linearly separable; don't have to guess explicit distributions.

Shewchuck Chapter 7 **Fall 2016**

Gaussian Discriminant Analysis:

Table of Contents Local *Scribe: Brandon McKinzie*

- **Fundamental assumption:** Each class C comes from a normal distribution.
- For a given x, want to maximize $P(X = x|Y = C)\pi_C$, where π_C prior probability of class c. Easier to maximize $ln(z)$ since increases monotonically for $z > 0$. The following gives the "quadratic in x" function $Q_C(x)$,

$$Q_C(x) = \ln\left((\sqrt{2\pi})^d P(x)\pi_C\right) \tag{83}$$

$$= -\frac{|x-\mu_C|^2}{2\sigma_C^2} - d\ln\sigma_C + \ln\pi_C \tag{84}$$

  where $P(x)$, a normal distribution, is what we use to estimate the class conditional $P(x|C)$.
- The Bayes decision rule r^* returns the class C that maximizes $Q_C(x)$ above.

Quadratic Discriminant Analysis (QDA)

- Suppose only 2 classes, C and D. Then

$$r^*(x) = \begin{cases} C & Q_C(x) - Q_D(x) > 0 \\ D & \text{otherwise} \end{cases} \tag{85}$$

  which is quadratic in x. The Baye's Decision Boundary (BDB) is the solution of $Q_C(x) - Q_D(x) = 0$.
- In 1D, BDB may have 1 or 2 points (solution to quadratic equation)
- In 2D, BDB is a ***quadric*** (e.g. for d=2, conic section).
- In 2-class problems, naturally leads to **logistic/sigmoid** function for determining $P(Y|X)$.

Newton's Method

- Iterative optimization for some smooth function $J(w)$.
- Can Taylor expand gradient about v:

$$\nabla J(w) = \nabla J(v) + (w - v)\nabla^2 J(v) + \mathcal{O}(|w - v|^2) \tag{86}$$

  where $\nabla^2 J(v)$ is the **Hessian matrix** of $J(w)$ at v, which I'll denote $\mathbf{H}$.

- Find critical point w where $\nabla J(w) = 0$:

$$w = v - H^{-1}\nabla J(v) \tag{87}$$

- Shewchuck defines **Newton's method** algorithm as:
  1. Initialize w.
  2. until convergence do:

     $e := \mathit{solve\_linear\_system}\left(\mathbf{H}e = -\nabla J(w)\right)$.

     $w := w + e$.

  where starting w must be "close enough" to desired solution.

Justifications & Bias-Variance (12)

- Overview: Describes models, how they lead to optimization problems, and how they contribute to underfitting/overfitting.
- Typical model of reality:

$$y_i = f(X_i) + \epsilon_i \tag{88}$$

  where $\epsilon_i \sim D'$ has mean zero.

- Goal of regression: find h that estimates f.

Fall 2016

Probability Review:

Table of Contents Local *Scribe: Brandon McKinzie*

Notation:

- **Sample Space** Ω: Set of all outcomes of a random experiment. For six-sided die, $\Omega = \{1, \ldots, 6\}$.
- **Event Space** $\mathcal{F}$: Set whose *elements* are *subsets* of Ω. Appears that $\mathcal{F}$ is required to be complete in a certain sense, i.e. that it should contain *all* possible events (combinations of possible individual outcomes).
- **Probability measure**: Function $P : \mathcal{F} \to \mathbb{R}$. Intuitively, it tells you what fraction of the total space of possibilities that $\mathcal{F}$ is in, where if $\mathcal{F}$ *is* the full space, $P(F) = P(\Omega) = 1$. Also required: $P(A) \geq 0 \;\; \forall A \in \mathcal{F}$.

Random Variables:[14]

- Consider experiment: Flip 10 coins. An example element of Ω would be of the form

$$\omega_0 = (H, H, T, H, T, H, H, T, H, T) \in \Omega \tag{89}$$

  which is typically a quantity too specific for us to really care about. Instead, we prefer real-valued *functions* of outcomes, known as **random variables**.
- R.V. X is defined as a function $X : \Omega \to \mathbb{R}$. They are denoted as $X(\omega)$, or simply X if ω dependence is obvious.
- Using our definition of the probability measure, we define the probability that $X = k$ as the probability measure over the space containing all outcomes ω where $X(\omega) = k$.[15]

$$P(X = k) := P(\{\omega : X(\omega) = k\}) \tag{90}$$

- **Cumulative Distribution Function:** $F_X : \mathbb{R} \to [0, 1]$ defined as[16]

$$F_X(x) \triangleq P(X \leq x) \tag{91}$$

[14] TIL I had no idea what a random variable really was.
[15] Oh my god yes, this is what I came here for.
[16] The symbol $\triangleq$ means equal by definition (hnnggg). In continuous case, $F_X(x) = \int_{-\infty}^{x} p_X(u)du$.

- **Probability Mass Function**: When X is a *discrete* RV, it is simpler to represent the probability measure by directly saying the probability of each possible value X can assume. It is a function $p_X : \Omega \to \mathbb{R}$ such that

$$p_X(x) \triangleq P(X = x) \tag{92}$$

- **Probability Density Function**: The derivative of the CDF.

$$f_X(x) \triangleq \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} \tag{93}$$

$$P(x \leq X \leq x + \delta x) \approx f_x(x)\delta x \tag{94}$$

EXPECTATION VALUE

- Discrete X: (PMF $p_X(x)$) Can either take expectations of X (the mean) or of some function $g(X) : \mathbb{R} \to \mathbb{R}$, also a random variable.

$$\mathbb{E}[g(X)] \triangleq \sum_{x \in Val(X)} g(x)p_X(x) \tag{95}$$

$$\mathbb{E}[X] \triangleq \sum_{x \in Val(X)} xp_X(x) \tag{96}$$

- Continuous X: (PDF $f_X(x)$), then

$$\mathbb{E}[g(X)] \triangleq \int_{-\infty}^{\infty} g(x)f_X(x)\mathrm{d}x \tag{97}$$

- **Properties**:

$$\mathbb{E}[a] = a \quad \forall a \in \mathbb{R} \tag{98}$$

$$\mathbb{E}[a\ f(X)] = a\mathbb{E}[f(X)] \tag{99}$$

$$\mathbb{E}[f(X) + g(X)] = \mathbb{E}[f(X)] + \mathbb{E}[g(X)] \tag{100}$$

$$\mathbb{E}[bool(X == k)] = P(X = k) \tag{101}$$

Variance: Measure of how concentrated the dist of a RV is around its mean.

$$Var[X] \triangleq \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{102}$$

$$= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{103}$$

with properties:

$$Var[a] = 0 \quad \forall a \in \mathbb{R} \tag{104}$$

$$\mathit{\Delta}[af(X)] = a^2 Var[f(X)] \tag{105}$$

Distribution	PDF or PMF	Mean	Variance
$Bernoulli(p)$	$\begin{cases} p, & \text{if } x = 1 \\ 1-p, & \text{if } x = 0. \end{cases}$	p	$p(1-p)$
$Binomial(n, p)$	$\binom{n}{k} p^k(1-p)^{n-k}$ for $0 \leq k \leq n$	np	npq
$Geometric(p)$	$p(1-p)^{k-1}$ for $k = 1, 2, \ldots$	$\frac{1}{p}$	$\frac{1-p}{p^2}$
$Poisson(\lambda)$	$e^{-\lambda}\lambda^x/x!$ for $k = 1, 2, \ldots$	λ	λ
$Uniform(a, b)$	$\frac{1}{b-a}$ $\forall x \in (a, b)$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
$Gaussian(\mu, \sigma^2)$	$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	μ	σ^2
$Exponential(\lambda)$	$\lambda e^{-\lambda x}$ $x \geq 0, \lambda > 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Covariance

- Recognize that the covariance of two random variables X and Y can be described as a *function* $g : \mathbb{R}^2 \to \mathbb{R}$. Below we define the expectation value for some multivariable function[17], and then we can define the covariance as a particular example.

$$\mathbb{E}[[]g(X, Y)] \triangleq \sum_{x \in Val(X)} \sum_{y \in Val(Y)} g(x, y) p_{XY}(x, y) \tag{106}$$

$$Cov[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{107}$$

- Properties:

$$Cov[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{108}$$

$$Var[X + Y] = Var[X] + Var[Y] + 2Cov[X, Y] \tag{109}$$

Random Vectors

- Suppose we have n random variables $X_i = X_i(\omega)$ all over the same general sample space Ω. Convenient to put them into a **random vector** X, defined as $X : \Omega \to \mathbb{R}^n$ and with $X = [X_1 X_2 \ldots X_n]^T$.

- Let g be some function $g : \mathbb{R}^n \to \mathbb{R}^m$. We can define expectations with notation laid out below.

$$g(X) = \begin{bmatrix} g_1(X) \\ g_2(X) \\ \vdots \\ g_m(X) \end{bmatrix} \qquad \mathbb{E}[g(X)] = \begin{bmatrix} \mathbb{E}[g_1(X)] \\ \mathbb{E}[g_2(X)] \\ \vdots \\ \mathbb{E}[g_m(X)] \end{bmatrix} \tag{110}$$

$$\mathbb{E}[g_i(X)] = \int_{\mathbb{R}^n} g(x_1, \ldots, x_n) f_{X_1, \ldots, X_n} dx_1 \ldots dx_n \tag{111}$$

- For a given $X : \Omega \to \mathbb{R}^n$, its **covariance matrix Σ** is the $n \times n$ matrix with $\Sigma_{ij} = Cov[X_i, X_j]$. Also,

[17] Discrete case shown only. Should be obvious how it would look for continuous.

$$\Sigma = \mathbb{E}[XX^T] - \mathbb{E}[X]\mathbb{E}[X]^T \tag{112}$$

$$= \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] \tag{113}$$

and it satisfies: (1) $\Sigma \succeq 0$ (pos semi-def), (2) Σ is symmetric.

Fall 2016

Probability Review:

Table of Contents Local *Scribe: Brandon McKinzie*

The eigenvalues of a matrix are the zeros of its **characteristic polynomial**, defined as $f(\lambda) = \det(A - \lambda I)$. A vector $v \neq 0$ is an **eigenvector** iff $v \in Null(A - \lambda I)$.

Regardless of offset of plane, the normal vector to ax + by + cz = d is $w = (a, b, c)$. For any point A not on the plane, closest point B to P where B *is* on the plane, is determined by the value of α that solves

$$(A - \alpha(a, b, c)) \cdot (a, b, c) = d \tag{114}$$

since, given α satisfies the equation, $B = (A - \alpha(a, b, c))$ is a point on the plane, constructed by following the direction of w "backwards" from A.

Direct comparison between MLE and **MAP**:[18]

$$\theta_{MLE} = \arg\max_{\theta} \sum_i \log\left(p_X(x|\theta)\right) \qquad \begin{aligned} p(\theta|x) &\propto p_X(x|\theta)p(\theta) \\ \theta_{MAP} &= \arg\max_{\theta} \sum_i \log\Big(p_X(x|\theta)p(\theta)\Big) \\ &= \arg\max_{\theta}\{\log\left[p_X(x|\theta)\right] + p(\theta)\} \end{aligned} \tag{115}$$

Support Vector Machines

19

- Note that (logistic regression)

$$g(\theta^T) \geq 0.5 \iff \theta^T x \geq 0 \tag{116}$$

- Switch to perceptron algorithm where

$$h_{w,b}(x) = g(w^T x + b) = \begin{cases} 1 & w^T x + b \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{117}$$

- Given a training example $(x^{(i)}, y^{(i)})$, define the **functional margin** of (w, b) w.r.t the training example as

[18] MAP also known as Bayesian Density Estimation
[19] Based off Andrew Ng's CS 229 Notes.

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) \tag{118}$$

where $\hat{\gamma}^{(i)} > 0$ means prediction is correct.[20] We can also define with respect to $S = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$ to be the *smallest* of the individual functional margins:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)} \tag{119}$$

- Now we move to **geometric margins**. First, consider figure 2[21]

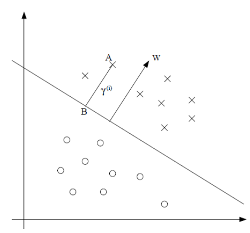


Figure 2: Decision boundary

- If we consider A as the ith data point, what is value of $\gamma^{(i)}$? The point B that is closest to A on the plane is given by $A - \tau \cdot w/||w||$ where τ is the distance $|AB|$ that we want to solve for. Since B is on the plane, we can solve for τ via

$$0 = w^T \left(x^{(i)} - \tau \frac{w}{||w||} \right) + b \tag{120}$$

$$\tau = \frac{w^T x^{(i)} + b}{||w||} \tag{121}$$

which leads to the general definition for the **geometric margin**, denoted *without the hat* $\gamma^{(i)}$ as

$$\gamma^{(i)} = y^{(i)} \left[\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right] \tag{122}$$

[20] Possible insight relating to regularization: Notice how perceptron classification $g(x)$ only depends on the sign of it's argument, and not on the *magnitude*. However, performing $x \to 2x$ causes our functional margin to double $\hat{\gamma}^{(i)} \to 2\hat{\gamma}^{(i)}$ and so it seems "*we can make the functional margin arbitrarily large without really changing anything meaningful.* This leads to, perhaps, defining a normalization condition that $||w||_2 = 1$. Hmmm...

[21] Alright retard, time to settle this once and for all. The plane containing point P_0 and the vector $\boldsymbol{n} = (a, b, c)$ consists of all points P with corresponding position vector $\boldsymbol{r}$ such that the vector drawn from P_0 to P is perpendicular to $\boldsymbol{n}$, i.e. the plane contains all points $\boldsymbol{r}$ such that $\boldsymbol{n} \cdot (\boldsymbol{r} - \boldsymbol{r_0}) = 0$

where clearly if $||w|| = 1$ is the same as the functional margin. Also can define the geometric over the whole training set similarly as was done for the functional margin.

- ***Optimizing (maximizing) the margin***.
  - Pose the following optimization problem

$$\max_{\gamma,w,b} \frac{\hat{\gamma}}{||w||} \quad \textit{S.T.} \tag{123}$$

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq \hat{\gamma} \tag{124}$$

  - Due to reasons primarily regarding how computing $||w||$ is non-convex/hard, we translate the problem as follows: (1) impose (on the *functional margin*[22]) constraint that[23], $\hat{\gamma} = 1$ which we can always satisfy with some scaling of w and b; (2) Instead of maximizing $1/||w||$, minimize $||w||^2$.

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 \quad \textit{S.T.} \tag{125}$$

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \tag{126}$$

    which gives us the **optimal margin classifier**.
  - Lot of subtleties: For SVM (at least in this class) we want maximize the margin, which we **define** to be $2/||w||$. Note that this is *not* a fixed scalar value, it changes as $||w||$ changes! The **support vectors** are any points $x^{(i)}$ such that $y^{(i)}(w^T x^{(i)} + b) = 1$.

PROOF THAT W IS ORTHOGONAL TO THE HYPERPLANE

- Claim: If w is a vector that classifies according to perceptron algorithm (equation 117), then w is orthogonal to the separating hyperplane.
- Proof: We proceed, using only the definition of a plane, by finding the plane that w is orthogonal to, and show that this plane must be the separating hyperplane.
- If we plug in w to the *point-normal form* of the equation of a plane, **defined** as the plane containing all points $\boldsymbol{r} = (x, y, z)$ such that w is orthogonal to the **PLANE**[24]

$$w_x x + w_y y + w_z z + d = 0 \tag{127}$$

$$\boldsymbol{w}^T \boldsymbol{r} + d = 0 \tag{128}$$

$$\tag{129}$$

[22] Recall that the functional margin alone does NOT tell you a distance.

[23] Also remember that $\hat{\gamma}$ is over the WHOLE training set, and evaluates to the smallest $\hat{\gamma}^{(i)}$

[24] NOTE HOW I SAID PLANE AND NOT ANY VECTOR POINTING TO SOME POINT IN THE PLANE

where, denoting $\boldsymbol{r}_0 = (x_0, y_0, z_0)$ as the vector pointing to some arbitrary point P_0 in the plane,

$$d = -(w_x x_0 + w_y y_0 + w_z z_0) \tag{130}$$
$$= -(\boldsymbol{w}^T \boldsymbol{r}_0) \tag{131}$$

which means that

$$0 = \boldsymbol{w}^T \boldsymbol{r} + d \tag{132}$$
$$= \boldsymbol{w}^T \boldsymbol{r} - (\boldsymbol{w}^T \boldsymbol{r}_0) \tag{133}$$
$$= \boldsymbol{w}^T (\boldsymbol{r} - \boldsymbol{r}_0) \tag{134}$$

QED

Spring 2016 Midterm

- Hard-margin SVM and perceptron will not return a classifier if data not linearly separable.
- Soft-margin SVM uses $y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i$, so $\xi_i \neq 0$ for both (1) misclassified samples and (2) all samples inside the margin.
- Large value of C in (Soft-margin SVM) $|w|^2 + C \sum \xi_i$ is prone to **overfitting training** data. Interp: Large C means we want most $\xi_i \to 0$ or small, and therefore the **decision boundary will be sinuous**, something we currently don't know how to do.
- **Bayes classifier** classifies to the most probable class, using the conditional (discrete) distribution $P(G|X)$.

$$\hat{G}(x) = \arg\max_{g \in G} Pr(g|X = x) \tag{135}$$

- $\Sigma^{1/2} = U \Lambda^{1/2} U^T$.

Multivariate Gaussians

- The covariance matrix $\Sigma \in \mathbf{S}^n_+ +$, the space of all symmetric, positive definite nxn matrices.
- Due to this, and since the inverse of any pos. def matrix is also pos. def, we can say that, for all $x \neq \mu$:

$$-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) < 0 \tag{136}$$

- **Theorem:** *For any random vector X with mean μ and covariance matrix Σ* [25]

[25] Also in Probability Review

$$\Sigma = \mathbb{E}[(X-\mu)(X-\mu)^T] = \mathbb{E}[XX^T] - \mu\mu^T \tag{137}$$

- **Theorem:** *The covariance matrix Σ of any random vector X is symmetric positive semidefinite.*
  - In the particular case of Gaussians, which require existence of Σ^{-1}, we also have that Σ is then full rank. "Since any full rank symmetric positive semidefinite matrix is necessarily symmetric positive definite, it follows that Σ must be **symmetric positive definite**.

- The DIAGONAL COVARIANCE MATRIX case. An n-dimensional Gaussian with mean $\mu \in \mathbb{R}^n$ and diagonal $\Sigma = \text{diag}(\sigma_1^2,\ \sigma_2^2, \ldots, \sigma_n^2)$ is the same as n independent Gaussian random variables with mean μ_i and σ_i^2, respectively. (i.e. $P(X) = P(x_1) \cdot P(x_2) \cdots P(x_n)$ where each $P(x_i)$ is a univariate Gaussian PDF.

- **ISOCONTOURS**. General intuitions listed below[26]
  - For random vector $X \in \mathbb{R}^2$ with $\mu \in \mathbb{R}^2$, isocontours are **ellipses** centered on (μ_1, μ_2).
  - *If* Σ diagonal, then principal axes lie along x and y axis. Otherwise, in more general case, they are along the covariance eigenvects. (right?)

- **Theorem:** *Let $X \sim \mathcal{N}(\mu, \Sigma)$ for some $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbf{S}^n_{++}$. Then there exists a matrix $B \in \mathbb{R}^{nxn}$ such that if we define $Z = B^{-1}(X-\mu)$, then $Z \sim \mathcal{N}(0, I)$.*

Misc. Facts

- The sum of absolute residuals is less sensitive to outliers than the residual sum of squares. [Todo: study the flaws of least-squares regression.]

- In LDA, the discriminant functions $\delta_k(x)$ are an *equivalent* description of the decision rule, classifying as $G(x) = \arg\max_k \delta_k(x)$, where (for LDA),

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \tag{138}$$

- Large value of C in soft-margin SVM objective function $|w|^2 + C\sum \xi_i$ is likely to **overfit** training data. This is because it will drive the ξ_i very low/zero, which means *it constructed a (likely nonlinear) decision boundary such that most points were either on or outside the margin.* The key here is that changing the ξ_i associated with points doesn't mean you're ignoring them or something, it means you are manipulating the decision boundary to more closely resemble your training distribution.

[26] Disclaimer: The following were based on an example with $n = 2$ and diagonal Σ. I've done my best to generalize the arguments they made here. I'm like, pretty sure I'm right, but...you know how things can go.

- Can't believe this is necessary, but remember that the sum in the following denominator is over y (not x):

$$P(Y = y_i|X = x_i) = \frac{f_i(x_i)\pi_i}{\sum_{y_j \in Y} f_j(x_i)\pi_j} \tag{139}$$

  If binary class classification, decision boundary is at $x = x*$ where $P(Y = 1|x*) = P(Y = 0|x*) = \frac{1}{2}$. If logistic regression, this occurs when the argument $h(x*)$ to the exponential in denominator is $\exp(h(x*)) = \exp(0) = 1$. So, to find the values of x along decision boundary, in this particular case, you'd solve $h(x) = 0$.

- **[DIS3.2]** Ok. First, never forget that

$$1 = \int_{x \in X|Y_i} f_{X|Y=Y_i}(x)dx \tag{140}$$

  and, therefore, if you're told that x_n sampled

  > iid and uniformly at random from 2 equiprobable classes, a disk of radius 1 (Y = +1) and a ring from 1 to 2 (Y = -1)

  then you should be able to see why (hint: the equation I just wrote) $f_{x|Y=+1} = 1/\pi$ for $||X|| \leq 1$ and $f_{x|Y=-1} = 1/3\pi$ for $1 \leq ||X|| \leq 2$. The fact that they are equiprobable mean $f_Y(Y = +1) = f_Y(Y = -1) = \frac{1}{2}$ which means you can write the density of X, f_X.

Elements of Statistical Learning

Contents

ESL **Fall 2016**

Linear Regression:

Table of Contents Local *Scribe: Brandon McKinzie*

- Assumption: The **regression function** $\mathbb{E}[Y|X]$ is linear[27] in the inputs $X_1, \ldots, X_p$.
- Perform well for...
  - Small numbers of training cases.
  - Low signal/noise.
  - Sparse data.

Models and Least-Squares

- The **model**:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{3.1}$$

- Most popular **estimation method** is least-squares.

$$\begin{aligned} RSS(\beta) &= \sum_{i=1}^{n} \Big(y_i - f(x_i)\Big)^2 & (3.2) \\ &= (\boldsymbol{y} - \boldsymbol{X}\beta)^T (\boldsymbol{y} - \boldsymbol{X}\beta)^T & (3.3) \end{aligned}$$

  which is reasonable if training observations (x_i, y_i) represent independent random draws from their population[28].

- First two derivatives wrt to parameter vector β:

$$\begin{aligned} \frac{\partial RSS}{\partial \beta} &= -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\beta) \\ \frac{\partial^2 RSS}{\partial \beta \partial \beta^T} &= 2\boldsymbol{X}^T\boldsymbol{X} \end{aligned} \tag{3.4}$$

[27] or reasonably approximated as linear
[28] and/or if y_i's conditionally indep given the x_i's.

- Assuming that $\mathbf{X}$ has full column rank so that $\mathbf{X}^T\mathbf{X}$ is positive definite[29], set first derive to 0 to obtain the unique solution:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.6}$$

- **Geometry of Least Squares** I GET IT NOW!
  - The $(p+1)$ column vectors of $\mathbf{X}$ span a subspace of $\mathbb{R}^N$.[30].
  - Minimizing $RSS(\beta) = ||\mathbf{y} - \mathbf{X}\beta||^2$ is choosing $\hat{\beta}$ such that the **residual vector** $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to this subspace[31]. Stated another way, (the optimal) $\hat{y}$ is the *orthogonal projection of* $\mathbf{y}$ *onto the column space of* X.
  - Since $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$, we can define this projection matrix (aka hat matrix), denoted as $\mathbf{H}$, where

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{X}\hat{\beta} \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{H}\mathbf{y} \end{aligned} \tag{3.7}$$

Why $Var(\hat{\beta}) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2$***.***

- Note: **Variance-covariance matrix** $\equiv$ Covariance matrix.
- Can express the **correlation matrix** in terms of the covariance matrix:

$$corr(\mathbf{X}) = \Big(diag(\Sigma)\Big)^{-1/2}\Sigma\Big(diag(\Sigma)\Big)^{-1/2} \tag{141}$$

  or, equivalently, the correlation matrix can be seen as the covariance matrix of the standardized random variables $X_i/\sigma(X_i)$.

- Recall from decision theory that, when we want find a function $f(X)$ for predicting some $Y \in \mathbb{R}$, we can do this by *minimizing the risk* (aka the expected prediction error EPE(f)). This is accomplished first by defining a loss function. Here we will use the

[29] A matrix is positive definite if it's symmetric and all its eigenvalues are positive. **What would we do here if X were *not* full column rank?**. **Answer:** $\mathbf{X}$ columns may not be linearly independent if, e.g., two inputs were perfectly correlated $\mathbf{x}_2 = 3\mathbf{x_1}$. The fitted $\hat{\mathbf{y}}$ will still be projection onto $C(\mathbf{X})$, but there will be more than 1 way (not unique) to express that projection. Occurs most often when one or more (qualitative) inputs are coded in a redundant fashion.

[30] This is the **column space** $C(\mathbf{X})$ of $\mathbf{X}$. It is the space of $\mathbf{X}v\ \forall v \in \mathbb{R}^N$, since the produce Xv is just a linear combination of the columns in X with coefficients v_i.

[31] Interpret: $X\beta$ will always lie *somewhere* in this subspace, but we want β such that, when we subtract each component (WOAH JUST CLICKED) from the prediction, they cancel exactly,i.e. $y_i - (\mathbf{X}\beta)_i = 0$ for all dimensions i in $C(X)$. The resultant vector $\mathbf{y} - \hat{y}$ will only contain components outside this subspace, hence it is orthogonal to it by definition.

squared error loss $L(Y, f(X)) = (Y - f(X))^2$. We can express $EPE(f)$ as an integral over all values that Y and X may take on (i.e. the joint distribution). Therefore, we can factor the joint distribution and define $f(x)$ via minimizing EPE piecewise (meaning at each value of $X = x$.) This whole description is written mathematically below.

$$\begin{aligned} EPE(f) &= \mathbb{E}[Y - f(X)]^2 && (2.9) \\ &= \int \Big[y - f(x)\Big]^2 f_{XY}(x, y)dxdy && (2.10) \\ &= \mathbb{E}_X\Big[\mathbb{E}_{Y|X}\Big[(Y - f(X))^2|X\Big]\Big] && (2.11) \end{aligned}$$

and therefore, the best predictor of Y is a function $f : \mathbb{R}^p \to \mathbb{R}$ that satisfies, for each x value separately

$$\begin{aligned} f(x) &= \arg\min_c \mathbb{E}_{Y|X}\Big[(Y - c)^2|X\Big] && (2.12) \\ &= \mathbb{E}[Y|X = x] && (2.13) \end{aligned}$$

which essentially defines what is meant by $\mathbb{E}[Y|X = x]$, also referred to as the **conditional mean**[32].

Bias-Variance Tradeoff

The expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities:

$$\mathbb{E}[y_0 - \hat{f}(x_0)]^2 = Var\Big(\hat{f}(x_0)\Big) + \Big[Bias(\hat{f}(x_0))\Big]^2 + Var(\epsilon) \qquad (142)$$

which is interpreted as the *expected test MSE*: the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. The **overall test MSE** can be computing the average (of this average) over all possible values of x_0 in the TEST set.

- What **bias** means here: On the other hand, bias refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. For example, linear regression assumes that there is a linear relationship between Y and $X_1, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f. In Figure 2.11, the true f is substantially non-linear, so no matter how many training observations we are given, it will not be possible to produce an accurate estimate using linear regression. In other words, linear regression results in high bias in this example. However, in Figure 2.10 the true f is very close to linear, and so given enough data, it should be possible for linear regression to produce an accurate estimate. Generally, more flexible methods result in less bias.

[32] At the same time, don't forget that least-squared error assumption was built-in to this derivation.

- Returning now to the case where know (aka assume) that the true relationship between X and Y is linear

$$Y = X^T\beta + \epsilon \tag{2.26}$$

and so *in this particular case* the least squares estimates are unbiased.

- oh my fucking god. This is the proof: (relies on the fact that $Var(\beta) = 0$ since β is the true (NON RANDOM) vector we are estimating)[33]

$$\begin{aligned}
Var[\hat{\beta}] &= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\Big] && (143)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\big(\mathbf{X}\beta + \epsilon\big)\Big] && (144)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (145)\\
&= Var\Big[\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (146)\\
&= Var\Big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big] && (147)\\
&= \mathbb{E}\Big[\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)^T\Big] && (148)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \mathbb{E}[\epsilon\epsilon^T]\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (149)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \sigma^2\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (150)\\
&= \sigma^2\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) && (151)\\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} && (152)
\end{aligned}$$

where we have assumed that the X are FIXED (not random)[34] and so the variance of (some product of Xs) $\times\ \epsilon$ is like taking the variance with a constant out front. We've also assumed that X^TX (and thus its inverse too) is symmetric, apparently.

Subset Selection (3.3)

- Two reasons why we might not be satisfied with 3.6:

[33] Also, $\forall a \in \mathbb{R} : Var(a + X) = Var(X)$

[34] another way of stating this is that we took the variance given (or conditioned on) each X

1. Prediction accuracy. Often have low bias, high variance. May improve if shrink coefficients. Sacrifices some bias to reduce variance.
2. Interpretation. Sacrifices some of the small details.

- Appears that subset selection refers to retaining a subset of the *predictors* $\hat{\beta}_i$ and discarding the rest.
- Doing this can often exhibit high variance, even if lower prediction error.

Shrinkage Methods (3.4)

- Shrinkage methods are considered *continuous* (as opposed to subset selection) and don't suffer as much from high variability.

ESL **Fall 2016**

Naive Bayes:

Table of Contents Local *Scribe: Brandon McKinzie*

Appropriate when dimension p of feature space is large. It assume that given a class $G = j$, the features X_k are independent:

$$f_j(X) \equiv f_j((X_1, X_2, \ldots, X_p)^T) = \prod_{k=1}^{p} f_{jk}(X_k) \tag{153}$$

which can simplify estimation [of the class-conditional probability densities $f_j(X)$] dramatically: The individual class-conditional marginal densities f_{jk} can each be estimated *separately* using 1D kernel density estimates.