3. Statistical Learning Theory

Decision theory process:

① Observe input x — Input space $\mathcal{X}$

② Take action a — Action space $\mathcal{A}$

③ Observe outcome y — Output space $\mathcal{Y}$

④ Evaluate a, y using loss function

Decision function takes an input $x \in \mathcal{X}$ and produces an action $a \in \mathcal{A}$.

Loss function takes action a and output y and produces loss $\ell \in \mathbb{R}$.

The main thing we want is a good decision function. How do we evaluate how good f is?

One way is to use ℓ, but that is only one point. Statistical Learning Theory tells us how to evaluate the average error of f.

Assume that there is a data generating function P_{xxy}. All input and output pairs are generated independently and identically from P_{xxy}.

Risk function is the true expected value of ℓ. $R(f) = E\,\ell(f(x), y)$

To calculate R, we need the distribution P_{xxy}, but we do not have it.

Bayes Decision function is the function which minimises $R(f)$. $f^* = \arg\min_f R(f)$

	Least Squares	Multi-class
Action A / Output space y	$\mathbb{R}$	$\{0, 1, \dots, K-1\}$
Loss ℓ	$(a-y)^2$	$I(a \neq y)$
Risk R	$E(f(x)-y)^2$	$E(\ell) = 0 \cdot P(f(x)=y) + 1 \cdot P(f(x)\neq y) = P(f(x)\neq y)$
Target f^*	$E(y \mid x)$	$\arg\max_k P(y=k \mid x)$

need to prove

But we cannot compute all these probabilities and expected values because we don't have P_{xxy}. So we need to use the data sample.

Empirical risk function

Let data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be drawn iid from P_{xxy}.

The empirical risk of f is: $\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$ ie. average loss.

Since (x_i, y_i) are iid, then $\ell(f(x_i), y_i)$ is also iid.

By the strong law of large numbers:

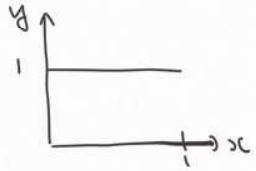
$$\text{almost surely} \quad \lim_{n\to\infty} \hat{R}_n(f) = E\,\ell(f(x), y)$$
$$= R(f)$$

This gives us a way to evaluate f.

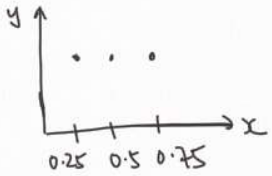
Empirical Risk minimisation. At this point, we might ask - can we minimise $\hat{R}(f)$ to get the best decision function f?

Answer: Yes, but only if we constrain the hypothesis space. Consider this example:

Truth: $X \sim \text{Uniform}(0,1)$, $Y = 1$



Sample of 3 points



We might naively predict $y=1$ when $x = [0.25, 0.5, 0.75]$, but while empirical risk is 0, it will clearly fail.

So we need to constrain our hypothesis space to ① functions that are "smooth" and "regular" in some way and ② easy to optimise over. That allows our function to extrapolate well into the space where we do not have data. e.g. we might want hypothesis space F to be continuous and differentiable.

Constrained Empirical Risk Minimisation. The ERM function in F is:

$$\hat{f} = \underset{f\in F}{\arg\min} \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

But note that this still falls short of the true target f^* in F:

$$f_F^* = \underset{f\in F}{\arg\min}\; E\,\ell(f(x_i), y_i)$$

4. Stochastic Gradient Descent

Empirical Risk Minimisation requires us to solve an optimisation problem. e.g. for least squares:

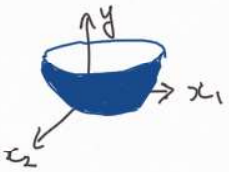
$F = \{ f(x) = w^T x \}$ How do we find w that minimises $\hat{R}(f)$?

Risk $R(f) = E\, l(f(x), y)$

$= E(w^T x - y)^2$

Empirical Risk $\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$

The typical ML approach is gradient descent. In a general setting, let the function $f: \mathbb{R}^d \mapsto \mathbb{R}$ ($x \in \mathbb{R}^d$, $y \in \mathbb{R}$) be differentiable. We want to find $x^* = \arg\min_{x \in \mathbb{R}^d} f(x)$.



The gradient is a vector in the input space, which gives us the direction which the input has to change to give us the steepest increase in output. e.g. on left, gradient $\in \mathbb{R}^2$. The gradient vector is the partial derivative of the output w.r.t each input dimension.

Algorithm: ① Initialise $x = 0$

② Repeat $x \leftarrow x + \widetilde{\eta} \nabla f(x)$ (stepsize, gradient)

③ Until stopping criterion reached.

How do we choose the step size η ("eta")? First, we must note that even if η is a constant, the step length can still vary because the norm/magnitude G of $\nabla f(x)$ is changing. E.g. as we approach the minima, G will decrease and we expect steps to get smaller.

Besides fixed step size, other strategies include backtracking line search, which adapts the step size.

Intuition on step size.

Scenario 1: Gradient descent keeps moving in same direction.

Scenario 2: Direction keeps changing.

Which scenario should we use a larger step size? Most probably scenario 1, there isn't much change in direction so we can take bigger steps.

Suggests that we should look at 2nd derivative, to see how much the gradient is changing, to choose our step size.

Convergence Theorem for Fixed Step size.

Suppose we have a function $f: \mathbb{R}^d \mapsto \mathbb{R}$ which is convex and differentiable, and ∇f gradient of f is **lipchitz continuous** with $L > 0$. Lipchitz continuity is the condition that the gradient cannot change too quickly, ie. $\frac{\|\nabla f(b) - \nabla f(a)\|}{\|b - a\|} \leq L \quad \forall\, x, y \in \mathbb{R}^d$

Theorem: Performing gradient descent with fixed step size $\eta \leq \frac{1}{L}$ will converge.

This result is useful for giving us a lower bound on what η to choose.

Stopping Criterion. A common way is to continue until the l_2 norm of gradient becomes small, ie: $\|\nabla f(x)\|_2 \leq \epsilon$ for some ϵ. This is reasonable because $\|\nabla f(x)\|_2 = 0$ at minimum. Another way in ML is to stop when validation error stops improving.

Minibatch Gradient Descent. For ML, we want to minimise empirical risk $\hat{R} = \overset{\text{mean}}{\frac{1}{n} \sum_{i=1}^{n}} \ell(f(x_i), y_i)$

Suppose that the loss function is differentiable w.r.t. parameter vector w. Then the gradient of the risk function is $\nabla_w \hat{R} = \overset{\text{mean}}{\frac{1}{n} \sum_{i=1}^{n}} \nabla_w \ell(f_w(x_i), y_i)$. Notice that the gradient is the **mean** of the gradient of individual losses. This means that to approximate the gradient, we can just take a sample of the individual losses.

Full gradient: $\nabla \hat{R}_n = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$

Minibatch gradient: $\nabla \hat{R}_N = \frac{1}{N} \sum_{j=1}^{N} \nabla_w \ell(f_w(x_j), y_j)$, $N \leq n$

The minibatch is an unbiased estimator of the full gradient. $E(\nabla \hat{R}_N) = \nabla \hat{R}_n$

Small N ⟷ Big N

Worse estimate, faster compute — Better estimate, slower compute

N of 1 is called **stochastic gradient descent**. $N = 32$ is a good default, with $N \gg 10$ we get a compute speedup.

SGD with fixed step size works well in practice but there is no theorem to guarantee convergence.

To guarantee convergence, we need to decrease step sizes following **Robbins-Monro conditions:**

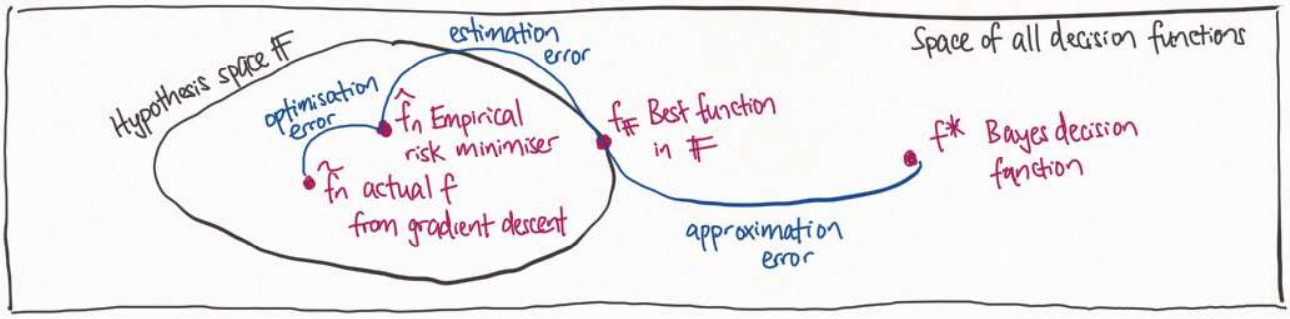
Let η_t be step size at time t. Then we need:

$$\sum_{t=1}^{\infty} \eta_t^2 < \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} \eta_t = \infty$$

This means that η_t should be decreasing, but not too fast or slowly.

$\eta_t = \frac{1}{t}$ works but $\eta_t = \frac{1}{\sqrt{t}}$ decreases too slowly.

5. Excess Risk Decomposition



Approximation error is the error from constraining the hypothesis space $\mathbb{F}$.

Approx. error $= R(f_{\mathbb{F}}) - R(f^*)$. If we increase $\mathbb{F}$ and consider more functions, approx. error must strictly decrease (or stay the same).

Estimation error is the error from fitting the function from a finite data sample.

$R(\hat{f}_n) - R(f_{\mathbb{F}})$. If we increase $\mathbb{F}$, we might expect to overfit more and estimation error might ↑.

Note that $R(\hat{f}_n)$ is a random variable with a distribution ⛰, because each time we draw a sample D, we get a new number for $R(\hat{f}_n)$.

Optimisation error is the error from the optimisation algorithm.

$R(\tilde{f}_n) - R(\hat{f}_n)$. We sometimes obsess over minimising this during training.

Excess risk for a decision function f is the excess risk when compared to bayes function f^*.

We can write:
$$\text{Excess Risk}(\hat{f}_n) = \underbrace{R(\hat{f}_n) - R(f_{\mathbb{F}})}_{\text{estimation error}} + \underbrace{R(f_{\mathbb{F}}) - R(f^*)}_{\text{approx. error}}$$

We see that there is a tradeoff between estimation and approximation error. As we increase $\mathbb{F}$, $R(f_{\mathbb{F}})$ decreases and so does approximation error, but estimation error rises. So we see that we need to choose a suitable $\mathbb{F}$, given the amount of data we have, to control estimation error.

We also have:
$$\text{Excess Risk}(\tilde{f}_n) = \underbrace{R(\tilde{f}_n) - R(\hat{f}_n)}_{\text{optimisation error}} + \underbrace{R(\hat{f}_n) - R(f_{\mathbb{F}})}_{\text{estimation error}} + \underbrace{R(f_{\mathbb{F}}) - R(f^*)}_{\text{approx. error}}$$

We see that we don't need to obsess over optimisation error, because at that point estimation and approximation error probably dominate. This is why simple gradient descent works well in practice.

6. L1 and L2 Regularisation

Regularisation loosely means any method that makes a model fit training data less well in hopes of generalising better. Regularisation allows us to increase or decrease our hypothesis space to an arbitrary degree.

Consider a nested sequence of hypothesis spaces: $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \dots \subset \mathcal{F}_n$ where each space is a subset of the next. Suppose we have a complexity measure function Ω which maps each space $\mathcal{F} \mapsto [0, \infty]$ ie a real number. We define each space $\mathcal{F}_r = \{ f \in \mathcal{F} \mid \Omega(f) \leq r \}$, ie comprising all functions with complexity $\leq r$. Then, by adjusting r we get a sequence of nested hypothesis spaces. r becomes a hyperparameter we can tune to find the "right" hypothesis space.

For linear decision functions $f: x \mapsto w^T x$, we can construct complexity measures by constraining w:

l_0: number of non-zero $w_i \leq r$

l_1: $\|w\|_1 = \sum_{i=1}^{d} |w_i| \leq r$

l_2: $\|w\|_2^2 = \sum_{i=1}^{d} w_i^2 \leq r$

This method of constraining w directly is called ivanov regularisation. In general, for ERM, we can write the objective as: $\tilde{f}_n = \underset{f \in \mathcal{F}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$ s.t. $\Omega(f) \leq r$

Alternatively, we have tikhonov regularisation which constrains w indirectly through a penalty. The objective is: $\tilde{f}_n = \underset{f \in \mathcal{F}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) + \lambda \Omega(f), \lambda \geq 0$

It turns out that for many loss functions l and complexity measure Ω, both the above objectives are equivalent, meaning that any solution f^* we obtain under one formulation, we can also obtain under the other. Lagrangian duality theory will give us the conditions for equivalence.

Now we consider the specific form of linear least squares regression.

Hypothesis space $\mathbb{H} = \{ w^T x \mid w \in \mathbb{R}^d \}$

Loss $\ell = (y - \hat{y})^2$

The unconstrained objective : $\hat{w} = \underset{w}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$

Ridge Regression : $\hat{w} = \underset{w}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_2^2 \quad , \lambda \geq 0$

Lasso Regression : $\hat{w} = \underset{w}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_1 \quad , \lambda \geq 0$

Apply regularisation, we get lasso and ridge regression.

Tweaking regularisation intensity. As we tweak the complexity threshold r, we get different solutions.

Define $\hat{w}_r = \underset{\|w\|_2^2 \leq r^2}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$, ie. soln under different r.

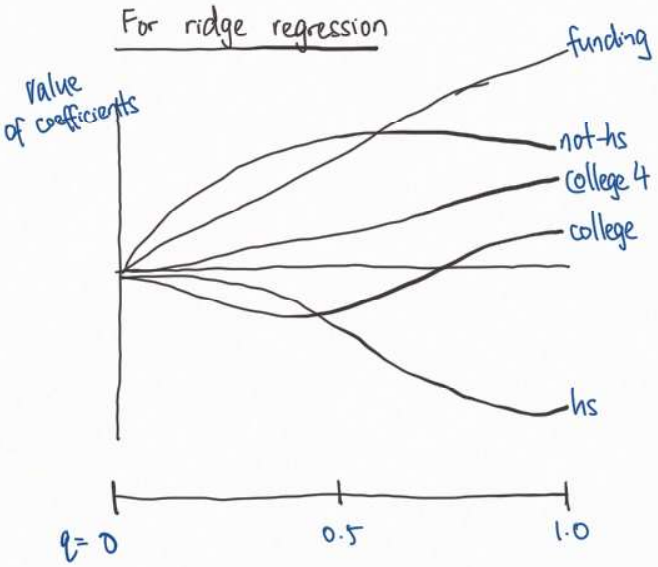
and $\hat{w} = \hat{w}_\infty$, ie. unconstrained soln.

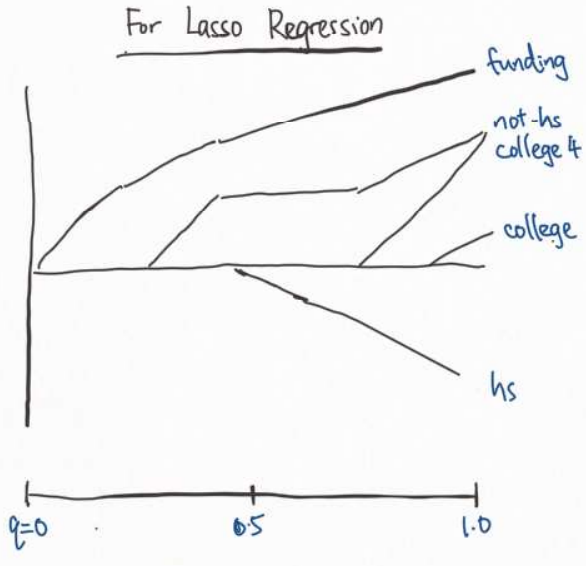
Then $q = \|\hat{w}_r\|_2 / \|\hat{w}\|_2$ gives us a measure of regularisation intensity running from 0 to 1.

As r varies : $r=0 \longrightarrow r=\infty$

So q varies : $q=0 \longrightarrow q=1$

constrained — unconstrained

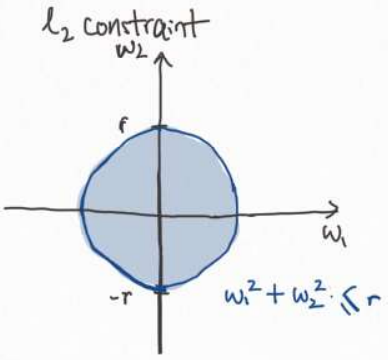


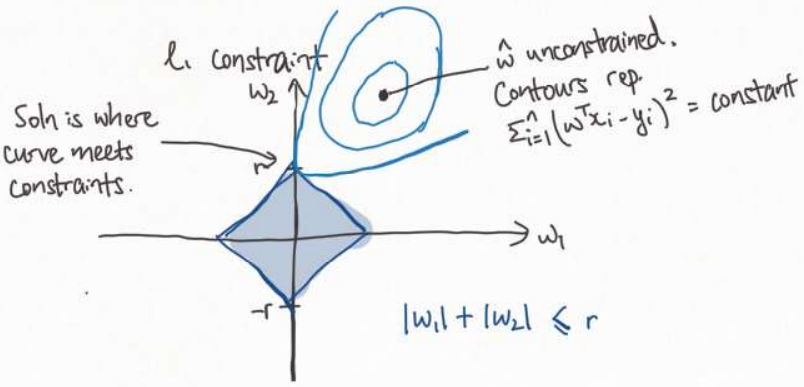


Lasso tends to give sparse solutions.

The feature sparsity of lasso is useful for reducing memory usage. But why does lasso give sparse solutions? To illustrate, we can look at the parameter space.

Let $f(x) = w_1 x_1 + w_2 x_2$

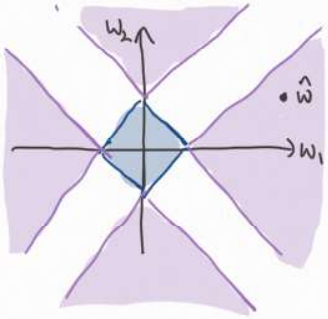




Why are the contour lines ellipsoids? We can show by writing the unconstrained objective: $\hat{R}_n(w) = \sum_{i=1}^{n}(w^T x_i - y_i)^2$ and show that $\hat{R}_n(w) = \frac{1}{n}(w-\hat{w})^T X^T X (w-\hat{w}) + \hat{R}_n(\hat{w})$

This formula tells us that it is an ellipsoid around $\hat{w}$.

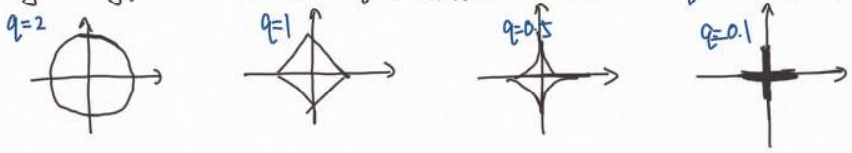
Sparse Regions



If $\hat{w}$ is in purple region, soln will be sparse.

Otherwise, soln will be on the edge.

More generally, we can have ℓ_q constraint where $\|w\|_q^q = |w_1|^q + |w_2|^q \leq r$



As we lower q, we get more sparse solutions.

Solving the lasso. The lasso objective is:

$$\hat{w} = \arg\min_{w} \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_1$$

Problem: $\lambda\|w\|_1$ is not differentiable, so we cannot use gradient descent.

To address this, we need to break up $\|w\|_1$ into positive and negative parts.

Let $a = w \times I(w \geq 0)$ positive part

$b = w \times I(w \leq 0)$ negative part $\quad a, b \geq 0$

It follows that: $a + b = |w|$

$a - b = w$

Rewriting the objective:

$$\min_{a,b,w} \sum_{i=1}^{n} ((a-b)x_i - y_i)^2 + \lambda(a+b)$$

s.t. $a \geq 0, \; b \geq 0$ — ①

$a - b = w$ — ②

$a + b = |w|$ — ③

We can remove constraint 3. Consider $a - b =$ constant k, i.e. $\sum_{i=1}^{n} ((a-b)x_i - y_i)^2$ unchanged.

Now propose: $a' = a - \min(a,b)$ and $b' = b - \min(a,b)$

$$a' - b' = a - \min(a,b) - b + \min(a,b) = a - b = k$$

Thus, we can keep lowering a', b' until one of them is 0. So $a+b$ must $= |k|$

We can also remove constraint 2: It does not actually constrain a, b, because if $w \in \mathbb{R}$, then a and b can take on any values in $\mathbb{R}$ anyway.

So the simplified objective:

$$\min_{a,b} \sum_{i=1}^{n} ((a-b)x_i - y_i) + \lambda(a+b)$$

$$a \geq 0, \; b \geq 0.$$

This can be solved more easily. Algorithms used to solve are projected SGD and coordinate descent.

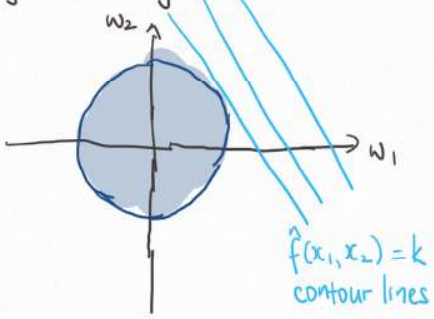
7. Lasso, Ridge, Elasticnet

Sometimes we encounter linearly dependent features. For e.g., suppose we estimate: $\hat{f}(x) = 4x_1$. Suppose we add a new feature $x_2 = x_1$. What happens?

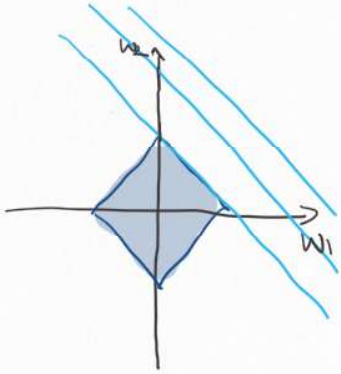
w_1	w_2	ℓ_1: $\|w\|_1$	ℓ_2: $\|w\|_2^2$
4	0	4	16
2	2	4	8
1	3	4	10

Notice that ℓ_1 norm makes no distinction between the different w_1 and w_2. However, ℓ_2 norm prefers to distribute the weights evenly bet. w_1 and w_2.

Pictorally, the contour lines of $\hat{f}$ is not an ellipse when we get linearly dependent features. Instead, we get a straight line.



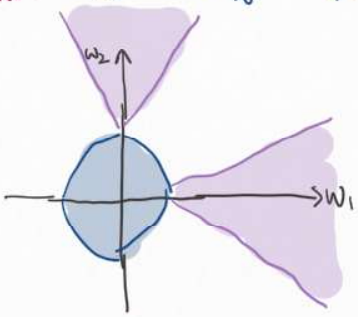
ℓ_2 norm distributes weight bet. w_2 and w_1.



ℓ_1 norm gives us multiple solutions.

Thus we see that lasso deals poorly with highly correlated features. It can be highly unstable and all the weight swinging bet. x_2 and x_1 when data changes slightly. ElasticNet comes in to retain the sparsity of lasso but also produce more stable models.

ElasticNet: $\hat{w} = \arg\min_w \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$



Purple regions is where we get sparse solutions under elasticnet.

8. Loss Functions

There are other losses than the square loss. Loss is a function of y and $\hat{y}$.

But a specific type is distance-based loss, which have 2 properties:

① They depend only on the residual $\quad \ell(y, \hat{y}) = \psi(y - \hat{y})$

② Loss is 0 when residual is 0. $\quad \psi(0) = 0$

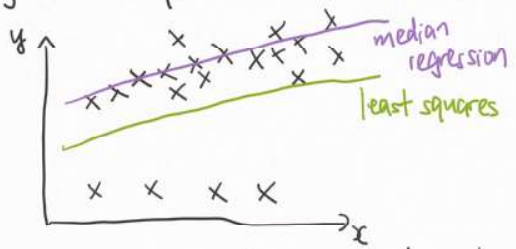
Note that it follows that such losses are translation invariant. This means that adding constant to both y and $\hat{y}$ results in the same loss.

Some distance based losses: ① Square (ℓ_2) loss : $\ell(r) = r^2$

② ℓ_1 loss : $\ell(r) = |r|$

③ Huber loss $\quad \ell(r) = \begin{cases} \frac{1}{2}r^2 & \text{for } |r| \leq \delta \\ \delta(|r| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$

Problems with ℓ_1 / ℓ_2 loss: ℓ_2 loss is susceptible to outliers - they amplify the loss a lot so regression will try to cater for these outliers. ℓ_1 loss is more robust to outliers, but is not differentiable.



Huber loss is both robust and differentiable. It is quadratic for $|r| \leq \delta$, and linear otherwise. It is designed to have derivatives match at $r = \delta$.

to draw: graph of loss functions.

Loss functions for classification.

Typically for classification, we would use $f(x)$ to generate a score ~~ER~~. We then use thresholds such as $f(x) > 0$ to predict 1 or -1.

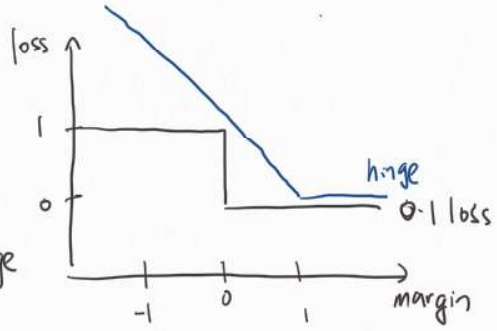
The margin is $yf(x)$, which tells us how correct we are.

The natural loss to use is 0-1 loss: $l = I(f(x) \neq y)$. The empirical risk is: $\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} I(f(x) \neq y)$. However, this is non-differentiable and discontinuous. It has been proven that this optimisation would be NP-hard.

More commonly, we have margin-based losses, which depend on $m = yf(x)$.

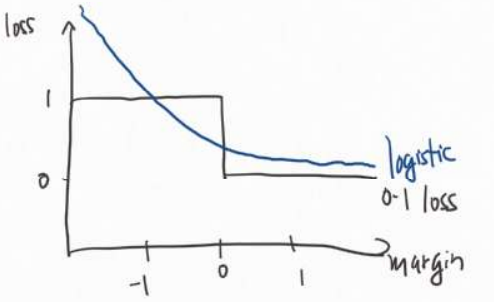
Hinge loss: $l = \max(1-m, 0)$

Hinge loss is an upper bound for 0-1 loss, which gives some reason for optimising hinge loss.



Logistic loss: $l = \log(1 + e^{-m})$

Logistic loss never stops penalising (go to 0), even if we are very confident.

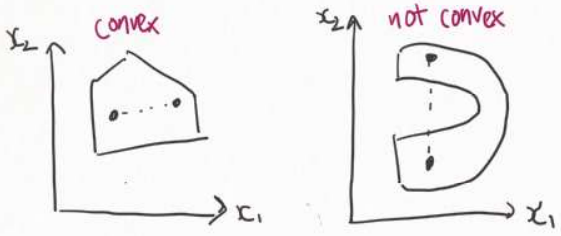


9. Lagrangian Duality and Convex Optimisation

Lagrangian duality gives us conditions for which constrained and penalised ERM are identical.

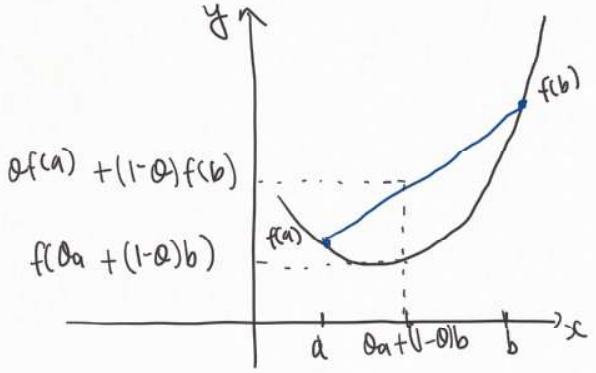
Convex Sets. A set C is convex if for any $x_1, x_2 \in C$ and $0 \leq \theta \leq 1$ we have $\theta x_1 + (1-\theta)x_2 \in C$.

ie. line segment lies inside C.



Convex functions. A function $f: \mathbb{R}^d \mapsto \mathbb{R}$ is convex if domain of f is a convex set and for any two points $a, b \in$ domain f and $0 \leq \theta \leq 1$:

$$f(\underbrace{\theta a + (1-\theta)b}_{\text{Function}}) \leq \underbrace{\theta f(a) + (1-\theta) f(b)}_{\text{Line Segment}}$$



Concave is the negative of convex.

Some examples of convex functions:

- Linear $ax + b$ is both concave and convex
- Modulo $|x|^p$ for $p \geq 1$ is convex
- Exponential e^{ax} is convex
- Norm All norms are convex
- Max $\max(x_1, \dots x_n)$ is convex

strictly convex - a function is strictly convex if the line segment connecting any two points lies strictly above the graph.

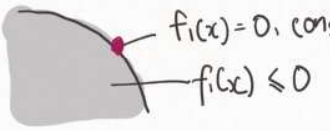
Convexity implies that a local minimum is the global minimum.

Strict convexity implies that a local minimum is also the unique global minimum.

The general optimization problem is: $\min f_0(x)$ s.t. $f_i(x) \leq 0 \quad i = 1, \dots, m$

The feasible set is the set of points satisfying all the constraints. A point x in the feasible set is a feasible point. A constraint is active at x if x is feasible and $f_i(x) = 0$.

The optimal value $p^* = \inf(f_0(x) \mid \text{feasible})$



Lagrangian duality is another way to express the optimization problem. Why?

The lagrangian is: $L(x, \lambda) = \underset{\text{obj.}}{f_0(x)} + \underset{\text{constraints}}{\sum_{i=1}^{m} \lambda_i f_i(x)} \qquad \lambda_i \geq 0$

① Supremum of Lagrangian returns constrained objective.

$$\sup_{\lambda \geq 0} L(x, \lambda) = \sup_{\lambda \geq 0} \left[f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) \right]$$

If all constraints are satisfied, supremum will return $f_0(x)$. For any $f_i(x) < 0$, λ_i will be set to 0 to make this function larger. If any constraint is violated, ie. $f_i(x) > 0$, then the supremum will return ∞.

② Infimum of sup returns us optimal solution.

$$p^* = \inf_{x} \sup_{\lambda \geq 0} L(x, \lambda) \quad \text{—— primal form.}$$

Due to inf, so long as there is one feasible point, we will choose that point over ∞. So we are assured that all constraints will be satisfied.

Lagrangian dual problem swaps the inf and sup.

$$d^* = \sup_{\lambda \geq 0} \inf_{x} L(x, \lambda)$$

Weak duality tells us that the dual d^* forms a lower bound for the primal p^*.

Theorem: For any $f: W \times Z \mapsto \mathbb{R}$,

$$\sup_{z} \inf_{w} f(w, z) \underset{\text{sandwich}}{\leq} \inf_{w} \sup_{z} f(w, z)$$

Proof: Choose $w_0 \in W$ and $z_0 \in Z$.

$$\inf_{w} f(w, z_0) \leq f(w_0, z_0) \leq \sup_{z} f(w_0, z)$$

$$\inf_{w} f(w, z_0) \leq \sup_{z} f(z, w_0) \quad \text{true } \forall\, w_0, z_0$$

$$\sup_{z} \inf_{w} f(w, z) \leq \inf_{w} \sup_{z} f(w, z) \quad // \text{ just add sup, inf.}$$

This is a general proof, showing that $p^* \geq d^*$ always.

Weak Duality gives us reason to solve the dual problem instead, as it forms a lower bound for the original problem. The dual problem is often easier to solve due to easier constraints.

Dual problem: $\max\ g(\lambda) = \inf_x L(x,\lambda) \quad s.t.\ \lambda \geq 0$

While weak duality always holds, in the case of convex optimisation, strong duality $p^* = d^*$ often holds as well. The additional conditions required are called constraint qualifications.

Slater's constraint qualification: strong duality holds so long as there is one point x which is strictly feasible, ie. $f_i(x) < 0 \ \forall i$.

If the constraints $f_i(x)$ are all affine functions, then all that is required is one point that is feasible, ie. $f_i(x) \leq 0$.

Strong Duality implies Complementary Slackness.

Theorem: If $p^* = d^*$ then $\lambda_i^* f_i(x^*) = 0 \quad \forall i = 1, \dots, m$

Proof:

$$\begin{aligned} p^* = f_0(x^*) &= d^* \\ &= \inf_x L(x, \lambda^*) \\ &\leq L(x^*, \lambda^*) \\ &= f_0(x^*) + \sum_{i=1}^m \underbrace{\lambda_i^*}_{\geq 0} \underbrace{f_i(x^*)}_{\leq 0} \\ &\leq f_0(x^*) \end{aligned}$$

$$f_0(x^*) \leq f_0(x^*) + \sum_{i=1}^m \lambda_i^* f_i(x^*) \leq f_0(x^*)$$

ie. $\sum_{i=1}^m \lambda_i^* f_i(x^*) = 0$

ie. $\lambda_i^* f_i(x^*) = 0 \quad \forall i = 1, \dots, m$ //

What does this mean? That when we have strong duality, either λ_i^* or $f_i(x^*)$ (or both) will be 0 for every constraint.

Lecture 10 - Support Vector Machines

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \underbrace{\left(1 - y_i f(x_i)\right)_+}_{\text{margin}} + \lambda \|w\|^2$$

Rewrite objective: $\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \max(0, 1 - y_i[w^T x_i + b])$

Not differentiable, can we reformulate?

SVM problem equivalent:

$$\min \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } \xi_i \geq \max(0, 1 - y_i[w^T x_i + b])$$

Moving max into constraints allows us to break it up.

Because min will always give us $\xi_i = \max(0, 1 - y_i[w^T x_i + b])$.

Rewrite again in standard form:

$$\min \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \xi_i$$

$$\text{s.t. } -\xi_i \leq 0 \quad \text{for } i = 1, \ldots, n$$

$$1 - y_i[w^T x_i + b] - \xi_i \leq 0 \quad \text{for } i = 1, \ldots, n$$

Note that we are optimising over w, b and ξ_i. y_i, x_i are data constants.

Qn: is the objective convex? Yes, it is quadratic in objective and affine in constraints.

Now we write the lagrangian. Let λ_i, α_i be lagrange multipliers.

$$L(w, b, \xi, \lambda, \alpha) = \underbrace{\frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n} \xi_i}_{\text{obj}} + \underbrace{\sum_{i=1}^{n} \alpha_i (1 - y_i[w^T x_i + b] - \xi_i) + \sum_{i=1}^{n} \lambda_i(-\xi_i)}_{\text{Constraints}}$$

$$= \frac{1}{2} w^T w + \sum_{i=1}^{n} \xi_i \left(\frac{c}{n} - \alpha_i - \lambda_i\right) + \sum_{i=1}^{n} \alpha_i (1 - y_i[w^T x_i + b])$$

Now we have primal and dual.

$$p^* = \inf_{w,b,\xi} \sup_{\alpha, \lambda \geq 0} L(w, b, \xi, \alpha, \lambda)$$

$$\geq \sup_{\alpha, \lambda \geq 0} \inf_{w,b,\xi} L(w, b, \xi, \alpha, \lambda) = d^*$$

Do we have strong duality? Yes, because it is ① convex, ② affine constraints, and we have at least ③ one feasible point, e.g. $w = 0$, $b = 0$, $\xi_i = 1$. which satisfies constraints.

So we know $p^* = d^*$.

Now we look at the dual function (i.e. inf of L).

$$g(\alpha, \lambda) = \inf_{w,b,\xi} L(w, b, \xi, \alpha, \lambda)$$

$$= \inf_{w,b,\xi} \underbrace{\frac{1}{2} w^T w}_{①} + \sum_{i=1}^n \xi_i \underbrace{\left(\frac{c}{n} - \alpha_i - \lambda_i\right)}_{②} + \sum_{i=1}^n \alpha_i \underbrace{\left(1 - y_i [w^T x_i + b]\right)}_{③}$$

How do we find inf of this convex function? Simple, take derivatives to find global min.

$$\partial_w L = w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \quad \longrightarrow \quad w = \sum_{i=1}^n \alpha_i y_i x_i$$

$$\partial_b L = -\sum_{i=1}^n \alpha_i y_i = 0 \quad \longrightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$

$$\partial_{\xi_i} L = \frac{c}{n} - \alpha_i - \lambda_i = 0 \quad \longrightarrow \quad \alpha_i + \lambda_i = \frac{c}{n}$$

Sub these back into dual function:

$$g(\alpha, \lambda) = \frac{1}{2} \left(\sum_{i=1}^n \alpha_i y_i x_i\right)^2 = \frac{1}{2} \sum_{i,j=1}^n \left(\alpha_i \alpha_j y_i y_j x_i^T x_j\right) \quad — ①$$

$$+ \sum_{i=1}^n \xi_i \left(\frac{c}{n} - \alpha_i - \lambda_i\right) = 0 \quad — ②$$

$$+ \sum_{i=1}^n \alpha_i \left(1 - y_i [w^T x_i + b]\right) = \sum_{i=1}^n \alpha_i - \sum_{i,j=1}^n \left(\alpha_i \alpha_j y_i y_j x_i^T x_j\right) \quad — ③$$

$$= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \left(\alpha_i \alpha_j y_i y_j x_i^T x_j\right)$$

(SVM dual problem)

The optimisation problem is now:

$$\sup_{\alpha} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_j^T x_i$$

$$\text{s.t.} \quad \sum_{i=1}^n \alpha_i y_i = 0$$

$$\alpha_i + \lambda_i = \frac{c}{n} \quad \text{for all } i$$

$$\alpha_i, \lambda_i \geq 0 \quad \text{for all } i$$

Since $\lambda_i \geq 0$, $\alpha_i \leq \frac{c}{n}$

i.e. $\alpha_i \in [0, \frac{c}{n}]$

- Interesting insight – when we have dual optimum α_i^*, primal soln is $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$, which means that it is a linear combination of x_i and y_i. We say w^* is in the "span of the data".
- Another insight – when regularising heavily, c is small. This means α_i will be close to 0, since $\alpha_i \leq \frac{c}{n}$. This means that any input x_i will have small impact on soln w^*, i.e. more robust

Next, we can also get some insights from Complementary Slackness equations.

Define optimal function $f^*(x_i) = w^* x_i + b^*$

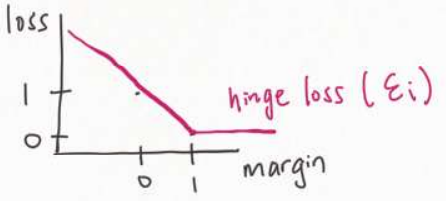
Margin is $y_i f^*(x_i)$,

incorrect classification when $m_i \leq 0$.

Margin error $m_i < 1$

on the margin $m_i = 1$

Good side $m_i > 1$



The "slack variable" $\xi_i^* = \max(0, 1 - y_i f^*(x_i))$ is the hinge loss on x_i, y_i.

From the graph, when $\xi_i^* = 0 \Rightarrow y_i f^*(x_i) \geq 1$ — ③

Recall Complementary slackness – lagrange multiplier × constraint = 0.

SVM problem has strong duality, so we must have Complementary slackness.

From the primal form:

$$\alpha_i^* (1 - \underbrace{y_i f^*(x_i)}_{\text{margin}} - \xi_i^*) = 0 \quad \text{— ① Complementary Slackness}$$

$$\lambda_i^* \xi_i^* = 0$$

sub $\lambda_i^* = \frac{C}{n} - \alpha_i^*$: $\left(\frac{C}{n} - \alpha_i^*\right) \xi_i^* = 0$ — ②

Implications: Using the ① and ② and ③ we can show:

If $\alpha_i^* = 0 \Rightarrow \xi_i^* = 0,\ y_i f^*(x_i) \geq 1$

$\alpha_i^* \in (0, \frac{C}{n}) \Rightarrow \xi_i^* = 0,\ y_i f^*(x_i) = 1$

$\alpha_i^* = \frac{C}{n} \Rightarrow \xi_i^* \geq 0,\ y_i f^*(x_i) \leq 1$

If $y_i f^*(x_i) < 1 \Rightarrow \xi_i^* > 0,\ \alpha_i^* = \frac{C}{n}$

$y_i f^*(x_i) = 1 \Rightarrow \xi_i^* = 0,\ \alpha_i^* \in [0, \frac{C}{n}]$

$y_i f^*(x_i) > 1 \Rightarrow \xi_i^* = 0,\ \alpha_i^* = 0$

Whenever margin is good, $\alpha_i^* = 0$. So we tend to have sparse solutions.

Recall $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$ So w^* will be linear combi of sparse vector. The entries which are non-zero are left standing, called support vectors.

Last piece — what about b^*?

Suppose for some i, $\alpha_i^* \in (0, \frac{C}{n})$

$\Rightarrow \quad \xi_i^* = 0 \;, \quad y_i f^*(x_i) = 1$

$\Rightarrow \quad y_i [x_i^T w^* + b^*] = 1$

$\Rightarrow \quad x_i^T w^* + b^* = y_i$ multiply y_i both side, $y_i \in \{-1, 1\}$

$b^* = y_i - x_i^T w^*$

What does this mean? No matter which i, this relationship will work!

In real life, due to numerical error, we may take the mean value over all the valid examples, i.e. $b^* = \text{mean}\{ y_i - x_i^T w^* \mid \alpha_i^* \in (0, \frac{C}{n}) \}$

Lecture 11: Subgradient Descent

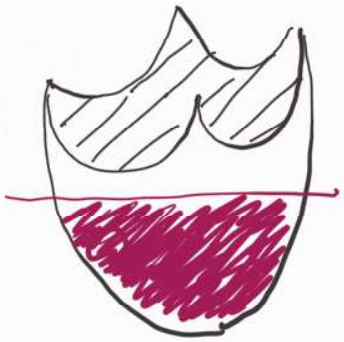
To tackle non-differentiable problems, directly.

Sublevel Sets. Recall convex functions and convex sets.

Qn: Why did we write $f_i(x) \leq 0$? This is required so that constraint set is convex.

Let $f : \mathbb{R}^d \to \mathbb{R}$. Then a level set is the set of points $x \in \mathbb{R}^d$ where $f(x) =$ some constant c. A sublevel set is $x \in \mathbb{R}^d$ where $f(x) \leq c$.

Theorem If f is convex, then all sublevel sets are convex.



e.g. this graph in $\mathbb{R}^3$ is convex.
Chop it off and the bottom part is still convex.
Contour plot shows this as well.

Another fact - intersection of convex sets is convex. In general, superlevel set $\{x \mid f(x) > c\}$ and level set $\{x \mid f(x) = c\}$ are not convex.

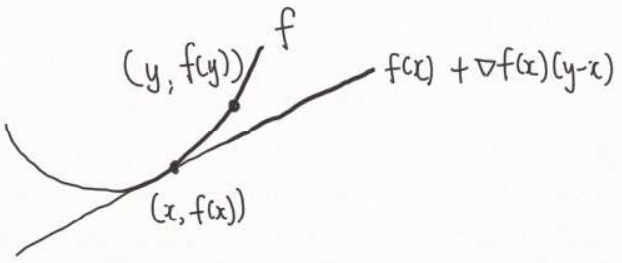
This is why standard form is

$$\min f_0(x)$$
$$\text{s.t. } f_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m.$$

where f_i is convex.

Each constraint set $\{x \mid f_i(x) \leq 0\}$ is a sublevel set and convex.
The feasible set $\{x \mid f_i(x) \leq 0, i = 1, \ldots, m\}$ is intersection of all constraint set and convex.
Another way to write feasible set is $x \in C$ where C is a convex set.

Next, look at differentiable function to draw conclusion for non-differentiable ones.
General idea of gradient descent is to predict $f(y)$, given $f(x)$ and $\nabla f(x)$ (gradient).



If f is differentiable, then:

$$f(y) \approx f(x) + \nabla f(x)(y - x)$$

ie. first order approximation.

If f is also convex (and differentiable), then the gradient becomes a global lower bound for f.

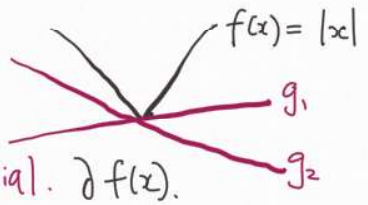
ie. $f(y) \geq f(x) + \nabla f(x)(y - x)$ for all x, y interesting that we have global info given just local info. Note that if we find a point where $\nabla f(x) = 0$, then $f(y) \geq f(x)$ for all y. This means that this point is a global minimum.

Now more to non-differentiable (but convex) functions. Use idea of global underestimators.

Find set of global underestimators, called subgradients.

Let $f: \mathbb{R}^d \to \mathbb{R}$. $g \in \mathbb{R}^d$ is a subgradient of f at x if for all z:

$$f(z) \geq f(x) + g^T(z - x)$$



At each x, there can be multiple g.

Set of all subgradients at x is called subdifferential. $\partial f(x)$.

f is subdifferentiable at x if $\exists$ at least one g at x.

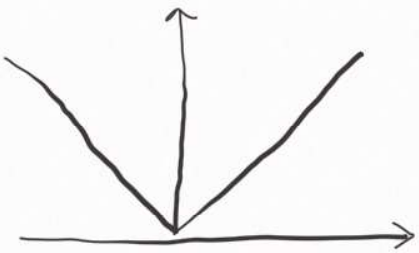
Some facts: ① If f convex and differentiable, $\partial f(x) = \{\nabla f(x)\}$ ie. one element

② At any x, either 0, 1 or $\aleph$ number of subgradients.

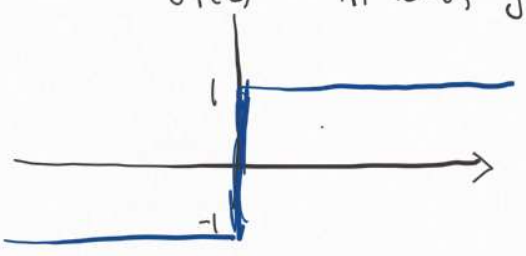
③ $\partial f(x) = \emptyset \implies f$ is not convex

④ when $0 \in \partial f(x)$, x is a global minimiser.

Consider $f(x) = |x|$



$\partial f(x)$ At $x = 0$, $g \in [-1, 1]$.



2D example.

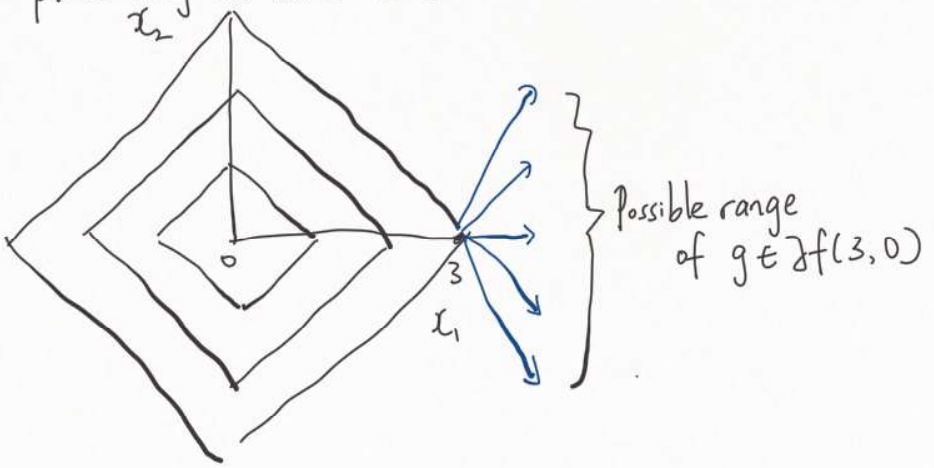
Consider $f(x_1, x_2) = |x_1| + 2|x_2|$. Find $\partial f(3, 0)$.

$\partial f(3, 0) = (1, [-2, 2])$ or $\{(1, g_2) \mid g_2 \in [-2, 2]\}$

g is a subgradient $\in \partial f(3, 0)$. Note: g points in a direction in $\mathbb{R}^2$. But it describes a plane in $\mathbb{R}^3$.

The plane of $h(x_1, x_2) = f(3, 0) + g^T(x_1 - 3, x_2 - 0)$ is a global underestimator of $f(x_1, x_2)$.

Contour plots may be easier to see.

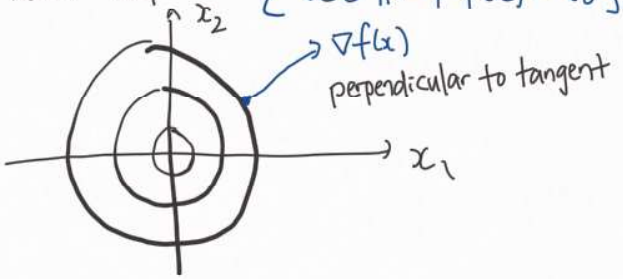


Theorem: For a continuously differentiable f:

If $\nabla f(x_0) \neq 0$, then $\nabla f(x_0) \perp$ to level set at x_0

where level set $= \{ x \in \mathbb{R}^d \mid f(x) = x_0 \}$

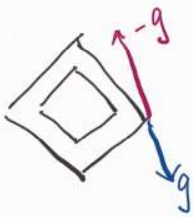
e.g.



How about for non-differentiable f?

Let f have a subgradient g at x_0. We cannot just do gradient descent by moving against the gradient.

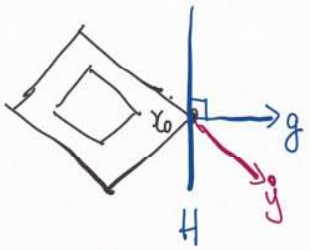
e.g.



We see that both g and $-g$ are ascent directions., not descent.

Theorem: Hyperplane H at x_0 must support level set S at x_0.

i.e. H contains x_0, and all of S lies on one side of H.



Proof: For any y, $f(y) \geq f(x) + g^T(y-x)$ by defn

Try y which is on same side as g.

$$g^T(y-x) > 0$$

$$\therefore \quad f(y) > f(x).$$

So y lies outside of level set S.

So all of S lies on H or to the $-g$ side of H.

No guarantee that $-g$ is descent direction. But can still do.

Algorithm: Subgradient Descent.

Repeat: $x := x_0 - tg$ where $t > 0$ and $g \in \partial f(x_0)$. (t: step size)

But does it work?

Theorem: Subgradient descent gets us closer to minimum.

Suppose f is convex but not differentiable.

① Let $x = x_0 - tg$ where $g \in \partial f(x_0)$ take a step in $-g$ direction

② Let z be any point where $f(z) < f(x_0)$

③ Then for small enough t, $\|x - z\|_2 < \|x_0 - z\|_2$

dist after step — dist before step.

Proof:

$$\|x - z\|_2^2$$
$$= \|x_0 - tg - z\|_2^2 \quad \text{by defn}$$
$$= \|x_0 - z\|_2^2 - 2tg^T(x_0 - z) + t^2\|g\|_2^2 \quad \text{expand the square}$$

this piece

By defn of subgradient: $f(z) \geq f(x_0) + g^T(z - x_0)$

$$f(z) - f(x_0) \geq g^T(z - x_0)$$
$$f(x_0) - f(z) \leq g^T(x_0 - z)$$

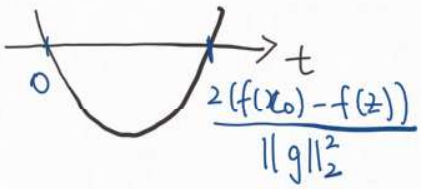
Therefore: $\|x_0 - z\|_2^2 - 2t[g^T(x_0 - z)] + t^2\|g\|_2^2$

$$\leq \|x_0 - z\|_2^2 - 2t(f(x_0) - f(z)) + t^2\|g\|_2^2$$

We want this to be -ve so that x is closer to z than x_0.

It is a quadratic function of t. So it is negative for

$$t \in \left(0, \frac{2(f(x_0) - f(z))}{\|g\|_2^2}\right)$$



In practice, we do not know z, but at least we know that if t is small, we will get closer to the minimum.

12. FEATURE EXTRACTION

Input space $\mathcal{X}$, need a mapping function $\phi(x)$ to map it to $\mathbb{R}^d$.

e.g. detect if a string is an email address. A feature could be: last 3 char is "com."

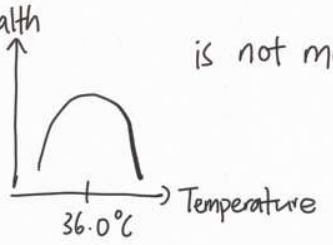
But requires expert knowledge. Instead, just make features of all possible combis, let regularisation handle it.

For dense vectors (e.g. numeric), array representation $[1.2, 3.6 \ldots]$

For sparse (e.g. one-hot), map or dictionary is good. $\{\text{"contains\_@"}: 1\}$

- Features not in map have default value 0. Saves space and speed.

Linear models rely a lot on how we encode the features. If not, have issues.

① Monotonicity. Linear always ↗ or ↘. e.g. 

is not monotonic

Expert encoding: $\phi(x) = [(x-36)^2]$

Better way: $\phi(x) = [1, x, x^2]$

Let linear model figure it out, more flexible.

② Saturation. e.g. user searches for "handbag". Model scores each product for relevance based on $N(x)$, ie. people who bought product.

But is 1,000 really 10x more relevant than 100?

Encoding: $\phi(x) = [\log(N(x)), \text{sigmoid etc.}]$

Note that base of log is not impt. because they are linear function of one another.

e.g. $\log_2 x = \frac{\log_{10}(x)}{\log_{10}(2)}$ — scalar.

Alternative: Discretization Bin the values. $\phi(x) = [I(5 \leq N(x) < 10, \ldots]$

Or use one sided buckets $\phi(x) = [I(N(x) < 10, N(x) < 100, \ldots]$.

For bins that don't add much to improving the loss, regularisation will make them 0.

③ Interactions. e.g. don't just want weight or height, but their ratio.

Use $\phi(x) = [1, h(x), w(x), h(x)w(x)]$

A predicate is a T/F input. e.g isSleeping, isDriving. Interaction term correrponds to AND condition.

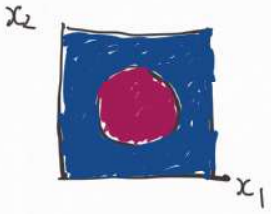
What does linearity mean?

Hypothesis space $\mathcal{F} = \{ w^T \phi(x) \mid w \in \mathbb{R}^d \}$ where $\phi(x)$ is non-linear feature map.

Linear in w and $\phi(x)$. Not linear in x.

Ideally we want a non-linear $f(x)$ for expressivity. But we also want to easily optimise $f^*(x)$. So this is a good compromise.

Geometric example with nonlinear boundary.



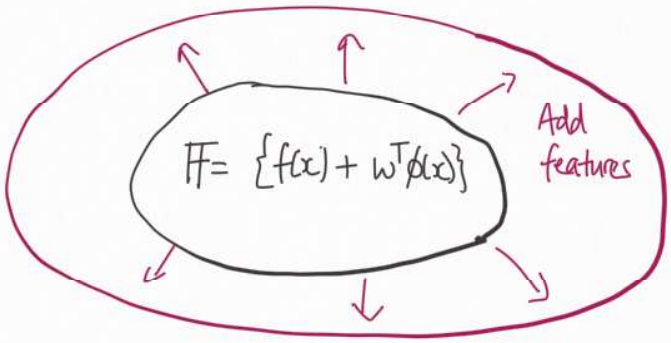
If we just use linear features $\phi(x) = [x_1, x_2]$, no hope.

Consider the level set $f(x_1, x_2) = c$.

Circle is $f(x_1, x_2) = \left(\frac{x_1 - a}{r}\right)^2 + \left(\frac{x_2 - b}{r}\right)^2$

All we need is $\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$

Think of feature extraction as expanding the hypothesis space of a linear $f(x)$.



Note: for regularised regression, scale of data matters, because w is penalised when large. So must always scale first.

Can we arbitrarily add features and leave it to regularisation to select?

Generally yes, but we do run the risk of finding strong features out of spurious correlation.

13. KERNEL METHODS

Linear methods reliant on feature engineering. Kernel methods are expressive in a different way without suffering heavy computation cost.

For linear, $\mathcal{F} = \{ w^T \phi(x) + b \mid w \in \mathbb{R}^d, b \in \mathbb{R} \}$

To get expressivity, using linear models, we need high dimension feature vectors.

e.g. if $x = (1, x_1, \dots x_d) \in \mathbb{R}^{d+1}$

Suppose we want all monomials to degree M: $x_1^{p_1} \dots x_d^{p_d}$ with $p_1 + \dots + p_d \leq M$

How many features? Using stars and bars, equivalent problem is choosing $d-1$ separators out of $M+d-1$ elements, i.e. $\binom{M+d-1}{d-1}$ $\quad |{*}{*}{*}||{*}{*}\dots$ $p_1=3$ $p_2=0$ $p_3=2$ ...

This gets large very quickly, so we need a different method.

Recall SVM dual problem: $\sup_\alpha \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \underbrace{\phi(x_j)^T \phi(x_i)}_{\text{kernel } k(x_j, x_i)}$

It is kernelised. $\quad$ s.t. $\sum_{i=1}^n \alpha_i y_i = 0$ & $\alpha_i \in [0, \frac{C}{n}]$ $\forall\, i = 1, \dots, n$

Defn. A method is kernelised if inputs only appear within inner products $\langle \phi(x), \phi(x') \rangle$.

The kernel function $k(x, x') = \langle \phi(x), \phi(x') \rangle$

Turns out we can compute k directly without first computing $\phi(x)$, $\phi(x')$.

e.g. Quadratic feature map. $\phi(x) = (\underbrace{x_1, \dots, x_d, x_1^2, \dots, x_d^2}_{2d}, \underbrace{\sqrt{2} x_1 x_2 \dots \sqrt{2} x_{d-1} x_d}_{\binom{d}{2} = \frac{d(d-1)}{2}})$

Number of features is $O(d^2)$.

But it can be shown that $k(x, x') = \langle x, x' \rangle + \langle x, x' \rangle^2$ which is $O(d)$.

To work with kernels, we usually precompute all of them.

Kernel matrix / Gram matrix $K = (\langle x_i, x_j \rangle)_{i,j} = \begin{pmatrix} \langle x_1, x_1 \rangle & \dots & \langle x_1, x_n \rangle \\ \vdots & & \vdots \\ \langle x_n, x_1 \rangle & \dots & \langle x_n, x_n \rangle \end{pmatrix}$

Note that $K = XX^T = \underset{n \times n}{\begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_n & — \end{pmatrix}} \underset{n \times d}{\begin{pmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{pmatrix}} \underset{d \times n}{}$

So K is dimension $n \times n$, note independent of d!

So we can swap out $k(x_i, x_j)$ for any kernel we like.

Linear kernel. $k(x, x') = x^T x'$

Input space: $\mathcal{X} = \mathbb{R}^d$ Feature map: $\phi(x) = x$

Feature space: $H = \mathbb{R}^d$, standard inner product.

Quadratic kernel. $k(x, x') = \langle x, x' \rangle + \langle x, x' \rangle^2$

Input space: $\mathcal{X} = \mathbb{R}^d$

Feature space: $H = \mathbb{R}^D$ where $D = d + \binom{d}{2} \approx \frac{d^2}{2}$

Feature map: $\phi(x) = (x_1, \dots, x_d, x_1^2, \dots, x_d^2, \sqrt{2}x_1x_2, \dots \sqrt{2}x_ix_j \dots \sqrt{2}x_{d-1}x_d)$

Computation cost without kernel: $O(d^2)$

Computation cost with kernel: $O(d)$.

Note: kernels attach coefficients to features, which could affect regularisation.

Polynomial Kernel $k(x, x') = (1 + \langle x, x' \rangle)^M$

Corresponds to feature map containing all monomials up to degree M.

Radial Basis Function / Gaussian Kernel $k(x, x') = e^{\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)}$

Where σ^2 is the bandwidth parameter. Note that it acts as a similarity score because when $x = x'$, $k = 1$. When x very different from x', $k = 0$.

What is the feature map $\phi(x)$? Turns out it corresponds to an infinite feature map.

So far, dual for SVM happened to be in kernelised form. What about other types of objective function, can we use kernels? Answer is yes, with representer theorem.

But we'll need some basic linear algebra.

An inner product space is a vector space V and an inner product, which is a mapping $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ and has properties:

① $\langle x, y \rangle = \langle y, x \rangle$ $\quad x, y \in V$

② $\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle$ $\quad a, b \in \mathbb{R}$

③ $\langle x, x \rangle \geq 0$ and $\langle x, x \rangle = 0$ iff $x = 0$

The norm of an inner product space is $\|x\| = \sqrt{\langle x, x \rangle}$ ie. inner product with itself.

e.g. $V \in \mathbb{R}^d$, standard euclidean inner product is an inner product space.

$\langle x, y \rangle = x^T y$ then norm $\|x\| = \sqrt{x^T x}$

What norms can be rewritten as an inner product?

Parallelogram Law: A norm $\|v\|$ can be generated by an inner product on V iff:

$$2\|x\|^2 + 2\|y\|^2 = \|x+y\|^2 + \|x-y\|^2$$

and if it can, the inner product is given by the polarization identity:

$$\langle x, y \rangle = (\|x\|^2 + \|y\|^2 - \|x-y\|^2)/2$$

e.g. ℓ_1 norm cannot be written as an inner product.

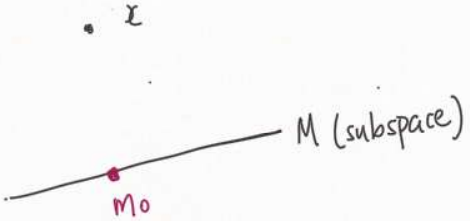
Defn: Two vectors are orthogonal ($x \perp y$) if $\langle x, y \rangle = 0$.

Defn: x is orthogonal to a set S ($x \perp S$), if $x \perp s$ for all $s \in S$.

Pythagorean theorem: If $x \perp y$, then $\|x+y\|^2 = \|x\|^2 + \|y\|^2$

Proof
$$\begin{aligned} \|x+y\|^2 &= \langle x+y, x+y \rangle \\ &= \langle x, x \rangle + \overbrace{\langle x, y \rangle + \langle y, x \rangle}^{0} + \langle y, y \rangle \\ &= \|x\|^2 + \|y\|^2 \end{aligned}$$

Projection $x \in V$ and M is subspace of inner product space of V.



Then m_0 is proj. of x onto M if:

$$\|x - m_0\| \leq \|x - m\| \quad \forall m \in M$$ ie. closest point to x

For projections to exist, we need a criteria called completeness. A Hilbert Space is a complete inner product space

Classical Projection Theorem. Let H be a hilbert space, and M be a closed subspace of H. (ie. a hyperplane through the origin.)
Then for any $x \in H$, there exists a unique $m_0 \in M$ for which

$$\|x - m_0\| \leq \|x - m\| \quad \forall m \in M.$$

Furthermore, m_0 is the projection iff $x - m_0 \perp M$

A simple corollary is the fact that projection reduces norm.
Theorem: $\|m_0\| \leq \|x\|$ with equality only when $m_0 = x$
Proof:

$$\|x\|^2 = \|m_0 + x - m_0\|^2$$
$$= \|m_0\|^2 + \|x - m_0\|^2 \quad \text{by pythagorean theorem.}$$
$$\|m_0\|^2 = \|x\|^2 - \underbrace{\|x - m_0\|^2}_{\geq 0}$$
$$\|m_0\| \leq \|x\| \quad //$$

Representer Theorem. We can rewrite our objective function in the general form:

$$J(w) = \min_{w \in H} R(\|w\|) + L(\langle w, \psi(x_1)\rangle, \ldots \langle w, \psi(x_n)\rangle)$$

- $w, \psi(x_1) \ldots \psi(x_n) \in H$
- $\|w\| = \sqrt{\langle w, w\rangle}$
- R is non-decreasing Regularisation
- L is arbitrary. Loss.

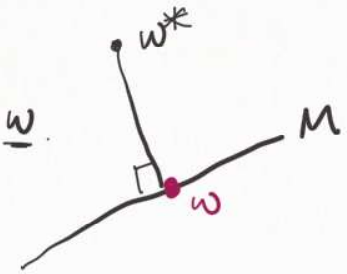
Then: If $J(w)$ has a minimiser, it has a form $w^* = \sum_{i=1}^{n} \alpha_i \psi(x_i)$

Note that lasso cannot be expressed in this form due to ℓ_1 loss, but ridge and SVM can be.

Proof of Representer Theorem:

Let $\mathbf{M}$ be the subspace spanned by $(\psi(x_1) \dots \psi(x_n))$

Suppose we have a w^* not in $\mathbf{M}$, and we find $\text{Proj}_M w^* = \underline{w}$.



Looking at $J(w) = \underbrace{R(\|w\|)}_{①} + \underbrace{L(\langle w, \psi(x_1)\rangle, \dots \langle w, \psi(x_n)\rangle)}_{②}$

If we change from w^* to w:

① $R(\|w\|) \leq R(\|w^*\|)$ because proj. reduces norm.

② Consider $\langle w^*, \psi(x_1)\rangle$

$= \langle w + w^* - w, \psi(x_1)\rangle$ ($\psi(x_1)$ is in $\mathbf{M}$.)

$= \langle w, \psi(x_1)\rangle + \langle w^* - w, \psi(x_1)\rangle$

$= \langle w, \psi(x_1)\rangle$ ($\langle w^* - w, \psi(x_1)\rangle$: orthogonal; $=0$)

So no change to L.

Therefore $J(w)$ must have minimiser of the form $\sum_{i=1}^n \alpha_i \psi(x_i)$. (ie in $\mathbf{M}$).

Significance of representer theorem. Suppose $\psi: X \mapsto \mathbb{R}^d$ where d is 300 million.

Suppose n is 100,000, so $n \ll d$.

We know $w^* = \sum_{i=1}^n \alpha_i \psi(x_i) = \underset{1\times n}{\alpha^T} \underset{n\times d}{\psi(x)}$

Although w lives in 300m space, the soln only lives in 100k subspace spanned by $\psi(x_i)$.

So representer greatly simplifies the problem space.

Now, how do we use the kernelised form to make predictions?

Normally, $f(x) = \langle w^*, \psi(x)\rangle$ but w^* could be large, even infinite dim.

Rewrite: $f(x) = \langle \sum_{i=1}^n \alpha_i \psi(x_i), \psi(x)\rangle$

$= \sum_{i=1}^n \alpha_i \langle \psi(x_i), \psi(x)\rangle$

$= \sum_{i=1}^n \alpha_i k(x_i, x)$.

Note that we rewrote in terms of $k(x_i, x)$, which is much smaller dim than $\psi(x)$.

ie. $f(x)$ is a linear combination of $k(x_i, x)$, and there are n elements. Its even better for SVM where many $\alpha_i = 0$, so less compute.

Kernelised Regularisation.

$$R(\|w\|) = \|w\|^2 \quad \text{e.g. for SVM}$$
$$= \langle w, w \rangle$$
$$= \left\langle \sum_{i=1}^n \alpha_i \psi(x_i), \sum_{j=1}^n \alpha_j \psi(x_j) \right\rangle$$
$$= \sum_{i,j=1}^n \alpha_i \alpha_j \langle \psi(x_i), \psi(x_j) \rangle$$
$$= \sum_{i,j=1}^n \alpha_i \alpha_j k(x_i, x_j)$$
$$= \alpha^T K \alpha$$
$$1 \times n \quad n \times n \quad n \times 1$$

So $R(\|w\|) = \sqrt{\alpha^T K \alpha}$

Define Kernel matrix

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

$K \in \mathbb{R}^{n \times n}$

Kernelised Predictions. Rewrite $f(x)$ in matrix form.

$$\text{Predictions} = \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} = \begin{pmatrix} \alpha_1 k(x_1, x_1) + \cdots + \alpha_n k(x_1, x_n) \\ \vdots \\ \alpha_1 k(x_n, x_1) + \cdots + \alpha_n k(x_n, x_n) \end{pmatrix}$$
$$= K\alpha.$$

So now we can rewrite $J(w)$:

$$J(w) = \min_{\alpha \in \mathbb{R}^n} R\left(\sqrt{\alpha^T K \alpha}\right) + L(K\alpha)$$

We have successfully switched optimisation over w ($d = 300m$) to α ($n = 100k$).

Using representer, sub back soln for $R(\|w\|)$ and $L(f(x))$ to get kernelised forms:

SVM

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|w\|^2 + \frac{C}{n} \sum_{i=1}^n \left(1 - y_i [\langle w, \psi(x_i) \rangle]\right)_+$$

Kernelised SVM

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{2}\alpha^T K \alpha + \frac{C}{n} \sum_{i=1}^n \left(1 - y_i (K\alpha)_i\right)_+$$

Ridge

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \left(w^T x_i - y_i\right)^2 + \lambda \|w\|^2$$

Kernelised Ridge

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \|K\alpha - y\|^2 + \lambda \alpha^T K \alpha$$

Note that kernel forms are quadratic convex forms of α, which can be optimised.

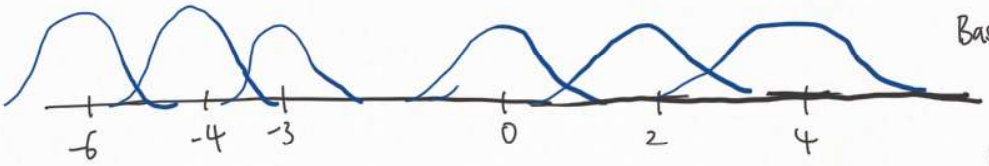
How does SVM prediction for RBF kernel look?

RBF $k(x, x') = e^{-(x'-x)^2}$

Suppose 6 training examples $x_i = -6, -4, -3, 0, 2, 4$.

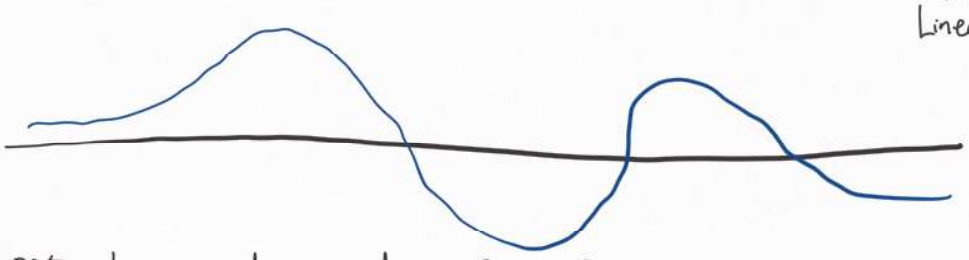
If representer theorem applies, then $f(x) = \sum_{i=1}^{6} \alpha_i k(x_i, x)$

Think of it as a weighted average of the similarity bet. point x and all other training points.



Basis functions

Prediction:
Linear combi of basis functions.



RBF always produces prediction of this form.

14. PERFORMANCE EVALUATION

When is a prediction function good? Need a way to evaluate performance.

First step is to compare with simple baseline models. ("no info" models).

Classification — always predict most freq. class

Regression — mean for sq. loss, median for ℓ_1-loss.

Start with simple model $\longrightarrow$ complex.

Can build oracle model to get upper bound of performance, i.e. use test data in training.

Confusion matrix

		Actual +	Actual −
Predict	+	TP	FP
	−	FN	TN

$$\text{Accuracy} = \frac{TP + TN}{\text{all cases}}$$

But accuracy alone is meaningless without looking at "no-info" rate.

For information retrieval:

$$\text{Precision} = \frac{TP}{TP + FP} \leftarrow \text{all predictions}$$

$$\text{Recall} = \frac{TP}{TP + FN} \leftarrow \text{all true cases}$$

F_1 combines both by taking harmonic mean.

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}$$

If either is 0, then F_1 is 0. In general, it is similar to mean but leans more to lower score.

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{(\beta^2 \text{precision}) + \text{recall}}$$, adjusting β allow us to weigh precision or recall higher.

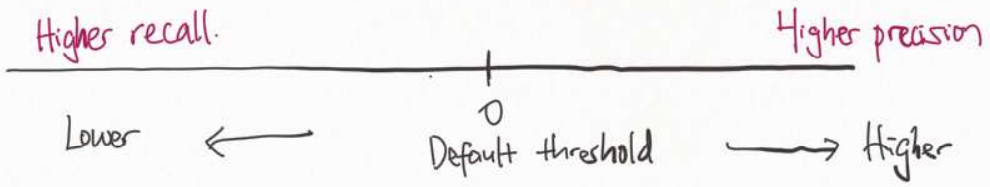
For medical diagnostic.

$$\text{Sensitivity (or TPR)} = \frac{TP}{TP + FN} \leftarrow \text{all true cases}$$

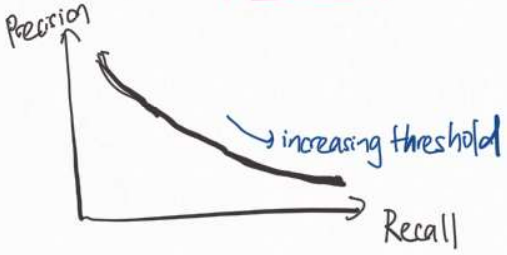
$$\text{Specificity (or TNR)} = \frac{TN}{TN + FP} \leftarrow \text{all neg. cases}$$

High specificity means few false alarms.

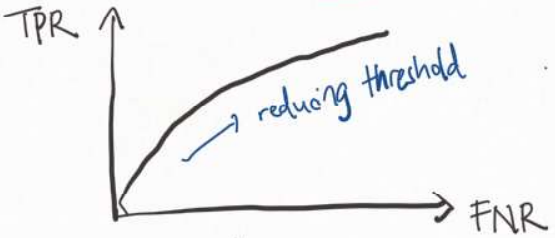
Can adjust performance by adjusting threshold. e.g. adjust score function.



Can plot precision-recall curve



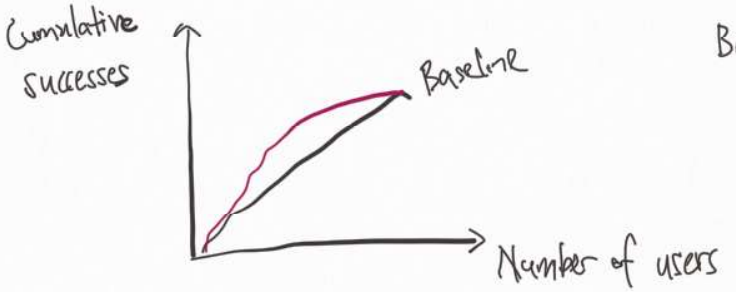
Can also plot ROC curve.



To get a single number from ROC curve, consider area under curve.

Another common measure is the Lift curve

Cell phone churn problem, want to target promo at highest likelihood to churn.



Baseline = select at random.

15. City Sense Case Study

Mobile app using taxi GPS ping to show where people are partying, and which areas have exceptionally high traffic. To do this, need to model "typical behaviour."
Need new type of action space - A = probability distribution on outcome y.
to be able to say probability of an event occurring.

For a given input x, we want to produce $p(y|x)$ ie. a distribution.
Can produce a prediction interval.

For citysense, they divide NYC into 400k grid cells, and consider time at hour level.
Aim is for someone to query grid ^{g} cell and week ^{h} hour, and we return a prob. distribution.

Input space $\mathcal{X} = \{ g, h \mid g \in \{1, \ldots, 400k\}, h \in \{0, \ldots, 167\} \}$

Outcome space y = Number of pickups $\{0, 1, 2, \ldots\}$

Action space A = prob. dist. of pickups.

Since have 27 weeks of data, set first 14 weeks as train and 13 weeks as test.

Trade-off bet. stratification and bucketing.

Build separate model for each g,h — Bucket natural groups together.

Stratification ⟵ (Lower bias) — (Lower variance) ⟶ Bucketing

Low bias, high variance. — Reduce variance (← more data) at expense of bias

Model is highly specific to unique idiosyncracies.

Clever bucketing can reduce variance and not lose much in bias.

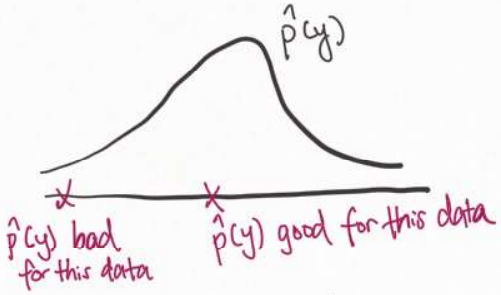
16. Maximum Likelihood Estimation

Setting - we want to estimate $p(y|x)$ where x is input and y is outcome. We want a full prob. distribution, either PMF or PDF.

How do we evaluate our estimated distribution $\hat{p}(y)$?

We want it to be descriptive of future data.

How to quantify?



Likelihood of a Predicted Distribution. Let D be the data which we assume drawn iid from true distribution $p(y)$. $D = (y_1, \ldots, y_n)$ The likelihood of this data is:

$$\hat{p}(D) = \prod_{i=1}^{n} \hat{p}(y_i) \quad \text{since data independent}$$

Maximum likelihood Estimation seeks to maximise $\hat{p}(D)$ by choosing the right probability distribution.

Parametric models - a set of probability distributions which are controlled by a parameter θ.

Poisson

Support $y = \{0, 1, 2 \ldots\}$

Parameter $\lambda > 0$

PMF $p(k, \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$

Where k is observed counts

Beta $y = (0, 1)$

$\alpha, \beta > 0$

$$p(y, \alpha, \beta) = \frac{y^{\alpha-1}(1-y)^{\beta-1}}{B(\alpha, \beta)}$$

Choice of model depends on data support and distribution.

Gamma $y = (0, \infty)$

$k, \theta > 0$

$$p(y, k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-y/\theta}$$

Likelihood for parametric model. Model: $p(y, \theta)$, data sample $D = \{y_1, \dots y_n\}$

The likelihood is then: $p(D, \hat{\theta}) = \prod_{i=1}^{n} p(y_i, \hat{\theta})$

Can take log : $\log p(D, \hat{\theta}) = \sum_{i=1}^{n} \log p(y_i, \hat{\theta})$

Sums are easier to work with than product.

The Maximum Likelihood Estimator is then:

$$\hat{\theta} = \arg\max_{\theta} \log p(D, \theta) = \arg\max_{\theta} \sum_{i=1}^{n} \log p(y_i, \theta)$$

Finding the MLE is an optimisation problem. Either closed form soln or SGD.

Example Observed taxi pickups $D = (k_1, \dots, k_n)$ over n weeks.

How to fit a poisson distribution?

$$\log p(D, \theta) = \sum_{i=1}^{n} \log \frac{\lambda^{k_i} e^{-\lambda}}{k_i!}$$

$$= \sum_{i=1}^{n} k_i \log \lambda - \lambda - \log(k_i!)$$

This is of the form $a \log \lambda - \lambda - b$, which is concave.

$$\frac{\partial}{\partial \lambda} \log p(D, \theta) = \sum_{i=1}^{n} \left[\frac{k_i}{\lambda} - 1\right] = 0$$

$$\sum_{i=1}^{n} \frac{k_i}{\lambda} = n$$

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} (k_i) \; //$$

So the MLE $\hat{\lambda}$ is simply the mean of the counts $k_1, \dots, k_n$.

Another good distribution for modelling counts is negative binomial distribution which is more flexible than poisson.

Note that MLE can also overfit, so need to choose best model on validation set.

Here we did MLE only looking at output y. Next chapter we look at fitting prob. dist. conditional on data x that we observe.

17. Conditional Prob. Models

Focus on linear models for now, generally known as Generalised Linear Model.

Simplest case → y is binary $\{0,1\}$. Given an x, we need to predict a dist. for $y=0$, $y=1$. But since it is binary, knowing $p(y=1)$ will also give us $p(y=0)$. Hence all we need to estimate: $\theta = p(y=1)$ i.e. bernoulli param.

$$x \xrightarrow[\mathbb{R}^d]{} w^T x \xrightarrow[\mathbb{R}]{} f(w^T x) \xrightarrow[[0,1]]{} \theta$$

f is the transfer function, e.g. $\frac{1}{1+e^{-x}}$ logistic $\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ Probit to map a $\mathbb{R}$ to $[0,1]$.

Bernoulli Regression. Let data $D = \{(x_1,y_1),\dots,(x_n,y_n)\}$. We want to estimate θ.

$\log \hat{p}(D) = \log \prod_{i=1}^n \theta_i^{y_i}(1-\theta_i)^{1-y_i}$ when $y_i=1$, we want $\theta_i = p(y_i=1)$

log-likelihood of data D, assuming our choice of prob. dist.

when $y_i=0$, we want $1-\theta_i = p(y_i=0)$

$= \log \prod_{i=1}^n f(w^T x_i)^{y_i}[1-f(w^T x_i)]^{1-y_i}$ sub $\theta_i = w^T x_i$ Key step to make it conditional on x_i.

$= \sum_{i=1}^n y_i \log f(w^T x_i) + (1-y_i)\log[1-f(w^T x_i)]$

Conversely, $J(w) = -\log \hat{p}(D)$ to minimise

Since $J(w)$ is differentiable, we can use SGD. f can be logit or probit.

Poisson Regression Output space is now $y = \{0,1,\dots\}$ Counts.

We want to estimate $\lambda \in (0,\infty)$. How to map $w^T x \to (0,\infty)$?

$$x \xrightarrow[\mathbb{R}^d]{} w^T x \xrightarrow[\mathbb{R}]{} f(w^T x) = e^{w^T x} \xrightarrow[(0,\infty)]{}$$ Standard approach.

Log Likelihood $\log \hat{p}(D) = \sum_{i=1}^n \log \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!}$ allow diff λ_i for each data pt.

$= \sum_{i=1}^n y_i \log \lambda_i - \lambda_i - \log(y_i!)$ key step to make it conditional on x

$= \sum_{i=1}^n y_i w^T x_i - e^{w^T x_i} - \log(y_i!)$ sub $\lambda_i = e^{w^T x_i}$

Maximise this w.r.t. w to get poisson regression fit. Concave function.

Gaussian Regression

We want to produce $N(\mu, \sigma^2)$. Assume σ^2 is known, for simplicity.

So we just need to estimate μ. $\underset{\mathbb{R}^d}{x} \to \underset{\mathbb{R}}{w^T x} \to \underset{\mathbb{R}}{\mu}$. No need transfer function.

$$\log \hat{p}(D) = \sum_{i=1}^n \log\left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{y_i - \mu_i}{\sqrt{2}\sigma}\right)^2}\right] \quad \text{by PDF of normal dist.}$$

$$= \sum_{i=1}^n \log\left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{y_i - w^T x_i}{\sqrt{2}\sigma}\right)^2}\right] \quad \text{sub } \mu_i = w^T x_i$$

$$= \underbrace{\sum_{i=1}^n \log \frac{1}{\sigma\sqrt{2\pi}}}_{\text{indep. of } w} + \underbrace{\sum_{i=1}^n \left[-\left(\frac{y_i - w^T x_i}{\sqrt{2}\sigma}\right)^2\right]}_{\text{we want to maximise this.}}$$

It follows that $w^* = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2$ which is our usual objective fn for least squares.

Multinomial Logistic Regression.

Here $y = \{1, \dots, k\}$. For each x_i, we want to produce k probabilities (1 for each class).

Let $\theta = (\theta_1, \dots, \theta_k)$ represent these probabilities.

$\to \theta_i \geq 0$ and $\sum_{i=1}^k \theta_i = 1$

To estimate θ from x: $\theta = (\langle w_1, x\rangle, \dots, \langle w_k, x\rangle)$

$$= \left(\frac{e^{w_1^T x}}{\sum_{i=1}^k e^{w_i^T x}}, \dots, \frac{e^{w_k^T x}}{\sum_{i=1}^k e^{w_i^T x}}\right)$$

Note: $\theta_i > 0$ and $\sum_{i=1}^k \theta_i = 1$.

So $\hat{p}(y|x) = \frac{e^{w_y^T x}}{\sum_{i=1}^k e^{w_i^T x}}$ where w_y is the parameter corresponding to the observed class y.

Then $\hat{p}(D)$ is $\prod_{i=1}^n \frac{e^{w_{y_i}^T x_i}}{\sum_{j=1}^k e^{w_j^T x_i}}$ where y_i is class of data i, x_i is input for data i and j denotes summation across all classes.

We can maximise $\hat{p}(D)$, which is concave in w.

Maximum Likelihood as ERM

In MLE, we generate a probability distribution (PDF or PMF) for a given input x. We call this P_x. P_x it self is a function – it takes a value and returns the density. For a given data y_i, the density is $P_x(y_i)$. MLE seeks to maximise $\sum_{i=1}^{n} \log P_x(y_i)$ by choosing the right distribution conditional on x.

Question – can we reframe MLE in our ERM framework?

Input Space – $\mathcal{X}$ Outcome space – $\mathcal{Y}$

All pairs x, y independent with distribution P_{xy}.

Action space $\mathcal{A} = p(y)$, ie. prob. distribution.

Hypothesis Space $F : \mathcal{X} \to \mathcal{A}$, maps from x to a prob. distribution, ie $f(x) = P_x$

What is our loss function? Reverse direction of MLE.

$$\ell(P_x, y) = -\log P_x(y) \quad \text{Loss}$$

$$R(f) = -E(\log P_x(y)) \quad \text{Risk}$$

$$\hat{R}(f) = -\frac{1}{n} \sum_{i=1}^{n} \log P_x(y_i) \quad \text{Empirical risk}$$

This is called negative conditional log likehood. By adopting this loss, we can express MLE as empirical risk minimisation.

18. Bayesian Methods

Classical statistics assumes a family of parametric distributions governs the world, ie $p(y|\theta)$ for some θ. If we knew the true θ, then all is solved. But in real world, we have data D instead of θ. So idea is how to estimate θ using D.

One method is point estimation, estimating $\theta = \hat{\theta}(D)$. Good estimators are:

- Consistent – As data $n \to \infty$, $\hat{\theta}_n \to \theta$
- Efficiency – $\hat{\theta}_n$ is as accurate as we can get from sample of size n.

eg MLE is a point estimation technique. First, we have the Likelihood function.

$L_D(\theta) = p(D|\theta) = \prod_{i=1}^{n} p(y_i|\theta)$ Holding D as fixed, how does L vary with θ?

The MLE estimator for θ is then $\hat{\theta} = \arg\max_{\theta} L_D(\theta)$, ie it is the value of θ that maximises the likelihood function.

Applying MLE to a coin flipping example. Suppose we observe data D with n_H heads and n_T tails. We assume that the behaviour of the coin is governed by a bernoulli dist. $p(\text{heads}|\theta) = \theta$, s.t. with prob. θ we get heads and prob. $1-\theta$ we get tails.

Then the likelihood of D we have observed = $L_D(\theta) = P(D|\theta) = \theta^{n_H}(1-\theta)^{n_T}$

We proceed to choose the value of θ that would maximise L_D.

$$\arg\max_{\theta} L_D(\theta) = \arg\max_{\theta} \log L_D(\theta)$$

$$= \arg\max_{\theta} \; n_H \log\theta + n_T \log(1-\theta)$$

$$\frac{\partial}{\partial\theta} = \frac{n_H}{\theta} - \frac{n_T}{1-\theta} = 0$$

$$n_H(1-\theta) = n_T(\theta)$$

$$\hat{\theta} = \frac{n_H}{n_H + n_T} \;//$$ which is the proportion of heads.

In contrast, bayesian approach introduces prior distribution, which is our belief about what θ is likely to be before seeing data.

Previously, we had $L_D(\theta) = p(D \mid \theta)$ and we tried to maximise it to get $\hat{\theta}$, a point estimate for θ. Now we go further and try to recover the full porterior distribution of θ after seeing data.

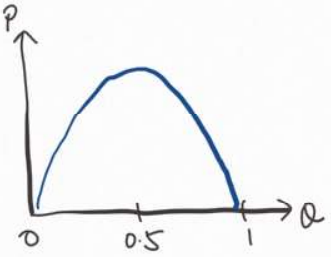
$$\underbrace{p(\theta \mid D)}_{\text{posterior}} = \frac{p(D \mid \theta)\, p(\theta)}{p(D)} \propto \underbrace{p(D \mid \theta)\, p(\theta)}_{\text{likelihood} \times \text{prior}}$$ as $p(D)$ is indep. of θ, bayes rule.

We already have $p(D \mid \theta)$, that is the likelihood function. We have to come up with $p(\theta)$ using our own knowledge. For $p(D \mid \theta)$, remember that we chose a family of dist. to represent the data, e.g. bernoulli for coin flipping.

eg. coin flipping example, we don't know $\theta = P(\text{heads})$ and we want to estimate.

Say we observe n_H heads and n_T tails.

Earlier we had $p(D \mid \theta) = \theta^{n_H}(1-\theta)^{n_T}$. Now we assume the prior follows a beta dist., which is good for modelling probabilities. Suppose $p(\theta) \sim \text{Beta}(2,2)$.



which spreads θ around 0.5.

$$p(\theta \mid D) \propto p(D \mid \theta)\, p(\theta)$$

$$= \underbrace{\theta^{n_H}(1-\theta)^{n_T}}_{\text{likelihood}} \times \underbrace{\theta^{h-1}(1-\theta)^{t-1}}_{\text{pdf of beta}}$$

$$= \theta^{n_H+h-1}(1-\theta)^{n_T+t-1}$$

$$= \theta^{n_H+1}(1-\theta)^{n_T+1} \;//$$

Notice that both prior and posterior are from beta dist. This is because beta family is conjugate to bernoulli family. ie. the likelihood of bernoulli is same form as the pdf of beta.

Conjugates are useful for efficient computation of posterior.

What can we do with a posterior dist.?

- We can calculate mean $E(\theta|D)$, or mode $\arg\max_\theta p(\theta|D)$, and use this θ in a similar way we used θ estimated using MLE.
- We can find the credible set, or "bayesian confidence interval." e.g. we are 95% confident that true θ lies in $[a, b]$.
- Another way to choose θ is to use Bayesian Decision Theory i.e. choose a loss function, then choose a θ that minimises risk.

Suppose we choose $\hat{\theta}$ as our action. Our loss is $\ell(\theta, \hat{\theta})$. (from posterior; chosen action)

Posterior risk $R(\hat{\theta}) = E[\ell(\theta, \hat{\theta}) \mid D]$

$$= \int_{-\infty}^{\infty} \ell(\theta, \hat{\theta})\, p(\theta|D)\, d\theta$$

(loss; posterior)

The bayes action is the choice of $\hat{\theta}$ that minimises the posterior risk.

Example: Square loss $\ell(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$

$$R(\hat{\theta}) = \int (\theta - \hat{\theta})^2 p(\theta|D)\, d\theta \quad \text{from defn}$$

$$\frac{dR}{d\hat{\theta}} = \int -2(\theta - \hat{\theta}) p(\theta|D)\, d\theta$$

$$= -2\int \theta\, p(\theta|D)\, d\theta + 2\hat{\theta}\underbrace{\int p(\theta|D)\, d\theta}_{=1}$$

$$= -2\int \theta\, p(\theta|D)\, d\theta + 2\hat{\theta}$$

$$= 0$$

$$\hat{\theta} = \int \theta\, p(\theta|D)\, d\theta = E(\theta|D) \;//$$

Hence we show that the bayes action for square loss is the mean of posterior distribution.

Example: zero-one loss $\ell(\theta, \hat{\theta}) = I(\theta \neq \hat{\theta})$

$$R(\hat{\theta}) = E[I(\theta \neq \hat{\theta}) \mid D]$$

$$= 1 \times P(\theta \neq \hat{\theta}|D) + \underbrace{0 \times P(\theta = \hat{\theta}|D)}_{=0}$$

$$= 1 - P(\theta = \hat{\theta}|D)$$

$$= 1 - \underbrace{p(\hat{\theta}|D)}_{\text{prob. of whatever } \hat{\theta} \text{ we choose, under } p(\theta).}$$

i.e. $\hat{\theta} = \arg\max_\theta p(\theta|D)$ which is the mode of the posterior. This is called the Maximum a Posteriori estimate.

19. Conditional Bayesian Models

Now we introduce x as input, and want to find $p(y|x,\theta)$. Earlier in MLE setting we choose a single $\hat{\theta}$ and use that to get $p(y|x)$ which we use to make decisions.

Now, instead, we integrate over θ to get weighted average for $p(y|x)$.

Setup

Assume a parametric family $p(y|x,\theta)$ governs our world.

Assume x is fixed and known. For each x_i, we observe a y_i sampled randomly from $p(y|x,\theta)$

Assume that outcomes $y_1, \dots y_n$ are independent (after condition on x).

- Recap: Under MLE, we have $L_D(\theta) = p(D|\theta) = \prod_{i=1}^{n} p(y_i|x_i,\theta)$

The solution is $\hat{\theta} = \arg\max_{\theta} L_D(\theta)$

For bayesian, we again introduce the prior distribution $p(\theta)$.

So Posterior dist. $p(\theta|D) \propto p(D|\theta)p(\theta)$

$= L_D(\theta)\, p(\theta)$

Likelihood x prior

After we find the posterior distribution, we want to get rid of θ to get $p(y|x)$ This is called the Predictive Distribution. We do so by integrating over prior $p(\theta)$ or posterior $p(\theta|D)$.

$p(y|x) = \int p(y|x,\theta)\, p(\theta)\, d\theta$ prior predictive distribution

$p(y|x) = \int p(y|x,\theta)\, p(\theta|D)\, d\theta$ posterior predictive distribution

ie. pdf we chose; posterior dist. of θ,

Example Gaussian Bayesian Regression

Suppose the true model of the world is $y = w_0 + w_1 x + \varepsilon$

where $\varepsilon \sim N(0, \sigma^2)$. So equivalently, $y \sim N(w_0 + w_1 x, \sigma^2)$

Our objective is to estimate w_0 and w_1. Assume prior $w \sim N(0, \Sigma_0) \in \mathbb{R}^2$ (covariance matrix)

The posterior dist. $p(w|D) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$ likelihood

$\times \; |2\pi\Sigma_0|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} w^T \Sigma_0^{-1} w\right)$ prior

(pdf of multivariate gaussian.)

Normal Dist. is conjugate for normal, so the posterior is normal as well.

There is a closed form soln: $w|D \sim N(\mu_p, \Sigma_p)$

$$\mu_p = \left(X^T X + \Sigma_0^{-1}\sigma^2\right)^{-1} X^T y$$

$$\Sigma_p = \left(\sigma^{-2} X^T X + \Sigma_0^{-1}\right)^{-1}$$

Derivation not provided.

Comparison w/ traditional Regression. In traditional regression, we assume data is iid, $y = w^T x + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$. The OLS estimator $\hat{w}$ has a sampling distribution of mean w and covariance $\mathrm{Cov}(\hat{w}) = (\sigma^{-2} X^T X)^{-1}$

Comparing with Σ_p under bayesian, the forms are identical apart from Σ_0^{-1} in bayesian.

If $\Sigma_0^{-1} = 0$, then both are same, because prior variance $\to \infty$.

If $\Sigma_0^{-1} > 0$, then Σ_p is smaller, because our prior contains some info.

Comparison w/ Ridge Regression. If we assume prior variance $\Sigma_0 = \frac{\sigma^2}{\lambda} I$

then $p(w|D) \propto \exp\left(\frac{-\lambda}{2\sigma^2}\|w\|\right) \cdot \prod_{i=1}^{n} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$

(prior) (likelihood)

Recall MAP $\hat{\theta} = \arg\max \hat{p}(\theta|D)$, so MAP for w is:

$$\hat{w}_{MAP} = \arg\min_w -\log p(w|D)$$

$$= \arg\min_w \frac{\lambda}{2\sigma^2}\|w\| + \sum_{i=1}^{n} \frac{(y_i - w^T x_i)^2}{2\sigma^2}$$

$$= \arg\min_w \lambda\|w\| + \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$\mu_p = \left(X^T X + \lambda I\right)^{-1} X^T y$$ which is ridge regression solution.

This is equivalent to the ridge regression objective, so we will get same soln.

Suppose we have found posterior distribution, which in this case is also normal.

How do we find prediction for y_{new}, given new input x_{new}?

Predictive distribution $p(y_{new} \mid x_{new}, D) = \int \underbrace{p(y_{new} \mid x_{new}, w)}_{\text{Likelihood under our chosen pdf. } w \text{ is a variable here ∴ we integrate over } w\text{'s posterior dist.}} \underbrace{p(w \mid D)}_{\text{Posterior dist. of } w} dw$

Turns out for normal dist. and normal prior, the predictive dist has a closed form that is also normal!

$$p(y_{new} \mid x_{new}, D) \sim N(\mu_{new}, \sigma_{new})$$

$$\mu_{new} = \mu_p^T x_{new}$$

$$\sigma_{new} = \underbrace{x_{new}^T \Sigma_p x_{new}}_{\text{From variance in } w} + \underbrace{\sigma^2}_{\text{Inherent variance in } y}$$

It is useful that we can decompose variance in our prediction into ① uncertainty of how well we are estimating w and ② inherent variance in y.

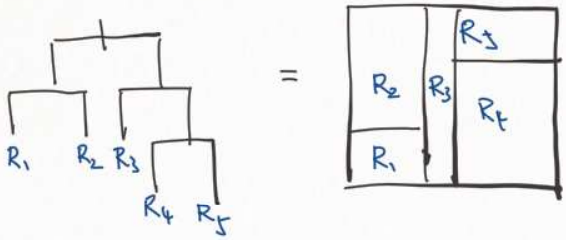
? Don't fully understand this lecture.

20. Classification and Regression Trees

We only consider binary trees here (each node only has 0 or 2 children)

Each split only uses one feature.

Tree splits are equivalent to region partitions.



Regression Trees The decision tree partitions $\mathcal{X}$ into M regions: $\{R_1, \ldots, R_M\}$

Each partition is disjoint, so $\mathcal{X} = R_1 \cup R_2 \ldots \cup R_M$

and $R_i \cap R_j = \phi \quad \forall\, i \neq j$

For a given partition, our final prediction is:

$$f(x) = \sum_{m=1}^{M} c_m \, I(x \in R_m)$$

value indicator

For square loss function, $l(\hat{y}, y) = (\hat{y} - y)^2$, best to predict average:

$$\hat{c_m} = \text{ave}(y_i \mid x_i \in R_m)$$

If we don't control complexity, we will overfit. Complexity measures include # terminal nodes and tree depth.

To find optimal tree is computationally intractable, so we proceed with greedy algorithm.

Let i denote training example $i \in 1 \ldots n$

j denote splitting variable $j \in 1 \ldots d$

s denote split point. $s \in \mathbb{R}$

Split $R_1(j,s) = \{x \mid x_j \leq s\}$ $\quad \hat{c_1}(j,s) = \text{ave}(y_i \mid x_i \in R_1)$ Prediction

$R_2(j,s) = \{x \mid x_j > s\}$ $\quad \hat{c_2}(j,s) = \text{ave}(y_i \mid x_i \in R_2)$

Loss $L(j,s) = \sum_{x_i \in R_1} y_i - \hat{c_1}(j,s)^2 + \sum_{x_i \in R_2} y_i - \hat{c_2}(j,s)^2$

Proceed to find best split point on best feature to minimise loss.

Continue iteratively to build a tree of a given complexity, or do backward pruning.

Classification Trees

Suppose y has K classes. Need a new way to split nodes.

Let node m represent region R_m, with N_m observations.

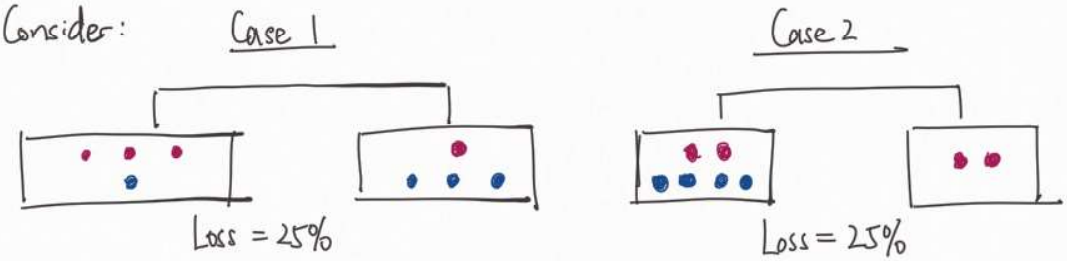
Let $\hat{p}_{mk} = \frac{1}{N_m} \sum I(y_i = k)$ denote % of observations with class k.

Naturally, we should predict for region R_m: $k(m) = \arg\max_k \hat{p}_{mk}$.

Which is the class with the highest proportion in R_m.

What loss function? 0/1 loss is natural, but not the best.

Why? Consider:

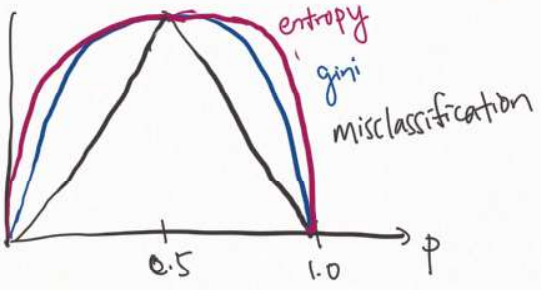


Both cases have 25% misclassification, but we might prefer Case 2 where the split is more pure. Also, left node of Case 2 can be potentially split further. So what we really want is a node impurity measure.

Consider a leaf node m for region R_m. We have 3 possible node impurity measures:

Misclassification error	$1 - \max \hat{p}_{mk}$
Gini	$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
Entropy	$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

For a binary classification problem, this is what they look like:



Can see that entropy really penalises when p is near 0.5, ie. when it is very impure.

Note: maximising info gain is equivalent to minimising entropy.

How then do we split trees?

Let R_L, R_R be potential regions for node split.

Let N_L, N_R be number of points in each region.

Let Q_L, Q_R be node impurity measures.

Then we want split that minimises $N_L Q_L + N_R Q_R$ i.e. weighted impurity

Breiman's trick for categorical features

Suppose we have categorical variable with q categories.

How many possible splits are these into two groups? $2^{q-1} - 1$ how?

This is intractable.

However, for binary classification, we can assign each category a number that is the % of class 0 in that category. Then, split as though it is a numeric feature.

This reduces the complexity to $q-2$.

Breiman proved that this algo is equivalent to searching over all possible splits.

Note on trees

- Trees ignore geometry. The feature scale, distance etc are irrelevant. Trees only care about rank / sort order. So they ignore some amount of information.
- Predictions are not continuous, so might not be as good for regression problems.

21. Basic Stats + Bootstrap

Some setup for basic, standard stats. Assume we have a prob. distribution P.

We want to estimate a parameter of P. (e.g. mean, variance)

We call this parameter $\mu = \mu(P)$. μ is <u>not</u> random.

Suppose we draw data iid from P. $D_n = (x_1, \dots, x_n)$

A statistic is a function of D_n: $s = s(D_n)$

Point estimators are statistics that try to estimate a parameter.

$\hat{\mu} = \hat{\mu}(D_n)$ is a point estimator if $\hat{\mu} \approx \mu$.

Note that s and $\hat{\mu}$ are random because D_n is random.

Since statistics are random, they have probability distributions, or sampling distribution.

The mean and variance of the sampling distribution are of interest, as they give us a confidence interval of $\hat{\mu}$, which tells us how well we are estimating μ.

$E(\hat{\mu})$ $\quad$ $Var(\hat{\mu})$

Bias $\quad$ $\text{Bias}(\hat{\mu}) = E\hat{\mu} - \mu$ $\quad$ ie $\hat{\mu} \pm \sqrt{Var(\hat{\mu})}$

Variance $\quad$ $\text{Var}(\hat{\mu}) = E\hat{\mu}^2 - (E\hat{\mu})^2$

An estimator is unbiased if $E\hat{\mu} - \mu = 0$.

How to calculate $E\hat{\mu}$ and $E\hat{\mu}^2$?

In an ideal world, we would get B independent samples of size n: $D_n^1, \dots, D_n^B$

Then calculate: $E\hat{\mu} = \frac{1}{B}\sum_{i=1}^{B} \hat{\mu}(D_n^i)$

$E\hat{\mu}^2 = \frac{1}{B}\sum_{i=1}^{B} \hat{\mu}(D_n^i)^2$

But in reality, we only have one data sample D_n. So how?

One way is to split data into B groups. Then calculate $\hat{\mu}(D_{n/B})$. But by cutting data up, the estimate for $\hat{\mu}$ is not so good anymore.

Can we get both a good point estimate $\hat{\mu}$ and a variance estimate?

Yes, the answer is the bootstrap.

We draw repeated bootstrap samples by drawing with replacement from D_n, samples of size n. The idea is to simulate having B independent samples from P by drawing B bootstrap samples from sample D_n. We then compute $\hat{\mu}(D_n^1), \ldots, \hat{\mu}(D_n^B)$ as though they are independent samples from P!

The final interval we report is: $\hat{\mu}(D_n) \pm \sqrt{\text{Bootstrap variance}}$

Note that in each bootstrap sample, some data points will not show up while others show up multiple times.

Each X_i has prob. $\left(\frac{n-1}{n}\right)^n$ chance of not being selected.

$$\left(\frac{n-1}{n}\right)^n = \left(1-\frac{1}{n}\right)^n \approx \frac{1}{e} \approx 0.37 \text{ for large } n.$$

So we expect $\approx 63\%$ of data points to show up in each bootstrap sample.

Amazing thing about boostrap is that distribution often comes close to what we'd get drawing independently from P.

22. Bagging and Randomforest

Benefits of averaging Consider some independent samples Z_i drawn iid from Z.

Let $EZ = \mu$ and $Var(Z) = \sigma^2$ (i.e. sampling distribution parameters)

If we use a single Z_i to estimate μ:

$EZ_i = \mu$ i.e. unbiased

$SD(Z_i) = SD(Z) = \sigma$

If we use average of $Z_1, \dots, Z_n$ instead:

$E\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \mu$

$Var\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \frac{\sigma^2}{n}$

So using the average is clearly preferred as it has much lower variance.

Note that this only works if samples are uncorrelated / independent.

If Zs are correlated, e.g. $Corr(Z_i, Z_j) = \rho \quad \forall i \neq j$

Then $Var\left(\frac{1}{n}\sum_{i=1}^{n} Z_i\right) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$

Even when n large, variance is still $\rho\sigma^2$, which limits reduction in variance. So samples must be independent.

Recall that $Var(X_1 + \dots + X_n) = Var(X_1) + \dots + Var(X_n) + 2\sum_{i<j} Cov(X_i, X_j)$

Extend idea of averaging to averaging over prediction functions.

Suppose we have B independent training sets drawn from the same distribution.

We use same algo to get B decision functions: $\hat{f}_1(x), \dots, \hat{f}_B(x)$.

Define $\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x)$

Note that we can consider each $\hat{f}(x)$ as a random variable, since training data is random.

$E\,\hat{f}_{avg}(x) = E\,\hat{f}_b(x)$ same expected value

$Var(\hat{f}_{avg}(x)) = \frac{1}{B} Var(\hat{f}_b(x))$ but smaller variance.

So we clearly prefer $\hat{f}_{avg}(x)$. In practice, we don't have B independent samples, but we can use the bootstrap.

Bagging uses the idea of bootstrap to compute $\hat{f}_{avg}$.

Draw B independent samples from original data D.

Let $\hat{f}_1, \dots, \hat{f}_B$ be prediction functions for each sample.

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x)$$

Empirically, $\hat{f}_{bag}$ performs similar to actually training on B independent samples.

Recall that for each bootstrap sample, ~37% of obs. do not get selected. So it forms a natural cross validation set. Can use this to compute out of bag error, which approximates test error.

When does bagging help? Conventional wisdom is that it helps when base decision functions are unbiased and also have high variance. This leads to idea of random forest.

In bagging, the $\hat{f}_b(x)$s are still somewhat dependent on each other because they are drawn from a limited data sample. Randomforest tries to reduce the dependence further.

Randomforest ① Create B bootstrap samples.

② Choose subset of features to split on. Size m, usually $m \approx \sqrt{p}$ where p is # of features.
To reduce dependence bet. trees.

③ Grow tree very large (to overfit)

④ Compute $\hat{f}_{avg}$ on trees.

23. Gradient Boosting

Gradient boosting is essentially a sequential ensemble.

Adaptive Basis Function Model Up to this point, we have constructed features by ourselves. Call these features $g_1, \dots, g_M$. We then create decision function $f(x) = \sum_{m=1}^{M} v_m g_m(x)$ and we optimise over the coefficients v_m.

The new idea is to allow the features / basis functions to vary as well. Instead of fixed g_m, we have an adaptive h_m.

The ERM objective is: $\hat{f} = \underset{f}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))$

$= \underset{v_m, h_m}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \sum_{m=1}^{M} v_m h_m(x))$

optimise over v_m AND h_m

We know how to optimise over v_m (scalar), but how to optimise over h_m (function)? One way is to consider class of functions that can be governed by a parameter θ_m. Then we can optimise over v_m AND θ_m. For some hypothesis space and loss, this is possible (e.g. neural networks is good e.g.)

But obviously not all functions can be parametrised easily, nor are they necessarily differentiable, e.g. trees. So we resort to solving a simpler problem – Forward Stagewise Additive Modelling (FSAM), which is a greedy search iteratively for the best next piece to add on to the function.

FSAM

① Initialise $f_0(x) = 0$

② For $m = 1$ to M:

a. Compute $(v_m, h_m) = \underset{v,h}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_{m-1}(x_i) + \underbrace{v}_{\text{v is scalar}} \underbrace{h(x_i)}_{\text{h is function}})$ ← new piece

b. Set $f_m = f_{m-1} + v_m h_m$

③ Return f_m.

FSAM can be easily solved for certain loss functions.

eg. L^2 Boosting using square loss

In each step, we want to minimise:

$$J(v,h) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - [f_{m-1}(x_i) + vh(x_i)]\right)^2$$

If H is closed under rescaling, it means that if $h \in H$, then $vh \in H \; \forall h \in \mathbb{R}$.

In this case, we can discard v without any issues.

$$J(v,h) = \frac{1}{n}\sum_{i=1}^{n}\left([\underbrace{y_i - f_{m-1}(x_i)}_{\text{residual from previous step}}] - h(x_i)\right)^2.$$

We can see that it reduces to performing least squares regression on the residual error on each step. Thus, so long as we can do regression on hypothesis space H, we are good to go!

For eg. we can use decision trees with small number of J terminal nodes as the base function.

$J=2$ gives regression stumps, $J=4$ to 8 is generally recommended.

Example Adaboost using exponential loss.

Define the following spaces: $y = \{-1, 1\}$

$A = \mathbb{R}$

Loss function $l = e^{\overbrace{-yf(x)}^{\text{margin}}}$ exponential loss

It turns out that with this setup, FSAM reduces to a form of Adaboost.

Note: exponential loss puts a very large loss on misclassified examples, so Adaboost tends to perform poorly when bayes error rate is high, or lots of mislabeled data.

We can do FSAM for some loss functions (square loss, exponential loss), but not in general.

Enter Gradient Boosting, a way to approximately boost any class of functions.

Recall that FSAM tried to find $v h_m(x)$ at each step that is the globally optimum step. If it is hard to solve this, how about we just try to find the locally best step direction? This is akin to gradient descent, but instead of changing parameters, we add at each step a function that points in the negative gradient direction. This is why it is also called functional gradient descent.

We want to minimise $J(f) = \sum_{i=1}^{n} l(y_i, f(x_i))$

We want to take the "gradient w.r.t f". Note that $J(f)$ only depends on the value of f at the n training points, i.e. $\vec{f} = (f(x_1), \dots, f(x_n))^T$.

Now the objective is : $J(\vec{f}) = \sum_{i=1}^{n} l(y_i, \vec{f_i})$

The negative gradient step at $\vec{f}$ is: $-g = -\nabla_{\vec{f}} J(\vec{f})$

$$= -\left(\partial_{\vec{f_1}} l(y_1, \vec{f_1}), \dots, \partial_{\vec{f_n}} l(y_n, \vec{f_n})\right)$$

This can be easily calculated since we just need to differentiate the loss function.

We want to take a step in $-g$ direction, but this is unconstrained and we might move out of our base hypothesis space. So we need a projection step to find the closest step to $-g$ direction in H.

Finding the closest h : $\arg\min_{h \in H} \sum_{i=1}^{n} (-g_i - h(x_i))^2$

Finally, choose a step size. Either through a line search, i.e. $\arg\min_{v>0} \sum_{i=1}^{n} l(y_i, f_{m-1}(x_i) + v h_m(x_i))$ (choose v)

or just set v between 0 to 1 (0.1 is common). v here is called shrinkage parameter.

Example: Binomial Boost

Using logistic loss for classification problem. $y = \{-1, 1\}$

Recall loss $\ell(y, f(x)) = \log\left(1 + e^{-yf(x)}\right)$

Recall gradient step is to find $-g = -\left(\partial_{\vec{f}_1} \ell(y_1, \vec{f}_1), \ldots, \partial_{\vec{f}_n} \ell(y_n, \vec{f}_n)\right)$

Compute $-\partial_{f_i} \ell(y_i, f_i) = -\partial_{f(x_i)} \log\left(1 + e^{-y_i f(x_i)}\right)$ For ith training example

$$= y_i e^{-y_i f(x_i)} \,/\, \left(1 + e^{-y_i f(x_i)}\right)$$

$$= y_i \,/\, \left(1 + e^{y_i f(x_i)}\right)$$ This is our unconstrained step direction.

After $m-1$ rounds, we have $f_{m-1}(x)$. We first compute the gradient, then project it.

So, step direction for m^{th} step is $h_m = \arg\min_h \sum_{i=1}^n \left(\frac{y_i}{1 + e^{y_i f_{m-1}(x_i)}} - h(x_i)\right)^2$ projection step

Finally, $f_m(x) = f_{m-1}(x) + \underbrace{\nu}_{\text{step size}} h_m(x)$

Example: Gradient Tree Boosting

User base hypothesis space of decision trees, with J as number of terminal nodes (hyperparameter).

Can use square loss (?) for regression task.

Variations on Gradient Boosting. One variation is Stochastic Gradient Boosting, which uses a fraction of the training set for the gradient step. This fraction is called the bag fraction, typically 50%. This method is faster and also prevents overfitting. How do we choose bag fraction? Think of it as choosing sufficient data to estimate the right gradient, given complexity of our hypothesis space.

Another variation is to do column subsampling, which empirically seems to be a better regularisation parameter than row sampling.

Newton step direction. Instead of finding $-g = -\nabla_{\vec{f}} J(\vec{f})$ (first order), another way is to find quadratic approximation to J at $\vec{f}$, and move in direction of minimiser of quadratic.

Recall $J(\vec{f}) = \sum_{i=1}^n \ell(y_i, \vec{f}_i)$.

Write $J(\vec{f} + \underbrace{\vec{r}}_{\text{step}}) \approx \sum_{i=1}^n \Big[\ell(y_i, \vec{f}_i) + \underbrace{\partial_{\vec{f}_i} \ell(y_i, \vec{f}_i)\, \vec{r}_i}_{\text{first term of taylor approx.}} + \underbrace{\tfrac{1}{2} \partial^2_{\vec{f}_i} \ell(y_i, \vec{f}_i)\, \vec{r}_i^{\,2}}_{\text{2nd term.}}\Big]$

We can easily find $\vec{r}$ which minimises this expression.

24. Multiclass Prediction

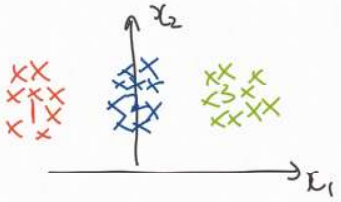
For task of predicting $y = \{1, .., k\}$ classes, so far we used multinomial logit and decision trees. This lecture looks at linear methods for multiclass problems.

One approach is one vs all. For each class i, we train a binary classifier that outputs a score on how likely x belongs to class i. End up with $h_1, \ldots, h_k$ classifiers.

Predict the class with highest score: $h(x) = \arg\max_i h_i(x)$

Problem – if there are too many classes then too many classifiers.

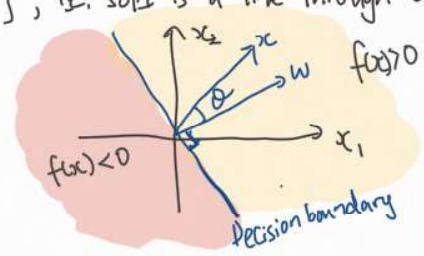
Other problem - one vs all might not learn optimal solution.



Toy problem: Train a linear binary classifier to split class 1, 2, 3:

$H = \{ f(x) = w^T x \}$, i.e. soln is a line through origin.

Recall linear algebra:



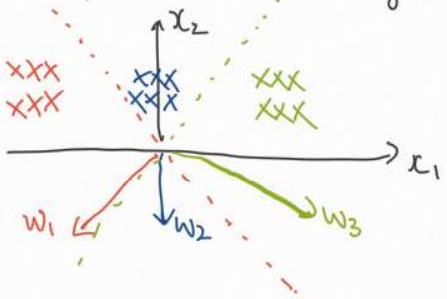
$f(x) = \langle w, x \rangle = \|w\| \|x\| \cos\theta$

$\cos\theta > 0 \iff f(x) > 0 \quad \theta \in (-90°, 90°)$

$\cos\theta < 0 \iff f(x) < 0 \quad \theta \notin [-90°, 90°]$

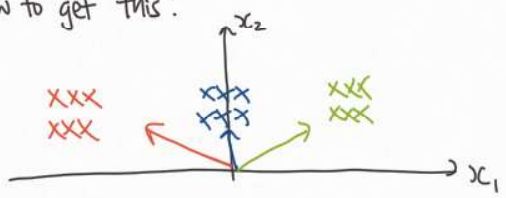
So the decision boundary is perpendicular to $\vec{w}$.

Back to problem: one vs all gives suboptimal fit:



Ok for class 1,3 but fail for 2.

How to get this?



Need to go beyond one vs all.

New approach: Create a compatibility score function that gives a score bet. input x and output class y. ie. $h(x, i)$. Prediction is $f(x) = \arg\max_y h(x, y)$.

Note that one vs all framework is subsumed under this if we set $h(x, i) = h_i(x)$

In words, we want $h(x, y)$ to be large when y is the correct class, and we want it to be small when y is the wrong class.

In math, we want $\underbrace{h(x_i, y_i)}_{\text{input}} \underbrace{y_i}_{\text{true label}} > h(x_i, y) \quad \forall\ y \neq y_i$

Equivalently: $h(x_i, y_i) > \max_{y \neq y_i} h(x_i, y)$

We can define a margin: $m_i = h(x_i, y_i) - \max_{y \neq y_i} h(x_i, y)$

In this setting, our classification is right if $m_i > 0$. We want m_i to be large.

Linear Compatibility Score Function. We can choose $h(x, y) = \langle w, \psi(x, y) \rangle$ where $\psi(x, y)$ extracts features relevant to how compatible input x is with a particular class y.

e.g. if we are predicting whether a word is noun, verb or adjective, a feature could be

$\psi_1(x, y) = I(x \text{ ends in "ly"} \text{ AND } y = \text{noun})$

$\psi_2(x, y) = I(x \text{ ends in "ness"} \text{ AND } y = \text{adjective})$

After estimating w, when we get a new input x (e.g. "apple"), we compute:

$\langle w, \psi(\text{apple}, \text{noun}) \rangle$, $\langle w, \psi(\text{apple}, \text{verb}) \rangle$, $\langle w, \psi(\text{apple}, \text{adjective}) \rangle$ and assign class with the highest score.

Example. Advertising x: Features of a user

y: A large set of banner ads.

We want to predict which ad the user will click on. Some possible features are:

$\psi_1 = I(x \text{ is interested in sports AND } y \text{ relevant to sports})$, $\psi_2(x \text{ in target group of } y)$

Earlier, we learnt SVM for binary classification:

$$\min_w \ \frac{1}{2}\|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \max(0, 1 - y_i w^T x_i)$$

Turns out we can extend this quite easily to multi-class using this framework:

$$\min_w \ \frac{1}{2}\|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \max_{y \neq y_i} [\max(0, 1 - m_{i,y}(w))]$$

Margin $m_{i,y}(w) = \langle w, \psi(x_i, y_i) \rangle - \langle w, \psi(x_i, y) \rangle$

Can optimise and find w.

25. k-Means Clustering

Formal setting. Let $\mathcal{X}$ be a space with distance metric d.

Usually $\mathcal{X} = \mathbb{R}^d$ and $d(x, x') = \|x - x'\|$

Dataset $D = \{x_1, \ldots, x_n\}$

Goal is to partition data into k disjoint sets $C_1, \ldots, C_k$.

We define centroid of C_i as: $\mu_i = \arg\min_{\mu} \sum_{x \in C_i} d(x, \mu)^2$ i.e. minimise distance to all points.

The k means objective: $\min_{\mu_1, \ldots, \mu_k} \underbrace{\sum_{i=1}^{k}}_{\text{clusters}} \underbrace{\sum_{x \in C_i}}_{\substack{\text{pts in} \\ \text{each cluster}}} d(x, \mu_i)^2$

The k-means algorithm:

1. Initialise by randomly choosing initial centroids $\mu_1, \ldots, \mu_k$
2. Repeat until convergence:
   - $\forall i$, let $C_i = \{ i = \arg\min_j d(x, \mu_j) \}$ Assign pts. to nearest cluster
   - $\forall i$, let $\mu_i = \arg\min_{\mu} \sum_{x \in C_i} d(x, \mu)^2$ Move centroid to be near points.
     - ↳ Note that for euclidean distance, $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$ i.e. mean

Note that algo does not guarantee finding global minimum.

26. Gaussian Mixture Models

Consider a generative model for data: ① Choose a random cluster $z \in \{1, \dots, k\}$

② Choose a point x from prob. distribution of z.

ie. $X \mid Z = z \sim N(\mu_z, \Sigma_z)$

The model parameters: Cluster probs $\pi = (\pi_1, \dots, \pi_k)$

Cluster means $\mu = (\mu_1, \dots, \mu_k)$

Cluster covariance $\Sigma = (\Sigma_1, \dots, \Sigma_k)$

The key q is, can we learn the model parameters from data?

① First Step. Suppose we already know π_z, μ_z, Σ_z. Suppose we now observe a data point x. How do we know which cluster z he belongs to?

We can first write out joint density. $p(x,z) = p(z)\,p(x \mid z)$

$= \pi_z \underbrace{N(x \mid \mu_z, \Sigma_z)}_{\text{shorthand for pdf of } p(x \mid z)}$

Then, we can find: $p(z \mid x) = \frac{p(x,z)}{p(x)}$

$= \underbrace{\pi_z N(x \mid \mu_z, \Sigma_z)}_{p(x,z) \text{ for a particular } z} \Big/ \underbrace{\sum_{z=1}^{k} \pi_z N(x \mid \mu_z, \Sigma_z)}_{p(x), \text{ marginal of } x \text{ found by summing over all } z\text{s}.}$

So we see that if we know π, μ and Σ, we can easily find the $p(z \mid x)$ which tells us its probability of belonging in each cluster. Clustering is trivial.

Generally, a mixture model is a model where we have a set of probability distributions, and we generate data by first choosing a distribution randomly, then generate x from the distribution.

Formally, $p(x)$ has a mixture distribution if: $p(x) = \sum_{i=1}^{k} w_i\, p_i(x)$

where $w_i \geq 0$, $\sum_{i=1}^{k} w_i = 1$, and each p_i is a probability density.

In the above model, $p(x) = \sum_{z=1}^{k} \underbrace{\pi_z}_{w_i} \underbrace{N(x \mid \mu_z, \Sigma_z)}_{p_i}$, which does take this form

How to learn parameters of a GMM? First review single gaussian distribution.

Estimating single gaussian using MLE. (recap)

Recall multivariate normal pdf: $p(x \mid \mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$

Apply log: $\log p(x \mid \mu, \Sigma) = -\frac{1}{2}\log|2\pi\Sigma| - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$

Log Joint density of a sample: $J(\mu, \Sigma) = \sum_{i=1}^{n} \log p(x \mid \mu, \Sigma)$

$$= -\frac{n}{2}\log|2\pi\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)$$

We want to maximise $J(\mu, \Sigma)$ by choosing μ and Σ.

So we find $\hat{\mu}, \hat{\Sigma}$ satisfying $\nabla_\mu J(\mu, \Sigma) = 0$ and $\nabla_\Sigma J(\mu, \Sigma) = 0$

After some work, we will get closed form solutions:

$$\hat{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad , \quad \hat{\Sigma}_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu}_{MLE})^T(x_i - \hat{\mu}_{MLE}) \;//$$

Can we apply same approach to GMM? No, we will run into problems.

The log joint density is: $J(\pi, \mu, \Sigma) = \log p(x \mid \pi, \mu, \Sigma)$

Recall marginal of x is: $p(x) = \sum_{z=1}^{k} \pi_z N(x \mid \mu_z, \Sigma_z)$

So $J(\pi, \mu, \Sigma) = \sum_{i=1}^{n} \log \sum_{z=1}^{k} \pi_z N(x \mid \mu_z, \Sigma_z)$

log hits summation, cannot simplify

We run into a dead end trying to solve MLE for GMM. It is possible to do gradient descent, but it is quite tricky.

So how to solve? Use Expectation Maximisation (EM) algorithm.

The algorithm puts together what we have above:

① If we know all π_z, μ_z, Σ_z, then we can easily find $p(z \mid x_i)$

② If we know z_i for every x_i, then the problem reduces to k single gaussian estimations, because we can solve for each cluster at a time. We can easily find π_z, μ_z, Σ_z. We don't have the true z_i, but we have the soft assignment given by $p(z \mid x_i)$ which we can use.

EM iteratively does step 1 and 2 to converge to soln.

EM Algorithm for GMM:

① Initialise parameters μ, Σ, π. There are k of each, if k is our choice of clusters.

Repeat until converge:

② "E" step: Denote soft assignments / responsibilities as γ_i^j, where

$$\gamma_i^j = p(z=j \mid x=x_i) \quad \text{i.e. prob. that } x_i \text{ belongs to cluster } j, \text{ or think of it as "responsibility" cluster } j \text{ takes for } x_i.$$

$$= p(z=j, x=x_i) \,/\, p(x_i)$$

$$= \pi_j \, N(x_i \mid \mu_j, \Sigma_j) \,/\, \sum_{c=1}^{k} \pi_c \, N(x_i \mid \mu_c, \Sigma_c)$$

Calculate γ_i^j using the current estimates of μ, Σ, π, & data i and cluster j.

③ "M" step: Re-estimate the parameters μ, Σ, π using updated responsibilities.

$$\mu_c^{new} = \frac{1}{n_c} \sum_{i=1}^{n} \underbrace{\gamma_i^c}_{\text{weight}} x_i \quad \text{think of it as weighted avg of all } x_i \text{ for cluster } c$$

$$\Sigma_c^{new} = \frac{1}{n_c} \sum_{i=1}^{n} \gamma_i^c \left(x_i - \mu_c^{new}\right)^T \left(x_i - \mu_c^{new}\right)$$

$$\pi_c^{new} = \frac{n_c}{n}$$

where $n_c = \sum_{i=1}^{n} \gamma_i^c$ is the "soft" count of points in cluster c.

Thus far no proof has been given for the EM algo, but intuitively it makes sense. The "E step" assigns clusters while the "M" step does MLE to update parameters.

k-means is a restricted form of GMM. If we fix each Σ_c to $\sigma^2 I$, and we let $\sigma^2 \to 0$, then the algorithm converges to k-means. The soft assignments will turn into hard assignments, because when σ is very small then all the probability for x_i will swing to the cluster it is slightly closer to. The point to note is that GMM is more flexible than k-means.

27. EM Algorithm for Latent Variable models

GMM was a specific application of EM algo. Here we derive the general form for a latent variable model, i.e. z can represent sth else. We use two inequalities:

① Jensen's Inequality If $f: \mathbb{R} \to \mathbb{R}$ is a convex function, and X is an r.v., then:

$$E\, f(x) \geq f(EX)$$

If f is concave then it flips.

Kullback-Leibler Divergence is a measure of how "different" two PMFs are.

Let $p(x)$ and $q(x)$ be two PMFs on $\mathcal{X}$.

Then $KL(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}$

② Gibbs Inequality tells us that:

$$KL(p||q) \geq 0$$

with $KL(p||q) = 0$ iff $p(x) = q(x)\ \forall\, x \in \mathcal{X}$

In the latent variable model setting, we observe $x = (x_1, \dots, x_n)$, but do not observe $z = (z_1, \dots, z_n)$. Hence we call our data the incomplete dataset.

Our objective is to use MLE to find $\theta^* = \arg\max_\theta \log p(x|\theta)$, where θ are parameters that govern distribution of x. But it is difficult to directly optimise this, because $\log p(x|\theta) = \log \sum_z p(x,z|\theta)$. Since the log hits Σ, it is typically intractable.

Hence, we will instead try to maximise the lower bound on $\log p(x|\theta)$.

Introduce $q(z)$ as any PMF on Z, as the support of z.

Then we have:

$$\begin{aligned} \log p(x|\theta) &= \log \textstyle\sum_z p(x,z|\theta) \\ &= \log \underbrace{\textstyle\sum_z q(z) \frac{p(x,z|\theta)}{q(z)}}_{\text{Taking expectation over } z} \qquad \text{log of expectation} \\ &\geq \textstyle\sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)} \qquad \text{expectation of log} \\ &= \mathcal{L}(q,\theta) \end{aligned}$$

by Jensen's (log is concave)

So $\mathcal{L}(q,\theta)$ forms a lower bound for $\log p(x|\theta)$, no matter our choice of q and θ.

We showed that $\log p(x|\theta) \geq \mathcal{L}(q,\theta)$

$$= \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)}$$

$\mathcal{L}(q,\theta)$ is often called the evidence based lower bound (ELBO). So the next step is to choose q and θ to maximise the ELBO.

That is, instead of directly finding: $\hat{\theta}_{MLE} = \arg\max_\theta \log p(x|\theta)$

We instead maximise the ELBO: $\hat{\theta}_{ME} = \arg\max_\theta \left[\max_q \mathcal{L}(q,\theta) \right]$

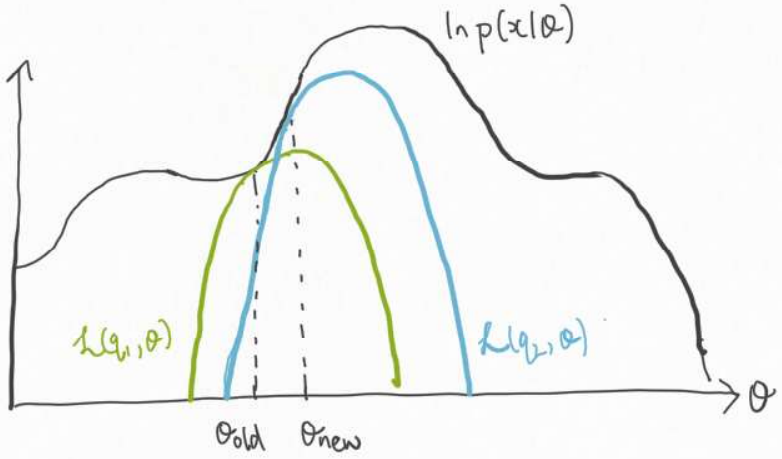
Illustration EM Algo:

① Choose θ_{old}

② Find best q^* (in diagram, $q_1 \to q_2$)

③ Find θ_{new}

④ Repeat



In the algo, we need to compute two things: ① $\arg\max_q \mathcal{L}(q,\theta)$ for a given θ

② $\arg\max_\theta \mathcal{L}(q,\theta)$ for a given q.

① Finding best q for a given θ.

Let's expand the ELBO:

$$\mathcal{L}(q,\theta) = \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)}$$

$$= \sum_z q(z) \log \frac{p(z|x,\theta)\, p(x|\theta)}{q(z)}$$

$$= \sum_z q(z) \log \frac{p(z|x,\theta)}{q(z)} + \sum_z q(z) \log p(x|\theta)$$

$$= -KL(q(z) \,||\, p(z|x,\theta)) + \log p(x|\theta)$$

KL divergence — original objective no q here

We want to find $\arg\max_q \mathcal{L}(q,\theta)$. We know that $KL \geq 0$ and $KL(a,b) = 0$ iff $a = b$.

Thus, we know that: $q^* = p(z|x,\theta)$ — recall this was "soft assignment" in GMM

Subbing back in: $\mathcal{L}(q^*,\theta) = -KL[q^*(z) \,||\, p(z|x,\theta)] + \log p(x|\theta)$

$KL = 0$

Note: amazing that we get back $\log p(x|\theta)$ as an equality. The gap from lower bound is precisely the KL divergence.

$$\mathcal{L}(q^*,\theta) = \log p(x|\theta)$$

This means that when we find best q, our solution should be just touching $\log p(x|\theta)$.

② Finding best θ for a given q.

Now we rewrite $\mathcal{L}(q,\theta)$ differently: $\mathcal{L}(q,\theta) = \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)}$

$= \sum_z q(z) \log p(x,z|\theta) - \sum_z q(z) \log q(z)$

Expected complete data log likelihood — no θ here

So we see that our objective reduces to maximising the expected complete data log likelihood. It really depends on our choice of $q(z)$ whether this would be easy to optimise. In the GMM, it was easy to do so.

Summary of General EM algorithm:

① Choose initial θ^{old}

② Expectation Step:

Set $q^*(z) = p(z|x,\theta^{old})$

Let $J(\theta) = \mathcal{L}(q^*,\theta) = \sum_z q^*(z) \log \frac{p(x,z|\theta)}{q(z)}$

this is an expectation w.r.t $q^*(z)$

③ Maximisation Step:

Find $\theta^{new} = \arg\max_\theta J(\theta)$

④ Go to step 2, until converged.

What guarantees do we have on the EM algo?

① EM gives monotonically increasing likelihood with each step.

For the E step, we chose $q^*(z) = p(z|x,\theta^{old})$. We showed that $\log p(x|\theta^{old}) = \mathcal{L}(q^*,\theta^{old})$

For the M step, we chose $\theta^{new} = \arg\max_\theta J(\theta)$. So $\mathcal{L}(q^*,\theta^{new}) \geq \mathcal{L}(q^*,\theta^{old})$

Combining, we see that: $\log p(x|\theta^{new}) \geq \mathcal{L}(q^*,\theta^{new})$ — $\mathcal{L}$ is lower bound

$\geq \mathcal{L}(q^*,\theta^{old})$

$= \log p(x|\theta^{old})$

Therefore $\log p(x|\theta^{new}) \geq \log p(x|\theta^{old})$

② Global maximum of ELBO is also global max of $\log p(x|\theta)$

Suppose we found global max of $\mathcal{L}(q, \theta)$: $\mathcal{L}(q^*, \theta^*) \geq \mathcal{L}(q, \theta) \quad \forall q, \theta$

where we know $q^*(z) = p(z|x, \theta^*)$

Claim: θ^* is also global maximum of $\log p(x|\theta)$

Proof: Consider any θ' and corresponding $q' = p(z|x, \theta')$

$$\log p(x|\theta') = \mathcal{L}(q', \theta') + \underbrace{KL[q', p(z|x, \theta')]}_{=0}$$

$$= \mathcal{L}(q', \theta')$$

$$\leq \mathcal{L}(q^*, \theta^*)$$

$$= \log p(x|\theta^*)$$

So this gives us good reason to maximise the ELBO.

③ EM algorithm is guaranteed to converge to local maximum.

"Parameter convergence for EM and MM algorithms" - Florin Vaida 2005

Let θ_n be value of θ after n steps.

Define transition function M as $M(\theta_n) = \theta_{n+1}$

Suppose $\ell(\theta) = \log p(x|\theta)$ is differentiable.

Let S be set of stationary points where $\nabla_\theta \ell(\theta) = 0$

Theorem: Under mild regularity conditions, for any starting point θ_0:

① $\lim_{n \to \infty} \theta_n = \theta^* \in S$ We will converge to some stationary point

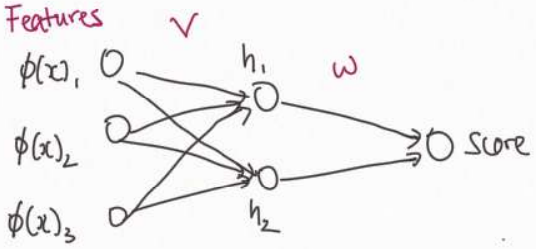
② $M(\theta^*) = \theta^*$ θ^* is a fixed point, EM converges

③ $\ell(\theta_n)$ strictly increases to $\ell(\theta^*)$ as $n \to \infty$, until convergence. EM always increasing likelihood.

Homework: Derive GMM from general EM algo.

28. Neural Networks

Think of neural networks as a new hypothesis space.



It is a linear prediction function with hidden layers.

$$h_i = \sigma(v_i^T \phi(x))$$

$$\text{score} = w_1 h_1 + w_2 h_2$$

$$= w_1 \sigma(v_1^T \phi(x)) + w_2 \sigma(v_2^T \phi(x)).$$

Activation functions. $\sigma(x) = \tanh(x)$ tanh

$\sigma(x) = \max(0, x)$ ReLU

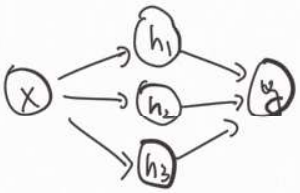
Example. Neural network regression

Input space $\mathbb{R}$ Output/Action space $\mathbb{R}$ 3 hidden layers

$$f(x) = w_0 + w_1 h_1(x) + w_2 h_2(x) + w_3 h_3(x)$$

where $h_i(x) = \sigma(v_i x + b_i)$ for $i = 1, 2, 3$

We need to fit 10 parameters: $b_1, b_2, b_3, v_1, v_2, v_3, w_0, w_1, w_2, w_3$.



We can think of it as learning the features h_1, h_2 and h_3 to make a good prediction, so it is similar to gradient boosting methods in that way.

Apply ERM to find $\theta = (b_1, b_2 \dots, w_3)$ parameters.

Find $\hat{\theta} = \arg\min_\theta \frac{1}{n} \sum_{i=1}^n (f_\theta(x_i) - y_i)^2$

Neural networks are not convex, so we are not guaranteed to find global min. But local min appears good enough.

Theorem from Leshno and Schocken (1991). A neural network with a sufficiently large hidden layer can universally approximate any continuous function on a compact set, iff the activation function is not a polynomial.

Example NN for Multinomial Logistic Regression

$x = \mathbb{R}^d$, $y = \{1, \dots, k\}$ Classification problem.

Compute linear score function $x \mapsto (\langle w_1, x\rangle, \dots, \langle w_k, x\rangle) = (s_1, \dots, s_k)$

Compute softmax $s \mapsto \left(\frac{e^{s_1}}{\sum_{i=1}^k e^{s_i}}, \dots, \frac{e^{s_k}}{\sum_{i=1}^k e^{s_i}} \right)$

How do we put neural networks in here?

Replace linear score function $x \mapsto (\langle w_1, x\rangle, \dots, \langle w_k, x\rangle)$

with non-linear score functions $x \mapsto (f_1(x), \dots, f_k(x))$

Then for learning, we maximise the log-likelihood:

$$\underset{f_1, \dots, f_k}{\arg\max} \sum_{i=1}^n \log \left[\text{Softmax}(f_1(x_i), \dots, f_k(x_i))_{y_i} \right]$$

subscript means we only pick the prediction corresponding to the correct class.

Going into more detail, taking all hidden layers to have m units:

$$\underset{m\times 1}{o^{(1)}} = \sigma\left(\underset{m\times d}{W^{(1)}} \underset{d\times 1}{x} + \underset{m\times 1}{b^{(1)}} \right)$$ takes input of $\mathbb{R}^d$ and convert to $\mathbb{R}^m$.

$$\underset{m\times 1}{o^{(2)}} = \sigma\left(\underset{m\times m}{W^{(2)}} \underset{m\times 1}{o^{(1)}} + \underset{m\times 1}{b^{(2)}} \right)$$

$$\underset{k\times 1}{a(o^{(2)})} = \underset{k\times m}{W^{(3)}} \underset{m\times 1}{o^{(2)}} + \underset{k\times 1}{b^{(3)}}$$

Full neural network is: $f(x) = (a \circ h^{(2)} \circ h^{(1)})(x)$ which gives k score functions.

We then maximise $J(\theta) = \frac{1}{n} \sum_{i=1}^n \log \left[\text{Softmax}(f(x_i))_{y_i} \right]$

where $\theta = (W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$, which we learn with gradient methods.

Regularisation

Neural networks are very expressive and can fit arbitrary functions. So we need to regularise.

We can easily do tikhonov regularisation (or called weight decay):

l_2: $J(w,v) = \sum_{i=1}^{n} \left(y_i - f_{w,v}(x_i) \right)^2 + \lambda_1 \|w\|_2^2 + \lambda_2 \|v\|_2^2$.

Or use dropout regularisation.

Multi-task learning Easily to add multiple outputs for NN. E.g. one output for cats, one for dogs

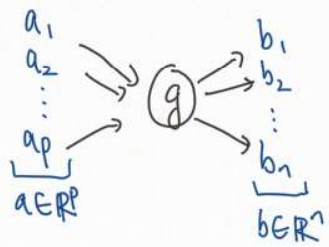
The idea is that basic features for both tasks should be similar. Adding auxiliary tasks should make the network more robust.

Overfeat: Paper showed that hidden layers of CNN are valuable and a linear function on them can give good results for recognition tasks.

29. Backpropagation and the Chain Rule

Backprop is essentially the chain rule, to compute the gradient of a neural network.

Partial Derivatives. Consider a function $g: \mathbb{R}^p \mapsto \mathbb{R}^n$

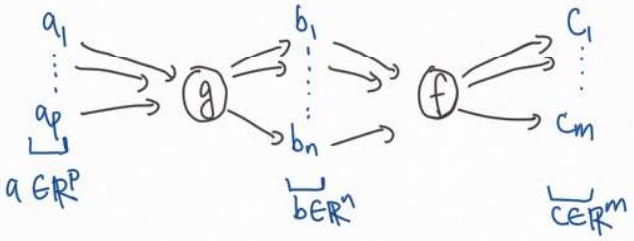


$\frac{\partial b_i}{\partial a_j}$ is the instantaneous rate of change of b_i as we change a_j. If we change a_j by some small δ, then

$$b_i = b_i + \frac{\partial b_i}{\partial a_j}\delta$$

In neural networks, computation graph is more complex. Take two layers for example.

Let $g: \mathbb{R}^p \mapsto \mathbb{R}^n$ and $f: \mathbb{R}^n \mapsto \mathbb{R}^m$



Suppose we want $\frac{\partial c_i}{\partial a_j}$. A change in a_j can affect c_i through each $b_1, \dots, b_n$. So the chain rule tells us that:

$$\frac{\partial c_i}{\partial a_j} = \sum_{k=1}^{n} \frac{\partial c_i}{\partial b_k}\frac{\partial b_k}{\partial a_j}$$

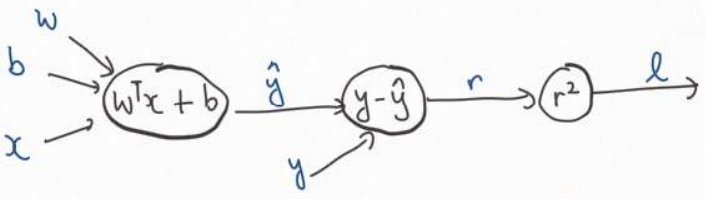
This is the core operation happening in backprop.

Example. Gradient on Least Squares

$\mathcal{H}: \{ f(x) = w^T x + b \}$ $w \in \mathbb{R}^n$, $b \in \mathbb{R}$.

Loss $\ell(w,b) = [(w^T x_i + b) - y_i]^2$ for a given data point.

We can represent the loss function as a computation graph. For a given training pt. (x,y):



$\ell = r^2$

$r = y - \hat{y}$

$\hat{y} = w^T x + b$.

Working our way backwards, we can find all the gradients, and do gradient descent:

$$\frac{\partial \ell}{\partial r} = 2r$$

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial \ell}{\partial r}\frac{\partial r}{\partial \hat{y}} = -2r$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = -2r$$

$$\frac{\partial \ell}{\partial w_j} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = -2r x_j$$

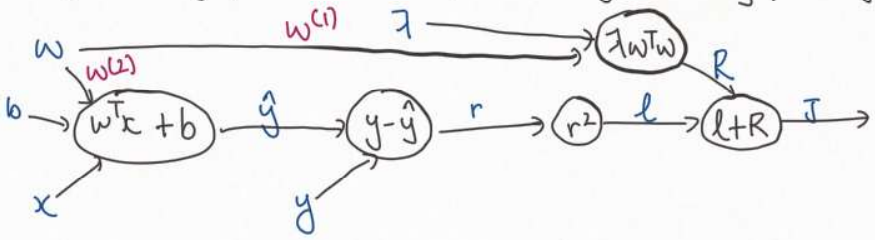
jth entry of w.

Example Gradient on Ridge Regression

$\mathcal{H}: \{ w^T x + b \}$

Loss $J(w, b) = [w^T x + b - y]^2 + \lambda w^T w$

The computation graph is more complex. For a given training pt. (x, y):



As before, we can easily compute:

$$\frac{\partial J}{\partial l} = 1$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{\partial J}{\partial l}\frac{\partial l}{\partial r}\frac{\partial r}{\partial \hat{y}} = -2r$$

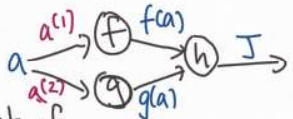
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = -2r$$

But what about $\frac{\partial J}{\partial w_j}$? It appears in two places.

The short digression on the right shows us how to handle this.

Nodes with multiple children.

Consider $J = h(f(a), g(a))$ where a enters computation in two places



It is useful to think of a as two copies, $a^{(1)}$ and $a^{(2)}$.

Then $\frac{\partial J}{\partial a} = \frac{\partial J}{\partial a^{(1)}}\frac{\partial a^{(1)}}{\partial a} + \frac{\partial J}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial a}$ by chain rule

$= \frac{\partial J}{\partial a^{(1)}} + \frac{\partial J}{\partial a^{(2)}}$

So we see it is just the sum of derivatives w.r.t. each copy.

Now, we have:

$$\frac{\partial J}{\partial w_j^{(1)}} = \frac{\partial J}{\partial R}\frac{\partial R}{\partial w_j^{(1)}} = 2\lambda w_j^{(1)}$$

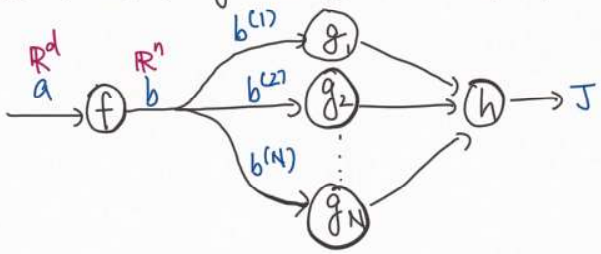
$$\frac{\partial J}{\partial w_j^{(2)}} = \frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j^{(2)}} = -2r x_j$$

$$\therefore \frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j^{(1)}} + \frac{\partial J}{\partial w_j^{(2)}} = 2\lambda w_j - 2r x_j \;//$$

General Backprop

Now we consider how to apply this to neural networks. Backprop works node-by-node backwards from the cost function J. To run the **backward step** on a given node f, we need to have already run backward on all of the children of f.

Consider the following node f, which maps from $\mathbb{R}^d \mapsto \mathbb{R}^n$:



$b^{(1)}, \ldots, b^{(N)}$ are copies of b, representing downstream computations from node f.

Running "backward" on b will give us $\frac{\partial J}{\partial b}$ and $\frac{\partial J}{\partial a}$.

$\frac{\partial J}{\partial b_j} = \sum_{k=1}^{N} \frac{\partial J}{\partial b_j^{(k)}}$ i.e. Sum of downstream impact of each copy of b.

$\frac{\partial J}{\partial a_i} = \sum_{j=1}^{n} \frac{\partial J}{\partial b_j} \frac{\partial b_j}{\partial a_i}$ ∵ each a_i can influence <u>every</u> b_j, so sum up effects.

This is essentially what happens in each node. The key idea is that we must sum up the downstream effects. But otherwise what remains is just to find an efficient implementation of the above.