# Machine Learning in Genomics

Michael Schatz

Feb 18, 2020

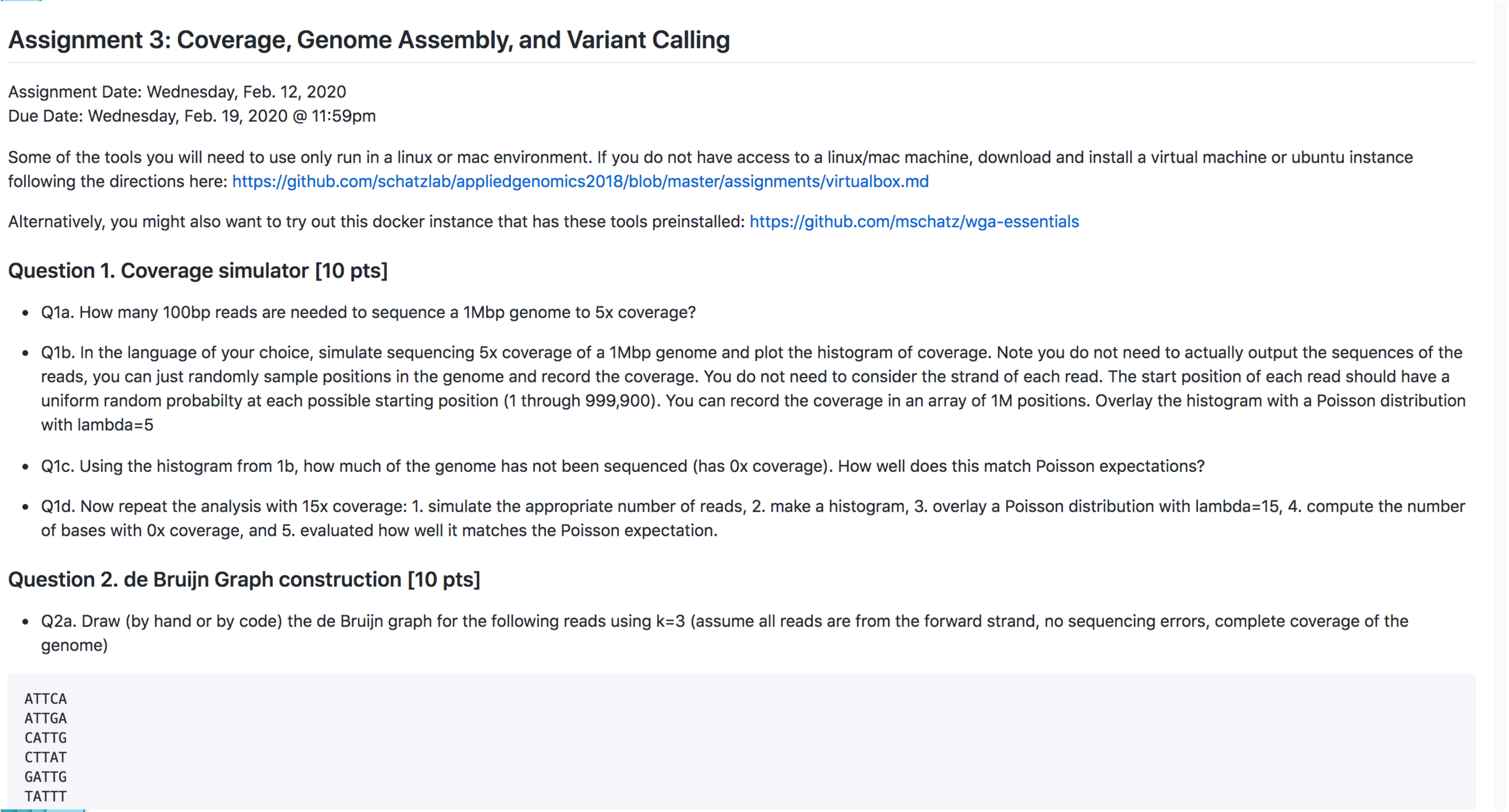
Lecture 8: Applied Comparative Genomics

## Assignment 2: Genome Assembly
## Due Wednesday Feb 12 @ 11:59pm

1. ***Setup Docker/Ubuntu***
2. ***Initialize Tools***
3. ***Download Reference Genome & Reads***
4. ***Decode the secret message***
   1. *Estimate coverage, check read quality*
   2. *Check kmer distribution*
   3. *Assemble the reads with spades*
   4. *Align to reference with MUMmer*
   5. *Extract foreign sequence*
   6. *dna-encode.pl -d*

https://github.com/schatzlab/appliedgenomics2020/blob/master/assignments/assignment2/README.md

# Assignment 3: Due Wed Feb 19



## Assignment 3: Coverage, Genome Assembly, and Variant Calling

Assignment Date: Wednesday, Feb. 12, 2020
Due Date: Wednesday, Feb. 19, 2020 @ 11:59pm

Some of the tools you will need to use only run in a linux or mac environment. If you do not have access to a linux/mac machine, download and install a virtual machine or ubuntu instance following the directions here: https://github.com/schatzlab/appliedgenomics2018/blob/master/assignments/virtualbox.md

Alternatively, you might also want to try out this docker instance that has these tools preinstalled: https://github.com/mschatz/wga-essentials

### Question 1. Coverage simulator [10 pts]

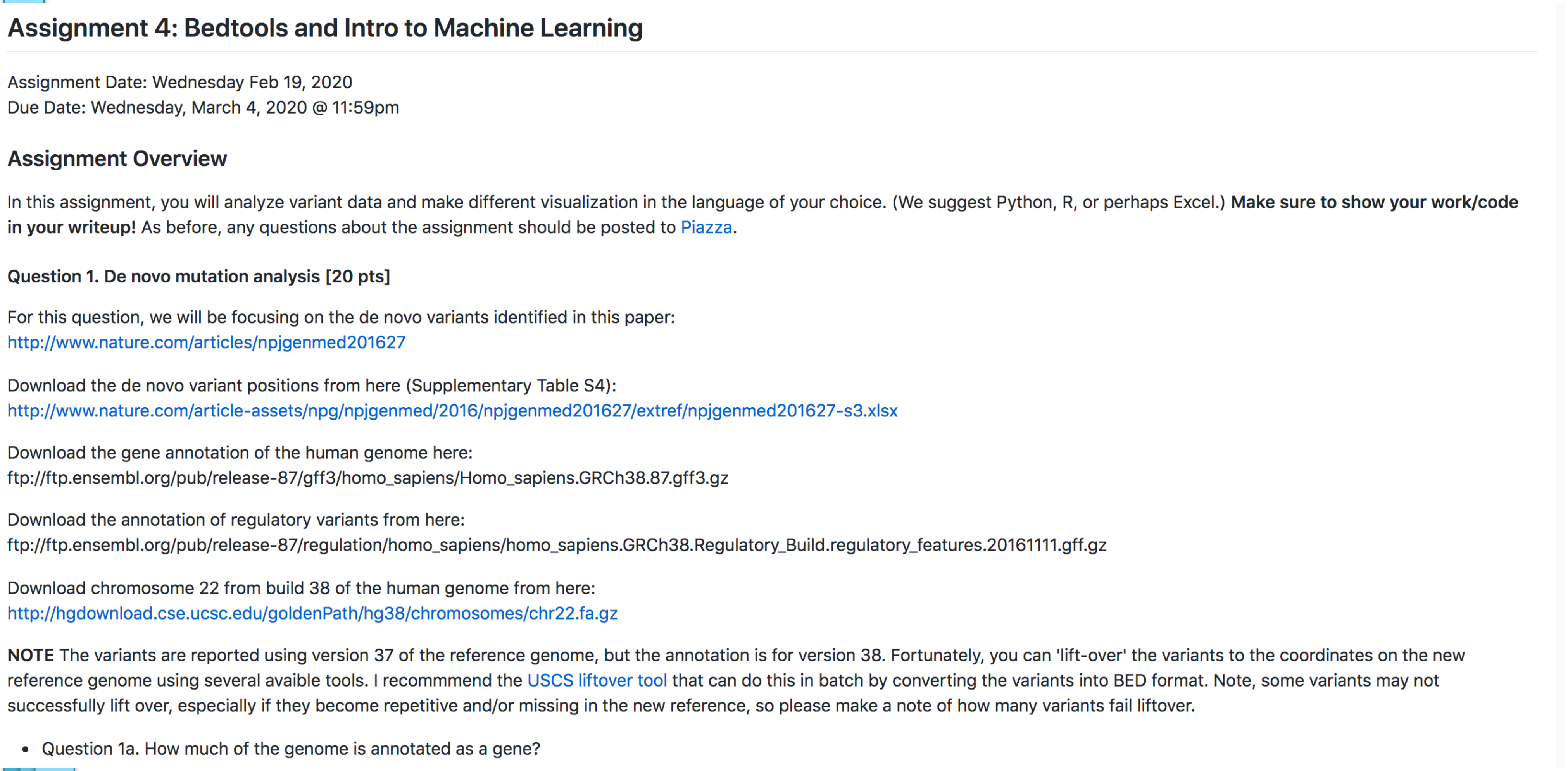
- Q1a. How many 100bp reads are needed to sequence a 1Mbp genome to 5x coverage?
- Q1b. In the language of your choice, simulate sequencing 5x coverage of a 1Mbp genome and plot the histogram of coverage. Note you do not need to actually output the sequences of the reads, you can just randomly sample positions in the genome and record the coverage. You do not need to consider the strand of each read. The start position of each read should have a uniform random probabilty at each possible starting position (1 through 999,900). You can record the coverage in an array of 1M positions. Overlay the histogram with a Poisson distribution with lambda=5
- Q1c. Using the histogram from 1b, how much of the genome has not been sequenced (has 0x coverage). How well does this match Poisson expectations?
- Q1d. Now repeat the analysis with 15x coverage: 1. simulate the appropriate number of reads, 2. make a histogram, 3. overlay a Poisson distribution with lambda=15, 4. compute the number of bases with 0x coverage, and 5. evaluated how well it matches the Poisson expectation.

### Question 2. de Bruijn Graph construction [10 pts]

- Q2a. Draw (by hand or by code) the de Bruijn graph for the following reads using k=3 (assume all reads are from the forward strand, no sequencing errors, complete coverage of the genome)

```
ATTCA
ATTGA
CATTG
CTTAT
GATTG
TATTT
```

# Assignment 4: Due Wed Mar 4



## Assignment 4: Bedtools and Intro to Machine Learning

Assignment Date: Wednesday Feb 19, 2020
Due Date: Wednesday, March 4, 2020 @ 11:59pm

### Assignment Overview

In this assignment, you will analyze variant data and make different visualization in the language of your choice. (We suggest Python, R, or perhaps Excel.) **Make sure to show your work/code in your writeup!** As before, any questions about the assignment should be posted to Piazza.

**Question 1. De novo mutation analysis [20 pts]**

For this question, we will be focusing on the de novo variants identified in this paper:
http://www.nature.com/articles/npjgenmed201627

Download the de novo variant positions from here (Supplementary Table S4):
http://www.nature.com/article-assets/npg/npjgenmed/2016/npjgenmed201627/extref/npjgenmed201627-s3.xlsx

Download the gene annotation of the human genome here:
ftp://ftp.ensembl.org/pub/release-87/gff3/homo_sapiens/Homo_sapiens.GRCh38.87.gff3.gz

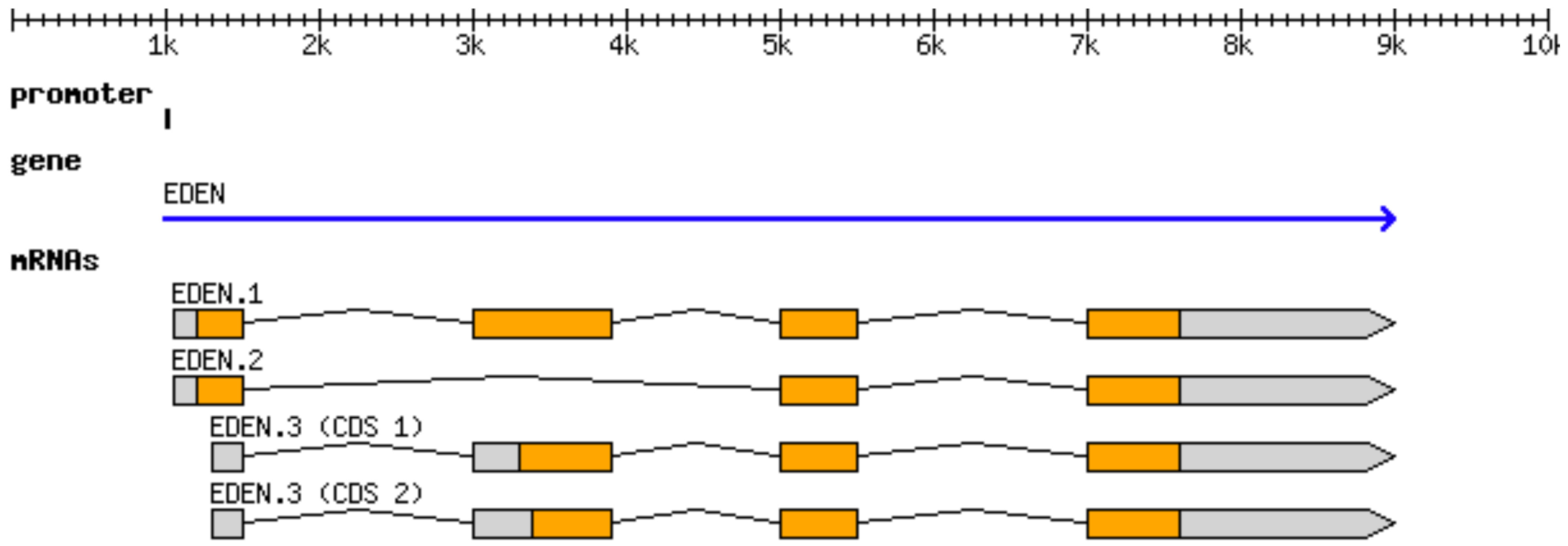
Download the annotation of regulatory variants from here:
ftp://ftp.ensembl.org/pub/release-87/regulation/homo_sapiens/homo_sapiens.GRCh38.Regulatory_Build.regulatory_features.20161111.gff.gz

Download chromosome 22 from build 38 of the human genome from here:
http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr22.fa.gz

**NOTE** The variants are reported using version 37 of the reference genome, but the annotation is for version 38. Fortunately, you can 'lift-over' the variants to the coordinates on the new reference genome using several avaible tools. I recommmend the USCS liftover tool that can do this in batch by converting the variants into BED format. Note, some variants may not successfully lift over, especially if they become repetitive and/or missing in the new reference, so please make a note of how many variants fail liftover.

- Question 1a. How much of the genome is annotated as a gene?

# Gene Models



- "Generic Feature Format" (GFF) records genomic features
  - Coordinates of each exon
  - Coordinates of UTRs
  - Link together exons into transcripts
  - Link together transcripts into gene models

http://www.sequenceontology.org/gff3.shtml

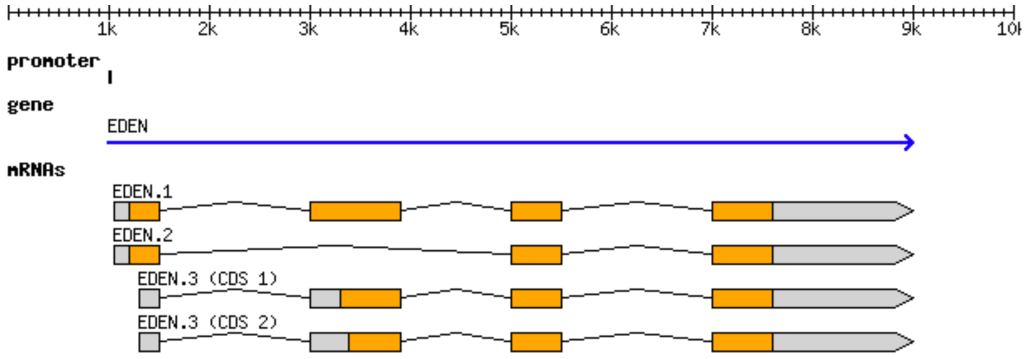
# GFF File format

GFF3 files are nine-column, tab-delimited, plain text files

| | |
|---|---|
| ***1. seqid:*** | The ID of the sequence |
| ***2. source:*** | Algorithm or database that generated this feature |
| ***3. type:*** | *gene/exon/CDS/etc…* |
| ***4. start:*** | 1-based coordinate |
| ***5. end***: | 1-based coordinate |
| ***6. score***: | E-values/p-values/index/colors/… |
| 7. ***strand:*** | "+' for positive "-" for minus, "." not stranded |
| ***8. phase:*** | For "CDS", where the feature begins with reference to the reading frame (0,1,2) |
| ***9. attributes:*** | A list of tag=value features<br>Parent: Indicates the parent of the feature (group exons into transcripts, transcripts into genes, …) |

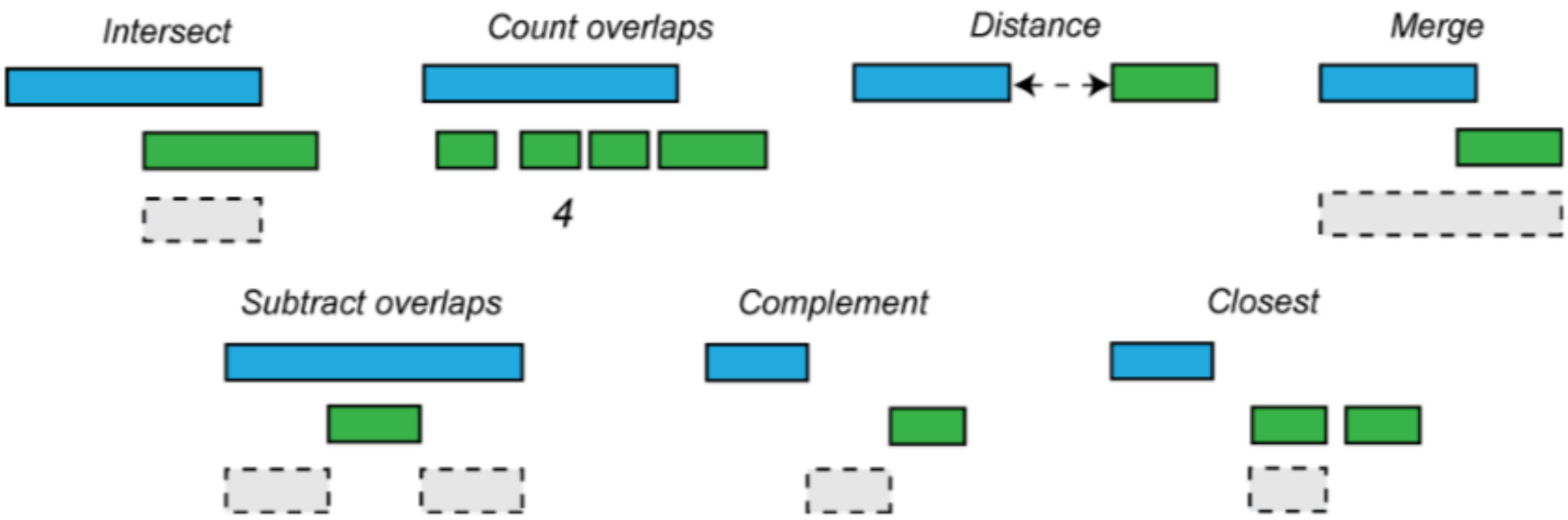
# GFF Example

Gene "EDEN" with 3 alternatively spliced transcripts, isoform 3 has two alternative translation start sites

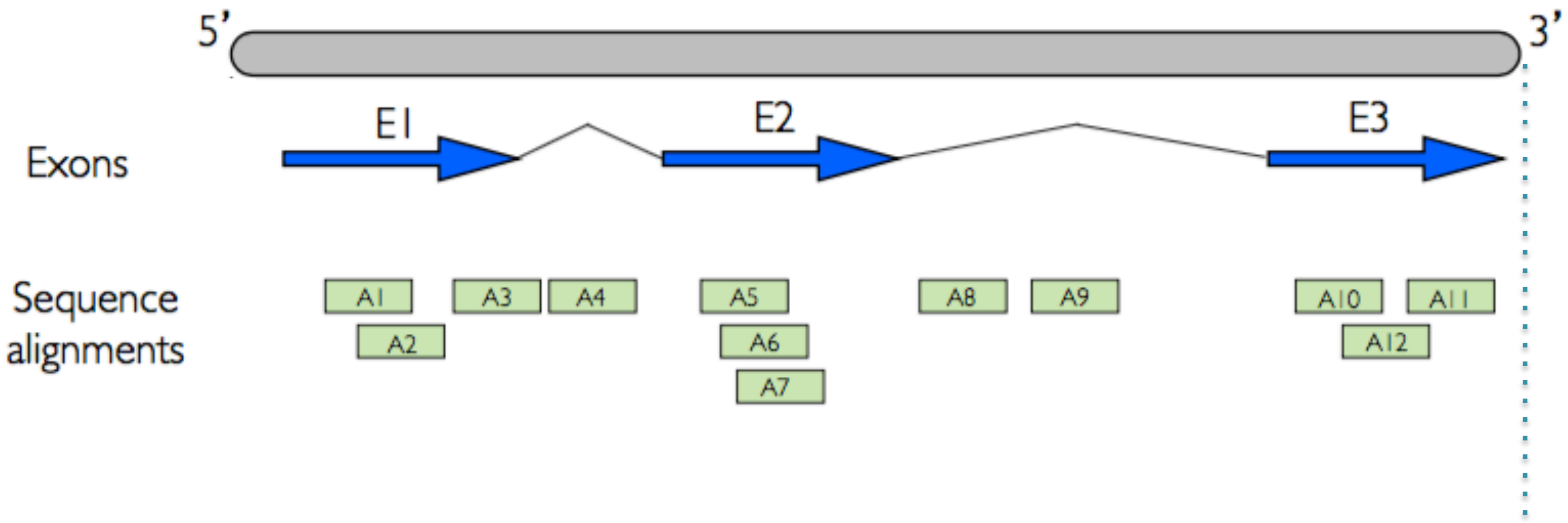


```
##gff-version 3
##sequence-region   ctg123 1 1497228
ctg123 . gene            1000  9000  .  +  .  ID=gene00001;Name=EDEN

ctg123 . TF_binding_site 1000  1012  .  +  .  ID=tfbs00001;Parent=gene00001

ctg123 . mRNA            1050  9000  .  +  .  ID=mRNA00001;Parent=gene00001;Name=EDEN.1
ctg123 . mRNA            1050  9000  .  +  .  ID=mRNA00002;Parent=gene00001;Name=EDEN.2
ctg123 . mRNA            1300  9000  .  +  .  ID=mRNA00003;Parent=gene00001;Name=EDEN.3

ctg123 . exon            1300  1500  .  +  .  ID=exon00001;Parent=mRNA00003
ctg123 . exon            1050  1500  .  +  .  ID=exon00002;Parent=mRNA00001,mRNA00002
ctg123 . exon            3000  3902  .  +  .  ID=exon00003;Parent=mRNA00001,mRNA00003
ctg123 . exon            5000  5500  .  +  .  ID=exon00004;Parent=mRNA00001,mRNA00002,mRNA00003
ctg123 . exon            7000  9000  .  +  .  ID=exon00005;Parent=mRNA00001,mRNA00002,mRNA00003

ctg123 . CDS             1201  1500  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             3000  3902  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             5000  5500  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1
ctg123 . CDS             7000  7600  .  +  0  ID=cds00001;Parent=mRNA00001;Name=edenprotein.1

ctg123 . CDS             1201  1500  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS             5000  5500  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2
ctg123 . CDS             7000  7600  .  +  0  ID=cds00002;Parent=mRNA00002;Name=edenprotein.2

ctg123 . CDS             3301  3902  .  +  0  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS             5000  5500  .  +  1  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3
ctg123 . CDS             7000  7600  .  +  1  ID=cds00003;Parent=mRNA00003;Name=edenprotein.3

ctg123 . CDS             3391  3902  .  +  0  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS             5000  5500  .  +  1  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
ctg123 . CDS             7000  7600  .  +  1  ID=cds00004;Parent=mRNA00003;Name=edenprotein.4
```

# BEDTools to the rescue!



# Plane Sweep to the Rescue!



How many comparisons does the plane sweep algorithm make?

Each read is compared to the "active set"

Relatively few exons overlap: average ~1.1 active exons/position

Total comparisons: 900M reads * 1.1 "active exons/read" = 990M comparisons ☺

Output is basically as fast as we can read the input data ☺

## Machine Learning Primer 1:

## Unsupervised Learning
## aka Clustering

# Clustering Refresher

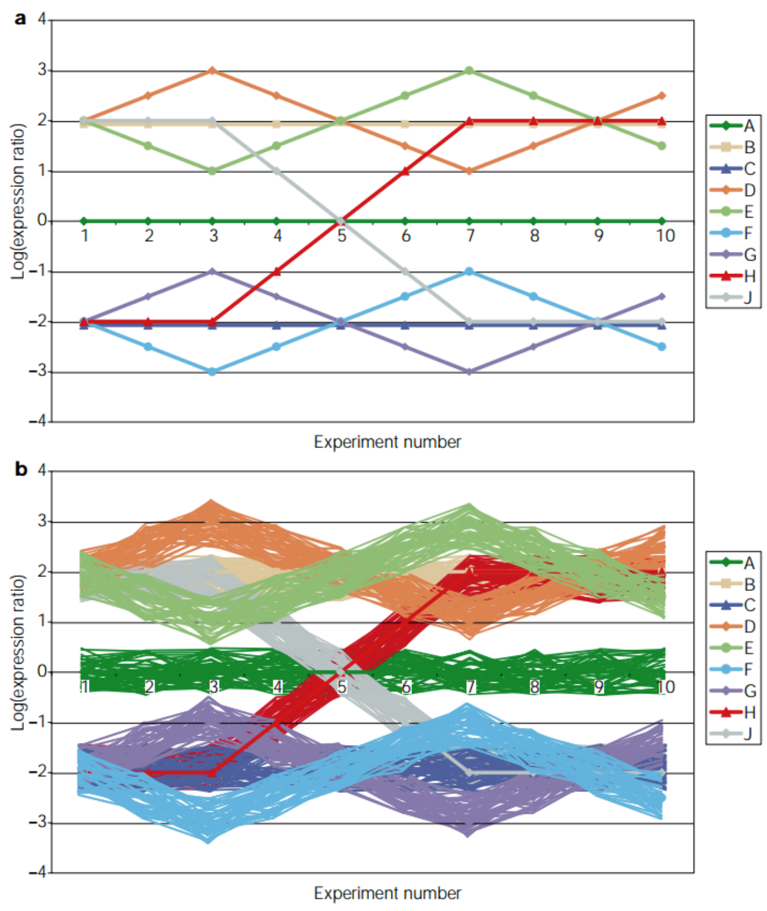
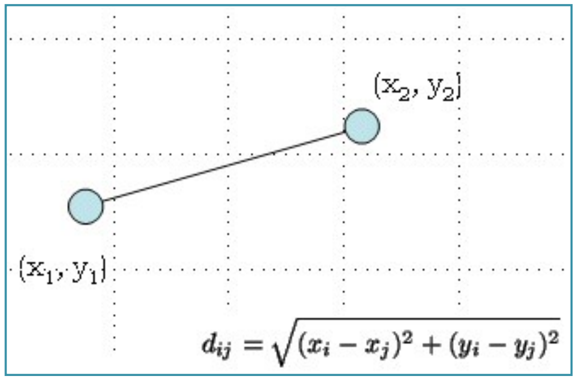
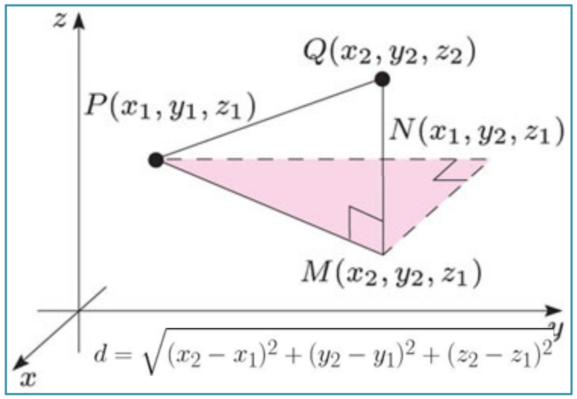


Figure 2 | **A synthetic gene-expression data set.** This data set provides an opportunity to evaluate how various clustering algorithms reveal different features of the data. **a** | Nine distinct gene-expression patterns were created with $\log_2$(ratio) expression measures defined for ten experiments. **b** | For each expression pattern, 50 additional genes were generated, representing variations on the basic patterns.

Euclidean Distance



$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$



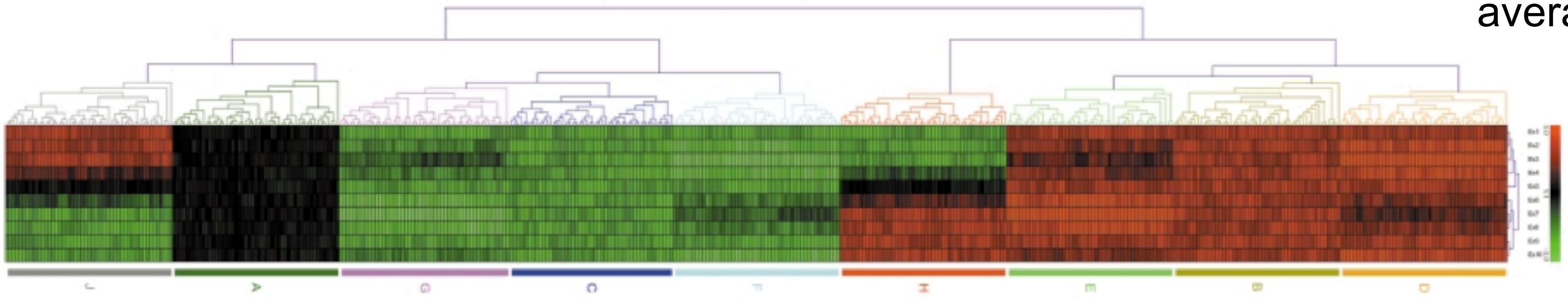
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

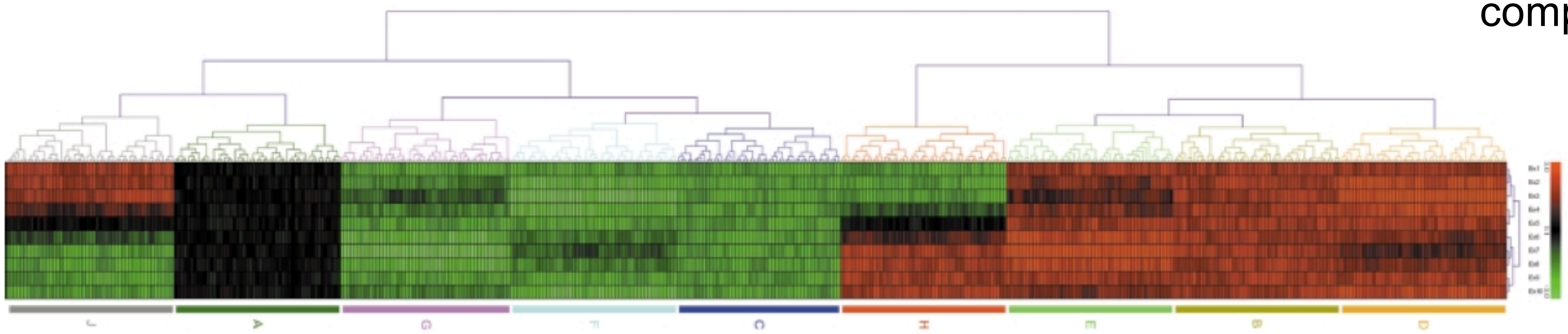
**Computational genetics: Computational analysis of microarray data**
Quackenbush (2001) *Nature Reviews Genetics.* doi:10.1038/35076576

# Hierarchical Clustering

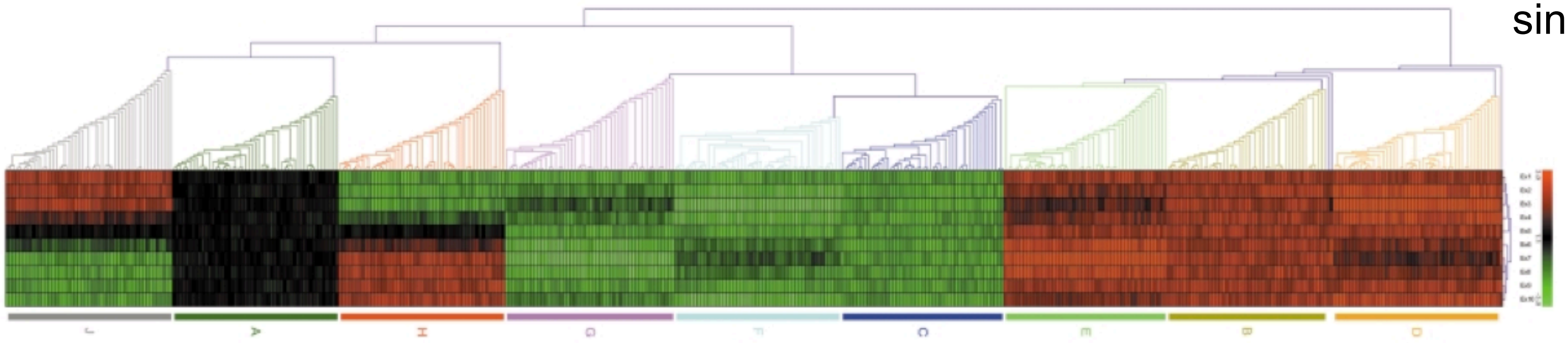
average



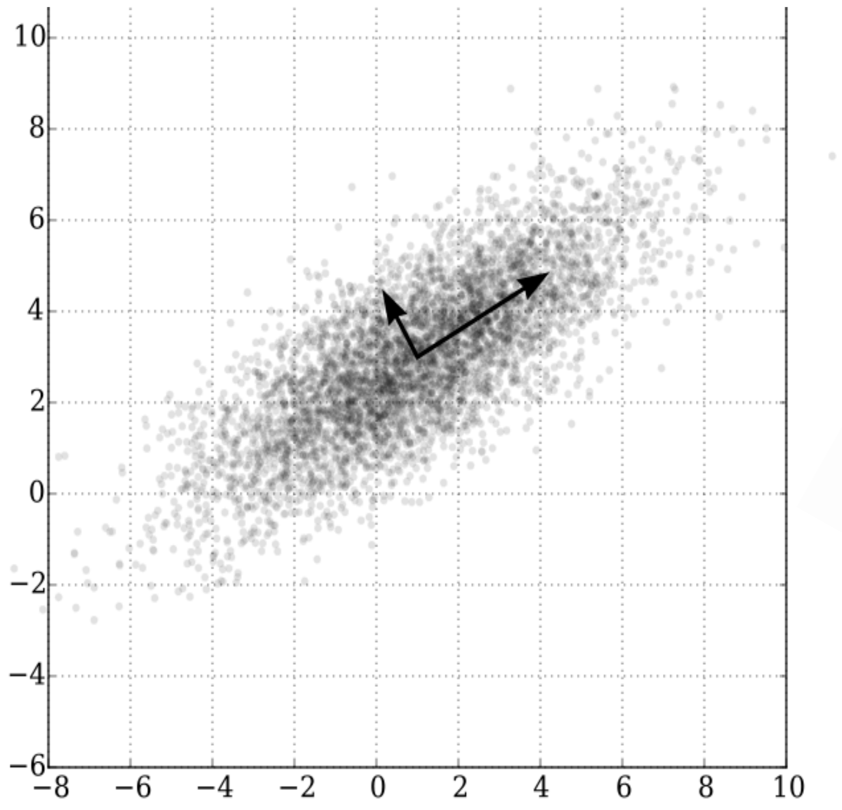
complete

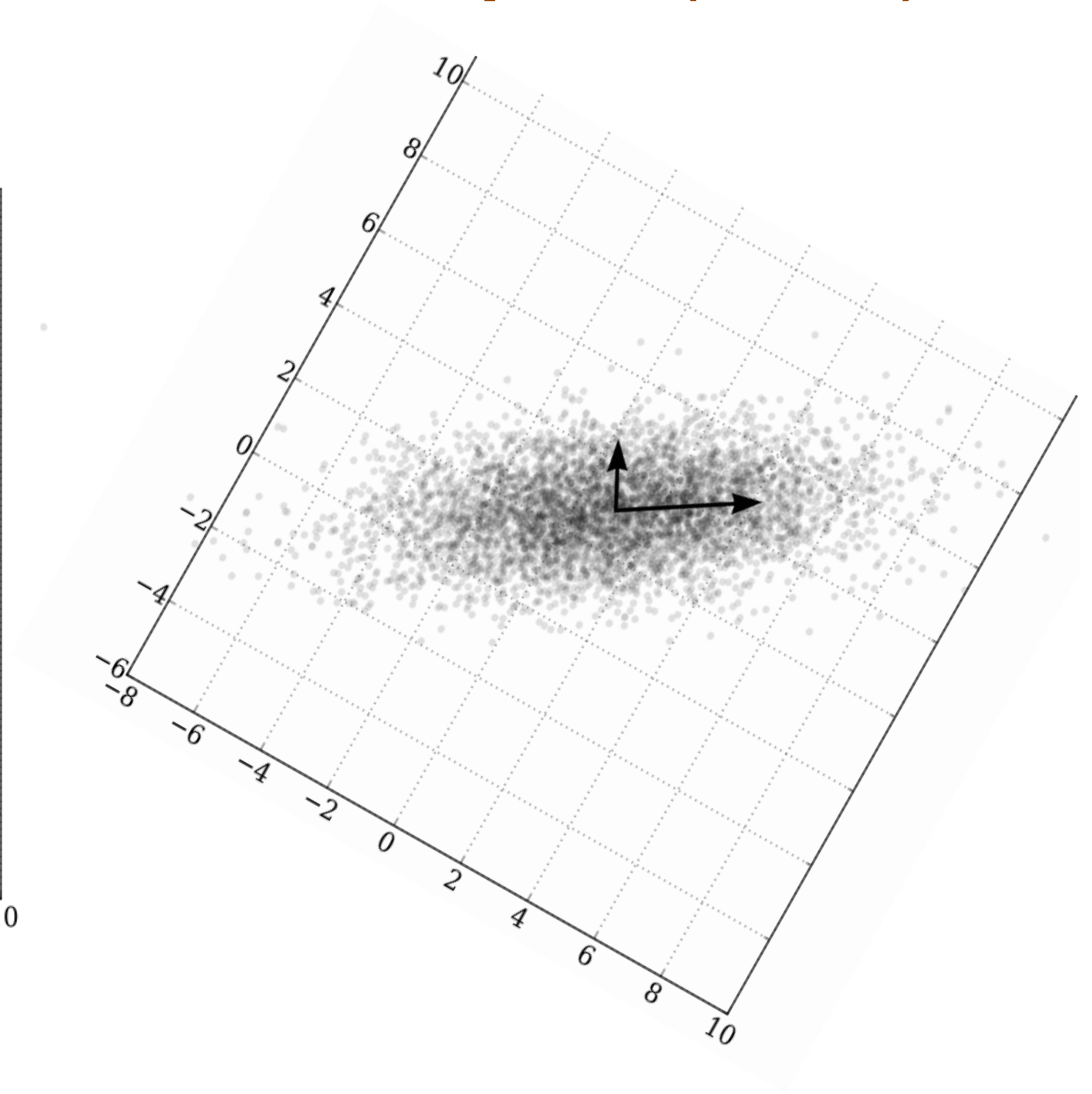


single



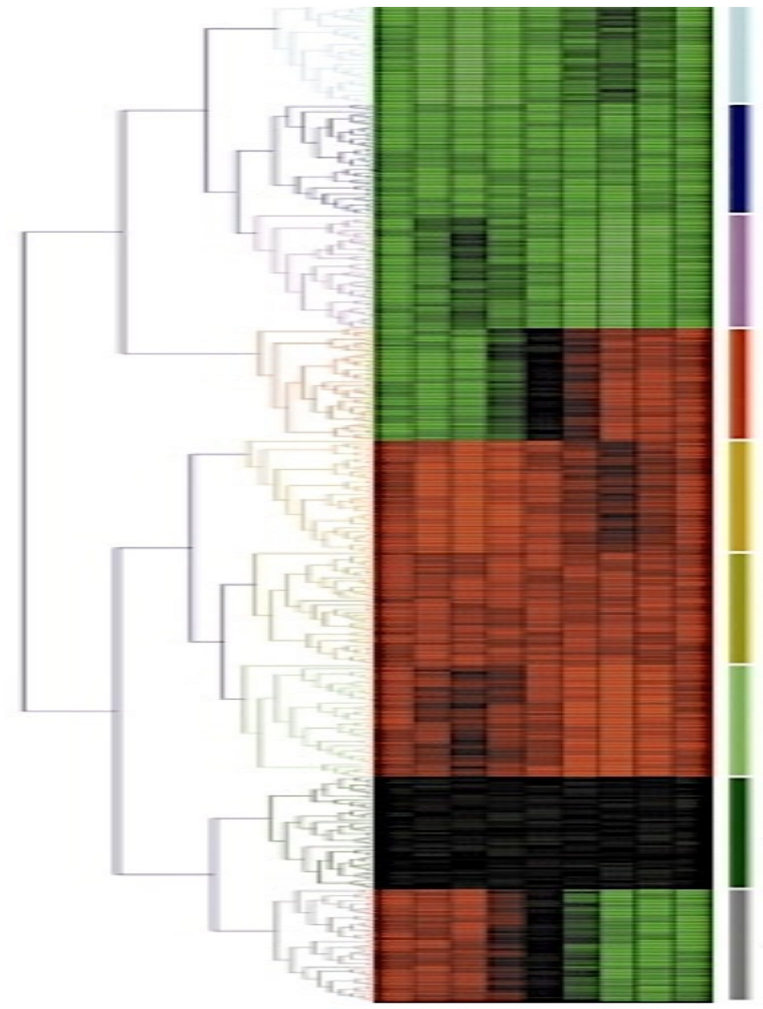
# Principle Components Analysis (PCA)





PC1: “New X”- The dimension with the most variability
PC2: “New Y”- The dimension with the second most variability

# Principle Components Analysis (PCA)



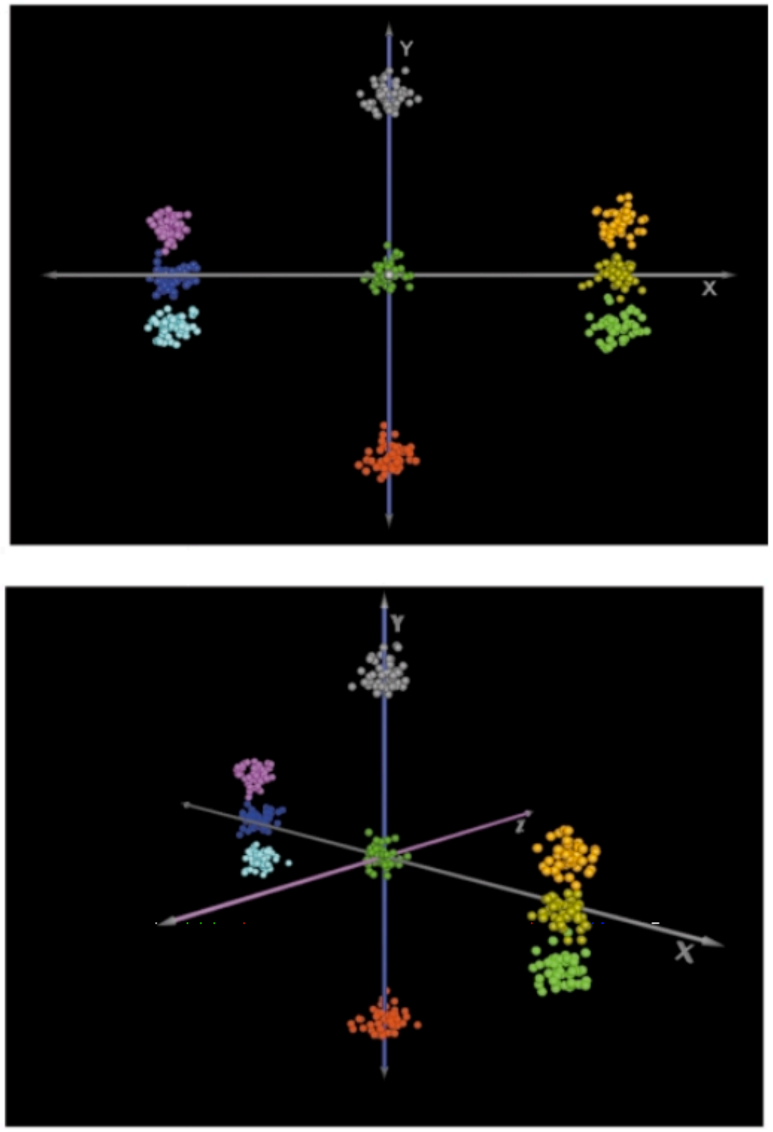


Figure 4 | **Principal component analysis.** The same demonstration data set was analysed using **a** | hierarchical (average-linkage) clustering and **b** | principal component analysis using Euclidean distance, to show how each treats the data, with genes colour coded on the basis of hierarchical clustering results for comparison.

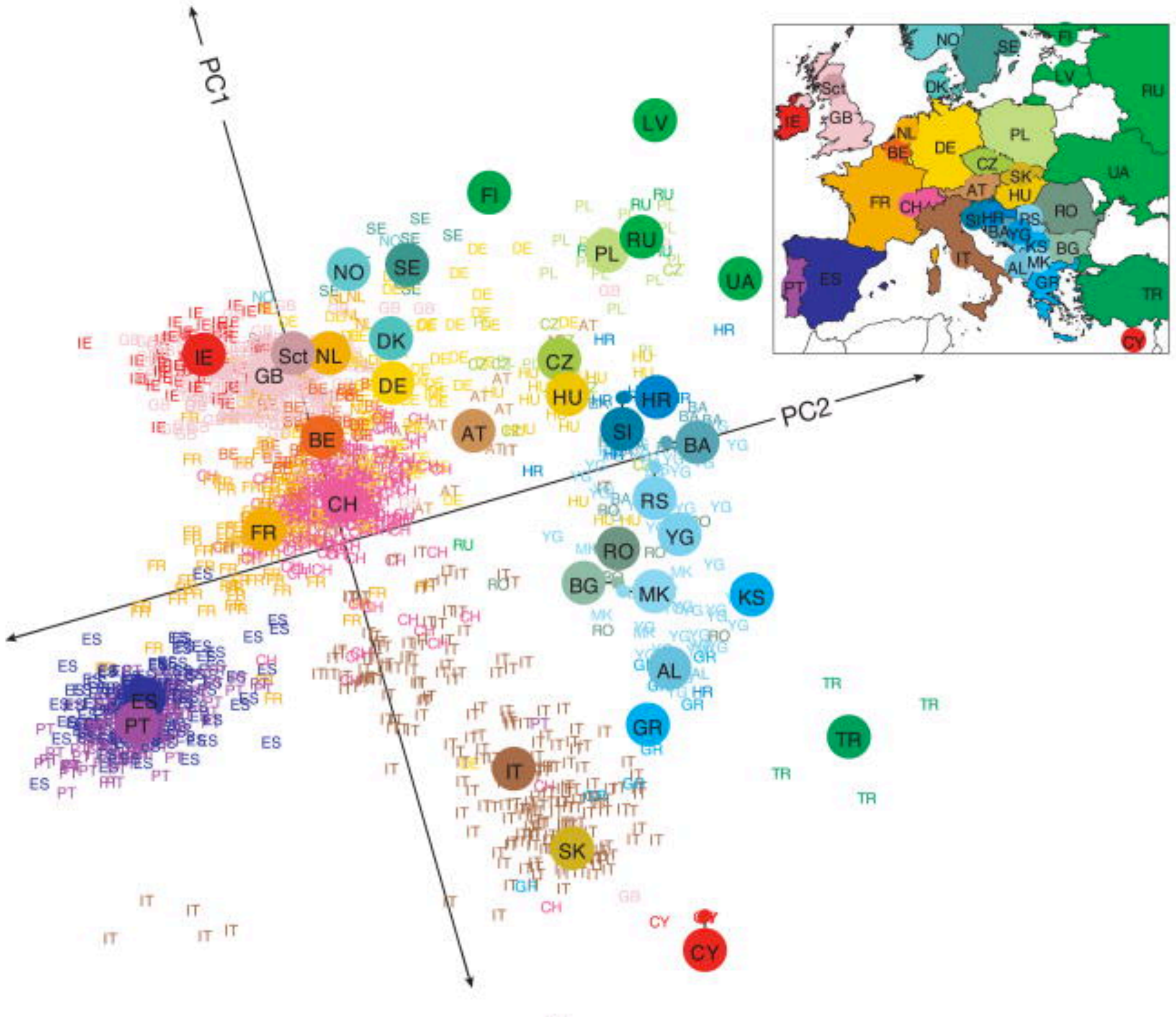
**Genotype Matrix**

|  | P1 | P2 | P3 … |
|---|---|---|---|
| SNP1 | 0 | 1 | 0 |
| SNP2 | 1 | 0 | 0 |
| SNP3 | 0 | 0 | 2 |
| … |  |  |  |

0 = hom ref

1 = het ref/alt

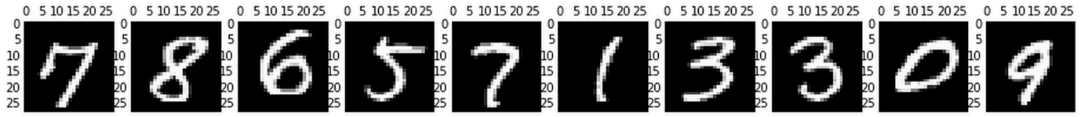
2 = hom alt

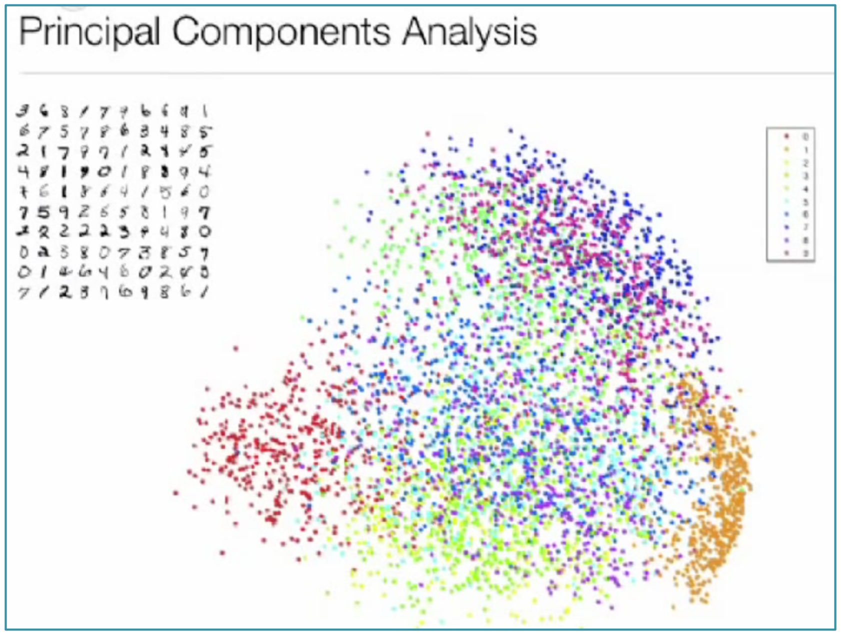


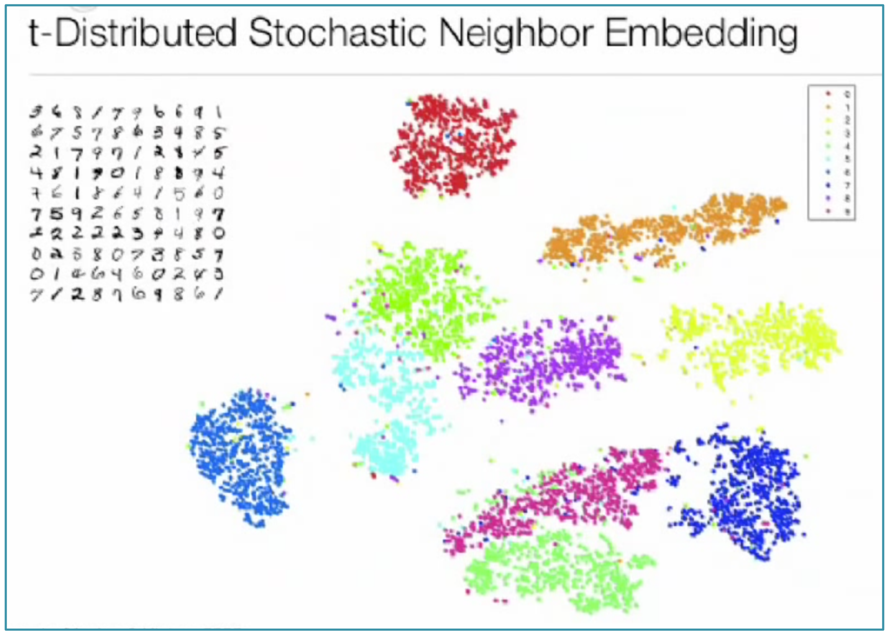
**Genes mirror geography within Europe**

Novembre et al (2008) Nature. doi: 10.1038/nature07331

# PCA and t-SNE





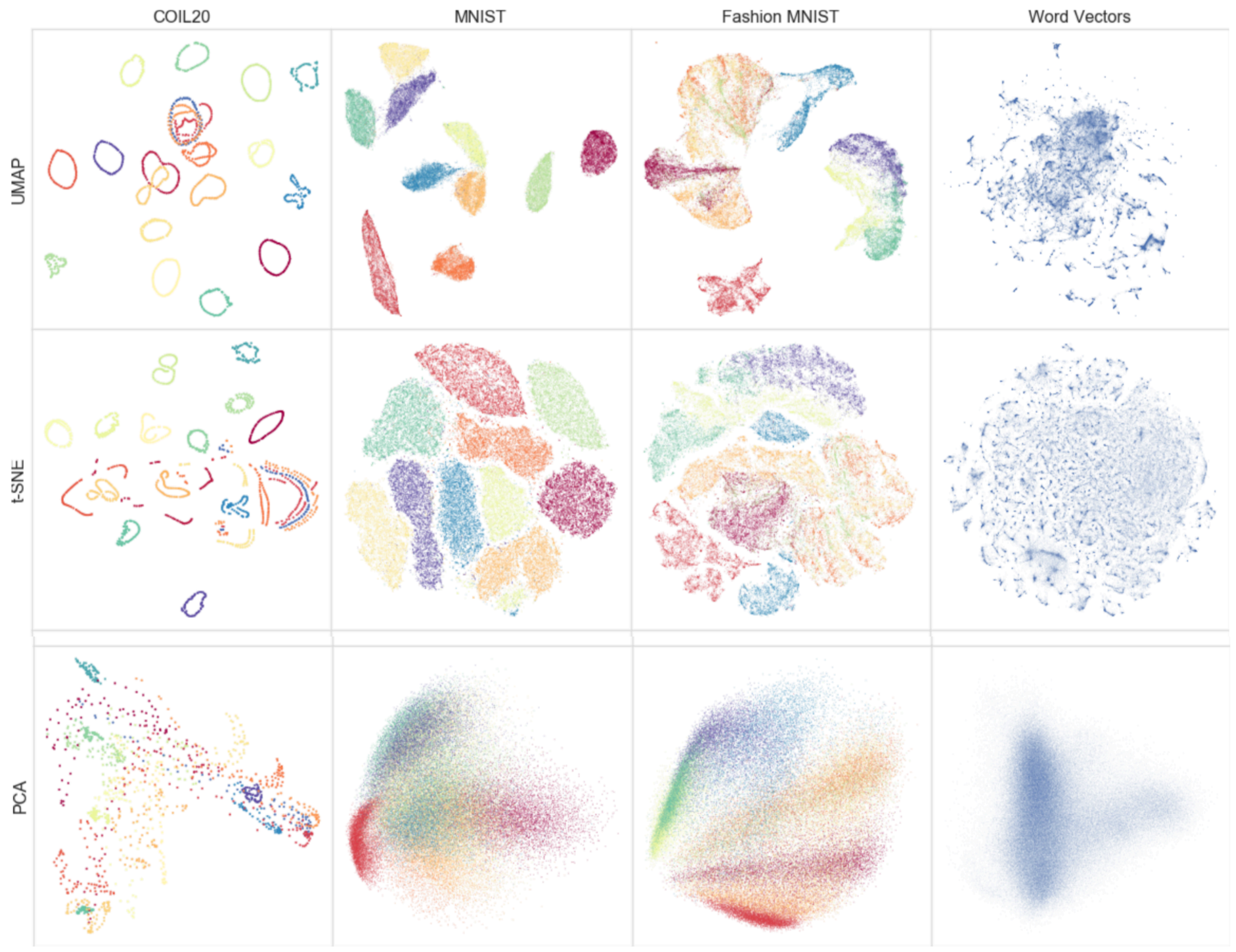


**t-distributed Stochastic Neighborhood Embedding**

- Non-linear dimensionality reduction technique: distances are only locally meaningful
- Rather than Euclidean distances, for each point fits a Gaussian kernel to fit the nearest N neighbors (perplexity) that define the probabilities that two points should be close together
- Using an iterative spring embedding system to place high probability points nearby

**Visualizing Data Using t-SNE**
van der Maaten & Hinton (2008) Journal of Machine Learning Research. 9: 2579–2605.
https://www.youtube.com/watch?v=RJVL80Gg3lA
https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1

# UMAP



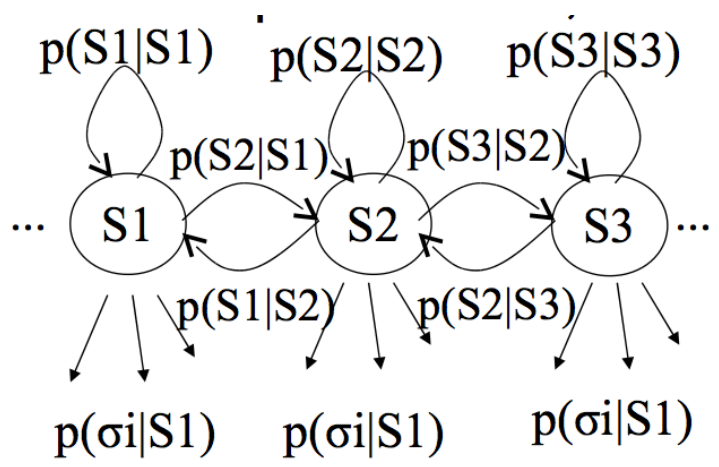
**UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction**
McInnes et al (2018) arXiv. 1802.03426
https://www.youtube.com/watch?v=nq6iPZVUxZU
https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668

# Machine Learning Primer 2:

# Hidden Markov Models

# What is an HMM?

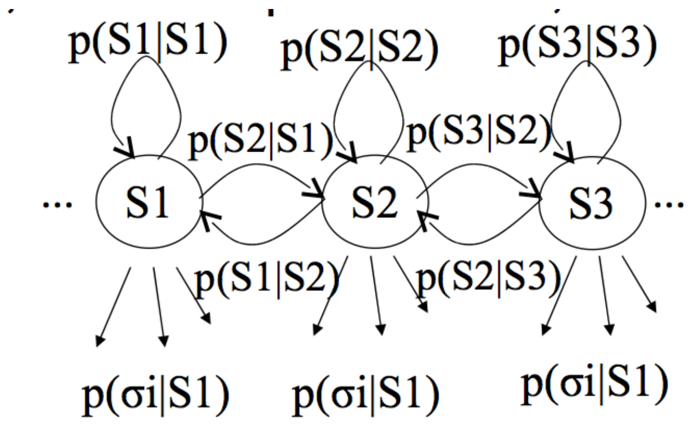
- Dynamic Bayesian Network
  - A set of states
    - {Fair, Biased} for coin tossing
    - {Gene, Not Gene} for Bacterial Gene
    - {Intergenic, Exon, Intron} for Eukaryotic Gene
    - {Modern, Neanderthal} for Ancestry



  - A set of emission characters
    - E={H,T} for coin tossing
    - E={1,2,3,4,5,6} for dice tossing
    - E={A,C,G,T} for DNA
  - State-specific emission probabilities
    - P(H | Fair) = .5, P(T | Fair) = .5, P(H | Biased) = .9, P(T | Biased) = .1
    - P(A | Gene) = .9, P(A | Not Gene) = .1 …
  - A probability of taking a transition
    - P($s_i$=Fair|$s_{i-1}$=Fair) = .9, P($s_i$=Bias|$s_{i-1}$ = Fair) .1
    - P($s_i$=Exon | $s_{i-1}$=Intergenic), …

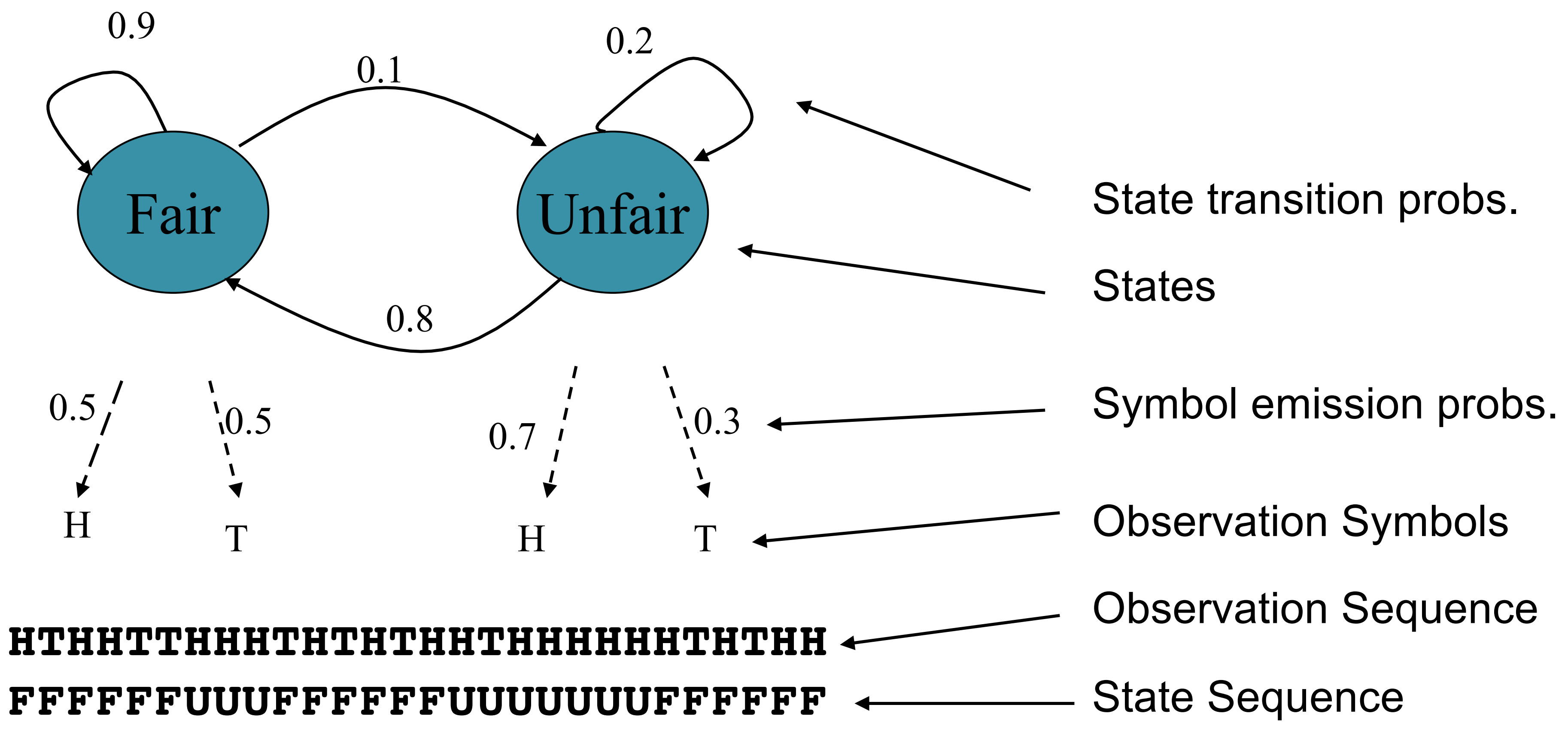
# Why Hidden?

- Observers can see the emitted symbols of an HMM (i.e., nucleotides) but have no ability to know which state the HMM is currently in (exon/intron/intergenic/etc).
  - But we can ***infer*** the most likely hidden states of an HMM based on the given sequence of emitted symbols.



**AAAGCATGCATTTAACGTGAGCACAATAGATTACA**

# HMM Example - Casino Coin



**Motivation:** Given a sequence of H & Ts, can you tell at what times the casino cheated?

Slide credit: Fatih Gelgi, Arizona State U.

## Three classic HMM problems

1. **Evaluation**: given a model and an output sequence, what is the probability that the model generated that output?

2. **Decoding**: given a model and an output sequence, what is the most likely state sequence through the model that generated the output?

3. **Learning**: given a model and a set of observed sequences, how do we set the model's parameters so that it has a high probability of generating those sequences?

## Three classic HMM problems

1. **Evaluation**: given a model and an output sequence, what is the probability that the model generated that output?

- To answer this, we consider all possible paths through the model

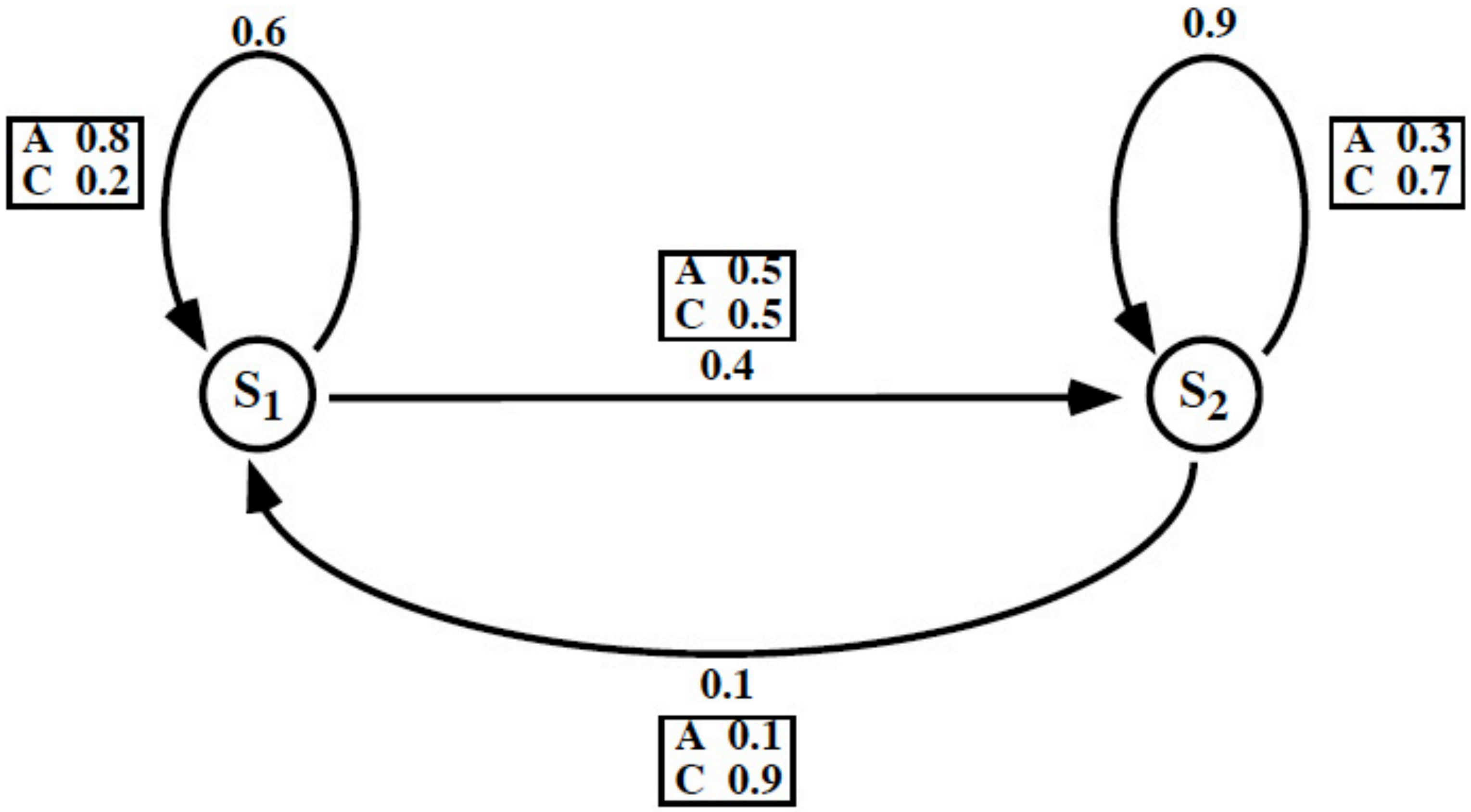
- Example: we might have a set of HMMs representing protein families -> pick the model with the best score

# Solving the Evaluation problem: The Forward algorithm

- To solve the Evaluation problem (probability that the model generated the sequence), we use the HMM and the data to build a *trellis*

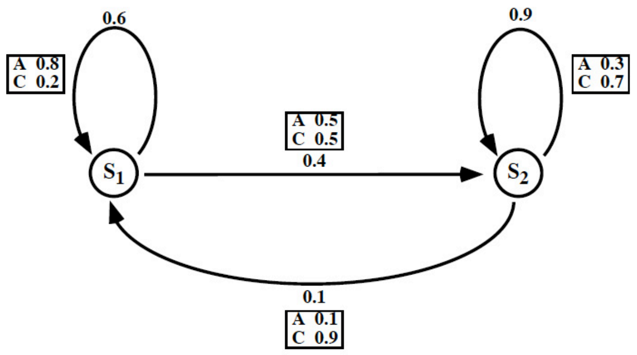
- Filling in the trellis will give tell us the probability that the HMM generated the data by finding all possible paths that could do it
  - Especially useful to evaluate from which models, a given sequence is most likely to have originated

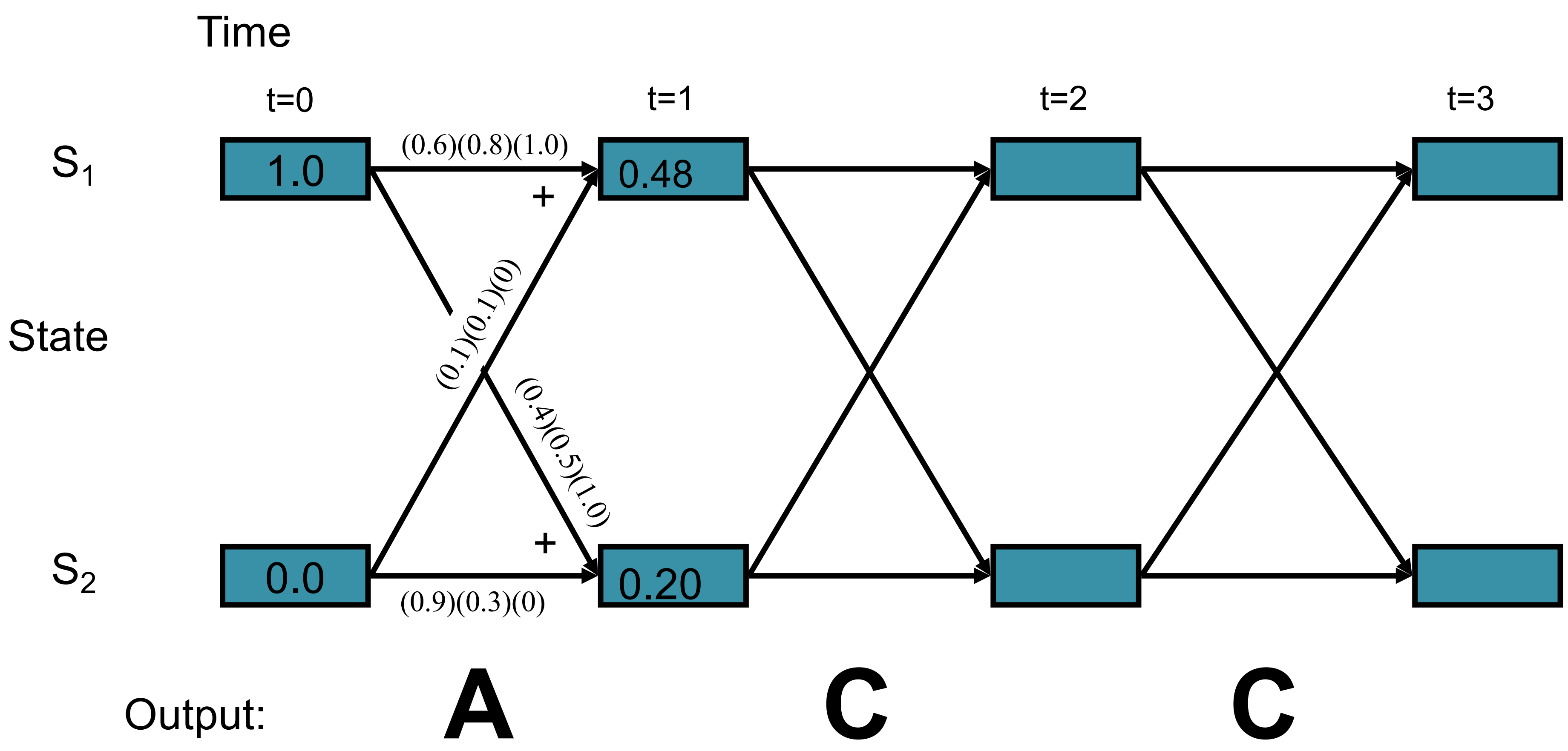
# Our sample HMM



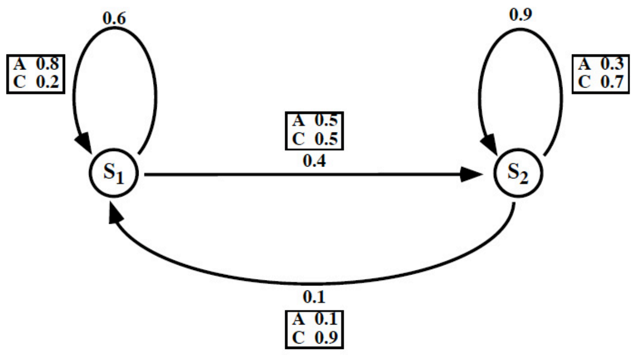
Let $S_1$ be initial state, $S_2$ be final state

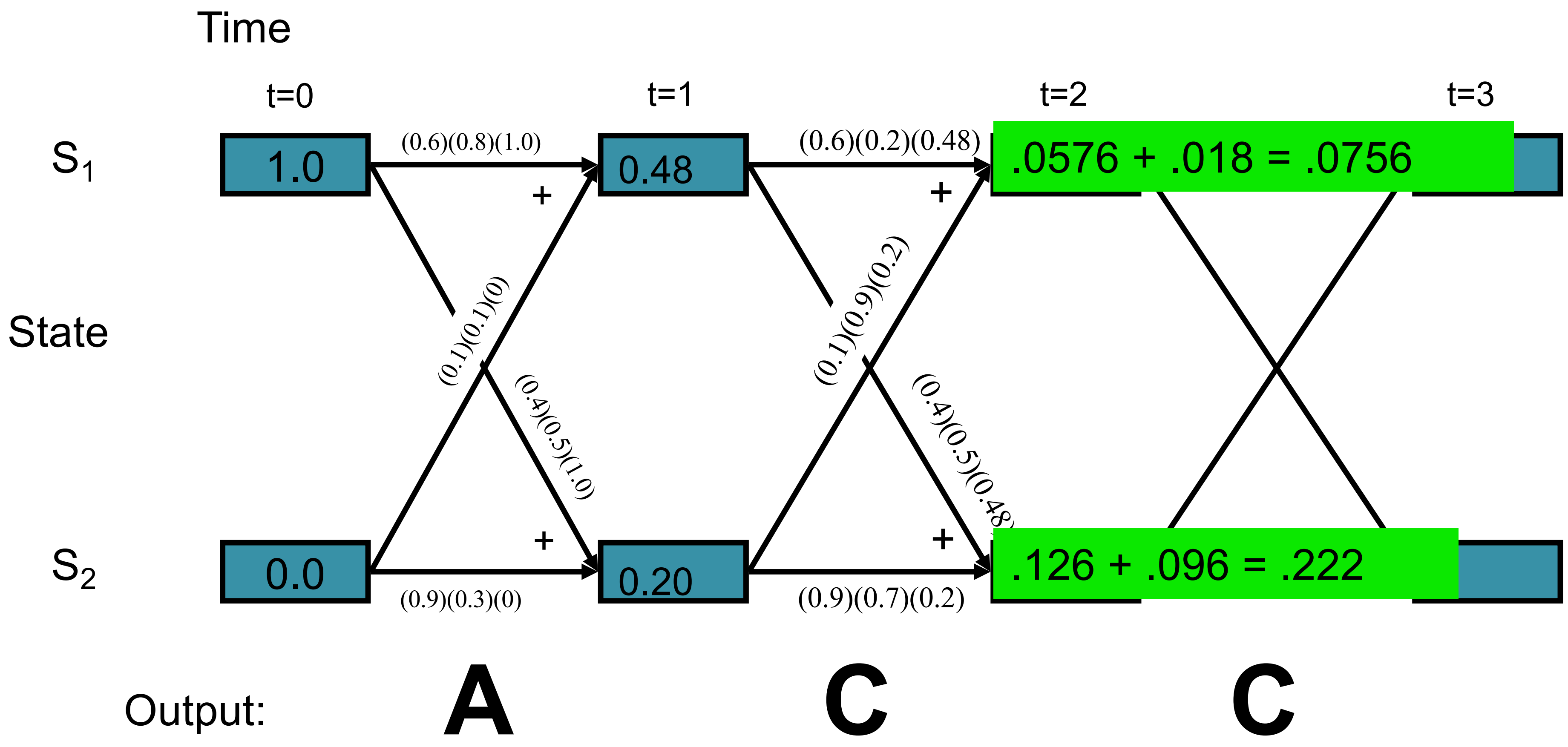
# A trellis for the Forward Algorithm



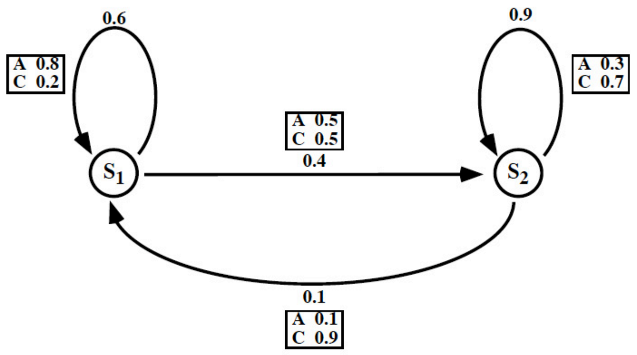


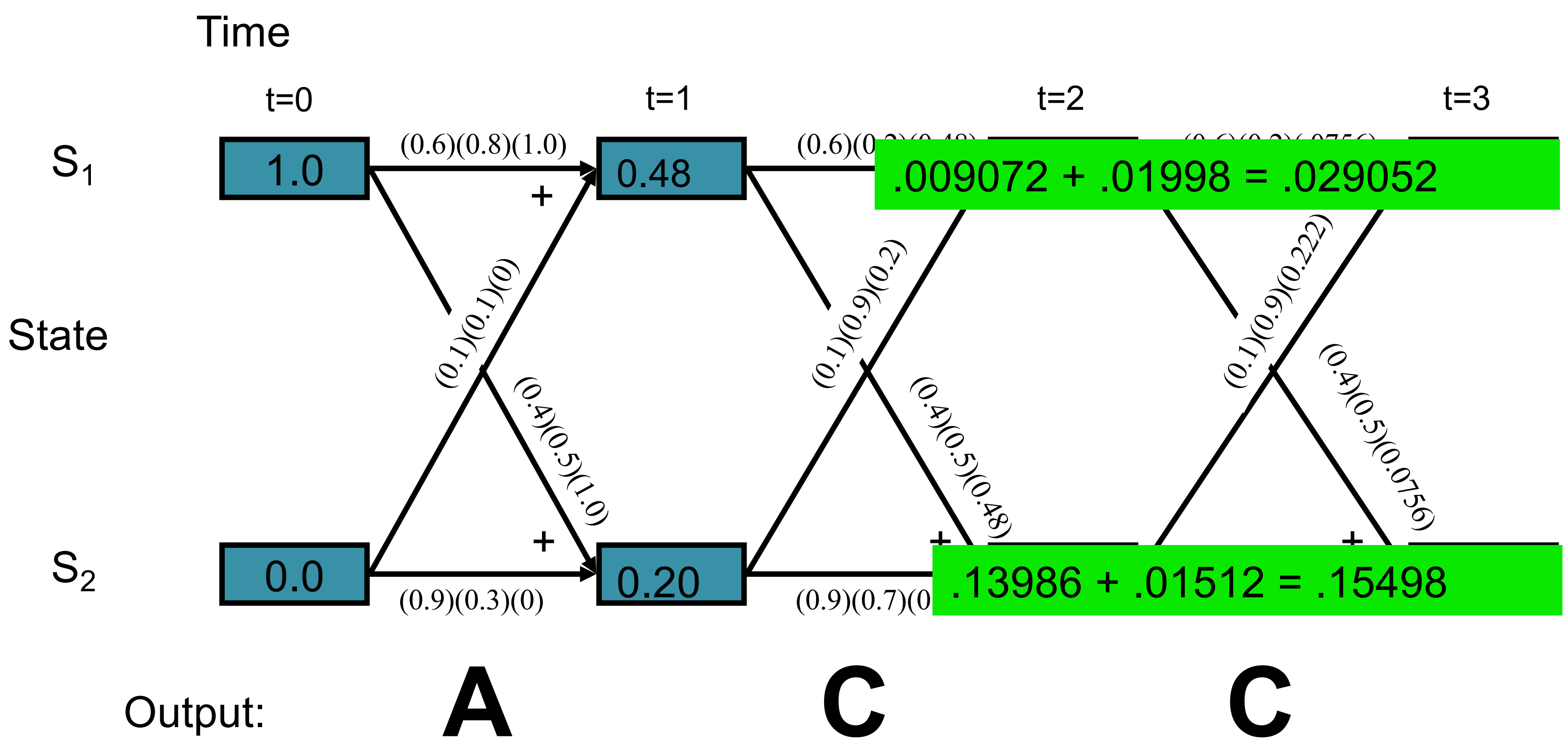
# A trellis for the Forward Algorithm



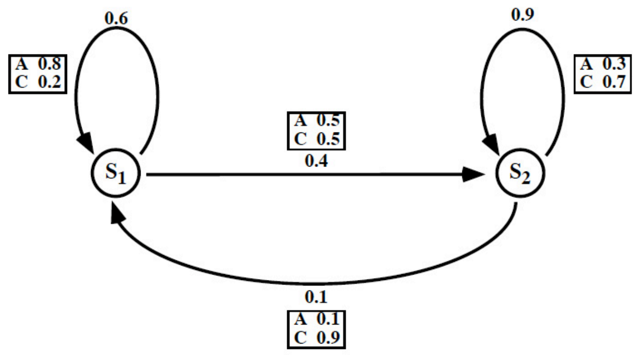


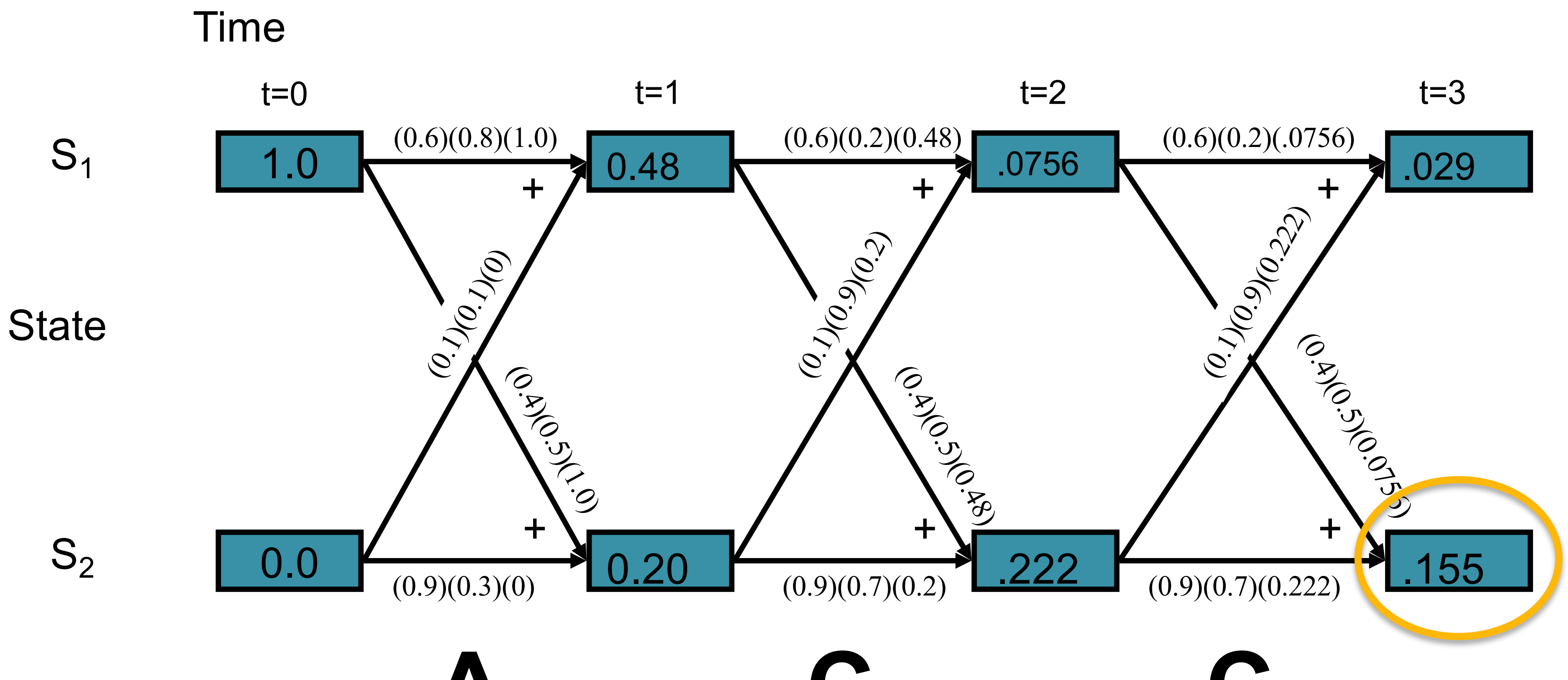
# A trellis for the Forward Algorithm





# A trellis for the Forward Algorithm





S2 is final state➔ 15.5% probability of this sequence given this model was used

# Probability of the model

- The Forward algorithm computes *P(y|M)*
- If we are comparing two or more models, we want the likelihood that each model generated the data: *P(M|y)*
  - Use Bayes' law: $$P(M \mid y) = \frac{P(y \mid M)P(M)}{P(y)}$$
  - Since P(y) is constant for a given input, we just need to maximize *P(y|M)P(M)*

# Three classic HMM problems

2. **Decoding**: given a model and an output sequence, what is the most likely state sequence through the model that generated the output?

- A solution to this problem gives us a way to match up an observed sequence and the states in the model.

AAAGCATGCATTTAACGAGAGCACAAGGGCTCTAATGCCG

The sequence of states is an annotation of the generated string – each nucleotide is generated in intergenic, start/stop, coding state

# Three classic HMM problems

2. **Decoding**: given a model and an output sequence, what is the most likely state sequence through the model that generated the output?

- A solution to this problem gives us a way to match up an observed sequence and the states in the model.

AAAGC ATG CAT TTA ACG AGA GCA CAA GGG CTC TAA TGCCG

The sequence of states is an annotation of the generated string – each nucleotide is generated in intergenic, start/stop, coding state

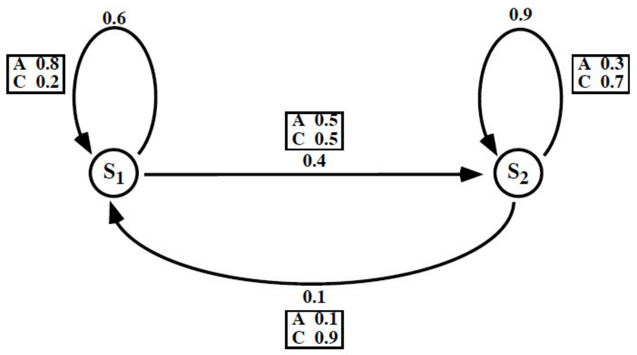
# Solving the Decoding Problem: The Viterbi algorithm

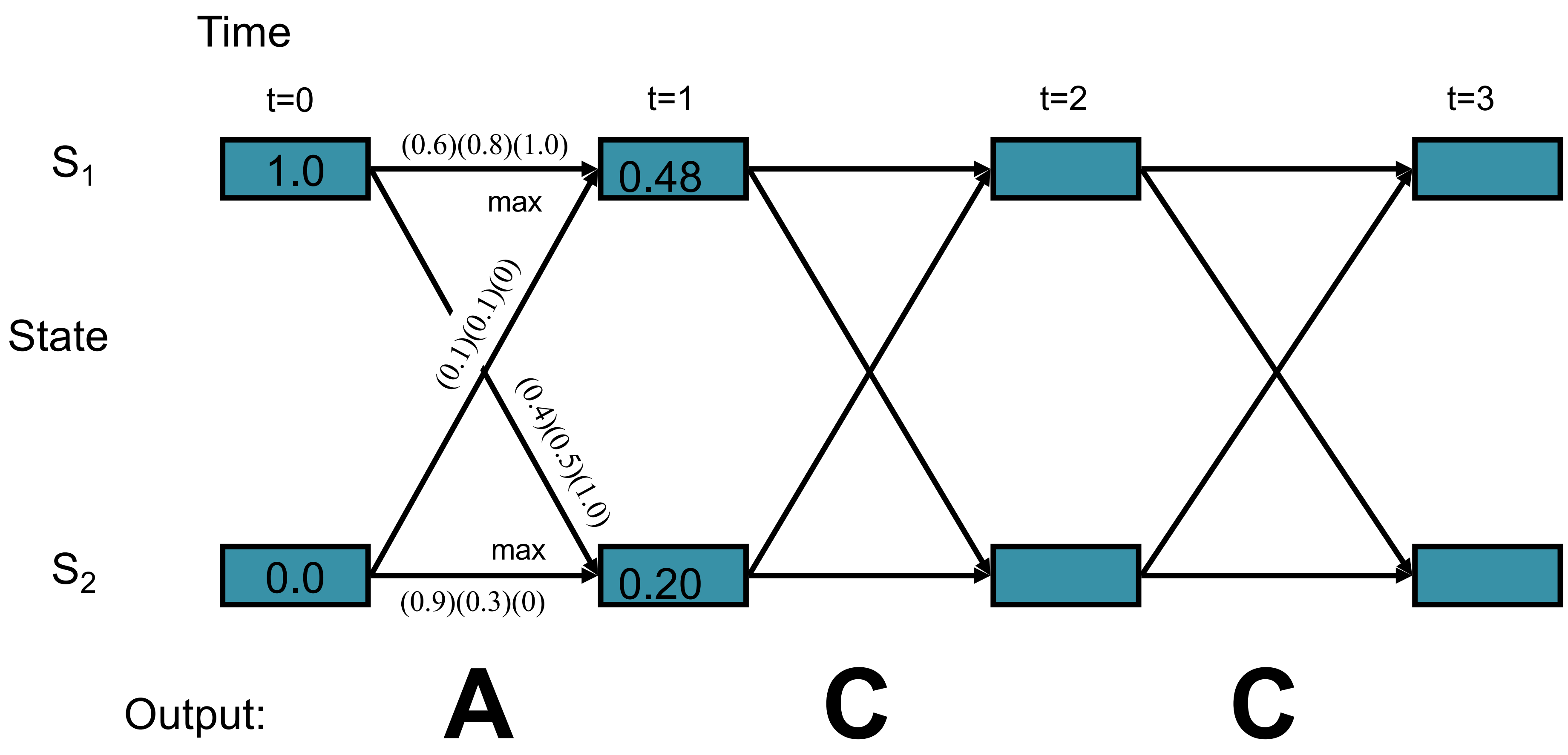
- To solve the decoding problem (find the most likely sequence of states), we evaluate the Viterbi algorithm

$$V_i(t) = \begin{cases} 0 & : \quad t = 0 \wedge i \neq S_I \\ 1 & : \quad t = 0 \wedge i = S_I \\ \max V_j(t-1) a_{ji} b_{ji}(y) & : \quad t > 0 \end{cases}$$

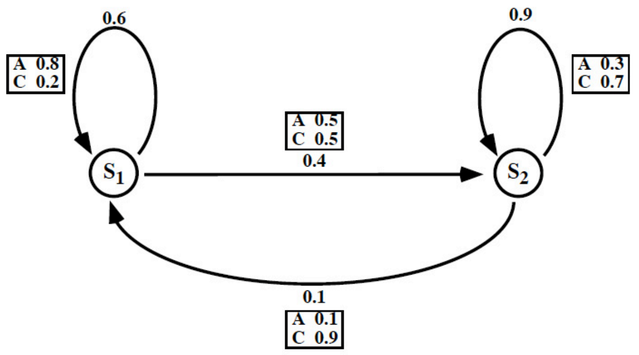
Where $V_i(t)$ is the probability that the HMM is in state $i$ after generating the sequence $y_1, y_2, \ldots, y_t$, following the *most probable path* in the HMM

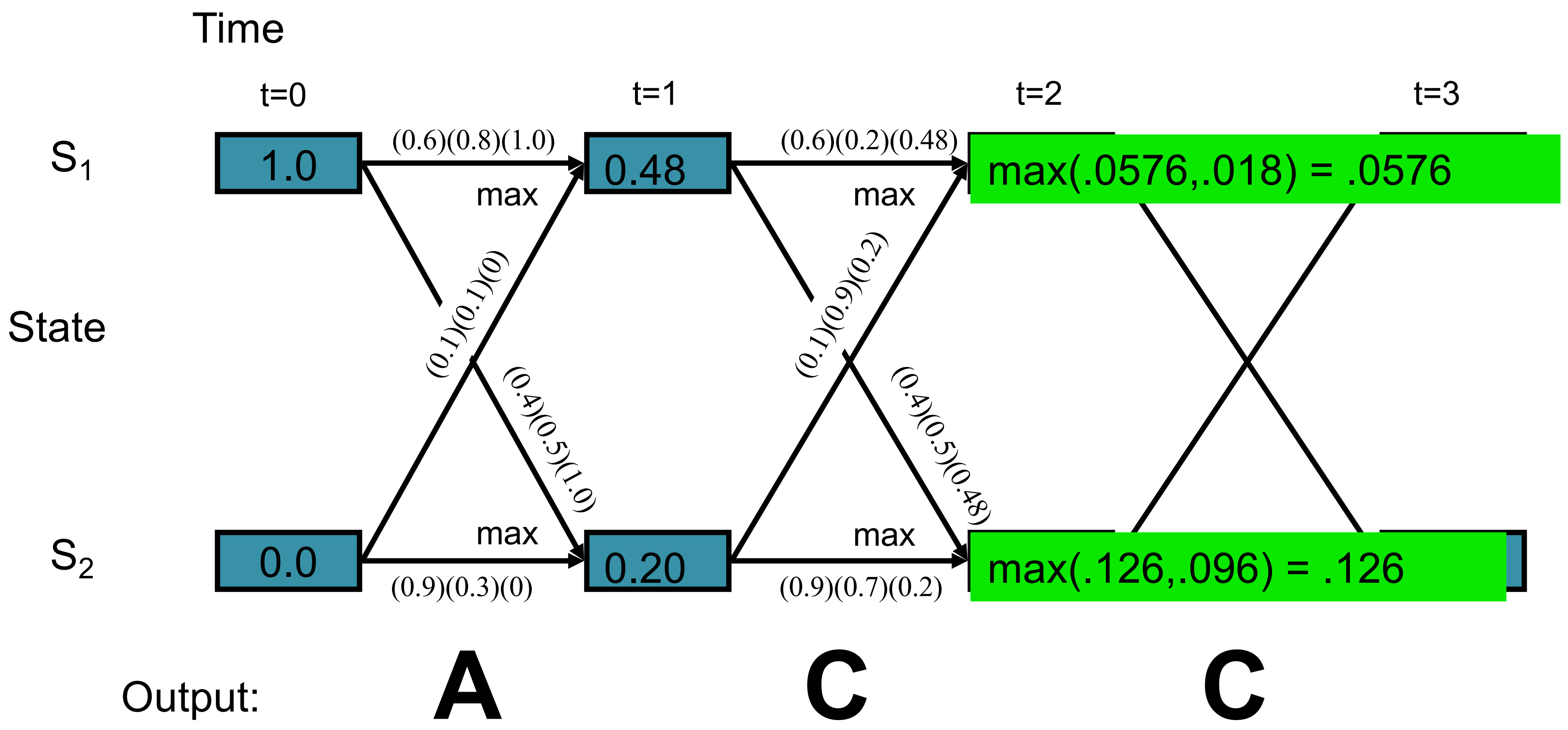
# A trellis for the Viterbi Algorithm



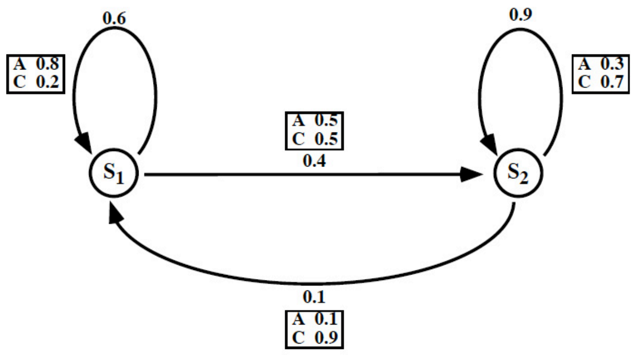


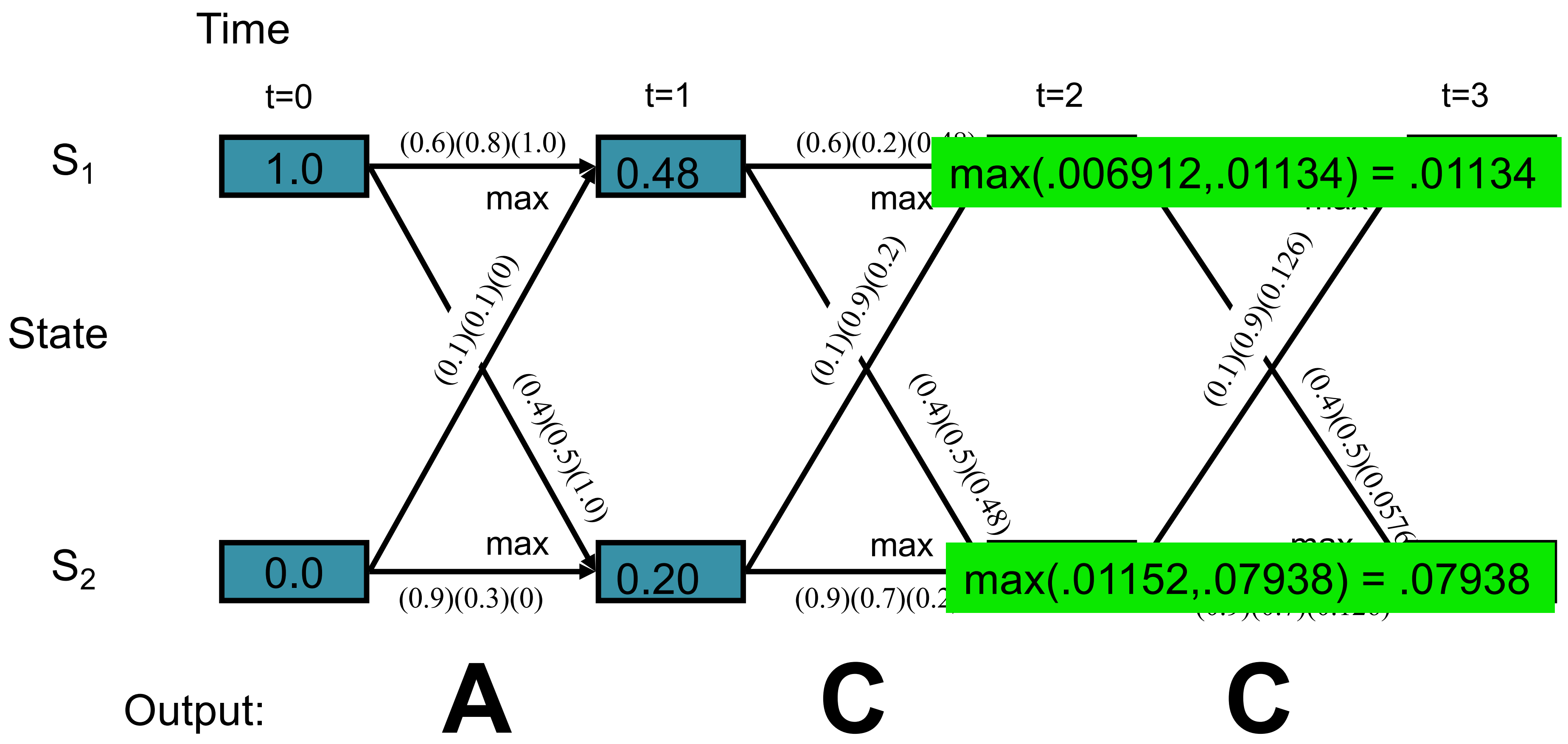
# A trellis for the Viterbi Algorithm



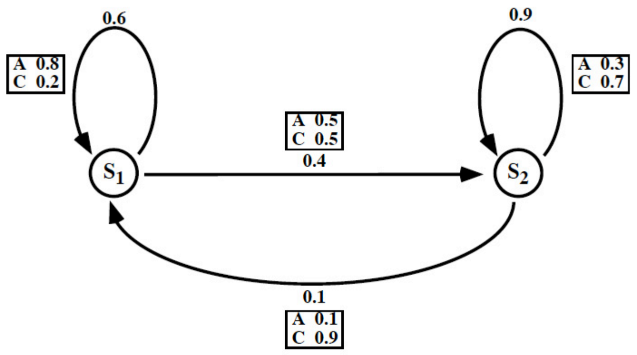


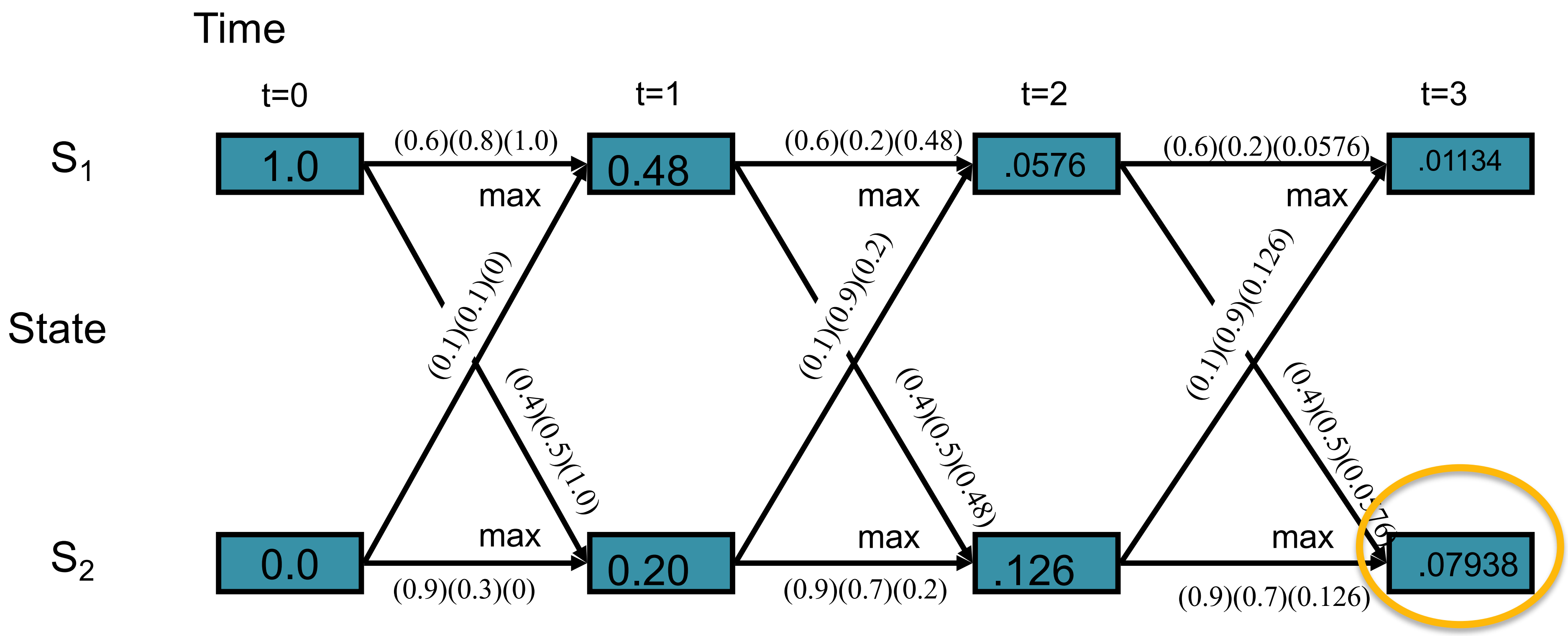
# A trellis for the Viterbi Algorithm





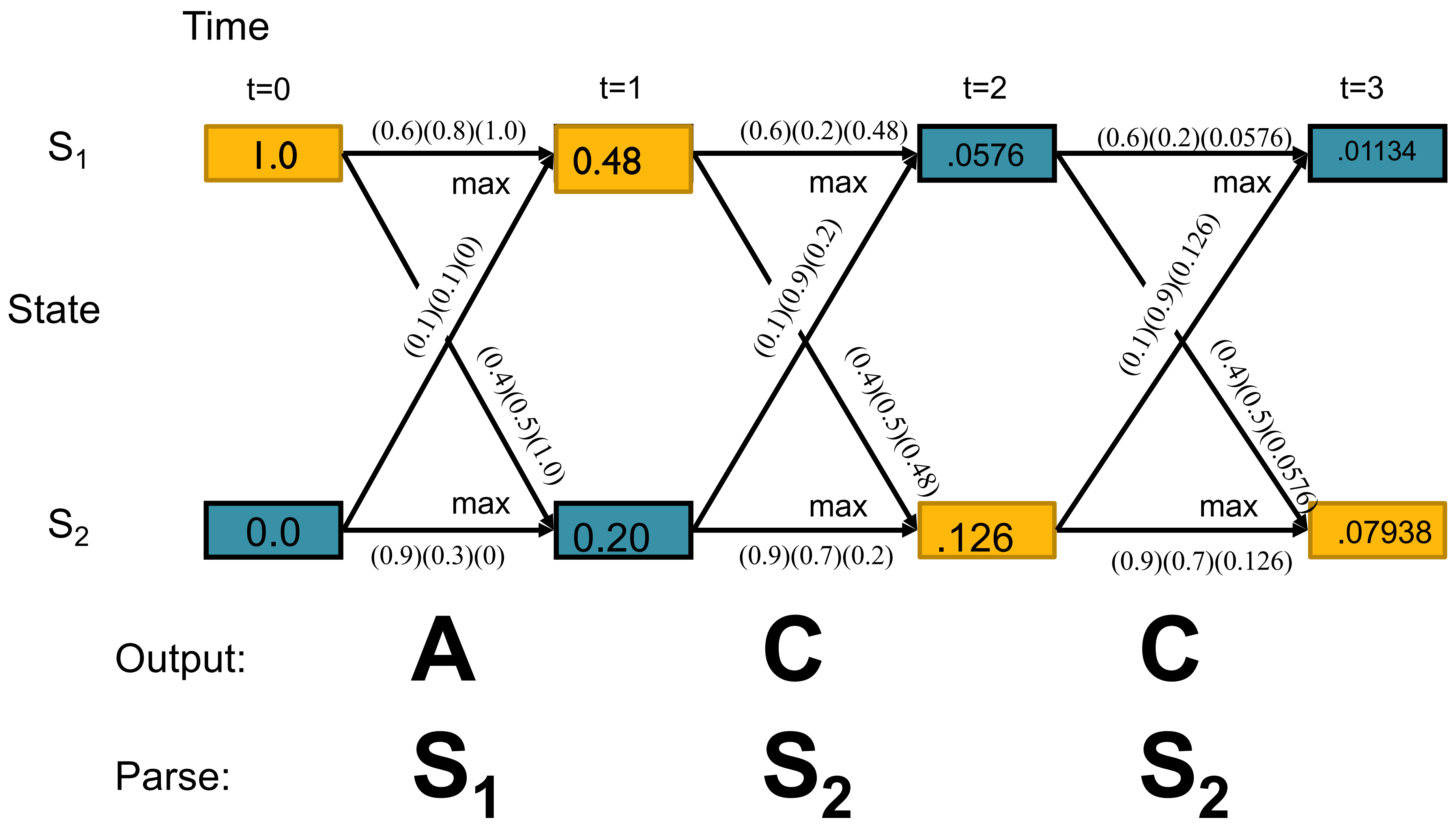
# A trellis for the Viterbi Algorithm





S2 is final state➔ the most probable sequence of states has a 7.9% probability

# A trellis for the Viterbi Algorithm



## Three classic HMM problems

3. **Learning**: given a model and a set of observed sequences, how do we set the model's parameters so that it has a high probability of generating those sequences?

- This is perhaps the most important, and most difficult problem.

- A solution to this problem allows us to determine all the probabilities in an HMMs by using an ensemble of training data

## Learning in HMMs:

- The learning algorithm uses Expectation-Maximization (E-M)
  - Also called the Forward-Backward algorithm
  - Also called the Baum-Welch algorithm
- In order to learn the parameters in an "empty" HMM, we need:
  - The topology of the HMM
  - Data - the more the better
  - Start with a random (or naïve) probability, repeat until converges