Final Review

CS466, Sp 2021

Final Exam

- Time: Monday, May 10, from 1:30 to 4:30 PM
- Superset of the exam questions posted (Piazza, Course Website)
  - No guidance/solution available
  - Your cheatsheet should not contain anything specific to the questions posted
- Materials: everything after the midterm
- Logistics soon

Side note:

- HW4, HW5 due on May 15
  - HW5 has an autograder; manual grading available. The autograder is strict, but the grading will be lenient

Outline of Session

- Materials Review (30~40 minutes)
- Q&A

Materials

- HMMs
  - Forward and Viterbi algorithm
- Machine Learning
  - Classification: (k-nearest neighbor, linear classifier)
  - SVM
  - Non-linear separability - feature maps
  - Clustering: k-means, hierarchical clustering
  - Linear regression
- Genome assembly
  - de Bruijn graphs and overlap graphs

HMMs

HMM

- Forward algorithm: likelihood computation
- Viterbi algorithm: decoding

Computing Likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

HMM

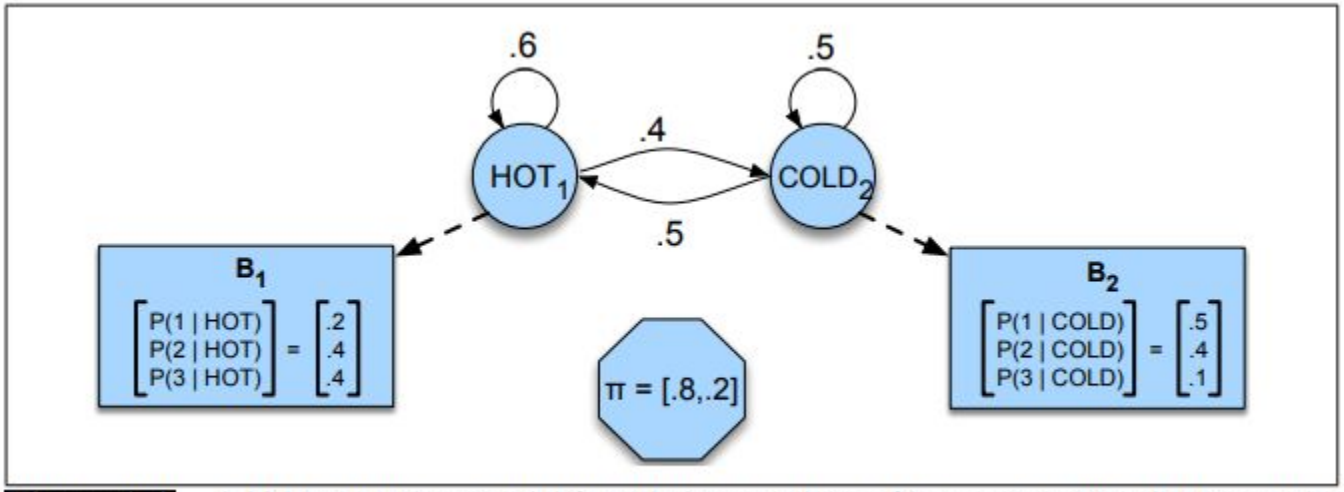


Figure A.2 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

- Probability of observing 3 1 3 -> forward algorithm
- Most probable state sequence when observing 3 1 3? -> Viterbi

HMM: Forward Algorithm

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q)$$

$$P(3\ 1\ 3) = P(3\ 1\ 3, \text{cold cold cold}) + P(3\ 1\ 3, \text{cold cold hot}) + P(3\ 1\ 3, \text{hot hot cold}) + \ldots$$

DP Algorithm

$\alpha_{t-1}(i)$	the **previous forward path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j

notations

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

HMM

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \le j \le N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

HMM: Viterbi Algorithm

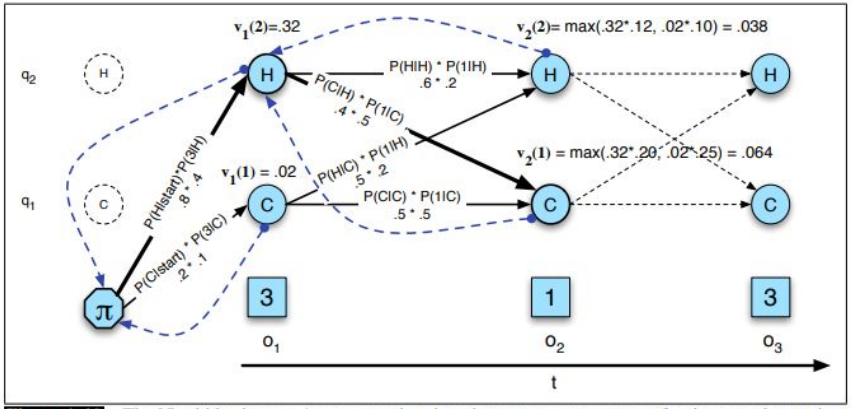


Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

1. **Initialization:**

$$v_1(j) = \pi_j b_j(o_1) \qquad 1 \le j \le N$$
$$bt_1(j) = 0 \qquad 1 \le j \le N$$

2. **Recursion**

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$
$$bt_t(j) = \operatorname*{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. **Termination:**

$$\text{The best score:} \quad P* = \max_{i=1}^{N} v_T(i)$$
$$\text{The start of backtrace:} \quad q_T* = \operatorname*{argmax}_{i=1}^{N} v_T(i)$$

Machine Learning

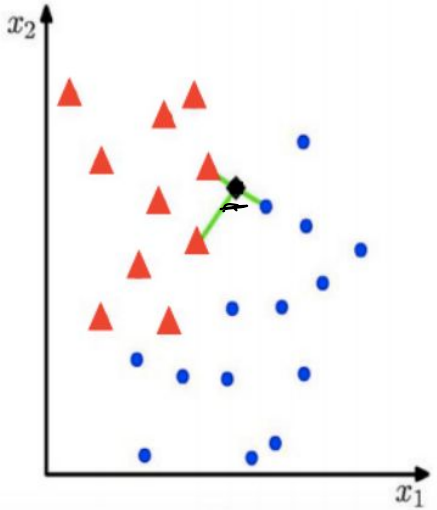
K-nearest neighbor

Algorithm

- For each test point, x, to be classified, find the K nearest samples in the training data
- Classify the point, x, according to the majority vote of their class labels

e.g. K = 3

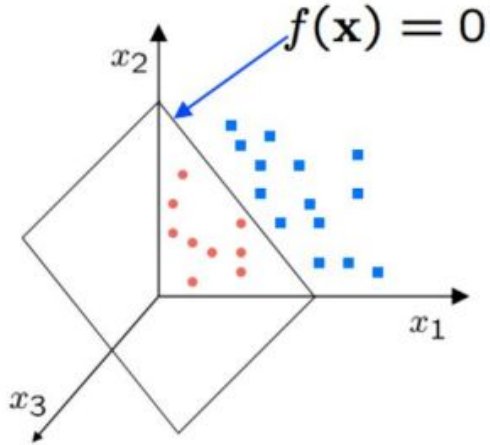
- applicable to multi-class case



Linear classifier

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 3D the discriminant is a plane, and in nD it is a hyperplane

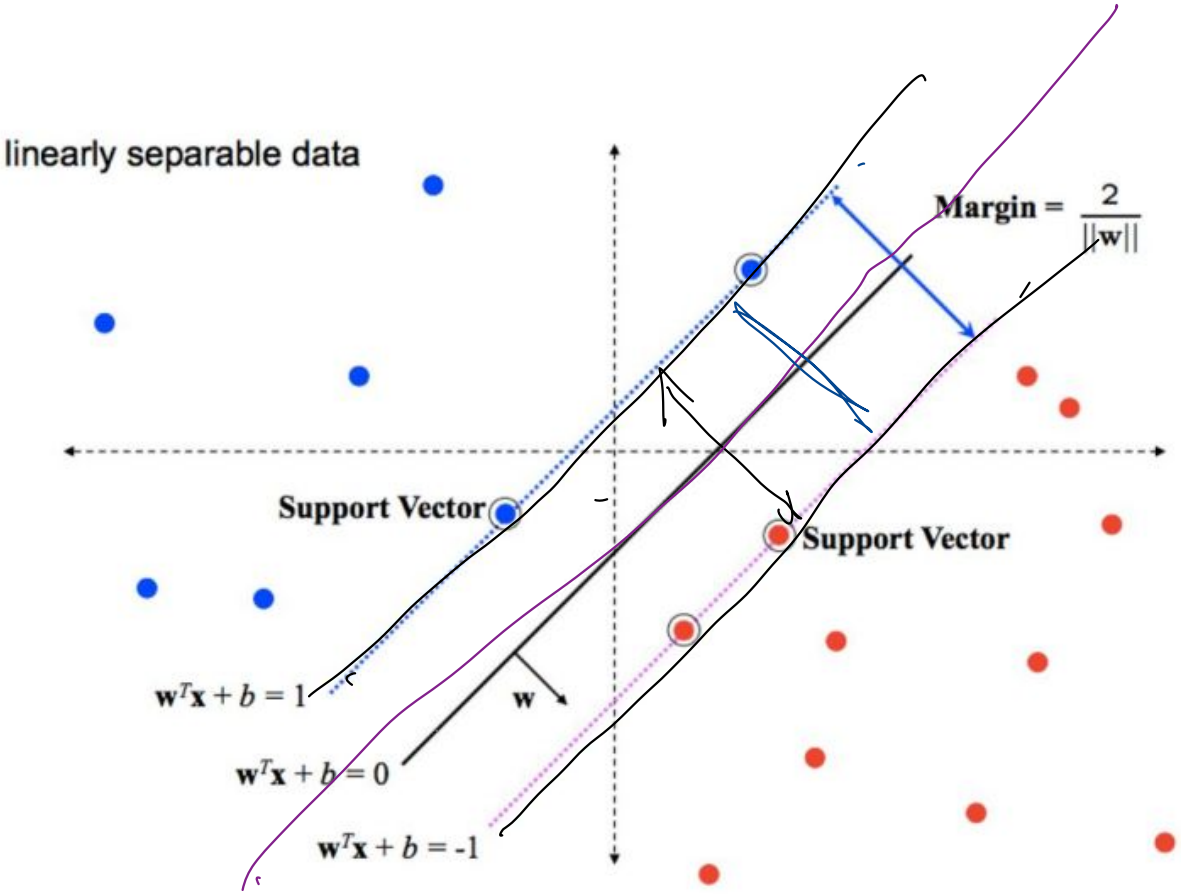
and a line in 2D

For a K-NN classifier it was necessary to `carry' the training data

For a linear classifier, the training data is used to learn **w** and then discarded

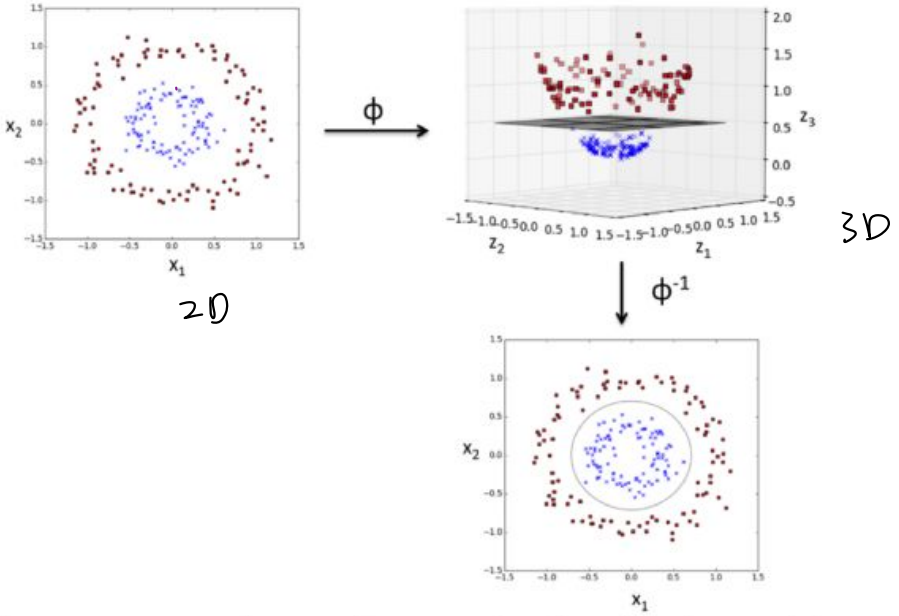
Only **w** is needed for classifying new data

SVM


linearly separable data
Margin = $\frac{2}{||w||}$
Support Vector
Support Vector
$w^T x + b = 1$
$w^T x + b = 0$
$w^T x + b = -1$
w

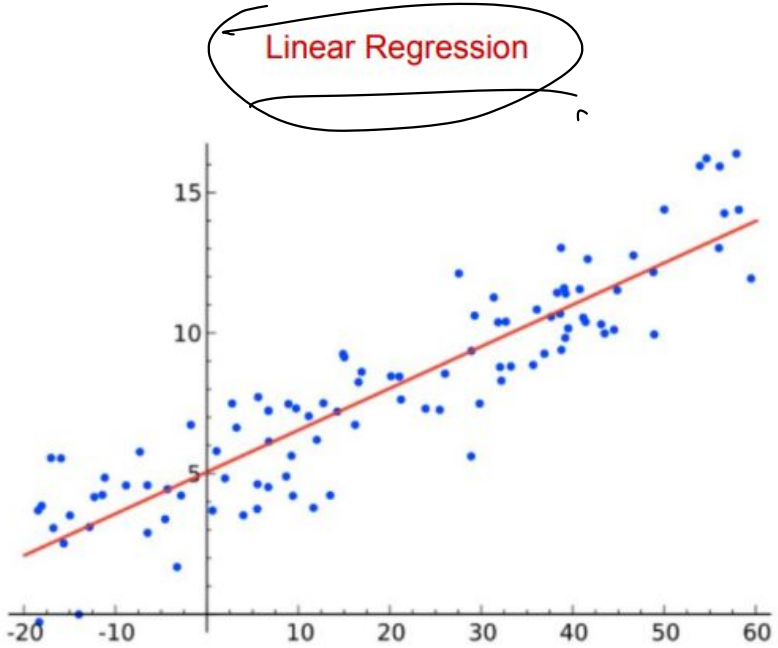
Feature maps

- Idea: transform linearly non-separable data to linearly separable data in some high dimension



picture source: "Python Machine Learning" by Sebastian Raschka

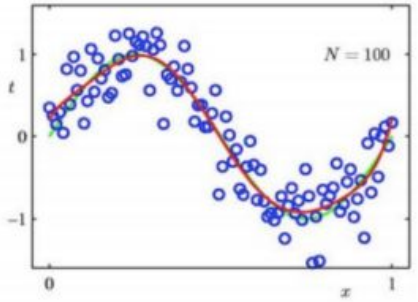
Regression



K-Nearest Neighbor Regression

Algorithm

- For each test point, x, find the K nearest samples x_i in the training data and their values y_i
- Output is mean of their values $f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} y_i$
- Again, need to choose (learn) K



Regression: Linear Regression

error : SSE

$$\beta^* = \arg\min_{\beta} \sum_i (y_i - \sum_j \beta_j X_{i,j})^2$$

$$= \arg\min_{\beta} (y - X\beta)^T (y - X\beta)$$

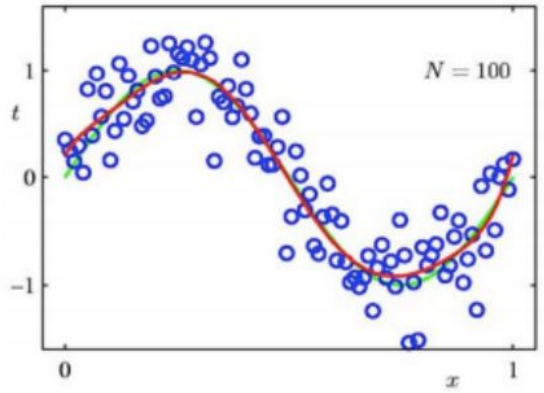
$$= (X^T X)^{-1} X^T y$$

closed form solution,

Regression: KNN Regression

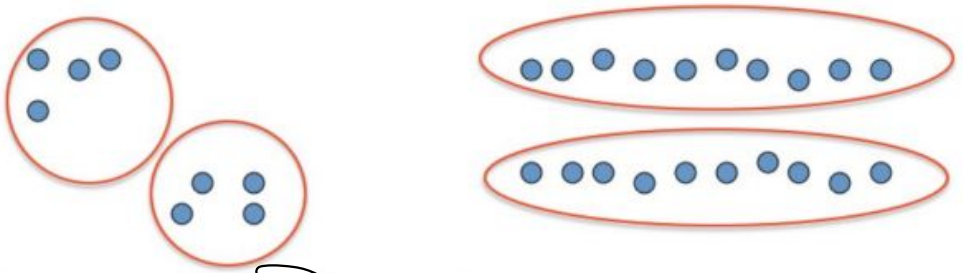
Algorithm

- For each test point, x, find the K nearest samples x_i in the training data and their values y_i
- Output is mean of their values $f(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} y_i$
- Again, need to choose (learn) K



Clustering

Basic idea: group together similar instances
Example: 2D point patterns



What could "similar" mean?

– One option: small Euclidean distance (squared)

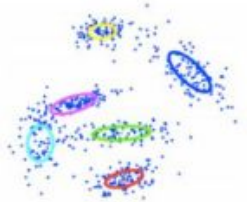
$$\text{dist}(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||_2^2$$

– Clustering results are crucially dependent on the measure of similarity (or distance) between "points" to be clustered

Clustering

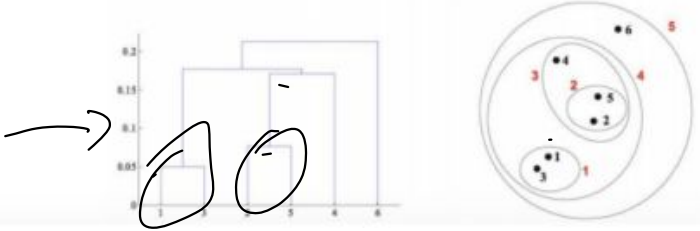
Flat or Partitional clustering (*K*-means, Gaussian mixture models, etc.)

- Partitions are independent of each other



Hierarchical clustering (e.g., agglomerative clustering, divisive clustering)

- Partitions can be visualized using a tree structure (a dendrogram)
- Does not need the number of clusters as input
- Possible to view partitions at different levels of granularities (i.e., can refine/coarsen clusters) using different *K*



K-means

- **Input:** N examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ ($\mathbf{x}_n \in \mathbb{R}^D$); the number of partitions K
- **Initialize:** K cluster centers $\mu_1, \ldots, \mu_K$. Several initialization options:
  - Randomly initialized anywhere in $\mathbb{R}^D$
  - Choose any K examples as the cluster centers
- **Iterate:**
  - Assign each of example $\mathbf{x}_n$ to its closest cluster center

$$\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\mathbf{x}_n - \mu_k||^2\}$$

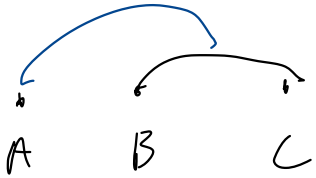
($\mathcal{C}_k$ is the set of examples closest to μ_k)

  - Recompute the new cluster centers μ_k (mean/centroid of the set $\mathcal{C}_k$)

$$\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \mathbf{x}_n$$

  - Repeat while not converged

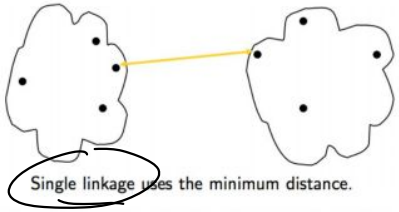
Agglomerative clustering

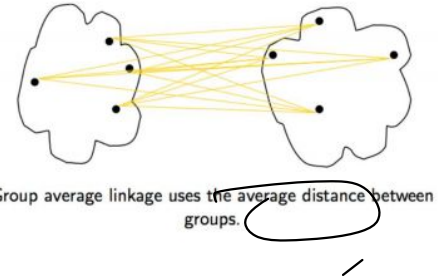


Algorithm:

- Maintain a set of clusters
- Initially, each instance in its own cluster
- Repeat:
  - Pick the two closest clusters
  - Merge them into a new cluster
  - Stop when there's only one cluster left

Distance between set?





Genome Assembly

Genome Assembly: de Bruijn

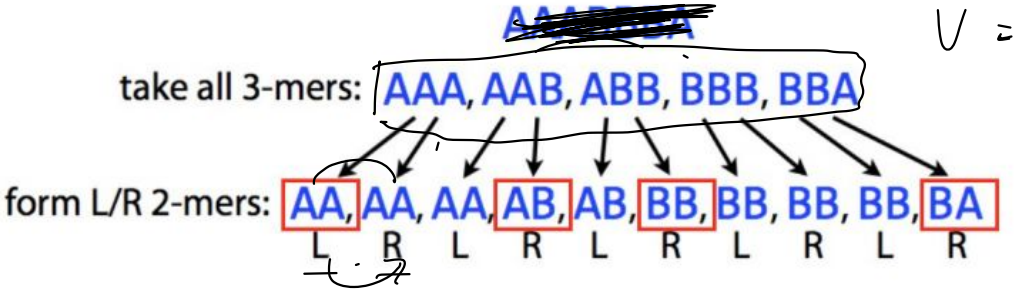
k :

1. Create multi-digraph G
2. Take all input kmers; for each kmer, split it into left (k-1)mer and right (k-1)mer. These are the set of nodes. For each pair of left and right (k-1)mer, add an edge from the left (k-1)mer to the right (k-1)mer. G is now complete.
3. Find an Eulerian path (a path that traverses each edge exactly once, O(|E|) time) on G. This path corresponds to one possible assembled genome

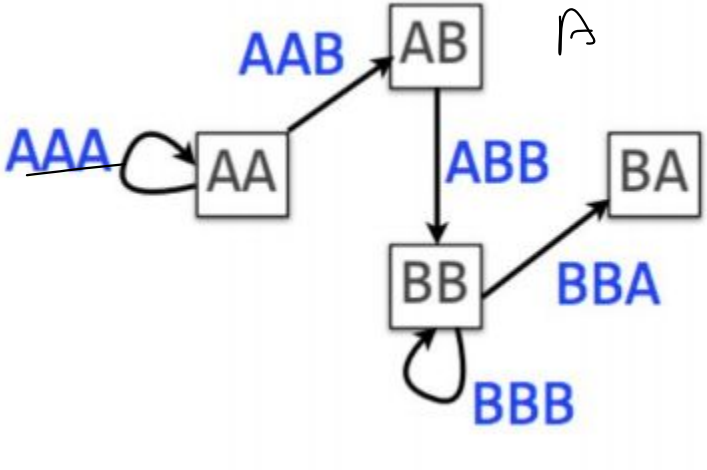
G = (V, E)

V = L-(k-1)mers

U R-(k-1)mers



Genome Assembly: de Bruijn



Genome Assembly: Overlap Graph

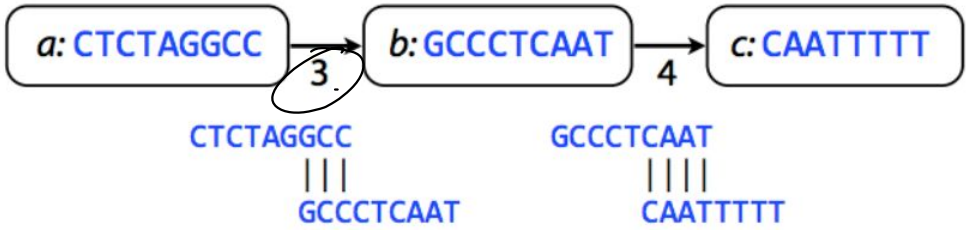
kmers

Below: overlap graph, where an overlap is a suffix/prefix match of at least 3 characters

A vertex is a read, a directed edge is an overlap between suffix of source and prefix of sink

Vertices (reads): { *a:* CTCTAGGCC, *b:* GCCCTCAAT, *c:* CAATTTTT }

Edges (overlaps): { (*a*, *b*), (*b*, *c*) }



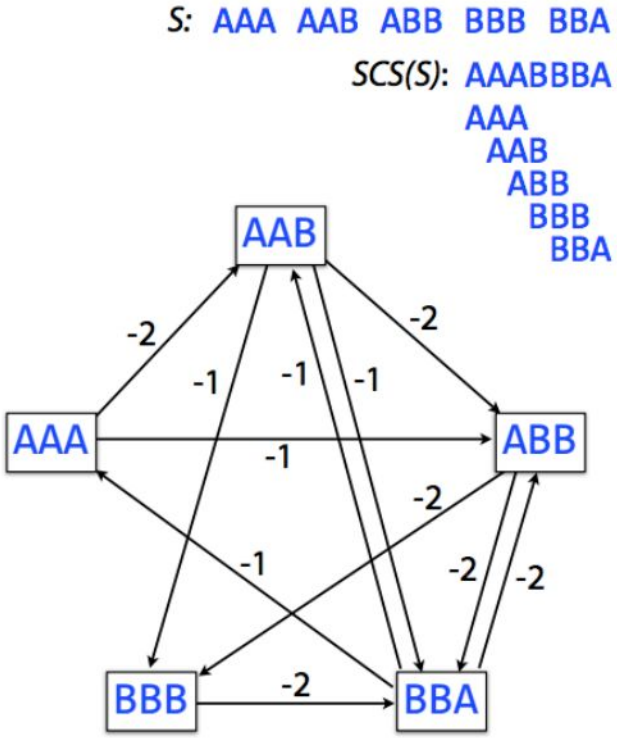
Genome Assembly: Overlap Graph

Can we solve it?

Imagine a modified overlap graph where each edge has cost = - (length of overlap)

SCS corresponds to a path that visits every node once, minimizing total cost along path

That's the *Traveling Salesman Problem* (*TSP*), which is NP-hard!



Genome Assembly: Overlap Graph

Can we solve it?

Imagine a modified overlap graph where each edge has cost = - (length of overlap)

SCS corresponds to a path that visits every node once, minimizing total cost along path

That's the *Traveling Salesman Problem* (*TSP*), which is NP-hard!