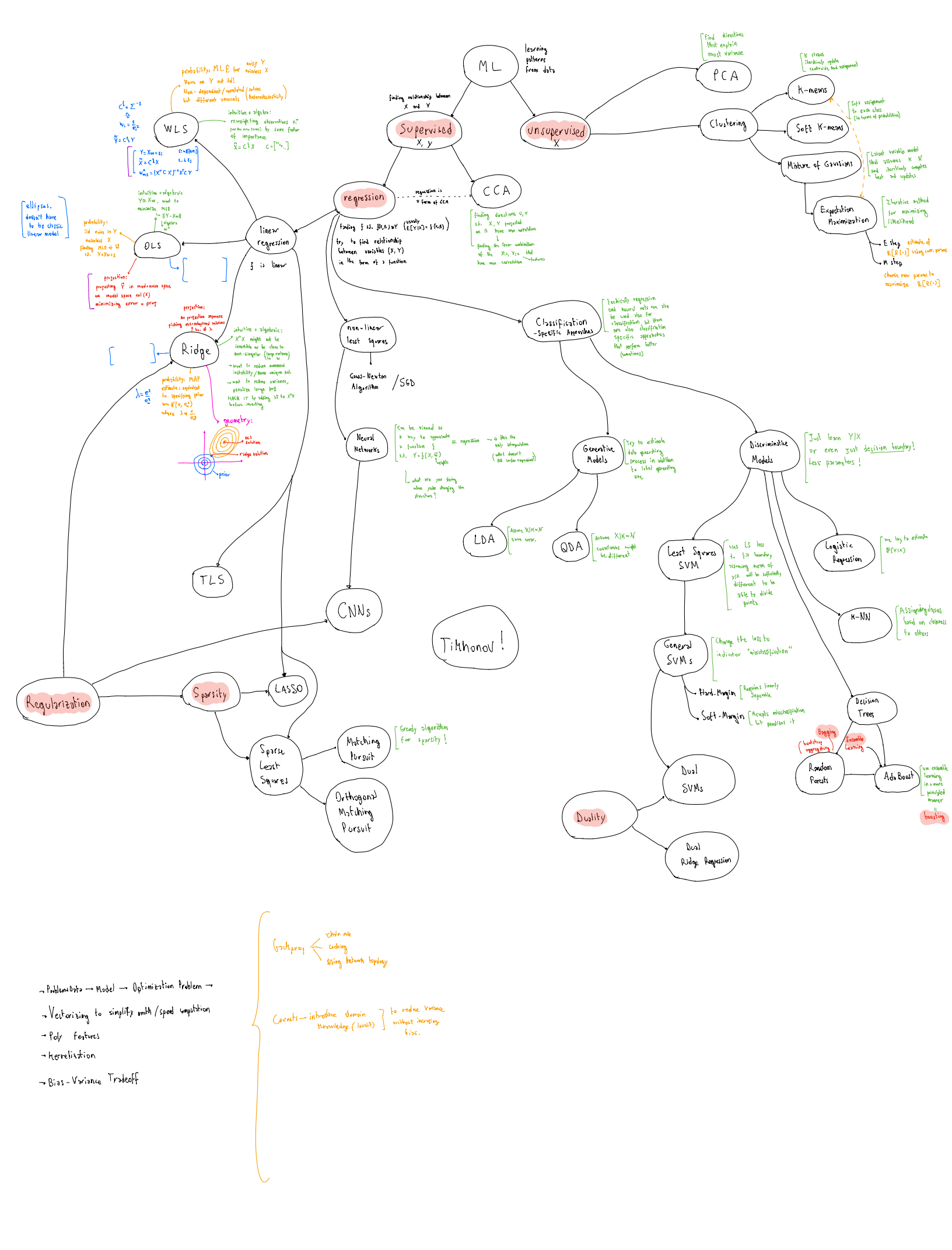

ML
learning patterns from data
Supervised
X, y
finding relationship between X and Y
unsupervised
X
PCA
Find directions that explain most variance
Clustering
K-means
K clusters iteratively update centroids, hard assignment
Soft K-means
Soft assignment to each class (in terms of probabilities)
Mixture of Gaussians
Latent variable model that assumes K N and iteratively computes best mu and variance
Expectation Maximization
Iterative method for maximizing likelihood
E step
estimate of E[R(·)] using curr. params
M step
choose new params to maximize E[R(·)]
regression
CCA
regression is a form of CCA
finding directions U, V s.t. X, V projected on it have max correlation
finding the linear combination of the X:i, Y:i that have max correlation
finding f s.t. f(X) ≈ Y (usually E[Y|X] ≈ f(Xw))
try to find relationship between variables (X, Y) in the form of a function
linear regression
f is linear
WLS
probability: MLE for noisy Y noiseless X
Noise on Y not iid!
Non-dependent / correlated but different amounts
(heteroskedasticity)
intuition + algebra:
reweighting observations $x^{(i)}$ (and the corresponding terms) by some factor of importance
$\hat{X} = C\,|\,X \quad C = [c^{(1)}, \dots]$
$C^{-1} = \Sigma^{-1}$
$w_{ii} = \frac{1}{\sigma_i^2}$
$\hat{Y} = C\,|\,Y$
$Y = Xw + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, \sigma)$
$\hat{X} = C\,|\,X$, $C_{i,i} = \frac{1}{\sigma_i}$
$w_{WLS}^* = (X^T C X)^{-1} X^T C Y$
ellipses. doesn't have to be classic linear model
OLS
intuitive + algebraic
$Y \approx Xw$, want to minimize MSE
$\leftarrow \|Y - Xw\|$ Laplacian w^*
probability:
iid noise in Y, noiseless X
finding MLE of $\vec{w}$ s.t. $Y \sim X w_{MLE}$
projections:
projecting $\hat{Y}$ in model-noise space on model space col(X)
minimizing error = proj
Ridge
projection:
no projection anymore picking anti-colinearity relations if $\lambda \neq 0$
intuitive + algebraic:
$X^T X$ might not be invertible or be close to non-singular (large variance in w)
→ want to reduce numerical instability / have unique sol.
→ want to reduce variance, penalize large $\|w\|$
HACK IT by adding λI to $X^T X$ before inverting
probability: MAP estimate: equivalent to specifying prior $w \sim \mathcal{N}(0, \sigma_w^2)$ where $\lambda \propto \frac{1}{\sigma_w^2}$
$\lambda = \frac{\sigma^2}{\sigma_w^2}$
geometry:
OLS solution
ridge solution
prior
TLS
non-linear least squares
Gauss-Newton Algorithm / SGD
Neural Networks
can be viewed as a way to approximate a function $\hat{f}$ s.t. $Y = \hat{f}(X, \vec{w})$ (weights)
≡ regression
→ is this the only interpolation (what doesn't fit under regression)
↳ what are you doing when you're changing the structure?
CNNs
Classification-Specific Approaches
technically regression and neural nets can also be used also for classification, but there are also classification specific approaches that perform better (sometimes)
Generative Models
Try to estimate lots generating process in addition to label generating one.
LDA
Assume $X|k \sim \mathcal{N}$ same cov.
QDA
Assume $X|k \sim \mathcal{N}$ covariances might be different
Discriminative Models
Just learn Y|X or even just decision boundary! less parameters!
Least Squares SVM
uses LS loss to fit boundary assuming mean of y|X will be sufficiently different to be able to divide points
Logistic Regression
we try to estimate P(y|x)
K-NN
Assigning classes based on closeness to others
General SVMs
Change the loss to indicator "misclassification"
Hard-Margin
Requires linearly Separable
Soft-Margin
Accepts misclassification but penalizes it
Decision Trees
Bagging
(bootstrap aggregating)
Ensemble Learning
Random Forests
AdaBoost
use ensemble learning in a more principled manner
boosting
Duality
Dual SVMs
Dual Ridge Regression
Tikhonov !
Regularization
Sparsity
LASSO
Sparse Least Squares
Matching Pursuit
Greedy algorithm for sparsity !
Orthogonal Matching Pursuit
→ Problem/Data → Model → Optimization Problem →
→ Vectorizing to simplify math / speed computation
→ Poly features
→ Kernelization
→ Bias-Variance Tradeoff
torch.prod
chain rule
caching
fitting between topology
Caveats → introduce domain knowledge (locality)
to reduce variance without increasing bias.

Non linear Least Squares

Until now just considering predictions fns $f(x)$ that are linear in x. However a lot of problems would require non-linear models in their natural form. We could do feature extension but it's not equivalent.

Lets consider an arbitrary model $f(x;\theta)$ (differentiable) that is not necessarily linear. Note that the likelihood of the data will be: → Idea: we always want to "fit" our model, i.e. find its parameters (or estimates of them) so that we can make predictions

Assume $Y_i = f(x_i;\theta) + \varepsilon_i$ $\quad \varepsilon_i \sim N(0, \sigma^2) \Rightarrow Y_i | x_i \sim N(f(x_i|\theta), \sigma^2)$

$$\Rightarrow \hat{\theta}_{MLE} = \underset{\theta}{\arg\min}\ I(\vec{X};\theta) = \underset{\theta}{\arg\min} \sum_i^n \log(Y_i|x_i)$$

$$= \underset{\theta}{\arg\min} \frac{1}{2}\sum_i^n (Y_i - f(x_i|\theta))^2 = \underset{\theta}{\arg\min}\ \varepsilon(\theta)$$

↳ just add this here, useful later

⇒ By first order optimality conditions we know that global minimum must occur at a point when gradient is zero (no convexity required)

$$\Rightarrow \nabla_\theta \varepsilon(\theta) = \sum_i^n (Y_i - f(x_i|\theta))\nabla_\theta f(x_i|\theta) = 0$$

$$= (Y - f(X|\theta))^T J(\theta) = 0 \quad \text{(in more compact notation)}$$

$$= J(\theta)^T (Y - F(\theta)) = 0 \qquad J(\theta) = \begin{bmatrix} \nabla_\theta f(x_1|\theta)^T \\ \vdots \end{bmatrix} \qquad F(\theta) = \begin{bmatrix} f(x_1|\theta) \\ \vdots \end{bmatrix}$$

↓ If not cvx, no closed solution (multiple points =0) even when convex, not always closed

if f linear, f of form $f(x_i|\theta) = \theta^T x_i$ ⇒ $\nabla_\theta f(x_i|\theta) = x_i$ will only depend on X ↳ leads us to OLS solution

Gauss - Newton Algorithm

Must resort to an iterative algorithm → GN at each step linearly approximates $F(\theta)$ for current value of θ

↓ ensures that J can only depend on x_i, so that we can solve for θ (if we want to)] reduces the problem to a least squares one

Have a guess θ at iteration K, θ^K

$$\rightarrow F(\theta) \approx \tilde{F}(\theta) = F(\theta^K) + \frac{\partial}{\partial\theta}F(\theta^K)(\theta - \theta^K)$$

$$= F(\theta^K) + J(\theta^K)\underbrace{(\theta - \theta^K)}_{\Delta\theta}$$

→ Now by FOC as above we get

$$J(\theta^K)^T(Y - \tilde{F}(\theta)) = J(\theta^K)^T[\underbrace{Y - F(\theta^K)}_{\Delta Y} - J(\theta^K)\Delta\theta]$$

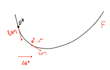
$$= J^T[\Delta Y - J\Delta\theta] = 0 \quad \Rightarrow J^T\Delta Y = J^T J\Delta\theta$$

$$\Rightarrow \theta - \theta^{(K)} = (J^TJ)^{-1}J^T\Delta Y$$

$$\Rightarrow \theta^* = \theta^{(K)} + (J^TJ)^{-1}J^T\Delta Y$$

↳ optimal θ given the approx. $\tilde{F}$

same as solving $\underset{\Delta\theta}{\arg\min} \| J\Delta\theta - Y\|_2^2$



$$\therefore \theta^{K+1} = \theta^K + \Delta\theta = \theta^*$$

Optimization

The general optimization problem will be of the form

$$\min_{x \in X} f(x)$$

most optimization does not have closed form solutions ↓ we just use iterative algorithms

Strong Convexity

A function is m-strongly convex iff any of these equivalent conditions hold:

i) $f(tx + (1-t)y) \le tf(x) + (1-t)f(y) - \frac{t(1-t)m}{2}\|y-x\|^2 \quad \forall x,y,t \in [0,1]$

ii)

v) $\nabla^2 f(x) \succeq mI$

Smoothness

M-smooth ⇔ Lipschitz continuous gradient function iff:

$$\|\nabla f(y) - \nabla f(x)\| \le M\|y-x\|$$

Gradients

$f: \mathbb{R}^n \to \mathbb{R}$ $\quad \nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n}\end{bmatrix}$

• Necessary condition for minimum $\nabla = 0$ → local min, local max; could also be saddle point

Hessian

If f twice differentiable

$$\nabla_x^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2}\end{bmatrix}$$

if x^* is a local minimum, $\nabla^2 f(x^*)$ is PSD (evaluated at x^* ⚠)

if $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \succ 0$ then strict local min (PD!)

Convexity

New property I hadn't seen before:

$$(\nabla f(x) - \nabla f(y))^T(x-y) \ge 0 \quad \forall x,y$$

"if x is greater than y, so will the derivative of f" → follows from second derivative ≥ 0 → first derivative is always INCREASING!

If X is a convex set, and f is a convex function, any local minima of f on X will also be a global minima

Gradient Descent

Iteratively takes steps in the direction of the steepest descent.

From Math 53: directional derivative

$$D_v f(w^{(t)}) = \langle \nabla f(w^t), v\rangle = \|v\|_2 \|\nabla f(w^t)\|_2 \cos\theta$$

want to be smallest! ⇒ $\theta = -\pi$

Initialize $w^{(0)}$
while $f(w^{(t)})$ not converged:
↳ $w^{(t+1)} \leftarrow w^{(t)} - \alpha_t \nabla f(w^{(t)})$

α_t is learning rate, can be fixed or adaptive

1.4.2 Stochastic Gradient Descent

Algorithm 3: Stochastic Gradient Descent
Initialize $\mathbf{w}^{(0)}$ to a random point
while $f(\mathbf{w}^{(t)})$ *not converged* **do**
Sample a random index i_t
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f_{i_t}(\mathbf{w}^{(t)})$

1.4.3 Coordinate Descent

Algorithm 4: Coordinate Descent
Initialize $\mathbf{w}^{(0)}$ to a random point
while $f(\mathbf{w}^{(t)})$ *not converged* **do**
Sample a random feature i_t
$w_{i_t}^{(t+1)} \leftarrow w_{i_t}^{(t)} - \alpha_t \frac{\partial}{\partial w_{i_t}} f(\mathbf{w}^{(t)})$

↳ partial derivative!

1.4.1 Gradient Descent with Momentum

Algorithm 2: Gradient Descent with Momentum
Initialize $\mathbf{w}^{(0)}$ to a random point
while $f(\mathbf{w}^{(t)})$ *not converged* **do**
$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \alpha_t \nabla f(\mathbf{w}^{(t)}) + \beta_t(f(\mathbf{w}^{(t)}) - f(\mathbf{w}_{t-1}))$

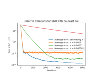
↳ push in old direction

Theoretical guarantees if dist $w^{(0)} \to w^*$

properties of f	stepsize α_t	Convergence rate to $f(w^*)$
Convex, L-Lipschitz	$\frac{R}{L\sqrt{t}}$	$O(\frac{1}{\sqrt{t}})$
m-strongly cvx L-Lipschitz	$\frac{2}{m(t+1)}$	$O(\frac{1}{t})$
cvx M-smooth	$\frac{1}{M}$	$O(\frac{1}{t})$
m-strongly cvx M-smooth	$\frac{1}{M}$	$O(e^{-t\frac{m}{M}})$

Efficiency

SGD will actually converge faster than GD if n is large enough (disc. 8)
↓ even though less improvement at every step



The SGD with decaying learning rate has an error of 0.013 at iteration 5000 and it would continue to decrease if we run more steps.

HW8, ©UCB CS 189, Spring 2018. All Rights Reserved. This may not be publicly shared without explicit permission. 6

In conclusion, when there is an exact solution, SGD with constant learning rate can find the optimal solution. When there is no exact solution, SGD with constant learning rate will approach a place close to the optimal, and the closeness is related to the learning rate. However, when the learning rate is too small, it will take a long time to converge. GD with a constant learning rate can still converge to the optimal. Lastly, SGD with decaying learning rate can converge to the optimal and faster than SGD with a small enough learning rate.

Newton's Method

Second order approximation

Why do we use GD for NNs instead of finding minimum? NNs will have very complicated non-linear functions. Minimizing the loss will not give you a closed form solution. We need $J_F(\theta)^T(y - F(\theta)) = 0$ so we need to compute $J_F(\theta)$, i.e. $\nabla f(x_i;\theta)$, but these gradients would be very complicated and not work (see orange box a couple pages above). Therefore we linearly approximate ??? (what exactly are we linearly approximating?)

We linearly approximate our NN output function as if output was linear in all inputs, and then use closed form solution? WRONG
↳ why Backprop then? Backprop computes exact gradients w.r.t. loss not w.r.t.

In NN we don't have to use lin. approx? B.c. of Backprop? ?? ?

We just directly work with ε, i.e. L and it's derivatives?

What is the point of looking at it from nodes perspective?

Backprop → gives us gradients $\frac{\partial L}{\partial w_i}$ $\frac{\partial L}{\partial b_i}$ → we use them to do G.D.

$L(x_i, \theta, y_i)$ ↳ loss depends on x_s, so does gradient!

We can't find min for

Stochastic Gradient Descent

With backpropagation we can efficiently compute $\frac{\partial L}{\partial w_i}$ or $\frac{\partial L}{\partial b_i}$ for any weight or bias in the network.

Again we want to minimize loss, but cannot find closed form solutions due to the complexity of the derivatives. However we can apply gradient descent!

Computing full gradient might be expensive, especially when we have a lot of data.

To speed things up, we might just want to compute $\nabla_w f$ for a subsection of X

We know $L(w) = \frac{1}{n}\sum_{i=1}^n l(h(x_i), y_i)$

and $\nabla_w L(w) = \frac{1}{n}\sum_{i=1}^n \nabla_w l(h(x_i), y_i)$

↳ computing the gradient takes time linear in n

In regular GD: $w^{k+1} \leftarrow w^k - \alpha_k \nabla_w L(w^k)$

In SGD: $w^{k+1} \leftarrow w^k - \alpha_k G_k$

where G_k is an RV which satisfies $E[G_k] = \nabla_w L(w^k)$

WTS $\lim_{k\to\infty} E[\|w^{(k)} - w^*\|^2] = 0$

$E[\|w^{k+1} - w^*\|^2]$

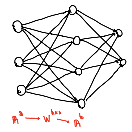
$= E[(w^{k+1} - w^*)^T(w^{k+1} - w^*)]$

$= E[(w^k - \alpha_k G_k - w^*)^T(w^k - \alpha_k G_k - w^*)]$

Gradient Descent & NN

Neural Networks

We only consider feed foreward (DAGs). Most basic is the multilayer perceptron



non linear fn σ

each node i at layer n_ℓ has associated weights $w_i \in \mathbb{R}^{n_{\ell-1}}$

⇒ output of node is $x \to \sigma_i(w_i^T x)$

Putting all weights together: $x \mapsto \sigma(Wx) \in \mathbb{R}^{n_\ell}$ where $W = \begin{bmatrix} w_1^T \\ \vdots \\ w_{n_\ell}\end{bmatrix}$

Nonlinearities

Usually $\sigma: \mathbb{R} \to \mathbb{R}$, applied elementwise

However, one large exception:

Softmax: $\mathbb{R}^n \to \mathbb{R}^n$

$$\sigma_i(x) = \frac{e^{x_i}}{\sum_i^n e^{x_i}}$$

Therefore, we can consider the entire network to be a function:

$$x \longmapsto \sigma_L(W_L \sigma_{L-1}(\cdots \sigma_2(W_2\sigma_1(W_1x))\cdots))$$

→ neural networks are "compositional"

Expressive Power

Important to note that all the power of neural nets comes from the non-linearities. Without them, we would just have a good ol' linear function → a glorified regression!

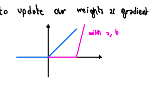
$$x \mapsto W_L W_{L-1} \cdots W_2 W_1 x = \tilde{W}x$$

Also $\text{rank}(\tilde{W}) \le \min_\ell \text{rank}(W_\ell) \le \min_\ell n_\ell$

Choosing non-linearity

Step function is intuitively right, but it doesn't enable us to update our weights as gradient is always zero.

Trying ReLU $\sigma(x) = \begin{cases} x & \text{if } x \ge 0 \\ 0 & \text{if } x < 0\end{cases}$



Trying (and failing) with the step function

Let the function of the network be

$$h(x) = \sum_j^K c_j\sigma(a_j + b_j x)$$

(1 layer deep) K is # hidden units

Trying to choose weights we for OLS with a step fn as σ we get:

$$\min_{a,b,c} \underbrace{\sum_i^n (y_i - h(x_i))^2}_{f(a,b,c)}$$

$$\frac{\partial f}{\partial c_j} = -2(y_i - h(x_i))\sigma(a_j + b_j x) \rightarrow \text{zero if step is off!}$$

$$\frac{\partial f}{\partial a_j} = \sum_i^n -2(y_i - h(x_i))\frac{\partial h(x_i)}{\partial a_j}$$

$$\frac{\partial f}{\partial b_j} = \sum_i^n -2(y_i - h(x_i))\frac{\partial h(x_i)}{\partial b_j}$$

these are zero always! } we never update a, b

bad, bad, bad!

Note that with weights $a, b \to \sigma(a + bx)$ the ReLU can shift and assume arbitrary slopes!

$$\frac{\partial f}{\partial a_j} = \sum_i^n -2(y_i - h(x_i))\frac{\partial h(x_i)}{\partial a_j} = \sum_i^n -2(y_i - h(x_i))c_j\left(\begin{cases} 0 & \text{if } a_j + b_jx_j < 0 \\ 1 & \text{if } a_j + b_jx_j \ge 0\end{cases}\right)$$

$$\frac{\partial f}{\partial b_j} = \sum_i^n -2(y_i - h(x_i))\frac{\partial h(x_i)}{\partial b_j} = \sum_i^n -2(y_i - h(x_i))c_j\left(\begin{cases} 0 & \text{if } a_j + b_jx_j < 0 \\ x_i & \text{if } a_j + b_jx_j \ge 0\end{cases}\right)$$

we have gradients!

Biblical Passage:

Universal Function Approximators

If $\sigma: \mathbb{R} \to \mathbb{R}$ non-constant, bounded, non-decreasing, and continuous, and let $S \subseteq \mathbb{R}^d$ be closed and bounded. Then for any continuous $f: S \to \mathbb{R}$ and any $\varepsilon > 0$, ∃ NN with 1 hidden layer with finitely many nodes, which we can write

$$h(x) = \sum_{j=1}^K c_j\sigma(\alpha_j + b_j^T x)$$

s.t. $\quad |h(x) - f(x)| < \varepsilon \quad \forall x \in S$

Training Neural Networks

In order to use GD and SGD, we need to be able to compute the gradient of the function we're trying to optimize.

With a NN, the function were trying to optimize is just the loss ℓ, that is defined in terms of our model $h(x)$ (the neural net function).

Computational Graphs

We assume our network is a DAG. This graph is also called Computational graph.

Key: each vertex v_j is the result of a differentiable computation (can be non-diff. at finitely many points like ReLUs)

Vertex ℓ is the loss function, also Weights are considered as vertices

Goal: efficiently calculate $\frac{\partial \ell}{\partial w_i}$ $\forall w_i \in W$

The primary mathematical tool employed in backpropagation in chain rule:

$$\frac{\partial \ell}{\partial w_i} = \sum_{v_j \in out(w_i)} \frac{\partial \ell}{\partial v_j}\frac{\partial v_j}{\partial w_i} = \sum_{v_j \in out(w_i)}\sum_{v_k \in out(v_j)} \frac{\partial \ell}{\partial v_k}\frac{\partial v_k}{\partial v_j}\frac{\partial v_j}{\partial w_i}$$

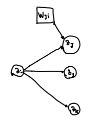
Main idea is to reuse components: many are the same!

Idea: use dynamic programming to compute the derivatives following the DAG dependencies: the main problem is computing $\nabla \ell(w)$ and the subproblems are $\frac{\partial \ell}{\partial w_i}$

→ Examples:

Fully connected layers

Each vertex calculates: $z_j = \sum_i w_{ji} a_i$



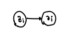
$$\therefore \frac{\partial z_j}{\partial w_{ji}} = a_i$$

$$\Rightarrow \frac{\partial \ell}{\partial w_{ji}} = \frac{\partial \ell}{\partial z_j} a_i$$

Also $\frac{\partial z_j}{\partial a_i} = w_{ji} \Rightarrow \frac{\partial \ell}{\partial a_i} = \sum_j^K \frac{\partial \ell}{\partial z_j} w_{ji}$

Element-wise nonlinearities

Let $a_i = \sigma(z_i)$



$$\Rightarrow \frac{\partial \ell}{\partial z_i} = \frac{\partial \ell}{\partial a_i}\frac{\partial a_i}{\partial z_i} = \frac{\partial \ell}{\partial a_i}\sigma'(z_i)$$

The starter code contains an implementation of linear regression for the MNIST dataset to classify 10 digits. Let $\mathbf{x}_i$ be the feature vector and $\mathbf{y}_i$ be a one-hot vector encoding of its class, i.e., $y_{i,j} = 1$ if and only if ith images is in class j and $y_{i,j} = 0$ if not. In order to use linear regression to solve a multi-class classification, we will use the *one-vs-all* strategy here. In particular, for each j we will fit a linear regression model $(\mathbf{w}_j, b_j)$ to minimize $\sum_i(\mathbf{w}_j^\top \mathbf{x}_i + b_j - y_{i,j})_2^2$. Then for any image $\mathbf{x}$, the prediction of its class will be $\arg\max_j(\mathbf{w}_j^\top\mathbf{x} + b_j)$.

Read the implementation and run it with batch size equal to 50, 100, and 200. **Attach the "epoch vs validation accuracy" plot. Report the running time for all the batch sizes. Explain the potential benefits and drawbacks of using small batch size, i.e., SGD vs GD.**

Solution:

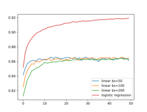


Figure 1: Epoch vs Accuracy

Figure 1 shows the plot. The running time for 50 epoches on my laptop is 51s, 28s, and 21s, respectively. The larger the batch size, the less time it takes to finish a constant number of epoches. The is because for larger batch size, there is less overhead due to the vectorization. On the other hand, when we use large batch size, we can see that it converges slower in practice. So, there is a tradeoff here.

CCA

$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ $Y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$

The goal is to find a subspace that maximizes the correlation between X, Y:

finding linear combination of x_i and y_j that have greatest correlation

Intuitively, this is similar to finding a linear relationship such that $YV \approx XU$

To measure correlation we use the Pearson Correlation Coefficient:

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)\,Var(Y)}}$$

→ Commutative

→ Invariant to linear transformations

Therefore, for zero mean X, Y: $Corr(X,Y) = \frac{\mathbb{E}[XY]}{\sqrt{\mathbb{E}[X^2]\,\mathbb{E}[Y^2]}} \approx \frac{\frac{1}{n}\sum_{i=1}^{n} x_i y_i}{\sqrt{\frac{1}{n}\sum x_i^2 \cdot \frac{1}{n}\sum y_i^2}} = \frac{x^T y}{x^T x\, y^T y}$

Neat fact: if X, Y are jointly Gaussian, their relationship can be entirely captured by $\rho(X,Y)$ (once normalized!)

$$\begin{bmatrix} \sigma_x^{-1} & 0 \\ 0 & \sigma_y^{-1} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \sigma_x^{-1} & 0 \\ 0 & \sigma_y^{-1} \end{bmatrix} \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} \begin{bmatrix} \sigma_x^{-1} & 0 \\ 0 & \sigma_y^{-1} \end{bmatrix}\right)$$

($\sigma_{xy} = \sigma_x \sigma_y \rho$)

$$\sim \mathcal{N}\left(0, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}\right)$$

Derivation

Let $X_{rv} \in \mathbb{R}^p$, $Y_{rv} \in \mathbb{R}^q$ be random vectors (note that usually we don't have random vectors but multiple instances of them). We want to find directions $u \in \mathbb{R}^p$, $v \in \mathbb{R}^q$ s.t. X, Y projected along these directions have max correlation:

$$\max_{u,v} \rho(X_{rv}^T u, Y_{rv}^T v) = \max_{u,v} \frac{Cov(X_{rv}^T u, Y_{rv}^T v)}{\sqrt{Var(X_{rv}^T u)\,Var(Y_{rv}^T v)}} = \max_{u,v} \frac{\mathbb{E}\left[u^T(X_{rv} - \mathbb{E}[X_{rv}])(Y_{rv} - \mathbb{E}[Y_{rv}])^T v\right]}{\sqrt{u^T Cov(X_{rv}) u \;\; v^T Cov(Y_{rv}) v}} = \max_{u,v} \frac{u^T Cov(X_{rv}, Y_{rv}) v}{\sqrt{u^T Cov(X_{rv}) u \;\; v^T Cov(Y_{rv}) v}}$$

We don't have the true covariances of these random vectors, but we can estimate them with our zero mean data $X \in \mathbb{R}^{n \times p}$, $Y \in \mathbb{R}^{n \times q}$

$$Cov(X_{rv}, Y_{rv}) = \mathbb{E}[X_{rv} Y_{rv}^T] \approx \frac{1}{n}\sum x_i y_i^T = X^T Y$$

$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ $\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

(...)

$$\max_{u,v} \rho(X_{rv}^T u, Y_{rv}^T v) \approx \max_{u,v} \rho(X^T u, Y^T v) = \max_{u,v} \frac{u^T X^T Y v}{\sqrt{u^T X^T X u\, v^T Y^T Y v}} = \max_{u,v} \frac{(Xu)^T (Yv)}{\sqrt{(Xu)^T Xu\, (Yv)^T Yv}}$$

We want to simplify the denom. by whitening X & Y s.t. $\tilde{X}^T \tilde{X} = I$

Note $X^T X = U_x \Lambda U_x^T$ ⇒ let $\tilde{u} = \underbrace{U_x \Lambda_x^{-\frac{1}{2}}}_{W_x} u$, $\tilde{v} = \underbrace{V_x \Lambda_x^{-\frac{1}{2}}}_{W_y} v$

$$\Rightarrow \max_{u,v} \frac{(Xu)^T (Yv)}{\sqrt{(Xu)^T Xu\, (Yv)^T Yv}} = \max_{\tilde{u},\tilde{v}} \frac{\tilde{u}^T \tilde{X}^T \tilde{Y} \tilde{v}}{\sqrt{\tilde{u}^T \tilde{X}^T \tilde{X} \tilde{u}\; \tilde{v}^T \tilde{Y}^T \tilde{Y} \tilde{v}}} = \max_{\tilde{u},\tilde{v}} \frac{u_w^T X_w^T Y_w v_w}{\sqrt{u_w^T u_w\, v_w^T v_w}}$$

why do this change of variables work?

Now we want to decorrelate X_w, Y_w ⇒ $\tilde{X}_w^T \tilde{Y}_w = D$ diagonal

$= U \Sigma V^T$ ⇒ let $u_d^T = u_w^T U$ ⇔ $u_d = U^T u_w$

and $v_d = V^T v_w$

$$\therefore \max_{u_w, v_w} \frac{u_w^T X_w^T Y_w v_w}{\sqrt{u_w^T u_w\, v_w^T v_w}} = \max_{u_w, v_w} \frac{u_w^T U \overbrace{U^T \Sigma V^T V}^{I} V^T v_w}{\sqrt{u_w^T U \overbrace{U^T u_w}^{I}\, v_w^T \overbrace{V V^T}^{I} v_w}} = \max_{u_d, v_d} \frac{u_d^T \Sigma v_d}{\sqrt{u_d^T u_d\, v_d^T v_d}} \overset{\text{WLOG}}{=} \max_{u_d, v_d} u_d^T \Sigma v_d$$

(true because correlation is invariant to scale!)

Optimal value for the first pair of canonical variables will just be setting $u_d = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$ $v_d = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$

as it corresponds to the largest singular value.

However now one has to recover the original u, v

$\Rightarrow$ $u = W_x u_w = W_x U u_d = W_x U e_1$ $\qquad v_w = \frac{1}{y}$

$v = W_y v_w = W_y V v_d = W_y V e_1$

More generally, for the best k directions, we choose: $U_d = \begin{bmatrix} I_k \\ 0_{p-k} \end{bmatrix}$ $V_d = \begin{bmatrix} I_k \\ 0_{q-k} \end{bmatrix}$ $\Rightarrow \begin{cases} U = W_x U_d \\ V = W_y V_d \end{cases}$

The columns of U & V will be linearly independent (as they are obtained by applying lin. matrices to orthogonal ones)

CCA vs PCA

A positive thing about CCA is that it's invariant to scalings and affine transformations. In particular, noise is just an affine transf.!

$Y = X + \varepsilon$

so CCA would not pick up on the noise while PCA would!

Note that both CCA and PCA can be considered as dimensionality reduction techniques, only that CCA is tangled at maximizing correlation (predictive power!)

CCA Regression

Once we've computed the CCA of X, Y now we can use it to predict X from Y or vice versa!

Note that Pearson Correlation maximizes linear correlation so we could expect X_{cca} Y_{cca} to be linearly related and want to find the matrix A that relates them.

$X_{cca} = XU$ $\qquad Y_{cca} = YV$

$$Y_{cca} \approx X_{cca} A \qquad \text{where } A = \left(X_c^T X_c\right)^{-1} X_c^T Y_c = \left(U^T X^T X U\right)^{-1} U^T X^T Y V$$

is this process symmetric in X & Y?

Therefore, to estimate $\tilde{Y}$ for a new $\tilde{X}$ we can do this:

$$\tilde{Y}_{cca} = \tilde{X}_{cca} A = \tilde{X} U A = \tilde{X} U \left(U^T X^T X U\right)^{-1} U^T X^T Y V$$

$\Rightarrow$ $\tilde{Y} =$

(e) Suppose you have a multi-variate regression problem, i.e. the feature matrix is $\mathbf{X} \in \mathbb{R}^{n \times p}$ and the regression target is $\mathbf{Y} \in \mathbb{R}^{n \times q}$ and $q > 1$. We know a prior that the number of regression targets is large and there are strong correlations between the multiple regression targets. For example, consider you have $n = 100$ samples. Each example has $p = 500$ features, and there are $q = 1000000$ regression targets.

There are two approaches you can solve the problem. The first approach is treat the multi-variate regression problem as q independent ridge regression problems. The second one is that first compute the CCA between $\mathbf{X}$ and $\mathbf{Y}$, which gives two projection matrices $\mathbf{U}_k$ and $\mathbf{V}_k$, then use q independent ridge regressions to fit $\mathbf{Y}_c \equiv \mathbf{Y}\mathbf{V}_k$ from $\mathbf{X}_c \equiv \mathbf{X}\mathbf{U}_k$, i.e. solve for $\mathbf{W}$ that satisfy $\mathbf{X}_c\mathbf{W} \approx \mathbf{Y}_c$. The final predictor is given by: $\mathbf{Y}_{predict} = \mathbf{X}(\mathbf{U}_k\mathbf{W}\mathbf{V}_k^{-1})$. What's the pros and cons of each approach?

Solution: The ridge regression is assuming that each of the fitting target is independent. On the other hand, the CCA approach has taken into account of the potential correlation among the fitting targets. Having taken advantage of the target correlation, CCA might have higher statistical efficiency than the set of independent ridge regressions. For example, the ridge regression use $n = 100$ examples to fit each target, with $p = 500$ dimensional features. It will results in severe overfitting in general. However, if we assume that the regression targets have large correlation with each other, conceptually we are using $nq = 10E8$ examples to fit each target, with the same number of features. Obviously CCA in this case will be much better than the first approach.

The independent ridge regression assumes identity Gaussian noise on each of the fitting target. On the other hand, the noise model for the CCA approach depends on the data, influenced via the computed matrices $\mathbf{U}_k$ and $\mathbf{V}_k$. It is conceptually much harder to make sense of this noise model and usually people use a noise model that does not depend on the data.

Let us now perform CCA on the two random vectors $\mathbf{X}$ and $\mathbf{Y}$. Recall that in order to find the first canonical directions, we look for vectors $\mathbf{u} \in \mathbb{R}^d$ and $\mathbf{v} \in \mathbb{R}^2$ such that $\rho(\mathbf{u}^\top \mathbf{X}, \mathbf{v}^\top \mathbf{Y})$ is maximized.

Show that the maximizing $\mathbf{u}^*$ is proportional to $\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$. Recall that $\mathbf{u}^*$ is that "direction" of $\mathbf{X}$ that contributes most to predicting $\mathbf{Y}$. **What is the relationship between $\mathbf{u}^*$ and the function $f(\mathbf{X})$ computed in part (a)?**

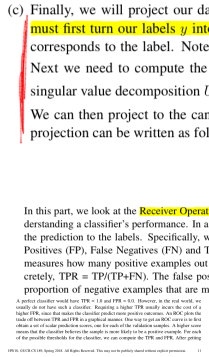
notice now that $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \mathbf{\Sigma}^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ is a scalar β. We also have $\mathbf{u}^* \propto \mathbf{\Sigma}_{XX}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \frac{1}{4+\beta}\mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\beta$.

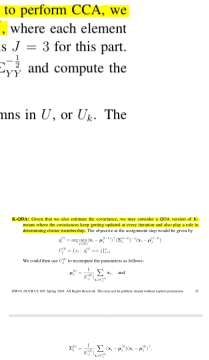
In other words, we have $\mathbf{u}^* = \gamma \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ for the component of $\mathbf{X}$ that has the highest predictive value for $\mathbf{Y}$. Comparing with the decision function $f(\mathbf{X}) = (\mathbf{X} - \frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2})^T \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, we see that this is the same direction predicted by LDA in part (a).

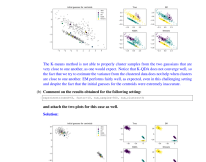
(c) Finally, we will project our data into the Canonical Variates. In order to perform CCA, we must first turn our labels y into a one-hot encoding vector $\vec{y} \in \{0,1\}^J$, where each element corresponds to the label. Note J is the number of class labels, which is $J = 3$ for this part. Next we need to compute the canonical correlation matrix $\Sigma_{XX}^{-\frac{1}{2}} \Sigma_{XY} \Sigma_{YY}^{-\frac{1}{2}}$ and compute the singular value decomposition $U\Lambda V^T = \Sigma_{XX}^{-\frac{1}{2}} \Sigma_{XY} \Sigma_{YY}^{-\frac{1}{2}}$.

We can then project to the canonical variates by using the first k columns in U, or U_k. The projection can be written as follows:

Concise CCA







Generative vs Discriminative Classification

When dealing with the problem of classifying points in discrete categories, there are 2 main approaches:

- Generative:

We model how the data was generated for each class k:
$$\begin{cases} f_k(x) = \mathbb{P}(X \mid Y = k) & \text{distrib. for class } k \\ \mathbb{P}(Y = k) & \text{prior prob. for each class} \end{cases}$$

(total of $k+1$ probability distributions (1 prior $+ k$ for classes) — each one $\in \mathbb{R}^d$ if $x_i \in \mathbb{R}^d$)

The idea is to assign the label with highest likelihood:
$$\hat{Y}_{pred} = \arg\max_k \mathbb{P}(X \mid k) = \arg\max_k \frac{f_k(x)\mathbb{P}(Y=k)}{\mathbb{P}(X=x)} = \arg\max_k \frac{f_k(x)\mathbb{P}(Y=k)}{\sum_{\tilde{k}} f_{\tilde{k}}(x)\mathbb{P}(Y=\tilde{k})} = \arg\max_k f_k(x)\mathbb{P}(Y=k)$$

(we can do this but don't need it here)

Pros:
- Can generate new samples (data augmentation)
- Quick to train

Cons:
- Inefficiency
- Num. Params: $\frac{d(d+1)}{2} + O(d^2)$ for cov. matrix $\rightarrow O(kd^2)$ if k classes

- Discriminative:

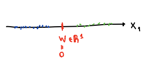
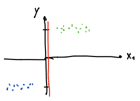
model only $Y \mid X$ or not even $\rightarrow$ just have a boundary of fn f s.t. $f(x) = \hat{Y}$ ← estimated

└ $O(d)$ params!

Least Squares SVM

An initial intuitive approach to classify points in a discriminative setting would be to try and fit a normal linear regression line between our classes (as a boundary) by setting $Y=-1$, $Y=1$.

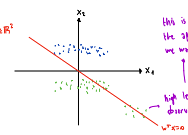
1D Case

Instead of considering w^Tx as our prediction, we can consider
$$\text{sign}(y_i\, w^Tx_i) = \begin{cases} 1 & \text{if } \text{sign}(w^Tx_i) = y_i \Rightarrow \begin{cases} w^Tx_i > 0 & \text{if } y_i = 1 \\ w^Tx_i < 0 & \text{if } y_i = -1 \end{cases} \\ -1 & \text{if } \text{sign}(w^Tx_i) \neq y_i \text{ (misclassified)} \end{cases}$$

What does this mean geometrically? It means that we are trying to find a hyperplane (defined by w) such that all points with $y_i=+1$ lie on one side and vice versa.

Note that if we don't include a constant, the hyperplane must pass through the origin.



this is not the approach we want — high leverage observations

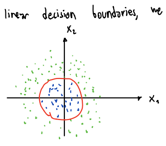
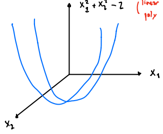
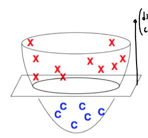
The corresponding loss function is: $\arg\min_{w} \sum_i^n (y_i - \text{sign}(w^Tx_i))^2 + \lambda \|w'\|_2^2$ → NP-Hard! $\Rightarrow$ (simplified (approximated)) $\arg\min_{w \in \mathbb{R}^{d+1}} \sum_i^n (y_i - w^Tx_i)^2 + \lambda\|w'\|_2^2$

($w \in \mathbb{R}^{d+1}$ ← bias; w' without bias term; leads to this)

(now we are penalizing distance from label in a weird way!)

Feature extension

To get non-linear decision boundaries, we can augment our features (polynomials, etc) in order to make our data more separable: $x_2^2 + x_1^2 - 2$ (linear comb. of poly features)

(distance from center)$^2 - c$

In this case, our loss becomes:
$$\arg\min_{w_k} \sum_i^n (y_i - \phi(x_i)^T w_k)^2 + \lambda\|w\|_2^2$$

Neural Network

Technically one could even use neural networks, in order to optimize over the best set of features:
$$\arg\min_{w_k} \sum_i^n (y_i - g_w(x_i))^2 + \lambda\|w\|_2^2$$

(To classify: (after fit)) $\hat{y}_i = \begin{cases} 1 & \text{if } g_w(x_i) > \theta \\ -1 & \text{if } g_w(x) < \theta \end{cases}$ for θ some threshold

(the power of NNs is that we can take derivatives through this even though non-linear and very complex)

Multiclass Extension

One could use the same framework for multiclass classification by using one-hot encoding. In this case we would have $y_i \in \mathbb{R}^K$ and $W \in \mathbb{R}^{K \times d+1}$ s.t. $Wx_i \in \mathbb{R}^K$ represents our "score" for each class

Loss Fn: $\arg\min_{W \in \mathbb{R}^{K \times (d+1)}} \sum_i^n \|y_i - Wx_i\|_2^2 + \lambda\|W\|_F$

(note that we are penalizing any "score" for classes that are not right)

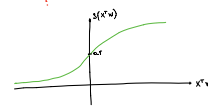
Logistic Regression

Discriminative classification method that assigns probabilities for each class (has a probabilistic interpretation, unlike SVM)

In logistic classification we use 0/1 labels.

We use the score x_i^Tw and transform it to a probability with the sigmoid (link function?) $s(z) = \frac{1}{1+e^{-z}}$

We let our prediction be
$$\hat{Y} = \arg\max_k \mathbb{P}(\hat{Y} = k \mid X = x_i) = \begin{cases} 1 & \text{if } s(x_i^Tw) > 0.5 \Leftrightarrow x_i^Tw > 0 \\ 0 & \text{else} \end{cases}$$



As a loss function, we will choose cross-entropy, as
$$L(w) = -\sum_i^n y_i \ln p_i + (1-y_i)\ln(1-p_i) = -\ln p_{y_i}$$
→ only way to have zero loss is $p_i = y_i$

Consider the classes as coming from a Bernoulli: $\hat{Y}_i \mid X_i \sim \text{Bern}(p_i)$
$$\mathbb{P}(Y_i = y_i) = p_i^{y_i}(1-p_i)^{(1-y_i)} \quad \text{where } p_i = \frac{1}{1+e^{-x_i^Tw}}$$

$$\Rightarrow \hat{w}_{LR} = \arg\max_w \mathbb{P}(Y_1 = y_1, \ldots, Y_n = y_n \mid x_1, \ldots, x_n, w)$$
$$\vdots$$
$$= \arg\min_w L(w) \rightarrow \text{Cross Entropy}$$

∴ it is wise to choose cross entropy as loss

Idea: conditional on X, each point has a prob. distribution $\hat{Y}_i \mid x_i \sim \text{Bern}(p_i)$ where $p_i = \frac{1}{1+e^{-x_i^Tw}}$

Loss? LS $\sum (y_i - s(x_i^Tw))^2 + \lambda\|w\|_2^2$ but no justification.

Turns out KL Divergence / MLE → Cross Entropy

$$L(w) = -\sum_i y_i \ln p_i + (1-y_i)\ln(1-p_i)$$
$$= \begin{cases} -\ln p_i = \frac{1}{s(x_i)} & \text{if } Y=1 \\ -\ln(1-p_i) = \frac{1}{1-s(x_i)} & \text{if } Y=0 \end{cases}$$

Information Theory Justification

KL-Divergence $= D_{KL}(P \| Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$ (not a true distance (not symmetric), always positive (0 iff $P=Q$))

$$\hat{w}_{LR} = \arg\min_w \sum_{i=1}^n D_{KL}(Y_i \| \hat{Y}_i)$$
$$= \arg\min_w \sum_{i=1}^n y_i \ln\frac{y_i}{p_i} + (1-y_i)\ln\frac{(1-y_i)}{(1-p_i)}$$
$$\vdots$$
$$= \arg\min_w -\sum_{i=1}^n y_i \ln p_i + (1-y_i)\ln(1-p_i)$$

$H(Y_i, \hat{Y}_i)$ Summing the "cross-entropies" $H(Y_i, \hat{Y}_i)$

$D_{KL}(Y_i \| \hat{Y}_i) = H(Y_i, \hat{Y}_i) - H(Y_i)$ (cross entropy; entropy of Y_i)

Cross-entropy between 2 distbns is # bits necessary to identify that coming from q rather than "true" distribution p

what are q and p in this case? why not symmetric?

→ $H(p,q) = -\sum_x p(x) \ln q(x)$

Multiclass Logistic Regression

In multiclass logistic we model the distribution of labels for each point to be a multinomial

$$\hat{Y} \mid x \sim \text{Multi}(\underbrace{p_1, \ldots, p_k}_{\hat{p}}) \qquad W \in \mathbb{R}^{K \times (d+1)}$$

where $\hat{p}_x = \overset{\sigma}{\text{softmax}}(Wx)$ (we have a weight for each class)
$$= [\sigma(w_1^Tx)_1 \quad \cdots \quad \sigma(w_K^Tx)_K]$$
→ softmax distribution (W, x) (probabilities sum up to one)

and $\hat{Y} = \arg\max_k \sigma(Wx)$

σ softmax fn: $\sigma(\hat{z})_j = \frac{e^{z_j}}{\sum_{i=1}^K e^{z_i}}$

(can the softmax be thought of as an operation that provides prob. for our multinomial estimate)

logistic $\subseteq$ softmax: (the logistic distribution is just a special case ($k=2$) of the softmax one)

$$\mathbb{P}(\hat{Y}_i = 1 \mid x_i, W) = \sigma(Wx_i)_1 = \frac{e^{w_1^Tx_i}}{e^{w_1^Tx_i} + e^{w_0^Tx_i}} = \frac{1}{1+e^{-(w_1-w_0)^Tx_i}} = s\left((w_1-w_0)^Tx_i\right)$$

$$\mathbb{P}(\hat{Y}_i = 0 \mid x_i, W) = \sigma(Wx_i)_0 = \frac{e^{w_0^Tx_i}}{\ldots} = 1 - s\left((w_1-w_0)^Tx_i\right)$$

The loss function can be justified as before with MLE / KL divergence (as reducing sum of cross-entropy between all pairs of predicted vs true distributions)

$$L(w) = -\sum_{i=1}^n \sum_{j=1}^K \delta_{j,y_i} \log \mathbb{P}(\hat{Y}_i = j \mid x_i, W)$$
$$= \sum_{i=1}^n H(Y_i, \hat{Y}_i)$$

(indicator $=1$ if predicted value for x_i is j)

Information Approach: We have an estimated and a "true" "distribution" for each training point and we want to minimize the sum of the cross entropies between them

Training

Logistic regression has no known closed form solution to minimize the loss. Therefore, one must minimize it by using GD or other stochastic forms.

GD stuff.

Assume your data comes from two classes and the prior for class k is $p(y=k) = \pi_k$. Also the conditional probability distribution for each class k is Gaussian, $\mathbf{x}|y = k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, that is $f_k(\mathbf{x}) = f(\mathbf{x}|y=k) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}\exp\left\{(\mathbf{x}-\boldsymbol{\mu}_k)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\}$. Assume that $\{\boldsymbol{\mu}_k\}_{k=0}^1, \boldsymbol{\Sigma}$ where estimated from the training data.

Show that $P(y|\mathbf{x}) = s(\mathbf{w}^\top\mathbf{x})$ is the sigmoid function, where $s(\zeta) = \frac{1}{1+e^{-\zeta}}$.

Solution: Let us denote $Q_k(\mathbf{x}) = \log\left((\sqrt{2\pi})^d\pi_k f_k(\mathbf{x})\right)$, so we get that

$$p(y=1|\mathbf{x}) = \frac{\pi_1 f_1(\mathbf{x})}{\pi_0 f_0(\mathbf{x}) + \pi_1 f_1(\mathbf{x})} = \left(1 + \frac{\pi_0 f_0(\mathbf{x})}{\pi_1 f_1(\mathbf{x})}\right)^{-1} = \left(1 + \frac{e^{Q_0(\mathbf{x})}}{e^{Q_1(\mathbf{x})}}\right)^{-1} = s(Q_1(\mathbf{x}) - Q_0(\mathbf{x}))$$

Now lets look at the expression $Q_1(\mathbf{x}) - Q_0(\mathbf{x})$

$$Q_1(\mathbf{x}) - Q_0(\mathbf{x}) = \log\left((\sqrt{2\pi})^d\pi_1 f_1(\mathbf{x})\right) - \log\left((\sqrt{2\pi})^d\pi_0 f_0(\mathbf{x})\right)$$
$$= \log\frac{\pi_1}{1-\pi_1} - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_0)^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_0)$$
$$= \log\frac{\pi_1}{1-\pi_1} + \mathbf{x}^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1-\boldsymbol{\mu}_0) - \frac{1}{2}\boldsymbol{\mu}_1^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0$$

Notice that we can write it out as:

$$Q_1(\mathbf{x}) - Q_0(\mathbf{x}) = \log\frac{\pi_1}{1-\pi_1} + \frac{1}{2}\boldsymbol{\mu}_0^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 - \frac{1}{2}\boldsymbol{\mu}_1^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + (\boldsymbol{\mu}_1-\boldsymbol{\mu}_0)^\top\boldsymbol{\Sigma}^{-1}\mathbf{x} = w_0 + \mathbf{w}^\top\mathbf{x}$$

In the previous part we saw that the posterior probability for each class is the sigmoid function under the LDA model assumptions. Notice that LDA is a generative model. In this part we are going to look at the discriminative model. We will assume that the posterior probability has Bernoulli distribution and the probability for each class is the sigmoid function, i.e. $p(y|\mathbf{x};\mathbf{w}) = q^y(1-q)^{1-y}$, where $q = s(\mathbf{w}^\top\mathbf{x})$ and try to find $\mathbf{w}$ that maximizes the likelihood function. **Can you find a closed form maximum-likelihood estimation of w?**

Basically logistic is assuming that the classes come from various normals (similar to LDA) same cov

Can't get a closed form $X^T(s - y) = 0$ ($s_i = s(w^Tx_i)$)

HW 9 Last Problem ⚠

Newton's Method Disc. 9

QDA & LDA

QDA & LDA are generative models for classification, that assume that the various classes of our data were generated by gaussian RVs. Given that we don't actually know the means or covariances of these $\mathcal{N}$s, we estimate them from the data:

$$\hat{\mu}_K = \frac{1}{n_K}\sum_{i \in K} x_i \qquad \hat{\Sigma}_K = \frac{1}{n_K - 1}\sum_{i \in K}(x_i - \hat{\mu}_K)(x_i - \hat{\mu}_K)^T$$

To predict which class a point belongs to, we just pick the maximum probability:

$$\hat{y} = \arg\max_K \mathbb{P}(K \mid x) = \arg\max_K \mathbb{P}(x \mid K)\,\mathbb{P}(K)$$

prior! → $\hat{\mathbb{P}}(K) = \frac{n_K}{n}$ easy!

$$= \arg\max_K \ln \mathbb{P}(x \mid K) + \ln \mathbb{P}(K)$$

$$= \arg\max_K \underbrace{\ln \mathcal{N}(\hat{\mu}_K, \hat{\Sigma}_K) + \ln \mathbb{P}(K)}_{Q_K(x)}$$

LDA

For LDA we only use one covariance matrix

$$\hat{\Sigma} = \frac{1}{n-1}\sum_i^n (x_i - \bar{x})(x_i - \bar{x})^T = \sum_{K=1}^{K} \frac{n_K}{n-1}\hat{\Sigma}_K$$

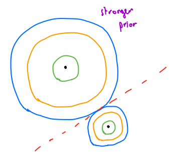
can think of this as a weighted average of estimated within class covariances

LDA vs. QDA

LDA
- has linear boundaries

QDA
- has quadratic boundaries

Priors change the size of the level sets



Isotropic → spherical covariance

Antisotropic → elliptical covariance → can be reduced to isotropic w. change of vars. — in LDA this ends up resulting in a linear boundary again

LDA, same prior:

→ classify by just which mean the new datapoint is closest to

boundary is when equidistant from both:

$$Q_A(x) = Q_B(x) \;\Downarrow\; \|x - \mu_A\|_2^2 = \|x - \mu_B\|_2^2$$

LDA, different prior:

$$Q_A(x) = Q_B(x)$$
$$\Downarrow$$
$$x^T\left(\hat{\Sigma}^{-1}(\hat{\mu}_A - \hat{\mu}_B)\right) + \left(\ln\left(\frac{\mathbb{P}(A)}{\mathbb{P}(B)}\right) - \frac{\hat{\mu}_A^T\hat{\mu}_A - \hat{\mu}_B^T\hat{\mu}_B}{2}\right) = 0$$

(priors)

$$x^T w + k = 0$$

boundary is still a linear function

normal boundary vector: $\Sigma_C^{-1}(\mu_C - \mu_D)$

QDA, different priors:

we end up getting

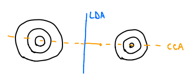
$$Q_A(x) = Q_B(x)$$
$$\Downarrow$$
$$\ln \mathbb{P}(A) - \tfrac{1}{2}(x - \hat{\mu}_A)^T \hat{\Sigma}_A (x - \hat{\mu}_B) = \ln \mathbb{P}(B) - \tfrac{1}{2}(x - \hat{\mu}_B)^T \hat{\Sigma}_B (x - \hat{\mu}_B)$$

unlike prev. cases, cannot cancel out the quadratic terms, so we are left with a quadratic boundary

From HW 9:

LDA and CCA are very similar: CCA on Y one-hot gives same result!

→ The predicted direction of max correlation (pred. power) for CCA is the same as the normal vector to the decision boundary!



(g) Compare and contrast k-means, soft k-means, and mixture of Gaussians fit with EM.

Solution: For k-means, we implicitly assume clusters are spherical and so this doesn't work for complex geometrical shaped data. Additionally, it uses hard assignment, meaning the $q_{i,1}$ probabilities are 0 or 1. This can be easier to interpret, but doesn't incorporate information from all data points to update each centroid. K-means will also usually have trouble with clusters that have large overlap (see Figure 2)

For soft k-means and EM we have soft assignments. For soft k-means, the weighted mean amounts to

$$r_{i,1} = \frac{\exp\{-B\|x_i - \mu_1\|^2\}}{\exp\{-B\|x_i - \mu_1\|^2\} + \exp\{-B\|x_i - \mu_2\|^2\}}$$
$$r_{i,2} = \frac{\exp\{-B\|x_i - \mu_2\|^2\}}{\exp\{-B\|x_i - \mu_1\|^2\} + \exp\{-B\|x_i - \mu_2\|^2\}}$$

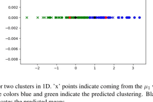


Figure 2: K-means for two clusters in 1D. 'x' points indicate coming from the μ_1 while 'o' indicates points coming from μ_2. The colors blue and green indicate the predicted clustering. Black dots indicate the true means, while red indicates the predicted means.

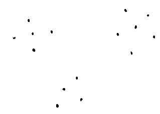
$$\mu_1^{t+1} = \frac{\sum_{i=1}^n r_{i,1}^{t+1} x_i}{\sum_{i=1}^n r_{i,1}^{t+1}} \qquad \mu_2^{t+1} = \frac{\sum_{i=1}^n r_{i,2}^{t+1} x_i}{\sum_{i=1}^n r_{i,2}^{t+1}}$$

where we have a stiffness parameter β, which can be intrepreted as the inverse variance. In cases where the clusters have different geometry, one might resort to EM. Note that EM is not unrelated to LDA/QDA. The setup is similar in that we probablistically determine the probabilities of coming from cluster k, but LDA/QDA does hard classification, EM probabilistic performs soft classification.

Clustering

Unsupervised learning methods.

Generally want: low inter-cluster similarity; high intra-cluster similarity — similarity can be thought of as a measure of "distance"



K-Means Clustering

We want to cluster points together, based on distance. We pick k, the number of classes, as a hyperparameter.

The loss we try to minimize is

$$\arg\min_{c_K, C_K} \sum_K^K \sum_{i \in C_K} \|x_i - c_K\|_2^2$$

(c_K: centroid positions; C_K: cluster assignments)

This problem doesn't have an efficient optimal solution (as combinatorial number of possible partitions K^n)

However, we can approach it greedily with an iterative method:

0) Pick random labels
1) estimate centroids $\vec{c}_K = \arg\min_{\vec{c}_K} \sum_{i \in C_K} \|x_i - \vec{c}_K\|^2 = \frac{1}{|C_K|}\sum_{i \in C_K} x_i \quad \forall K$
2) pick labels (class) $\hat{y}_i = \arg\min_K \|x_i - c_K\|_2^2$

Loyd's Algorithm

Guaranteed to converge:
1) Only finitely many (M) possible centroid pairs
2) Each update cannot increase obj. value

In practice we run with multiple initializations

Problems with simple K-means:
- no probabilistic interpretation: just hard labels (hard assignment!)
- each feature equally important → related to spherical gaussians

Soft K-means

Soft rather than hard assignment

What we want to do is:

1) estimate centroids: intuitively $\vec{c}_K = \sum_i^N \mathbb{P}(K \mid x_i)\, x_i$ — we don't have probabilities!

How do we fix this?

Instead of probabilities, let's deal with a score! What metrics do we have that can serve as "score"? Distance itself!

$$\Rightarrow \text{Let } \vec{z}(x_i) = \begin{bmatrix} \vdots \\ -\beta\|x_i - c_K\|^2 \\ \vdots \end{bmatrix} \forall K$$

be the "score" vector: scores are lower the further away points are! (larger dist → lower score!; β: softness parameter)

To transfer this to probabilities, we can just use the softmax function: $r_K(x_i) = \sigma_K(\vec{z}(x_i))$ ($r_K(x_i)$: prob. of x_i of being class K; σ_K: Kth entry)

∴ now centroids will be computed as

$$\vec{c}_K = \arg\min_{c_K} \sum_{i=1}^N r_K(x_i)\|x_i - c_K\|^2 = \frac{\sum_i^N r_K(x_i)\, x_i}{\sum_i^N r_K(x_i)}$$

(probabilities as weights → how important is this point?) → weighted average

Labels do not have a hard assignment anymore → we can just look at $r(x_i)$ as a vector of probabilities for point x_i for each class!

Problems with soft K-means:
→ how to choose β?
→ clusters still spherical!

$\beta \uparrow$: distance is very penalized — enlarges differences between points → very skewed probs (1 → closest, 0 → all others)

$\beta \downarrow$: distance isn't penalized much → more regularization → everything has same probability as $\beta \to 0$

Mixture of Gaussians

A way to make clustering fully probabilistic: we consider each class to have been generated by it's own gaussian $\mathcal{N}(\mu_K, \Sigma_K)$. However we don't know the parameters for these gaussians or the labels (which gausian generated which datapoint), so we somehow try to estimate both: moreover, we are missing also the prior on z, the classes.

z is our latent variable: we can't observe it directly from the data, but we can try to infer things about it

Each cluster is a generative model

The likelihood for a datapoint under this model is

$$\mathbb{P}(x_i \mid \vec{\theta}) = \sum_{K=1}^K \mathbb{P}(x_i, z = K \mid \vec{\theta}) = \sum_{K=1}^K \mathbb{P}(x_i \mid z = K)\,\mathbb{P}(z = K)$$

$\mathcal{L}$ for all datapoints:

$$\mathcal{L}(x_1, \dots, x_n \mid \theta) = \prod_i^n \mathcal{L}(x_i \mid \theta) = \prod_i^n \sum_{K=1}^K \mathbb{P}(x_i \mid z = K)\,\mathbb{P}(z = K)$$

$$\Rightarrow \ell(x_1, \dots, x_n \mid \theta) = \sum_i^n \log \sum_{K=1}^K P(x_i \mid z = K)\,\mathbb{P}(z = K)$$

can't just solve for this → can't MLE purely → EM! coupled params inside the log!

when performing QDA we know (or at least, we can estimate) $\mathbb{P}(z=K)$ from the data! (proportion of K)

EM - Expectation Maximization

A method of dealing with latent variable models and fitting them to data.

Based on the idea of complete log likelihood (likelihood of data treating z_i, the latent variable, as fixed) → $L_c(x_i, z_i \mid \theta) = \log p(x_i, z_i \mid \theta)$

↳ we can consider all values of z_i by taking the expected complete log likelihood → $\mathbb{E}_q[L_c(x_i; z_i \mid \theta)] = \sum_K q(z = K \mid x_i)\log p(x_i; z_i \mid \theta)$

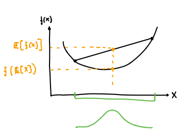
where q is a fixed estimate of the TRUE distribution of the latent variable $p(z \mid x)$

Another idea EM makes use of is Jensen's Inequality:

if f is convex

$$f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$$

(convex combination; order)



All distributions are "over" something. Assigning value to each point in space:

MVN → $\mathbb{R}^n$ space

Binomial → $\mathbb{R}^1$ space → prob. of 3 Heads, 2 Heads etc.

⇕

this works because there is an isomorphism between the domain of these functions and our geometries

The main idea is update q and maximize expected log likelihood iteratively:

log likelihood for x_i datapoint is:

$$\ell_i(\theta, x_i) = \log p(x_i; \theta) = \log \sum_K^K p(x_i, z = K \mid \theta)$$
$$= \log \sum_K^K \frac{q(z = K \mid x_i)\, p(x_i, z = K \mid \theta)}{q(z = K \mid x_i)}$$
$$= \log \mathbb{E}_q\left[\frac{p}{q}\right]$$
$$\geq \mathbb{E}_q\left[\log \frac{p}{q}\right] = \sum_K q \log \frac{p}{q} = \underbrace{-\sum_K q \log q}_{H(q)} + \underbrace{\sum_K q \log p}_{\mathbb{E}_q[\log p] = \mathbb{E}_q[L_c]}$$
$$= F_i(q; \theta)$$

∴ we have found in terms of q a quantity that we can maximize that is upper bounded by the log likelihood.

$$\Rightarrow \ell(\theta, \vec{x}) \geq \sum_i^n F_i(q_i \mid \theta) = \vec{F}(\vec{q} \mid \theta)$$

Algorithm: the final version of the algorithm will therefore be:

1) Initialize θ^0
2) Expectation step (E step) (soft imputation): find q^{t+1} that maximizes $F(q^{t+1}, \theta)$ (our lower bound on likelihood) based on our θ estimate
   i.e. $q_i^{t+1}(z_i = K \mid x_i) = p(z_i = K \mid \theta^t, x_i) \quad \forall i, \forall K$ — $O(nK)$ values to update
3) Maximization step (M step): find θ^{t+1} that maximizes $F(q^{t+1}, \theta)$ (lower bound on likelihood) based on our q estimate
   $$\theta^{t+1} = \arg\max_\theta F(q^{t+1}, \theta) = \arg\max_\theta \mathbb{E}_q[L_c(\vec{x}, z \mid \theta)]$$
   (as entropy doesn't depend on θ)
4) Repeat until convergence

	K-means	soft-K-means	EM
(E) q	assignment to closest centroid →	soft assignment based on softmax distance-β →	soft $p(\theta)$ based on fully prob. model!
(M) θ	update centroid loc →	// soft →	θ optimal.

EM for MoG

(E) step: estimate q

$$q^{t+1}(z = K \mid x_i) = p(z = K \mid x_i, \theta^t) = \frac{p(z = K, x_i \mid \theta^t)}{p(x_i \mid \theta^t)} = \frac{p(x_i \mid z = K, \theta^t)\, p(z = K \mid \theta^t)}{\sum_K p(x_i \mid K, \theta^t)\, P(K)} = \frac{\alpha_K\, P(x_i \mid z = K, \theta^t)}{\sum_K \alpha_K\, P(x_i \mid K, \theta^t)}$$

($p(z=K \mid \theta^t) = \alpha_K$: prior for each class; $P(x_i \mid z = K, \theta^t)$: $\mathcal{N}(\mu_K, \Sigma_K)$)

(M) step: optimize $\theta = (\vec{\mu}, \vec{\Sigma}, \vec{\alpha})$ (each indexed by K)

$$\mu_K^{t+1} = \frac{\sum_i^n q^t(z = K \mid x_i)\, x_i}{\sum_i^n q^t(q = K \mid x_i)}$$

$$\Sigma_K^{t+1} = \frac{\sum_i^n q_{iK}^t (x_i - \mu_K^{t+1})(x_i - \mu_K^{t+1})^T}{\sum_i^n q_{iK}^t}$$

$$\alpha_K = \frac{1}{N}\sum_i^N q_{iK}^t$$

↳ "what fraction of each datapoint" is in class K
↳ this guarantees $\sum_K \alpha_K = 1$

(derivations)

Loss functions in classification: Generalization & Stability

We tried to classify points with Least Squares SVM, by using the loss

$$L(w) = \sum_i^n (y_i - x_i^T w)^2 + \lambda \|w\|_2^2 \qquad (y \in \{-1, 1\})$$

However, we saw that this loss doesn't quite cut it, as it behaves in undesireable ways.

A general form of objective function in classification is:

$$\min_{w,b} \frac{1}{n} \sum_i^n L(y, x^T w - b) + \lambda \|w\|_2^2$$

What we would optimally want to minimize in classification is the L_{step} step loss (in place of the loss above)

$$L_{STEP}(y, w^T x - b) = \begin{cases} 1 & y(w^T x - b) < 0 \quad \text{(missclassified)} \\ 0 & y(w^T x - b) \geq 0 \quad \text{(classified correctly)} \end{cases}$$

However this loss is not differentiable or convex.

Another loss function we saw is the logistic loss (where $y \in \{0, 1\}$)

where $\text{logit}(w^T x - b) = \sigma(w^T x - b) = \frac{e^{w^T x - b}}{1 + e^{w^T x - b}} = \frac{1}{1 + e^{-(w^T x - b)}}$

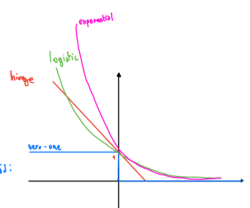
$$L_{LR}(y, w^T x - b) = y \ln \frac{1}{s(w^T x - b)} + (1 - y) \ln \left(\frac{1}{1 - s(w^T x - b)} \right)$$

(logistic)

leading to the optimization problem:

$$\min_{w,b} \frac{1}{n} \sum_i^n L_{LR}(y, w^T x - b)$$

(works well without regularization)



The hinge loss function is a convex and differentiable variant of the step loss:

$$L_{hinge}(y, w^T x - b) = \max \left[0, 1 - y(w^T x - b) \right]$$

$$\Rightarrow \min_{w,b} \frac{1}{n} \sum_i^n L_{HINGE}(y, w^T x - b) + \lambda \|w'\|_2^2$$

→ penalizes choices of w for which margin is small !

SVM Loss

[why is intercept so different from all other axes ? what makes it different ? Affine vs. linear i think]

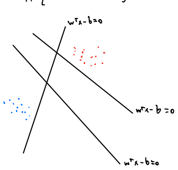
SVM It turns out that using the hinge loss instead of LS loss has much more desireable properties. Given that it doesn't have the square term and doesn't penalize confident correct predictions (which LS-SVM does), it will actually separate the data if it is separable.

This is similar to the perceptron algorithm. However, the perceptron
- fails if data is not linearly separable
- cannot distinguish between two solutions

goal of perc. alg.: find w, b s.t. $\begin{cases} w^T x_i - b \geq 0 & \text{if } y_i = 1 \\ w^T x_i - b < 0 & \text{if } y_i = -1 \end{cases} \Leftrightarrow y_i (w^T x_i - b) \geq 0 \quad \forall i$

} equivalent to finding a $d-1$ dimensional hyperplane that separates points.

$H = \{x \mid w^T x - b = 0\}$

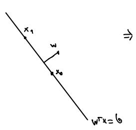


3 equally valid perceptron solutions

Hard - Margin SVMs

Only work on linearly separable data, but try to maximize the margin (min. distance of boundary to datapoints)

We know that
- all points classified as +1 are on the ≥ side of the hyperplane
- " " " " −1 " " " " ≤ " " " "
} and distance from it is GREATER than margin
- margin is non-negative



$\Rightarrow (x_1 - x_0)^T w = x_1^T w - x_0^T w = 0 - 0 = 0$

$\therefore (x_1 - x_0) \perp w$

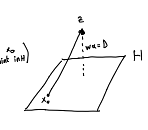
$\therefore H \perp w$ as x_1, x_0 are any two points on H

} $\therefore$ the distance from H will be a multiple of w for any point z

$\therefore D_{z \text{ to } H} =$ [size of] projection of $z - x_0$ onto w ! (where x_0 any point in H)

$= \frac{|(z - x_0)^T w|}{\|w\|_2}$

$= \frac{|z^T w - x_0^T w|}{\|w\|_2} = \frac{|z^T w - b|}{\|w\|_2}$ (with $x_0^T w = b$)



$\therefore$ the distance of any training point x_i from the hyperplane is $\frac{|x_i^T w - b|}{\|w\|_2}$

$\therefore$ Assuming the margin has size m, we know that $\frac{|x_i^T w - b|}{\|w\|_2} \geq m \quad \forall i \Leftarrow \frac{(x_i^T w - b) y_i}{\|w\|_2} \geq m \quad \forall i$

(assumes data is linearly separable → this will always be positive)

this form enforces separation (+1 points on pos. side, −1 points on neg. side) — this one didn't !

$\therefore$ we get that our opt. problem will be:

$$\max_{m,w,b} m \qquad m \geq 0, \quad \frac{(x_i^T w - b) y_i}{\|w\|_2} \geq m \quad \forall i$$

} non-unique solution: if w, b, m is a solution so is $\alpha w, \alpha b, m$

To make the solution unique we add the constraint $m = \frac{1}{\|w\|_2}$ so we get

$$\max_{w,b} \frac{1}{\|w\|_2} \qquad \frac{(x_i^T w - b) y_i}{\|w\|_2} \geq \frac{1}{\|w\|_2} \Leftrightarrow (x_i^T x - b) y_i - 1 \geq 0 \quad \forall i$$

↑ we are penalizing any configuration that doesn't maximize margin

→ it is NOT penalising "kind of unsure" points as much as it is enforcing the max margin constraint

⇕

Standard form: $\min_{w,b} \|w\|_2^2 \quad$ s.t. $(x_i^T x - b) y_i - 1 \geq 0 \quad \forall i$

Soft Margin SVMs

Problem with hard margin is that enforces correct classification for all points. This is not always possible! Solution: add slack variables to relax the constraint and then penalize missclassifications by how much they are wrong.

Our constraints therefore become $\begin{cases} (x_i^T w - b) y_i \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0 \end{cases}$

→ we add this constraint in order for the ξ_is not to be able to reduce the loss for any point beyond the margin

How can we penalize the ξ_is?

We modify the loss function to take them into account, adding a hyperparameter C (how much we want to penalize missclassifications)

$$\min_{w,b,\xi_i} \frac{1}{2} \|w\|^2 + C \sum_i^n \xi_i \qquad \text{s.t.} \begin{cases} (x_i^T w - b) y_i \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0 \end{cases}$$

	small C	large C
Desire	Really large margin	Little misclassification
Danger	Underfitting	Overfitting
Outliers	Less sensitive	More sensitive

SVMs and Tikhonov Regularization Learning

We can express Soft-Margin SVMs in an unconstrained fashion as:

$$\min_{w,b,\xi_i} \frac{1}{2} \|w\|^2 + C \sum_i^n \xi_i \quad \text{s.t.} \begin{cases} \xi_i \geq 1 - y_i (w^T x_i - b) \\ \xi_i \geq 0 \end{cases} \forall i \Leftrightarrow \xi_i \geq \max(0, 1 - y_i(w^T x_i - b)) \Leftrightarrow \xi_i = \max(0, 1 - y_i (w^T x_i - b))$$

↑ at optimal, must be tight !

$$\Rightarrow \min_{w,b} \frac{1}{2} \|w\|_2^2 + C \sum_i^n \max(0, 1 - y_i (w^T x_i - b))$$

⇓

This is just Tikhonov:

$$\min_{w,b} \frac{1}{n} \sum_i^n \underbrace{\max(0, 1 - y_i (w^T x_i - b))}_{= L_{HINGE}} + \lambda \|w\|_2^2 \qquad \text{where } \lambda = \frac{1}{2nC}$$

regularized empirical risk minimization, as seen in section for loss fns (left)

empirical risk $\frac{1}{n} \sum_i L(x_i, y_i)$

Dual Ridge Regression (We assume knowledge of duality)

Normal Ridge is of the form

$$\min_w \|Xw - y\|_2^2 + \lambda \|w\|_2^2 \Rightarrow \min_{z,w} \|z\|_2^2 + \lambda \|w\|_2^2 \quad \text{s.t. } z = Xw - y$$ so that we can take the dual

$$\Leftrightarrow p^* = \min_{z,w} \max_\alpha \|z\|_2^2 + \lambda \|w\|_2^2 + \alpha^T (Xw - y - z)$$

$$\Rightarrow d^* = \max_\alpha \min_{z,w} \mathcal{L}(z, \alpha, w)$$

$\nabla_w \mathcal{L}(z, \alpha, w) = 2 \lambda w + X^T \alpha = 0 \Rightarrow w^* = -\frac{X^T \alpha}{2\lambda}$

$\nabla_z \mathcal{L}(z, \alpha, w) = -2z + \alpha = 0 \Rightarrow z^* = \frac{\alpha}{2}$

} $\mathcal{L}(z^*, w^*, \alpha) = \frac{\|\alpha\|_2^2}{4} + \frac{1}{4\lambda} \alpha^T X X^T \alpha + \alpha^T \left(\frac{-X X^T \alpha}{2\lambda} - y - \frac{\alpha}{2} \right)$

$= -\frac{1}{4} \alpha^T \alpha - \frac{1}{4\lambda} \alpha^T X X^T \alpha - \alpha^T y$

$$d^* = \max_\alpha \underbrace{-\frac{1}{4} \alpha^T \alpha - \frac{1}{4\lambda} \alpha^T X X^T \alpha - \alpha^T y}_{g(\alpha)} \quad \text{(maximization of concave)}$$

derivative!

$$\nabla_\alpha g(\alpha) = -\frac{1}{2} \alpha - \frac{1}{2\lambda} X X^T \alpha - y = 0 \Rightarrow \alpha^* = -2\lambda (I\lambda + X X^T)^{-1} y$$

$$\Rightarrow w^* = -\frac{X^T \alpha}{2\lambda} = X^T (X X^T + I\lambda)^{-1} y$$

↳ Kernel Ridge!

Only thing left to do is check strong convexity → KKT:
- → have stationarity
- → slackness doesn't apply
- → need primal feas.

$Xw^* - y - z^* = -\frac{X X^T \alpha}{2\lambda} - y - \frac{\alpha}{2}$

$= -\frac{(\lambda I - X X^T) \alpha}{2\lambda} - y =$ (plug in α^*) $= \ldots = 0 \quad \checkmark$

KKT Conditions

If x^*, ν^*, λ^* optimal and strong duality holds ⇒ KKT conditions are satisfied

If we have points x^*, ν^*, λ^* that satisfy KKT & Primal is convex ⇒ strong duality holds !

Standard form:

$$\min_x f_0(x) \quad \text{s.t.} \quad f_i(x) \leq 0 \quad \forall i = 1, \ldots, m; \quad h_i(x) = 0 \quad \forall i = 1, \ldots, n$$

$$p^* = \min_x \max_{\lambda \geq 0, \nu} f_0(x) + \sum_i^m \lambda_i f_i(x) + \sum_i^n \nu_i h_i(x)$$

KKT conditions:

1) + 2) Primal feasibility: (ineq. ⊕ equalities) $\begin{cases} f_i(x^*) \leq 0 \\ h_i(x^*) = 0 \end{cases}$

3) Dual feasibility: $\lambda_i \geq 0 \quad \forall i$

4) Complementary slackness: $\lambda_i f_i(x^*) = 0$

(for free as we know 1) so $f_i(x^*) \leq 0$ ⇒ λ_i will make $\lambda_i f_i(x^*) = 0$; if $f_i = 0$ λ_i could be greater than 0!)

5) Stationarity: $\nabla_x f(x^*) + \sum_i^m \lambda_i \nabla_x f_i(x^*) + \sum_i^n \nu_i \nabla_x h(x^*) = 0$

Dual SVMs

Solving linearly separable classification problem is equivalent to finding the points of the two convex hulls which are closest to each other and the maximum margin hyperplane bisects and is normal to the line segment joining these two closest points.

$$\max_\alpha \mathbf{1}^\top \alpha^* - \frac{1}{2} \alpha^* \mathbf{Q} \alpha^* \tag{26}$$

$$s.t. \quad \alpha^\top \mathbf{y} = 0 \tag{27}$$

$$0 \leq \alpha_i \leq C\phi_i \quad i = 1, \ldots, n \tag{28}$$

From the dual formulation above, how would you kernelize the SVM with custom margins? What role does the ϕ_i play in the kernelized version?

Solution: We can kernelize the SVM in the same way as in the normal SVM (e.g., note that in the definition of **Q**, we have an $XX^\top$; we replace this with the kernel matrix K).

The ϕ_i serve to adjust the constraints on α_i. If ϕ_i is very large (in the primal this means we want the margin violations to be small for the data point x_i), the constraint on α_i will be very loose. Similarly, if ϕ_i is very small (in the primal this means we allow the margin violation to be large for the data point x_i), the constraint on α_i becomes much tighter.

4 Stability: A Unified Approach to Generalization for Classification and Regression

In this problem, we will study how well machine learning algorithms can generalize from the dataset they have been trained on (the training set) to an unseen dataset (the test set). Assume all data is generated from some distribution $\mathcal{D}$ and we sample a training set $S \sim \mathcal{D}^n$ where $\mathcal{D}^n$ is the distribution of n datapoints drawn i.i.d. from $\mathcal{D}$. In this problem we consider parameter estimators of the form

$$\mathbf{w}(S) = \arg\min_{\mathbf{w}} L_S(\mathbf{w}) + \lambda \|\mathbf{w}\|^2 \tag{29}$$

where $L_S(\mathbf{w})$ is the empirical loss function

$$L_S(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n \ell(\mathbf{w}, z_i) \qquad \text{where } z_i = (\mathbf{x}_i, y_i)$$

evaluated on the training data set $S = (z_1, z_2, \ldots, z_n)$. We are interested in the expected loss

$$L_{\mathcal{D}}(\mathbf{w}) = \mathbb{E}_{z \sim \mathcal{D}}[\ell(\mathbf{w}, z)] \tag{30}$$

on a yet unseen test datapoint z drawn from $\mathcal{D}$.

We call an estimator **ϵ_n-stable** if

$$\mathbb{E}_{S \sim \mathcal{D}^n, z' \sim \mathcal{D}, i \sim U(n)}[\ell(\mathbf{w}(S^{(i)}(z')), z_i) - \ell(\mathbf{w}(S), z_i)] \leq \epsilon_n \tag{31}$$

where $S = (z_1, z_2, \ldots, z_n)$ and $S^{(i)}(z') = (z_1, z_2, \ldots, z_{i-1}, z', z_{i+1}, \ldots, z_n)$ and $U(n)$ is the uniform distribution on $\{1, \ldots, n\}$. All of this randomness is realized independently for the expectations above.

Philosophically, an estimator is stable if the training error on a point z_i doesn't depend too strongly on the point z_i itself being in the training set. (i.e. It doesn't overfit too much to itself.) This philosophical goal is captured mathematically by swapping out one of the training points z_i for a different hypothetical training point z' drawn at random from the same distribution. And then demanding that this swapped training set $S^{(i)}(z')$ does not cause the resulting learned predictor to be too far off while trying to predict the original z_i as compared to the original training set S that included z_i.

We will show that if an estimator is **ϵ_n-stable**, then the test error must be close to the training error in expectation, namely

$$\mathbb{E}_{S \sim \mathcal{D}^n}[L_{\mathcal{D}}(\mathbf{w}(S))] \leq \mathbb{E}_{S \sim \mathcal{D}^n}[L_S(\mathbf{w}(S))] + \epsilon_n. \tag{32}$$

In the first few parts of this problem, we will establish this result and test it on the example of simple averaging (which we already know must learn by the weak law of large numbers) to understand it better. In the second set of parts, we will apply it to show the generalization properties of ridge regression and soft-margin SVMs.

Derive a generalization bound (32) for soft margin SVMs.

Solution: With (41) and the result of part (e) above the result follows immediately.

Since the soft-margin SVM learning rule is essentially ridge-regularized (with the constant C in the traditional formulation just being $\frac{1}{\lambda}$ by multiplying through), this means that we fall exactly into the scenario of part (e) above. This means that soft-margin SVMs inherit the $O(\frac{\rho^2}{\lambda n})$ generalization performance from ridge-regularization more generally. Here, the ρ^2 corresponds to the largest possible size of the test or training points.

Nearest Neighbour Classification

A very intuitively simple approach to classification is just to classify each point according to its neighbours.

Closeness is loosely defined (we can come up with our own measure)

(note training time $O(1)$ as we just store the dataset)

K is a hyperparameter that changes the decision boundary a lot

Training accuracy is always 100% (with $K=1$)

As $K \nearrow$ bias $\nearrow$ Variance $\searrow$ (low K overfits)

Bias Variance Analysis

Let $D = \{(x_i, y_i)\}_{i=1}^n$ be our dataset and our hypothesis $h(z) = \frac{1}{K}\sum_i^n N(x_i, z, K)$ where $N(x_i, z, K) = \begin{cases} y_i & \text{if } x_i \text{ is one of } K \text{ closest to } z \\ 0 & \text{else} \end{cases}$

Also suppose true labels are $y_i = f(x_i) + \varepsilon_i$ (basically we are applying KNN to a regression problem)

WLOG, let $x_1, \dots, x_K$ be the closest K. $\varepsilon_i \sim N(0, \sigma^2)$

$$\therefore \text{bias}^2 = \left(\mathbb{E}[h(z)] - f(z)\right)^2 = \left(\frac{1}{K}\sum_i^K \mathbb{E}[y_i] - f(z)\right)^2 = \left(\frac{1}{K}\sum_i^n f(x_i) - f(z)\right)^2$$

} $\therefore$ as $K \to +\infty$ $\mathbb{E}[h(z)]$ goes to average label for all x; when $K=1$ bias is just $f(x_1) - f(z)$

$$\text{Also variance} = \mathrm{Var}(h(z)) = \frac{1}{K^2}\mathrm{Var}\left(\sum_i^K y_i\right) = \frac{1}{K^2}\mathrm{Var}\left(\sum_i^K f(x_i) + \varepsilon_i\right) = \frac{1}{K^2}\mathrm{Var}\left(\sum_i^K \varepsilon_i\right) = \frac{1}{K^2}\sum_i^K \mathrm{Var}(\varepsilon_i) = \frac{\sigma^2}{K}$$

} as $K \to \infty$ Variance $\to 0$; variance is maximized at $K=1$

Properties:

- Computational Complexity: $O(1)$ training, $O(n)$ space for storing data. Takes $O(n)$ for predictions, but active research in approximations that take less time.
- Flexibility
  - Can be modified to output probabilities $P(Y|X)$ by defining them as proportion of labels among the neighbours (for $K > 1$)
  - Can learn very complicated non-linear boundaries
  - Can be applied both to discrete labels (majority vote / prob.) and continuous (average label)
- Non-Parametric: KNN is non-parametric (does not based on having fixed parameters for a distribution) → # parameters grows with n

  (In contrast, Linear Reg, LDA, QDA, are all parametric)
- Behaviour in high dimensions: Does not do well in high dimensions, as calculating similarity usually involves iterating all d entries of each datapoint (inner products for example take $O(d)$ time). Moreover, as d increases, the datapoints will tend to be further and further away, detracting from the accuracy of the method.
- Theoretical Properties: 1-NN has impressive theoretical guarantees → when $n \to \infty$ the expected error for 1-NN is upper bounded by $2\varepsilon^*$ where ε^* is the Bayes optimal error

  If $\begin{matrix} n \to \infty \\ K \to \infty \end{matrix}$ and $\frac{K}{n} \to 0$ then Error (KNN) $\to \varepsilon^*$

Curse of Dimensionality

Volume of a ball: as $d \to \infty$ nearly all the volume of a ball is on an arbitrarily thin shell

& Concentration of Measure

Improving KNN

1. More data
2. Reduce dimensionality of the features / pick better features (throw some pixels away / lower the resolution)
3. Modify the distance function: Minkowski distances

   $$D_p(x, z) = \left(\sum_i^d |x_i - z_i|^p\right)^{\frac{1}{p}}$$
4. Use Kernels to compute distances in higher dim. space with $O(d)$ computation

Sparsity

Many times we want sparse solutions. This is also a form of regularization.

Sparse Least Squares

$$\min_w \|Xw - y\|_2^2 \quad \text{s.t. } \|w\|_0 \le K$$

($\|w\|_0$ ↳ num non-zero elements; Hamming distance between w and 0)

Why Sparsity? { → Regularization; → Speed up at test time (discard useless features) }

Problem: $\|\ \|_0$ is not convex. Solving is NP-Hard → we will approximate this problem → relaxing constraint: LASSO; → greedily: OMP

(is not even a proper norm)

LASSO

is a relaxed version of Sparse Least Squares

$$\min_w \|Xw - y\|_2^2 \quad \text{s.t. } \|w\|_1 \le K$$

Due to strong duality, this is equivalent to

$$\min_w \|Xw - y\|_2^2 + \lambda\|w\|_1$$

Finding Optimal Solution

Why CD instead of SGD?

There is no closed form solution with linear algebra due to $\|w\|_1$ (not differentiable)

However, objective is still convex, so we can use a gradient descent approach, Coordinate Descent.

In the case of LASSO, our obj. fn is jointly convex in all components of w, so we can apply coordinate descent with the guarantee of finding a global minimum.

By using Coord. Descent we obtain the update:

$$w_i \leftarrow \operatorname*{argmin}_{w_i} L(w)$$

this update has a closed form solution. First lets rewrite the loss

$$L(w) = \|Xw - y\|_2^2 + \lambda\|w\|_1 = \Big\|\sum_j^d x_j w_j - y\Big\|_2^2 + \lambda\sum|w_i|$$

(x_j: col. of X)

$$= \|w_i x_i - r\|_2^2 + \lambda|w_i| + C \qquad \text{where } r = \sum_{j\ne i}^d x_j w_j - y$$

Suppose $w_i^* > 0$:

$$\nabla_{w_i} L(w) = \lambda \sum_j^n 2x_{ji}(w_i x_{ji} + r_j) = 0 \Rightarrow w_i^* = \frac{-\lambda - \overbrace{\sum_j^n x_{ji} r_j}^{a}}{\underbrace{\sum_j^n x_{ji}^2}_{b}} = \frac{-\lambda + a}{b}$$

This only holds if $w_i^* = \frac{-\lambda + a}{b}$ is actually positive

If negative or 0, there is no optimum in $(0, \infty)$

When $w_i^* < 0$:

$$w_i^* = \frac{\lambda + a}{b} \quad \left(\text{only when } \frac{\lambda + a}{b} \text{ actually negative}\right)$$

If positive or 0, there is no optimum in $(-\infty, 0)$

If neither, then there is no optimum in $(-\infty, 0) \cup (0, +\infty)$.

However we know that there must exist an optimum $\Rightarrow w_i^* = 0$ (as jointly convex)

$$\therefore w_i^* = \begin{cases} 0 & \text{if } |a| \le \lambda \\ \frac{-\lambda + a}{b} & \text{if } \frac{-\lambda + a}{b} > 0 \\ \frac{\lambda + a}{b} & \text{if } \frac{\lambda + a}{b} < 0 \end{cases}$$

} By increasing λ we make sparsity more common.

[Not a grad. descent Update?] (other ws fixed as constants?)

→ After having been set to 0, weights can be "reactivated" in subsequent iteration.

[Need to center data to zero in order to penalize all features similarly]

Orthogonal Matching Pursuit

Extension of Matching pursuit: at each iteration t, we maintain a set I^t of all features selected by the algorithm so far.

Instead of updating one component of the weight vector at a time, we update all weights corresponding to the features in I^t using Least Squares

Algorithm

Initialize $w^0 = 0$ and $r^0 = y - Xw^0 = y$

Initialize features $I^0 = \phi$

while $\|w\|_0 \le K$:

- find the feature for which length of projected residual onto x_i is minimized

  $$i = \operatorname*{argmin}_j \left(\min_v \|r^{t-1} - v x_j\|\right) = \operatorname*{argmax}_j \frac{|\langle r^{t-1}, x_j\rangle|}{\|x_j\|}$$
- add the feature to the set: $I^t = I^{t-1} \cup \{i\}$
- Estimate the best linear fit possible using the current features:

  $$w^t = \operatorname*{argmin}_{w \in \mathbb{R}^t} \|Xw - y\|_2^2$$
- update residual vector: $r = Xw^t - y$

Idea: If we don't refit, we are not guaranteed that new residual will be orthogonal to the span of canonical basis vectors, and therefore is not optimal.

$\therefore$ OMP ensures optimality on the set of features I^t at each step.

Now the problem is convex, but why does it work?

Derivative Intuition

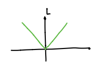
In SVM we have the term: $C\sum \xi_i$, $\xi_i \ge 0$

$\Rightarrow \frac{\partial L}{\partial \xi_i} = C$ } whatever ξ_i is the reward I get by lowering it is C → lower until equilibrium

If instead we had: $C\sum \xi_i^2$

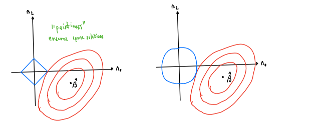
$\frac{\partial L}{\partial \xi_i} = 2\xi_i C$ } the reward I get by lowering it is C depends on ξ_i → if ξ_i is small, not really worth lowering more

LASSO: $\|w\|_1 = \sum|w_i|$



} same idea: if I'm not at zero, I'll always get a reward of λ by taking a unit step towards zero

Geometric Intuition



Coordinate Descent

while not converged:

{ pick feature index i; update w_i to $\operatorname*{argmin}_{w_i} L(w)$ } Idea: Just update one component of the weight at a time

SGD: Optimizes $L(w)$ for current batch

Coordinate Descent: Optimizes $L(w)$ for current feature

We are guaranteed to find the global minimum when $L(w)$ is jointly convex (not if element-wise)

Why?

$f(x, y)$ element-wise cvx if $\frac{\partial}{\partial x \partial y} f(x, y) = H_f \ge 0$ when x or y are fixed.

Why not just set gradients to zero?

$\min_{x,y} f(x, y) = \min_x \min_y f(x, y)$ is true, but!

{ $f(x,y)$ not jointly convex $\therefore \min_y f(x,y)$ not necessarily convex (from EE 127) $\therefore \min_x$ not guaranteed to have a global minimum! }

Matching Pursuit

Rather than relaxing $\|\ \|_0$ constraint, MP finds an approximate solution to the Sparse Least Squares in a Greedy fashion.

Starts with a completely sparse solution $w^0 = 0$ and iteratively updates w until sparsity constraint $\|w\|_0 \le K$ can no longer be met. At each iteration updates w in the coordinate that would reduce the residuals the most

Set $w^0 = 0$ and $r^0 = y - Xw^0 = y$ (residual)

while $\|w\|_0 \le K$:

- find feature i for which the length of the projected residual onto x_i is maximized

  $$i = \operatorname*{argmin}_j \left(\min_v \|r^{t-1} - v x_j\|\right) = \operatorname*{argmax}_j \frac{|\langle x_j, r^{t-1}\rangle|}{\|x_j\|_2} \quad (1)$$

  ← no square!
- update w by adding the new feature

  $$w_i^t = w_i^{t-1} - \frac{\langle r^{t-1}, x_i\rangle}{\|x_i\|_2^2}$$

  (squared) } why does this work?
- update r:

  $$r^t = y - Xw^t$$

Why does this work?

Consider (1) → $\min_v \|r^{t-1} - v x_j\|$ is a projection problem: finding distance of the span of x_j from r^{t-1}

$$v^* = \frac{\langle r^{t-1}, x_j\rangle}{\langle x_j, x_j\rangle} \Rightarrow \|r^{t-1} - v^* x_j\| = \left\|r^{t-1} - \frac{\langle r^{t-1}, x_j\rangle}{\|x_j\|_2} x_j\right\|$$

The outer optimization minimizes the distance:

there's something weird here

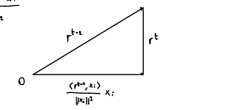
$$i = \operatorname*{argmin}_j \left\|r^{t-1} - \frac{\langle r^{t-1}, x_j\rangle}{\|x_j\|_2^2} x_j\right\|$$

$$= \operatorname*{argmin} \|r^{t-1}\|^2 + \frac{\langle r^{t-1}, x_j\rangle^2}{\|x_j\|_2^{4}}\|x_j\|_2^2 - 2 r^{t-1\,T} x_j \frac{\langle r^{t-1}, x_j\rangle}{\|x_j\|_2^2}$$

($r^{t-1\,T} x_j = \langle x_j, r^{t-1}\rangle$)

$$= \operatorname*{argmin}_j - \frac{\langle r^{t-1}, x_j\rangle^2}{\|x_j\|_2^2}$$

$$= \operatorname*{argmax}_j \frac{|\langle r^{t-1}, x_j\rangle|}{\|x_j\|_2}$$



$$\therefore r^t = r^{t-1} - \frac{\langle r^{t-1}, x_i\rangle}{\|x_i\|^2} x_i \overset{\text{equivalent to}}{\Rightarrow} w_i^t = w_i^{t-1} - \frac{\langle r^{t-1}, x_i\rangle}{\|x_i\|^2} x_i$$

$\therefore$ at each iteration, the squared length of the residual monotonically decreases

In this question, we will analyze the variance of learning sparse linear models. In particular, we will analyze two simple procedures (computing the OLS solution and then just keeping the biggest entries or just keeping entries bigger than a threshold) that perform feature selection in linear models, and show quantitatively that feature selection lowers the variance of linear models. This should make sense to you at an intuitive level: enforcing sparsity is equivalent to deliberately constraining model complexity; think about where this puts you on the bias variance trade-off.

However, note that there is a subtle difference between feature selection before training and after training. If we use fewer features to begin with, our results so far imply that we will have low variance because of smaller model complexity. What we learn from working through this problem is that selecting features adaptively (that is based on the training data itself) does not hurt either, if done properly. In other words, although there is a philosophical difference between doing feature selection before or after using the data, post training feature selection still leads to variance reduction under certain assumptions.

As you have learned in lecture and discussion, Orthogonal Matching Pursuit is a greedy method to pick relevant coordinates that can explain the data best. It searches for a direction (among the columns) such that the residual error can be explained best and updates the entire fit in the augmented space. Matching Pursuit (MP) is a simpler variant where the entire fit is not updated at every step, just the fit along the new coordinate that was chosen.

Decision Trees, Ensembling & Boosting

Decision Trees

→ Can be used for both classification & regression
↳ each branch represents a split in the data

Can represent an arbitrarily complex classifier.

Training

Trained in recursive fashion going downwards from the root, at each step finding the best split of the data.
The base case is when we stop splitting the data, and give a prediction (leaf nodes)

In CS 189 we only consider splits of the type $x_j \geq v$, $x_j < v$. There are finitely many possible splits, and they can be found by ordering the x datapoints by x_j

Idea: At each split, we want to reduce the classifier's uncertainty as much as possible! (how to quantify?)

Entropy & Information

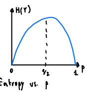
"Surprise" of a discrete R.V. Y taking on value K:

$$\log \frac{1}{\mathbb{P}(Y=K)} = -\log \mathbb{P}(Y=K)$$

surprise → 0 as p → 1; → ∞ as p → 0

Entropy of an RV. Y is the expected surprise across K:

$$H(Y) = \mathbb{E}\left[-\log P(Y)\right] = -\sum_K P(Y=K) \log P(Y=K)$$



Entropy vs. p for $Y \sim \text{Bernoulli}(p)$

In general, entropy is higher the closer the distribution is to a uniform distribution

Entropy Approach to Decision Trees

Concretely we don't have the probability for each class, but we can estimate it as:

$$\hat{\mathbb{P}}(Y=K) = \frac{\#\ y_i = K}{n}$$

We then know that we want to pick a split that minimizes entropy: of what?

parent set $H(Y)$ one side of tree $H(Y \mid X_{j,v}=1)$ other side $H(Y \mid X_{j,v}=0)$

We can't control $H(Y)$ but can control the other two.
A reasonable thing to do is minimize the conditional average: "conditional entropy"

$$\min_{j,v} H(Y \mid X_{j,v}) = \mathbb{P}(X_{j,v}=1)\, H(Y \mid X_{j,v}=1) + \mathbb{P}(X_{j,v}=0)\, H(Y \mid X_{j,v}=0)$$

Equivalently, we can maximize the mutual information (information gained)

$$\max I(X_{j,v}; Y) = H(Y) - H(Y \mid X_{j,v})$$

↳ always non-negative

Gini Impurity

Another way to measure the quality of a split. Measures how often a randomly chosen element would be incorrectly labeled if it were labeled randomly according to the distribution of the subset

$$G(Y) = \sum_K \mathbb{P}(Y=K)\left(1-\mathbb{P}(Y=K)\right) = \sum_K \mathbb{P}(Y=K) - \mathbb{P}(Y=K)^2 = 1 - \sum_K \mathbb{P}(Y=K)^2$$

(picking a point with label K; picking wrong label)

Gini for Decision Trees

Can define a Gini for splits:

$$G(Y \mid X_{j,v}) = P(X_{j,v}=1)\, G(Y \mid X_{j,v}=1) + P(X_{j,v}=0)\, G(Y \mid X_{j,v}=0)$$

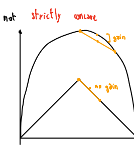
Why not misclassification rate?

An intuitive approach would be to just minimize the weighted misclassification rate

$$M(Y) = 1 - \max_K \mathbb{P}(Y=K) \rightarrow M(Y \mid X_{j,v})$$

However, this doesn't work as multiple splits end up having the same value of $M(Y \mid X_{j,v})$ but actually be of different quality.

The problem lies with the fact that $M(Y)$ is not strictly concave



Stopping Criteria

→ Depth limit

→ Node purity: proportion of points of same class is sufficiently high

→ Information Gain: if lower than a threshold

Note: the above are not mutually exclusive

Consider also pruning: recombining splits if reduces validation error

Random Forests

The main idea is to combine multiple Decision Tree classifiers. This technique falls under ensembling.
Ensembling relies on the idea that having multiple partially-independent measurements will reduce the variance of our prediction, without biasing the estimate

Because the way we created decision trees was deterministic, we need to introduce stochasticity in order to create a forest of different trees.

- Bagging (Bootstrap Aggregating): sample $m < n$ datapoints uniformly with replacement, and use them as training set
- Feature Randomization: at each split, only consider $K < d$ random features

Both the size of random subsample and number of features can be chosen with cross validation.

Boosting

Random Forests: All classifiers same importance

Boosting: weigh classifiers by accuracy

Boosting Idea: combine models (weak learners) in a principled manner — Focus on finding learners that the overall learner currently does not predict very well

Boosting vs Matching Pursuit

In both, overall predictor is an additive combination of pieces which are selected one-by-one in a greedy fashion

Ada Boost (Adaptive Boosting)

One of the most common boosting algorithms.
Assume we have a dataset $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$

Algorithm

1) Initialize weights $w_i = \frac{1}{n}$ $\forall i = 1, \dots, n$ training points
2) Repeat for $m = 1, \dots, M$
   a) Build a classifier $G_m : \mathbb{R}^d \rightarrow \{-1, +1\}$ where datapoints are weighted according to w_i
   b) Compute weighted error rate $e_m = \frac{\sum_{\text{misclassified}} w_i}{\sum w_i}$
   c) Re-weigh the training points: $w_i \leftarrow w_i \begin{cases} \sqrt{\frac{1-e_m}{e_m}} & \text{if misclassified by } G_m \\ \sqrt{\frac{e_m}{1-e_m}} & \text{else} \end{cases}$
   d) Optional: normalize weights w_i to sum to 1

a) → To weigh datapoints for a classifier, we can just resample with replacement (n times) with probabilities $\vec{w}$ to create our training data

c) Assuming $e_m \leq \frac{1}{2} \Rightarrow \frac{1-e_m}{e_m} > 1$ so more weight on misclassified points and less on well classified

How to make a prediction? (once we have weak learners)
Could use logistic regression with $G_m(x)$ as features
Better: $\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$ (weight for each classifier!) $\quad h(x) = \text{sign}\left(\sum_i^M \alpha_m G_m(x)\right)$

Note: $e_m \rightarrow 0$: $\alpha_m \rightarrow +\infty$; $e_m \rightarrow 1$: $\alpha_m \rightarrow -\infty$

Derivation

The derivation will be inductive. Suppose we have computed the classifiers $G_1, \dots, G_{m-1}$ along with corresponding weights $\alpha_1, \dots, \alpha_{m-1}$. We now want to compute the next classifier G_m and it's weight α_m. The output of the model so far is $F_{m-1}(x) = \sum_i^{m-1} \alpha_i G_i(x)$. We want to minimize emirical risk:

$$\therefore \arg\min_{\alpha_m, G_m} \frac{1}{n} \sum_i^n L\left(y_i, F_{m-1}(x_i) + \alpha_m G_m(x_i)\right) \quad \text{for some Loss fn } L(y_i, \hat{y}_i)$$

For Ada Boost, we use exponential loss: $L(y_i, \hat{y}_i) = e^{-y_i \hat{y}_i}$

if $y_i\hat{y}_i > 0$ ($\hat{y}_i$ right sign) then loss decreases exponentially with $|\hat{y}_i|$ (confidence of prediction)
if $y_i\hat{y}_i < 0$ loss increases exponentially with confidence

we get

$$\Rightarrow \arg\min_{\alpha_m, G_m} \frac{1}{n} \sum_i^n e^{-y_i\left(F_{m-1}(x_i) + \alpha_m G_m(x_i)\right)}$$

$$= \arg\min_{\alpha_m, G_m} \frac{1}{n} \sum_i^n e^{-y_i F_{m-1}(x_i)} e^{-y_i \alpha_m G_m(x_i)}$$

($e^{-y_i F_{m-1}(x_i)} = w_i^{(m)}$, constant w.r.t. α_m, G_m)

$$= \arg\min_{\alpha_m, G_m} \sum_{y_i = G_m(x_i)} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq G_m(x_i)} w_i^{(m)} e^{\alpha_m} \overset{(\dots)}{=} \arg\min_{\alpha_m, G_m} \left(e^{\alpha} - e^{-\alpha}\right) \sum_{y_i \neq G_m(x_i)} w_i^{(m)} + e^{-\alpha} \sum_{i=1}^n w_i^{(m)}$$

best G_m minimizes total weight of misclassified points

$$= \arg\min_{\alpha_m, G_m} (1-e_m)\, e^{-\alpha} + e_m e^{\alpha} \quad \text{where } e_m = \frac{\sum_{y_i \neq G_m(x_i)} w_i}{\sum w_i}$$

Setting derivative to zero:

$$\alpha_m = \frac{1}{2}\ln\left(\frac{1-e_m}{e_m}\right)$$

$$\therefore w_i^{(m+1)} = e^{-y_i F_m(x_i)} = \dots = w_i^{(m)} \sqrt{\frac{e_m}{1-e_m}}^{\,y_i G_m(x_i)}$$

Even once training error goes to zero, test error will continue to decrease: refining it's fit.

How?

Gradient Boosting

Adaboost assumes exponential loss fn. Gradient Boosting is more general technique: allows for any differentiable loss.
Here G is no longer necessarily a classifier. Could be real-valued (if regression)

We have objective

$$\min_{\alpha, G} \sum_i^n L\left(y_i, F_{m-1}(x_i) + \alpha G(x_i)\right)$$

By Taylor expansion we have: $L\left(y_i, F_{m-1}(x_i) + \alpha G(x_i)\right) \approx L\left(y_i, F_{m-1}(x_i)\right) + \frac{\partial L}{\partial \hat{y}}\left(y_i, F_{m-1}(x_i)\right) \alpha G(x_i)$

1 Orthogonal Matching Pursuit

Consider the problem setting, where you are given $\mathbf{X}$ and $\mathbf{y}$ where $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^n$. That is we have n observations that are given by linear combinations of d-features. How can we find the original $\mathbf{w}$ such that $\mathbf{y} = \mathbf{X}\mathbf{w}^*$? We've learned many techniques for this so far, but suppose you have an under-determined system $n \ll d$ and we want a k-sparse solution (a signal $\mathbf{w}$ is called k-sparse if $|\mathbf{w}|_0 = k$). In the lecture, we discussed Lasso (ℓ_1-regularized regression) as one of the powerful methods to solve such problems. And furthermore, we saw that coordinate descent can be employed to solve the lasso-penalized regression problem. With coordinate descent, we make progress along a randomly chosen coordinate or feature. A natural question arises: can we choose the coordinates in a smart way? And what if we optimize (rather than taking simply a step) along that coordinate? Let's review one technique, that attempts a greedy coordinate selection procedure: orthogonal matching pursuit. Let us introduce some notation: we use $\mathbf{x}_j$ to denote the j-th column of the matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_d \\ | & | & & |. \end{bmatrix}$$

Note that the vector $\mathbf{x}_j \in \mathbb{R}^n$ denotes the j-th feature for all n data points.

OMP Algorithm:

1. Initialize the residue $\mathbf{r}^0 = \mathbf{y}$ and initialize the index set to be $I^0 = \emptyset$. Set your estimate $\hat{\mathbf{w}}^0 = 0$.
2. Repeat for $t = 1, \dots$ **until** $\|r^t\|_2 <$ threshold or $t > k$(sparsity budget):
   - **Index update**: Find an index j^t which reduces the residue most. In other words, find the best one-feature linear fit to the existing residual vector, and then update the index set by including the index corresponding to the best feature. Mathematically this update can be written as:
   $$\mathbf{r}^t = \mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^{t-1}$$
   $$j^t = \arg\min_i(\min_v \|\mathbf{r}^t - v\mathbf{x}_i\|_2^2)$$
   $$I^t = I^{t-1} \cup \{j^t\}$$
   - **Estimate update**: Estimate the best linear fit of the target $\mathbf{y}$ using the features obtained so far. Given that we have found t good features, we now find the best linear fit for the

target $\mathbf{y}$ using these t-features. Define $\mathbf{X}_t = \left[\mathbf{x}_{j^1}, \dots, \mathbf{x}_{j^t}\right]$ made up of these t-features. Then we determine $\hat{\mathbf{w}}^t$ as the solution for the following least-squares problem:

$$\hat{\mathbf{w}}^t = \arg\min_{\mathbf{w} \in \mathbb{R}^t} \|\mathbf{y} - \mathbf{X}_t\mathbf{w}\|_2^2$$

We now discuss under what conditions can we expect such a greedy-coordinate-finding procedure to provide us a good solution. To keep the discussion centered around key ideas, we discuss the simplest case possible: recovery of a one-sparse signal.

OMP is also related to the classical idea of Boosting. As you have already seen (or may see in the next few lectures) boosting is a powerful recipe of training a set of weak learners (that do not fit the data very well by themselves) and combining them to find a strong learner (that fits the data well). The general idea is as follows: In the first step, we use one weak learner to fit a given dataset. In the next step, we use another weak learner to improve upon the first learner, by putting more weights on the data points that the first learner was unable to fit well (wrong classification or large squared error in regression). We repeat this process until a desirable accuracy is achieved.

Can you see OMP as an illustration of boosting for regression?

Solution: Note that since we fit residuals, it is equivalent to improving upon the past weak learners. Also we only pick one learner at a time (hence we can consider our learners to be weak). However, the estimation update step, refits using all the features discovered so far, and hence strictly speaking the weak learner at next step is one new feature plus all the features identified so far.

In matching pursuit, we just have the index update step and hence the weak learner at each step is just a new feature.

(e) (BONUS) *MP as gradient boosting* The two previous greedy principles of matching-pursuit (column selection and linesearch) can be extended to more general settings of non-linear and non-parametric function classes $\mathcal{F}$ and general loss functions $\mathcal{L}$ (which need not be squared loss). Consider the general learning problem where again we are given a bunch of pairwise training samples in the form of $(\mathbf{x}_i, y_i)$ for $i = 1, \dots, n$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{Y} \subset \mathbb{R}$ and we want to learn a function $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ from a function space $\mathcal{F}$ (with possibly non-linear and non-parametric basis elements, such as trees) which we can then use to predict y_{test} by $f(\mathbf{x}_{test})$ for a test sample $\mathbf{x}_{test}$.

For this purpose we minimize the following (average) loss $\mathcal{L} : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\mathcal{L}(\mathbf{f}(\{\mathbf{x}\}_{i=1}^n)) = \mathcal{L}(f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, f(\mathbf{x}_i)).$$

for some point-wise loss function $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ for one sample. Notice that the loss maps a vector to a scalar, as in the linear settings. Here, we use $\mathbf{f}(\{\mathbf{x}\}_{i=1}^n) := (f(\mathbf{x}_1), \dots, f(\mathbf{x}_n))$ to denote the vector of function values of $f \in \mathcal{F}$ at the points $\mathbf{x}_1, \dots, \mathbf{x}_n$.

As in gradient descent (4), we want to minimize the loss using an iterative algorithm with updates of the form

$$f^{t+1} = f^t + \alpha_t \widetilde{f}^t.$$

Note that our notation follows the convention that $f^t \in \mathcal{F}$ is the current iterate and $\widetilde{f}^t \in \mathcal{W}$ is the direction that we are adding at each step. The main question is how to choose $\widetilde{f}^t$. Let's assume that it is for example computationally much simpler to search over a smaller set of functions $\mathcal{W} \subset \mathcal{F}$ at each timestep.

Similar to the parametric update in (3), we then choose the function $\widetilde{f} \in \mathcal{W}$ for which the function value vector $\widetilde{\mathbf{f}}(\{\mathbf{x}\}_{i=1}^n)$ has the maximal inner product with the negative gradient, i.e.

$$\widetilde{f}^t = \arg\max_{\widetilde{f} \in \mathcal{W}} \langle -\nabla\mathcal{L}(\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)), \widetilde{\mathbf{f}}(\{\mathbf{x}\}_{i=1}^n)\rangle \tag{9}$$

Fit Page

where the gradient $\nabla\mathcal{L}(\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)), \widetilde{\mathbf{f}}(\{\mathbf{x}\}_{i=1}^n) = \nabla_{\mathbf{v}}\mathcal{L}(\mathbf{v})\big|_{\mathbf{v}=\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)}$ which you have already computed above for the square loss.

Now we want to see how Matching Pursuit can be viewed as an instance of this framework, also called *gradient boosting or functional gradient descent*. Specifically, **show that the particular choices of**

$$\mathcal{F} = \{f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} : \mathbf{w} \in \mathbb{R}^d\} \text{ and } \mathcal{W} = \{\widetilde{f}(\mathbf{x}) = \pm\mathbf{e}_i^\top\mathbf{x} : i = 1, \dots, d\},$$

where $\mathbf{e}_i$ is the i-th standard basis element with a 1 in the i-th position and zeros elsewhere, yield Matching Pursuit as a form of gradient boosting. Notice that $\mathcal{F}$ is the function space of all linear functions and $\mathcal{W}$ is the set of functions on vectors $\mathbf{x} \in \mathbb{R}^d$ which pick out signed coordinates of $\mathbf{x}$.

Hint: For matching pursuit, what are $\widetilde{\mathbf{f}}^t(\{\mathbf{x}\}_{i=1}^n)$ and $\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)$?

Solution:
We offer two ways to derive this equivalence.

1. Going from MP to $\mathcal{F}, \mathcal{W}$ Notice that $\mathbf{X}\mathbf{w}$ can be viewed as a *vector* of function values with $f(\mathbf{x}) = \mathbf{w}^\top\mathbf{x}$ evaluated at $\mathbf{x}_i$ for $i = 1, \dots, n$. Similarly, one column vector of the matrix $\mathbf{X}$ can be written as $\mathbf{X}\mathbf{e}_i$ and thus yields functions of the form $\widetilde{f}(\mathbf{x}) = \mathbf{e}_i^\top\mathbf{x}$. Therefore, the definitions of $\mathcal{F}, \mathcal{W}$ above reflect the effective functions (and hence function spaces) we are fitting and updating in the matching pursuit method when viewed as a gradient boosting method.

2. Going from $\mathcal{F}, \mathcal{W}$ to MP With the given definition of $\mathcal{F}$, we obtain

$$\mathcal{F}(\{\mathbf{x}\}_{i=1}^n) = \{\mathbf{v} = \mathbf{X}\mathbf{w} : \mathbf{w} \in \mathbb{R}^d\} = \{\sum_{i=1}^d \phi_i(\mathbf{X})w_i : \mathbf{w} \in \mathbb{R}^d\},$$

a linear combination of the columns of $\mathbf{X}$ which matches the matching pursuit iterates $\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)$ with the weights w_i corresponding to $\frac{\langle \mathbf{r}^{t_i}, \phi_i(\mathbf{X})\rangle}{\|\phi_i(\mathbf{X})\|_2^2}$ where t_i is the time when index i was selected.
With the definition of $\mathcal{W}$, we obtain

(f) (BONUS) *AdaBoost as gradient boosting* Now let's have a look at non-linear and non-parametric functions such as decision trees for classification. In this problem we will establish that AdaBoost is also an instance of gradient boosting on a particular loss function (which is not the zero-one loss!).

Notice how each decision tree can be viewed as a function $\widetilde{f} : \mathbb{R}^d \rightarrow \{-1, +1\}$. Since in each iteration we fit a tree to the weighted samples, our function set $\mathcal{W}$ now corresponds to the set of decision trees with a fixed set of parameters.

Remember the AdaBoost algorithm from lecture and the last homework. At each iteration t, it upweights each sample i by a different w_i^t in every iteration and finds a new tree which solves the reweighted classification problem

$$\widetilde{f}^t_{Ada} = \arg\min_{\widetilde{f} \in \mathcal{W}} \sum_{i=1}^n w_i^t \mathbb{I}(y_i \neq \widetilde{f}(\mathbf{x}_i)) \text{ where } w_i^t = \frac{\exp(-y_i f^t(\mathbf{x}_i))}{\sum_{i=1}^n \exp(-y_i f^t(\mathbf{x}_i))}$$

where f^t is a linear combination of functions in $\mathcal{W}$ found by previous iterations and $\mathbb{I}(A) = 1$ if A is true and 0 otherwise (the indicator function). Notice that the samples which the current classifier f^t gets wrong are assigned a larger weight.

The update for the overall classifier reads

$$f^{t+1} = f^t + \alpha_t \widetilde{f}^t_{Ada} \tag{10}$$

The classification rule is then obtained by taking the sign of the final iterate at time τ, i.e. $f_{\text{final}}(\mathbf{x}) = \text{sign}(f^\tau(\mathbf{x}))$ which lies in a bigger space $\mathcal{F} \supset \mathcal{W}$.

Show that $\widetilde{\mathbf{f}}^t_{\text{Ada}}(\{\mathbf{x}\}_{i=1}^n) = \widetilde{\mathbf{f}}^t(\{\mathbf{x}\}_{i=1}^n)$ in the gradient boosting framework (9) for the loss $\ell(y, f(\mathbf{x})) = e^{-yf(\mathbf{x})}$. This implies that finding the best fit for the reweighted samples is equivalent to fitting the gradient. For the gradient in (9), notice that we again view $\mathcal{L}$ as a function of a vector $\mathbf{v}$ with

$$\mathcal{L}(\mathbf{v}) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, f(\mathbf{x}_i)) = \frac{1}{n}\sum_{i=1}^n e^{-y_i v_i} \tag{11}$$

and the gradient $\nabla\mathcal{L}(\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)) = \nabla_{\mathbf{v}}\mathcal{L}(\mathbf{v})\big|_{\mathbf{v}=\mathbf{f}^t(\{\mathbf{x}\}_{i=1}^n)}$.

Hint: Now note that for $y \in \{-1, +1\}$ and $\widetilde{f} : \mathbb{R}^d \rightarrow \{-1, +1\}$, we have

$$y_i\widetilde{f}(\mathbf{x}_i) = 2(1 - \mathbb{I}(y_i \neq \widetilde{f}(\mathbf{x}_i))) - 1. \tag{12}$$

Fit Page

Hint: Recall that the maximizer does not change if you scale by or add to the function a constant factor which is independent of the object you are searching over, i.e.

$$\arg\max_{\widetilde{f} \in \mathcal{W}} CH(\widetilde{\mathbf{f}}(\{\mathbf{x}\}_{i=1}^n)) + G(\mathbf{f}(\{\mathbf{x}\}_{i=1}^n)) = \arg\max_{\widetilde{f} \in \mathcal{W}} H(\widetilde{\mathbf{f}}(\{\mathbf{x}\}_{i=1}^n))$$

for some functions H, G that depend on the function value vectors $\mathbf{f}, \widetilde{\mathbf{f}}$, and a scalar $C \in \mathbb{R}$. f does not depend on $\widetilde{f}$.

Solution: First let's compute the negative gradient of $\mathcal{L}(\mathbf{v})$ with respect to $\mathbf{v}$ evaluated at $\mathbf{v} = \mathbf{f}(\{\mathbf{x}\}_{i=1}^n)$ (and thus $\mathbf{v}_i = f(\mathbf{x}_i)$) which has entries

$$-\frac{\partial}{\partial f(\mathbf{x}_i)}\mathcal{L}(\mathbf{f}(\{\mathbf{x}\}_{i=1}^n)) := -\frac{\partial}{\partial f(x_i)}\mathcal{L}(\mathbf{v}_i)\big|_{\mathbf{v}_i = \widetilde{f}(\mathbf{x}_i)}$$

$$= y_i \exp(-y_i \mathbf{f}(\mathbf{x}_i)).$$

In the following, we drop the dependence on time t for $\widetilde{\mathbf{f}}^t$ and $\mathbf{f}^t$ to simplity notation. Then, by definition we have

$$\langle -\nabla\mathcal{L}(\mathbf{f}(\{\mathbf{x}\}_{i=1}^n)), \widetilde{f}(\{\mathbf{x}\}_{i=1}^n)\rangle = \frac{1}{n}\sum_{i=1}^n -\frac{\partial}{\partial f(\mathbf{x}_i)}\mathcal{L}(\mathbf{f}(\{\mathbf{x}\}_{i=1}^n))\widetilde{f}(\mathbf{x}_i)$$
$$= \frac{1}{n}\sum_{i=1}^n y_i\widetilde{f}(\mathbf{x}_i)\exp(-y_i f(\mathbf{x}_i))$$
$$= \frac{1}{n}\sum_{i=1}^n [2(1 - \mathbb{I}(y_i \neq \widetilde{f}(\mathbf{x}_i))) - 1]\exp(-y_i f(\mathbf{x}_i))$$

where the last equality follows from (12). Now see that by the second hint, we have

$$\arg\max_{\widetilde{f} \in \mathcal{W}} \frac{1}{n}\sum_{i=1}^n [2(1 - \mathbb{I}(y_i \neq \widetilde{f}(\mathbf{x}_i))) - 1]\exp(-y_i f(\mathbf{x}_i))$$
$$= \arg\max_{\widetilde{f} \in \mathcal{W}} -\frac{1}{n}\sum_{i=1}^n \mathbb{I}(y_i \neq \widetilde{f}(\mathbf{x}_i))\exp(-y_i f(\mathbf{x}_i))$$

for any conseq… arg m… Conseq… same se…

If you normalize the columns of $\mathbf{X}$ (i.e. scale the features) so that $\|\mathbf{x}^{(i)}\|_2^2 = 1$ for each column $\mathbf{x}^{(i)}$, **what is the equivalent Tikhonov regularization matrix if you use dropout? Explain why (or in what context) dropout's induced implicit Tikhonov regularization might be better than the identity matrix that vanilla ridge regression uses when we are working with unnormalized columns of X.**

Solution: If we normalize the columns of $\mathbf{X}$, $\Gamma = \mathbf{I}$ will be the identity matrix, which means that linear regression with dropout is equivalent to ridge regression. When the features in $\mathbf{X}$ are not normalized, the corresponding term in the Tikhonov regularization matrix will be larger if the scale of the corresponding feature is bigger.

When we do not have any reason to put a lot of importance on the units used to represent the individual features, this is probably what we want because it will match the magnitude of $\mathbf{x}^{(i)}w_i$ and $\Gamma_i w_i$. In other word, it will act like doing ridge regression as though each feature was normalized even if we do not explicitly normalize them. This explains the attractiveness of dropout in algorithmic scenarios like within neural nets where normalization of features might not be natural.

To see the fact that dropout is a way of doing regularization, **prove that**

$$\arg\min_{\mathbf{w}} \ell'(\mathbf{w}) = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \frac{1-p}{p}\|\Gamma\mathbf{w}\|_2^2, \tag{13}$$

where $\Gamma = \text{diag}(\mathbf{X}^\top\mathbf{X})^{1/2}$ — i.e. Γ is a diagonal matrix with just the Euclidean norms of the feature vectors along the diagonal. This is saying that applying dropout *on the linear regression problem* is equivalent to doing the ridge regression with a special diagonal Tikhonov regularization matrix. For the effect of dropout on more complex models such as logistic regression or convolution neural networks, it is hard to obtain a closed-form solution. But the idea dropout is equivalent to some kind of regularization still applies.