# Lecture 2 - KNN

Given training data $D = \{(x_i, y_i)\}_{i=1}^{N}$

To classify new observation

1. Define distance measure (Euclidean)
2. Find Nearest neighbor for all new data points
3. Label them w/ label of their k nearest neighbor (pick the majority)

$$p(y=c \mid x, k) = \frac{1}{K} \sum_{i \in N_k(x)} I(y_i = c) \qquad \text{where } I(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

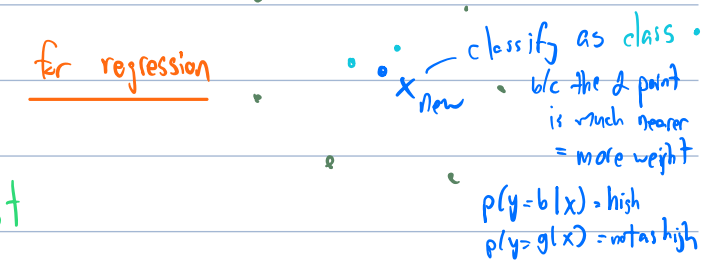
$N_k(x)$: k-NN of vector x

## k-NN w/ weighted Classification

Pick the weighted majority label. Weight is inversely proportional to distance (closer = bigger weight)

$$p(y=c \mid x, k) = \frac{1}{\Sigma} \sum_{i \in N_k(x)} \frac{1}{d(x, x_i)} I(y_i = c) \qquad d(x, x_i) = \text{distance btw } x \text{ and } x_i$$

$$\hat{y} = \arg\max_c p(y=c \mid x, k) \qquad \hat{y} = \frac{1}{\Sigma} \sum \frac{1}{d(x, x_i)} y_i \text{ for regression}$$




Hyperparameter - k - pick k that generalise the best

Distance Measure — $L_2$ norm (Euclidean) $= \|u-v\|_2 = \sqrt{\sum_i (u_i - v_i)^2}$ | $L_1$-norm $= \sum_i |u_i - v_i|$

$L_\infty$ norm $= \max_i |u_i - v_i|$ | Angle $= \cos\theta = \frac{u^T v}{\|u\| \|v\|}$ | Mahalanobis $: \sqrt{(u-v)^T \Sigma^{-1} (u-v)}$ ($\Sigma$: PSD and symmetric)

## Measure Classification Performance

- Confusion Table

| Predict \ Actual | T | F |
|---|---|---|
| T | TP | FP |
| F | FN | TN |

- Accuracy - $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision - $\frac{TP}{TP+FP}$
- Recall / Sensitivity (Fraud) - $\frac{TP}{TP+FN}$
- Specificity - $\frac{TN}{FP+TN}$
- FNR (rate) - $\frac{FN}{FN+TP}$
- FPR - $\frac{FP}{FP+TN}$

F1 - Score - $\frac{2 \cdot \text{precision} \cdot \text{recall}}{}$

precision + recall

## Scaling issue

Data standardization - Scale each feature (col) to zero mean and unit variance

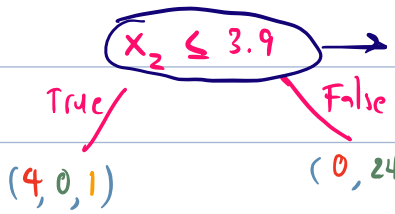
$$x_i = \frac{x_i - \mu_i}{\sigma_i}$$

## The Curse of Dimensionality

The higher the dimension, the further away the NN are. If the NN are too far away, then there's no notion of neighbor and NN doesn't work well.

# of data has to grow exponentially with the # of feature (dim)

## Lecture 2: Decision Tree

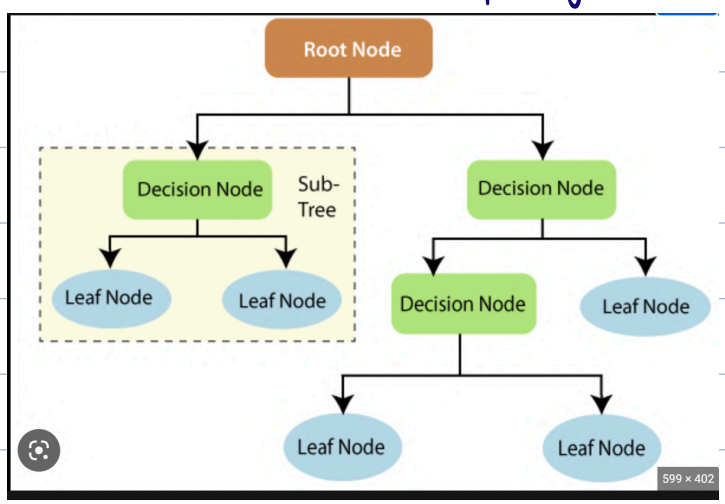



Node - Feature Test → lead to decision boundary (usually only check 1 feature at a time

eg. no $x_1 > 2$ & $x_2 < 3$)

Branch - Subsection of decision tree

Leaf - Node that doesn't split any further. Produce prediction or distribution




To classify new sample x

1. Find region R that contain x and get the class distribution $n_R = (n_{c_1,R}, n_{c_2,R} \ldots n_{c_K,R})$

2. The prob that a data point $x \in R$ should be classified to class c in region R is (each)

$$p(y=c \mid R) = \frac{n_{c,R}}{\sum_{c' \in C} n_{c',R}}$$

3. Sample x class = class that is the most common in the corresponding region

$$\hat{y} = \arg\max_c p(y=c|x) = \arg\max_c p(y=c|R) = \arg\max_c n_{c,R}$$

If region = (r, g, b) = {4, 10, 6} → $p(r) = \frac{4}{4+10+6} = \frac{4}{20} = 0.2$

$p(b) = \frac{10}{20} = 0.5$ $p(g) = \frac{6}{20} = 0.3$

Naive Idea - Build all possible tree and test it on validation dataset to evaluate performance

Heuristic

Impurity measure let $\pi_c = p(y=c|t)$ ← prob of picking class c in dataset t

$i(t) = 0$ → node is pure $i(t) = 1$ = class are equally distributed

1. Misclassification Rate → $i_E(t) = 1 - \max_c \pi_c$
2. Entropy → $i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log_2 \pi_{c_i}$ (let $\log_2 0 = 0$)
3. Gini Index → $i_G(t) = \sum_{c_i \in C} \pi_{c_i} (1 - \pi_{c_i}) = 1 - \sum_{c_i \in C} \pi_{c_i}^2$

$\pi_{c_i}$: prob of picking element; $(1-\pi_{c_i})$: prob of misclassified

Measure how often a randomly chosen instance would be misclassified if it was classified according to class distribut

The improvement when performing split feature s of dataset t into $t_R$ and $t_L$ for $i(t) = i_E(t)$

$$\Delta i(s,t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

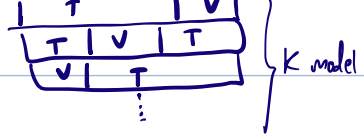
relative size of left & right region compare to before split

$p_L = p_R =$ data after latest split / total data before the latest split

Overfitting

- Poor generalisation
- Training performance increase w/ every split, Validation error got higher.
- Validation perf. tells us how well our model generalise, not training perf.

## K-fold Cross Validation




- Split learning data into K folds
- take one group as validation and use K-1 folds for training
- Train model K times w/ diff division of training & validation data
- Avg the error over all model
- Try w/ diff setting of hyperparameter & model →
- Use all your training model and best hyperparameter for final training and testing of model

## LOOCV – Leave one out Cross Val..

- Train all but one sample → if N sample → N-fold cross validation → train model N time

## Stopping Criteria (Pre-pruning)

- Distribution is pure ($i(t)=0$)
- accuracy on validation set
- Max depth reach
- # samples in each branch below threshold $t_n$
- benefit of splitting is below certain threshold $\Delta i(s,t) < t_\Delta$

## Post Pruning → Reduce error pruning

Let T be our decision tree and t one of its inner nodes

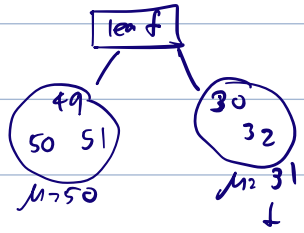
Pruning T wrt. to $t$ $(T \setminus T_t)$ mean deleting all descendant nodes of t (but not node t itself)

- $err_{D_v}(T)$ = validation error of whole tree
- $err_{D_v}(T \setminus T_t)$ = validation error of tree w/o descendant of t
- Prune tree at the node that reduce the validation error the most → reduce overfitting

  until for all nodes t $err_{D_v}(T) < err_{D_v}(T \setminus T_t)$

## DT for regression (if $y_i$ is real value rather than class)

- Compute mean at the leave
- Use MSE as splitting heuristic
- purity – all value are closed to the mean




$$MSE = \frac{1}{2}\left[(30-31)^2 + (32-31)^2\right] = 1$$

# Ensemble

- Idea - Aggs prediction of many (diverse) classifier to improve perf by reducing variance of the model by averaging

Bagging - Create new dataset by sampling training set
- Train seperate classifiers for each dataset
- Combine predictions (Majority vote or average)

Baosting - Incrementally train (weak) classifier that correct previous mistakes
- Give higher weight (Focus more) on misclassified example

Stacking - Train meta-classifier w/ base classifiers' predictions as feature

Bucket of models

## Random Forest

- Bagging + Tree
- can be highly correlated
- reduce benefit of bagging since we need diverse classifier (for bagging)

Idea - Use only a subset of randomly sampled data and feature to learn each tree
(bagging at instance level (subset of sample) + bagging at feature level (subset of feature))

## Boosting : AdaBoost and XGBoost

Incrementally train weak learners (eg. one-level-trees)

Initialise weight vector w/ uniform weight

Loop → train weak learner on weighted example → Increase weight for misclassified example

Predict the (error-based weighted) majority

# Lecture 3: Probabilistic Inference

Assume data is i.i.d (independent identical distribution)

D Data and parameter $\theta$ (could be probability of a coin toss)

## Maximum Likihood Estimator (MLE)

$$\theta_{MLE} = \max_{\theta} p(D|\theta) \rightarrow \max_{\theta} \log(p(D|\theta)) \rightarrow \text{Point estimate}$$

find $\theta$ that maximize the likelihood function $p(D|\theta)$ (most likely to see observed data D)

The likelihood func is not prob distribution over $\theta$ since $\int p(D|\theta)\, d\theta \neq 1$ in general

## MLE for any coin toss

$$\theta_{MLE} = \frac{|T|}{|T|+|H|}$$

find prob that next coin flip is T $\rightarrow$ $F_{11} \sim B(\theta_{MLE})$

For Bernoulli, probability = parameter

$$F_{11} \sim B(\theta_{MLE}) = p(F_{11}=T|\theta_{MLE}) = Ber(F_{11}=T|\theta_{MLE}) = \theta_{MLE} = \frac{|T|}{|T|+|F|} = \frac{3}{3+7} = 30\%$$

But MLE doesn't consider prior knowledge $\rightarrow$ Bayesian Inference

## Bayesian Inference

$p(\theta)$ prior distribution represent our belief before we observe any data

Constraint for $p(\theta)$

- $p(\theta)$ must not depend on the data right now
- $p(\theta) \geq 0$ for all $\theta$
- $\int p(\theta)\, d\theta = 1$

$p(\theta|D)$ is the posterior distribution. Update our belief in the value of $\theta$ after observing data

$$p(\theta|D) = \frac{P(D|\theta)\,P(\theta)}{P(D)}$$

$P(D|\theta)$ = likelihood func

$P(\theta)$ = prior distribution

$p(D)$ = evidence $\rightarrow$ act like normalizing constant (so that $\int P(\theta|D) = 1$)

$\hookrightarrow$ we can obtain evidence using sum rule of prob.

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

Posterior $\propto$ Prior $\cdot$ Likelihood

Observing more data $\rightarrow$ less effect on prior $\rightarrow$ distribution lead toward current measurement more (closer to MLE)

If $p(\theta)=0$ → can't update posterior prob. → always be 0

## Maximum a posterior estimation (MAP) & estimating Posterior Distribution (Fully Bayesian)

MLE - $\theta_{MLE} = \arg\max_\theta p(D|\theta) \rightarrow \max_\theta \log p(D|\theta)$

→ return $\theta$ that give the highest likelihood to see data D

→ ignore prior belief & perform poorly if little data available

MAP - $\theta_{MAP} = \arg\max_\theta p(\theta|D) \rightarrow \max_\theta \log p(\theta|D)$

→ return $\theta$ - the maximizer of the posterior distribution (most probable $\theta$ in the distri. given the data D)

→ $\theta_{MAP}$ is a point estimate and $p(\theta_{MAP}|D)$ is a highest probability (mode) in posterior distribution

**Fully Bayesian Approach** - Compute the posterior distribution instead of just mode $\theta_{MAP}$

With an entire distribution, we can answer question such as

- Where is the mean under $p(\theta|D)$ → not at maximizer b/c not symmetric
- Variance of $p(\theta|D)$

**Goal**: get $P(\theta|D)$ dist. by finding suitable $p(\theta)$ and normalizing constant $p(D)$ either by brute force or pattern matching

brute force: $\int_0^1 p(\theta|D)\, d\theta$ ; pattern matching: conjugate prior

choose $p(\theta)$ which is a conjugate prior of the likelihood function → The resulted Posterior Dist. will be in the same family as the prior (might not be the same distri as prior always)

(Pattern Matching)

e.g. likelihood = $p(D|\theta) \sim Ber(\theta) = \theta^{|T|}(1-\theta)^{|H|}$

Conjugate prior distribution = $p(\theta) = p(\theta|a,b) \sim Beta(a,b)$

So $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \propto p(D|\theta)\cdot p(\theta) = \theta^{|T|}(1-\theta)^{|H|} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$

$\propto \theta^{|T|+a-1}(1-\theta)^{|H|+b-1}$

$\theta_{MAP} = \arg\max_\theta p(\theta|D) = \arg\max_\theta \theta^{|T|+a-1}(1-\theta)^{|H|+b-1} = \frac{|T|+a-1}{|H|+|T|+a+b-2}$

$p(\theta|D) = \theta^{|T|+a-1}(1-\theta)^{|H|+b-1}$ → not normalise → find $P(D)$ ⇒ pattern matching

($\alpha$ = $|T|+a-1$, $\beta$ = $|H|+b-1$)

$\text{Beta}(\theta|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$ → already normalise ←

So we know that, in order to normalise $p(\theta|D)$, we need to match $p(\theta|D)$ w/ Beta,

so the appropriate normalising constant = $\frac{\Gamma(|T|+a+|H|+b)}{\Gamma(|T|+a)\,\Gamma(|H|+b)}$

$$\therefore p(\theta|D) = \frac{\Gamma(|T|+a+|H|+b)}{\Gamma(|T|+a)\,\Gamma(|H|+b)} \theta^{|T|+a-1}(1-\theta)^{|H|+b-1} = \text{Beta}(|T|+a, |H|+b)$$

With more data, the posterior dist. become more peaky → more certain about our estimate $\theta$

## Posterior Predictive Distribution (predicting unknown data)

$p(f|D,a,b)$ = posterior predictive dist. $p(\theta|D,a,b)$ = posterior dist.

Using sum rule of probability

$$p(f|D,a,b) = \int_0^1 p(f,\theta|D,a,b)\,d\theta$$

$$= \int_0^1 p(f|\theta,D,a,b)\,p(\theta|D,a,b)\,d\theta$$

if we know $\theta$, the next prediction f is independent of previous data D,a,b

$$= \int_0^1 p(f|\theta)\,p(\theta|D,a,b)\,d\theta$$

basically a weighted average for each prediction

$p(\theta|D,a,b)$ act as a weight of a prediction, how much do you believe in the $\theta$ already

The prediction prob $p(f|\theta)$ is depend on $\theta$ and if the prob of $\theta$ showing up is small, then the overall predictive prob will be even lower.

MLE & MAP return $\theta_{MLE}$ & $\theta_{MAP}$ which can be use in the probability to get the highest probability

Fully Bayesian return full distribution of $p(\theta|D)$

As we observe lots of data, the diff in prediction btw this 3 method become less noticable

# Lecture 4 : Linear Regression

Data set $D = \{(x_i, y_i)\}_{i=1}^{N}$ $\quad X = \{x_1, \dots, x_N\}$ = observation $x_i \in \mathbb{R}^D$ $\quad X \in \mathbb{R}^{N \times D}$

$y = \{y_1, \dots, y_N\}$ = target $\quad y_i \in \mathbb{R}$

$y_i = f(x_i) + \varepsilon_i$ $\quad \varepsilon_i \sim N(0, \beta^{-1})$ ← noise

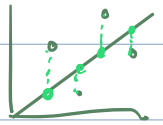
there's always a bias term

feature vector $\vec{x} = (1, x_1, \dots x_D)^T$ $\quad$ weight vector $\vec{w} = (w_0, w_1, \dots, w_D)^T$

Choose best w to fit the data

Loss function - measure error btw our model (parameterized by w) and observed data $D = \{(x_i, y_i)\}_{i=1}^{N}$

↳ 1) Least Square ⇒ $E_{LS}(w) = \frac{1}{2} \sum_{i=1}^{N} (f_w(x_i) - y_i)^2 = \frac{1}{2} \sum_{i=1}^{N} (w^T x_i - y_i)^2$



Find optimal $w^*$ to min error

$$w^* = \arg\min_w \frac{1}{2} \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

As matrix form → $w^* = \min_w \frac{1}{2} (Xw - y)^T (Xw - y)$

$$\nabla_w E_{LS}(w) = \frac{1}{2} \frac{\partial}{\partial w} (w^T X^T X w - 2 w^T X^T y + y^T y)$$

Note $\frac{\partial}{\partial x} x^T A x = 2Ax$ or $(A + A^T)x$ if A is not symmetric

($X^T X$ is symmetric)

$$= \frac{1}{2} (2 X^T X w - 2 X^T y) = X^T X w - X^T y = 0$$

$$w^* = \underbrace{(X^T X)^{-1} X^T}_{X^+} y$$ = Normal equation of LS problem

## Non linear Dependency

Polynomial - Universal function approx.

for 1D → $f_w(x) = w_0 + \sum_{j=1}^{M} w_j x^j$

or more generally $f_w(x) = w_0 + \sum_{j=1}^{M} w_j \phi_j(x) = w^T \phi(x)$

where $\phi(x)$ is non linear, but the whole function $f_w(x)$ is still in linear form

$\phi(x)$ can be eg. 1. Polynomial - $\phi_j(x) = x^j$ $(x, x^2, x^3, \dots)$

2. Gaussian - $\phi_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2s^2}\right)$

3. Sigmoid - $\phi_j(x) = \sigma(a) = \frac{1}{1 + e^{-a}}$

use along with $E_{LS}(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - y_i)^T = \frac{1}{2} (\phi(x) w - y)^T (\phi(x) w - y)$

and $w^* = \min E_{LS}(w) = (\phi(x)^T \phi(x))^{-1} \phi(x)^T y$

(It was $(X^T X)^{-1} X^T y$ before but since X got transform, it effect here as well)

## Choose Degree Polynomial M

- The higher the degree, the higher the variance → overfitting (try to fit perfectly to every point)
- The lower, the higher the bias → underfit
- Can be max $N-1$ $N$ = # of data
- Add Regularization like Ridge Regression (L2) to reduce $w$ to reduce overall Loss

$$E_{ridge}(w) = \frac{1}{2}\sum_{i=1}^{N}[w^T\phi(x_i) - y_i]^2 + \frac{\lambda}{2}\|w\|_2^2$$

$\lambda$ = regularization strength

larger $\lambda$ lead to smaller weight

## Bias / Variance Tradeoff

High bias - model too simplistic to capture the underlying pattern of data

High variance - model try to capture everything including noise which lead to high fluctuation
- lead to overfitting

too high $\lambda$ → high bias

model too high capacity / $\lambda$ too low → high variance

Popular technique to balance btw the 2: select large capacity model and keep variance in check by choosing appropriate $\lambda$

## Probabilistic Linear Regression

$$y_i = f_w(x_i) + \varepsilon_i \rightarrow \varepsilon_i \sim N(0, \beta^{-1}) \quad \text{where} \quad \beta = \frac{1}{\sigma^2}$$

if noise ($\varepsilon$) is gaussian, then $y_i$ follow gaussian as well

$$y_i \sim N(f_w(x_i), \beta^{-1})$$

Likelihood of a single sample $p(y_i | f_w(x_i), \beta) = N(y_i | f_w(x_i), \beta^{-1})$

For dataset D $p(y | X, w, \beta) = \prod_{i=1}^{N} p(y_i | f_w(x_i), \beta)$

Maximum Likelihood wrt. $w, \beta$ return parameter $w, \beta$ that will give highest probability to see $y$

$$w_{ML}, \beta_{ML} = \arg\max_{w,\beta} p(y | X, w, \beta)$$

Use Negative log likelihood (NLL) - $w_{ML}, \beta_{ML} = \arg\min_{w,\beta} -\ln p(y | X, w, \beta)$

let $E_{ML}(w, \beta)$ = Maximum likelihood Error function

$$E_{ML}(w, \beta) = -\ln p(y | X, w, \beta) = -\ln \left[\prod_{i=1}^{N} N(y_i | f_w(x_i), \beta^{-1})\right]$$

In the end $\Rightarrow$ $w_{ML} = \arg\min_w E_{LS}(w) = (\Phi(x)^T \Phi(x))^{-1} \Phi(x)^T y$

Which mean Maximizing the likelihood estimate is equivalent to minimizing the Least Square Error Func.

$\beta_{ML} = \arg\min_\beta E_{LS}(w_{ML}, \beta)$ (now a constant)

$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{i=1}^{N} \{ w_{ML}^T \phi(x_i) - y_i \}^2 =$ Mean Square Error

↳ Use MLE can easily lead to overfit → use MAP & Posterior Distribution instead

## find w, β on Posterior Distribution, MAP, Fully Bayesian

$$p(w|X, y, \beta, \alpha) = \frac{p(y|X, w, \beta) \cdot p(w|\alpha)}{p(y|X, \beta, \alpha)}$$

$\alpha$ = precision of a distribution

if $\alpha \to 0$ $w_{MAP} \to w_{ML}$

Choose prior $p(w|\alpha)$? $\Rightarrow$ Gaussian b/c likelihood is also gaussian

$$w_{MAP} = \max_w \frac{p(y|X, w, \beta) \cdot p(w|\alpha)}{p(y|X, \beta, \alpha)} = \max_w \ln p(y|X, w, \beta) + \ln p(w|\alpha) - \underbrace{\ln p(y|X, \beta, \alpha)}_{\text{constant = ignore}} = \arg\min_w -\ln p(y|X, w, \beta) - \ln p(w|\alpha)$$

let $E_{MAP} = -\ln p(y|X, w, \beta) - \ln p(w|\alpha) + \text{const}$

In the end $E_{MAP} \propto E_{ridge}(w) + \text{const} = \frac{1}{2} \sum_{i=1}^{N} (w^T \phi(x_i) - y_i)^2 + \frac{\lambda}{2} \|w\|_2^2 + \text{const}$ where $\lambda = \frac{\alpha}{\beta}$

$\beta = \frac{\alpha}{\lambda}$

## Compute full posterior distribution

$p(w|D) \propto p(y|X, w, \beta)\, p(w|\alpha)$

Since both likelihood & prior is Gaussian, the posterior is gaussian as well

$p(w|D) \sim N(w|\mu, \Sigma)$ w/ $w_{MAP} = \mu$ (mode = mean)

if $\alpha \to 0$ $w_{MAP} \to w_{ML}$

## Posterior Predictive Distribution - predict new data $(x_{new}, y_{new})$

Plugging in estimated parameter $w_{ML}, \beta_{ML}$ / $w_{MAP}, \beta_{MAP}$, we get Predictive Distribution to make new prediction $\hat{y}_{new}$ for $x_{new}$

Recall that

$y_i \sim N(f_w(x), \beta^{-1})$ so $p(y_i|x_i, w, \beta) = N(y_i | w^T \phi(x_i), \beta^{-1})$

For $y_{new}, x_{new}$

For ML: $p(y_{new} \mid x_{new}, w_{ML}, \beta_{ML}) = N(\hat{y}_{new} \mid w_{ML}^T \phi(x_{new}), \beta_{ML}^{-1})$ ($w$ is constant, variance is fixed)

For MAP: $p(\hat{y}_{new} \mid x_{new}, w_{MAP}, \beta) = N(\hat{y}_{new} \mid w_{MAP}^T \phi(x_{new}), \beta^{-1})$ $\beta = \frac{\alpha}{\lambda}$

Alternatively, we can use the full posterior dist. $p(w \mid D)$

This allow us to compute Posterior Predictive Distribution ($w_{ML}$, $w_{MAP}$)

$w$ is now random variable and not fixed like in Predictive Distribution — it has its own distribution

$$p(\hat{y}_{new} \mid x_{new}, D) = \int p(\hat{y}_{new}, w \mid x_{new}, D)\, dw$$

$$= \int p(\hat{y}_{new} \mid x_{new}, w) \cdot p(w \mid D)\, dw$$

$D$ is gone b/c already have $w$

$x_{new}$ is gone b/c it's independent of $w$

Gaussian · Gaussian

$$= N(\hat{y}_{new} \mid \mu^T \phi(x_{new}), \beta^{-1} + \phi(x_{new})^T \Sigma \phi(x_{new}))$$

$\mu = w_{MAP}$

variance change with new data (uncertainty)

Advantage - More accurate estimate about uncertainty in the prediction (e.g. data-dependent uncertainty estimate)

# Lecture 5 : Linear Classification

**Classification** - Output $y$ belong to one of $C$ classes

**Regression** - Output $y$ is continuous

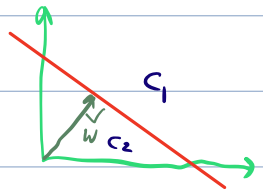
## Zero-one loss (loss function)

- denote number of misclassified samples

$$\ell_{0,1}(y, \hat{y}) = \sum_{i=1}^{N} I(\hat{y}_i \neq y_i)$$ → sum up all the time that predicted wrong

## Hyperplane as a decision boundary

For 2 Class - Separate points from 2 classes by a hyperplane

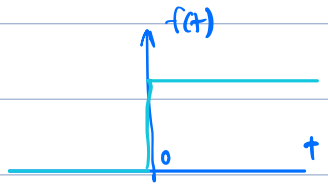



$$\text{hyperplane} = w^T x + w_0 = \begin{cases} 0 & \text{if } x \text{ is on the plane} \\ >0 & \text{if } x \text{ is on normal side of plane} \\ <0 & \text{else} \end{cases}$$

$w$ = normal vector
$w_0$ = offset

## Perceptron

Decision Rule : $\hat{y} = f(w^T x + w_0)$ where $f(t) = \begin{cases} 1 & \text{if } t>0 \\ 0 & \text{ow.} \end{cases}$

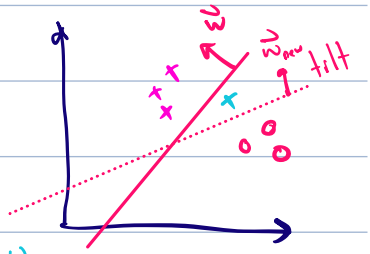



### Learning Rule for Perceptron

- Tilt the hyperplane to include the x that was misclassified

$$w \leftarrow \begin{cases} w + x_i & \text{if } y_i = 1 \\ w - x_i & \text{if } y_i = 0 \end{cases} \qquad w_0 = \begin{cases} w_0 + 1 & \text{if } y_i = 1 \\ w_0 - 1 & \text{if } y_i = 0 \end{cases}$$

↳ I think $y_i = 0$ is for class down hyperplane
so if it misclassified (that mean $\hat{y} = 1$)
then the hyperplane has to be tilted down to include the point




- Only work if data is linearly seperable
- Can scale up to multiple class by learning many hyperplane
- One-versus-rest classifier
  - for K class, use K-1 Classifier. Each classifier solves a 2 class problem ($C_K$ or not $C_K$)
  - Each point is classified to class C if the distance from hyperplane is maximal (deep inside class c) (we don't know if it's intersect)
- One-versus-one classifier
  - for K class, introduce $K(K-1)/2 = \binom{K}{2}$ binary classifier (one for every pair of classes)
  - Each classifier classified point to either $C_i$ or $C_j$
  - Each point is classified to class C accordig to majority vote amongst all classifier (we don't know if the point get same ... of vote

or get no vote)

- Multiclass Discriminant - train each classifier (C linear functions)

$f_c(x) = w_c^T x + w_{0c}$ (hyperplane) w/ decision rule $\hat{y} = \arg\max_{c \in C} f_c(x)$ predict class for point x w/ the class that has the biggest $f_c(x)$

Divide the region, not based on side of hyperplane, but based on the distance from the boundary. (The decision boundary is where $y_k(x) = y_j(x)$ $\forall j \neq k$

If point lies on the decision boundary, it can either be k or j. It's undecided!)

## Basis functions

Apply a nonlinear transformation $\phi : \mathbb{R}^D \to \mathbb{R}^M$ (can learn the transformation w/ NN)

linear transformation can stretch, rotate or shift but cannot seperate data and make it linearly seperable.

## Limitation of Hard-decision based classifier

1) No measure of uncertainty (probability for each class)
2) Can't handle noisy data
3) Poor generalisation (try too hard to classify all point correctly)
4) Difficult to optimise

## Probabilistic models for Classification

$$p(y=c|x) = \frac{p(x|y=c)\,p(y=c)}{p(x)}$$

2 Type of model - Generative → Model joint distribution (get $p(y=c|x)$ by finding $p(y=c)$ & $p(x|y=c)$ first)

- Discriminative → Directly model the distribution $p(y=c|x)$

Given $p(y=c|x)$ we can make prediction $\hat{y}$ $\hat{y} = \max_{c \in C} p(y=c|x)$

## Generative models for linear Classification

$$p(y=c|x) \propto p(x|y=c)\,p(y=c)$$

$p(x|y=c)$ = class conditional - probability of generating point x given that it belong to class c

$p(y=c)$ - class prior - prior prob. of a point belonging to class c

## Applying generative model

- Choose parametric model for class conditional and class prior (Bernoulli if class = 2 or

Categorical distribution for class $(\alpha)$

$p(x|y=c,\psi)$ and $p(y=c|\theta)$

estimate model's parameters $\{\psi,\theta\}$ from data $D$ (using eg. MLE) - obtain estimate $\{\hat{\psi},\hat{\theta}\}$ → this step is called learning

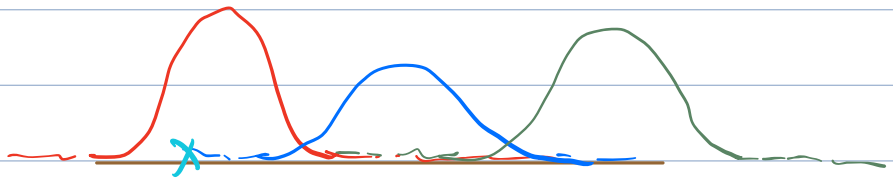
Once fitted, we can perform inference - classify label for $x$ using Bayes rule

$$p(y=c|x,\hat{\psi},\hat{\theta}) \propto p(x|y=c,\hat{\psi})\,p(y=c|\hat{\theta})$$

Can also generate new data

→ sample a class label $y_{new} \sim p(y|\hat{\theta})$ (predicting it's a cat)

→ sample new data $x_{new} \sim p(x|y=y_{new},\hat{\psi})$ (predicting value of pixel $x$ knowing it's a cat)



let $\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.3 \\ 0.4 \end{bmatrix} = p(y|\hat{\theta})$ prob of each class

$x_{new}$ →

- $p(y=\text{red}|x,\hat{\psi},\hat{\theta}) = p(x|y=\text{red},\hat{\psi},\hat{\theta})\,p(y=\text{red}) =$ big prob $\times$ 0.3 = still big
- $p(y=\text{blue}|x,\hat{\psi},\hat{\theta})$ = small $\times$ 0.3 = small prob
- $p(y=\text{green}|x,\hat{\psi},\hat{\theta})$ = very small $\times$ 0.4 = small prob
- after finding prob of all class(y) for point $x$, choose the class w/ highest prob

## How to choose class prior

$p(y=c)$ $y \sim \text{Categorical}(\theta)$ $\theta \in \mathbb{R}^C$ is prob of each class

so $p(y=c)=\theta_c$ and $0 \le \theta_c \le 1$ and $\sum_{c=1}^{C} \theta_c = 1$

the Maximum Likelihood estimate for $\theta$

$$\theta_c^{MLE} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(y=c)$$

e.g. if b=3 r=5 g=2

$\theta_b = \frac{3}{10}$ $\theta_r = \frac{5}{10}$ $\theta_g = \frac{2}{10}$

## Choose class conditional

dist.

→ Use multivariate normal for each class $p(x|y=c) = N(x|\mu,\Sigma)$ ↗ has $D \times D$ param

### Linear Discriminant Analysis (assume shared covariance $\Sigma$)

let class conditional = $p(x|y=c) = N(x|\mu_c,\Sigma)$ $\mu_c$ = mean for each class $\Sigma$ = same covariance matrix for all class

For $C=2$ - Hyperplane = boundary (normally not the case)

$p(y=1|x) = \sigma(w^T x + w_0) \xrightarrow{so} p(y=0|x) = 1-\sigma(w^T x + w_0)$ or $y \sim \text{Bernoulli}(\sigma(w^T x + w_0))$

For $C > 2$ – Use softmax function

$$p(y=c|x) = \frac{p(x|y=c)p(y=c)}{\sum_{i=1}^{C} p(x|y=i)\,p(y=i)} = \frac{\exp(w_c^T x + w_{c0})}{\sum_{i=1}^{C} \exp(w_i^T x + w_{i0})}$$

**LDA** will learn linear decision boundary (it use hyperplane which create decision boundary)

**Naive bayes** (Assume d features of sample are independent given the class)

let class conditional = $p(x|y=c) = N(x|\mu_c, \Sigma_c) \rightarrow \mu$ and $\Sigma$ for each class c

$$p(x_1, x_2, \dots, x_d | y=c) = \prod_{i=1}^{d} p(x_i|y=c)$$

Since $x_1, \dots x_d$ is indep. given class c, the covariance $\Sigma_c$ is diagonal matrix → only have D param instead of DxD (like in LDA)

LDA param = $C \cdot D + D \cdot D$ NB param = $C(D+D)$

$p(y=c|x) = \sigma(x^T W_2 x + w_1^T x + w_0) \rightarrow$ quadratic func! → quadratic decision boundary

* If $\Sigma_c$ is same for all class → same solution as LDA

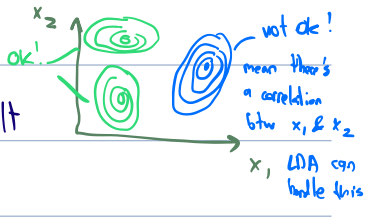
## Pros of NB

- Allow diff covariance structure
- Independent feature allows to easily handle diff data types/mix data types ($x_1$ = numerical, $x_2$ = categorical...)

use Gaussian distribution; use categorical distribution

## Cons of NB

- blc features are indep. It ignore the correlation btw the feature → so boundary cannot be tilt




## Discriminative Model for Linear Classification (model $p(y|x)$)

- In Generative model $w, w_0$ are precompute before plugging into $\sigma$ to find $p(y=c|x) \propto p(x|y=c)\,p(y=c)$

  $w, w_0$ also depend on parameters of class conditionals $\mu_0, \mu_1, \Sigma$

In Discriminative model w and $w_0$ is random variable that can be optimize

## Logistic Regression

let $w^T x = w^T x + w_0$

$y|x \sim \text{Bernoulli}(\sigma(w^T x + w_0))$

Learning logistic regression → find good parameter w that "explain" the training set D

(Assume sample are i.i.d) $p(y|w, X) = \prod_{i=1}^{N} p(y_i|x_i, w) \rightarrow p(y_i|x_i, w) = \text{Bernoulli}(\dots)$

$$= \prod_{i=1}^{N} \sigma(w^T x)^{y_i} (1-\sigma(w^T x))^{1-y_i}$$

Optimize w $\xrightarrow{\text{learning}}$ Find MLE for w → $w_{MLE} = \arg\max_w p(y|w|x)$

## Negative Log Likelihood (NLL) – Error Function

$$E(w) = -\log p(y|w,X) = -\sum_{i=1}^{N} (y_i \log \sigma(w^T x) + (1-y_i)\log(1-\sigma(w^T x)))$$

→ this loss function is Binary Cross Entropy

↳ no $\frac{1}{n}$ b/c use total loss in ML lecture

and Finding MLE for w is equivalent to minimizing E(w)

$$w^* = \arg\min_w E(w)$$

But b/c MLE can lead to overfitting, we can use MAP estimation which will include regularization

$$E(w) = -\log p(y|w,X) + \lambda \|w\|_2^2 \text{ (for L-2 norm / ridge)}$$

## Multiclass Logistic Regression

Use Softmax $p(y=c|x) = \frac{\exp(w^T x)}{\sum_{i=1}^{C} \exp(w_i^T x)}$

And optimal w → $w^* = \arg\max_w p(Y|w,X)$ → use NLL → $-\log p(Y|w,X) = E(w)$

$$E(w) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \frac{\exp(w_c^T x)}{\sum_{j=1}^{C} \exp(w_j^T x)}$$ where $y_{ic}$ = one-hot encoding $= \begin{cases} 1 & \text{if sample } i \text{ belong to class } c \ (i=c) \\ 0 & \text{ow.} \end{cases}$

This is called Cross Entropy

no $\frac{1}{n}$ b/c use total loss in ML lecture

## Generative VS Discriminative Model

- Discriminative model achieve better perf in general for classification task (less assumption required)
- Generative work well but frgile when assumption are violate (same covariance / independent feature)
- Generative models provide benefits of better handling missing data, detecting outliers, generating new data
  - handling missing data: create new data to fill in the blank
  - detecting outliers: if data has low prob in all class, it's most likely an outliers

---

## Note to Recap

Linear Classification for C=2 → use Bernoulli to predict output (label)

→ Find optimal $w^*$ → $\arg\max_w p(y|w,x)$ $y|x \sim$ Bernoulli

→ Use NLL → get loss function $E(w) = BCE = -\sum y \log y + (1-y)\log(1-y)$

→ $w^* = \arg\max_w p(y|w,x) = \min_w E(w)$ (no closed form)

Linear Regression → use Gaussian to predict output (continous)

→ Find optimal $w^*$ → $\arg\max_{w,\beta} p(y|w,x,\beta) = \arg\max_{w,\beta} N(f_w(x), \beta^{-1})$

→ Use NLL → get loss function $E_{LS}(w)$ = Least square $= \frac{1}{2}(f_w(x)-y)^2$

→ $w^* = \arg\max_{w,\beta} p(y|w,x,\beta) = \min_w E_{LS}(w) = (X^T X)^{-1} X^T y$

# Lecture 6: Optimization (Convex!)

Denote $f(\theta)$ = objective function

Find minimum of objective function → convex

## Convexity

$X$ is convex set iff $\forall x,y \in X \quad \lambda x + (1-\lambda)y \in X$

$f(x)$ is convex function iff $\forall x,y \in X$ $\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda)y)$

For convex func – each local minimum is a global minimum (could be more than 1 global min)

– if $f_0$ is convex, $\nabla f_0(\theta^*) = 0$ then $\theta$ is a global minimum

– Rate of rise on function $= f(y) - f(x) \geq \frac{f((1-\lambda)x + \lambda y) - f(x)}{\lambda}$ or $f(y) - f(x) \geq (y-x)^T \nabla f(x)$ when $\lambda \to 0$

∴ if $f: X \to \mathbb{R}$ is differentiable and $X$ is convex, then $f$ is convex iff $\forall x,y \in X$ $f(y) \geq f(x) + (y-x)^T \nabla f(x)$

## Verify Convexity

1) $\lambda f(x) + (1-\lambda) f(y) \geq f(\lambda x + (1-\lambda) y)$ for all $x,y \in X$

2) Hessian matrix is PSD on the convex set ($x^T \nabla^2 f(x,y,\dots) x$) then func is convex on the convex set

(for 1 variable it's convex on a domain, if $\nabla^2 f(x) \geq 0$ for all $x$ in the domain)

if $\nabla^2 f(x)$ or $x^T H(x,y\dots) x \leq 0$, then func is concave

Linear func is both convex and concave

3) Obtain by operation that preserve convexity

Let $f_1$ and $f_2$ be convex func and let $g$ be concave

– $h(x) = f_1(x) + f_2(x)$ is convex

– $h(x) = \max\{f_1(x), f_2(x)\}$ is convex

– $h(x) = c \cdot f_1(x)$ is convex if $c \geq 0$

– $h(x) = c \cdot g(x)$ is convex if $c \leq 0$

– $h(x) = f_1(Ax+b)$ is convex ($A$ matrix, $b$ vector)

– $h(x) = m(f_1(x))$ is convex if $m: \mathbb{R} \to \mathbb{R}$ is convex and non-decreasing

Proof

let $z(x) = h(f(x)) \to z'(x) = h'(f(x)) \cdot f'(x) \to z''(x) = h''(f(x)) \cdot f'(x) \cdot f'(x) + f''(x) \cdot h'(f(x))$

and $f$ is convex

$= h'(f(x))\, f'(x) + f'(x)\, h'(f(x)) \geq 0$

must be $\geq 0$ (so $h$ must be convex) | $\geq 0$ b/c $f(x)$ is convex (non-decreasing) | $h'f(x)$ must be $\geq 0$

# Convex Set

1. Prove definition $\lambda x + (1-\lambda) y$
2. if A and B are convex set, then $A \cap B$ is convex set

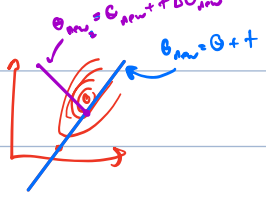
Unconstrained Minimization problem → e.g. Ordinary Least Square Regression → $\theta^* = f'(\theta) = 0$ ($Y = \beta_0 + \beta_1 X + \varepsilon$ (line))

## First Order Optimization (exploit gradient)

### - Gradient Descent

Gradient - Point at steepest ascent direction

GD w/ Line Search - get descent direction, then $\theta_{new} = \theta + t\Delta\theta$ (move point until optimum)




- Turn multidimension prob into one-dimension prob (do one dim at a time

Step 1) $\Delta\theta = -\nabla f(\theta)$ 2) Line Search: $t^* = \min_{t>0} f(\theta + t\Delta\theta)$ 3) Update: $\theta_{new} = \theta + t^*\Delta\theta$

$\nabla f(\theta)$ doesn't point to global min, it point to local min.

You can run GD Line Search in parallel (partial gradient) and aggregate them.

But line search is expensive, for each tested step size → scan through all datapoints

### - Learning Rate

Instead of using line search to find optimal $t$ → simply pick a learning rate $\gamma$

$\theta_{t+1} = \theta_t - \gamma \nabla f(\theta_t)$ $\gamma$ → too small → slow convergence, end up in local min or saddle points | $\gamma$ → too large → oscillate, never converge

Solution - Learning rate schedule → Let $\gamma$ be decreasing function of iteration $t$

Momentum - $\gamma$ increase (accelerate) as long as gradients keep pointing at the same direction

$\theta_{t+1} \leftarrow \theta_t - m_t$ $m_t \leftarrow \gamma \nabla f(\theta_t) + \gamma m_{t-1}$ $\gamma$ = momentum factor

Adagrad - Diff learning rate per parameter

Adam - $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(\theta_t)$ Bias Correction: $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ (similar to momentum)

- $v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla f(\theta_t)^2$ $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ (gradient fluctuate alot → if gradient is large, it will decrease the step if → small, → increase)
- $\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$

## Second Order – Newton

$\nabla^2 f(\theta) \succeq 0$ = Hessian is PSD

Taylor Expansion of $f$ at point $\theta_t$ (step $\delta$) $f(\theta_t + \delta) = f(\theta_t) + \delta^T \nabla f(\theta_t) + \frac{1}{2}\delta^T \nabla^2 f(\theta_t)\delta + O(\delta^3) := g(\delta)$

Minimize Approx. leads to $\nabla g(\delta) = \nabla f(\theta) + \nabla^2 f(\theta)\delta = 0 \rightarrow \delta = (\nabla^2 f(\theta))^{-1} \nabla f(\theta)$

And the update step $\theta_{t+1} = \theta_t - [\nabla^2 f(\theta_t)]^{-1} \nabla f(\theta_t)$ Repeat until convergence

\+ Fast convergence (require less iteration) – Require compute inverse Hessian ($O(d^3)$) ⇒ Only use for low dim problem

## SGD

(Exact) expectation can be approx. by smaller sample ($S \in \{1,\dots,n\}$)

Instead of computing the loss function for all instances, only compute loss func to the subset of data.

$$\mathbb{E}_{i\in\{1,\dots,n\}}[L_i(\theta)] \approx \frac{1}{|S|}\sum_{j\in S} L_j(\theta) \rightarrow \frac{1}{n}\sum_{i=1}^{n} L_i(\theta) \approx \frac{1}{|S|}\sum_{j\in S} L_j(\theta) \rightarrow \sum_{i=1}^{n} L_i(\theta) \approx \frac{n}{|S|}\sum_{j\in S} L_j(\theta)$$

original expectation loss; batch mean loss; times n to upscale to original loss; n = # of whole data

The gradient computation can also be done using only subset of data → noisy unbiased estimate

1. Randomly pick small subset S of the data (mini batch)
2. Compute gradient based on mini batch
3. Update weight $\theta_{t+1} \leftarrow \theta_t - \gamma\left(\frac{n}{|S|}\sum_{j\in S}\nabla L_j(\theta_t)\right)$ (Update happen for every mini batch/iteration)
4. Pick new subset and repeat 2–4

Larger mini batch lead to a more stable gradient (smaller variance in the estimated gradient)

SGD converge almost surely to global min when objective func is convex (and local min for non-convex)

Converge need more iteration to converge than GD but b/c it use mini batch, each iteration is faster

## Distributed Learning

- Distribute computation across multiple machine

Data Parallelism (CNN) vs Model Parallelism (SGD)

## Lecture 7/8: Deep Learning

Basis function $\phi(x)$ – Apply (non-linear) transformation $\phi$ that map samples to a space where they are linearly separable

Multi-layer Perceptron (MLP) = Fully Connected NN Feed Forward NN = NN w/ 1 hidden layer

Most activation func apply element-wise. Exception is softmax (use all input for normalize)

Use activation func to allow model to solve non linear problem.

## Universal Approximation Theorem

MLP w/ linear output layer and 1 hidden layer can approx. any continuous function defined over a closed and bounded subset of $\mathbb{R}^D$ with constraint of activation func and given # of hidden units are large enough

In short: there exist MLP that can approx. any func well

## Loss function

Diff task require changing

1. Activation function in the final layer
2. Loss function

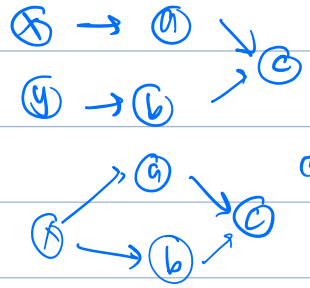
| Prediction target | p(y\|x) | Final layer Acti. func. | Loss Func |
|---|---|---|---|
| Binary (C=2) | Bernoulli | Sigmoid | BCE |
| Discrete (C>2) | Categorical | Softmax | CE |
| Continuous | Gaussian | Identity (basically don't use any) | MSE (Mean Square Error) $E(w) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$ |

Other loss func

- Mean Absolute Error - useful if we have outliers
- Huber Loss - MSE good at close dist, MAE good at far away point, this func combine both benefit

## Backpropagation

1. Write func as composition of modules
2. Work out local derivative of each module symbolically
3. Do forward pass for given input x → cache the result for each func. to be use in backprop
4. Compute local derivative for x
5. Obtain global derivative by multiplying local deriv.




Compute derivs. along each paths

$c(a,b) = a(x) + b(y)$

$\frac{\partial c}{\partial x} = \frac{\partial c}{\partial a}\cdot\frac{\partial a}{\partial x} + \frac{\partial c}{\partial b}\cdot\frac{\partial b}{\partial x}$ (∂b/∂x = 0)

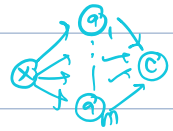
$\frac{\partial c}{\partial y} = \frac{\partial c}{\partial a}\cdot\frac{\partial x}{\partial y} + \frac{\partial c}{\partial b}\cdot\frac{\partial b}{\partial y}$ (∂x/∂y = 0)




$c(a,b) = a(x) + b(x)$

$\frac{\partial c}{\partial x} = \frac{\partial c}{\partial a}\cdot\frac{\partial a}{\partial x} + \frac{\partial c}{\partial b}\cdot\frac{\partial b}{\partial x}$

Multivariate Chain Rule




$\frac{\partial c}{\partial x} = \sum_{i=1}^{m}\frac{\partial c}{\partial a_i}\cdot\frac{\partial a_i}{\partial x}$

## Jacobian and gradient

Consider func $f: \mathbb{R}^n \to \mathbb{R}^m$ (take n variable, output m var) ← or func.

let $a = f(x)$ ← $f(x_1, x_2, \dots x_n)$

$$J = \frac{\partial a}{\partial x} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial a_m}{\partial x_1} & \cdots & \frac{\partial a_m}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

e.g. loss func.

let $g : \mathbb{R}^m \to \mathbb{R}$ and let $c = g(a)$

↳ $g(a_1, \dots a_m)$

**the gradient $\nabla_a c$ is the transpose of the Jacobian $\frac{\partial c}{\partial a} \in \mathbb{R}^{1 \times m}$**

$c = g(a_1, \dots a_m)$ $\quad \frac{\partial c}{\partial a} = \left(\frac{\partial c}{\partial a_1}, \dots, \frac{\partial c}{\partial a_m}\right)$

$$\nabla_a c = \left(\frac{\partial c}{\partial a}\right)^T = \begin{pmatrix} \frac{\partial c}{\partial a_1} \\ \vdots \\ \frac{\partial c}{\partial a_m} \end{pmatrix} = \left(\frac{\partial c}{\partial a_1}, \dots \frac{\partial c}{\partial a_m}\right)^T$$

$\nabla_a c$ has same shape as $a \in \mathbb{R}^{m \times 1}$

and $\frac{\partial c}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial c}{\partial a_i} \cdot \frac{\partial a_i}{\partial x_j}$ can be write in matrix form $\frac{\partial c}{\partial x} = \frac{\partial c}{\partial a} \cdot \frac{\partial a}{\partial x}$ where $\left[\frac{\partial a}{\partial x}\right]_{ij} = \frac{\partial a_i}{\partial x_j}$

Also $\nabla_x c = \left(\frac{\partial c}{\partial a} \cdot \frac{\partial a}{\partial x}\right)^T = \left(\frac{\partial a}{\partial x}\right)^T \left(\frac{\partial c}{\partial a}\right)^T = \left(\frac{\partial a}{\partial x}\right)^T \nabla_a c$

## Matrix Calculus

| | | Input: Scalar | Vector | Matrix |
|---|---|---|---|---|
| Output | Scalar | Scalar | Vector | Matrix |
| | Vector | | Matrix | 3-way tensor |
| | matrix | | | 4-way tensor |

$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial w}$ materialise/computing $\frac{\partial a}{\partial w}$ independently is expensive bc it's a 3 way tensor

However, looking at the $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial w}$, you can see that $\frac{\partial E}{\partial w_{kl}} = \frac{\partial E}{\partial a_k} x_l$

$\frac{\partial E}{\partial w} = \left(\frac{\partial E}{\partial a}\right)^T x^T$ Matrix operations are much more efficient thanks to vectorization (GPU)

## Convolution – Do feature extraction

Filter have $L \times H \times C_{in} \times C_{out}$ $\quad C_{out}$ = # of filter $\quad C_{in}$ = # of input's channel

Weight sharing – Weight of the kernel (filter) are shared for every patch

Padding – Valid – Do not use Padding – reduce size in output $P = 0$

SAME – Adding padding ($P = K/2$) to preserve size $\quad K$ = size of filter

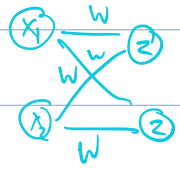
FULL – Increase output size ($P = K-1$)

Stride – distance btw positions the kernel is applied. → $S > 1$ are a way to downsample the signal

the output size is $P_{l+1} = \frac{P_l + 2P - K}{S} + 1$

Pooling - Doing aggregation & downsample at a same time → Max pooling, mean pooling, Lp norm

## Weight Initialization




$z_1 = x_1 w + x_2 w$ $\quad \frac{\partial z_1}{\partial w} = x_1 + x_2$

$z_2 = x_1 w + z_2 w$ $\quad \frac{\partial z_2}{\partial w} = x_1 + x_2$

Issue of naive weight init

1. Weight symmetry - If 2 hidden node have exactly the same bias and (initial weight w/ same value) incoming & outgoing weight, they will always get exactly the same gradient (update will lead to the same result)
   - So they can never learn to extract diff feature
   - Break symmetry by initializing the weights to have small random value
2. Weight Scale
   - If a hidden unit has a big # of input signal, small change on many of its incoming weight can cause learning to overshoot.
   - If large # of output signal, can cause the same during backward pass
   - Wrong Scale can lead to vanishing/exploding gradient

Vanishing & Exploding gradient - Deep network can suffer from this problem
- Vanish from very high/low input and use saturating activation (sigmoid / tanh)

↳ Solution → Change architecture (batch norm, ReLU), Gradient clipping (limit gradient)

## Xavier Initialization

Preserve the signal's properties by using weight matrices w/ $\mu = 0$ and

$$Var(W) = \frac{2}{\#\text{ of incoming} + \#\text{ of outgoing signal}}$$

Initial the weight for every FC layer

## Regularization

$\min_W L + \|W\|_F$

- Reduce Overfitting → $L_2$ norm shrink parameter equally, $L_1$ norm to promote sparsity

Combine w/ other method like Data augmentation, injecting noise, dropout to reduce overfitting

Dropout - randomly drop some % of neuron (make input = 0) random new set for each iteration (batch)

## Batch Normalisation

- Stabilize the distribution of each layer's activation → more stable gradient → training is more stable
- done after each layer → allow deeper network

$\hat{x} = \frac{x - \mu_B(x)}{\sigma_B(x) + \varepsilon}$ calculate $\mu$ and $\sigma$ for each minibatch to normalise the batch training data

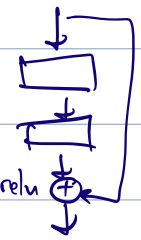
Introduce $\gamma$ and $\beta$ hyperparameter allow model to learn to undo the BN

$y = \gamma \hat{x} + \beta$

## Skip Connections

Highway Network → gating mechanism

Residual Connections → help w/ vanishing gradient, ignore the layer




## Lecture 9: SVM and Kernels

Linear Classifier assign class of x

$h(x) = \text{sign}(w^T x + b)$ w/ $\text{sign}(z) = \begin{cases} -1 & \text{if } z<0 \\ 0 & \text{if } z=0 \\ 1 & \text{if } z>0 \end{cases}$

Idea: Find the hyperplane that seperate data the best → have largest margin possible so that new data is more likely to fall on the correct side of boundary

So we try to maximize size of margin $m = \frac{2}{\|w\|_2} = \frac{2}{\sqrt{w^T w}}$ w/ constraint $y_i(w^T x_i + b) \geq 1$ for all i

Maximize $\frac{2}{\|w\|}$ = minimizing $\frac{1}{2} w^T w$ → Constrained Convex Optimization Problem → Lagrange

Lagrange dual function $g(\alpha) = \min_\theta L(\theta, \alpha) = \min\left(f_0(\theta) + \sum_i \alpha_i f_i(\theta)\right)$ give lower bound on the optimal value $f_0(\theta^*)$

Lagrange dual problem find the $\alpha$ that gives best lower bound $\max_\alpha g(\alpha)$ $\alpha_i \geq 0$

Apply Lagrange to SVM – $y_i(w^T x_i + b) - 1 \geq 0 \rightarrow -(y_i(w^T x_i + b) - 1) \leq 0$

$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_i \alpha_i (y_i(w^T x_i + b) - 1)$

(*) $\nabla_w L = w - \sum_i \alpha_i y_i x_i = 0 \rightarrow w = \sum \alpha_i y_i x_i$ weights are linear comb of training sample

(**) $\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0$

Substitute (*) and (*)(*) back into $g(\alpha)$ and find Lagrange Dual problem to get $\alpha^*$

$w = \sum_i \alpha_i^* y_i x_i$ From the complementary slackness condition $\alpha_i^* f_i(\theta^*) = 0$ (Strong Duality $g(\alpha^*) = f_0(\theta^*)$)

if $\alpha_i^* \neq 0$, then $f_i(\theta^*) = 0 \rightarrow y_i(w^T x_i + b) - 1 = 0 \rightarrow b = y_i - w^T x_i$ (recover bias)

Furthermore, $y_i(w^T x_i + b) - 1 = 0$ is the point at the margin

if $\alpha_i^* = 0$, then point is outside of margin and $w = \sum \alpha_i y_i x_i = 0$ for those point

So only the point on the margin (point $x_i$ where $\alpha_i > 0$ = support vector) are count toward min. problem (will $w = \sum_s \alpha_i y_i x_i$ where $s$ = support vector point)

Only need to remember few training sample $x_i$ w/ $\alpha_i > 0$

## Soft Margin SVM

Hard Margin - doesn't work w/ data that is not linearly separable

Idea: Relax the constraint but punish the relax of constraint by introducing slack variable $\xi_i \geq 0$

for every training sample. $\xi_i$ gives the distance of how far the margin is violated in unit of $\|w\|$

$\xi_i = 0$ → outside of margin, correct side. $0 < \xi_i < 1$ correct side but inside margin gap $\xi_i > 1$ - wrong side (also include margin gap but wrong side)

So the new constrain func is $y_i(w^T x_i + b) \geq 1 - \xi_i$ for all $i$ and $\xi_i \geq 0$

And the new cost func is $f_0(w, b, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$ where $C$ = how heavy a violation is punish. if $C \to \infty$, then we have hard margin SVM

$$L(w, b, \xi, \alpha, \mu) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i - \sum_i \alpha_i \left( y_i (w^T x_i + b) - 1 + \xi_i \right) - \sum_i \mu_i \xi_i$$

($\alpha$: Lagrange variable)

$$\nabla_w L = w - \sum \alpha_i y_i x_i = 0 \qquad \frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \qquad \frac{\partial L}{\partial \xi} = C - \alpha_i - \mu_i = 0 \to \alpha_i = C - \mu_i$$

since $\alpha_i \geq 0$ and $\mu_i \geq 0$ then $0 \leq \alpha_i \leq C$ (when $\mu_i = C$, $\alpha_i = C - C = 0$; when $\mu_i = 0$, $\alpha_i = C$)

Substitute into $g(\alpha)$ and find maximum $\alpha^*$ with Dual Problem and get same solution as Hard margin except $0 \leq \alpha_i \leq C$ so $\alpha_i$ cannot go to infinity.

Look at constrain func - $y_i(w^T x_i + b) \geq 1 - \xi_i$ ($\xi_i \geq 1 - y_i(w^T x_i + b)$) and $\xi_i \geq 0$ → we can rewrite into unconstrain opt. problem.

if point is on the wrong side/ on the margin $y_i(w^T x_i + b) = 1 - \xi_i$ then $\xi_i = 1 - y_i(w^T x_i + b)$

but b/c for wrong side $y_i(w^T x_i + b) < 1$ $\therefore \xi_i > 0 = 1 - y_i(w^T x_i + b)$

if point is on the correct side and outside margin $y_i(w^T x_i + b) = 1 - \xi_i$ then $\xi_i = 1 - y_i(w^T x_i + b)$

b/c for correct side $y_i(w^T x_i + b) \geq 1$ so $\xi_i \leq 0$ but b/c $\xi_i \geq 0$ $\therefore \xi_i = 0$

$$\xi_i = \begin{cases} 1 - y_i(w^T x_i + b) & \text{if } y_i(w^T x_i + b) < 1 \\ 0 & \text{else} \end{cases}$$

Hinge loss $= \max(0, 1 - y_i(w^T x_i + b))$ → 0 = predict correct, 1 = misclassified

And we can rewrite objective func as an unconstrained optimization problem

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i + b)\}$$

We can optimize hinge loss directly using gradient based method

# Kernels

Kernel function $k: \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ $\quad K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

Kernel represent inner product in the feature space spanned by $\phi$, this make our model nonlinear

Why use kernel instead of basis function $\phi$

- Computing basis function can be too expensive, but directly evaluating kernel is often easier than computing $\phi(x)$
- Kernel allow us to compare similarity btw arbitary data (graph, non-numerical data)

## Kernel perserving operation

let $k_1: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $k_2: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be kernel

$K(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2)$

$K(x_1, x_2) = c \cdot k(x_1, x_2)$ if $c > 0$

$K(x_1, x_2) = k_1(x_1, x_2) \cdot k_2(x_1, x_2)$

$K(x_1, x_2) = k_3(\phi(x_1), \phi(x_2))$ where $k_3: \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ and $\phi: X \to \mathbb{R}^M$

$= \phi(x_1)^T \phi(x_2)$ → eg. Prove that 1 is kernel → $K(x_1, x_2) = k_3(\phi(x_1), \phi(x_2)) = \phi(x)^T \phi(x_2)$

let $\phi(z) = 1$ so $K(x_1, x_2) = (1)^T(1) = 1$

eg. $\exp\left(-\frac{x_1^T x_1}{2\sigma^2}\right)\exp\left(-\frac{x_2^T x_2}{2\sigma^2}\right)$ → let $\phi(z) = \exp\left(-\frac{z^T z}{2\sigma^2}\right)$

$K(x_1, x_2) = x_1^T A x_2$ w/ $A \in \mathbb{R}^{N \times N}$ and $A$ is symmetric and PSD $\quad K_3(\phi(x_1), \phi(x_2)) = \exp\left(-\frac{x_1^T x}{2\sigma^2}\right)^T \exp\left(-\frac{x_2^T x_2}{2\sigma^2}\right)$

scale so transpose doesn't matter

Kernel - Polynomial = $K(a, b) = (a^T b)^p$ or $(a^T b + 1)$

- Gaussian Kernel = $K(a, b) = \exp\left(-\frac{\|a-b\|^2}{2\sigma^2}\right)$

if 2 point are similar to each other, then K is large. If $\sigma$ is smaller, the decision boundary is more fine grain → tend to overfit (look at smaller neighborhood area)

## Classify new point w/ Kernelised SVM

Due to complimentary slackness ($g(\alpha^*) = f_0(\theta^*)$ - strong duality)

This mean that both constrain of Soft margin SVM must be 0

(*) $\alpha_i^* f_i(\theta^*) = 0$ and (*)(*) $\mu \xi_i = 0$ (note that $\alpha = C - \mu$)

Hard margin has (*) constrain but not (*)(*) so hard margin support vector = $x_i$ where $\alpha_i > 0$ = $x_i$ on margin

For Soft margin → (*) → $\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) = 0$ → $y_i(w^T x_i + b) - 1 + \xi_i = 0$ → $y_i(w^T x_i + b) = 1 - \xi_i$ = $y_i(w^T x_i + b) = 1$ = lies on margin if $\xi_i = 0$

(*)(*) → $(C - \alpha)\xi_i = 0$

∴ Support Vector for SM is when $\alpha > 0$ → contribute to weight (support vector)

If $\alpha < C$, then $\xi_i$ must be 0 and such point lies on the margin

so support vector is point where $0 < \alpha \leq C$

If $\alpha = C$, then $\xi_i > 0$ and point can lies inside the margin and/or wrong side

Include all x on and inside margin

Note $\xi_i = 0$ on margin / correct side $\xi_i > 0$ inside margin / wrong side

Support Vector where $0 < \alpha < C$ has $\xi = 0$, hence

$$y_i \left( \sum_{(j | x_j \in S)} \alpha_j y_j k(x_i, x_j) + b \right) = 1$$ and bias can be recover $$b = y_i - \sum_{(j | x_j \in S)} \alpha_j y_j k(x_i, x_j)$$

and new point x can be classified as $h(x) = \text{sign}\left( \sum_j \alpha_j y_j k(x_j, x_k) + b \right)$

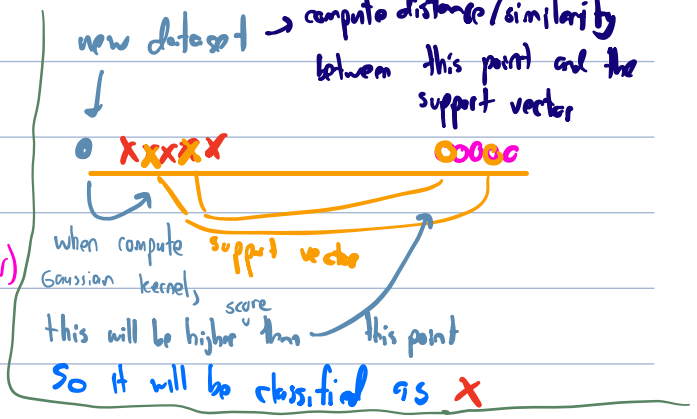
act like K-NN

## Multiple Class SVM

One-vs-rest : Train C SVM model for C classes,

for each SVM of class x, predict if data belong to class x or other (all the rest of the class). (Predict either class x or other)

the winner is then the class, where the distance from the hyperplane is maximal




One-vs-one : Train $\binom{C}{2}$ classifiers (all possible pairing) and evaluate all.

Winner is the class with the majority vote (votes are weighted according to the distance from the margin)

| $C_1$ | $C_2$ | Result |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 3 |
| 1 | 3 | 1 |

C=1=win

## Lecture 10-11 : Dimensional Reduction

**Unsupervised Learning** - modeling $p(x)$

generative transformation

- find the hidden (latent) structure in the data (latent distri. $p(z)$ and $p(x|z)$, latent z usually unknown, has to be estimate)
- eg. Clustering, Anomaly detection
- Can be view as Compression (higher to lower dim)

### Dimensionality Reduction

Idea : Try to reduce dimensionality while avoid info loss

### Feature Selection

Choose "good" dim using prior knowledge or appropriate heuristic (remove low variance dim)

Dim Reduction via Linear Transf. → Change of basis (orthogonal basis trans.) and discard low variance dim.

Transform w/ orthogonal basis → preserve length $F^T F = F F^T = I$

$X' = X \cdot F$ is the matrix containing all the transform point

Mean vector → $\bar{x}' = \bar{X} \cdot F \rightarrow \bar{X} \cdot F = \frac{1}{N} \sum_i x_i \cdot F = \frac{1}{N} \sum_i (x_i \cdot F) = \frac{1}{N} \sum_i x_i' = \bar{x}'$

$\Sigma_{X'} = F^T \cdot \Sigma_X \cdot F \rightarrow F^T \cdot \left(\Sigma \frac{1}{N}(X^TX - \underbrace{\bar{X}\bar{X}^T}_{0 \text{ cus zero mean}})\right) \cdot F = \frac{1}{N}\Sigma (XF)^T(XF) = \frac{1}{N}\Sigma X'^TX' = \Sigma_{X'}$

## PCA

Goal: Transform data s.t. $Cov(x_i, x_j) = 0$ for all $i \neq j$ (so data point are not linearly correlate anymore)

**Approach**

1. Center the data ($\tilde{X} = X - \bar{X}$ where $\bar{X}$ = mean of each column, $\tilde{X}$ = zero mean)
2. Compute $\Sigma_{\tilde{X}} = \begin{bmatrix} Var(x_1) & Cov(x_1,x_2) & \cdots & Cov(x_1,x_d) \\ Cov(x_2,x_1) & Var(x_2) & & \\ & & \ddots & \\ & & & Var(x_d) \end{bmatrix} = \frac{1}{N}\tilde{X}^T\tilde{X}$

$Var(X_j) = \frac{1}{N} X_j^T X_j - \bar{x}^2$ (mean $\tilde{x} = 0$)

$Cov(X_{j1}, X_{j2}) = \frac{1}{N} X_{j1}^T X_{j2} - \bar{x}_{j1}\bar{x}_{j2}$

3. Compute Eigenvector Decomposition of $\Sigma_{\tilde{X}}$ to transform the coordinate

$\Sigma_{\tilde{X}} = \Gamma \Lambda \Gamma^T$ where $\Gamma$ = eigenvector matrix, $\Lambda$ = diagonal matrix w/ eigenvalue $\lambda_i$

Transform data = $Y = \tilde{X} \cdot \Gamma$ and $\Sigma_Y = \Gamma^T \Sigma_{\tilde{X}} \Gamma = \Gamma^T(\Gamma \Lambda \Gamma^T)\Gamma = \Lambda$ is the new covariance matrix of new coordinate system

Variance of new system = $\Sigma_Y = \Lambda = \lambda_i$ and $Cov(Y_i, Y_j) = 0 \ \forall i \neq j$

Coordinate w/ low variance (low $\lambda_i$) can be ignored → only keep $\Gamma$ col corresponding to largest $K$-th $\lambda_K$

Pick $k$ s.t. 90% of var is explain → $\sum_{i=1}^{k} \lambda_i \geq 0.9 \sum_{i=1}^{d} \lambda_i$

### Alternative View of PCA

1. Maximize Variance 2. Minimize Residual 3. Projected data

**Low rank approx** – approx. original data A by a low rank matrix B

B has lower rank when approx. $A = \begin{bmatrix} 1.01 & 2.05 & 0.9 \\ -2.1 & -3.05 & 1.1 \\ 2.99 & 5.01 & 0.3 \end{bmatrix} \approx \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} = B$ rank(A) = 3 rank(B) = 2

So by approx., when we rewrite coordinate in new system, it require less coordinate to describe. (bc less dim)

Best = min the sum of reconstruction error $\min_{B, rank(B)=K} \|A - B\|_F^2 = \sum_{i=1}^{N}\sum_{j=1}^{D} (a_{ij} - b_{ij})^2$ (error btw original & new approx. matrix)

## SVD

$A = U \Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ $A \in \mathbb{R}^{n \times d}$ $U \in \mathbb{R}^{n \times r}$ $\Sigma \in \mathbb{R}^{r \times r}$ $V \in \mathbb{R}^{d \times r}$ $r = \min(n, d)$

Reading the $U, \Sigma, V$

$U$ = span(C(A)) → column of A → how significant each data row (eg. user) effect on concept (r col)

$V$ = span(C($A^T$)) → row of A → how significant each feature (eg. movie) effect on concept (r col)

$\Sigma$ = strength of each concept

$P = U\Sigma = AV$ give coordinate of points after being projected on the projection axis (new coord. system)

V act as a transformation matrix (like $\Gamma$ in PCA)

Approx. matrix by setting smallest $\sigma_{k+1} \dots \sigma_r$ to 0 and get best k approx.

→ choose k s.t. we can still explain var $\geq 90\%$ $\sum_{i=1}^{k} \sigma_i^2 \geq 0.9 \sum_{i=1}^{r} \sigma_i^2$ $\text{tr}((UX)^T(UX)) = \text{tr}(X^T U^T U X) = \text{tr}(X^T X) = \|X\|_F^2$

**Some Math Fact:** $\|X\|_F = \|X^T\|_F$ $\|X\|_F^2 = \text{trace}(X^T X)$ $\|UX\|_F^2 = \|VX\|_F^2 = \|X\|_F^2$ if U, V = orthogonal matrix

$\|A\|_F^2 = \text{tr}((U\Sigma V^T)(U\Sigma V^T)) = \text{tr}(U\Sigma^2 V^T) = \text{tr}(UU^T\Sigma^2) = \text{tr}(\Sigma^2) = \|\Sigma\|_F^2 = \sum_{i=1}^{r} \sigma_i^2$

if $\text{rank}(B) = k$ $A - B = \sum_{i=k+1}^{r} \sigma_i u_i v_i^T$ and $\|A - B\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$

In PCA, $\Sigma_X = X^T X = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^2 V^T = \Gamma \Lambda \Gamma^T$ if $V = \Gamma$ and $\Sigma^2 = \Lambda$ — Can use eigenvalue decomposition to calculate the singular decomposition

## Matrix Factorization

System like Recommendation System has alot of missing data

if we use SVD, we have to fill those missing value → eg. fill w/ 0 → might be misleading

SVD need to sum over all entry → What if we sum over existing entries

approx. $R \approx QP^T$ $\min_{P,Q} \sum_{(u,i)\in S} (r_{ui} - q_u p_i^T)^2$ $q_u$ = row of Q, $p$ = row of P (Q, P doesn't need to be orthogonal or unit length anymore)

$Q = U\Sigma$ $P = V$

## Alternating Optimization

- Pick initial $P^{(0)}$ and $Q^{(0)}$, $t = 0$ → missing entries are replace w/ 0 (or mean)
- Alternating keep one variable fix and optimizing for the other

$$P^{(t+1)} = \arg\min_P f(P, Q^{(t+1)}) \quad \text{(optimize P, fix Q)}$$

$$Q^{(t+1)} = \arg\min_Q f(P^{(t+1)}, Q)$$

$$f(P,Q) = \sum (r_{ui} - q_u p_i^T)^2$$

- Repeat until converge (minimize Sum of Square Error $f(P,Q)$)

$$p^{(t+1)} = \sum_i \arg\min_{p_i} \sum_{u \in S_{*,i}} (r_{ui} - q_u p_i^T)^2 = \sum_{(u,1)\in S} (r_{u1} - q_u p_1^T)^2 + \sum_{(u,2)\in S} (r_{u2} - q_u p_2^T)^2 + \dots = \text{is an OLS regression prob.} \min_w \sum_{i=1}^{n} (y_i - w^T x_i)$$

($S_{*,i}$: only user u who have entry at col i)

## Use SGD on Matrix Factorization instead

$$L = \sum_{(u,i)\in S} (r_{ui} - q_u p_i^T)^2$$

- Pick random u and i w/ rating $r_{ui}$ (batch = 1)
- Compute gradient w.r.t. the parameter $\frac{\partial L}{\partial q_u}$, $\frac{\partial L}{\partial p_i}$
- Update parameter $q_u \leftarrow q_u - \eta \frac{\partial L}{\partial q_u}$, $p_i \leftarrow p_i - \eta \frac{\partial L}{\partial p_i}$

## Predict Missing value

Now that we have matrix Q, P, we can do matrix mult. to find missing rating in matrix R

**Cold start problem** → new user has no data → Ask user to choose movie they like to create data

## Regularization

top k rank of matrix

Overfit – too many parameter, not enough datapoint (too many missing data)

$$\min \sum_{(u,i)\in S} (r_{ui} - q_u p_i^T)^2 + \lambda_1 \sum_u \|q_u\|^2 + \lambda_2 \sum_i \|p_i\|^2$$

"error" | "length"

If matrix has more data, the error term will dominate, so min problem will try to optimize error term more

If matrix has less data, the regularization will have more effect and it will try to reduce regularization term more

for each iteration in Alternative Opt. $p^{(t+1)} = \sum_i \arg\min \sum_{i\in S_{*,i}} (r_{ui} - q_u p_i^T)^2 + \lambda_2 \|p_i\|^2$

for each iteration in SGD $q_u \leftarrow q_u - \eta\left(\frac{\partial L}{\partial q_u} + \frac{\partial}{\partial q_u}\lambda\|q_u\|^2\right) = q_u - \eta\left(\frac{\partial L}{\partial q_u} + 2\lambda_1 q_u\right)$

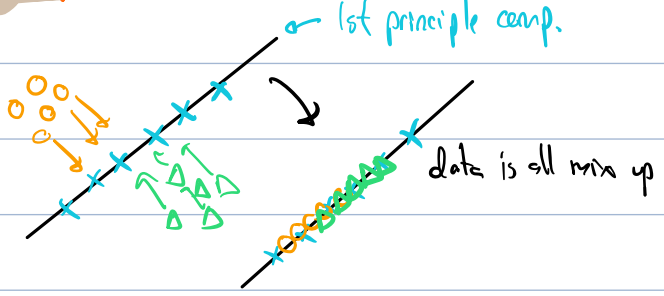
L1 – enforce sparsity → better interpretation (only few value contribute to result), less storage, faster processing

## Non-Negative Matrix Factorization

SVD might lead to factors containg negative value → hard to interpret

Constrained Optimization – $\min_{P\geq 0, Q\geq 0} \|A - QP^T\|_F$

## Neighbor Graph Methods




PCA tries to find global structure

- Far away point can become nearest neighbor
- Loss clustering structure (local similarity) after projected

Idea: Preserve local structure – low dimensional neighborhood similar to the original neighborhood

1. Construct neighbor graph of high dim data
2. Initialize points in low dim space → construct neighborhood graph of this low dim data
3. Optimize coordinates in low dim space s.t. the neighborhood graph is similar
   (move point around iteratively s.t. the points are group together again like in high dim data)

## tSNE - t Distributed Stochastic Neighbor Embedding → perform non linear dimensionality reduction

Minimize the KL-Divergence (measure of dissimilarity) → use gaussian kernel

$$\min_{y_i} KL(P\|Q) = \sum_i \sum_{j \neq i} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

↑ move $y_i$ around

0 if distribution is identical $p_{ij} = q_{ij}$

where $p_{ij}$ = high dim similarity for input $x_i$ and $x_j$

$q_{ij}$ = low dim similarity for parameter $y_i$ (to be optimized) and $y_j$

We try to make distribution of higher & lower dim neighbor graph as similar as possible

Perplexity - $\sigma^2$ - Variance of distribution to be consider = Area of neighborhood to preserve

## Autoencoder

PCA/SVD can only capture linearity

Autoencoder is a NN that Find compact representation of data by learning to reconstruct its input

Goal: Minimize reconstruction error $\min_w \frac{1}{N} \sum_{i=1}^{N} (f(x_i, w) - x_i)^2$

Find latent representation $z \in \mathbb{R}^L$ $\dim(z) << \dim(x)$ → perform non linear dimensionality reduction

$f_{enc}(x) = z$ encoder = project data to lower dim

$f_{dec}(z) \approx x$ decoder = reconstruct data from the latent code

$L << D$ allow autoencoder to capture the important feature of the data

$L \geq D$ make autoencoder learn just an identity function (learn to copy every pixel)

# Lecture 12: Clustering

Dimensionality Reduction - project high dim data into low dim space such that similarity stay the same ($z_i \in \mathbb{R}^k$)

Clustering - Group object $x_i$ into K cluster based on their similarity ($z_i \in$ discrete number)

Goal: Assign $x_i$ to one of the K clusters. s.t. (in case of non overlapping cluster)

- Objects within same group are similar to eachother
- Object btw diff group are dissimilar from eachother

## K-Means Clustering (based on distance)

Similarity = use Distance function

1. Manhattan distance $\|x_i - x_j\|_1$
2. Euclidean distance $\|x_i - x_j\|_2$

Mahalanobis $\sqrt{(x_i - x_j)^T \Sigma^{-1} (x_i - x_j)}$ generalisation of $L_2$ dist

if $\Sigma^{-1} = I$ then we got $L_2$

$\mu_k$ = centroid of cluster k  $z_{ik}$ = cluster indicator = $x_i$ belong to cluster k

K-means objective : Find best centroid $\mu$ and cluster assignments $z$

$$\min_{z,\mu} J(x,z,\mu) = \min_{z,\mu} \sum_i^N \sum_k^K z_{ik} \|x_i - \mu_k\|_2^2$$

Use **Alternating Optimization** to update $z$ and $\mu$ in turn

1. initialise centroid $\mu$
2. Update cluster indicator $z_{ik}$ (solve $\min_z J(x,z,\mu)$)  $z_{ik} = \begin{cases} 1 & \text{if } k = \min_j \|x_i - \mu_j\|^2 \\ 0 & \text{else} \end{cases}$

   calculate distance btw $x_i$ and $\mu_k$. Assign $x_i$ to cluster with least distance
3. Update Centroid (solve $\min_\mu J(x,z,\mu)$)

$\mu_k = \frac{1}{N_k} \sum_{i=1}^N z_{ik} x_i$  $N_k$ = # of data in cluster  compute mean only count data that belong to cluster.

Iterate btw 2 & 3

- initial location of $\mu$ effect perf. → if far away from data can lead to bad result

**K-Means ++ Algorithm** ← solution

1. Choose $\mu_1$ at random among the data points
2. For each point $x_i$ compute $D_i^2 = \|x_i - \mu_1\|_2^2$
3. Sample next $\mu_2$ from $\{x_i\}$ w/ probability proportional to $D_i^2$

   so the further away $x_i$, the more likely it's to get chosen as the next $\mu$
4. Recompute $D_i$ for all centroid $\mu$  $D_i^2 = \min\{\|x_i - \mu_1\|_2^2, \|x_i - \mu_2\|_2^2\}$
5. Repeat 3 & 4 to find next $\mu$ until get all k centroid.

Limitation of K-Mean

either it's in cluster(1) or not (0) → no probability for each cluster → GMM

Can't detect cluster w/ overlapping convex hull, sensitive to outliers, no uncertainty measure, sensitive to initialisation.

**Gaussian Mixture Model (GMM)** (Probabilistic model)

Goal : Design probabilistic model for data $p(x|\theta)$, estimate $\theta$

Latent Variable, marginalization → $p(x) = \sum_j p(x, y_j)$

$$p(x|\theta) = \sum_z p(x,z|\theta) = \sum_z p(x|z,\theta)\, p(z|\theta)$$ and then $\theta^* = \arg\min_\theta p(x|\theta)$

$z$ are never observe and only use to simplify model = latent variable

$p(x,z|\theta) = p(x|z,\theta)\, p(z|\theta)$  Param $\theta = \{\pi, \mu, \Sigma\}$

likelihood, prior

where $p(z|\theta) = \text{Category}(\pi)$ ← cluster; prob for each cluster $\pi = [0.2\ 0.5\ 0.3]$

given x belong to cluster k

$p(x|z_k = 1, \theta) = \mathcal{N}(x|\mu_k, \Sigma_k)$ ← param for each cluster k

# Generative Process of GMM

We have K cluster, known parameter $\{\mu_k, \Sigma_k\}$ and $\pi$ for all cluster

1. Sample cluster $z_k$ based on $\pi \rightarrow z \sim Cat(\pi)$
2. Get $\mu_k, \Sigma_k$ from $z_k$ then we can sample $x$ from the likelihood distribution $x \sim N(\mu_k, \Sigma_k)$ for $z_k$

## What if we know x but not $\pi, \Sigma, \mu$

$$p(x|\pi,\mu,\Sigma) \overset{\text{margin}}{=} \sum_{k=1}^{K} p(x, z_k=1|\pi,\mu_k,\Sigma_k) = \sum_{k=1}^{K} p(z_k=1|\pi)\, p(x|z_k=1,\mu_k,\Sigma_k) = \sum_{k=1}^{K} \pi_k \cdot N(x|\mu_k,\Sigma_k)$$

GMM Likelihood $= p(x|\pi,\mu,\Sigma) = \sum_{k=1}^{K} \pi_k \cdot N(x|\mu_k,\Sigma_k)$ = weighted sum of gaussian distribution

Log likelihood $\log p(x|\pi,\mu,\Sigma) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} \pi_k \cdot N(x_i|\mu_k,\Sigma_k)\right)$

Learning - $\pi^*, \mu^*, \Sigma^* = \arg\max_{\pi,\mu,\Sigma} \log p(x|\pi,\mu,\Sigma)$

Inference - Given $\{\pi,\mu,\Sigma\}$, we want to assign point to cluster

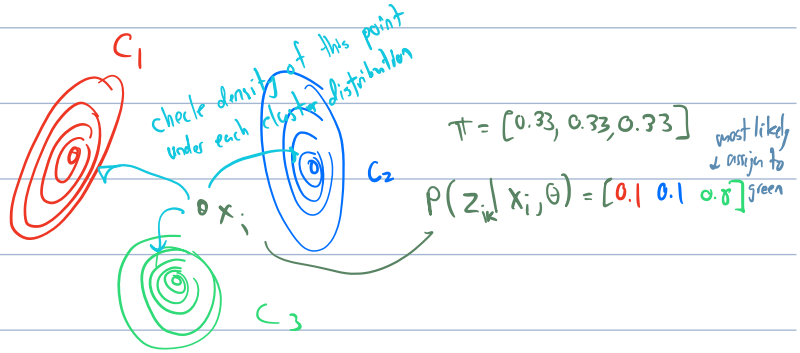
- Compute Posterior Distribution of cluster indicator $z$ $P(z|x,\pi,\mu,\Sigma)$

## Inference in GMM

$$p(z_{ik}=1|x_i,\pi,\mu,\Sigma) = \frac{p(z_{ik}=1|\pi)\, p(x_i|z_{ik}=1,\mu_k,\Sigma_k)}{p(x_i|\pi,\mu,\Sigma)} = \frac{\pi_k N(x_i|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j N(x_i|\mu_j,\Sigma_j)}$$

We call this "responsibility" of component k for observation i

$\gamma(z_{ik}) = p(z_{ik}=1|x_i,\pi,\mu,\Sigma)$ → prob of assigning to the cluster given data



# Learning in GMM

add z to the $p(x|\theta)$ to make it easier to solve

$$\pi^*, \mu^*, \Sigma^* = \max_{\pi,\mu,\Sigma} \log p(X, Z | \pi, \mu, \Sigma) \Rightarrow \text{Estimate } \mu^{(t+1)}, \pi^{(t+1)}, \Sigma^{(t+1)} = \max_{\pi,\mu,\Sigma} \mathbb{E}_{z\sim\gamma_t(z)} [\log p(x, z | \pi, \mu, \Sigma)]$$

## Use EM Algorith (Expectation Maximization)

### E Step

Compute & Update responsibility $\gamma$ (posterior distribution) based on initial/previous $\pi_t, \Sigma_t, \mu_t$

$$\gamma_t(z_{ik}) = p(z_k | x_i, \pi_t^{(t)}, \mu_k^{(t)}, \Sigma_k^{(t)}) = \frac{p(z_k|\pi^{(t)}) p(x_i | \mu_k^{(t)}, \Sigma_k^{(t)})}{p(x_i | \pi^{(t)}, \mu_k^{(t)}, \Sigma_k^{(t)})} = \frac{\pi_k N(x_i | \mu_k^{(t)}, \Sigma_k^{(t)})}{\sum_{j=1}^{K} \pi_j^{(t)} N(x_i | \mu_j^{(t)}, \Sigma_j^{(t)})}$$

→ get $\gamma_t(z_{ik})$ to be use in M step

### M-Step - update $\theta$

Maximize Expectation $\theta^* = \max_\theta \mathbb{E}_{z\sim\gamma_t(z)}(\log p(x,z|\theta))$ → get optimal $\pi^*, \mu^*, \Sigma^*$

$$\theta^{(t+1)} = \max_\theta \mathbb{E}_{z\sim\gamma_t(z)}(\log p(x,z|\theta)) = \max_\theta \sum_{k=1}^{K} \gamma_t(z_{ik}) \log(x_i, z_{ik}|\theta)$$

use $\gamma_t(z)$ that was calculate from previous step

$$\mu_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_t(z_{ik}) x_i \qquad \Sigma_k^{(t+1)} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_t(z_{ik}) (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T \qquad \pi_k^{(t+1)} = \frac{N_k}{N}$$

get $\mu^{(t+1)}, \Sigma^{(t+1)}, \pi^{(t+1)}$ → use in E step to compute new $\gamma_{t+1}(z_{ik})$ → get $\theta^{(t+2)}$ from M-step → iterate until converge

## Kinda Explanation of EM ...? (go read pg. 30-34)

Denote $L(q, \theta) = \mathbb{E}_{z\sim q}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] = \mathbb{E}_{z\sim q}[\log p(x,z|\theta)] + H[q(\cdot)]$

$L(q,\theta)$: latent distribution $(z)$; $\mathbb{E}_{z\sim q}$: GMM; $H[q(\cdot)]$: indep. of $\theta$

It's proved that $L(q,\theta)$ is lower bound of $\log p(x|\theta)$ for any $q$ → any lower bound

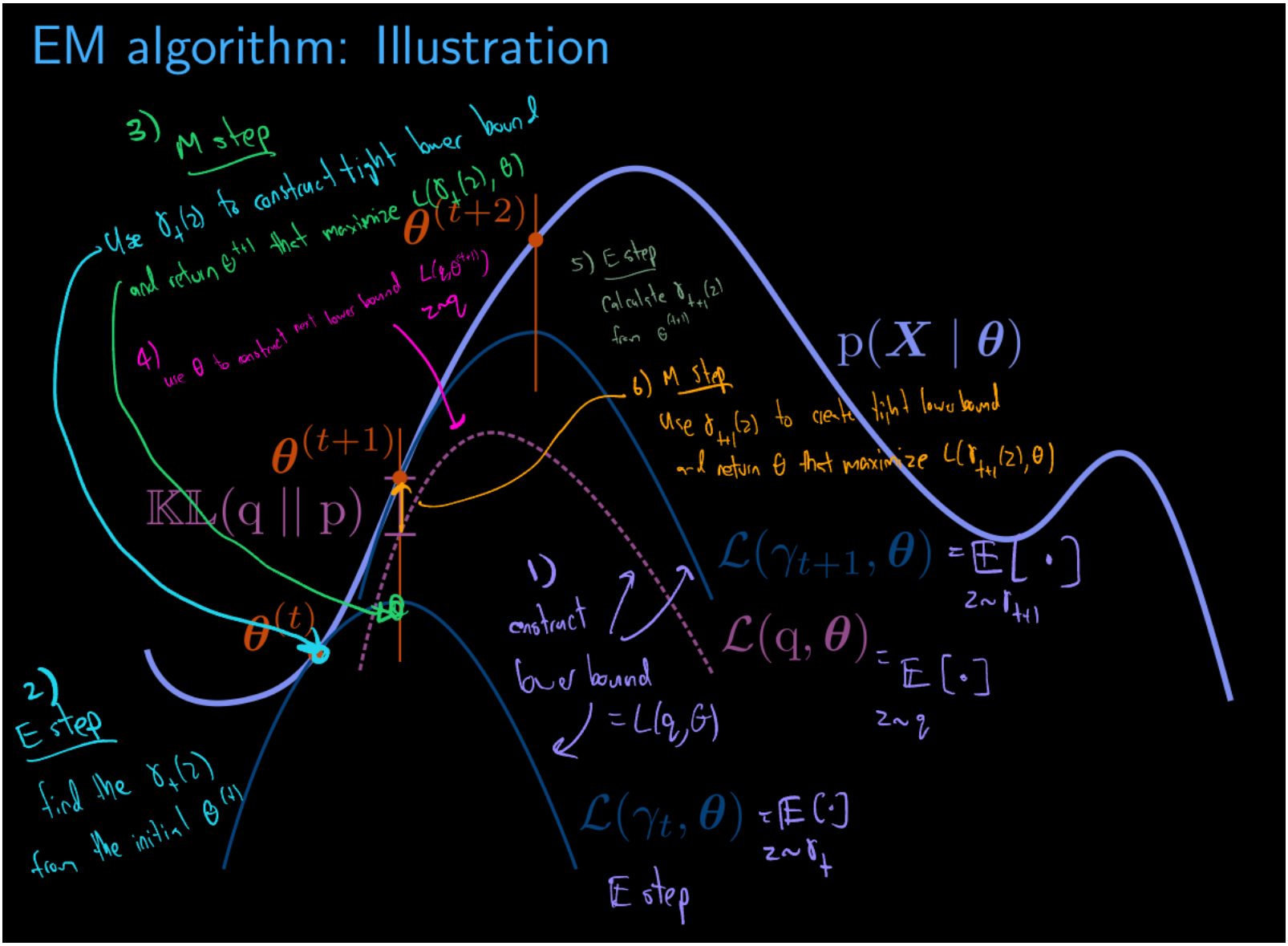
If $q(z) = \gamma(z) = p(z|x,\theta)$, then we maximize $L(q,\theta)$ and $L(q,\theta) = \log p(x|\theta)$ (tight lowerbound)

From the maximum $L(q,\theta)$ → we find $\theta$ that maximize $L(q\theta)$ → get $\theta^{(t+1)}$ to use in $L(q,\theta)$ and $\gamma(z)$ to get the next tight lower bound

↳ $\theta$ that have higher.

As a consequence - $L(q,\theta)$ is push up against $\log p(X|\theta)$ always

- If $L(q,\theta)$ doesn't change, we are at local maximum of $\log p(X|\theta)$ and EM converge




## Hierarchical Clustering

Agglomerative – Bottom up, start with all cluster and start merging each pair until get 1 big cluster

Divisive – top-down – start from 1 Cluster and split it

How to choose pair (let $x_i \in G$, $x_j \in H$ cluster $G, H$)

Complete-linkage → max $d(x_i, x_j)$ → look at max dist of sample btw 2 cluster

Single Linkage → min $d(x_i, x_j)$ → min dist

Centroid → dist from center of cluster

Unweighted Average → Mean sample distance $\frac{1}{N_G N_H} \sum_{x_i \in G} \sum_{x_j \in H} d(x_i, x_j)$

Weighted Average