KNN: $||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}||_2 = \sqrt{\sum_{j=1}^{p}(x_j^{(a)} - x_j^{(b)})^2}$ $\text{cosine}(\mathbf{x}^{(a)}, \mathbf{x}^{(b)}) = \frac{\mathbf{x}^{(a)} \cdot \mathbf{x}^{(b)}}{||\mathbf{x}^{(a)}||_2 ||\mathbf{x}^{(b)}||_2}$

Accuracy: $\text{Accuracy} = \frac{\text{Num Correct Predictions}}{\text{Num Predictions}} = \frac{\sum_{i=1}^{N} \mathbb{I}[t^{(i)} = y^{(i)}]}{N}$

Decision Trees:

Entropy: $H(X) = -\mathbb{E}_{X \sim p}[\log_2 p(X)] = -\sum_{x \in X} p(x) \log_2 p(x)$

Joint entropy: $H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)$

Conditional Entropy: $H(Y|X = x) = -\sum_{y} p(y|x) \log_2 p(y|x)$ p(y|x) = p(x, y)/p(x) $p(x) = \sum_y p(x, y)$

Chain rule: H(X, Y) = H(X|Y) + H(Y) = H(Y |X) + H(X) **indep**, H(Y |X) = H(Y), but

H(Y |Y) = 0. H(Y |X) ≤ H(Y) **Exp. Cond.Entro:** $H(Y|X) = \mathbb{E}_x[H(Y|x)] = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x)$ **Information gain:** $IG(Y|X) = H(Y) - H(Y|X)$

Decision Tree Algo: Split on most informative attribute. We desire small tree with informative nodes near the root. **Occam's Razor:** Find simplest hypothesis that fits the observation **DT Problems:** 1. Exponentially less data at lower levels. 2. Too big a tree can overfit data. 3. Mistakes at top-level propagate down tree. **DT Cont. attr.:** Split based on a threshold, chosen to maximize IG – Decision tree can also be used for regression. Choose splits to minimize squared error, rather than maximize information gain. **Compared to kNN:** Good with discrete attributes, missing values/Robust to scale of inputs/More interpretable less curse of diimensionality. kNN handles features that interact in complex ways/incorp interesting distance measures (e.g. shape context) **Linear Regression: $\mathbf{w}^T \mathbf{x}$ (bias goes in w)** $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$ **Squared loss function (bad x):**

$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$ **Cost function :** $\mathcal{E}(w, b) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)})$

$\mathcal{E}(\mathbf{w}) = \frac{1}{2N}||\mathbf{X}\mathbf{w} - \mathbf{t}||_2^2$ $\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) = \frac{1}{N}\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t})$

$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$ **Gradient Desc:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla_{\mathbf{w}}\mathcal{E}$ **Polynomial Regression:** Fit data to M polynomial feature map: y = w^T ψ(x) where ψ(x) = [1, x, x2 , ...]^T . Large M = large coefficients **L sq reg:** $\mathcal{R}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_j w_j^2$

Cost: $\mathcal{E}_{\text{reg}}(\mathbf{w}) = \mathcal{E}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w})$. Large lambda-> underfit **Linear model for binary class:** $z = \mathbf{w}^T\mathbf{x}$, $y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$ **0-1 Loss:** Hard to optimize due to step function. Changing weights by small amount has no effect on loss. **Logistic Reg**: **Logistic Activation Function (sigmoid):** $\sigma(z) = \frac{1}{1 + e^{-z}}$ **Logit**: $\sigma^{-1}(y) = \log(y/(1-y))$

Cross Entropy loss: $\mathcal{L}_{CE}(y, t) = \begin{cases} -\log(y) \text{ if } t = 1 \\ -\log(1 - y) \text{ if } t = 0 \end{cases}$

$= -t\log(y) - (1 - t)\log(1 - y)$

$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial y} \cdot y(1-y) \cdot x_j = (-\frac{t}{y} + \frac{1-t}{1-y}) \cdot y(1-y) \cdot x_j = (y - t)x_j$

$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})\mathbf{x}^{(i)}$

Fancy words: (non) Parametric, Hypothesis (space), inductive bias.

$\frac{\partial \mathcal{E}}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}(y^{(i)}, t^{(i)})}{\partial w}$ Stochastic Gradient desc: #repeat until converge: #a group of mini batches of size k#for each mini batch #estimate the gradient using the mini batch size # update the parameters based on the estimate. Each pass of the inner loop is an iteration -> One iteration is one upadate for each weight Each pass in the outerloop is called an espoch: -> one espoch is one pass over the entire data-set num_iter = num_espoch * N/k

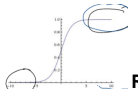
Problem with gradient desc: reviens **Multi Class Classification:**

$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ $\mathbf{y} = \text{softmax}(\mathbf{z})$ $\mathcal{L}_{CE} = -\mathbf{t}^T\log(\mathbf{y})$

$y_k = \text{softmax}(z_1, ..., z_K)_k = \frac{e^{z_k}}{\sum_{m=1}^{K} e^{z_m}}$ $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{CE}}{\partial z_k}\frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k)\mathbf{x}$

$\mathbf{w}_k \leftarrow \mathbf{w}_k - \frac{\alpha}{N}\sum_{i=1}^{N}(y_k^{(i)} - t_k^{(i)}) \cdot \mathbf{x}^{(i)}$ (for each row of W) **Multilayer perceptrons:**

$y = \phi\left(\mathbf{w}^\top\mathbf{x} + b\right)$.

Sigmoid:  **ReLU:** $\text{ReLU}(z) = \max(z, 0)$ **(derivative is 1,0)**

Size of inner layers h: another hyperparameter

$\mathbf{h}^{(1)} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$

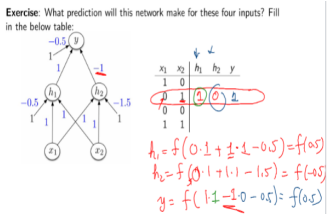
$\mathbf{h}^{(2)} = f(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$

...

$\mathbf{h}^{(L-1)} = f(\mathbf{W}^{(L-1)}\mathbf{h}^{(L-2)} + \mathbf{b}^{(L-1)})$

$\mathbf{z} = \mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}$

$\mathbf{y} = \text{softmax}(\mathbf{z})$



(xor)

Training NN (backprop)

```
for i = 1..N
    Compute v_i as function of Pa(v_i)
v̄_N = 1
for i = N-1..1
    v̄_i = Σ_{j∈Ch(v_i)} v̄_j ∂v_j/∂v_i
```

For $z = W_1x + b$, $h = \sigma(z)$, $y = W_2h + b_2$, $L = \frac{1}{2}||t - y||^2$, $\bar{L} = 1$, $\bar{y} = \bar{L}(y - t)$, $\bar{W}_2 = \bar{y}h^\top$, $\bar{b}_2 = \bar{y}$, $\bar{h} = W_2^T\bar{y}$, $\bar{z} = \bar{h} \circ \sigma'(z)$ $\bar{W}_1 = \bar{z}x^\top$, $\bar{b}_1 = \bar{z}$.
Want $\frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b}$. In sigmoid activation with squared loss,

$$L = \frac{1}{2}(y - t)^2, y = \sigma(z), z = wx + b$$

$$\frac{\partial \mathcal{L}}{\partial y} = y - t, \frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial y}\sigma'(z), \frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z}x, \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$$

$\bar{\mathcal{L}} = 1$
$\bar{y_k} = \bar{\mathcal{L}}(y_k - t_k)$
$\bar{w}_{ki}^{(2)} = \bar{y_k}h_i$
$\bar{b}_k^{(2)} = \bar{y_k}$
$\bar{h}_i = \sum_k \bar{y_k}w_{ki}^{(2)}$
$\bar{z}_i = \bar{h}_i\sigma'(z_i)$
$\bar{w}_{ij}^{(1)} = \bar{z}_i x_j$
$\bar{b}_i^{(1)} = \bar{z}_i$

$\bar{\mathcal{L}} = 1$
$\bar{\mathbf{y}} = \bar{\mathcal{L}}(\mathbf{y} - \mathbf{t})$
$\bar{\mathbf{W}}^{(2)} = \bar{\mathbf{y}}\mathbf{h}^T$
$\bar{\mathbf{b}}^{(2)} = \bar{\mathbf{y}}$
$\bar{\mathbf{h}} = \mathbf{W}^{(2)T}\bar{\mathbf{y}}$
$\bar{\mathbf{z}} = \bar{\mathbf{h}} \circ \sigma'(\mathbf{z})$
$\bar{\mathbf{W}}^{(1)} = \bar{\mathbf{z}}\mathbf{x}^T$
$\bar{\mathbf{b}}^{(1)} = \bar{\mathbf{z}}$

Vectorized: $\mathbf{z} \in \mathcal{R}^N, \mathbf{y} \in \mathcal{R}^M$ $\bar{z_j} = \sum_k \bar{y_k}\frac{\partial y_k}{\partial z_j}$ $\bar{\mathbf{z}} = \frac{\partial \mathbf{y}}{\partial \mathbf{z}}^\top \bar{\mathbf{y}}$,

$$(\frac{\partial \mathbf{y}}{\partial \mathbf{z}})_{M \times N} = \begin{pmatrix} \frac{\partial y_1}{\partial z_1} & \cdots & \frac{\partial y_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial z_1} & \cdots & \frac{\partial y_m}{\partial z_n} \end{pmatrix}$$

Bias Variance decomp:

Bias: How badly will our model perform on average, across the possible datasets we could receive. How wrong is the expected prediction. E(y* -y)^2. High bias comes from low-capacity models.
Variance: Var(y), How the average prediction (across various traing set samples) differs from the prediction we get from one particular training set. High capacity models have high variance.
Bayes Error: var(t) The inherent unpredictiblity of the targets.

Bagging: Take a single dataset D with n examples. Generate m new datasets, each by sampling n training examples from D, with replacement. Average the predictions of the models train on each of these datasets. Problem: Datasets are not indep. so there's no 1/m variance reduction. Possible to show that if var σ 2 and correlation ρ:

$$Var(1/m\sum_{i=1}^{m} h_i(x)) = (1/m(1 - p) + \rho)\sigma^2$$

May be more advantageous to introduce more variability into algo, as long as it reduces the correlation between samples. Bagging reduces overfitting by averaging predictions but does not reduce bias. **Random Forests:** Bagged decision trees where when choosing each of node, choose split on random subset from d input features. Improves the variance reduction of bagging by reduced correlation b/w tree ~ p. Bagging does not control Bayes error.

Random useful things: $\sigma = \sqrt{\frac{\sum(X - \mu)^2}{N}}$ $\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$\mathbb{E}[(\hat{f}(\mathbf{x}) - t)^2] = \underbrace{(\mathbb{E}[y_* - y])^2}_{\text{bias}} + \underbrace{\text{Var}(y)}_{\text{variance}} + \underbrace{\text{Var}(t)}_{\text{Bayes error}}$$

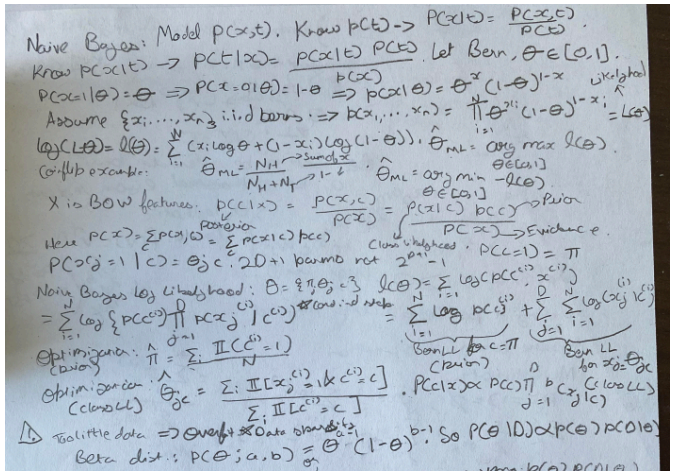
Fairness metrics: Demographic parity, Equalized Odds (Accuracy parity), Indiv. Fairness. **Demographic Parity:** Acceptance rates of applications from both groups must be equal Also known as "independence" (terminology from statistics) Problem: Fairness is measured at a group level Model can hire qualified people from one group, and random people from the other. **Equalized Odds:** Model should be equally accurate across both groups Also known as "accuracy parity" Problem: False positives and false negatives have different impacts Does not help to close the gap between the two groups. **Individual Fairness:** Similar individuals from different groups should be treated similarly Problem: Hard to determine appropriate measure of "similarity" of inputs. **Ideas: Pre-processing**: remove information correlated to sensitive attributes **Add regularization term:** add a "fairness" regularizer We saw regularizers that encourage smaller weights This kind of regularizer would encourage "fairness" **Post-processing:** change the way we use a model to make predictions.

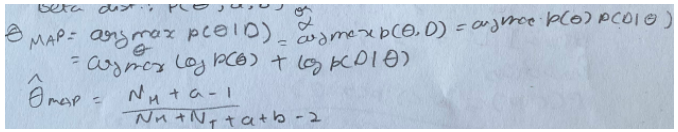
Naive Bayes

Law of total probability:

$$p(\mathbf{x}) = \sum_c p(\mathbf{x}|c)p(c)$$

specifying a joint distribution p(c, x1, . . . , xD) over D + 1 binary variables requires $2^{D+1} - 1$specifying a joint distribution over D + 1 binary variables requires $2^{D+1} - 1$ **Solution**: assume each word feature x_i is conditionally independent given class c, 2D + 1 now





Baye's Classifier: Given features $x = [x_1, x_2, ..., x_D]^T$ we want to compute class probabilities using Bayes Rule $P(c|x) = \frac{p(x|c)}{p(x)} = \frac{p(x|c)p(c)}{p(x)}$, formally, posterior = $\frac{\text{class likehood x prior}}{\text{evidence}}$

Log Likelihood: $\sum_{i=1}^{N} \log p(c^{(i)}) = \sum_{i=1}^{N} c^{(i)} \log \pi + \sum_{i=1}^{N} (1 - c^{(i)}) \log(1 - \pi)$

Naive Bayes (Inference): Apply Bayes' Rule, shorthand notation: $p(c|x) \propto p(c) \prod_{j=1}^{D} p(x_j|c)$ – cheap learning algorithm. At training time, estimate parameters using log likelihood, compute co-occurrence counts of each feature with the labels. At test time, apply Bayes' Rule.

MLE Want to maximize the log likelihood: $l(\theta) = \sum log(p(X_i|\theta))$.
Too little data can cause overfitting.

Bayesian Parameter Estimation: MLE treat observation as random variables, but parameters are not. Bayesian treats parameters as random variables as well. Define *prior distribution* $p(\theta)$, which encodes beliefs about parameter before observing data. *likelihood* $p(D|\theta)$, same as MLE. Updating beliefs by computing posterior distribution using Bayes' Rule: $p(\theta|D) = \frac{p(\theta)p(D|\theta)}{\int p(\theta')p(D|\theta')d\theta'}$

MAP Estimator: Find the most likely parameter settings under the posterior. Want to maximize the log likelihood: $l(\theta) = \log(p(w)) + \sum \log(p(X_i|\theta))$. $\theta_{MAP} = \text{argmax}_\theta \log p(\theta) + \log p(D|\theta)$

Exercise: Learning each θ_{jc}

Formula	
$\hat{\theta}_{ML}$	$\frac{N_H}{N_H+N_T}$
$\mathbb{E}[\theta\|\mathcal{D}]$	$\frac{N_H+a}{N_H+N_T+a+b}$
$\hat{\theta}_{MAP}$	$\frac{N_H+a-1}{N_H+N_T+a+b-2}$

Exercise Compute the MAP estimate for π and each θ_{jc} in this data set, assuming a Beta distribution priors for each parameter with $a = 2$, $b = 2$.

- he was funny and sad, "positive"
- movie was sad, "positive"
- great movie, "positive"
- movie was not funny, "negative"
- movie was not sad, "negative"

Then, use your parameters to estimate the probabilities that these reviews are positive:

- movie was funny
- he was not funny

	+ve	-ve
he	2/5	1/4
great	2/5	1/4
and	2/5	1/4

Gaussian Discriminnt Analysis:

PDF of the univariate Gaussian distribution:

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

PDF of the multivariate Gaussian distribution:

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right] \quad \boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] = \begin{pmatrix}\mu_1 \\ \vdots \\ \mu_D\end{pmatrix}$$

$$\text{Cov(x,y)} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N} \qquad \boldsymbol{\Sigma} = \text{Cov}(\mathbf{x}) = \mathbb{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top]$$

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)} \qquad \hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \hat{\mu})(\mathbf{x}^{(i)} - \hat{\mu})^\top$$

Gaussian Discriminative Analysis (GDA):
Assumes that $p(\mathbf{x}|t)$ is distributed according to multivariate Gaussian distribution.

$$p(\mathbf{x}|t=k) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_k|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right]$$

Where $|\Sigma_k|$ denotes the determinant of the matrix, and D is dimension of $\mathbf{x}$. Each class k has a mean vector $\boldsymbol{\mu}_k$ and a covariance matrix $\boldsymbol{\Sigma}_k$
Use MLE to find empirical mean and empirical covariance matrix: $\widehat{\boldsymbol{\mu}_k}$ *and* $\widehat{\boldsymbol{\Sigma}_k}$. $\boldsymbol{\Sigma}_k$ has $O(D^2)$ parameters – could be hard to estimate
To find log-likelihood of GDA: Assume data $\mathbf{x}$ are drawn from gaussian, with μ and Σ. We calculate log-likelihood $l(\mu, \Sigma)$, then take derivative w.r.t μ and Σ, evaluate at zero to find the MLE $\hat{\mu}$ and $\hat{\Sigma}$.
Decision boundary of GDA: usually quadratic. It is based on class posterior.
GDA makes decisions by comparing class probabilities.

$$\log p(t_k|\boldsymbol{x}) = \log p(\boldsymbol{x}|t_k) + \log p(t_k) - \log p(\boldsymbol{x}) \quad \#By\ bayes\ rule$$
$$= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_k^{-1}| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) + \log p(t_k) - \log p(\boldsymbol{x})$$

Now, suppose I have two classes, l and k. Then the decision boundary equals the follow: decision boundary is $\log p(t_k|\boldsymbol{x}) = \log p(t_l|\boldsymbol{x})$, which is equivalent to

$$(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) = (\boldsymbol{x}-\boldsymbol{\mu}_l)^T\boldsymbol{\Sigma}_l^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_l) + C_{kl}$$

Quadratic function in $\mathbf{x} \Rightarrow$ **quadratic decision boundary!**
What is C_{kl}? What if $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_l$? **-> can be derived. Easy.**

Spectral Decomposition

For matrix A, anything satisfying Av = λv, is an eigenvector and its associated eigenvalue
Our DxD matrix will have at most D eigenvalues
Since our A is symmetric then the Spectral Theorem tells us that: All its **eigenvalues** are **real-valued.**
There is a full set of linearly independent **eigenvectors** that **form a basis for R^D**
These **eigenvectors** can be chosen to be **real-valued** and **orthonormal.**

Quadratic Forms of **Symmetric matrices**
$v^TAv > 0$, for all v =\= 0 -> the quadratic form curves upwards; A is **positive definite**; A ≻ 0
$v^TAv \geq 0$ for a v -> A is **positive semidefinite**; A ≽ 0
$v^TAv < 0$ for all v =\= 0, we say A is **negative definite**, denoted A ≺ 0
v^TAv can be positive or negative then A is **indefinite**
Claim: A matrix A is positive semi-definite iff all of its eigenvalues are nonnegative.
If A is both diagonal and positive definite (i.e. its diagonal entries are positive), then the ellipses are axis-aligned.
Comparing GDA vs Logistic Regression: GDA makes stronger modeling assumption: assumes class-conditional data is multivariate Gaussian. If this is true, GDA is asymptotically efficient. But LR is more robust, less sensitive to incorrect modeling assumptions. When the class-conditional distributions are non-Gaussian (true almost always), LR usually beats GDA. Also, GDA has quadratic (when not shared covariance), LR has linear decision boundary.

Gaussian Naive Bayes classifier assumes that the likelihoods are Gaussian:

$$p(x_j|c=k) = \frac{1}{\sqrt{2\pi\sigma_{jk}}} \exp\left[\frac{-(x_j - \mu_{jk})^2}{2\sigma_{jk}^2}\right]$$

- this is just a 1-dim Gaussian, one for each input dimension
- Model the same as GDA with diagonal covariance matrix

Unsupervised learning

K-means: Group data points into clusters. Relies on assumption that data depends on latent variables. The objective:
$\min_{\{\mathbf{m}_k\},\{\mathbf{r}^n\}} \sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$
Where $\{\mathbf{m}_k\}$ are the k means, $\mathbf{r}^n \in \mathbb{R}^K$ is the assignment vector for $\mathbf{x}^n$. $\mathbf{r}^n$ is one-hot if hard k-means and like a probability distribution if soft k-means. Finding the min is NP-hard in general.
Alternating minimization involves fixing one of $\{\mathbf{m}_k\}$, or $\{\mathbf{r}^n\}$, optimizing on that, then alternate to other. First initialize clusters, then iterate between assignment step (assign data points to nearest clusters), and refitting step (move every cluster centre to the means of the data points assigned to it). This algo is guaranteed to converge to a local optimum. **Test of convergence**: if the assignments don't change over an iteration, then stop. **Refitting Step**: For each centre: $\mathbf{m}_k = (\sum_n r_k^{(n)}\mathbf{x}^n)/(\sum_n \mathbf{r}_k^{(n)})$.
Assignment Step: For soft k-means:
$r_k^{(n)} = (\exp(-\beta||\mathbf{m}_k - \mathbf{x}^n||^2))/(\sum_j \exp(-\beta||\mathbf{m}_j - \mathbf{x}^n||^2)))$
$\mathbf{r}^{(n)} = \text{softmax}(-\beta\{||\mathbf{m}_k - \mathbf{x}^n||^2\}_{k=1}^K)$
For regular: $r_k^{(n)} = \mathbb{I}[k'^{(n)} = k]$, for $k = 1, ..., K$, where $k'^{(n)} = \text{argmin}_k||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$. As β goes to infinity, soft k-means becomes regular.

By marginalizing out z, we get a density over the observables:

$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)\,p(z)$$

This is called a **latent variable model**.
If $p(z)$ is a categorical distribution, this is a **mixture model**, and different values of z correspond to different **components**.

GMM:

$$p(\mathbf{x}) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \qquad \sum_{k=1}^{K}\pi_k = 1$$

Maximum likelyhood principle:

$$\log p(\mathcal{D};\theta) = \sum_{n=1}^{N}\log p(\mathbf{x}^{(n)};\theta)$$

Gaussian Mixture Models: A generative approach to clustering. We need a method to judge what it means to cluster the data well. Assume that the data was generated from a model. We then adjust the model parameters to maximize the probability that the data was generated by our model. Given $\mathcal{D}$, we assume that the data point, $\mathbf{x}$, was generated by choosing a cluster $k \in \{1, ..., K\}$, such that $p(z = k) = \pi_k$, and sampling from $\mathcal{N}(\mathbf{x}|\mu_k, \boldsymbol{\Sigma}_k)$, i.e. $p(\mathbf{x}|z = k) = \mathcal{N}(\mathbf{x}|\mu_k, \boldsymbol{\Sigma}_k)$. To find the best parameters, we want to max:

$$\log p(\mathcal{D}) = \sum_{n=1}^{N}\log p(\mathbf{x}^{(n)}) = \sum_{n=1}^{N}\log\left(\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \boldsymbol{\Sigma}_k)\right)$$

EM Algorithm for GMM: A local algorithm for GMM. First initialize the starting μ_k and π_k. Then iterate between the E step and M step until convergence (check by looking at log likelihood).
E step: posterior probability over z over current model (how much we think a datapoint was generated by a cluster):

$$r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)}) = \frac{\pi_k\mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \boldsymbol{\Sigma}_j)}$$

M step: update the model parameters π_k, μ_k to optimize the expected complete data log-likelihood. $\mu_k = (1/N_k)\sum_{n=1}^{N} r_k^{(n)}\mathbf{x}^{(n)}, \pi_k = N_k/N$, where $N_k = \sum_{n=1}^{N} r_k^{(n)}$.

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_{i=1}^{N} r_k^{(i)}}\sum_{i=1}^{N} r_k^{(i)}(\mathbf{x}^{(i)} - \mu_k)(\mathbf{x}^{(i)} - \mu_k)^\top$$

Both k-means and EM suffer from getting stuck at local minima BMM: Bernouilli Mixture Model

$$p(\mathbf{x}) = \sum_k\prod_j p(x_j|z=k)\,p(z=k) \qquad r_k^{(i)} = p(z^{(i)} = k|\mathbf{x}^{(i)}) = \frac{p(z=k)\,p(\mathbf{x}|z=k)}{\sum_\ell p(z=\ell)\,p(\mathbf{x}|z=\ell)}$$

$$\text{Proj}_S(\mathbf{x}) = \sum_{i=1}^{K} z_i\mathbf{u}_i \text{ where } z_i = \mathbf{x}^\top\mathbf{u}_i \qquad \text{Proj}_S(\mathbf{x}) = \mathbf{Uz} \text{ where } \mathbf{z} = \mathbf{U}^\top(\mathbf{x} - \hat{\mu})$$

Min reconstruction error/max variance of z's

$$\min_{\mathbf{U}} \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2 \qquad \max_{\mathbf{U}} \frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\|^2$$

Because the columns of $\mathbf{U}$ are orthogonal, $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, the norm of centered reconstruction is equal to norm of representation:

$$\|\tilde{\mathbf{x}} - \hat{\boldsymbol{\mu}}\|^2 = \|\mathbf{Uz}\|^2 = \mathbf{z}^\top\mathbf{U}^\top\mathbf{Uz} = \mathbf{z}^\top\mathbf{z} = \|\mathbf{z}\|^2$$

So, we need to choose a = Q^Tu that maximizes
Rotate X using Q-> covariance of transformed data will be diagonal
Projected variance = constant - reconstruction error

PCA: ui perp basis

$$\tilde{\mathbf{x}} = \text{Proj}_S(\mathbf{x}) = \sum_{i=1}^{K} z_i\mathbf{u}_i \text{ where } z_i = \mathbf{x}^\top\mathbf{u}_i \in \mathbb{R}$$

$$\mathbf{a}^\top\boldsymbol{\Lambda}\mathbf{a} = \sum_{j=1}^{D}\lambda_j a_j^2$$

Our goal is to show that $\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2 = \text{const} - \frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}} - \hat{\boldsymbol{\mu}}\|^2$

Claim: The vectors $\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}$ and $\tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}$ are **orthogonal** (and we can thus apply the Pythagorean theorem).

Proof: First, we can write $\tilde{\mathbf{x}}^{(i)} = \hat{\boldsymbol{\mu}} + \mathbf{U}\mathbf{U}^\top(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})$.

We know that two vectors $\mathbf{a}, \mathbf{b}$ are orthogonal $\iff \mathbf{a}^\top\mathbf{b} = 0$.

So we compute $(\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}})^\top(\tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})$:

$$\begin{aligned}&(\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}})^\top(\tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}) \\ &= (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top\mathbf{U}\mathbf{U}^\top(\hat{\boldsymbol{\mu}} - \mathbf{x}^{(i)} + \mathbf{U}\mathbf{U}^\top(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})) \\ &= (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top\mathbf{U}\mathbf{U}^\top(\hat{\boldsymbol{\mu}} - \mathbf{x}^{(i)}) + (\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}})^\top\mathbf{U}\mathbf{U}^\top(\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}) \\ &= 0\end{aligned}$$

The Pythagorean Theorem tells us:

$$\|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\|^2 + \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2 = \|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\|^2 \quad \text{for each } i$$

By averaging over data and from observation 2, we obtain

$$\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}^{(i)} - \hat{\boldsymbol{\mu}}\|^2}_{\text{projected variance}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2}_{\text{reconstruction error}} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}\|^2}_{\text{constant}}$$

Therefore, **projected variance = constant - reconstruction error**

Maximizing the variance is equivalent to minimizing the reconstruction error!

Principle Componenet Analysis (PCA): Reduces the dimension of data which can help with saving computation/memory and reducing overfitting.
Given datapoints, consider the sample mean $\hat{\mu}$ and the sample covariance $\hat{\Sigma} = \frac{1}{N}\sum(x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^\top$. Can find a subspace S of $\mathbb{R}^D$ with orthonormal basis $u_1, ..., u_K$ that "best represents" our $x^{(i)} - \hat{\mu}$ (or minimizes reconstruction squared error loss or maximizes variance of reconstructions (sum of squares of distances to the mean)) by doing spectral decomposition of $\hat{\Sigma}$, sorting the eigenvalues (which are all non-negative as $\hat{\Sigma}$ is positive semidefinite) from biggest to smallest, then taking the first K eigenvectors as $u_1, ..., u_k$. We project points $x^{(i)} - \hat{\mu}$ onto S by applying the matrix $U^\top$ for $U = (u_1, ..., u_k)$. We can then

"unproject" by applying U. The features of the projected datapoints are uncorrelated.
Deriving PCA: Maximize $a^T\Lambda a = \sum_{j=1}^{D}\lambda_j a_j^2$ for $a = Q^Tu$, assume $_i$ are in descending order, by observation, since u is a unit vector, then by unitarity, a is also a unit vector: $a^Ta = u^TQQ^Tu = u^Tu$. By inspection, set $a = \pm 1$, and $a_j = 0$ for $j \neq 1$. Hence, $u = Qa = q_1$ (the top eigenvector)