Week 3

**Bayes & Naive Bayes**

**Week 3b Regression and Gradients**

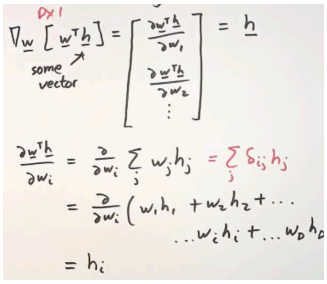
Residual & minimize sum of squared residuals

$$\begin{aligned}\mathbf{r}^\top\mathbf{r} &= (\mathbf{y}-X\mathbf{w})^\top(\mathbf{y}-X\mathbf{w})\\ &= \mathbf{y}^\top\mathbf{y}-2\mathbf{w}^\top X^\top\mathbf{y}+\mathbf{w}^\top X^\top X\mathbf{w}.\end{aligned}$$

Find the gradient: (partial derivative, wT gone)

$$\nabla_{\mathbf{w}}[\mathbf{r}^\top\mathbf{r}] = -2X^\top\mathbf{y}+2X^\top X\mathbf{w}.$$

(why delta W [WT h] = h:)




set all gradient = 0 (move in any direction can't improve the cost), get closed form solution

Assume inverse (X^T X)^(-1) exist (able to compute it)

Assume only one weight could solve this equation

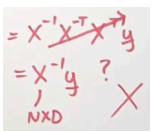
need N >= D, and datapoints not linear dependent

since if N < D (or D linear dependent), then matrix (XT X)^(-1) is low rank (rank N), not invertible

solution: L2 regularization (add datapoints)

$$\begin{aligned}&-2X^\top\mathbf{y}+2X^\top X\mathbf{w} = \mathbf{0}\\ \Rightarrow\quad &X^\top X\mathbf{w} = X^\top\mathbf{y}\\ \Rightarrow\quad &\mathbf{w} = (X^\top X)^{-1}X^\top\mathbf{y}.\end{aligned}$$

why can't expand the inverse and cancel out terms, like this:



since X is N by D matrix, usually we have many more datapoints than dimension, can't invert (inverse only exist for square matrix)

(XT X)^(-1) XT is pseudo inverse

**steepest-descent** method:

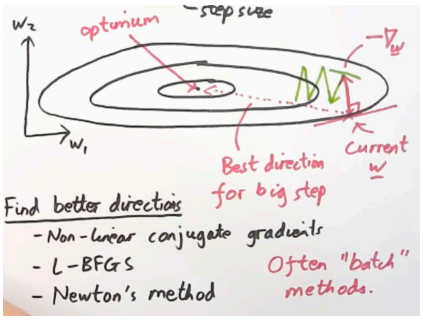
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} [\mathbf{r}^\top \mathbf{r}],$$

Drawback: Gradient descent not in sensible direction (zig zag)

Why gradient descent instead of closed form solution:

Numerical algorithm takes long time, have accuracy guarantees

sometimes, iterative algorithms can have good tradeoffs in accuracy & time (if want guarantee " accurately solving most difficult numerical problem")




rewrite the gradient: (batch gradient descent)

$$\begin{aligned}\nabla_{\mathbf{w}}[\mathbf{r}^\top \mathbf{r}] &= 2X^\top (X\mathbf{w}) - 2X^\top \mathbf{y} \\ &= 2X^\top (\mathbf{f} - \mathbf{y}) \\ &= 2\sum_n \mathbf{x}^{(n)} (f^{(n)} - y^{(n)}).\end{aligned}$$

Mini batch gradient descent: mini batch size B (SGD: B = 1)

since batch GD expensive: one step need to go trough all points

$$= -2N \frac{1}{N} \sum_n \underline{x}^{(n)} (y^{(n)} - f^{(n)})$$
$$\approx -2N \frac{1}{B} \sum_{b=1}^{B} \underline{x}^{(b)} (y^{(b)} - f^{(b)})$$

weights are pulled most in the direction of inputs that had large prediction errors (f-y)

why: look at matrix dimensions & interpretation for mat mul

SGD, each sample gives a crude one-sample Monte Carlo approximation
could drop N, since N could combine into step size alpha

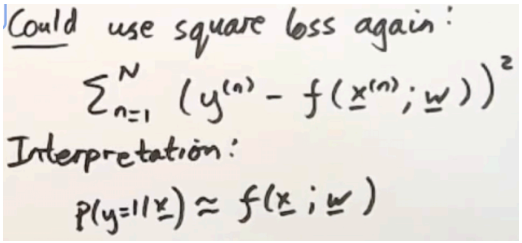
$$\frac{1}{N}\nabla_{\mathbf{w}}[\mathbf{r}^\top \mathbf{r}] \approx 2\mathbf{x}^{(n)}(f^{(n)} - y^{(n)}).$$

**Week 3c**

**Logistic regression**, fit binary labels with linear regression
transform output into [0,1]
(could replace x with basis function)

$$f(\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}.$$

**Loss**: could use **least square**
with logistic sigmoid, y bounded to [0,1], function values = probability (if got enough data & flexible enough function)
Before, this is not very good interpretation, since if fit RBFs, would got probability <0, or function value >1, can't be probability




Given this interpretation (function output = probability), more popular to **maximize likelihood** (of the parameters P(D | w) ), maximize prob of data given w:

$$L(\mathbf{w}) = \prod_{n=1}^{N} P(y^{(n)} \mid \mathbf{x}^{(n)}, \mathbf{w}),$$

**Least square & maximum likelihood:**
different ways to fit weights, got different parameters
interpret one outlier differently: square error doesn't change much with summing an outlier,
but in MLE, multiply big product by number close to 0 changes the whole loss
what if use sum of log likelihood?

Advantage of MLE: if lots of data, converge faster; for small dataset, might prefer square loss

**Minimize negative log likelihood:**

why sum of log likelihood: could apply Monte Carlo trick to do SGD
Normally write in 2 terms (y=1 and y=0):

$$\text{NLL} = -\log L(\mathbf{w}) = -\sum_{n=1}^{N} \log \left[\sigma(\mathbf{w}^\top \mathbf{x}^{(n)})^{y^{(n)}} (1-\sigma(\mathbf{w}^\top \mathbf{x}^{(n)}))^{1-y^{(n)}}\right],$$

or

$$\text{NLL} = -\sum_{n:y^{(n)}=1} \log \sigma(\mathbf{w}^\top \mathbf{x}^{(n)}) - \sum_{n:y^{(n)}=0} \log(1-\sigma(\mathbf{w}^\top \mathbf{x}^{(n)})).$$

But a **trick**: here use label z {-1,1}, use z = 2y-1 to transform from y {0,1}
when label = 1, just sigmoid(WT X),
when label = 0, want 1-sigmoid, = sigmoid(-WT X)
so could simply put 1/-1 label Z inside

There is a trick to write the cost function more compactly. We transform the labels to be $z^{(n)} \in \{-1,+1\}$ where $z^{(n)} = (2y^{(n)}-1)$, and noticing $\sigma(-a) = 1-\sigma(a)$, we can write:

$$\text{NLL} = -\sum_{n=1}^{N} \log \sigma(z^{(n)} \mathbf{w}^\top \mathbf{x}^{(n)}). \tag{6}$$

**Gradient:** (by chain rule)

$$\nabla_{\mathbf{w}} \text{NLL} = -\sum_{n=1}^{N} \nabla_{\mathbf{w}} \log \sigma_n = -\sum_{n=1}^{N} \frac{1}{\sigma_n} \nabla_{\mathbf{w}} \sigma_n = -\sum_{n=1}^{N} \frac{1}{\sigma_n} \sigma_n(1-\sigma_n) \nabla_{\mathbf{w}} z^{(n)} \mathbf{w}^\top \mathbf{x}^{(n)},$$
$$= -\sum_{n=1}^{N} (1-\sigma_n) z^{(n)} \mathbf{x}^{(n)}.$$

$\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$

when classifier is confident and correct on an example, it contributes little to the gradient

SGD improve the examples the classifier gets wrong

Check derivatives https://media.ed.ac.uk/media/1_kyd8kz63

**Week 4 Bayesian regression**

Bayesian classifier: look for a particular set of parameters, do classification with this set of parameters
Bayesian methods: put beliefs on parameters by probability distribution

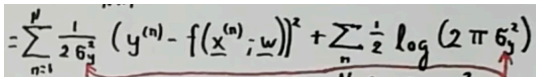
**Probabilistic model for regression:** just a Gaussian around regression line f(x,w)
assume variance sigma y is known and constant

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y;\, f(\mathbf{x}; \mathbf{w}),\, \sigma_y^2),$$

**MLE, negative log likelihood of normal distribution:**
Sum of log likelihood of individual observations: assume observations are independent
**assume constant variance**, **= minimizing square error** (linear regression + constant noise)

$$\begin{aligned} -\log p(\mathbf{y} \mid X, \mathbf{w}) &= -\sum_n \log p(y^{(n)} \mid \mathbf{x}^{(n)}, \mathbf{w}) \\ &= \frac{1}{2\sigma_y^2} \sum_{n=1}^{N} \left[ (y^{(n)} - f(\mathbf{x}^{(n)}; \mathbf{w}))^2 \right] + \frac{N}{2} \log(2\pi\sigma_y^2). \end{aligned}$$

Since (log of normal distribution PDF)



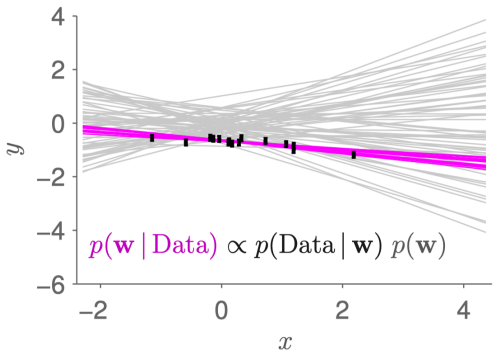
**if different variance for different datapoint**, sigma y is now depends on data point n , **can't put this terms outside the sum**:

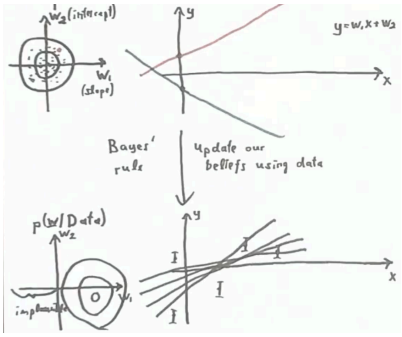
$(\sigma_y{}^{(n)})$

Interpretation: squared difference of each sample weighted by its precision:
if a training point have greater precision, closer to line f(x,w), constrain the resulting lines a lot more, more weight it should carry
**Precision:**

$$\frac{1}{(\sigma_y{}^{(n)})^2}$$

systematically express the uncertainty based on limited and noisy data




**Bayesian regression:**

**prior** e.g. each point of w = a function f(x,w)
(think of parameter space & data point space)
Where to get prior belief: prior knowledge about the problem / have seen some data beforehand

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w};\, \mathbf{0}, 0.4^2\mathbb{I}),$$

posterior by bayesian rule, inputs X fixed
likelihood p(D | w) or p(y | w, X ) (= product over each datapoint)

$$p(\mathbf{w} \mid \mathcal{D}) = p(\mathbf{w} \mid \mathbf{y}, X) = \frac{p(\mathbf{y} \mid \mathbf{w}, X)\, p(\mathbf{w})}{p(\mathbf{y} \mid X)} \propto p(\mathbf{y} \mid \mathbf{w}, X)\, p(\mathbf{w})$$

**conjugate p rior: product of the prior and likelihood**, get same functional form as the prior

get Gaussian posterior:

$$\begin{aligned} p(\mathbf{w} \mid \mathcal{D}) &\propto p(\mathbf{w})\, p(\mathbf{y} \mid \mathbf{w}, X), \\ &\propto \mathcal{N}(\mathbf{w};\, \mathbf{0},\, \sigma_w^2\mathbb{I}) \prod_n \mathcal{N}(y^{(n)}; \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}^{(n)}),\, \sigma_y^2), \\ &\propto \mathcal{N}(\mathbf{w};\, \mathbf{0},\, \sigma_w^2\mathbb{I})\, \mathcal{N}(\mathbf{y};\, \Phi\mathbf{w},\, \sigma_y^2\mathbb{I}), \end{aligned}$$

Vector rep

$$p(\mathbf{w} \mid \mathcal{D}) = \mathcal{N}(\mathbf{w};\, \mathbf{w}_N, V_N)$$

posterior mean & covariance (gives numerical result)

$$V_N = \sigma_y^2(\sigma_y^2 V_0^{-1} + \Phi^\top\Phi)^{-1}$$

$$\mathbf{w}_N = V_N V_0^{-1}\mathbf{w}_0 + \frac{1}{\sigma_y^2} V_N \Phi^\top \mathbf{y}.$$

**Week 4b**, Bayesian inference and prediction

how to use the generated distribution

**Solve prediction problems:** if can't immediately write down expressions for P(y | data),

introduce model parameters and/or latent variables z

by sum rule

$$P(y \mid \text{data}) = \sum_{z} P(y, z \mid \text{data})$$

by product rule (z already known from right probability)

$$P(y \mid \text{data}) = \sum_{z} P(y \mid z, \text{data})\, P(z \mid \text{data})$$

**Prediction P (Ytest | Xtest, D):**

for unknown y in test location x, with training data D, use sum rule & product rule to introduce w (is continuous)

P (y|x,w) instead of P (y|x,w,D); P(w|D) instead of P(w|x,D): doesn't depend on D & test location x

$$p(y \mid \mathbf{x}, \mathcal{D}) = \int p(y, \mathbf{w} \mid \mathbf{x}, \mathcal{D})\, \mathrm{d}\mathbf{w}.$$

$$p(y \mid \mathbf{x}, \mathcal{D}) = \int p(y \mid \mathbf{x}, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, \mathrm{d}\mathbf{w}.$$

here p(w|D) = the Gaussian posterior above

the other term (observation model p( Ytest | Xtest, w ) ): gaussian noise around a function f(x,w):

$$p(y \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(y;\, \mathbf{w}^\top \mathbf{x}, \sigma_y^2).$$

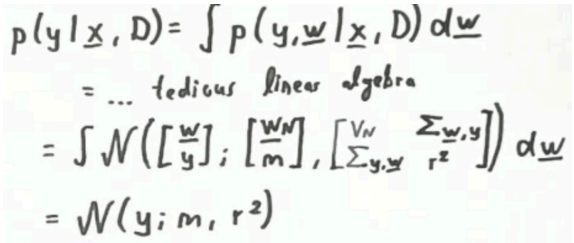
w could be very high dimension, need to **solve high dimension integral** here

2 ways to solve integral: Mechanistic way & Easy way

**Mechanistic way:**

inside the integral is the product of 2 gaussian

use tedious linear algebra to find mean & covariance of this **joint distribution**



**Easy way:**
observation model P (ytest | xtest, w) have particular form: regression line f(x) + constant noise

belief about the function form: f(x) = XT w (x is one test example, D dimension)

$$y = f(\mathbf{x}) + v = \mathbf{x}^\top \mathbf{w} + v, \quad v \sim \mathcal{N}(0, \sigma_y^2)$$

belief about w:

$$p(\mathbf{w} \mid \mathcal{D}) = \mathcal{N}(\mathbf{w};\, \mathbf{w}_N, V_N)$$

**2 steps to get P(ytest | xtest, D):**
1. consider belief about function f

f is linear combination of w, w is gaussian, so P(f) is gaussian

=> **p (f | Xtest , D)** is gaussian,

**mean** is linear combination xT wN

**covariance** is general equation of how covariance changes given a linear transformation (because…)

Distribution over $f$ at test point $\underline{x}$ :

$$p(f \mid \underline{x}, D) = \mathcal{N}(f;\, \underline{x}^T \underline{w}_N,\, \underline{x}^T V_N \underline{x})$$

because $Cov[A\underline{w} + \underline{b}] = A\,Cov[\underline{w}]\,A^T$

$$\mu_f = \mathbb{E}[\mathbf{x}^\top \mathbf{w}] = \mathbf{x}^\top \mathbf{w}_N \qquad \text{var}[f] = \mathbf{x}^\top V_N \mathbf{x},$$

$$\text{var}[f] = \mathbb{E}[(f - \mu_f)(f - \mu_f)] = \mathbb{E}[\mathbf{x}^\top (\mathbf{w} - \mathbf{w}_N)(\mathbf{w} - \mathbf{w}_N)^\top \mathbf{x}] = \mathbf{x}^\top \text{cov}[\mathbf{w}]\mathbf{x}$$

2. Consider belief about y:

Final belief P(ytest | xtest , D) = p (f | Xtest , D) + gaussian noise v

$$p(y \mid \mathcal{D}, \mathbf{x}) = \mathcal{N}(y;\, \mathbf{x}^\top \mathbf{w}_N,\, \mathbf{x}^\top V_N \mathbf{x} + \sigma_y^2).$$

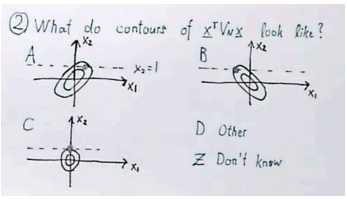
mean, covariance corresponds to m, r^2 in mechanistic way

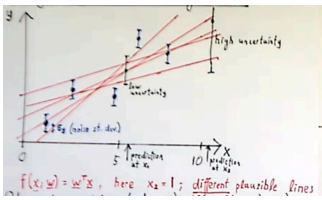
covariance: allow us to quantify how certain we are about prediction (original goal)

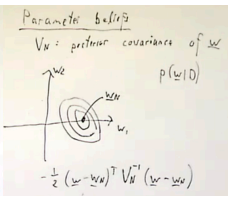
covariance becomes smaller the more data we get, = more certain about the line model; but always have sigma y (when observing infinite data, only sigma y)

larger x, larger covariance, larger uncertainty (most certain at x=0, when bias=0 (according to graph) )

find contour of covariance in x space: look at x VS y graph








Make decisions: To have one output y (instead of just Bayesian approach):

**Loss function:** L(y, y hat ) (y hat is point estimate)

**Cost function:** expected loss given training data D

$$c = \mathbb{E}_{p(y\,|\,\mathbf{x},\mathcal{D})}[L(y,\hat{y})] = \int L(y,\hat{y})\, p(y\,|\,\mathbf{x},\mathcal{D})\, \mathrm{d}y.$$

if squared loss, differentiate the cost w.r.t y

$$L(y,\hat{y}) = (y-\hat{y})^2$$

$$\frac{\partial c}{\partial \hat{y}} = \mathbb{E}_{p(y\,|\,\mathbf{x},\mathcal{D})}[-2(y-\hat{y})] = -2(\mathbb{E}_{p(y\,|\,\mathbf{x},\mathcal{D})}[y] - \hat{y})$$

set derivative = 0, **point estimate y hat = posterior mean**

$$\hat{y} = \mathbb{E}_{p(y\,|\,\mathbf{x},\mathcal{D})}[y]$$

posterior mean corresponds to using an L2 regularized model (but uncertainty of bayesian still useful)

(MSE: symmetric loss, if underestimate the true value, cost would be as big as if you overestimate it, might not appropriate )

(e.g. when the cost of underestimating sales is much higher than the cost of overestimating)