kNN: CSC311 Material Summary. Good Luck, Have Fun!

Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$$

For small k: overfit. **For large k:** underfit.
Curse of dimensionality: d high → most pts are far away & approx. Same distance (distance variance is very low)
Normalizations: nearest neighbors can be sensitive to the ranges of different features.
Without normalization, the performance of kNN will be reduced.
Decision Tree:
Entropy:

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

$$H(Y|X=\text{raining}) = -\sum_{y\in Y} p(y|\text{raining})\log_2 p(y|\text{raining})$$

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{X\sim p(x)}[H(Y|X)] \\ &= \sum_{x\in X} p(x)H(Y|X=x) \\ &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 p(y|x) \\ &= -\mathbb{E}_{(X,Y)\sim p(x,y)}[\log_2 p(Y|X)] \end{aligned}$$

Chain rule: $H(X,Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)$
If X and Y are independent, then X doesn't tell us anything about Y: H(Y|X)=H(Y); But Y tell us everything about Y: H(Y|Y)=0.
By knowing X, we can only decrease uncertainty about Y: H(Y|X) <= H(Y)

$$\begin{aligned} H(X,Y) &= -\sum_{x\in \text{Im}(X)}\sum_{y\in \text{Im}(Y)} p(x,y)\log p(x,y) = \\ &= -\sum_{x\in \text{Im}(X)}\sum_{y\in \text{Im}(Y)} p(x,y)\log[p(y)p(x|y)] = \\ &= -\sum_{\substack{x\in \text{Im}(X)\\ y\in \text{Im}(Y)}} p(x,y)\log p(y) - \sum_{\substack{x\in \text{Im}(X)\\ y\in \text{Im}(Y)}} p(x,y)\log p(x|y) = \\ &= -\sum_{y\in \text{Im}(Y)} p(y)\log p(y) - \sum_{\substack{x\in \text{Im}(X)\\ y\in \text{Im}(Y)}} p(x,y)\log p(x|y) = \\ &= H(Y) + H(X|Y). \end{aligned}$$

	Cloudy	Not Cloudy
Raining	24/100	1/100
Not Raining	25/100	50/100

Example:
How much information about cloudiness do we get by discovering whether it is raining?

$$\begin{aligned} IG(Y|X) &= H(Y) - H(Y|X) \\ &\approx 0.25 \text{ bits} \end{aligned}$$

$$H(Y) = -\left(\frac{49}{100}\lg\left(\frac{49}{100}\right) + \frac{51}{100}\lg\left(\frac{51}{100}\right)\right)$$

$$H(Y|X) = \sum_{x\in X} p(x)H(Y|X=x) = \frac{25}{100} * \left(-\frac{24}{25}\lg\left(\frac{24}{25}\right) - \frac{1}{25}\lg\left(\frac{1}{25}\right)\right)$$

$$+\frac{75}{100} * \left(-\frac{25}{75}\lg\left(\frac{25}{75}\right) - \frac{50}{75}\lg\left(\frac{50}{75}\right)\right) \qquad \text{So} \quad IG(Y|X) = H(Y) - H(Y|X) = 0.25$$

If X is completely uninformative about Y: **IG(Y|X) = 0**
If X is completely informative about Y: **IG(Y|X) =H(Y)** (i.e. in this case, H(Y|X) = 0)
Probability rules:
Bayes: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

$$p(t=1|\mathbf{x}) = \frac{p(\mathbf{x}|t=1)p(t=1)}{p(\mathbf{x})} \quad \#\text{ By Bayes rule}$$
$$= \frac{p(\mathbf{x}|t=1)p(t=1)}{p(\mathbf{x}|t=0)p(t=0) + p(\mathbf{x}|t=1)p(t=1)}$$

Correlation $\rho = Cov(x,y)$,
so $Cov(x,y) = \rho \cdot \sigma^2$
Var(X) = E[X^2]-(E[X])^2
Var(aX+b)=a^2Var[X]
Decision Tree Construction: Start with empty decision tree and complete training set. – Split on the most informative attribute, partitioning dataset. – Recurse on sub partitions. **Decision Tree miscellany:** -Exponentially less data at lower levels – Too big of a tree can overfit the data -**Greedy algorithm don't necessarily yield the global optimum** – Mistakes at top-level propagate down tree.
Decision Tree's advantage over knn: 1. Good with discrete attributes **2.** Easily deals with missing values **3.** Robust to scale of inputs, only depends on ordering (kNN has curse of dimensionality, but D.T. doesn't). **4.** Fast at test time **5.** More interpretable.
kNN's advantage over decision tree: 1. Able to handle attributes/features that interact in complex ways (e.g. pixels)
2. Can incorporate interesting distance measures
Bias-Variance Decomposition:

- Suppose the training set $\mathcal{D}$ consists of N pairs $(\mathbf{x}^{(i)}, t^{(i)})$ sampled **independent and identically distributed (i.i.d.)** from a **sample generating distribution** p_{sample}, i.e., $(\mathbf{x}^{(i)}, t^{(i)}) \sim p_{\text{sample}}$.
  - Let p_{dataset} denote the induced distribution over training sets, i.e. $\mathcal{D} \sim p_{\text{dataset}}$
- Assume (for the moment) that t is deterministic given x!
- There is a distribution over the loss at $\mathbf{x}$, with expectation $\mathbb{E}_{\mathcal{D}\sim p_{\text{dataset}}}[L(h_{\mathcal{D}}(\mathbf{x}), t)]$.
- For each query point $\mathbf{x}$, the expected loss is different. We are interested in quantifying how well our classifier does over the distribution p_{sample}, averaging over training sets: $\mathbb{E}_{\mathbf{x}\sim p_{\text{sample}}, \mathcal{D}\sim p_{\text{dataset}}}[L(h_{\mathcal{D}}(\mathbf{x}), t)]$.

$$\begin{aligned} \mathbb{E}_{\mathbf{x},\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - t)^2] &= \mathbb{E}_{\mathbf{x},\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}] + \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}] - t)^2] \\ &= \mathbb{E}_{\mathbf{x}}[\mathbb{E}_{\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}])^2 + (\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}] - t)^2 + \\ &\qquad 2(h_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}])(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}] - t)\,|\,\mathbf{x}]] \\ &= \underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}])^2]}_{\text{variance}} + \underbrace{\mathbb{E}_{\mathbf{x}}[(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})\,|\,\mathbf{x}] - t)^2]}_{\text{bias}} \end{aligned}$$

$\mathbb{E}_{\mathcal{D}}[2(h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x])(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] - f(x))|x]] = 2(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] - f(x))\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x]|x].$

But $\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x]|x] = 0$. So:

$$\mathbb{E}_{\mathcal{D}}[2(h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x])(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] - f(x))|x] = 0.$$

Bagging: Suppose we could somehow sample $\boldsymbol{m}$ independent **training sets** $\{D_i\}_{i=1}^m$ from $p_{dataset}$. We could then learn a predictor $h_i := h_{D_i}$ based on each one, and take the average of all the predictors. $h = \frac{1}{m}\sum_{i=1}^m h_i$
Bias: Unchanged

$$\mathbb{E}_{\mathcal{D}_1,\ldots,\mathcal{D}_m \overset{iid}{\sim} p_{\text{dataset}}}[h(\mathbf{x})] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}_{\mathcal{D}_i\sim p_{\text{dataset}}}[h_i(\mathbf{x})] = \mathbb{E}_{\mathcal{D}\sim p_{\text{dataset}}}[h_{\mathcal{D}}(\mathbf{x})]$$

Variance: Reduced

$$\operatorname*{Var}_{\mathcal{D}_1,\ldots,\mathcal{D}_m}[h(\mathbf{x})] = \frac{1}{m^2}\sum_{i=1}^m \operatorname*{Var}_{\mathcal{D}_i}[h_i(\mathbf{x})] = \frac{1}{m}\operatorname*{Var}_{\mathcal{D}}[h_{\mathcal{D}}(\mathbf{x})].$$

Problem of bagging: In reality we don't have p_{sample} that can randomly generate datas. Solution: Take a single dataset D with **n** examples. Then generate m new datasets, each by sampling **n** training examples from D, with replacement.

- Problem: the datasets are not independent, so we don't get the $1/m$ variance reduction.
  - Possible to show that if the sampled predictions have variance σ^2 and correlation ρ, then

$$\text{Var}\left(\frac{1}{m}\sum_{i=1}^m h_i(\mathbf{x})\right) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2.$$

$$\begin{aligned} \text{Var}(y_{\text{avg}}) &= \text{Var}\left(\frac{1}{m}\sum_{i=1}^m y_i\right) \\ &= \frac{1}{m^2}\,\text{Var}\left(\sum_{i=1}^m y_i\right) \\ &= \frac{1}{m^2}\sum_{i=1}^m\sum_{j=1}^m \text{Cov}(y_i, y_j) \\ &= \frac{1}{m^2}\left[m\sigma^2 + m(m-1)\rho\sigma^2\right] \\ &= \frac{1}{m}\sigma^2 + \frac{m-1}{m}\rho\sigma^2 \end{aligned}$$

Therefore, we use **random forest. (bagged decision trees, with when choosing each node, choose a rand set of d input features.** It reduces the variance by **reducing the correlation between the trees.** (i.e. between each model)

Bayes Optimality

- Let's return to quantifying expected loss and make the situation slightly more complicated (and realistic): what if t is not deterministic given $\mathbf{x}$? i.e. have $p(t|\mathbf{x})$
- We can no longer measure bias as expected distance from true target, since there's a distribution over targets!
- Instead, we'll measure distance from $y_*(\mathbf{x}) = \mathbb{E}[t\,|\,\mathbf{x}]$
  - This is the best possible prediction, in the sense that it minimizes the expected loss

Want to show: $\text{argmin}_y\, \mathbb{E}[(y-t)^2\,|\,\mathbf{x}] = y_*(\mathbf{x}) = \mathbb{E}[t\,|\,\mathbf{x}]$ (Distribution of $t \sim p(t|\mathbf{x})$)

- **Proof:** Start by conditioning on (fixing) $\mathbf{x}$.

$$\begin{aligned} \mathbb{E}[(y-t)^2\,|\,\mathbf{x}] &= \mathbb{E}[y^2 - 2yt + t^2\,|\,\mathbf{x}] \\ &= y^2 - 2y\mathbb{E}[t\,|\,\mathbf{x}] + \mathbb{E}[t^2\,|\,\mathbf{x}] \\ &= y^2 - 2y\mathbb{E}[t\,|\,\mathbf{x}] + \mathbb{E}[t\,|\,\mathbf{x}]^2 + \text{Var}[t\,|\,\mathbf{x}] \\ &= y^2 - 2yy_*(\mathbf{x}) + y_*(\mathbf{x})^2 + \text{Var}[t\,|\,\mathbf{x}] \\ &= (y - y_*(\mathbf{x}))^2 + \text{Var}[t\,|\,\mathbf{x}] \end{aligned}$$

- The first term is nonnegative, and can be made 0 by setting $y = y_*(\mathbf{x})$.
- The second term doesn't depend on y! Corresponds to the inherent unpredictability, or **noise**, of the targets, and is called the **Bayes error** or **irreducible error**.
  - This is the best we can ever hope to do with any learning algorithm. An algorithm that achieves it is **Bayes optimal**.

Consider the expression inside the expectation. Since given $\mathcal{D}$, $h_{\mathcal{D}}$ is determined we can treat the expression inside the expectation, like 3.1. which gives

$$\mathbb{E}_{\mathcal{D}}[\mathbb{E}_{x,y}[(h_{\mathcal{D}}(x) - y)^2|\mathcal{D}]] = \mathbb{E}_{\mathcal{D}}[\mathbb{E}_x[(h_{\mathcal{D}}(x) - \mathbb{E}_y[y|x])^2|\mathcal{D}] + \mathbb{E}_{x,y}[(\mathbb{E}_y[y|x] - y)^2|\mathcal{D}]]$$

Since $\mathbb{E}_y[y|x] - y$ does not depend on $\mathcal{D}$ and $h_{\mathcal{D}}(x) - \mathbb{E}_y[y|x]$ does not depend on y, we can write:

$$\mathbb{E}_{x,y,\mathcal{D}}[(h_{\mathcal{D}}(x) - y)^2] = \mathbb{E}_{x,\mathcal{D}}[(h_{\mathcal{D}}(x) - \mathbb{E}_y[y|x])^2] + \mathbb{E}_{x,y}[(\mathbb{E}_y[y|x] - y)^2]$$

The last term is Bayes error. In the first term we have $\mathbb{E}_y[y|x]$ and it depends on x deterministically, so just like 2.1 we can decompose it:

$$\mathbb{E}_{x,\mathcal{D}}[(h_{\mathcal{D}}(x) - \mathbb{E}_y[y|x])^2] = \mathbb{E}_{x,\mathcal{D}}[(h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] + \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] - \mathbb{E}_y[y|x])^2]$$

Putting all of these terms together we get:

$$\mathbb{E}_{x,y,\mathcal{D}}[(h_{\mathcal{D}}(x) - y)^2] = \mathbb{E}_{x,\mathcal{D}}[(h_{\mathcal{D}}(x) - \mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x])^2] + \mathbb{E}_{x,\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[h_{\mathcal{D}}(x)|x] - \mathbb{E}_y[y|x])^2] + \mathbb{E}_{x,y}[(\mathbb{E}_y[y|x] - y)^2]$$

Linear Regression: Optimal weights: $\boldsymbol{w^{LS} = (X^T X)^{-1} X^T t}$
Polynomial curve fitting: Fit the data using a degree-M polynomial function of the form $y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^M w_i x^i$

This is called **feature mapping:** $y = w^T\psi(x)$ where $\psi(x) = [1, x, x^2, \dots]^T$. **M to small:** underfit. **M too large:** overfit

Regularization: Why regularization? Because we want to reduce overfitting on complex model. Why complex model? Data might support complex models!

λ penalizes weight value here. $\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2}\sum_j w_j^2$

If you fit training data poorly, J is large. If your optimal weights have high values, R is large.

For least squares problem, we have $\mathcal{J}(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$

$$\mathbf{w}_\lambda^{Ridge} = \underset{\mathbf{w}}{\text{argmin}}\, \mathcal{J}_{\text{reg}}(\mathbf{w}) = \underset{\mathbf{w}}{\text{argmin}}\, \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$=(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$$

Logistic Regression:

Binary classification using linear model:

x_0	x_1	x_2	t
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

$z = w_0x_0 + w_1x_1 + w_2x_2$

need: $w_0 < 0$

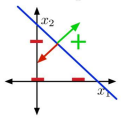
need: $w_0 + w_2 < 0$

need: $w_0 + w_1 < 0$

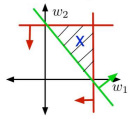
need: $w_0 + w_1 + w_2 > 0$

} AND

Data Space



Weight Space



Visualization

- Slice for $x_0 = 1$ and
- example sol: $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$
- decision boundary:

$w_0x_0 + w_1x_1 + w_2x_2 = 0$

$\implies -1.5 + x_1 + x_2 = 0$

- Slice for $w_0 = -1.5$ for the constraints
- $w_0 < 0$
- $w_0 + w_2 < 0$
- $w_0 + w_1 < 0$
- $w_0 + w_1 + w_2 > 0$

Logistic activation function: How do we represent prediction?

Logistic function is called **sigmoid**.

$z = \boldsymbol{w}^T\boldsymbol{x} + b, \sigma(z) = \frac{1}{1+e^{-z}}, y = \sigma(z)$. *PS*: $\sigma'(z) = \sigma(z)\cdot(1-\sigma(z))$

Loss function for logistic regression:

$L_{CE} = -t\log y - (1-t)\log(1-y)$

Gradient of logistic loss:

$$\frac{\partial\mathcal{L}_{\text{CE}}}{\partial w_j} = \frac{\partial\mathcal{L}_{\text{CE}}}{\partial y}\cdot\frac{\partial y}{\partial z}\cdot\frac{\partial z}{\partial w_j} = \left(-\frac{t}{y} + \frac{1-t}{1-y}\right)\cdot y(1-y)\cdot x_j$$

$$=(y-t)x_j$$

Problem: if t = 1 and I am really confident that it's a negative example. i.e. z << 0. Then log(y) might be numerically 0.

Solution: $L_{LCE}(z,t) = L_{CE}(\sigma(z),t) = tlog(1+e^{-z}) + (1-t)\log(1+e^{-z})$

Neural Networks:

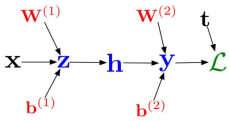
Multiclass classification: Targets form a discrete set {1,...,K}, K classes. Targets are represented as **1-hot** vectors. t=[0,0,..,1,0,..,0]

Prediction for the probability of each class y_k *is between* [0,1]

$$y_k = softmax(z_1,\dots,z_K)_k = \frac{e^{z_{k'}}}{\sum_{k'}e^{z_{k'}}}, where\ z_k is\ called\ logits$$

CE for multiclass classification: $L_{CE}(\boldsymbol{y},\boldsymbol{t}) = -\boldsymbol{t}^T(log\boldsymbol{y})$

NN's diagram, in both scalar and vector form:



Forward pass:

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

$z_i = \sum_j w_{ij}^{(1)}x_j + b_i^{(1)}$

$\mathbf{h} = \sigma(\mathbf{z})$

$h_i = \sigma(z_i)$

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

$y_k = \sum_i w_{ki}^{(2)}h_i + b_k^{(2)}$

$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2 \quad \frac{1}{2}\sum_k (y_k - t_k)^2$

Backward pass:

$\overline{\mathcal{L}} = 1$	$\overline{\mathcal{L}} = 1$
$\overline{y_k} = \overline{\mathcal{L}}(y_k - t_k)$	$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$
$\overline{w_{ki}^{(2)}} = \overline{y_k}h_i$	$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$
$\overline{b_k^{(2)}} = \overline{y_k}$	$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$
$\overline{h_i} = \sum_k \overline{y_k}w_{ki}^{(2)}$	$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$
$\overline{z_i} = \overline{h_i}\,\sigma'(z_i)$	$\overline{\mathbf{z}} = \overline{\mathbf{h}}\circ\sigma'(\mathbf{z})$
$\overline{w_{ij}^{(1)}} = \overline{z_i}x_j$	$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$
$\overline{b_i^{(1)}} = \overline{z_i}$	$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$

GD Rules: let g be gradient. Then

GD: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\sum_{i=1}^N g_i$

SGD: Choose $i \sim U[1,N]$, $\boldsymbol{w} \leftarrow \eta g_i$

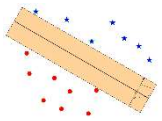
Mini-batch GD: Choose a subset $M \subset \{1,\dots,N\}$, $\boldsymbol{w} \leftarrow \eta\sum_{i\in M}^{|M|} g_i$

Computation cost: SGD < mSGD < GD

Convergence rate: SGD < mSGD < GD

SGD's mathematical justification: if you sample a training example **uniformly at random**, the stochastic gradient is an **unbiased estimate** of the batch gradient. $E\left[\frac{\partial L^{(i)}}{\partial\boldsymbol{\theta}}\right] = \frac{1}{N}\sum_{i=1}^N\frac{\partial L^{(i)}}{\partial\boldsymbol{\theta}} = \frac{\partial J}{\partial\boldsymbol{\theta}}$

SVM:



Goal: Minimize the hinge loss that leads to an optimal separating hyperplane.

What is the optimal separating hyperplane? A **hyperplane** is described by points $\boldsymbol{x} \in \mathbb{R}^D$ s.t. $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0$.

We want to find the hyperplane that minimize the training loss:

$\min_{\boldsymbol{w},b}\sum_{i=1}^N \max\{0, 1 - t^{(i)}z^{(i)}(\boldsymbol{w},b)\}$ (minimizing the hinge loss! $\max\{0, 1 - t^{(i)}z^{(i)}(\boldsymbol{w},b)\}$ is the hinge loss). Often add $\frac{\lambda}{2}||\boldsymbol{w}||^2$ in the end (regularizer). **Decision boundary** is linear. Can only do binary classification.

To find **w** and b: Find all the three points that are support vectors. Then, substitute their coordinates to $\boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{b}$. The right side equals whether it is a positive example (1) or negative example (-1). E.g. $1w_1 + 4w_2 + b = 1, 2w_1 + 3w_2 + b = 1, 4w_1 + 5w_2 + b = -1$, Then solve for this system of linear equation! You will get **w** and b.

What is the benefit of the hinge loss compared to 0-1 loss? Minimum of a function will be at its critical points. But the gradient of the 0-1 loss is zero everywhere it's defines. This means that changing the weights by a very small amount probably has no effect on the loss. We can use the 0-1 loss function, but minimizing this loss is computationally difficult, and it cannot distinguish different hypotheses that achieve the same accuracy. The hinge loss is a relaxation of the 0-1 loss, it can be minimized using gradient descent.

Claim: hinge loss = 0 iff all points are correctly classified.

Boosting:

AdaBoost Algorithm:

- Input: Data $\mathcal{D}_N$, weak classifier WeakLearn (a classification procedure that returns a classifier h, e.g. best decision stump, from a set of classifiers $\mathcal{H}$, e.g. all possible decision stumps), number of iterations T
- Output: Classifier $H(x)$
- Initialize sample weights: $w^{(n)} = \frac{1}{N}$ for $n = 1,\dots,N$
- For $t = 1,\dots,T$
  - Fit a classifier to data using weighted samples $(h_t \leftarrow \text{WeakLearn}(\mathcal{D}_N, \mathbf{w}))$, e.g.,

$$h_t \leftarrow \underset{h\in\mathcal{H}}{\text{argmin}}\sum_{n=1}^N w^{(n)}\mathbb{I}\{h(\mathbf{x}^{(n)}) \neq t^{(n)}\}$$

  - Compute weighted error $\text{err}_t = \frac{\sum_{n=1}^N w^{(n)}\mathbb{I}\{h_t(\mathbf{x}^{(n)})\neq t^{(n)}\}}{\sum_{n=1}^N w^{(n)}}$
  - Compute classifier coefficient $\alpha_t = \frac{1}{2}\log\frac{1-\text{err}_t}{\text{err}_t}$ $(\in(0,\infty))$
  - Update data weights

$$w^{(n)} \leftarrow w^{(n)}\exp\left(-\alpha_t t^{(n)}h_t(\mathbf{x}^{(n)})\right)\left[\equiv w^{(n)}\exp\left(2\alpha_t\mathbb{I}\{h_t(\mathbf{x}^{(n)}) \neq t^{(n)}\}\right)\right]$$

- Return $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(\mathbf{x})\right)$

Weak classifiers which get lower weighted error (err_t) get more weight in the final classifier. (Since err_t low => α_t high, so when computing H(x) it gets more weight)

Also, when updating weights: $w^{(n)} \leftarrow w^{(n)}\exp(2\alpha_t I\{h_t(\boldsymbol{x}^{(\boldsymbol{n})} \neq t^{(n)}\})$

If $err_t \approx 0$, a_t high so misclassified examples get more attention

If $err_t \approx 0.5$, a_t low so misclassified examples get less attention

(i.e. stronger classifiers misclassified => add more weight!)

AdaBoost Minimizes the training error.

AdaBoost penalizes more regarding error compared to logistic regression.

Theorem: Assume that at each iteration of AdaBoost the WeakLearn returns a hypothesis with error $err_t \leq \frac{1}{2} - \gamma$ for all t = 1,..,T with $\gamma > 0$. The training error of the output hypothesis $H(\boldsymbol{x}) = sign(\sum_{t=1}^T a_t h_t(\boldsymbol{x}))$ is at most $L_N(H) = \frac{1}{N}\sum_{i=1}^N I\{H(\boldsymbol{x}^{(i)} \neq t^{(i)})\} \leq \exp(-2\gamma^2 T)$. This is under the simplifying assumption that each weak learner is γ-better than a random predictor. It is called a geometric convergence. -> Fast! **AdaBoost is an additive model.**

Q: How does Adaboost reduce bias? By making each classifier focus on previous mistakes **Q: Sometimes the test error decreases even after the training error is zero. How does that happen?** The confidence of the classifier is not just the training error. After the training error is zero, the confidence continues to increase as each example goes further away from the decision boundary. The farther an example is, the more confident the model is in its label, or another way of thinking about it, the decision is more robust to noise in the input features.

Additive models: We obtain the additive model $H_m(x) = \sum_{i=1}^m a_i h_i(\boldsymbol{x})$ with

$$h_m \leftarrow \underset{h\in\mathcal{H}}{\text{argmin}}\sum_{i=1}^N w_i^{(m)}\mathbb{I}\{h(\mathbf{x}^{(i)}) \neq t^{(i)}\},\quad \alpha = \frac{1}{2}\log\left(\frac{1-\text{err}_m}{\text{err}_m}\right),$$

$$\text{where } \text{err}_m = \frac{\sum_{i=1}^N w_i^{(m)}\mathbb{I}\{h_m(\mathbf{x}^{(i)}) \neq t^{(i)}\}}{\sum_{i=1}^N w_i^{(m)}},\ w_i^{(m+1)} = w_i^{(m)}\exp\left(-\alpha_m h_m(\mathbf{x}^{(i)})t^{(i)}\right).$$

Bagging vs Boosting:

Bagging: Train classifiers independently on random subsamples of training data. 1. reduces variance 2. bias is not changed much 3. parallel 4. want to minimize

correlation between ensemble elements **Boosting:** Train classifier sequentially, each time focusing on training data points that were previously misclassified. 1. reduces bias 2. increases variance 3.sequential 4. high dependency between ensemble elements

Generative model vs Discriminative model:

Discriminative approach: estimate parameters of decision boundary/class separator directly from labeled examples. **1.** Tries to solve: How do I separate the classes? **2.** Learn p(t|**x**) directly (logistic regression models) **3.** Learn mappings from inputs to classes (linear/logistic regression, decision tress etc)

Generative approach: model the distribution of inputs characteristic of the class (Bayes classifier) **1.** Tries to solve: What does each class "look" like? **2.** Build a model of $p(\boldsymbol{x}|t)$ **3.** Apply Bayes rule i.e. p(t, **x**) = …

Key difference: Is there a distributional assumption over inputs?

Bayes Classifier: Given features $\boldsymbol{x} = [x_1, x_2, \dots, x_D]^T$ we want to compute class probabilities using Bayes Rule:

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

More formally:

$$posterior = \frac{Class\ likelihood \times prior}{Evidence}$$

To compute $p(c|\boldsymbol{x})$ we need: $p(\boldsymbol{x}|c)$ and $p(c)$

Naïve Bayes assumption: Naïve Bayes assumes that the world features x_i are **conditionally independent** given the class c.

$$p(c, x_1, \dots, x_D) = p(c)p(x_1|c) \dots p(x_D|c)$$

The log-likelihood can be decomposed into:

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p\big(c^{(i)}, \boldsymbol{x}^{(i)}\big) = \sum_{i=1}^{N} \log p\big(c^{(i)}\big) + \sum_{j=1}^{D}\sum_{i=1}^{N} \log p\big(x_j^{(i)}|c^{(i)}\big)$$

To do learning: (during training) Find MLE for $\pi = p\big(c^{(i)} = 1\big)$ and $\theta = p(x_j^{(i)} = 1|c)$ (Since good values of θ should assign high probability to the observed data.)

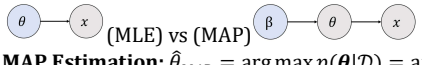
For the prior we maximize: $\sum_{i=1}^{N} \log p\big(c^{(i)}\big)$. Then $p\big(c^{(i)}\big) = \pi^{c^{(i)}}\left(1-\pi^{1-c^{(i)}}\right)$

For the class likelihood θ we maximize: $\sum_{i=1}^{N} \log p\big(x_j^{(i)}|c^{(i)}\big)$. Then $p(x_j^{(i)}|c = \theta_{jc}^{x_j^{(i)}}\big(1-\theta_{jc}\big)^{1-x_j^{(i)}})$

To do prediction: (during test time) We predict the category by performing inference in the model. Apply **Bayes' Rule:**

$$p(c|\boldsymbol{x}) = \frac{p(c)p(\boldsymbol{x}|c)}{\sum_{c'} p(c')p(\boldsymbol{x}|c')} = \frac{p(c)\prod_{j=1}^{p} p(x_j|c)}{\sum_{c'} p(c')\prod_{j=1}^{p} p(x_j|c')} \propto p(c)\Pi_{j=1}^{p} p(x_j|c)$$

MLE's disadvantage: Not good with sparse data! e.g., What if you flip the coin **only twice** (so data is sparse) and get H both times? Then $\theta_{ML} = \frac{N_H}{N_H+N_T} = \frac{2}{2+0} = 1$, i.e. the probability of getting head is **1**, according the model's belief! Therefore, it assigned getting tail's outcome probability 0. : (Solution: Add prior distribution to theta!



(MLE) vs (MAP)

MAP Estimation: $\hat{\theta}_{MAP} = \arg\max_{\theta} p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\theta} p(\boldsymbol{\theta}, \mathcal{D}) =$
$\arg\max_{\theta} p(\boldsymbol{\theta})p(\mathcal{D}|\boldsymbol{\theta}) = \arg\max_{\theta} logp(\boldsymbol{\theta}) + \log p(\mathcal{D}|\boldsymbol{\theta})$

Gaussian Discriminative Analysis (GDA):

Assumes that $p(\boldsymbol{x}|t)$ is distributed according to multivariate Gaussian distribution.

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

Where $|\Sigma_k|$ denotes the determinant of the matrix, and D is dimension of **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 **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.

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

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

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

Quadratic function in **x** ⇒ **quadratic decision boundary!**

What is $C_{k,l}$? What if $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_l$? **-> can be derived. Easy.**

Simplifying GDA model: What if **x** is high-dimensional?

For Gaussian Bayes Classifier, if input **x** is high-dimensional, then covariance matrix has many parameters $O(D^2)$. We can save some parameters by using a shared covariance for the classes, i.e. $\boldsymbol{\Sigma}k = \boldsymbol{\Sigma}l$. **If so, we have a linear decision boundary!**

Comparing Naïve Bayes vs GDA:

Both Naïve Bayes and GDA are generative models. But:

Naïve Bayes	GDA
Works on a set of discrete inputs	Works on continuous inputs
Assume conditional independence given class, so p(x1,x2|c)=p(x1|c)p(x2|c). (i.e. Covariance is **0**!) Objective is to figure out, for each x_i and each c, the $p(x_i|c)$ that maximizes the likelihood of the observed data	Assume the underlying data is generated from Gaussian distribution. This let us specify covariance through covariance matrices, instead of being 0. Also, see the following sentence:

GDA's objective is then to figure out the Gaussian parameters (mean and covariance matrix) that provide the best explanation (highest likelihood) for the observed data.

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.

PCA:

PCA is a linear model. Goal: Dimensionality reduction -> saves computation!

Need to choose $D \times K$ matrix $\boldsymbol{U}$ **with orthonormal columns.**

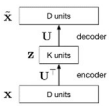
Two criteria: **1.** Minimize the **reconstruction error:** Find vectors in a subspace that are closest to data points. $\min_{U} \frac{1}{N}\sum_{i=1}^{N}\left|\left|\boldsymbol{x}^{(i)} - \tilde{\boldsymbol{x}}^{(i)}\right|\right|^2$ **2.** Maximize the variance of reconstructions: Find a subspace where data has the most variability.

$\max_{U} \frac{1}{N}\sum_{i}^{N}\left|\left|\tilde{\boldsymbol{x}}^{(i)} - \hat{\boldsymbol{\mu}}\right|\right|^2$. **Proof the data and its reconstruction has the same means:** _____

To find the best $\tilde{\boldsymbol{x}}$ (I.e. to find the best projection):

Let **U** be a matrix with columns $\{\boldsymbol{u}_k\}_{k=1}^{K}$ then $\boldsymbol{z} = \boldsymbol{U}^T(\boldsymbol{x} - \hat{\boldsymbol{\mu}})$.

Also, $\tilde{\boldsymbol{x}} = \hat{\boldsymbol{\mu}} + \boldsymbol{U}\boldsymbol{z} = \hat{\boldsymbol{\mu}} + \boldsymbol{U}\boldsymbol{U}^T(\boldsymbol{x} - \hat{\boldsymbol{\mu}})$. Here, $\boldsymbol{U}\boldsymbol{U}^T$ is the projector on subspace, and $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ **(**$\boldsymbol{U}\boldsymbol{U}^T \neq \boldsymbol{I}$**!!!)**



Proof that dimensions of z are decorrelated: $Cov(z) = Cov\big(\boldsymbol{U}^T(\boldsymbol{x}-\boldsymbol{\mu})\big) = \boldsymbol{U}^T Cov(\boldsymbol{x})\boldsymbol{U} = \boldsymbol{U}^T\boldsymbol{\Sigma}\boldsymbol{U} = \boldsymbol{U}^T\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{U} =$ $(\boldsymbol{I}\quad \boldsymbol{0})\begin{pmatrix}\boldsymbol{I}\\ \boldsymbol{0}\end{pmatrix} = top\ left\ K \times K$ block of $\boldsymbol{\Lambda}$. Since the covariance matrix is diagonal, this means the features are uncorrelated. **PCA can be viewed as a linear autoencoder.** (as shown on the left). If K < D, then **W1** maps **x** to a K-dimensional space, so it's doing dimensionality reduction.

Unsupervised Learning:

K-means algorithm: Objective: Find cluster centers $\{\boldsymbol{m}_k\}_{k=1}^{K}$ and assignments $\{r^{(n)}\}_{n=1}^{N}$ to minimize the sum of squared distances of data points $\{\boldsymbol{x^{(n)}}\}$ to their assigned cluster centers. **1.** Data sample $n = 1, \dots, N$: $\boldsymbol{x}^{(n)} \in \mathbb{R}^D$ (observed) **2.** Cluster center $k = 1, \dots, K$: $\boldsymbol{m}_k \in \mathbb{R}^D$ (not observed) **3.** Responsibilities: Cluster assignment for sample n: $\boldsymbol{r}^{(n)} \in \mathbb{R}^K$ 1-of-K encoding (not observed)

Mathematically, we want:

$$\min_{\{\boldsymbol{m_k}\},\{\boldsymbol{r}^{(n)}\}} J\big(\{\boldsymbol{m}_k\},\{\boldsymbol{r}^{(n)}\}\big) = \min_{\{\boldsymbol{m_k}\},\{\boldsymbol{r}^{(n)}\}} \sum_{n=1}^{N}\sum_{k=1}^{K}\left|\left|\boldsymbol{m}_k - \boldsymbol{x}^{(n)}\right|\right|^2$$

The algorithm: Initialization: Set K cluster means $\boldsymbol{m_1}, \dots, \boldsymbol{m_K}$ to random values.

Repeat the following until convergence (i.e. until assignments do not change):

Assignment: Optimize J w.r.t. $\{\boldsymbol{r}\}$: Each data point $\boldsymbol{x}^{(n)}$ assigned to nearest center

$$\hat{k}^{(n)} = \arg\min_{k}\left|\left|\boldsymbol{m}_k - \boldsymbol{x}^{(n)}\right|\right|^2$$

And **Responsibilities** (1 hot, if not using soft K-means):

$$r_k^{(n)} = I(\hat{k}^{(n)} = k)\ \text{for}\ k = 1, \dots, K$$

Refitting: Optimize J w.r.t. $\{\boldsymbol{m}\}$: Each center is set to mean of data assigned to it.

$$\boldsymbol{m}_k = \frac{\sum_n r_k^{(n)}\boldsymbol{x}^{(n)}}{\sum_n r_k^{(n)}}$$

K-means algorithm reduces the cost **at each iteration.**

Soft K-means: different assignment step.

$$r_k^{(n)} = \frac{\exp[-\beta\left|\left|\boldsymbol{m}_j - \boldsymbol{x}^{(n)}\right|\right|^2]}{\sum_j \exp[-\beta\left|\left|\boldsymbol{m}_j - \boldsymbol{x}^{(n)}\right|\right|^2]} \Rightarrow \boldsymbol{r}^{(n)} = \text{softmax}(-\beta\left\{\left|\left|\boldsymbol{m}_k - \boldsymbol{x}^{(n)}\right|\right|^2\right\}_{k=1}^{K})$$

$\beta \to \infty \Rightarrow$ Soft K-means becomes K-means. **Proof:** when beta is approaching infinity, the assignment step will be much more sensitive to the distance between each point and centers. Thus, if any center is slightly closer to the data point, it will immediately become the choice.

EM Algorithm:

Initialize the means $\hat{\boldsymbol{\mu}}_k$ and mixing coefficients $\hat{\pi}_k$

Iterate until convergence:

E-step: Evaluate the responsibilities $r_k^{(n)}$ given current parameters.

$$r_k^{(n)} = p\big(z^{(n)} = k|\boldsymbol{x}^{(n)}\big) = \frac{\hat{\pi}_k\mathcal{N}(\boldsymbol{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \boldsymbol{I})}{\sum_{j=1}^{K}\hat{\pi}_j\mathcal{N}(\boldsymbol{x}^{(n)}|\hat{\boldsymbol{\mu}}_j, \boldsymbol{I})} = \frac{\hat{\pi}_k\exp\{-\frac{1}{2}\left|\left|\boldsymbol{x}^{(n)} - \hat{\boldsymbol{\mu}}_k\right|\right|^2\}}{\sum_{j=1}^{K}\hat{\pi}_j\exp\{-\frac{1}{2}\left|\left|\boldsymbol{x}^{(n)} - \hat{\boldsymbol{\mu}}_j\right|\right|^2\}}$$

M-step: Re-estimate the parameters given the current responsibilities.

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_k^{(n)}\boldsymbol{x}^{(n)}, \hat{\pi}_k = \frac{N_k}{N}\ \text{with}\ N_k = \sum_{n=1}^{N} r_k^{(n)}$$

Evaluate the log likelihood and check for convergence.

$$\log p(D) = \sum_{n=1}^{N}\log(\sum_{k=1}^{K}\hat{\pi}_k\,\mathcal{N}(\boldsymbol{x}^{(n)}|\hat{\boldsymbol{\mu}}_k, \boldsymbol{I}))$$ -> When this converges, stop!

What happened in the EM algorithm: 1. The maximum likelihood objective $\sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)})$ was hard to optimize. **2.** However, the likelihood for $D_{complete}$ is easy to compute. $\log(D_{complete}) = \sum_{n=1}^{N}\sum_{k=1}^{K} I[z^{(n)}=k](\log \mathcal{N}(\boldsymbol{x}^{(n)}|\boldsymbol{\mu}_k, \boldsymbol{I}) + \log \pi_k)$. **3.** We don't know about $z^{(n)}$, so we replaced $I[z^{(n)}=k]$ with resibilties $r_k^{(n)} = p(z^{(n)}=k|\boldsymbol{x}^{(n)})$. That is, we replaced $I[z^{(n)}=k]$ with its **expectation** under $p(z^{(n)}|\boldsymbol{x}^{(n)})$. Then, this is now easy to optimize, we can get

$\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k}\sum_{n=1}^{N} r_k^{(n)}\boldsymbol{x}^{(n)}$, $\hat{\pi}_k = \frac{N_k}{N}$ with $N_k = \sum_{n=1}^{N} r_k^{(n)}$ (E-step)

Both k-means and EM suffer from getting stuck at local minima.

Claim: For EM algorithm with shared covariance $\frac{1}{\beta}\boldsymbol{I}$, it becomes soft k-means algorithm. **Proof:** _____

K-means vs EM:

K-means	EM
Assignment Step: Assign each data point to the closest cluster. Refitting Step: Move each luster center to the average of the data assigned to it.	E-step: Compute the posterior probability over z given our current model. ($r_k^{(n)} = p(z^{(n)}=k|\boldsymbol{x}^{(n)})$) M-step: Maximize the probability that it would generate the data it is currently responsible for.

Matrix Factorization:

Can view the following as Matrix Factorization.

PCA: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$**.** SVD of the data matrix $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{Z}\boldsymbol{U}^T$, with $\boldsymbol{Z} = \boldsymbol{Q}\boldsymbol{S}$. The eigen-decomposition of $\hat{\boldsymbol{\Sigma}}$ follows directly from the eigen-decomposition of $\boldsymbol{X}^T\boldsymbol{X}$: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X} = \frac{1}{N}\boldsymbol{U}\boldsymbol{S}\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{S}\boldsymbol{U}^T = \boldsymbol{U}\left[\frac{\boldsymbol{S}^2}{N}\right]\boldsymbol{U}^T$. The SVD gives **U**, which is equivalent to the learned basis of PCA. First K principal components corresponds first K columns of **U**. Ultimately, PCA with K principal components finds the *optimal* rank-K approximation of $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, in terms of error $\left||\boldsymbol{X}^T - \boldsymbol{U}\boldsymbol{Z}^T|\right|_F^2$.

$\min\left||\boldsymbol{X}^T - \boldsymbol{U}\boldsymbol{Z}^T|\right|_F^2$ *over* $\boldsymbol{Z} \in \mathbb{R}^{N\times K}, \boldsymbol{U} \in \mathbb{R}^{D\times K}$. Note that the case K = D corresponds to the entire SVD of **X**.

Recommender Systems:

Sparse ratings matrix: **R,** Let the representation of user n be $\boldsymbol{u_n}$ the representation of movie $\boldsymbol{m}$ be $\boldsymbol{z}_m$. n = 1,..,K. We want to enforce $\boldsymbol{R} \approx \boldsymbol{U}\boldsymbol{Z}^T$**.** It is impossible to do $\min_{\boldsymbol{U},\boldsymbol{Z}} \sum_{i,j}\left(R_{ij} - \boldsymbol{u}_i^T\boldsymbol{z}_j\right)^2$, since most data entries R_{ij} are **missing!**

Solution to missing data: Let $O = \{(n,m)$: every (n,m) of matrix **R** is observed$\}$ Then we use square error loss to solve: $\min_{\boldsymbol{U},\boldsymbol{Z}} \sum_{(n,m)\in O}(R_{nm} - \boldsymbol{u}_n^T\boldsymbol{z}_m)^2$

Problem: this equation is very hard to minimize. It is non-convex. Solution is to use alternating least squares, A.k.a. ALS: fix **Z** and optimize **U,** then followed by fix **U** and optimize **Z**, and so on until convergence.

ALS for Matrix Completion algorithm
1. Initialize **U** and **Z** randomly
2. repeat until convergence
3. for $n = 1, .., N$ **do**
4. $\mathbf{u}_n = \left(\sum_{m:(n,m)\in O} \mathbf{z}_m\mathbf{z}_m^\top\right)^{-1}\sum_{m:(n,m)\in O} R_{nm}\mathbf{z}_m$
5. for $m = 1, .., M$ **do**
6. $\mathbf{z}_m = \left(\sum_{n:(n,m)\in O} \mathbf{u}_n\mathbf{u}_n^\top\right)^{-1}\sum_{n:(n,m)\in O} R_{nm}\mathbf{u}_n$

K-means: Can view K-means as a matrix factorization too. **1.** Stack 1-of K vectors $\boldsymbol{r_i}$ for assignments into a $N \times K$ matrix **R**, and stack the cluster centers $\boldsymbol{m_k}$ **into a matrix** $K \times D$ matrix **M**. We want to let $\boldsymbol{X} \approx \boldsymbol{R}\boldsymbol{M}$, through reconstruction!

K-means distortion function in matrix form: $\sum_{n=1}^{N}\sum_{k=1}^{K} r_k^{(n)}\left||\boldsymbol{m}_k - \boldsymbol{x}^{(n)}|\right|^2 = \left||\boldsymbol{X} - \boldsymbol{R}\boldsymbol{M}|\right|_F^2$. We can do co-clustering so that **X** will give a block structure.

Sparse Coding: $\boldsymbol{x} \approx \sum_{k=1}^{K} s_k\boldsymbol{a_k} = \boldsymbol{A}\boldsymbol{s}$**,**where **s** is the sparse vector. **1.** We would like to choose **s** to accurately reconstruct the image, $\boldsymbol{x} \approx \boldsymbol{A}\boldsymbol{s}$ but encourage sparsity in **s**.

2. We want to minimize the following function: $\min_{\boldsymbol{s}}\left||\boldsymbol{x} - \boldsymbol{A}\boldsymbol{s}|\right|^2 - \beta||\boldsymbol{s}||_1$

Here, β is a hyperparameter that trades off reconstruction error vs sparsity.

We can learn a dictionary by optimizing both **A** (aka the dictionary) and $\{\boldsymbol{s}_i\}_{i=1}^N$ to trade off reconstruction error and sparsity

$$\min_{\{\boldsymbol{s}_i\},\boldsymbol{A}} \sum_{i=1}^{N}\left||\boldsymbol{x}^{(i)} - \boldsymbol{A}\boldsymbol{s}_i|\right|^2 + \beta\left||\boldsymbol{s}_i|\right|_1, \text{subject to } \left||\boldsymbol{a}_k|\right|^2 \le 1 \text{ for all } k$$

Can fit using ALS over A and S, just like anything mentioned before.

Reinforcement Learning:

Goal: Find a policy π that maximizes the value function.

Optimal value function:$Q^*(s,a) = \sup_{\pi} Q^\pi(s,a)$

Given Q^*, the optimal policy can be obtained as $\pi^*(s) = \arg\max_a Q^*(s,a)$

The goal of an RL agent is to find a policy π that is close to optimal, i.e., $Q^\pi \approx Q^*$.

The value function satisfy the following recursive relationship:

$$Q^\pi(s,a) = r(s,a) + \gamma\int_S Q^\pi(s',\pi(s'))P(s'|s,a)\,\mathrm{d}s' := (T^\pi Q^\pi)(s,a)$$

$$\left(T^\theta Q\right)(s,a) := r(s,a) + \gamma\int_S Q(s',\theta(s'))P(s'|s,a)\,\mathrm{d}s'$$

T^π is called the **Bellman operator.** We define the bellman optimality operator T^*:

$$(T^*Q)(s,a) := r(s,a) + \gamma\int_S \max_{a'\in A} Q(s',a')\,P(s'|s,a)\,\mathrm{d}s'$$

Claim: $Q^\pi = T^\pi Q^\pi$, $Q^* = T^*Q^*$. The solutions of these fixed-pt equations are unique.

Value-based approaches try to find a $\hat{Q}$ such that $\hat{Q} \approx T^*\hat{Q}$ (thus $\hat{Q} \approx Q^*$) Then, we apply greedy policy of $\hat{Q}$ in practice.

Greedy policy of $\hat{\boldsymbol{Q}}$: $\pi(s;\hat{Q}) := \arg\max_{a\in A}\hat{Q}(s,a)$

Finding the optimal value function using Value Iteration:

Assume we know the model P and R, and the state-action space $S \times A$ is small, Then we can use the bellman optimality equation. We start from an initial function Q_1. For each $k = 1,2, ...,$ apply $Q_{k+1} \leftarrow T^*Q_k$.

$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma\int_S \max_{a'\in A} Q_k(s',a')\,P(s'|s,a)\,\mathrm{d}s'$ (continuous)

$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma\sum_{s'\in S} \max_{a'\in A} Q_k(s',a')\,P(s'|s,a)$ (discrete)

Claim: The VI converges to the optimal value function. This is because of the contraction property of the Bellman (optimality) operator.

i.e., $\left||T^*Q_1 - T^*Q_2|\right|_\infty \le \gamma\left||Q_1 - Q_2|\right|_\infty$.

Problem for VI: **1.** when state-action space is large, you cannot integrate exactly. **2.** We often do not know the dynamics P and the reward function R, so we cannot calculate the Bellman operators. Solution → Batch RL!

Batch RL: Suppose we are **only** given the following dataset.

1. A batch, i.d. $D_N = \{(S_i, A_i, R_i, S_i')\}_{i=1}^N$
2. $(S_i, A_i) \sim v$ (v is a distribution over $S \times A$)
3. $S_i' \sim P(\cdot\,|S_i, A_i)$; **4.** $R_i \sim R(\cdot\,|S_i, A_i)$

We can estimate $Q \approx Q^*$ using the data given.

Claim: $t_i = R_i + \gamma\max_{a'\in A} Q(S_i', a')$ is a noisy version of $(T^*Q)(S_i, A_i)$.

Proof: $E[t_i|S_i, A_i] = E\left[R_i + \gamma\max_{a'\in A} Q(S_i', a')|S_i, A_i\right] = r(S_i, A_i) + \cdots = (T^*Q)(S_i, A_i)$

The Algorithm of Batch RL: Given the dataset D_N and action value function estimate Q_k, we firstly construct the dataset $\left\{\boldsymbol{x}^{(i)}, t^{(i)}\right\}_{i=1}^N$ with $\boldsymbol{x}^{(i)} = (S_i, A_i)$ and $t^{(i)} = R_i + \gamma\max_{a'\in A} Q(S_i', a')$. By the proof above we can treat the problem of estimating Q_{k+1} as a regression problem with noisy data. We solve a regression problem. We minimize the squared error.

$$Q_{k+1} \leftarrow \arg\min_{Q\in F}\frac{1}{N}\sum_{i=1}^{N}\left|Q(S_i, A_i) - \left(R_i + \gamma\max_{a'\in A} Q_k(S_i', a)\right)\right|^2 =: \tilde{Q}_k$$

The policy of the agent is selected to be the **greedy** policy w.r.t. the final estimate of the value function: At state $s \in \mathcal{S}$, the agent chooses $\pi(s; Q_K) \leftarrow \arg\max_{a\in\mathcal{A}} Q_K(s,a)$

This method is called **Approximate Value Iteration (AVI).**

Online Learning: Motivation: RL problems are often interactive: The agent interacts with the environment and updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.

Q-Learning with epsilon-greedy policy:

- Parameters:
  - Learning rate: $0 < \alpha < 1$: learning rate
  - Exploration parameter: ε
- Initialize $Q(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$
- The agent starts at state S_0.
- For time step $t = 0, 1, ...,$
  - Choose A_t according to the ε**-greedy policy**, i.e.,

$$A_t \leftarrow \begin{cases}\text{argmax}_{a\in\mathcal{A}} Q(S_t,a) & \text{with probability } 1-\varepsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \varepsilon\end{cases}$$

  - Take action A_t in the environment.
  - The state of the agent changes from S_t to $S_{t+1} \sim \mathcal{P}(\cdot|S_t, A_t)$
  - Observe S_{t+1} and R_t
  - Update the action-value function at state-action (S_t, A_t):

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_t + \gamma\max_{a'\in\mathcal{A}} Q(S_{t+1},a') - Q(S_t,A_t)\right]$$

RL miscellaneous:

Q: What property does Transition probability follow? **S:** Markov property: the future depends on the past only through the current state

Q: Why do we need the discount factor? (Why cannot we always use the cumulative rewards, i.e. γ = 1?) **S:** The cumulative reward is problematic for continuing tasks because the final time step would be T = ∞, and the return could easily be infinite.

Q: VI vs AVI: S: VI: Finding the optimal policy / value function when P and R known (planning problem) AVI: We are given dataset of size N, we can approximately perform VI using these data by regression, minimizing MSE.

Q: What is the **difference between Batch RL and online RL?** S: In batch RL, we have a batch of data coming from the previous interaction of the agent with the environment. This allowed us to use tools from the supervised learning literature. But RL problems are often interactive: the agent continually interacts with the environment, updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.

Q: What is the difference between **exploration** and **exploitation** and how are they related in the Q-Learning algorithm? S: The RL agent should collect as much information about the environment as possible (exploration), while benefiting from the knowledge that has been gathered so far in order to obtain a lot of rewards (exploitation). In Q-Learning, the -greedy policy ensures that most of the time (probability 1 -) the agent exploits its incomplete knowledge of the world by choosing the best action, but occasionally (probability) it explores other actions.