SMAI

Book: (Pattern Classification
↳ Duda, Hart & Stork)

1) Frequentist Approach → Assume there are a fixed set of parameters
2) Bayesian Approach → Varying parameters

---

* Classification & Clustering

- Preprocessing (Noise removal) → Feature Extraction (Invariance to translation, rotation & scaling) → Classification → Post Processing (Risk + Context)

Note: Training & Test data must be drawn from the same distribution

---

Classification:

- Model, Decision boundary, cost.
  - Decision boundary ↑ Adjust this boundary.
  - cost. ↑ Minimize this (Decision theory) cost.
- Generalization vs Overfitting.
- Post processing after classification can correct errors based on context.

Learning:

1) Supervised → Training data
2) Unsupervised → ex: Clustering
3) Reinforcement → ex: RNNs (Use same data multiple times)

} Also Semi supervised

Probability Density Function:

- $P(a < x < b) = \int_a^b p(x)dx$

  ↑ Continuous probability function

- Different types of functions.

$\Rightarrow p(x) \geq 0$

$\Rightarrow \int_{-\infty}^{\infty} p(x)dx = 1$

- Gaussian,

$* \quad p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

---

* Density Estimation:

- $P = \int_R p(x) \cdot dx$ } Probability of belonging to a region R.

$\Rightarrow$ If R is small, so $p(x)$ is almost constant

$P = \int_R p(x)dx \approx p(x)\int_R dx = p(x) \cdot V \rightarrow$ "Volume" of region R.

- So in 2D $\rightarrow R \Rightarrow (x_1, x_2) \rightarrow$ 2 Dimensional we consider a small square/circle So the area ("volume") $\rightarrow 0$.

$\Rightarrow$ Consider we drew n data samples, $\{x_1, x_2, \ldots, x_n\}$ if there are k points out of n lying in a region R. (centered at x) (volume of $R \rightarrow 0$) ($P = k/n$ in that region)

$* \Rightarrow$ But $P = p(x) \cdot V$

$\therefore \boxed{p(x) = \frac{k/n}{V}}$ } x is the point around which we consider region R.

- Since $V \rightarrow 0$, we would try make $n \rightarrow \infty$, assume we have ∞ samples.

---

* Parzen Window → (Density Estimation)

1). • Consider R to be a hypercube centered at x.

• h = Length of edge

⇒ Volume of hypercube (V) = h^d → Dimension of data.

• Window function,

$$\phi\left(\frac{x_i - x}{h}\right) = \begin{cases} 1, & \frac{|x_{ik} - x_k|}{h} \leq \frac{1}{2}, k = 1,2,\ldots,d \\ 0, & \text{otherwise} \end{cases}$$

⇒ So total number of samples within R is k

$$k = \sum_{i=1}^{n} \phi\left(\frac{x_i - x}{h}\right)$$

∴ Parzen prob density ⇒

$$\boxed{p(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} \phi\left(\frac{x_i - x}{h}\right)}$$

Note:
⇒ We can use other shapes like spheres etc instead of hypercubes.

——— x ———

Parzen Window Dichotomizer:

• Classify into 2 classes → +1 & −1

⇒ Consider parzen window & if majority of neighbours around new point x. are +1, then this point goes to +1 etc.

——— x ———

Convergence of Parzen Window:

• $p_n(x) = \frac{k_n}{h \cdot V_n}$

- $V_n \to 0, k_n = 0, p_n(x) = 0$
- $V_n \to 0, k_n \neq 0, p_n(x) = \infty$
- $n \to \infty, V_n \to 0$ (Desired)
  $n \to \infty, k_n \to \infty$
  $n \to \infty, k_n/n \to 0$

⇒ So we can reduce our window size as n↑.

——— x ———

(k_n = Varying, V = Const)

* Note : Parzen estimation → Reduce volume of window based on n

KNN estimation → Reduce volume of window by fixing number of samples enclosed by it (k_n = Const, V = Varying)

———— x ————

* KNN → (Density Estimation)

2)

- Consider classes → w_i, total = c classes

$$\Rightarrow p(x, w_i) = \frac{(k_i/n)}{V}$$ } k_i = Samples in window belonging to class i.

$$\rightarrow p(w_i|x) = \frac{p(x, w_i)}{\sum_{j=1}^{c} p(x, w_j)} = \frac{k_i}{k}$$

(k = Samples in region with volume V)

⇒ We try fix $k_n = \beta/\sqrt{n}$, For various values of β.

———— x ————

* Nearest Neighbour Classifier (NN)

- Eager Learning → Pre compute based on training data
- Lazy Learning → Given test, calculate then and there

→ NN-Classifier is Lazy Learning

- NN rules partitions space into voronoi cells

———— x ————

* k-NN Classifier

- Consider k nearest neighbours to classify.

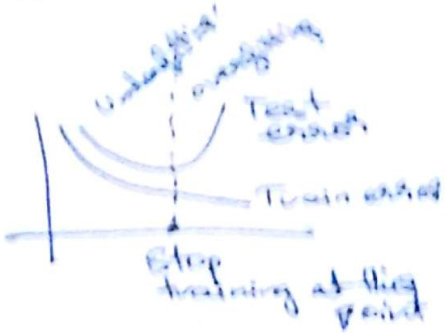
→ k should be odd to avoid ties.

- We could also use distance based weighting etc, so the target function is continuous.
- As the feature dimension increases, accuracy of the classifier drops.

---

* Classifier Evaluation:

- Note:



1) Cross Validation

→ Random Subsampling:
- Consider some data as training & other as test

→ K-Fold Cross Validation:
- Consider windows of k samples as test data, and rest as training data
- ⇒ n/k tests in total.

→ Leave 1 out cross validation:
- Use 1 sample for test (In case our data is very small)

Note: Iris dataset → KNN Classifier, Matlab/C++, K-fold cross validation

---

Tutorials: Wed ⇒ 5:00 to 6:00

3

- Plane → $(x - x_0) \cdot n = 0$
  - x, x_0: 2 points on plane
  - n: Normal

---

* Linear Discriminant Functions:

- We try find parameters of a LDF which minimizes the training error.

$\Rightarrow g(x) = \begin{cases} >0, \text{Class A} \\ <0, \text{Class B} \\ =0, \text{Boundary} \end{cases}$

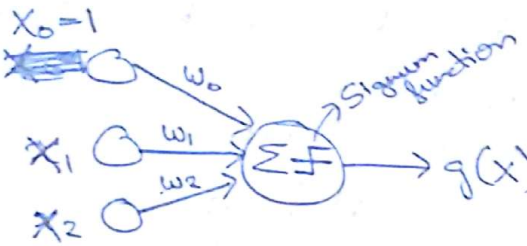
$\Rightarrow g(x) = w^T x + w_0$ → Does this actually represent a hyperplane? Prove below

↳ Hyperplane to seperate the classes

→ In 2D, this becomes a line, $g(x) = mx + c$

---

* Perceptron:



$w = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$

$x = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$

- $g(x) = w_0^T + w_1^T x_1 + w_2^T x_2$ } We will ignore the transpose and assume it exists from now on

This is the bias for the node, So we use $x_0 = 1$

- We use the signum function so that if $g(x) \geq 0$, $f_{g(x)} = 1$, else $f_{g(x)} = 0$. (So we can split it into class A or class B)

Note:

* - $g(x) = w^T x + w_0$, Consider this, w = Normal of hyperplane

Consider an x,

$\Rightarrow x = x_p + \frac{rw}{\|w\|}$, Consider x_p to be projection of x on plane. r = Dist of x from x_p.

(Representation diagram in slides)

- $g(x) = w^T\left(x_p + \frac{rw}{\|w\|}\right) + w_0$

$g(x) = \underbrace{w^T x_p + w_0}_{0 \text{ since } x_p \text{ lies on plane}} + \frac{r(w^T w)}{\|w\|}$

$\therefore g(x) = r\|w\|$

$\Rightarrow r = \frac{g(x)}{\|w\|}$, $r \propto g(x)$, $r_0 = \frac{w_0}{\|w\|}$

- So $g(x)$ can be used to show which side of the plane the point lies on, so which class it belongs to.

* $\therefore$ Its a linear discriminant function & its valid & represents a hyper plane.

$\Rightarrow$ Generalized LDFs

- $g(x) = w^T x + w_0$

$\Rightarrow$ We can use this kind of $g(x)$ as an LDF since $r \propto g(x)$, as we proved above.

$\Rightarrow g(x) = \sum_{i=0}^{d} w_i x_i$, $x_0 = 1$ } For the bias term.

$\Rightarrow \boxed{g(x) = a^T Y}$, $Y = \begin{bmatrix} x_0 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} x_0 \\ x \end{bmatrix}$ → Augmented feature space.

In this space, there is no bias, so this hyperplane goes through origin.

$a = [w_0 : w^T]$

$\Rightarrow Y = \phi(x)$

- ϕ can be any arbitrary mapping which projects original data points to a higher dimension, hoping that we might find a linear hyperplane to divide the data. ex: $\begin{pmatrix} -2 \Rightarrow +1 & 2 \Rightarrow +1 \\ -1 \Rightarrow -1 & \\ 1 \Rightarrow -1 & \end{pmatrix}$
- Take squares & we can divide $\Rightarrow$

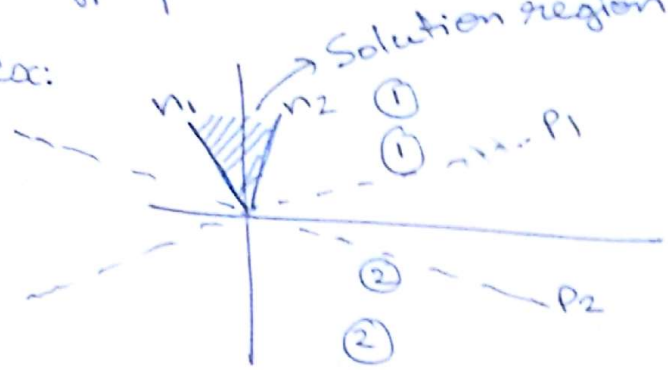
Note: Curse of dimensionality, using higher dimensi spaces requires a lot more time on computa

---

Solution region: → Slides to transform later on

- All possible solution normals for acceptable hyperplanes.

ex:



(Two category linear separabl

- We then take the dual form where a^T form the axes.

4

---

*Gradient Descent: (For linearly seperable data)

- η = Learning rate. } +ve scale factor
- Scalar function $J(a)$ = Classification error described by a.

⇒ Minimize $J(a)$ using gradient descent.

* $$a(k+1) = a(k) - \eta(k) \nabla J(a(k))$$

- Set a $\emptyset$ as a threshold, keep following the algorithm till, $|\eta(k) \nabla J(a(k))| < \emptyset$, then stop. as we could limit num of iterations, till k<

---

* Perceptron Criterion function:

- J should be a continuous function & not discrete.

- $J_p(a) = \sum_{y \in Y} -a^T y$, Y = Set of misclassified samples.
(Piecewise linear)

($a^T y$ will be ≤ 0 for incorrect (Normalized inputs) , $g(x) = a^T y$ $g(x) \leq 0$ for incorrect)

- J_p is our criterion function.

$\Rightarrow * \nabla J_p(a) = \sum_{y \in Y} (-y)$

$\therefore a(k+1) = a(k) + \eta(k) \sum_{y \in Y} y$ ← Set of misclassified points.

* This will only work (converge) if data is linearly seperable.

$\Rightarrow$ This is a Batch Perceptron.

Note: Single Sample Perceptron, iterate over each point doing, $a = a + \eta y^i$, if $a^T y^i < 0$ else we ignore it.
$\Rightarrow$ Similar to batch except that we go one point at a time.

---

*

*Relaxation Procedures:

- Class of criterion functions & minimization meth

1) $\Rightarrow J_q(a) = \sum_{y \in Y} (a^T y)^2$, Convergence to boundary Dominated by longest sample.
(Squared error)

*

2) $J_r(a) = \frac{1}{2} \sum_{y \in Y} \frac{(a^T y - b)^2}{\|y\|^2}$ → To stay a little away from the boundary

Minimum Squared Error: (Mainly for non linearly separable data)

* MSE considers all samples instead of just misclassified ones.

$Y = [y_1 \ldots\ldots y_n]^T$ } Normalized inputs. (Parameters) $a = [a_0 \ldots\ldots a_d]^T$

$y_i = [y_{i0} \ldots\ldots y_{id}]$ $b = [b_1 \ldots\ldots b_n]^T$

We want, $\Rightarrow \boxed{Ya = b}$, $a = Y^{-1}b$ not possible, since Y = rectangular & might be Singular.

($b > 0$) ↳ The distance of points we want from "perfect hyperplane"

$\Rightarrow$ Error $e = Ya - b$ ← Normalized

$\boxed{J(a) = \|e\|^2 = \|Ya - b\|^2}$ } MSE error function.

(Square of norm)

$\nabla J(a) = 2Y^T(Ya - b)$ → Set this to 0 for finding pseudo inverse

Now, $= 0$

$Y^T Ya = Y^T b$ ←

$\therefore \boxed{a = (Y^T Y)^{-1} Y^T b = Y^{\dagger} b}$ } When we achieve 0 error, so $\nabla J(a) = 0$ when we have no error. (Minima)

* * Pseudo inverse of Y. ← ~~Contradicts~~ pseudo inverse

$\Rightarrow Y^T Y$ = Square matrix & often non-singular.

* If $Ya > 0$, ie $\forall i\ (a^T y_i) > 0$, then a hyperplane is always guaranteed. (For normalized y)

$\Rightarrow Ya = b + \varepsilon$, in reality,

so some entries of b can be -ve.

$|b_i| < |\varepsilon|$ (But we try to keep b =

$\therefore$ Even for linearly separable, we might not find a "perfect" hyperplane, but a reasonable

Note: If no info is given, assume $b = [1 \dots 1]^T$

⇒ When we take $ay > 0$ for +ve samples and $ay < 0$ for -ve samples, if we normalize ($y_{-ve} \to -y_{-ve}$) then $ay > 0$ for all inputs.

$\because a y_{-ve} < 0 \Rightarrow a(-y_{-ve}) > 0$ } for -ve inputs.

∴ We can solve this easily if we just assume $ay = b$ ($b > 0$) since this is just an equation & not an inequality.

$\therefore Ya = b$

Normalized inputs.

- Now if we manage to find an a which can do this then we are done
- Using pseudo inverse, we can estimate a value for a. For estimated a^*, If $Ya^* > 0$, then its a seperating plane

⇒ This method is a lot of times simpler, and faster than gradient descent & also works for non linearly seperable case.

Note: Some kinds of variations in b (Non uniform scaling of b) might help at times.
⇒ There are ways in which this is done.

Note: Normalized Inputs:

- If $y \in Y$ is a +ve sample we use y as it is
- If $y \in Y$ is a -ve sample we use $-y$.

$\because a y_{-ve} < 0$

$a(-y_{-ve}) > 0$

∴ To always have $ay > 0$ we consider normalized +ve/-ve inputs

---

* [illegible] $a(k+1) = a(k) - \eta(k) \cdot \nabla J(y_k)$

Note:
Iterative Gradient Descent could be used with the MSE error function.

———x———

* Widrow-Hoff / LMS Procedure: (Stochastic)
  - Using all our datapoints at once may be computationally very expensive.

⇒ So we do,

$$a(k+1) = a(k) - \eta(i)\, y_i (y_i^T a(k) + b$$

for $i = \text{mod}(i+1, n)$

  - Annealing $\eta(k) = \eta(1)/k$ for learning rate

⇒ This is using GD along with MSE error on one sample at a time.

———x———

* Ho-Kashyap Procedure: (Vary b as well instead of fixing)
  - Using modified gradient descent, minimize $J(a,b)$
  - Dont fix $b = [1 \ldots\ldots 1]$, we use the MSE error here as well.
  - $J(a,b) = \|e\|^2 = \|Ya - b\|^2$

⇒ Fix b and minimize $J(a,b)$ wrt a

⇒ Then fix a and minimize $J(a,b)$ wrt b.

* • $b(k+1) = b(k) - \eta(k)\nabla_b J$
  • $a(k+1) = a(k) - \eta(k) Y^T (Ya(k) - b(k+1))$

$\Rightarrow \nabla_b J = 2(Ya - b)(-1) = -2e$

$\therefore e = \frac{-1}{2}\nabla_b J$

- $\nabla_b J$ should always be -ve, so that we always keep adding to b, ie $b(k+1) \geq b(k)$

$\Rightarrow$ In this way we can ensure b is never going to be -ve.

$\therefore$ To do this we cant ensure $\nabla_b J < 0$, so we do $\frac{1}{2}(\nabla_b J - |\nabla_b J|)$ instead of $\nabla_b J$

$$\therefore b(k+1) = b(k) - \eta(k) \cdot \frac{1}{2}(\nabla_b J - |\nabla_b J|)$$

$$= b(k) + \eta(k)\left(\frac{-1}{2}\nabla_b J + \frac{1}{2}|\nabla_b J|\right)$$

$$b(k+1) = b(k) + \eta(k)(e + |e|)$$

- So b keeps increasing or remains equal.

$\Rightarrow$ Pseudocode in slides.

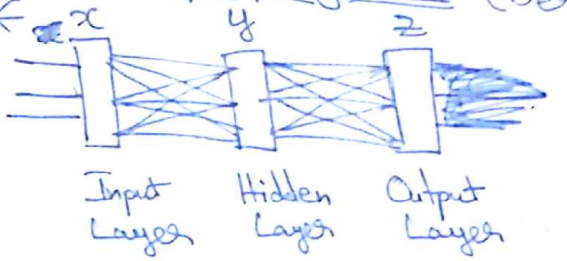
- Instead of updating $a(k+1)$ iteratively we could also just do $a(k+1) = Y^+ b(k+1)$ (Using pseudo inverse)

6

Neural Nets:

- Simple perceptrons can only handle linearly seperable cases well. So extending to neural nets which can identify complex boundaries was done.
- Example of how non-linearity is modelled in slides.

*Backpropogation: (Derivation)



- $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0}$

$\Rightarrow w_{ab}$ = weight lying between node a to node b

$net_j = \sum_{i=0}^{d} x_i w_{ji} = w_j^T x$

- $y_j = f(net_j) = sgn(net_j)$
- $z_k = f(net_k) = sgn(net_k)$

, $net_k = \sum_{j=1}^{nH} y_j w_{kj} + w_{k0}$

$net_k = \sum_{j=0}^{nH} y_j w_{kj} = w_k^T y$

- $g_k(x) = z_k = \sum_{j=1}^{nH} w_{kj} f\left(\sum_{i=1}^{d} x_i w_{ji} + w_{j0}\right) + w_{k0}$

$\Rightarrow J(w) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|t - z\|^2$

$\Delta w = -\eta \frac{\partial J}{\partial w}$, $\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}}$, $w(m+1) = w(m) + \Delta w$

Now,

$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -(\delta_k) \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k y_j$

Now,

$-\delta_k = +\frac{\partial J}{\partial net_k} = +\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k}$

$= +1 \cdot \frac{1}{2}(t_k - z_k)(-1) \cdot f'(net_k)$

$-\delta_k = -(t_k - z_k) \cdot f'(net_k)$

- $\frac{\partial net_k}{\partial w_{kj}} = y_j$

$$\Rightarrow \frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot x_i$$

$$= \frac{\partial J}{\partial y_j} \cdot f'(net_j) \cdot x_i \longrightarrow (1)$$

Now,

$$\frac{\partial J}{\partial y_j} = \sum_{k=1}^{c} \frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j}$$

$$= \sum_{k=1}^{c} -(t_k - z_k) \cdot f'(net_k) \cdot w_{kj}$$

$$= -\sum_{k=1}^{c} \delta_k \cdot w_{kj} \quad , \text{ Put this in } (1)$$

$$\longrightarrow \frac{\partial J}{\partial w_{ji}} = -\left(\sum_{k=1}^{c} \delta_k \cdot w_{kj}\right) \cdot f'(net_j) \cdot x_i$$

$$= -\delta_j x_i$$

- $\Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} = \eta \delta_j x_i$

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial w_{kj}} = \eta (t_k - z_k) \cdot f'(net_k) \cdot y_j$$

*Note: δ_k & δ_j = Sensitivity of backpropogation.

———x———

Stochastic BackProp:

- Initialize weights randomly
- Select samples randomly and modify weights.

⇒ At a certain point, test/validation error rises even though training error keeps decreasing, this is when we stop.

———x———

*Activation Function:

- Should be non linear (Can be linear for small values of
- Should saturate at some point
- Should be continuous and smooth.
- Should be defined
- (Can be monotonically increasing)

a) Linear:

$$y = x$$

b) Sigmoid (Logistic) ⇒ 0 to 1 range

$$y = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}, \quad \frac{dy}{dx} = \frac{-(-e^{-x})}{(1+e^{-x})^2} = y-(1-y)$$

c) Hyperbolic Tangent ⇒ −1 to 1 range

$$y = a\left[\frac{e^{bx}-e^{-bx}}{e^{bx}+e^{-bx}}\right] \quad \left(a=1.716,\ b=2/3 \text{ Generally}\right)$$

———x———

*Practical Aspects:

$$X = [x_1, \ldots, x_m]^T_{n \times d}, \quad \tilde{X} = X - \text{mean}(X)$$

* • $X_{norm} = \tilde{X}/\sigma$ → Standard deviation of each dimension.
  • Use this normalized data only!

- For targets prefer values in range $[-1, 1]$

$\Rightarrow$ Add some random noise to inputs (Training with noise) (To increase training data size)

- To generate more training data, add translation, rotation etc to training data.

* - In general, prefer to have total num of weights to be around $n/10$ (n = Training samples)

$\Rightarrow$ ~~Initialize~~ Initialize weights to random -ve & +ve values, not zeros.

——— x ———

8

- Backprop as feature mapping: Intermediate layer outputs can be used as features.

——— x ———

1) Learning rates:

- For quadratic ~~error~~ criterion J.

$$\eta_{opt} = \left(\frac{\partial^2 J}{\partial w^2}\right)^{-1}$$ → Newton's Methods can be used as well.

* Momentum:

2) - $w(m+1) = w(m) + (1-\alpha)\Delta w(m) + \alpha \Delta w(m-1)$

- Use inertia during weight update.

* Weight Decay: (Normalization)

3) - $w_{new} = w_{old}(1-\varepsilon)$

$\Rightarrow$ Weights which are not updated frequently will go to 0.

* Hints: (Additional nodes in output layer)

4) - While training, have extra output nodes which help train the network better in backprop.

- These will be removed during testing since we don't require them.

5) Online, Batch or Stochastic training

- Stochastic is preferred over batch
- Online when storing samples is not possible while training.

6) Criterion Functions: (Note: Softmax layer & Logsoftmax loss function at back of book) (Tutorial)

- Minkowski error → generalization of Squared Sum error
- A few entropy based criterions.

———— x ————

* Convolutional Neural Networks:

- Use multiple filters convolve over entire image.
- Learn filter weights.

(Translational equivariance)

⇒ Spatial Pooling:

- Reduce image size as we move through the network.

———— x ————

* Recurrent Neural Networks: (& LSTM

- Outputs are fed back into the network as inputs.
- Good for linguistics etc.

———— x ————

Probability & Bayes

- $P(A|B) = P(A,B)/P(B)$

Prior knowledge based classification:

↳ Consider 2 classes w_1 & w_2

- State of nature = Variable 'w', it can either be w_1 or w_2.

↳ $P(w_1) + P(w_2) = 1$ } Based on prior knowledge ($P(w_1) = x_1$, $P(w_2) = x_2$)

↳ If $P(w_1) > P(w_2)$, then $w \to w_1$ else $w \to w_2$.

$\Rightarrow P(w_j, x) = P(w_j|x) \cdot P(x) = P(x|w_j) \cdot P(w_j)$

(class, feature)

$\Rightarrow P(w_j|x) = \frac{P(x|w_j) \cdot P(w_j)}{P(x)}$

(posterior) (likelihood) (prior) (evidence)

$\left(P(x) = \sum_j P(x|w_j) \cdot P(w_j)$, Just a normalizing term$\right)$

$\Rightarrow P(w_j|x) \propto P(x|w_j) \cdot P(w_j)$

↳ Now instead of just $P(w_1)$, $P(w_2)$ we use $P(w_1|x)$ & $P(w_2|x)$ (where our sample has x as the feature)

↑ Bayes decision rule

Sensitivity, Recall, Hitrate $= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$

- We want sensitivity to be as high as possible.
- ex: Cancer test, we want to reduce false negatives

- Precision $= \frac{TP}{TP+FP}$
- Fallout $= \frac{FP}{FP+TN}$
- Specificity $= \frac{TN}{TN+FP}$
- False Discovery rate. $= \frac{FP}{FP+TP}$

- Loss $\lambda_{ij} = \lambda(\alpha_i | w_j)$

  $\Rightarrow \alpha_i$ = Action taken (Assigned Label)

  w_j = State of nature (Actual Label)

- Bayes risk $= R(\alpha_i | x) = \sum_{j=1}^{c} \lambda(\alpha_i | w_j) P(w_j)$

  - We choose α_i if $R(\alpha_i | x) < R(\alpha_j, x)$ $\forall j \neq i$

$\Rightarrow$ We often use $\frac{p(x|w_1)}{p(x|w_2)}$ as a decision boundary to select a class

(Images in slides)

We can get these from R, using comparisons & bayes rule.

— END of Mid —

10

Expectation:

- $\mathcal{E}[f(x)] = \sum_{x \in D} f(x) \cdot P(x)$

$$\mathcal{E}[f(x)] = \int_{-\infty}^{\infty} f(x) \cdot P(x) \cdot dx$$

* Covariance Matrix:

- $cov(X, Y) = \mathcal{E}[(X - \mathcal{E}(X))(Y - \mathcal{E}(Y))]$ (n datapoints, m features)

$= \mathcal{E}[XY] - \mathcal{E}[X]\mathcal{E}[Y]$ (Covariance between features)

$$\Sigma = \begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) & \cdots \\ \vdots & & \\ \cdots & & cov(x_m, x_m) \end{bmatrix}_{m \times m}$$ } Covariance matrix.

$\Rightarrow cov(X, X) = variance(X)$

(Using this we can establish relation b/w different features using multiple samples)

(Note: Eigen values & eigen vectors)

* Entropy:

$$H(P(X)) = -\sum_{k=1}^{K} P(X=k) \log\left(P(X=k)\right)$$

- Entropy is maximized if $P(X=k) = \frac{1}{K}$ $\forall k$
  (Uniform distribution)
- Entropy is minimized for direc-delta function
  ↳ $P(X=k) = 1$, for one k & 0 elsewhere

* KL Divergence:

- Note: If $y = Ax+b$, [redacted] = variance(y) = Var$(Ax+b)$
  (Independent of b)

---

* Normal Densities:

1) Univariate Normal Density:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)} = N(\mu, \sigma)$$

(Gaussian)

- 95% of area is under $\mu-2\sigma$ to $\mu+2\sigma$
- Data that is randomly sampled, we can assume it comes from an underlying gaussian.

2) Multivariate Normal Density:

3) Whitening Transform:

* • Transform arbitrary set of variables X with a covariance matrix to a new set of variables Y with covariance matrix of Y = Identity matrix

⇒ Input vector changes into white noise vectors.

• $X, \Sigma \longrightarrow Y, I$ $\quad \phi^{-1} = \phi$

$\Sigma = \phi \Lambda \phi^{-1} \Rightarrow$ SVD of Σ (→ Single value decomposition theorem), $\phi = [\phi_1, \dots, \phi_d]$

$\Lambda = \text{diag}(\lambda_1, \dots, \lambda_d)$

$A_w = \phi \Lambda^{-1/2}$, $Y = w_0 X$

$A_w^T \Sigma A_w = I$, Now $Y = w_0 X = \phi^T X$

(On solving a few equations we get this)

4) Mahalanobis Distance Metric:

• $r^2 = (x-\mu)^T \Sigma^{-1} (x-\mu)$ } Squared Mahalanobis Distance (Skewed distance)

⇒ $r^2 = (x-y)^T S^{-1} (x-y)$ } Distance b/w 2 random variables (ex: b/w 2 features we have)

• If $S = I$, then this is the euclidean distance

• $\Sigma^{-1} = U \Lambda^{-1} U^T = \sum_{i=1}^{d} \frac{1}{\lambda_i} u_i u_i^T$

$\therefore r^2 = (x-\mu)^T \Sigma^{-1} (x-\mu)$

$= \sum_{i=1}^{d} \frac{1}{\lambda_i} (x-\mu)^T u_i u_i^T (x-\mu)$

$= \sum_{i=1}^{d} \frac{1}{\lambda_i} y_i^2$

Note: Mid I ⇒ Chapters 1, (2.1–2.3), & 4, (5.1–5.7, 5.8.1, 5.9), (6.1–6.5, 6.8, 6.10*)

Multicategory Discriminant Functions:

- Consider $g_i(x)$ for different classes one at a time. Calculate costs and decide on a class for the input.

$(g(x) = w^T x) \rightarrow$ Extend this notion.

$\Rightarrow$ But we can use Bayes,

$$g_i(x) = P(\omega_i|x) = \frac{P(x|\omega_i) \cdot P(\omega_i)}{\sum_j P(x|\omega_j) \cdot P(\omega_j)}$$

or $g_i(x) = P(x|\omega_i) \cdot P(\omega_i)$

$(g_i(x) = -R(\alpha_i|x))$ → we can make this assumption

- Apply a log to $g_i(x)$, and consider this the new $g_i(x)$

$$g(x) = f(g_i(x)) = \ln p(x|\omega_i) + \ln P(\omega_i)$$

$\Rightarrow$ We generally choose class i if $g_i(x) - g_j(x) > 0$ $\forall j$

∴ This means,

$$g(x) = g_i(x) - g_j(x) = \ln \left(\frac{P(x|\omega_i)}{P(x|\omega_j)}\right) + \ln \left(\frac{P(\omega_i)}{P(\omega_j)}\right)$$

(Suppose its 2 class & we choose class i)

$\Rightarrow$ Suppose we assume $P(x|\omega_i)$ comes from a normal multivariate density. → Likelihood density.

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_i| + \ln P(\omega_i)$$

- We dont care about constants or terms

Case 1: (Hyperspherical Clusters)

* $\Sigma_i = \sigma^2 I$ } For all classes.

$|\Sigma_i| = \sigma^{2d}$

$\Sigma_i^{-1} = (1/\sigma^2) I$

- $g_i(x) = -\frac{\|x-\mu_i\|^2}{2\sigma^2} + \ln P(\omega_i)$

(Since we can drop the invariant terms)

(Σ_i = Constant for all i)

$\Rightarrow$ Its similar to distance metric but we also consider prior probabilities.

- $g_i(x) = -\frac{(x-\mu_i)^T (x-\mu_i)}{2\sigma^2} + \ln P(\omega_i)$

$= -\frac{1}{2\sigma^2}(x^T x + \mu_i^T \mu_i - 2\mu_i^T x) + \ln P(\omega_i)$

(Since we ignore terms not dependent on class)

$= \frac{-1}{2\sigma^2}\mu_i^T \mu_i + \frac{\mu_i^T}{\sigma^2} x + \ln P(\omega_i)$

Finally, $g_i(x) \Rightarrow w_i^T x + w_{i0}$ } This is of the form of $g(x)$ we had initially

* *

- $w_i = \frac{1}{\sigma^2}\mu_i$

$w_{i0} = -\frac{1}{2\sigma^2}\mu_i^T \mu_i + \ln P(\omega_i)$

- Now getting back to $g(x) = g_i(x) - g_j(x)$

$$g(x) = \left(\frac{\mu_i - \mu_j}{\sigma^2}\right)^T x - \frac{1}{2\sigma^2}\mu_i^T\mu_i + \frac{1}{2\sigma^2}\mu_j^T\mu_j + \ln(P(\omega_i)) - \ln P(\omega_j)$$

$$= \left(\frac{\mu_i - \mu_j}{\sigma^2}\right)^T x - \frac{1}{2\sigma^2}(\mu_i - \mu_j)^T(\mu_i + \mu_j) - \ln\left(P(\omega_i)/P(\omega_j)\right)$$

$$= (\mu_i - \mu_j)^T\left(x - \left\{\frac{(\mu_i + \mu_j)}{2} - \frac{\ln\left(\frac{P(\omega_i)}{P(\omega_j)}\right)}{(\mu_i - \mu_j)^T}\right\}\right)$$

Constant

$$g(x) = w^T(x - x_0)$$

$g(x) = 0$ (If both classes are equally probable) $\Rightarrow g_i(x) = g_j(x)$

* Case2: (Hyperellipsoidal clusters)

- $\Sigma_i = \Sigma$ } for all i, but its not $\Sigma \neq \sigma^2 I$

Similar to above, we get,

$$\Rightarrow g_i(x) = w_i^T x + w_{io}$$

$$w_i = \Sigma^{-1}\mu_i$$

$$w_{io} = -\frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i + \ln P(\omega_i)$$

* Case3: (Hyperquadrical Clusters)

Σ_i = Arbitrary. (we cant

$$g_i(x) = x^T w_i x + w_i^T x + w_{io}$$

$$w_i = -\frac{1}{2}\Sigma_i^{-1}, \quad w_i = \Sigma_i^{-1}\mu_i$$

$$w_{io} = -\frac{1}{2}\mu_i^T\Sigma_i^{-1}\mu_i - \frac{1}{2}\ln|\Sigma_i| + \ln P(\omega_i)$$

—x—

*Receiver Operating Characteristic Curves (Slides) (ROC Curves)

- Consider tasks like object detection, we draw up curves of (Object detected & Object not detected) - (Sensitivity vs (100-Specificity) plots.

⇒ the curves could be used to detect precision of machine etc.

— x —

12

Parameter Estimation: (Mean & Variance)

- In the DF's we saw earlier, we assumed likelihood $= P(X|\omega_i) = l(\theta)$

⇒ We assumed its a normal distribution, but

* how do we figure out the parameters? (6.4)

$D = [x_1, x_2, \ldots, x_n]$ } Input training data, where all these belong to ω_i

⇒ We need to estimate μ_i & σ_i

$P(x_k|\omega_i) = N(\theta) \Rightarrow l(\theta) = \ln(p(D|\theta))$ } θ = Mean we are trying to find out.

We want to maximize this $l(\theta) = \ln(p(D|\theta)) = \ln\left(\prod_{j=1}^{n} p(x_j|\theta)\right)$

1) Maximum Likelihood Estimation: (MLE)

* a) $\hat{\theta} = \arg\max_{\theta} l(\theta)$

$$p(x_k|\theta) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-(x-\theta)^T \frac{1}{\Sigma}(x-\theta)}$$

$$\ln(p(x_k|\theta)) = \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma| - \frac{1}{2}(x-\theta)^T \Sigma^{-1}(x-\theta) \longrightarrow (1)$$

• Since we want to maximize $l(\theta)$ in maximum likelihood estimation,

$$\Rightarrow \nabla_\theta l(\theta) = \nabla_\theta \sum_{k=1}^{n} \ln p(x_k|\theta) = 0 \quad \} \text{ set it to } 0$$

(Log is a monotonically increasing function, so apply log & maximize it is fine.)

$$\therefore \nabla_\theta \ln p(x_k|\theta) = \Sigma^{-1}(x-\theta) \quad \rightarrow \text{(considering } \Sigma = \text{constant)}$$

$$\nabla_\theta l(\theta) = \sum_{k=1}^{n} \left(\Sigma^{-1}(x_k-\theta)\right) = 0$$

$$* \quad \therefore \boxed{\frac{1}{n}\sum_{k=1}^{n} x_k = \theta} \rightarrow \text{This is our mean (as we might have expected)}$$

——— x ———

b) Unknown μ & $\Sigma = \sigma^2$: (Unlike before where we assumed we knew σ)

$$* \quad * \quad \ln(p(x_k|\theta)) = -\frac{1}{2}\ln 2\pi\theta_2 - \frac{1}{2\theta_2}(x_k-\theta_1)^2$$

$\Rightarrow \theta_1 = \text{Mean}, \theta_2 = \text{Variance}$

$$\nabla_\theta l(\theta) = \begin{bmatrix} \nabla_{\theta_1} l(\theta) \\ \nabla_{\theta_2} l(\theta) \end{bmatrix} \quad \} \text{Both would be 0.}$$

Consider $\nabla\theta_2$,

$$0 = \nabla_{\theta_2} l(\theta) = \nabla_{\theta_2}\left(\sum_{k=1}^{n}\left(-\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln\theta_2 - \frac{1}{2}\frac{(x-\theta_1)^2}{\theta_2}\right)\right)$$

$$\Rightarrow \sum_{k=1}^{n} \frac{-1}{2\theta_2} + \frac{1}{2}\frac{(x_k-\theta_1)^2}{\theta_2^2} = 0$$

$$\Rightarrow -n\theta_2 + \sum_{k=1}^{n}(x_k-\theta_1)^2 = 0$$

$$\boxed{\sigma^2 = \hat{\theta}_2 = \frac{1}{n}\sum(x_k-\hat{a})^2} \quad \boxed{\hat{\theta}_1 = \frac{1}{n}\sum_{k=1}^{n} x_k} \quad \text{As we got above.}$$

2) Bayesian Parameter Estimation: (BPE)

**

$$P(\omega_i|x,D) = \frac{p(x|\omega_i,D_i)P(\omega_i)}{\sum_{j=1}^{c} p(x|\omega_j,D_j)\cdot P(\omega_j)}$$

Class conditional densities, for a certain ω_i.

$$\Rightarrow p(x|D) = \int p(x,\theta|D)\,d\theta = \int p(x|\theta)\,p(\theta|D)\cdot d\theta$$

Now $p(\theta|D) = \frac{p(D|\theta)\cdot p(\theta)}{\int p(D|\theta)\cdot p(\theta)\cdot d\theta}$ } From Bayes

- We are saying that there are 2 gaussians present, one is of the data & the other is of the means itself → Its also a variable curve.

$\Rightarrow$ If $p(\theta|D)$ peaks at some value, $\hat{\theta}$, then

$$p(x|D) = p(x|\hat{\theta})$$

i.e. $p(x|\mu) \Rightarrow N(\mu,\sigma^2)$
$P(\mu) \Rightarrow N(\mu_0,\sigma_0^2)$ } Normal distributions

a) Now, $p(\mu|D) = \frac{p(D|\mu)\cdot p(\mu)}{\int p(D|\mu)\cdot p(\mu)\cdot d\mu}$ } Constant

(Consider univariate gaussians)

$$= \alpha\left(\prod_{k=1}^{n} p(x_k|\mu)\right)\cdot p(\mu)$$

$$= \alpha'\left(\prod_{k=1}^{n} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\right)\cdot e^{-\frac{1}{2}\left(\frac{\mu-\mu_0}{\sigma_0}\right)^2}$$

~~$= \alpha'\prod_{k=1}^{n} e^{-\frac{1}{2}\left(\frac{(x-\mu)^2}{\sigma^2}+\frac{(\mu-\mu_0)^2}{\sigma_0^2}\right)}$~~

$$p(\mu|D) = \alpha' \; e^{-\frac{1}{2}\left(\frac{(\mu-\mu_0)^2}{\sigma_0^2} + \sum_{k=1}^{n}\left(\frac{x_k-\mu}{\sigma}\right)^2\right)}$$

$$= \alpha'' \exp\left[-\frac{1}{2}\left(\left(\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\mu^2 - 2\mu\left(\frac{\mu_0}{\sigma_0^2}+\frac{1}{\sigma^2}\sum x_k\right)\right)\right] \longrightarrow (1)$$

(The other constants not dependent on μ are ignored)

- Now comparing (1) this with the form

(Assume it's also a gaussian)

$$\therefore \; \beta \cdot \exp\left[-\frac{1}{2}\left(\frac{\mu-\mu_n}{\sigma_n}\right)^2\right]$$

$$\Rightarrow \frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}$$

$$= \frac{n\sigma_0^2 + \sigma^2}{\sigma^2\sigma_0^2}$$

$p(\mu|D)$ gaussian's parameters

$$\therefore \; \sigma_n^2 = \frac{\sigma^2\sigma_0^2}{\sigma^2 + n\sigma_0^2}$$

$$\Rightarrow \mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2}\right)\bar{x}_n + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0$$

($\bar{x}_n$ ↑ mean of x_i)

- As $n \to \infty$, $\mu_n = \bar{x}_n$, $\sigma_n \to 0$
  - ↳ Mean converges to sample mean
  - ↳ $\mu_n = \mu_0$ if $\sigma_0 = 0$
  - ↳ If $\sigma_0 \gg \sigma$, then μ_n = sample mean.

$\Rightarrow$ We also can derive that, by putting this in (1) (Slides)

$$p(x|D) \sim N(\mu_n, \sigma^2+\sigma_n^2)$$

($p(x|\mu) \cdot p(\mu|D)$)

↳ If we know this, we know prob of X belonging to class C with samples D.

Each class's distribution.

Note: Recursive Bayes Learning,

- Consider current samples, learn parameters
- Add another sample & update

Repeat these

*MLE vs BPE: (Slides)

- MLE is easier to calculate.

⇒ BPE * → Bayes or Indistinguishability errors
↳ Overlapping class conditional densities (The underlying distrib are of this kind
→ Model Errors
↳ Disparity b/w true / assume distribution
↳ Not Gauss
→ Estimation Errors
↳ Small data size.

?

———— x ————

14

*Principal Component Analysis: (PCA)

- Move from n dimensional data to $k << n$ dimensional data so that the dimensions are as independent as possible.
(Try to have minimal correlation b/w dimensio

⇒ Preserve the most of the variance of the da on dimensionality reduction.

$$X_{data} = [x_1, x_2, \dots, x_n] \quad x_i \in \mathbb{R}^d$$

1) • $J_0(x_0) = \sum_{i=1}^{n} \|x_i - x_0\|^2$, we want to minim this error

Let $\hat{x}_i = x_0$ (0 Dimensional)

$\therefore \nabla J_0(x_0) = 2 \sum_{i=1}^{n} (x_i - x_0) = 0$,

$$\therefore \quad n x_0 = \sum_{i=1}^{n} x_i$$

$$x_0 = \frac{1}{n} \sum_{i=1}^{n} x_i \quad \} \text{This is the mean}$$

* (1 Dimensional variant) → Single eigen vector & value

2) $\hat{x}_i = a_i e + m$ } Projection of data along some axis shifted towards mean.

* (Let $\|e\| = e^T e$) (Unit vector)

$$J_1(a_1, a_2, \ldots, a_n, e) = \sum_{i=1}^{n} \|(a_i e + m) - x_i\|^2$$

$$= \sum_{i=1}^{n} \|a_i e - (x_i - m)\|^2$$

$$= \sum_i a_i^2 e^T e - 2 a_i e^T (x_i - m) + \|x_i - m\|^2$$

$$J_1(a_1, a_2, \ldots, a_n, e) = \sum_i a_i^2 - 2 \sum a_i e^T (x_i - m) + \|x_i - m\|^2 \quad \rightarrow (1)$$

$$\nabla_{a_i} J_1 = 2 a_i - 2 e^T (x_i - m) = 0 \quad \text{(Minimize)}$$

* $a_i = e^T (x_i - m)$ } Substitute in (1),

$$\therefore \quad J_1(e) = \sum_i a_i^2 - 2 \sum a_i^2 + \sum \|x_i - m\|^2$$

$$= - \sum a_i^2 + \sum \|x_i - m\|^2$$

$$\therefore \quad J_1(e) = - \sum_i e^T (x_i - m)(x_i - m)^T e + \text{const}$$

$$J_1(e) = - e^T \underbrace{S_{d \times d}}_{\text{Scatter matrix of data}} e \quad , \text{ Subject to } e^T e = 1$$

$$J_1(e) = -e^T S e + \lambda (e^T e - 1) \quad \} \text{Lagrange optimization}$$

$\nabla_e J_1(e) = -2Se + 2\lambda e = 0$

$\therefore Se = \lambda e$ } Eigenvectors & eigenvalues of S & e are λ & e.

3) (k- dimensional variant)

$$x = m + \sum_{i=1}^{k} a_i e_i$$

Note: We can prove that λ = Variance for data after projection $\rightarrow \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{d} x_{ij} e_j - \mu\right)^2 = \lambda$. (Video proof)

Note: This PCA can be applied to face classification (eigen faces)

Note: Once you have matrix of eigen vectors

$E = [e_1\ e_2\ \cdots\ e_k] = k \times d$ dimensional

$X_i = d \times 1$ dimensional data point

$\hat{x}_i = E \cdot (X_i - m)$ } We get a k dimensional data point

($X_i - m$: Subtract mean)

Note: How to choose k? $\Rightarrow \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} = \rho$

(Sorted orders of eigen values)

Try keep $\rho \in [0.8, 0.95]$

⟹ If we drop more dimensions we lose more, but after a certain point the eigen values become quite small compared to others so we can stop at that point.

———x———

15

* Fisher's Linear Discriminant Analysis: (LDA)

- PCA tries to find a projection of data so as to maximize overall scatter.
- LDA tries to find a projection so as to minimize intra class scatter & maximize inter class scatter.

⟹ $y_i = w^T x_i$ } Projection of data on w.
(Projected data)

$X = (\{x_1, \dots x_{n_1}\}, \{x_{n_1+1}, \dots x_{n_2}\})$ } 2 classes datapoints

$w^T m_1 = \bar{m}_1 = \frac{1}{n_1} \sum_{i=1}^{n_1} y_i$

$w^T m_2 = \bar{m}_2 = \frac{1}{n_2} \sum_{i=1}^{n_2} y_{i+n_1}$ } Means of 2 classes.

$\Rightarrow \bar{s}_i^2 = \frac{1}{n_i} \sum_{y_i \in Y_i} (y_i - \bar{m}_i)^2$

($y_i \in Y_i$: i^{th} class)

$\bar{s}_i^2 = \frac{1}{n_i} \sum_{x \in X_i} (w^T x - w^T m_i)^2$

$\bar{s}_i^2 = \sum_{x \in X_i} w^T (x - m_i)(x - m_i)^T w$

$= w^T S_i w$

(S_i : Scatter matrix for a single class)

⇒ Now our criterion function,

* $J(w) = \frac{|\bar{m}_1 - \bar{m}_2|^2}{\bar{s}_1^2 + \bar{s}_2^2}$ } Maximize inter class scatter
} Minimize intra class scatter

↑ We want to maximize this.

$$= \frac{w^T (m_1 - m_2)(m_1 - m_2)^T w}{w^T S_1 w + w^T S_2 w}$$

$$J(w) = \frac{w^T S_B w}{w^T (S_1 + S_2) w} = \frac{w^T S_B w}{w^T S_w w}$$

($S_w = S_1 + S_2$)

→ Scatter matrix of means

⇒ $\alpha(w, \lambda) = w^T S_B w - \lambda w^T S_w w$

$\nabla_w \alpha(w, \lambda) = 0 = 2 S_B w - 2\lambda S_w w$

↑ Maximize

$\Rightarrow S_B w = \lambda S_w w$

$\Rightarrow (S_w^{-1} S_B) w = \lambda w$

—— x ——

Note: Rank of a matrix = Number of non zero eigen values.

* A matrix is invertible if no eigen value is 0.

—— x ——

- So our w's are basically eigen vectors of $(S_w^{-1} S_B)$.

⇒ Here we considered a single eigen vector so just one w corresponding to the maximum eigen value. (we could use multiple eigen vectors as well)

—— x x ——

* Multiple Discriminant Analysis (MDA)

- Generalize LDA for multiple classes.

- $S_B = \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T$ (N_i = # of samples in class i)

- $S_W = \sum_{i=1}^{c} \sum_{x \in D_i} (x - m_i)(x - m_i)^T$

↓

$S_W = S_1 + S_2 + \dots + S_c$ } Rank is at most $N - c$, so it's always singular.

- We want to maximize $J(W) = \frac{|W^T S_B W|}{|W^T S_W W|}$

$\Rightarrow S_T$ (Total Scatter) $= \sum_{i=1}^{N} (x_i - m)(x_i - m)^T$

$$S_T = \sum_{i=1}^{c} \sum_{x \in D_i} (x - m_i + m_i - m)(x - m_i + m_i - m)^T$$

$$? = \sum_{i=1}^{c} \sum_{x \in D_i} (x - m_i)(x - m_i)^T + \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T$$

$\therefore S_T = S_W + S_B$ //

? • Now the issue is that S_W is not invertible

? ⇒ So we apply fisherface solution using PCA

* to reduce dimensionality.

* $W_{PCA} = \underset{W}{\arg\max} |W^T S_T W|$, $S_T = \sum_x (x - m)(x - m)^T$

⇒ We just replace S_B with $W_{PCA} S_B W_{PCA}^T$ & S_W with $W_{PCA} S_W W_{PCA}^T$ in $J(W)$.

* Fisher Faces: (& Eigen Faces)

- Similar to eigen faces but instead of eigen vectors from PCA, we use LDA analysis.

⟹ Much better than PCA since we optimize on putting ~~extra~~ examples of the same class as close as possible & move different classes as far as possible.

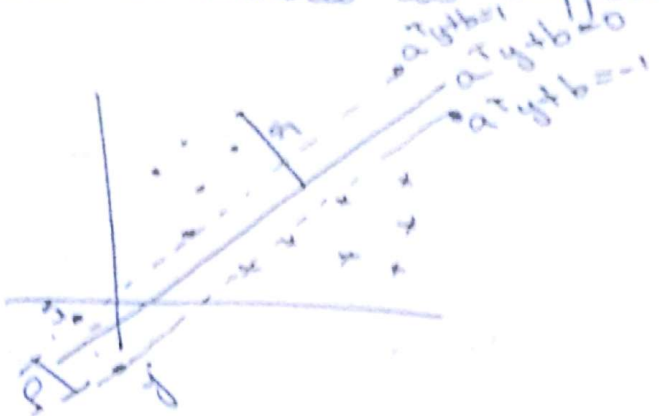
- Once we get the LDA vectors, just project the data on them and then use that data to classify the faces.

Note: Limitations of Fisher's LDA

- LDA gives at most c-1 feature projections.
- LDA assumes unimodal gaussian likelihoods.
- LDA fails if class discrimination is captured in the variance & means are similar. (Since LDA only uses means & doesn't care about the variance)

Support Vector Machine: (SVM)

- Datapoints close to the boundary in all classes are considered as support vectors.



- Now consider a margin around the boundary.

$$\rho = x_i - x_j$$ } Assume 2 hypothetical points on each of the boundary with margin

$$= \frac{a^T y_i + b}{\|a\|} - \frac{a^T y_j + b}{\|a\|}$$

$$\rho = \frac{2}{\|a\|} = \frac{1}{\frac{1}{2} a^T a}$$

↑ We want to maximize this

$$a^* = \arg \min_a \frac{1}{2} a^T a$$ } Primal Formulation

(Constraints, $z_i (a^T y_i + b) \geq 1$ $\forall$ datapoints ~~in normalized form.~~

z_i ↓ Class -1 or $+1$

- This is a quadratic optimization problem where we optimize a quadratic function subject to linear constraints.

* $L(a, \alpha, b) = \frac{1}{2} a^T a - \sum \alpha_i \{(a^T y_i + b) z_i - 1\}$

*

- This is the dual form where we assign a lagrange multiplier α_i with each constraint.

$$\Rightarrow L(a, \alpha, b) = \frac{1}{2} a^T a - \sum \alpha_i a^T y_i z_i - \sum \alpha_i b z_i + \sum \alpha_i \quad \rightarrow (1)$$

$\arg\max_{\alpha_i} \min_{a,b} L(a, \alpha, b)$ such that $\alpha \geq 0$

$$\nabla_a L(a, \alpha, b) = 0 \Rightarrow a = \sum \alpha_i y_i z_i$$
$$\nabla_b L(\alpha, a, b) = 0 \Rightarrow \sum \alpha_i z_i = 0$$

} Substitute these in (1),

*

$$\Rightarrow L(\alpha) = \sum \alpha_i - \frac{1}{2} \sum_{i,k} \alpha_i \alpha_k z_i z_k y_i^T y_k$$

↑ Dual (Maximize) Formulation

$\alpha_i \geq 0$ } Constraint.
$\sum \alpha_i z_i = 0 \quad \forall_i$ ↑

$\frac{\partial L(\alpha_i)}{\partial \alpha_i} = 0$

- This would give us the optimal α_i's

$\Rightarrow a = \sum \alpha_i y_i z_i$. } This is very similar to gradient descent or perceptron

$\Rightarrow b_k = z_k - \sum_{j=1}^{n} \alpha_j z_j y_j^T y_k \quad \} \forall a_k > 0$

- We then take an average of b_k's to get the b (margin).

* Non zero α_k says that y_k is a support vector.

$\Rightarrow f(y) = \underbrace{\sum_{j=1}^{n} \alpha_j z_j y_j^T y + b}_{(a^T y + b)}$ } The final classification function.

Note: This is just like gradient descent but in a statistical way where we get appropriate α_i's for each y_i so we set a correspondingly.

---

* Soft Margin SVM:

- To allow some slack, we try to minimize $\sum \xi_i$ to minimize misclassified

$$z_i(a^T y_i + b) \geq 1 - \xi_i$$ } Set this additionally

* $\Rightarrow$ So we have, our primal formulation

$$a^* = \arg\min_a \frac{1}{2} a^T a + C \sum \xi_i$$

$$\forall_i \xi_i \geq 0, \; z_i(a^T y + b) \geq 1 - \xi_i$$

↳ And the dual formulation,

$$* \; L(a, \alpha, b, \beta, \xi) = \frac{1}{2} a^T a - \sum \alpha_i \{(a^T y_i + b) z_i - 1 + \xi_i\} - \sum \beta_i \xi_i + C \sum \xi_i$$

$$L(a, \alpha, b, \beta, \xi) = \frac{1}{2} a^T a - \sum \alpha_i a^T y_i z_i - \sum \alpha_i z_i + \sum \alpha_i - \sum \alpha_i \xi_i - \sum \beta_i \xi_i + C \sum \xi_i - \sum \xi_i (\alpha_i + \beta_i)$$

Now, we make

$\nabla_\xi L = 0$, $\nabla_a L = 0$, $\nabla_b L = 0$

$\alpha_i + \beta_i = C$; $a = \sum_i \alpha_i z_i y_i$; $\sum \alpha_i z_i = 0$

$0 \le \alpha_i < C$

$\Rightarrow L(\alpha) = \sum \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_i \alpha_k z_i z_k y_i^T y_k$

* Maximize this.

- We just have an additional constraint here, $\rightarrow 0 \le \alpha_i < C$ ∀.
- We have a similar formulation as before.

17

* Transductive SVM:

- We learn from training & test data simultaneously (Transductive)

→ This is a weak semi supervised setup

→ Normal SVMs till now were Induction based → fully supervised.

- Consider $x_1, \ldots, x_n$ } Training data
  $x_{n+1}, \ldots, x_m$ } Test data.

→ $\arg \min_{z_{n+1}, \ldots, z_m} \left(\arg \min_{a, \xi, \eta, b} \left(\frac{1}{2} a^T a + C \sum_{i=1}^{n} \xi_i + D \sum_{i=n+1}^{m} \eta_i \right) \right)$

Constraints $z_i (a^T y_i + b) \ge 1 - \xi_i$, $\xi_i \ge 0$ ∀ $i = 1, \ldots, n$

$z_i (a^T y_i + b) \ge 1 - \eta_i$, $\eta_i \ge 0$ ∀ $i = n+1, \ldots, m$

⇒ We do this iteratively

- Assume $z_{m1} \dots z_m$ and then minimize others
- Assume others are constant & minimize $z_{m1}, \dots, z_m$.

• Repeat those 2 steps.

---

* Multi Category SVM:

- One vs All (other class) } n SVMs
- One vs One } $n \times (n-1)$ SVMs
  (Each other class)

$a_i^* y_k - a_j^* y_k > 0$

⇒

---

* Non Linear SVM:

• Linear classification in non linear higher dimensional space.

$$\arg \max_{\alpha_1, \dots \alpha_n} \sum_{k=1}^{n} \alpha_k - \sum_{k=1, j=1}^{n} \alpha_k \alpha_j z_k z_i \phi(y_k)^T \phi(y_j)$$

⇒ ϕ = Mapping of data point to another space.

* Kernels:

*

- Kernel or Gram Matrix

$$K(k,j) = \phi(y_k)^T \cdot \phi(y_j)$$

$$= \langle \phi(y_k), \phi(y_j) \rangle$$

↓ $n \times n$ matrix

→ Inner Product

$\Rightarrow K \Rightarrow$ positive semi-definite matrix } Eigen valu

↳ Symmetric

↘ Desired but not always the

- Any matrix satisfying symmetric & positive semi definite matrix behaviour is a kern

i) Linear Kernel

$$K(k,j) = y_k^T y_j$$

ii) Polynomial Kernel

$$K(k,j) = (1 + y_k^T y_j)^p$$

* iii) Gaussian / Radial Basis Function (RBF) Kernel

$$K(k,j) = \exp\left(-\frac{\|y_k - y_j\|^2}{2\sigma^2}\right) = \exp(-\gamma \|y_k -$$

γ

iv) Sigmoid Kernel

$$K(k,j) = \tanh(\beta_0 y_k^T y_j + \beta_1)$$

(This is not positive semi definite)

- If ϕ is over complicated, we can end up overfitting.

$\Rightarrow$ With kernels we can avoid having to define ϕ explicitly (Useful at times)

* Kernel SVM: (example at the back of the book)

- Only consider support vectors → So efficient where there is a lot of data
- Overfitting can be controlled by soft margin approach
- Flexibility to choose any kernel.

* ⟹ One big issue is outliers, if they get selected as support vectors, we end up detecting completely wrong boundaries.

- Using just the kernel matrix K we can find out the SVMs parameters.

18/1

* Kernel Trick: (Project lower dimensional data to higher dimensions or infinite dimensional data to finite dimensions)

Note: $K = X^T X$ } Kernel Matrix

* $K = U \Lambda U^T$ } Eigen decomp

Every kernel should satisfy this ↗

$= U \Lambda^{1/2} \cdot \Lambda^{1/2} U^T$

$= U \Lambda^{1/2} \cdot (U^T \Lambda^{1/2})^T$

$= \phi^T \cdot \phi$, $Y = U^T \Delta^{1/2}$

$\phi : X \to Y$

- At times input data could be text etc, (ie. input dimension is variable & large), we often use various functions to compute the kernel matrix
  - ↳ ex: Edit Distance or Num of common substrings

* For strings.

↳ Kernel for documents
↳ Bag of words, TFIDF

↳ Kernels for graphs
↳ Counting matching random walks

- We can join up multiple kernels one after another and prove they are also kernels (Slides for example)

---

* Kernelized KNN:

- Instead of $\|x_i - x_j\|^2$,

$$\|\phi(x_i) - \phi(x_j)\|^2 = -2K(x_i,x_j) + K(i,i) + K(j,j)$$

---

* Kernelized PCA: (Slides also has the proof)

- $X \in R^d$, $\rightarrow S_{d \times d}$ scatter matrix

$\Rightarrow Y = \phi(X)$, $Y_i = \phi(x_i) \in R^p$

$$S' = \sum_i^n (\phi(x_i) - m')(\phi(x_i) - m')^T \quad \} \text{ New scatter matrix}$$

(Assume $m' = 0$) (Centered data)

$$\Rightarrow S'_{pp} = \sum_i \phi(x_i)\phi(x_i)^T \rightarrow (1)$$

- $S'_{pp} w_{pi} = \lambda w_{pi}$ } Over new eigen vectors w, with eigen values λ
  → we want to find these w (with $w^T w = 1$, etc.)
- Replacing (1) in above,

(We don't know this scatter matrix since we don't know 'ϕ' function)

$$\frac{1}{\lambda}\sum_i \phi(x_i)\cdot\phi(x_i)^T w = w \rightarrow (2)$$

$$\Rightarrow \sum_i \phi(x_i)\cdot\frac{\phi(x_i)^T w}{\lambda} = w$$

$$\Rightarrow \sum_i \phi(x_i)\cdot\alpha_i = w \rightarrow (3), \quad \alpha_i = \text{Scalar value}, \quad \alpha_i = \frac{\phi(x_i)^T w}{\lambda}$$

- Substitute (3) in (2)
- $\left(\sum_i \phi(x_i)\cdot\phi(x_i)^T\right)\sum_k \phi(x_k)\cdot\alpha_k$

$$= \lambda\sum_k \phi(x_k)\cdot\alpha_k$$

$\Rightarrow$ Multiply both sides by $\phi(x_l)^T \; \forall l$

- $\sum_{i,k,l} \phi(x_l)^T\phi(x_i)\cdot\phi(x_i)^T\cdot\phi(x_k)\alpha_k$

$$= \lambda\sum_{k,l} \phi(x_k)^T\phi(x_l)^T\alpha_k$$

$$\Rightarrow K\cdot K\cdot\alpha = \lambda\cdot K\cdot\alpha$$

$$K\cdot\alpha = \lambda\cdot\alpha, \quad \alpha^T\alpha = 1$$

$\rightarrow$ we learn α's using kernel matrix

* Kernel Matrix (we know this)

From (3),

- Now, $W'_{px1} = Y_{pxn}\cdot\alpha_{nx1}$ } we then get w using our new α's.

Y_{pxn}: All inputs $\phi(X)$, dxn

Now,

- $w^T w = \alpha^T Y^T Y\alpha$

$$= \alpha^T K\alpha = \lambda\alpha^T\alpha = \lambda$$

We want this

$$\Rightarrow \frac{w^T w}{\lambda} = 1, \quad \hat{w} = \frac{Y\alpha}{\sqrt{\lambda}}$$ } These are the w's we need.

$\therefore \hat{w}^T\hat{w} = 1$

Note: The Kernelized methods are very
** useful since a lot of times input
data might not be vectorizable,
strings, graphs etc. We define a
kernel based on some comparison
and then using that we can get
vectorizable data.

— x —

Note: Kernel LDA, Multiple Kernel Learning,
** Non-linear dimensionality reduction.

— x —

Note: Mid 2 — 2.1–2.3, 2.5, 2.6, 28.3
(Duda book) 3.1, 3.2, 3.3, 3.4, 3.5.1, 3.7, 3.8
5.11, 5.12

— x —

19

Data Clustering:

(Unsupervised, no labels given)

Similarity Measures (between datapoints)
- Cosine distance
- Jaccard distance (Sets) $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$
- Minkowski distance (City-block & Euclidean)
- Mahalanobis distance

- Normalization is not always helpful. (Context dependent)

* Criterions: (Minimization approaches)

1) Sum of squared error: (Not always the best criterion)

↳ $J_e = \sum_{i=1}^{c} \sum_{x \in D_i} \|x - m_i\|^2$, m_i = Mean of i^{th} cluster

(For each cluster)

- Minimizing this via optimization is good but not always the best. (example in slides)

2) Related minimum variance:

* $J_e = \frac{1}{2} \sum_{i=1}^{c} n_i \bar{s}_i$, $\bar{s}_i = \frac{1}{n_i^2} \sum_{x \in D_i} \sum_{x' \in D_i} \|x - x'\|^2$

(J_e: Minimize.)

$\left(s_i = \frac{1}{n_i^2} \sum_{x \in D_i} \sum_{x' \in D_i} s(x, x') \quad , \quad s_i = \min_{\substack{x \in D_i, \\ x' \notin D_i}} s(x, x') \right)$

— x —

* Hierarchical Clustering: (Agglomerative)

* 1) • Agglomerative clustering (non-parametric model) (non generative)
  ↓
  Bottom Up

⇒ Initially assume all datapoints belong to different clusters.

↳ Keep merging the nearest 2 clusters (Keep doing this till you get required k clusters)

Cluster Distance measures

* * ↳ $d_{min}(D_i, D_j) = \min_{x \in D_i, x' \in D_j} ||x - x'||$

↳ $d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{x' \in D_j} ||x - x'||$

↳ $d_{mean}(D_i, D_j) = ||m_i - m_j||$

⇒ Using $d_{min}(D_i, D_j)$ for merging of clusters by linking the 2 points yields an MST (Minimum Spanning Tree) solution.

- Using d_{max} → we end up getting fully connected subgraphs (large clusters)

⇒ For d_{avg}, we can use any other similarity measure we like.

- d_{mean} = Computationally simple.

—— x ——

2) K-Means Clustering

** (Iterative method) (example in slides)

- Initialize means randomly
- Then iterate between

  E-step → Assign each datapoint to nearest mean.

  M-step → Update cluster means.

$$\Rightarrow J_e = \sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij} \cdot \|x_i - \mu_j\|^2 \quad , a_{ij} \in \{0,1\}$$

(For hard clustering)

$$\left(\sum_{j=1}^{k} a_{ij} = 1 \ \forall\, i\right)$$

(Each point is assigned to only one cluster)

- We assume that each of the k clusters have same variance so that we can move points into clusters with the nearest mean. So our criterion error function is correct.

$$\Rightarrow \nabla_{\mu_j} J_e = 0 \text{ , To minimize } J_e .$$

$$* \quad \sum_{i=1}^{n} a_{ij} \cdot (x_i - \mu_j) = 0 \quad \forall\, j \in \{1 \dots k\}$$

$$\Rightarrow \mu_j = \frac{1}{n_j}\sum_{i=1}^{n} x_i \cdot a_{ij}$$ subject to the fact that a_{ij} is known

$$\mu_j = \frac{\sum_{i=1}^{n} x_i \cdot a_{ij}}{\sum_{i=1}^{n} a_{ij}}$$

- Similarly if you fix μ_j's, then we can optimize on a_{ij}.

Limitations ⇒ This is converged to local minima, so sensitive to initialization. Sensitive to outliers.

⇒ We also always assume our clusters to be convex, so fails for certain distributions
- Complexity is also quite high $O(N \cdot K \cdot d \cdot L)$.

* a) Probabilistic Kmeans:

* - Instead of hard classification, we assume a probabilistic assignment of datapoints to clusters.

! $0 \leq a_{ij} \leq 1$. (Instead of only 2 possible values) (This is tough so we use another formulation)

⇒ We represent probability mixture of data as a gaussian mixture model

$$p(x) = \sum_{k=1}^{K} \pi_k \cdot N_k(x, \mu_k, \Sigma_k), \quad \sum_{k=1}^{K} \pi_k = 1, \quad 0 \leq \pi_k$$

(Multivariate gaussians)

↑ Given all μ_k and Σ_k.

$$p(x) = \sum_{k=1}^{K} p(k) \cdot p(x|k)$$

Note: K can be determined using Bayesian mixture model (The number of clusters)

⇒ Here its similar to MLE except that we want this $p(x)$ to be maximized.
$(\log(p(x)))$ → Maximize.

* - $l(\Theta) = \ln(p(X|\Theta_k))$ → All datapoints
$= \ln\left(\sum_H p(x, H|\Theta)\right)$
$= \ln\left(\sum_H \frac{q(H)}{q(H)} \cdot p(x, H|\Theta)\right)$

- We intro H as a missing para

⇒ log of summation is hard to compute

⇒ Jensen's Inequality:

* $\log E(x) \geq E[\log x]$ //

Using jensen's equality,

$$\ell(\Theta) \geq \sum_H q(H) \ln\left(\frac{p(X,H|\Theta)}{q(H)}\right)$$

$$\ell(\Theta) \geq \sum_H p(H|X,\Theta) \cdot \ln\left(\frac{p(X,H|\Theta)}{p(H|X,\Theta)}\right)$$

$$\ell(\Theta) \geq \sum_H p(H|X,\Theta) \cdot \ln(p(X|\Theta))$$

↑ Basically Maximize RHS so that the lower bound of $\ell(\Theta)$ increases.

(Let, $q(H) = p(H|X,\Theta)$

$q(H) = \frac{p(H,X|\Theta)}{p(X|\Theta)}$)

GMM

- Given datapoints, we try to optimize and achieve μ_j's, Σ_js and π_js.

→ This is done using what we derived above, optimizing the RHS sum of logs would optimize our $\ell(\Theta)$ which is what we want.

——x——

b) Fuzzy K-Means: (Slides)

*
- Points belong to a cluster in a fuzzy manner. Points near cluster boundary belong lesser to the cluster etc.

C) Kernel K-Means:

* • Useful when data has ∞ dimensions etc where we cannot compute the means.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{K} a_{ij} \| \phi(x_i) - \tilde{\mu}_j \|^2$$

$$\tilde{\mu}_j = \frac{\sum_{i=1}^{N} a_{ij} \phi(x_i)}{\sum_{i=1}^{N} a_{ij}}$$

⇒ We move on to kernalized implementation so that we know $\phi(x_i) \cdot \phi(x_j)$ but not ϕ individually. (Derivation back of notes)

Note: Read up on Mixture Models → Imp for exams.

---

*3) Spectral Clustering:

* • Apply graph based analysis to move classes into blobs so we can use k-means directly. (Projection onto another euclidean space)

⇒ Initially form a graph from all data points and perform 'graph partitioning'.

* • Graph construction → K-nearest neighbour link them from datapoints.

(or ε neighbourhood (sphere & all neighbours within

(Note: Semi supervised methods also specify if 2 points must belong to a class or must not belong)

a) Mincut Graph Partitioning:

⇒ 2 way partitioning (2 clusters)

- We want to divide it into clusters such that we minimize these

→ $RatioCut(C_1, \dots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{cut(C_i, \bar{C}_i)}{|C_i|}$

* → $NormCut(C_1, \dots, C_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{cut(C_i, \bar{C}_i)}{vol(C_i)}$

($\bar{C}_i$ = All other points apart from cluster C_i)

- This is NP, so we can't solve it. We move on to a relaxed version

* b) Graph Projection for Partitioning: (Proof in slides)

* - We define a function f which gives a value for each node in the graph.

$$f(v_i) = \begin{cases} \sqrt{|\bar{C}|/|C|} & \text{if } v_i \in C \\ -\sqrt{|C|/|\bar{C}|} & \text{if } v_i \in \bar{C} \end{cases}$$ } we will prove why this is the right f next.

⇒ So the ratio cut criterion here is (k=2)

~~RatioCut(C)~~

⇒ $|V| \cdot RatioCut(C, \bar{C}) \Rightarrow (|C| + |\bar{C}|) \left\{ \frac{1}{2} \frac{Mincut(C, \bar{C})}{|C|} + \frac{1}{2} \frac{Mincut(\bar{C}, C)}{|\bar{C}|} \right\}$

(we want to minimize this)

$$\Rightarrow \frac{1}{2}\,\text{Min cut}(c,\bar{c})\left\{\frac{|c|+|\bar{c}|}{|c|}+\frac{|c|+|\bar{c}|}{|\bar{c}|}\right\}$$

$$\Rightarrow \frac{1}{2}\,\text{Min cut}(c,\bar{c})\left\{\frac{|\bar{c}|}{|c|}+\frac{|c|}{|\bar{c}|}+2\right\}$$

$$\Rightarrow \frac{1}{2}\sum_{\substack{j\in c\\ j\in \bar{c}}} w_{ij}\left(\sqrt{\frac{|\bar{c}|}{|c|}}+\sqrt{\frac{|c|}{|\bar{c}|}}\right)^2+\frac{1}{2}\sum_{\substack{i\in\bar{c}\\ j\in\bar{c}}} w_{ij}\left(-\sqrt{\frac{|\bar{c}|}{|c|}}-\sqrt{\right.$$

Minimize wrt f. $\Rightarrow \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(f(v_i)-f(v_j)\right)^2$ } We want to minimize this (Quadratic form)

$\longrightarrow$ (1)

- Retrieving the exact f is also an NP problem so we relax it and try to solve

∴ We have showed that $f(v_i)$ is what we assumed it should be initially.

$$\sum_{i=1}^{n} f(v_i)=0, \quad \|f\|=\sqrt{\sum_{i=1}^{n} f(v_i)^2}$$

$f \perp 1$ } f is orthogonal to $[1,1,\ldots]$ vector.

$$\|f\|=\sqrt{|c|+|\bar{c}|}=\sqrt{n}$$

$\Rightarrow$ Graph Incidence Matrix:

$$** \quad \underbrace{\nabla_{ij}}_{\nabla_{ev}}=\begin{cases}\nabla_{ev}=-1 & \text{if } v \text{ is initial vertex of } e\\ \nabla_{ev}=1 & \text{if } v \text{ is terminal vertex}\\ \nabla_{ev}=0 & \text{otherwise}\end{cases}$$

(Each row corresponds to an edge)

(Sum of each row=0

$\Rightarrow$ This matrix is like the derivative.

$$(\nabla f)(e_{ij})=f(v_j)-f(v_i)$$

$L=\nabla^T\nabla$ } Laplace (2nd derivative)

$\Rightarrow (Lf)(v_i) = \sum_{v_j \sim v_i} (f(v_i) - f(v_j))$

$L = D - A$, D = Degree matrix $D_{ii} = d(v_i)$

- Diagonal entries of L are degree of vertices and the other entries represent the adjacency matrix.

$$(Lf)(v_i) = \sum_{v_j \sim v_i} w_{ij}(f(v_i) - f(v_j))$$

Quadratic form, (Minimize this) (from (1))

$$f^T L f = \frac{1}{2} \sum_{e_{ij}} w_{ij}(f(v_i) - f(v_j))^2$$

(L = +ve semi definite & symmetric)

Finally, we want to optimize this.

$\Rightarrow \arg\min_f \frac{f^T L f}{f^T f}$

- We now know that this is nothing but requiring us to find eigen vectors & values of L.

($\lambda_{min} = 0$, (the smallest λ)) for a connected graph)

$\Rightarrow$ If we want to project to k dimensions
$*$ we use multiple eigen vectors of L. (smallest non zero eigenvalues corresponding (Non zero corresponding eigen vectors))

(Note: If we have multiple connected components in graph, $L = \begin{bmatrix} L_1 & \cdots & 0 \\ 0 & & L_n \end{bmatrix}$ where each L_i is a laplacian matrix for that connected component i.

$\Rightarrow$ Once we project these points, we can apply simple clusterings like k-means etc to get good clustering.

Decision Trees.

(Classification)

- Nominal data (categorical) is used. We don't use vectorial data.
- Very useful in businesses etc since they can provide stepwise explanation for the outcome we chose.

⇒ { Decision trees → Target is continuous (Regression tree)
* → Target is discrete (Classification tree)

(examples in slides)

- Leaf nodes of tree corresponds to target value.

* (Decision nodes specify test on a single attribute)

⇒ We stop splitting nodes when

* → i) All elements of training at this point at the node belong to the same class. (node is pure)
→ ii) we have used up all attributes (We can also use pairs of attributes etc if we want) (we also need to take care that we don't overfit)

- Every decision tree can be represented as a binary tree. We test (attr = val ?), and branch as yes/no. These are CART trees (Classification & Regression Trees).

———— x ————

Note: If we choose single attributes in each test,

* then we end up with monothetic trees & boundaries are just multiple hypercubes, not generalized enough.

*Classification & Regression Trees: (CART)

*

- Purity of node = Similarity of target values of training data at that node.

⇒ To split a node, we choose an attribute which gives us higher purity in descendents.

i) • Entropy Impurity $= -\sum_j P(\omega_j) \cdot \log_2 P(\omega_j)$ (ω_j: Class label.)

* (Entropy=0 for a pure node)

ii) Gini Impurity $= \sum_{i \neq j} P(\omega_i) \cdot P(\omega_j) = \frac{1}{2} - \sum_j \frac{P^2(\omega_j)}{2}$

iii) Misclassification Impurity $= 1 - \max_j P(\omega_j)$

- We can also say we choose an attribute which maximizes the decrease in impurity (~~increase~~ maximize information gain) ($i(X)$ = Impurity at X)

$$\Delta i(N) = i(N) - P_L i(N_L) - (1-P_L) i(N_R)$$

- This is a greedy approach & finds local optimals.

* • Having branching factor >2 decreases the impurity more. So we tend to use gain ratios as well (slides)

⇒ We stop splitting based on thresholds the * amount of information gain by best attributes split or the amount of purity ~~of~~ or height of tree etc.,

- exc: Global Criterion : $\alpha \cdot$ size of tree $+ \sum_{N \in \text{leaf nodes}} i(N)$

* Pruning:

* • When we stop splitting too early, its called the
* Horizon Effect

• In order to account for this, using the global criterion won't work, so what we do is build a complete tree (overfit) and then start merging nodes based on criterions (pru

• This is computationally expensive.
• The merging of nodes provides good interpretabi

—x—

⇒ We then need to assign labels for impu leaf nodes. Majority voting does this.

• We perform PCA first & then use them in
* decision trees so that the trees are simple and don't overfit (example in slides)
(we don't want correlation in data)

• At nodes we can use multi attribute functi

⇒ Handling class priors etc can be done
* by modifying impurity formulae,

ex: $i(N) = \sum_{ij} \lambda_{ij} P(w_i) P(w_j)$

* Missing Attributes: (ex: Slides)

• At a node if primary attribute is missing we define surrogate splits by considering other attributes such that the correlation with primary attribute split is the highest.

—x—

Limitations:

- Finding best decision tree is an NP-complete problem
- Greedy just gives us a local optima.
-

Note:

* • ID3 tree (a kind of decision tree)
  - → Branches from a node = No. of values per attribute
  - → Real valued attributes are binned.
  - → Stop when leaf nodes are completely pure.
* • C4.5
  - → ID3 for categorical attributes
  - → CART for real valued attributes.

? • Bagging, Random forest, Rotation forest

---

*Bayesian Nets:

- Graphical Model = Graph Theory + Probability Theory.
  * → Directed → Bayes Nets
    → Undirected → MRFs

- (A) → (B) indicates that B is caused by A.

$$P(B, A) = P(B|A) \cdot P(A)$$

- Top-Down } Reasoning
  Bottom-Up } Diagnostic

⇒ Conditional Independence:

* i) (A) → (B) → (C) (Head to tail)

- (C) is conditionally independent on (A) given (B).

ii) (B) → (C), (B) → (A) (Tail to tail)

- (C) is conditionally independent on (A) given (B).

* iii) * (A) → (C), (B) → (C) (Head to Head)

- If (C) is not observed (A), (B) are conditionally independent. But if (C) is observed then (A) depends on (B), (C) and (B) depends on (A), (C)

(Explaining Away)

⇒ **Arc Reversal:** (Examples in slides)

- Based on Bayes,

$$p(x_1, x_2) = p(x_1|x_2) \cdot p(x_2)$$
$$= p(x_2|x_1) \cdot p(x_1)$$

(we flipped an edge)

⇒ we could also add a new edge and flip another.

———— x ————

* **D-Separation:** (Determine if two variables are conditionally independent)

*

- Two variables (A) & (B) are conditionally independent if,
  - ↳ There is a node (V) which is observed and all paths b/w A & B go through (V) or start at (V).

(A) ⋯→ (V) →⋯ (B) or (A) ⋯→ (V) ←⋯ (B)

———— x ————

* **Note:** (Message Passing) Given a network and a few nodes where data is observed, we calculate the probabilities of all other nodes by moving to neaby nodes of these nodes. (BFS like)

(we can either move downwards reasoning or move upwards diagnostic to get probabilities of all unknown nodes)

———— x ————
———— x ————