Contents

Machine Learning

CS 189

Contents

Machine Learning	**August 30**

Classification

Table of Contents   Local   *Written by 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 **September 1**

Gradient Descent

Table of Contents Local *Written by 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]

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

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	**September 6**

Stochastic Gradient Descent

Table of Contents   Local   *Written by 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 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 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 **September 8**

Risk Minimization & Optimization Abstractions

 Written by 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:

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

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

- 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**. 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 **September 13**

Decision Theory

Table of Contents Local

Written by 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 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 - $\text{Pr}()_{max}$.

- **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 loss."

- Loss function in regression pretty clear (e.g. least squared loss). Classification is less so. Can use the **"Doubt option"**. Classifier humility [4]. In some range of inputs near the decision boundary, just say "idk".
- Good classifiers have loss closer to Bayes risk, given certain choice of features.
- Three ways of building classifiers:

[4] Wanna see my posterior

- **Generative:** Model the class conditional distribution $P(\widetilde{x}|c_k)$. Model priors $P(c_k)$. Use bayes duh. Want to understand distributions under which data were *generated*. Physicists can get away with this. They have "models"
- **Discriminative:** 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...
  - Posterior probabilities $P(c_i|x)$ turn out to be logistic σ functions.

Machine Learning **September 15**

Multivariate Gaussians and Random Vectors

Table of Contents Local *Written by Brandon McKinzie*

⇒ Recht's Decision notation: $P(x|H_0)$.

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

- Def: vector x with probability density $p(x) : \mathbb{R}^n \rightarrow \mathbb{R}$.
- 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 $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. 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 $\Rightarrow B^T A B$ is p.s.d. $\forall B$.
- If A is p.s.d., then $f(x) = x^T A x$ 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. ⇒ p(x) is a density.

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

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

Machine Learning	**September 20**

Maximum Likelihood

Written by 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	**September 22**

LDA & QDA

Written by 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**
- Many people prefer LDA because optimization is hard.

Machine Learning | **September 27**

Regression

Written by 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 $p(x|y)$...
- $P(data|\theta) = p(y_1, \dots, y_n | x_1, \dots, x_n, \theta) = \prod p(y_i | x_i, \theta)$.
- 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. $A^T(y - Aw) = \mathbf{0}$.

Bias-Variance Tradeoff

- 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 value of x is (by definition) μ, we have that

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

  - How about squared error

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

    which uses the **cyclic property of the trace**: if dot product is scalar, then it is equal to trace of outer product[5].
  - 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

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

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

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

[5] 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[6], 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

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

[6]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 **October 6**

Regularization

Table of Contents Local *Written by Brandon McKinzie*

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

Machine Learning	**October 20**

Neural Networks

Table of Contents   Local   *Written by 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 non-linearity. 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

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

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:

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

- Then we can compute numerical approx to derivative for *one* of the weights:

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

a process typically called the **forward pass**[7] This is $\mathcal{O}(n)$ if there are n weights in the network. But since this is just the derivative for *one* 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:[8]

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

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

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

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

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

[8] 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**]:

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

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

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

Machine Learning	**October 25**

Neural Networks II

Table of Contents Local *Written by 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

$$\delta_i^{(l-1)} = \frac{\partial err(w)}{\partial S_i^{(l-1)}} \tag{66}$$

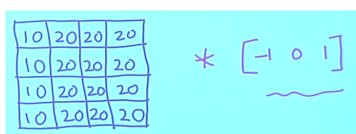
$$= \sum_j \frac{\partial err(w)}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial x_i^{(l-1)}} \frac{\partial x_i^{(l-1)}}{\partial S_i^{(l-1)}} \tag{67}$$

$$= \sum_j \delta_j^{(l)} w_{ij}^{(l)} g'(S_i^{(l-1)}) \tag{68}$$

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.

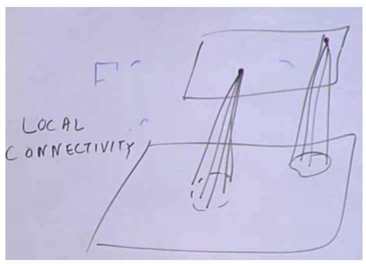
- Very small spot of light means neuron fires, as you increase size of spot, inhibition from surround cells kick in, and its output is diminished. Uses example of ON/OFF cells in retinal ganglia. Neurons can only individually communicate positive values, but multiple neurons can "encode" negative values.
- **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."

[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).



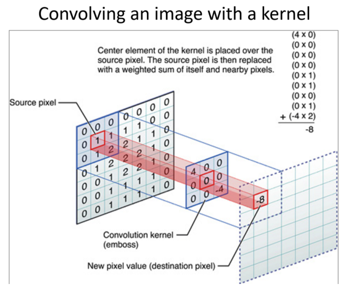
Also recall that the receptive field of retinal ganglion cells can be modeled as a **Difference of Gaussians**:

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

Convolution. The 3x3 convolution kernel[9] 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}).

- 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**.

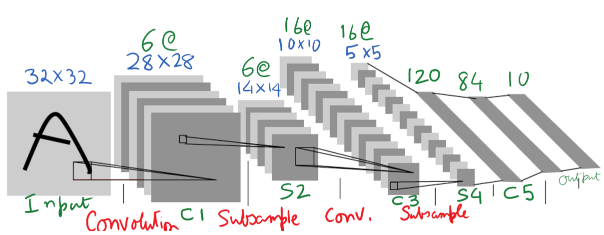
[9]Interchangeable terminology: conv kernel/filter/mask



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) → 28x28 grid. Six such masks. Each masks give 5^2 parameters + 1 for bias.
- Stopped at **[1 hour]**.

Machine Learning **November 1**

Kernel Methods

Table of Contents Local *Written by Brandon McKinzie*

Motivation. Previously, we focused on mapping inputs X to Y via some function f. Neural nets learn features at the same time as learning the linear map. Now we are going to enter the portion of the course on

→ Kernels
→ Nearest Neighbors
→ Decision Trees

which have a different feel than standard linear maps. Today, we focus on **kernels**.

Review of the Optimization Problem. Our usual problem goes as follows: We have a data matrix X that is n by d. We want to solve the following (ridge regression) optimization problem:

$$\min_w ||Xw - y||^2 + \lambda ||w||^2 \tag{70}$$

whose solution can be written in one of the two equivalent forms:

$$w = \left(X^T X + \lambda I_d\right)^{-1} X^T y \tag{71}$$

$$= X^T \left(X X^T + \lambda I_n\right)^{-1} y \tag{72}$$

where the **key idea** to notice is that 71 involved optimizing matrices like $X^T X \in \mathbb{R}^{d,d}$, while 72 involves optimizing matrices like $XX^T \in \mathbb{R}^{n,n}$.

If $n >> d$, then eq 71 is preferred. If $d >> n$, then eq 72 is preferred.

Quick Derivations

Comparing the optimization for w^* compared with $w^* = X^T\alpha^*$. This shows we can arrive at the two equivalent solutions 71 72.

$$w^* = \left(X^TX + \lambda I_d\right)^{-1} X^Ty \qquad \leftrightarrow \qquad \min_w \frac{1}{2}||Xw - y||^2 + \frac{\lambda}{2}||w||^2 \tag{73}$$

$$w^* = X^T\alpha^* = X^T\left(XX^T + \lambda I_n\right)^{-1} y \leftrightarrow \qquad \min_\alpha \frac{1}{2}||XX^T\alpha - y||^2 + \frac{\lambda}{2}||X^T\alpha||^2 \tag{74}$$

Computing α^* consists of computing XX^T ($\mathcal{O}(n^2d)$) and inverting the resulting $n \times n$ matrix (($\mathcal{O}(n^3)$)). Computing w^* consists of computing X^TX ($\mathcal{O}(nd^2)$) and inverting the resulting $d \times d$ matrix (($\mathcal{O}(d^3)$)). So you might want to find α^* when $n << d$.

More detailed look at $w* = X^T\alpha*$:

$$\begin{aligned} w* &= \sum_{i=1}^n x_i \left[\left(XX^T + \mu I_n\right)^{-1} y\right]_i && (75)\\ &= \sum_{i=1}^n x_i \sum_{i=1}^d \left(u_j\Lambda^{-1}u_j^T\right) y_j && (76)\\ &\triangleq \sum_{i=1}^n \alpha_i x_i && (77)\end{aligned}$$

where the u_j are eigenvectors of $XX^T + \mu I_n$ and each y_j is a scalar in this problem.

General optimization problem. Beginning with the newest form for $w*$ above but now for generality we allow

$$w = \sum_{i=1}^n \alpha_i x_i + v \quad \text{where} \quad v^Tx_i = 0 \ \ \forall i$$

and we end up finding that minimizing over α yields $v = 0$ (we did this in homework). The corresponding generalized optimization problem is

$$\min_w \sum_{i=1}^n \text{loss}(w^Tx_i, y_i) + \lambda||w||^2 \tag{78}$$

$$\min_w \sum_{i=1}^n \text{loss}(\sum_{j=1}^n \alpha_j x_j^T x_i, y_i) + \lambda \sum_{j=1}^n\sum_{k=1}^n \alpha_j\alpha_k x_j^T x_k \tag{79}$$

which we see is now over just **the n data points instead of the d dimensions**.

- This changed our problem from d dimensions to n.
- Notice how this new form **only depends on dot products of features** and not the features [alone] themselves.

Big Idea: Kernels

New interpretation: want to find coefficients α such that dissimilar inner products $x_j^T x_i$ become small while similar inner products become large. Solution: the n by n **Kernel matrix**[a] [24:00] $K_{ij} \equiv x_i^T x_j$. With this new definition, our optimization problem can be rewritten as

$$\min_{\alpha} \sum_{i=1}^{n} \text{loss}(\{K\alpha\}_i, y_i) + \lambda \alpha^T K \alpha \tag{80}$$

[a] also known as Gram matrix

Kernel Trick. This new process of rewriting optimization in above form is an example of the **Kernel trick**, where we realize that instead of working with the *features*, we can replace all dot products with the kernel matrix. Also note that "everything has been reduced to dot products" since our new hypothesis function is just

$$f(x) = w^T x = \sum_{i=1}^{n} \alpha_i x_i^T x \tag{81}$$

Kernel functions. Now, imagine we have a function $k(x, z)$ that evaluates inner products. [We've been doing]**Explicit lift**: Map x to higher dimensional space and then perform kernel trick to map it back down to smaller space. *Now*, we want a way to eliminate the middle step. Consider a kernel function on scalar points x, z that *appears to lift* them into a new $d = 3$ space

$$k(x, z) = (1 + xz)^2 \qquad x, z \in \mathbb{R} \tag{82}$$

$$= \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix}^T \begin{bmatrix} 1 \\ \sqrt{2}z \\ z^2 \end{bmatrix} \tag{83}$$

and we see that this is actually equivalent to the dot product in the lifted space. **KEY POINT**: We've actually skipped the process of lifting entirely!

Rather than explicitly (1) lifting the points into the higher dimensional space, then (2) using the kernel trick of computing the dot product to bring back into scalar space, we can just compute $k(x, z)$ directly in one swoop.

Examples. Let $x \in \mathbb{R}^d$ be number of quadratic monomials of $\mathcal{O}(d^2)$. The **quadratic kernel** and **linear kernel** respectively:

$$k(x, z) = \left(1 + x^T z\right)^2 \tag{84}$$

$$k(x, z) = x^T z \tag{85}$$

Radial Basis Functions. For the **Gaussian kernel**, our hypothesis function takes the form of a **RBF**:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i k(x_i, x) && (86) \\ &= \sum_{i=1}^{n} \alpha_i \exp\left[-\gamma ||x_i - x||^2\right] && (87) \end{aligned}$$

where points closest to x will correspond with largest exponentials and thus have their α_i contributing more to overall sum. Remember how quickly exponentials fall off with distance. All we trust is regions nearby.

- Large γ: approaches delta functions. Kernel matrix basically looks like the identity.
- Note that $k(x, x) = 1$ for ALL γ.
- Small γ: Then $exp(\gamma x^T z) \approx 1 + \gamma x^T z$ approaches linear regression.
- How to pick γ? **Cross-validation**.
- If k_1, k_2 are kernels, then $\alpha_1 k_1 + \alpha_2 k_2$ is also a kernel, where $\alpha_1, \alpha_2 > 0$.

Note the feature space of the kernel has an *infinite* number of dimensions:

$$\begin{aligned} k(x, z) &= \exp(-\gamma ||x - z||^2) && (88) \\ &= \exp(2\gamma x^T z) \exp(-\gamma ||x||^2) \exp(-\gamma ||z||^2) && (89) \\ &= \left(\sum_{k=1}^{\infty} \frac{(2\gamma x^T z)^k}{k!} \right) \exp(-\gamma ||x||^2) \exp(-\gamma ||z||^2) && (90) \end{aligned}$$

Recap. For convenience, I've rewritten our general optimization problem below, fully expanded out.

$$\min_w \sum_{i=1}^{n} \text{loss}(\sum_{j=1}^{n} \alpha_j x_j^T x_i, y_i) + \lambda \sum_{j=1}^{n} \sum_{k=1}^{n} \alpha_j \alpha_k x_j^T x_k \tag{91}$$

It is crucial to notice that our algorithm can now be written entirely in terms of the inner products of training data. **Since this is the general case, we've made no assumptions about where the x_i came from, therefore, we are free to substitute $\phi(x_i)$ in for all the x_i.** This is permissible since $\phi(x)$ is nothing more than a feature map to different dimensional space, e.g.

$$\phi(x) = \begin{bmatrix} x & x^2 & x^3 \end{bmatrix}^T \tag{92}$$

Given some feature mapping ϕ, define the corresponding **Kernel** as

$$K(x, z) = \phi(x)^T \phi(z) \tag{93}$$

which will now replace any instance of the inner product $\langle x, z \rangle >$ in our algorithm.

> Often, $K(x, z)$ may be very *inexpensive* to calculate, even though (x) itself may be very *expensive* to calculate (perhaps because it is an extremely high di- mensional vector).

Example 1. Let our kernel be

$$K(x, z) = (x^T z)^2 \tag{94}$$

$$= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \tag{95}$$

which allows us to see that $K(x, z) = \phi(x)^T \phi(z)$ where $\phi(x)$ is a vector consisting of all pairwise products $x_i x_j$, $1 \le i \le n$, $1 \le j \le n$. **Important**: Notice that finding $K(x, z)$ takes $\mathcal{O}(n)$ time, as opposed to $\phi(x)$ taking $\mathcal{O}(n^2)$ time.

The Kernel matrix. Suppose K is a valid kernel corresponding to some mapping ϕ. Consider some finite set of m points[10] $\{x^{(1)}, \dots x^{(m)}\}$ and define a square, m by m matrix $\mathbf{K}$ to be defined such that

$$\mathbf{K}_{ij} \triangleq K\big(x^{(i)}, x^{(j)}\big) = \phi(x^{(i)})^T\phi(x^{(j)}) = \phi(x^{(j)})^T\phi(x^{(i)}) = \mathbf{K}_{ji} \tag{96}$$

This matrix is called the **Kernel matrix**, symmetric by definition[11].

Theorem: *(Mercer) Let $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be given. Then for K to be a valid kernel, it is necessary and sufficient that for any $\{x^{(1)}, \dots x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is **symmetric and positive semi-definite**.*

[10] not necessarily the training set

[11] It is also p.s.d. For any vector z, we have

$$\begin{aligned} z^T\mathbf{K}z &= \sum_i\sum_j z_i\mathbf{K}_{ij}z_j && (97)\\ &= \sum_i\sum_j\sum_k z_i\phi_k(x^{(i)})\phi_k(x^{(j)})z_j && (98)\\ &= \sum_k\left(\sum_i z_i\phi_k(x^{(i)})\right)^2 && (99)\\ &\geq 0 && (100)\end{aligned}$$

Kernels - Discussion 10

Transforming the Risk Function. Begin with risk function J we want to minimize over w:

$$J(w) = (\phi(X)w - y)^2 + \lambda |w|^2 \tag{101}$$

In the familiar case where $\phi(X) = X$, we found that we could represent the optimal $w*$ as $\sum_i \alpha_i x_i = X^T \alpha$. More generally, we can let $w* = \phi(X)^T \alpha$. Substituting this into the risk function yields

$$J(\alpha) = \left(\phi(X)\phi(X)^T\alpha - y\right)^2 + \lambda |\phi(X)^T \alpha|^2 \tag{102}$$

$$= (K\alpha - y)^2 + \lambda \alpha^T K \alpha \tag{103}$$

which is now a minimization over α, where[12] $K = \phi(X)\phi(X)^T$. The closed form solution, along with how to use it for classifying some test set X_{test} is:

$$\alpha^* = (K + \lambda I_n)^{-1} y \tag{104}$$

$$\hat{y} = \phi(X_{test})w = \phi(X_{test})\phi(X_{train})^T \alpha^* = K\alpha^* \tag{105}$$

Kernel Trick. For many feature maps ϕ, can find equivalent expression that avoids having to actually compute the feature map (ϕ). For example, we can write the quadratic feature map $\phi(x) = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 & \sqrt{2}x_1 & \sqrt{2}x_2 & 1 \end{bmatrix}^T$ in the form

$$K = \phi(X)\phi(X)^T = (1 + XX^T)^2 \tag{106}$$

[12]Note that this is consistent with $K_{ij} = k(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$.

Machine Learning **November 3**

Nearest Neighbors

Written by Brandon McKinzie

[started at 2:17]

Key idea: Just store all training examples $\langle x_i, f(x_i) \rangle$.

- **(Single) NN:** Given query x_q, locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$.
- **KNN:** Given x_q, vote among its k nearest neighbors (if discrete-valued target function).

Nearest-Neighbor Rule. For *non-parametric* models, the number of parameters grows with the number of training examples, as opposed to *parametric models*, which is independent of training set size.

For KNN, the number of parameters is $\frac{N}{k}$.

which is still considered non-parametric because it grows with N. Sidenote: SVM is parametric, while *kernel* SVM is non-parametric.

Classification with NN. NN classifier allows you to be much more creative about decision boundary (as opposed to, say, a linear classifier). If $N = 2$ decision boundary is a line. For higher N, decision boundary described by a Voronoi diagram. The union of all the Voronoi cells associated with class A gives all locations that would be classified as A.

Analysis/Performance. Let $\epsilon*(x)$ denote the error of optimal prediction, and let $\epsilon_{NN}(x)$ denote the error of the (*single*) nearest-neighbor classifier. Then

$$\lim_{n\to\infty} \epsilon_{NN} \leq 2\epsilon* \tag{107}$$

$$\lim_{n\to\infty, k\to\infty, k/n\to 0} \epsilon_{kNN} = \epsilon* \tag{108}$$

which means it can be rather hard to outperform kNN when we have extremely large datasets.

- **Advantages:** No training needed. Learn complex functions easily (no assumptions on shape of target function). Don't lose information, i.e. has a nice way of adjusting to the data; not limited/restricted by the number of parameters.

  → **Categories are overrated**, because it reduces our ability to adjust to new data. kNN can, in a sense, create new categories on the fly when needed.

- **Disadvantages**: Slow at query time. Lots of storage. Easily fooled by irrelevant attributes (related to curse of dimensionality).

Dimensionality. Remedies: (1) get more data, increase N. (2) decrease the size of space d ("pushing" points together). Bag-of-words models, which have the effect of marginalizing histograms, can be useful in greatly reducing dimensionality.

Learning better features. ML in general can basically be thought of as (1) learn a feature, and then (2) apply a kNN or linear classifier. For example, convolutional neural networks learn the important features of an image by embedding the pixels in a lower dimensional space where they can essentially classify via kNN.

Machine Learning **November 8**

Decision Trees

Table of Contents Local *Written by Brandon McKinzie*

Terminology. Inputs x need not be real-valued. Two types of discrete variables of interest are

→ **Categorical/Nominal:** have 2 or more categories but there is no intrinsic order. Examples: hair color, zip code, gender, types of housing.

→ **Ordinal:** discrete but there is a natural order. Example: education (HS-college-grad school). One can often convert these to real numbers (e.g. years of education).

Entropy and Information.

- Consider some random variable like the result of a biased coin toss. Let $P(H) = p$. If $p = 1$, there is no information gain when we flip the coin and observe heads. Interested in maximizing the information gain.

- Now consider general case of n-sided coin. When does knowing about its outcome give us the most information? A: when $p = 1/n$. Introduce notion of **Information/Surprise**:

$$I(p) \triangleq -\log_b(p) \tag{109}$$

  where p is probability of the event happening, and b is the branching factor (we almost always have b=2). Knowing a rare event has occurred is more informative.

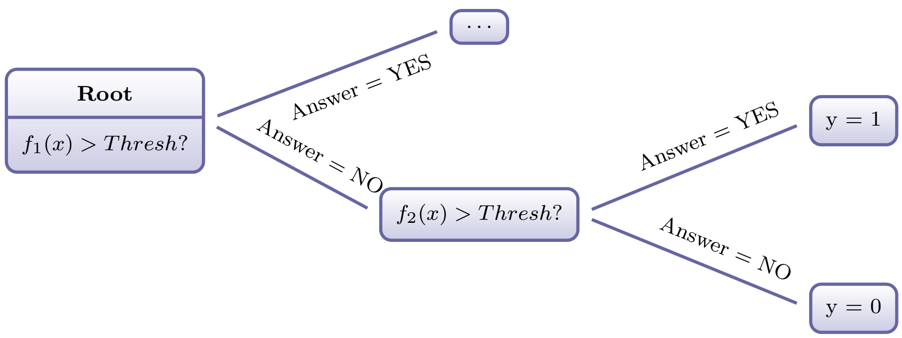
- **Entropy**: the value of information:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_b(p_i) \tag{110}$$

  which is maximized when all n values are equally likely. One way of maximizing entropy is through the use of *Lagrange multipliers*.

Training a Decision Tree. Assume we have data $\{x_1, \ldots, x_n\}$ and desired output y. where each x_i is d-dimensional. Ideally, leaves should be **pure cases**: meaning all outcomes at leaf [from training set] belong to single class. This approach can be used for both classification and regression.

- **Training time**: Construct the tree by picking the "questions" at each node of the tree. This is done to decrease entropy with each level of the tree. A single leaf node is then associated with a set of training examples.[13]
- At **test time**, given a new example that we drop in at root, and it ends up at one of the leaves. What y should we then predict? Predict the *majority vote* from set of values given at the end node. Below, the end nodes only give one value so we would just use that value as our prediction.



If we make the decision tree deep enough, we can guarantee zero training error (bad; overfitting). Can reduce training error by **pruning** or averaging trees together [50:00]. Suppose at test time you land at a leaf containing $\{y = 0, y = 1\}$, meaning "the posterior probability that $y = 1$ is 50 percent. Decision trees give way of computing posterior probabilities from empirical distribution at leaf.

Random Forests. Train multiple trees for solving the same problem. Each tree will give a value for posterior probabilities. Averaging the posterior probabilities over *many* trees is the basic idea of a **random forest** (ensembling decision trees)[14].

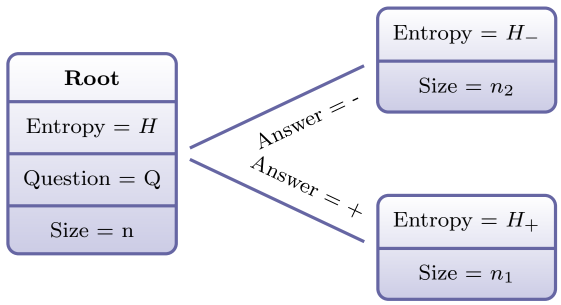
- **Boosting**: rather than just random averaging, assign *weights* to each of the trees in a certain way. Disadvantage: if the labels have noise, then we may end up with more weights on wrong examples. This is opposed to the case of random averaging, which is very robust to noise.

[13] Similar to KNN where distance function is the answer of these questions.

[14] Factoid: ensembling trees gives better performance gain than ensembling neural nets.

Creating the Trees. At the leaves, we want low entropy. Entropies are computed from *empirical frequencies*, not model probabilities. Example: 256 countries that are all equally likely. Then highest possible entropy is 8, which is at the root of the tree. **Creation procedure:**

1. **Begin at root**. Enropy H. Say our questions are T/F, meaning binary branching[15].



2. **Calculate next layer entropy**. Our **goal** is to reduce the entropy as we go down the tree. Take the average over all nodes in the layer. Here, the entropy of the second layer will be
$$\frac{n_1 H_+ + n_2 H_-}{n_1 + n_2}$$
where our diagram above shows the 1st (root) and second layer.

3. **Calculate Information gain**. The information gain, for asking this question is the difference in entropy:
$$I_{1\to 2} := H - \frac{n_1 H_+ + n_2 H_-}{n_1 + n_2} \tag{111}$$

[15]We denote yes as +, and no as −. The H values are the entropy at the given node.

ISLR - Tree-Based Methods

For classification trees, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. Sometimes we also want to know the *class proportions* among the training observation that fall into a given region. To grow the tree, use recursive binary splitting. The two most common criteria used for making the binary splits are the **Gini index** and/or the **cross-entropy** defined, respectively as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{112}$$

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk} \tag{113}$$

where subscripts correspond to mth region, kth class. These are used to evaluate the quality of a particular split when building a classification tree.

Machine Learning	**November 10**

Decision Trees II

Table of Contents   Local   *Written by Brandon McKinzie*

Recap.

- **Advantages** of decision trees: can use a mix of feature types, the output is very explainable.
- **Basic Tree Construction**: While building the tree, at each level, choose the question such that the entropy over the children is minimized[16]. This is a **greedy** approach, meaning that we pick the [single] question that maximizes IG *for our next step only.*
- **Generating different trees:** With previous procedure, we would *always get the same tree.* One possible remedy: Choose a random subset of features *at each level* to optimize question-asking [**randomization**].
  → **Example: Randomization.** Suppose our randomly chosen feature [indices] are $1, 4, 6$. Then, we repeat the following for each index $i \in \{1, 4, 6\}$: Choose threshold such that question, $x_i >$ thresh, maximizes information gain for this feature. After we have our 3 best questions, one for each of these features, we then pick the question out of these that maximizes the IG.
  → **Example: Nominal Feature.** Say feature is hair color can be {black, brown, red}. Possible questions, where h denotes some sample's hair color:

$$h \in \{\text{black}, \text{brown}\}? \quad h \in \{\text{red}, \text{black}\}? \quad h \in \{\text{brown}\}$$

Digit Recognition. Suppose we have three trees, denoted as A, B, C, that each give us their prediction, say $P_A(i)$, $0 \leq i \leq 9$, for tree A, for some test image of a digit. Our prediction for some sample point is just the average

$$P(i) := \frac{1}{3} \sum_{t \in A,B,C} P_t(i)$$

[16]Another way of saying this is "we want to maximize the purity of the children"

Boosting::Motivation. Assign weights to each of our trees based on how much we "trust" their classification abilities[17] How do we decide which classifiers are "experts"? Based on their test performance. We also want classifiers to be experts on *specific examples* as well.

Boosting::Terminology.

→ **Weak Learner:** classifier that is correct at least 50% of the time. T
→ **Error rate:** denoted ϵ_t.
→ **Weighting factor**: for tree t is denoted

$$\alpha_t := \frac{1}{2} \ln \big(\frac{1-\epsilon_t}{\epsilon_t} \big)$$

which is computed during the training process of classifier t (online).

AdaBoost Algorithm. *an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules* [Shapire]. Procedure: Assume we've been given a labeled training set of (x_i, y_i), $1 \le i \le m$. First, initialize $D_1(i) = 1/m \quad \forall i$. Then, for each of T total classifiers $t = 1, \ldots, T$, do:

- Train classifier t using distribution D_t.
- This will yield its corresponding hypothesis function h_t. The **aim** is to select h_t corresponding lowest weighted error $\epsilon_t := Pr_{i \sim D_t}(h_t(x_i) \neq y_i)$
- Compute α_t from formula 1.18 above.
- Update $D_{t+1}(i)$ for each sample i as a lookup table defined as

$$D_{t+1}(i) = \frac{D_t(i) \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}}{Z_t} \tag{114}$$

  where Z_i is just a simple normalization factor.

and, upon completion, output final hypothesis as The combination rule over classifiers

$$h(x) := \sum_{t=1}^{T} \alpha_t h_t(x) \tag{115}$$

[17] Analogous to asking a bunch of people questions but weighing the opinions of experts higher.

Misc. Below is how Jitendra described this in lecture, which more informal than above. [52:00]

- What we instruct the classifiers to do. Tell 2nd classifier to work on the hard problems that the first classifier screwed up on. Every new classifier in sequence works harder and harder, meaning they're looking at samples previous classifier got wring. **i.e. making experts**.
- Have probability/weighting distribution over samples. Initially, its equal weight over all training data. But in round 2, give the distribution more weight to examples that prev classifier got wrong. This probability distribution over examples is denoted $D_t(i)$, where examples indexed by i, where

$$\sum_i D_t(i) = 1$$

- Big drawback of boosting comes from label noise. Simple averaging, however, does *not* have the same issue. Boosting could repeatedly weigh the bad labels, making it worse and worse.

Machine Learning **November 15**

Unsupervised Learning

Table of Contents Local *Written by Brandon McKinzie*

Basic Idea. Some set of data, and we want to find a mapping into "no idea." Not all data is useful, and we explore two techniques for dealing with this: dimensionality reduction and clustering.

- **Dimensionality Reduction**. Can we compress the dimension of our dataset? Reducing d has the effect of reducing runtime, storage, generalization, and interpretability. Note that we can choose to reduce n, d, or both.
- **Clustering**. Reducing n has effect of reducing runtime and "understanding archetypes"[18], segmentation, and outlier removal. Segmentation: figuring out what features are relevant/not relevant in terms of the examples.

Most unsupervised learning methods appeal to **SVD**, or at least to **matrix factorization**.

$$\mathbf{X}_{d,n} = \mathbf{A}_{d,r}\mathbf{B}_{r,n} \tag{116}$$

where we might imagine $\mathbf{A}$ as tall and thin, and $\mathbf{B}$ as short and wide. This is relevant, e.g., in the case where we want to see if our data matrix $\mathbf{X}$ can be written as a product using only a subset r of the n examples.

[18] a.k.a. understanding what types of examples are representative.

Singular Value Decomposition. So, how do we factorize matrices? By using Singular Value Decomposition (SVD): Every matrix $\mathbf{X} \in \mathbb{R}^{d,n}$, where $n \geq d$, admits a factorization

$$\mathbf{X}_{d,n} = U_{d,d} S_{d,d} V_{n,d}^T \qquad \text{where} \tag{117}$$
$$U^T U = I_d, \quad V^T V = I_d, \tag{118}$$
$$S = \text{diag}(\sigma_i), \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d \geq d \tag{119}$$

and the σ_i are the **singular values**, and U, V are the **singular vectors**.

- **Interpretation**. An equivalent way of writing this, as well as multiplication by a vector $\boldsymbol{z}$:

$$\mathbf{X} = \sum_{i=1}^{d} \sigma_i u_i v_i^T \qquad \text{and} \qquad \mathbf{X}\boldsymbol{z} = \sum_{i=1}^{d} \sigma_i u_i (v_i^T \boldsymbol{z}) \tag{120}$$

  which means we (1) find the amount that $\boldsymbol{z}$ lies in the direction of v_i, (2) we scale it by σ_i, and (3) multiply the output by u_i.

- **Singular values vs. Eigenvalues**. [35:00] Notice that v_i is *not* an eigenvector of X. However, v_i *is* an eigenvector of $X^T X$ with eigenvalue σ_i^2.

$$\mathbf{X} v_i = \sigma_i u_i \tag{121}$$
$$\mathbf{X}^T \mathbf{X} v_i = \sigma_i \mathbf{X}^T u_i = \sigma_i^2 v_i \tag{122}$$
$$\mathbf{X}\mathbf{X}^T u_i = \sigma_i^2 u_i \tag{123}$$

Computation and Examples. In general, we use the following relation.

$$\mathbf{X}\mathbf{X}^T = USV^TVSU^T = US^2U^T \tag{124}$$

$$\mathbf{X}^T\mathbf{X} = VS^2V^T \tag{125}$$

→ **Symmetric PSD matrices.** Consider some symmetric, psd matrix A, which means it's eigenvalues are non-negative. This means can write **eigenvalue decomposition** $A = W\Lambda W^T$, with $WW^T = I$ and Λ is diagonal as λ_i. In this case, the *eigenvalue decomposition is the same as the SVD*, which is always the case for symmetric psd matrices.

→ **Symmetric only**. ($\sigma_i = |\lambda_i|$) Now consider just some symmetric B but not necessary psd. $B = W\Lambda W^T$ is diagonalizable. $W^TW = I$. The eigvalues of a symmetric matrix are real. Might not be positive. [50:00]. How to get decomposition?

- Suppose first k entries of Λ are non-negative. Consider diagonal Γ with

$$\Gamma_{ii} = \begin{cases} 1 & i \leq k \\ -1 & \text{otherwise} \end{cases} \tag{126}$$

which means $\Lambda\Gamma$ is psd and $W\Gamma$ is orthogonal. Our decomposition is thus

$$B = W(\Lambda\Gamma)(W\Gamma)^T \tag{127}$$

→ **General Square Matrix**. For arbitrary square matrices, *there is no general relationship between its singular values and eigenvalues.* As an example, consider $C = \left[\begin{smallmatrix} 1 & 10^{12} \\ 0 & 1 \end{smallmatrix}\right]$. Its eigenvalues are clearly 1. Its singular values are $\{10^{12}, 10^{-12}\}$. Also, note that

$$\max_z ||Cz|| = \sigma_1 \qquad \text{where } ||z|| = 1 \tag{128}$$

→ Rank-related discussion starts at [1:00:00]. Lecture ends 8 minutes later.

SVD - Independent Notes

Background Material.

- **Eigendecomposition**. Let A be a square $N \times N$ matrix with N linearly independent eigenvectors q_i. Then A can be factorized as $A = Q\Lambda Q^{-1}$ where Q is the square $N \times N$ matrix whose ith column is the eigenvector q_i of A.

SVD Definition. Suppose M is $m \times n$ matrix with either real or complex entries. Then there exists a factorization, called a SVD of M of the form

$$M = U\Sigma V^T \tag{129}$$

where

- U is a $m \times m$ unitary matrix.[19]
- Σ is a diagonal $m \times n$ matrix with non-negative real numbers. The diagonal entries σ_i are the **singular values** of M. Convention is to list the singular values in descending order.
- V^T is $n \times n$ unitary matrix.

Relation to eigenvalue decomp. SVD is general, while eigenvalue decomp is a special case for certain square matrices.

[19] Unitary: square matrix with conjugate transpose equal to its inverse.

Proving the outer product summation of X to myself. $X = \sum_{i=1}^{d} \sigma_i u_i v_i^T$

$$U = \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \tag{130}$$

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_d \end{bmatrix} \qquad \leftrightarrow \qquad V^T = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_d^T \end{bmatrix} \tag{131}$$

$$USV^T = \begin{bmatrix} u_1 & u_2 & \cdots & u_d \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_d \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_d^T \end{bmatrix} \tag{132}$$

$$= \begin{bmatrix} u_1\sigma_1 & u_2\sigma_2 & \cdots & u_d\sigma_d \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_d^T \end{bmatrix} \tag{133}$$

$$\sum_{i=1}^{d} \sigma_i u_i v_i^T \tag{134}$$

Machine Learning **November 17**

Principal Component Analysis

Written by Brandon McKinzie

SVD Review.

- **Econ-Sized SVD:** If all $\sigma_{r+i} = 0$ in the S matrix, we can just use all the nonzero diagonals (the first r): $\mathbf{X} = \mathbf{U}_r \mathbf{S}_r \mathbf{V}_r^T$ where[20] we are still assuming that $\mathbf{X}$ is $d \times n$. This allows us to compute a $d \times n$ matrix with only $d \cdot r + r + r \cdot n$ entries (from each of U_r, S, V_r).
- **Dimensionality Reduction**. Can reduce problem to r-dimensional without losing any classification power with

$$\hat{\mathbf{X}} := \mathbf{U}_r^T \mathbf{X} = \mathbf{S}_r \mathbf{V}_r^T \tag{135}$$

If our classifier is the $w^T x$ kind, then notice we can let $w = U_r \alpha$, which means we now classify with $w^T x = \alpha^T U_r^T x$.

PCA Algorithm. Our inputs are X matrix of shape $d \times n$.

1. **Centering**. Subtract out the means[21] such that our new data matrix is

$$X_c = \begin{bmatrix} x_1 - \mu_x & x_2 - \mu_x & \dots & x_n - \mu_x \end{bmatrix} \tag{136}$$

2. **SVD**. Compute $\text{SVD}(X_c) = USV^T$.
3. **Return** $[\hat{X} = S_r V_r^T,\ U_r,\ \mu_x]$

[20] In case it's not obvious, dimensions of U_r and V_r are $d \times r$ and $n \times r$, respectively.

[21] We will assume this has been done for the rest of lecture.

View 1: Maximizing Variance. Want variance of low-dimensional $\hat{X}$ to be as close as possible to the actual X. For example, if we choose $r = 1$, then we want to find the direction of maximum variance. Note: AFAIK, u here is not related to U from SVD. Or not?

$$\max_{||u||=1} = \mathrm{Var}(\{u^T x_i\}) \tag{137}$$

Subtract out the first principal component[22] from x_i.

$$\widetilde{x_i} = x_i - (u_1^T x_i)u_1 \tag{138}$$

so $\widetilde{X}$ is centered and orthogonal to u_1. Furthermore $\sigma_1(\widetilde{X}) = \sigma_2(X)$.

Comparing variance. The total variance of the first r principal components, and the total variance in X are, respectively,

$$\sum_{i=1}^{r} \sigma_i(X)^2 \qquad \text{and} \qquad \sum_{i=1}^{d} \sigma_i(X)^2 \tag{139}$$

View 2: Maximum Likelihood. Assume our data can be expressed as

$$X_i = \alpha_i z + \omega_i \tag{140}$$

- z is unknown, not random. $\mathbb{R}^d$.
- α_i is unknown, not random. $\mathbb{R}^1$.
- $\omega_i \sim \mathcal{N}(0, \sigma^2 I_d)$. Added white noise.

Our goal is to find z. Do this via

$$\max_{\alpha, z} \log p(X, \alpha, z) \tag{141}$$

$$= -\frac{1}{2\sigma_2} \sum_{i=1}^{n} ||x_i - \alpha_i z||^2 + [\text{unimportant}] \tag{142}$$

$$\min_{\alpha, z} ||X - z\alpha^T||_F^2 \tag{143}$$

where optimal solution is

$$Z\alpha^T = \sigma_1 U_1 V_1^T \tag{144}$$

which (Z and α) aren't unique, but as a product it is.

[22] Can prove it's "gone" by evaluating $u_1^T \widetilde{x}_i$ (it is zero)

$$\min_{A \in \mathbb{R}^{n \times k},\ Z \in \mathbb{R}^{d \times k}} ||X - ZA^T||_F^2 \tag{145}$$

View 3: Minimizing Projection Error [49:00]. Thinking more like regression.

$$\text{dist}(x_i, L(w)) = ||x_i - \frac{w^T x_i}{||w||^2} w||^2 \tag{146}$$

Latent Factor Analysis. [1:03:00]. Uses PCA to recognize relationships between terms and concepts by applying dimensionality reduction to a term-document matrix. Each row is a document and each column is a word, so X_{ij} represents the number of occurrences of word j in document i. We see that this term-document matrix X is effectively a bag-of-words model, representing an unstructured piece of text.

PCA - Andrew Ng's CS 229 Notes

Data Pre-Processing. Before running PCA, ensure the following is done:

1. Let $\mu = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$
2. Replace each $x^{(i)}$ with $x^{(i)} - \mu$.
3. Let $\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_j^{(i)}\right)^2$ be the variance for the jth feature.
4. Replace each $x_j^{(i)}$ with $x_j^{(i)}/\sigma_j$.

Intuition: Major Axis of Variation. [One approach is to] let the unit vector u so that when the data is projected onto the direction corresponding to u, the variance of the projected data is maximized. Note that the length of the projection of a data point x onto u is given by[23] $x^T u$. Also note that the new projected vector[24] is $(x^T u)\hat{u}$.

Maximizing Variance. We can formalize the previous paragraph with the following optimization problem.

$$u \leftarrow \underset{||u||=1}{\arg\max}\left[\frac{1}{n}\sum_{i=1}^{n}(x^{(i)^T}u)^2\right] \tag{147}$$

$$= \underset{||u||=1}{\arg\max}\, u^T\left(\frac{1}{n}\sum_{i=1}^{n} x^{(i)}x^{(i)^T}\right)u \tag{148}$$

It should be clear that this will return the maximal (principal) eigenvector of Σ, which is the quantity in big parentheses[25].

[23] $x^T u = ||x||\,||u||\cos\theta = x_{\parallel}$, where $x_{\parallel}$ is the component of x along u.
[24] Remember that our data is now mean-zero.
[25] If this is not obvious, you need to review.

Recap. We have found that if we wish to find a 1-dimensional subspace with which to approximate the data, we should choose u to be the principal eigenvector of Σ. More generally, if we wish to project our data into a k-dimensional subspace ($k < n$), we should choose $u_1, \ldots, u_k$ to be the top k eigenvectors of Σ. The u_i's now form a new, orthogonal basis for the data. *Because Σ is symmetric, the u_i's will (or always can be chosen to be) orthogonal to each other.* Finally, to represent $x^{(i)}$ in the new k-dimensional subspace, we project it onto the k **principal components** we found.

$$\hat{x}^{(i)} \leftarrow \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \qquad \text{where} \qquad \hat{x}^{(i)} \in \mathbb{R}^k \tag{149}$$

Properties of PCA.

- PCA can be derived by picking the basis that minimizes the approximation error arising from projecting the data onto the k-dimensional subspace spanned by them.
- More to come ...

Machine Learning	**November 22**

Clustering

Written by Brandon McKinzie

[started at 5:42; goal: end at 7:00]

Introduction.

- **Why cluster?** (1) Find archetypes, (2) segmentation, (3) faster lookups.
- **Approaches.** (1) k-means [quantization], (2) agglomeration [hierarchy], (3) spectral [segmentation].

K-Means. Partition the data points into k-clusters. Let μ_j denote the mean of points in cluster j, where a point x_i is in C_j if

$$||x_i - \mu_j||^2 \le ||x_i - \mu_j'||^2 \tag{150}$$

$$\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i \tag{151}$$

and our optimization problem is to minimize the differences between points and the k means.

$$\min_{\mu_1, \ldots, \mu_k} \sum_{i=1}^{n} \min_j ||x_i - \mu_j||^2 \tag{152}$$

The Algorithm. Initialize $\mu_1, \ldots, \mu_k$ by strategy of choice. Repeat the following until convergence (i.e. assignments stop changing).

1. Set $i \in C_j$ if $||x_i - \mu_j||^2 \le ||x_i - \mu_j'||^2$ for all $1 \le j' \le k$.
2. Set $\mu_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i$ for $j = 1, \ldots, k$.

Initializing μ_i: k-means++. Procedure:

1. Choose μ_1 at random from data positions $x_1, \ldots, x_n$.
2. For $c = 1, \ldots, k-1$, do
   (a) Set $d_i = \min_j ||x_i - \mu_j||^2$ for all $i = 1, \ldots, n$.
   (b) Set $\mu_{c+1} = x_i$ with probability $p_i = d_i / \sum_i d_i$.

Hierarchical Clustering. Alternative approach which doesn't require us to specify the number of clusters K beforehand. Results in a tree-based representation of the observations, called a **dendogram**. First, we define the four most commonly-used types of **linkage** (dissimilarity between two groups of observations).

- **Complete linkage**. The *largest* dissimilarity of all pairwise between observations in cluster A and in B (Maximal intercluster dissimilarity).
- **Single linkage**. ... *smallest* ...... (minimal ...).
- **Average linkage**. ... *average* ...... (mean ...)
- **Centroid linkage**. Dissimilarity between the *centroid* for cluster A and B.
- **Summary and Equations for each:**

$$[COMPLETE] \qquad d(A,B) = \max\{\text{dist}(x,z) : x \in A,\ z \in B\} \tag{153}$$

$$[SINGLE] \qquad d(A,B) = \min\{\text{dist}(x,z) : x \in A,\ z \in B\} \tag{154}$$

$$[AVERAGE] \qquad d(A,B) = \frac{1}{|A|\ |B|} \sum_{x \in A,\ z \in B} \text{dist}(x,z) \tag{155}$$

$$[CENTROID] \qquad d(A,B) = \text{dist}(\mu_A,\ \mu_B) \tag{156}$$

Now, we can give an example via a greedy algorithm.

1. Initialize with n clusters $C_i = \{x_i\}$ for $i = 1, \ldots, n$.
2. Repeat the following until there is one giant cluster:
   (a) For all pairs of clusters (A, B), compute $d(A, B)$.
   (b) For the minimum pair, link $C_{new} = A \cup B$.

Spectral Clustering[26]. View data as a graph, where the nodes are the $x_1, \ldots, x_n$, the edges w_{ij} is the similarity $\text{sim}(x_i, x_j)$. Some common similarity measures are

$$\text{sim}(x_i, x_j) = \frac{x_i^T x_j}{||x_i||\ ||x_j||} \tag{157}$$

$$\text{sim}(x_i, x_j) = \text{k}(x_i, x_j) \tag{158}$$

$$\text{sim}(x_i, x_j) = \begin{cases} 1 & ||x_i - x_j|| < D_0 \\ 0 & \text{otherwise} \end{cases} \tag{159}$$

Now clustering is the process of *graph partitioning*. We start with a fully connected graph, and want to remove as few edges as possible. [56:00]

- **Partition**. We define a partition V_1, V_2 of our node set V as two sets where

$$V_1 \cup V_2 = V \qquad \text{and} \qquad V_1 \cap V_2 = \emptyset \tag{160}$$

- **Cut set**. The amount of weight we've cut. It is just the sum of connections that were previously between V_1 and V_2 that we've now cut out after partitioning.

$$\text{cut}(v_1, v_2) = \sum_{i \in V_1} j \in V_2 w_{ij} \tag{161}$$

- **Optimization problem**. Define a *balanced cut*

$$\min \text{cut}(V_1, V_2) \quad \text{subj. to} \quad |V_1| = |V_2| = \frac{n}{2} \tag{162}$$

which is (very) NP-hard. 2^N different splits; combinatorial problem.

Graph Laplacian. [1:05] Define a cut indicator (vector) $v \in \mathbb{R}^n$ where

$$v_i = \begin{cases} 1 & i \in V_1 \\ -1 & i \in V_2 \end{cases} \tag{163}$$

[26] Page 544 of ESL

$$\begin{aligned}
\mathrm{cut}(V_1, V_2) &= \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} && (164) \\
&= \frac{1}{4} v^T L v && (165) \\
L_{ij} &= \begin{cases} -w_{ij} & i \neq j \\ \sum_l w_{il} & i = j \end{cases} && (166)
\end{aligned}$$

$$\min \frac{1}{4} v^T L v \tag{167}$$

Shewchuck Notes

Contents

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*

$$y_i X_i \cdot w \geq 0 \tag{168}$$

- 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**

$$L(z,\ y_i) = \begin{cases} 0 & y_i z \geq 0 \\ -y_i z & \text{otherwise} \end{cases} \tag{169}$$

  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.

$$R(w) = \sum_{i=1}^{n} L(X_i \cdot w, y_i) = \sum_{i \in V} (-y_i\ X_i \cdot w) \tag{170}$$

  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[27].
  - x in the "negative" class $\Rightarrow$ x and w must be on the **opposite** side of x's hyperplane.

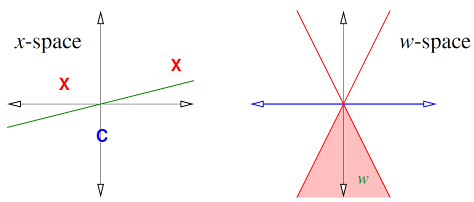


Figure 1: Illustration of how three sample points in x-space (left) can constrain the possible values for w in w space (right).

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 $\nabla R(w)$.[28]

$$\nabla R(w) = \nabla \sum_{i \in V} (-y_i \; X_i \cdot w) \tag{171}$$

$$= -\sum_{i \in V} (y_i \; X_i) \tag{172}$$

- **Algorithm:**
  - $w \leftarrow$ arbitrary nonzero (e.g. any $y_i X_i$)
  - while $R(w) > 0$

[27] x transforms into a hyperplane in w space defined as all w that satisfy $x \cdot w = 0$.

[28] Recall that the gradient points in direction of steepest *ascent*.

        * $V \leftarrow$ 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.

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
    $$y_i(w \cdot x_i + \alpha) \geq 1 \tag{173}$$
    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**.

Shewchuck Chapter 4

Soft-Margin SVMs

Table of Contents Local

Written by Brandon McKinzie

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

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \tag{174}$$

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[29].
  - Find w, α, and ξ_i that minimize our objective function,

$$|w|^2 + C \sum_{i=1}^{n} \xi_i \tag{175}$$

subject to

$$y_i(X_i \cdot w + \alpha) \geq 1 - \xi_i \quad \text{for all } i \in [1, n] \tag{176}$$
$$\xi_i \geq 0 \quad \text{for all } i \in [1, n] \tag{177}$$

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).

[29] 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

Decision Theory

Table of Contents   Local   *Written by 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:

$$R(r) = \mathbb{E}\big[L(r(X), Y)\big] \tag{178}$$

$$= \sum_y P(Y = y) \sum_x P(X = x | Y = y) L(r(x), y) \tag{179}$$

    In ESL Chapter 2.4, this is denoted as the **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

$$r^*(x) = \begin{cases} 1 & L(-1, 1)P(1|x) > L(1, 1)P(-1|x) \\ -1 & \text{otherwise} \end{cases} \tag{180}$$

    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

Gaussian Discriminant Analysis

Table of Contents Local

Written by 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 $Q_C(x)$,

$$Q_C(x) = \ln\left((\sqrt{2\pi})^d P(x)\pi_C\right) \tag{181}$$

$$= -\frac{|x - \mu_C|^2}{2\sigma_C^2} - d\ln\sigma_C + \ln\pi_C \tag{182}$$

  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

$$r^*(x) = \begin{cases} C & Q_C(x) - Q_D(x) > 0 \\ D & \text{otherwise} \end{cases} \tag{183}$$

  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:

$$\nabla J(w) = \nabla J(v) + (w - v)\nabla^2 J(v) + \mathcal{O}(|w - v|^2) \tag{184}$$

  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 $\nabla J(w) = 0$:

$$w = v - H^{-1}\nabla J(v) \tag{185}$$

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

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

     $w := w + e$.

  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:

$$y_i = f(X_i) + \epsilon_i \tag{186}$$

  where $\epsilon_i \sim D'$ has mean zero.
- Goal of regression: find h that estimates f.

Elements of Statistical Learning

Contents

ESL

Linear Regression

Table of Contents   Local   *Written by Brandon McKinzie*

- Assumption: The **regression function** $\mathbb{E}[Y|X]$ is linear[30] 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[31].

- First two derivatives wrt to parameter vector β:

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

- Assuming that $\mathbf{X}$ has full column rank so that $\mathbf{X}^T\mathbf{X}$ is positive definite[32], set first derive to 0 to obtain the unique solution:

[30] or reasonably approximated as linear

[31] and/or if y_i's conditionally indep given the x_i's.

[32] A matrix is positive definite if it's symmetric and all its eigenvalues are positive. **What would we**

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

- **Geometry of Least Squares** I GET IT NOW!
  - The $(p+1)$ column vectors of $\mathbf{X}$ span a subspace of $\mathbb{R}^N$.[33].
  - Minimizing $RSS(\beta) = ||\vec{y} - \mathbf{X}\beta||^2$ is choosing $\hat{\beta}$ such that the **residual vector** $\vec{y} - \hat{\vec{y}}$ is orthogonal to this subspace[34]. Stated another way, (the optimal) $\hat{y}$ is the *orthogonal projection of $\vec{y}$ 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

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

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:

$$corr(\mathbf{X}) = \big(diag(\Sigma)\big)^{-1/2}\Sigma\big(diag(\Sigma)\big)^{-1/2} \tag{187}$$

  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 prediction error EPE(f)). This is accomplished first by defining a loss function. Here we will use the 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

---

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.

[33]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.

[34]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 $\vec{y} - \hat{y}$ will only contain components outside this subspace, hence it is orthogonal to it by definition.

piecewise (meaning at each value of $X = x$.) This whole description is written mathematically below.

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

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

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

which essentially defines what is meant by $\mathbb{E}\left[Y|X = x\right]$, also referred to as the **conditional mean**[35].

Bias-Variance Tradeoff

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

$$\mathbb{E}\left[y_0 - \hat{f}(x_0)\right]^2 = Var\big(\hat{f}(x_0)\big) + \left[Bias(\hat{f}(x_0))\right]^2 + Var(\epsilon) \qquad (188)$$

which is interpreted as the *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.

[35] 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*.

- This is the proof: (relies on the fact that $Var(\beta) = 0$ since β is the *true* (NON RANDOM) vector we are estimating)[36]

$$\begin{aligned}
Var[\hat{\beta}] &= Var\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}\right] & (189)\\
&= Var\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\big(\mathbf{X}\beta + \epsilon\big)\right] & (190)\\
&= Var\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\right] & (191)\\
&= Var\left[\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\right] & (192)\\
&= Var\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\right] & (193)\\
&= \mathbb{E}\Big[\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon\Big)^T\Big] & (194)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \mathbb{E}\left[\epsilon\epsilon^T\right]\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) & (195)\\
&= \Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\ \sigma^2\ \Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) & (196)\\
&= \sigma^2\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)\Big(\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\Big) & (197)\\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} & (198)
\end{aligned}$$

where we have assumed that the X are FIXED (not random)[37] 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:

  1. Prediction accuracy. Often have low bias, high variance. May improve if shrink coefficients. Sacrifices some bias to reduce variance.

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

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

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 **July 30, 2017**

Linear Methods for Classification (Ch. 4)

Table of Contents Local *Written by Brandon McKinzie*

Logistic Regression (4.4). Motivation: we want to model $\Pr[G = k \mid X = x]$, for each of our K classes, via linear functions in x.

$$\log \frac{\Pr[G = k \mid X = x]}{\Pr[G = K \mid X = x]} = (\beta_0)_k + \beta_k^T x \qquad \forall k \in [1, K-1] \tag{199}$$

$$\Pr[G = k \mid X = x] = \frac{e^{\beta_{0k} + \beta_k^T x}}{1 + \sum_{\ell=1}^{K-1} e^{\beta_{0\ell} + \beta_\ell^T x}} \quad \forall k \in [1, K-1] \tag{200}$$

$$\triangleq p_k(x; \theta) \tag{201}$$

where $\theta \triangleq \{\beta_{01}, \beta_1^T, \ldots, \beta_{(K-1)0}, \beta_{K-1}^T\}$.

ESL

Naive Bayes

Written by 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:

$$f_j(X) \equiv f_j((X_1, X_2, \ldots, X_p)^T) = \prod_{k=1}^{p} f_{jk}(X_k) \tag{202}$$

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.

Boosting Methods (10.1)

Terminology:

- **Weak Classifier**: one whose error rate is only slightly better than random guessing.

The AdaBoost algorithm.

1. Initialize observation weights $w_i := 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M[38]:
   (a) Fit classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i} \tag{203}$$
   (c) Compute $\alpha_m = \log\left((1 - \text{err}_m)/\text{err}_m\right)$.
   (d) Update $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, \ldots, N$.
3. Output $G(x) = \sum_{i=1}^m \alpha_m G_m(x)$.

---

[38] where M is the number of weak classifiers (trees) that we want to train.

ESL

Basis Expansions and Regularization (Ch 5)

Table of Contents   Local   *Written by Brandon McKinzie*

Introduction. Core idea is to augment/replace the vector of inputs X with additional variables, which are transformations of X, and then use linear models in this new space of derived inputs features. Specifically, we model a *linear basis expansion* in X:

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X) \tag{204}$$

$h_m(X) : \mathbb{R}^p \mapsto \mathbb{R}$.

where $h_m(X)$ is the mth transformation of X, $m = 1, \ldots, M$.

Piecewise Polynomials and Splines. Assume that X is one-dimensional. We will explore increasingly complex cases that build off each other.

- **Piecewise-constant**. Take, for example, the case where we have 3 basis functions:

$$h_1(X) = I(X < \xi_1), \quad h_2(X) = I(\xi_1 \leq X < \xi_2), \quad h_3(x) = I(\xi_2 \leq X) \tag{205}$$

  Solving for β_i in each derivative of $RSS(\beta)$ w.r.t. β_i, for $f(X) = \sum_{i=1}^{3} \beta_i h_i(X)$, yields $\hat{\beta}_i = \bar{Y}_i$, the mean of Y in the ith (of 3) regions. Note that this is the degree-0 case: piecewise constant.

- **Piecewise-linear**. Instead of just learning the best-fit constant β_i for the ith region, we can now also learn the best *slope* for the line in that region. This means we have 3 additional parameters to fit, and $f(X)$ becomes:

$$\begin{aligned} f(X) &= \beta_1 I_1 + \beta_2 I_2 + \beta_3 I_3 + \beta_4 I_1 \cdot X + \beta_5 I_2 \cdot X + \beta_6 I_3 \cdot X && (206) \\ &= (\beta_1 + \beta_4 X) I_1 + (\beta_2 + \beta_5 X) I_2 + (\beta_3 + \beta_6 X) I_3 && (207) \end{aligned}$$

  where I've denoted the ith identity function from 205 as I_i.
- **Continuous piecewise-linear**. We would typically prefer the lines to meet (have the same value) at the region boundaries of $X = \xi_1$ and $X = \xi_2$. In other words, require that:

$$\begin{aligned} f(\xi_1^-) &= \beta_1 + \beta_4 \xi_1 = \beta_2 + \beta_5 \xi_1 = f(\xi_1^+) && (208) \\ f(\xi_2^-) &= \beta_2 + \beta_5 \xi_2 = \beta_3 + \beta_6 \xi_2 = f(\xi_2 +) && (209) \end{aligned}$$

which also reduces our free parameters from 6 to 4[39]. We can express this constraint more directly by using a basis that incorporates them[40].

$$h_1(X) = 1, \quad h_2(X) = X, \quad h_3(X) = (X - \xi_1)_+, \quad h_4(X) = (X - \xi_2)_+ \tag{210}$$

In general, **an order-M spline with knots $\xi_j,\ j = 1, \ldots, K$ is a piecewise-polynomial of order M, and has continuous derivatives up to order $M - 2$.**

B-splines. Let ξ_0 and ξ_K denote two **boundary knots**, which typically define the domain over which we wish to evaluate our spline. Define the augmented knot sequence τ such that

$$\tau_i \le \tau_2 \le \cdots \le \tau_M \le \xi_0 \tag{211}$$
$$\tau_{M+j} = \xi_j,\ j = 1, \cdots, K \tag{212}$$
$$\xi_{K+1} \le \tau_{K+M+1} \le \tau_{K+M+2} \le \cdots \le \tau_{K+2M} \tag{213}$$

It is customary to make all $\tau_{i,i \le M} = \xi_0$ and $\tau_{j,j \ge K+M+1} = \xi_{K+1}$.

The ith B-spline basis function of order m ($m \le M$) for the knot sequence τ is denoted by $B_{i,m}(x)$, and defined recursively as follows:

$$B_{i,1}(x) = \begin{cases} 1 & \tau_i \le x \le \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \qquad i = 1, \ldots, K + 2M - 1 \tag{214}$$

$$B_{i,m}(x) = \frac{x - \tau_i}{\tau_{i+m-1} - \tau_i} B_{i,m-1}(x) + \frac{\tau_{i+m} - x}{\tau_{i+m} - \tau_{i+1}} B_{i+1,m-1}(x) \quad i = 1, \ldots, K + 2M - m \tag{215}$$

[39] Because, for example, now $\beta_1 = \beta_2 + (\beta_5 - \beta_4)\xi_1$.

[40] Note: I was not able to derive this form from the previous equations and constraints. Moving on because not important right now.

ESL

Model Assessment and Selection (Ch. 7)

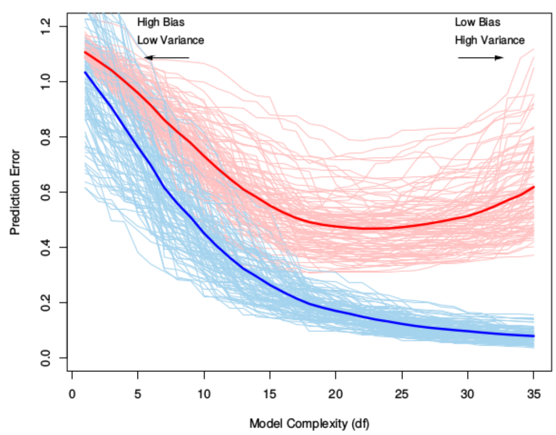
 Written by Brandon McKinzie

Bias, Variance, and Model Complexity. We first define the three main quantities:

$$\text{Generalization (Test) Error} \qquad \mathrm{Err}_{\mathcal{T}} = \mathbb{E}\left[L(Y, \hat{f}(X)) \mid \mathcal{T}\right] \tag{216}$$

$$\text{Expected Prediction Error} \qquad \mathrm{Err} = \mathbb{E}\left[L(Y, \hat{f}(X))\right] = \mathbb{E}\left[\mathrm{Err}_{\mathcal{T}}\right] \tag{217}$$

$$\text{Training Error} \qquad \overline{\mathrm{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{218}$$



Bias-Variance Decomposition. Let $Y = f(X) + \varepsilon$, $\mathbb{E}[\epsilon] = 0$, $\mathrm{Var}[\varepsilon] = \sigma_\varepsilon^2$. We derive the expected prediction error of a fit $\hat{f}(X)$ at an input point $X = x_0$, using squared-error loss:

$$
\begin{aligned}
\operatorname{Err}(x_0) &= \mathbb{E}_{\varepsilon,\hat{f}}\left[(Y-\hat{f}(x_0))^2 \mid X=x_0\right] && (219)\\
&= \mathbb{E}_{\varepsilon,\hat{f}}\left[(f-\hat{f}(x_0)+\varepsilon)^2\right] && (220)\\
&= \mathbb{E}_{\varepsilon}\left[\varepsilon^2\right]+\mathbb{E}_{\hat{f}}\left[(f-\hat{f}(x_0))^2\right] && (221)\\
&= \sigma_\varepsilon^2+\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)^2\right]+f^2-2f\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right] && (222)\\
&= \sigma_\varepsilon^2+\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)^2\right]-\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]^2+\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]^2+f^2-2f\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right] && (223)\\
&= \sigma_\varepsilon^2+\operatorname{Var}\left[\hat{f}(x_0)\right]+\left(\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]^2-2f\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]+f^2\right) && (224)\\
&= \sigma_\varepsilon^2+\operatorname{Var}\left[\hat{f}(x_0)\right]+\left(\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]-f\right)^2 && (225)\\
&= \sigma_\varepsilon^2+\operatorname{Var}\left[\hat{f}(x_0)\right]+\operatorname{Bias}^2\left(\hat{f}(x_0)\right) && (226)
\end{aligned}
$$

ε and $\hat{f}$ are independent.

Remember, $\operatorname{Err}(x_0)$ is the MSE of the prediction at $X = x_0$, taken over all possible models $\hat{f}$ (and the irreducible error from ε). The bias is the difference between the average estimate, $\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]$, and the true mean of Y, $f(x_0)$. Finally, the variance measures how much the distribution over all possible $\hat{f}$ deviates from their average, $\mathbb{E}_{\hat{f}}\left[\hat{f}(x_0)\right]$.

Bootstrap Methods. A general tool for assessing statistical accuracy. Seeks to estimate the conditional error $\operatorname{Err}_{\mathcal{T}}$ but typically estimates well only the expected prediction error Err. Denote our training set $\mathcal{T}$ by $\mathbf{Z} = (z_1, \ldots, z_N)$, where $z_i = (x_i, y_i)$. The basic procedure is as follows:

1. We randomly draw datasets with replacement from $\mathbf{Z}$, with each sampled dataset, $\mathbf{Z}^{*b}$, having the same size, N, as $\mathbf{Z}$.
   - This is done B times, producing B bootstrap datasets.
2. Refit the model to each of the bootstrap datasets, and examine the behavior of the fits over the B replications.

Concept Summaries

Contents

Concept Summaries

Probability Review

Table of Contents   Local   *Written by 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:[41]

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

$$\omega_0 = (H, H, T, H, T, H, H, T, H, T) \in \Omega \tag{227}$$

  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 $X(\omega) = k$.[42]

$$P(X = k) := P(\{\omega : X(\omega) = k\}) \tag{228}$$

- **Cumulative Distribution Function:** $F_X : \mathbb{R} \to [0, 1]$ defined as[43]

$$F_X(x) \triangleq P(X \leq x) \tag{229}$$

[41] TIL I had no idea what a random variable really was.

[42] Oh my god yes, this is what I came here for.

[43] 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

$$p_X(x) \triangleq P(X = x) \tag{230}$$

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

$$f_X(x) \triangleq \frac{dF_X(x)}{dx} \tag{231}$$

$$P(x \leq X \leq x + \delta x) \approx f_x(x)\delta x \tag{232}$$

Expectation Values

– 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.

$$\mathbb{E}\left[g(X)\right] \triangleq \sum_{x \in Val(X)} g(x) p_X(x) \tag{233}$$

$$\mathbb{E}\left[X\right] \triangleq \sum_{x \in Val(X)} x p_X(x) \tag{234}$$

– Continuous X: (PDF $f_X(x)$), then

$$\mathbb{E}\left[g(X)\right] \triangleq \int_{-\infty}^{\infty} g(x) f_X(x) \mathrm{d}x \tag{235}$$

– **Properties**:

$$\mathbb{E}\left[a\right] = a \quad \forall a \in \mathbb{R} \tag{236}$$

$$\mathbb{E}\left[a\ f(X)\right] = a\mathbb{E}\left[f(X)\right] \tag{237}$$

$$\mathbb{E}\left[f(X) + g(X)\right] = \mathbb{E}\left[f(X)\right] + \mathbb{E}\left[g(X)\right] \tag{238}$$

$$\mathbb{E}\left[bool(X == k)\right] = P(X = k) \tag{239}$$

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

$$Var[X] \triangleq \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \tag{240}$$

$$= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 \tag{241}$$

with properties:

$$Var[a] = 0 \quad \forall a \in \mathbb{R} \tag{242}$$

$$\Delta[af(X)] = a^2 Var[f(X)] \tag{243}$$

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[44], and then we can define the covariance as a particular example.

$$\mathbb{E}[g(X, Y)] \triangleq \sum_{x \in Val(X)} \sum_{y \in Val(Y)} g(x, y) p_{XY}(x, y) \tag{244}$$

$$Cov[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{245}$$

- Properties:

$$Cov[X, Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \tag{246}$$

$$Var[X + Y] = Var[X] + Var[Y] + 2Cov[X, Y] \tag{247}$$

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

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 $X = [X_1 X_2 \dots X_n]^T$.

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

$$g(X) = \begin{bmatrix} g_1(X) \\ g_2(X) \\ \vdots \\ g_m(X) \end{bmatrix} \qquad \mathbb{E}\left[g(X)\right] = \begin{bmatrix} \mathbb{E}\left[g_1(X)\right] \\ \mathbb{E}\left[g_2(X)\right] \\ \vdots \\ \mathbb{E}\left[g_m(X)\right] \end{bmatrix} \tag{248}$$

$$\mathbb{E}\left[g_i(X)\right] = \int_{\mathbb{R}^n} g(x_1, \dots, x_n) f_{X_1, \dots, X_n} dx_1 \dots dx_n \tag{249}$$

– 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,

$$\Sigma = \mathbb{E}\left[XX^T\right] - \mathbb{E}\left[X\right]\mathbb{E}\left[X\right]^T \tag{250}$$

$$= \mathbb{E}\left[(X - \mathbb{E}\left[X\right])(X - \mathbb{E}\left[X\right])^T\right] \tag{251}$$

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

Matrix	Properties
$\left(XX^T + \mu I_n\right)$	P.D. with real eigvals $\lambda_i >$ when $\mu > 0$.
p.s.d A, B.	Product AB NOT guaranteed p.s.d.
$C = [A \circ B]_{ij} = A_{ij}B_{ij}$	C is p.s.d. (A,B still p.s.d from above)

Positive Semi-definite.

- Eigenvalues non-negative.

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

$$(A - \alpha(a,b,c)) \cdot (a,b,c) = d \tag{252}$$

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**:[45]

$$\theta_{MLE} = \arg\max_\theta \sum_i \log\left(p_X(x|\theta)\right) \qquad \begin{aligned} p(\theta|x) &\propto p_X(x|\theta)p(\theta) \\ \theta_{MAP} &= \arg\max_\theta \sum_i \log\Big(p_X(x|\theta)p(\theta)\Big) \\ &= \arg\max_\theta \mathbb{E}\left[\log\left[p_X(x|\theta)\right] + p(\theta)\right] \end{aligned} \tag{253}$$

[45]MAP also known as Bayesian Density Estimation

- Note that (logistic regression)

$$g(\theta^T) \geq 0.5 \iff \theta^T x \geq 0 \tag{254}$$

- Switch to perceptron algorithm where

$$h_{w,b}(x) = g(w^T x + b) = \begin{cases} 1 & w^T x + b \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{255}$$

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

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) \tag{256}$$

  where $\hat{\gamma}^{(i)} > 0$ means prediction is correct.[46] 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:

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)} \tag{257}$$

- Now we move to **geometric margins**. First, consider figure 2[47]

[46] 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...

[47] 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 $\boldsymbol{n} \cdot (\boldsymbol{r} - \boldsymbol{r_0}) = 0$

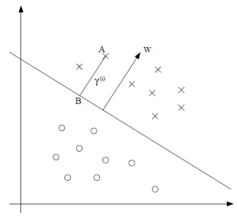


Figure 2: Decision boundary

- 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

$$0 = w^T \left(x^{(i)} - \tau \frac{w}{||w||} \right) + b \tag{258}$$

$$\tau = \frac{w^T x^{(i)} + b}{||w||} \tag{259}$$

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

$$\gamma^{(i)} = y^{(i)} \left[\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right] \tag{260}$$

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

$$\max_{\gamma, w, b} \frac{\hat{\gamma}}{||w||} \quad S.T. \tag{261}$$

$$y^{(i)} \left(w^T x^{(i)} + b \right) \geq \hat{\gamma} \tag{262}$$

  - Due to reasons primarily regarding how computing $||w||$ is non-convex/hard, we translate the problem as follows: (1) impose (on the *functional margin*[48]) constraint that[49], $\hat{\gamma} = 1$ which we can always satisfy with some scaling of w

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

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

and b; (2) Instead of maximizing $1/||w||$, minimize $||w||^2$.

$$\min_{\gamma,w,b} \frac{1}{2}||w||^2 \quad S.T. \tag{263}$$

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \tag{264}$$

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 255), 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**[50]

$$w_x x + w_y y + w_z z + d = 0 \tag{265}$$

$$\boldsymbol{w}^T \boldsymbol{r} + d = 0 \tag{266}$$

$$\tag{267}$$

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

$$d = -(w_x x_0 + w_y y_0 + w_z z_0) \tag{268}$$

$$= -(\boldsymbol{w}^T \boldsymbol{r}_0) \tag{269}$$

which means that

$$0 = \boldsymbol{w}^T \boldsymbol{r} + d \tag{270}$$

$$= \boldsymbol{w}^T \boldsymbol{r} - (\boldsymbol{w}^T \boldsymbol{r}_0) \tag{271}$$

$$= \boldsymbol{w}^T (\boldsymbol{r} - \boldsymbol{r}_0) \tag{272}$$

QED

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

- 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 $P(G|X)$.

$$\hat{G}(x) = \arg\max_{g \in G} Pr(g|X = x) \tag{273}$$

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

- 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 $x \neq \mu$:

$$-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) < 0 \tag{274}$$

- **Theorem:** *For any random vector X with mean μ and covariance matrix Σ* [51]

$$\Sigma = \mathbb{E}\left[(X-\mu)(X-\mu)^T\right] = \mathbb{E}\left[XX^T\right] - \mu\mu^T \tag{275}$$

- **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 $\Sigma = \text{diag}(\sigma_1^2,\ \sigma_2^2, \dots, \sigma_n^2)$ 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[52]
  - 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?)

[51] Also in Probability Review

[52] 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.

- **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),

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{276}$$

- 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.

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

$$P(Y = y_i | X = x_i) = \frac{f_i(x_i)\pi_i}{\sum_{y_j \in Y} f_j(x_i)\pi_j} \tag{277}$$

  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

$$1 = \int_{x \in X|Y_i} f_{X|Y=Y_i}(x)dx \tag{278}$$

  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.

Activation Functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$

PCA

What the heck is PCA? An attempt to remove the many ambiguities.

- **Goal**. Dimensionality reduction. Want to find the best rank-r approximation to the data.
- **Algorithm**.
  1. **Center the data**. $X_c = \begin{bmatrix} x_1 - \mu_x & x_2 - \mu_x & \dots & x_n - \mu_x \end{bmatrix}$ **Why?** Because soon we will want to project along certain axes of the data, which won't make sense unless they [the axes] pass through the origin.
  2. **SVD**. Compute SVD$(X_c) = USV^T$. **Why?** Because (1) the columns of U contain the principal axes that we want to project X onto and (2), this just ends up meaning we return a portion of SV^T (more in next step).
  3. **Return** $[\hat{X} = S_r V_r^T,\ U_r,\ \mu_x]$ **Why?** Because (1) $S_r V_r^T$ is our best rank-r approximation to the data, (2) the columns of U_r give us the principal axes, and (3) μ_x tells us how to un-center our data if we should want to do that.
- **Why maximize variance?** We want our rank-r approximation $\hat{X}$ to resemble the actual X as closely as possible. Intuitively, since dimensionality reduction results in (obviously) lost dimensions from the original X, we should only remove the dimensions that provide the least amount of information about how the samples (rows of X) are distributed. In other words, to best represent X in a smaller number, r, of dimensions, we should choose the dimensions where the data is most spread out/all over the place, as opposed to all close to the mean.

Kernel PCA

- **Motivation for Kernels:** If we want to map sample points to a very high-dimensional feature space, the kernel trick can save us from having to compute those features explicitly, thereby saving a lot of time.
- **Covariance Matrix**. Assume zero mean, $\sum_{i=1}^{n} x_i = 0$. The sample covariance matrix is

$$\hat{\Sigma} = \sum_{i=1}^{n} x_i x_i^T = X^T X \tag{279}$$

  where X is our $n \times d$ data matrix.

Kernel PCA

- Not really sure why: Any vector w $\in R^d$ can be written as $w = w_n + X^T\alpha$ for some w_n in nullspace of X and some $\alpha \in R^n$.

Bias-Variance Tradeoff

Closed-Form Solutions and Properties. Don't forget to put these on your cheat sheet:

$$\hat{\theta}_{ols} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} \tag{280}$$

$$\hat{\theta}_{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda I_d)^{-1}\mathbf{X}^T\vec{y} \tag{281}$$

Also, for $\hat{\theta}_{ols}$, don't forget that

$$\mathrm{Var}(\hat{\theta}_{ols}) = \mathbb{E}\left[\hat{\theta}_{ols}\hat{\theta}_{ols}^T\right] \tag{282}$$

i.e. $\mathbb{E}\left[(\hat{\theta}_{ols} - \mathbb{E}\left[\hat{\theta}_{ols}\right])(\hat{\theta}_{ols} - -\mathbb{E}\left[\hat{\theta}_{ols}\right])^T\right] = \mathbb{E}\left[\hat{\theta}_{ols}\hat{\theta}_{ols}^T\right]$ which is just a consequence of the fact that

$$\mathrm{Var}(X + \mu) = \mathrm{Var}(X) + \mathrm{Var}(\mu) + 2\mathrm{Cov}(X, \mu) \tag{283}$$

$$= \mathrm{Var}(X) \tag{284}$$

Classifiers

Bayes Classifier, Bayes Risk, and Related.

→ The test error rate, $\mathrm{Ave}(\mathrm{I}(\mathrm{y}_0 \neq \hat{\mathrm{y}}_0))$, is minimized on average by the **Bayes classifier**, which assigns test point x_0 to class j for which $\Pr(Y = j \mid X = x_0)$ is largest.
→ Assume we have access to posterior distributions $Pr(Y = j \mid X = x)$.

KNN.

→ Essentially a Bayes classifier that estimates the posterior $\Pr(Y = j \mid X = x)$ as

$$\Pr(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j) \tag{285}$$

where $\mathcal{N}_0$ is the set of K training points closest to x_0.

Risk, Empirical Risk Minimization, and Related

Function Properties. A function $f(x)$ is *convex* iff[53] $f''(x) \geq 0$ for all x in any interval $[a, b]$ in its domain.

Expectations. [SOLVED] My Piazza Post

- **Q:** Why is $\mathbb{E}_S \left[\mathrm{loss}(w^T x_i, y_i)\right] = R[w]$?
- **A:** Because $\mathbb{E}_S$ is <u>NOT</u> the empirical expectation. It is the expectation over

$$S \sim D^n \qquad \text{where} \qquad D^n := \bigtimes_{i=1}^{n} D$$

which means we can state the following:

$$\mathbb{E}_{S \sim D^n}[\mathrm{loss}(w^T x_i, y_i)] = \mathbb{E}_{(x,y)\sim D}[\mathrm{loss}(w^T x, y)]$$

[53]If $f''(x) > 0$, then we say it's *strictly* convex.

Bayes Classifier, Bayes Risk, and Related.

→ The test error rate, $\mathrm{Ave}(\mathrm{I}(\mathrm{y}_0 \neq \hat{\mathrm{y}}_0))$, is minimized on average by the **Bayes classifier**, which assigns test point x_0 to class j for which $\Pr(Y = j \mid X = x_0)$ is largest.

→ **Bayes error rate**. Below, the expectation is over all values that X can take on.

$$1 - \mathbb{E}\left[\max_j \Pr(Y = j \mid X)\right]$$

→ Assume we have access to posterior distributions $Pr(Y = j \mid X = x)$.

Trees

Intuition/Conceptual.

- **Splits**. One useful way of visualizing splits for a given feature, it to *sort* the [labeled] data along that feature. Then, the split is just the left half of the sorted data goes to one branch, the other half goes to the other branch.