Training Tied-State Models

Rita Singh and Bhiksha Raj

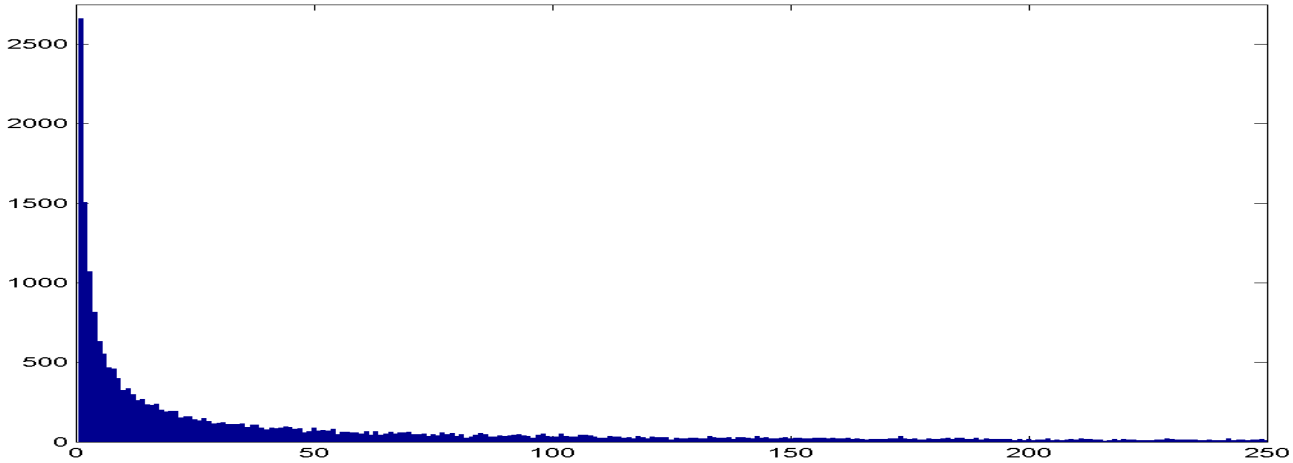
Recap and Lookahead

- Covered so far:
  - String Matching based Recognition
  - Introduction to HMMs
  - Recognizing Isolated Words
  - Learning word models from continuous recordings
  - Building word models from phoneme models
  - Context-independent and context-dependent models
  - Building decision trees

  - Exercise: Training phoneme models
  - Exercise: Training context-dependent models
  - Exercise: Building decision trees

- Training tied-state models

Data Insufficiency Remains a Problem



"Count of counts" histogram for the 24979 triphones in 1.5 million words of Broadcast News

- Most triphones are never seen
  - 58% of triphones are not seen in this corpus
- Most of the rest are seen too infrequently to build good models for them
  - 86% of all triphones are seen less than 10 times
- Problems:
  - How to build models for triphones that *are* seen in training data
  - What do we do about unseen triphones

Why Unseen Triphones Are a Problem

- Word sequences in a test utterance will often need triphones that were not seen in the training data

- Hypothetical example:
  - We never had a word ending with "*Z*" followed by a word beginning with "*S*" in our training data.
  - The test recording is "*RECOGNIZE SPEECH*"
  - Do not have the necessary model components to compose the test word sequence
  - It cannot be recognized

Solutions

- Backoff
  - Instead of triphone “S(Z,P)” for “*Recognize Speech”* use model for context independent phoneme “S”
- Replacement by perceived similarity
  - Use models for the closest sound unit instead
  - E.g. use a triphone for “S(S,P)” instead of “S(Z,P)” in *“Recognize Speech”*
- Clustering and Parameter Sharing: HMMs for many different sound units have the same parameter
  - Decree (based on similarity) that different sounds have similar probability distributions
  - Permits pooling of data to get larger data sets
- Prediction of units by parts
  - Compose HMMs based on clustering and similarity
  - Uses Decision Trees

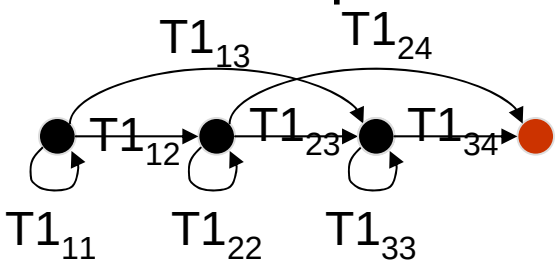
Parameter Sharing

- An HMM for a phoneme has the following parameters:
  1. A set of initial state probabilities π(phoneme)
     - Sometimes denoted by transition probabilities from a generating state
  2. A set of transition probabilities $\mathbf{T}_{\text{phoneme}} = \{T_{\text{phoneme}}(s_i, s_j)\}$
  3. A set of state output distributions, one for every state in the HMM: $P_{\text{phoneme}}(O|\, s_i)$

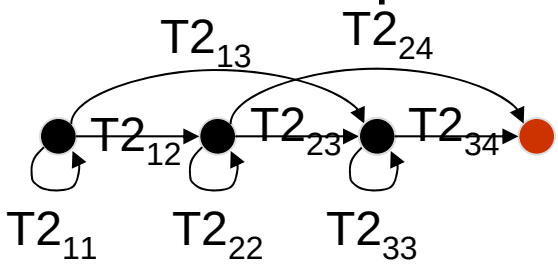
- Two HMMs are said to share parameters if any of these parameters is identical for both. E.g. if
  - π(phoneme1) = π(phoneme2) OR
  - $\mathbf{T}_{\text{phoneme1}} = \mathbf{T}_{\text{phoneme2}}$ OR
  - $P_{\text{phoneme2}}(O|\, s_i) = P_{\text{phoneme2}}(O|\, s_j)$

Parameter Sharing: Transition Matrix

HMM for triphone 1



HMM for triphone 2

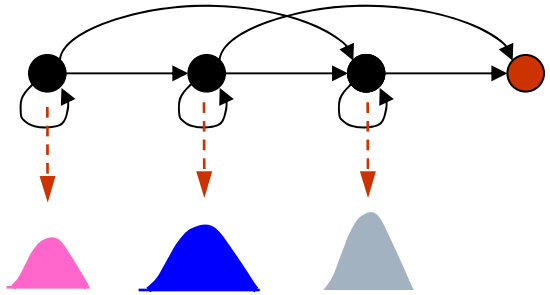


- Sharing Transition Matrices:
  - $T1_{11} = T2_{11}$, $T1_{12} = T2_{12}$, $T1_{13} = T2_{13}$
  - $T1_{22} = T2_{22}$, $T1_{23} = T2_{23}$, $T1_{24} = T2_{24}$
  - $T1_{33} = T2_{33}$, $T1_{34} = T2_{34}$

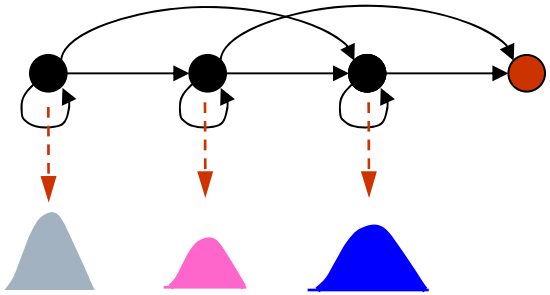
- Transition counts from both triphones will be combined to compute transition probabilities

Parameter Sharing: State O/P PDF

HMM for triphone 1



HMM for triphone 2



- Sharing State Output Probabilities: The same probability densities are shared by states of different HMMs
  - Indicated by colour in the figure
  - To train the density data belonging to the states of all HMMs sharing it are pooled
    - E.g. data from the first state of the HMM for triphone1 and the second state of the HMM for triphone2 would be pooled to learn the parameters of the shared distribution

Parameter Sharing Mechanisms

- Parameter sharing can be effectively used to train models for N-phones for which we have very few training instances
- Common sharing mechanisms:
- Phonemes: Share state output distributions across different phonemes
  - E.g., the central portions of ZH and SH may share state output distributions
  - i.e. if we model ZH and SH with 5-state HMMs, then we set $P_{ZH}(O|\ s_2) = P_{SH}(O|\ s_2)$
- Diphones: Share state output distributions across different diphones
  - E.g. $P_{AX\_D}(O|\ s_0) = P_{AX\text{-}T}(O|\ s_0)$; $P_{AX\_D}(O|\ s_4) = P_{EH\text{-}D}(O|\ s_4)$
- Triphones: Share transition matrices and state output distributions
  - All triphones of the form AX(*,*) have the same transition matrix
  - $P_{AX(B,D)}(O|\ s_0) = P_{AX(P,T)}(O|\ s_0)$ etc.

Advantages of Parameter Sharing

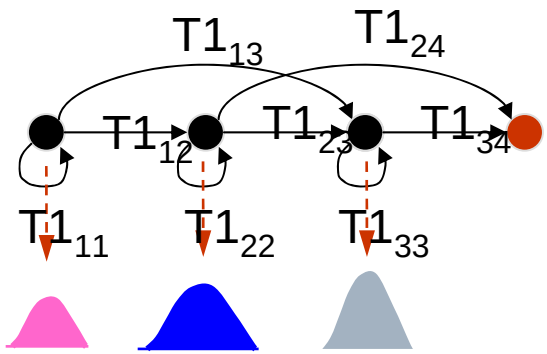
- Parameter sharing can be used to alleviate data insufficiency problems
  - E.g. We have very few instances of phoneme ZH
  - But SH occurs very frequently
  - We have decided to share $P_{ZH}(O|\ s_2) = P_{SH}(O|\ s_2)$
  - We train the shared output distribution from data from the central regions of all instances of both ZH and SH
  - This gives us enough data to learn the state output distribution of s_2 in ZH properly
- By appropriately sharing other states, we can learn all the parameters of the HMM of ZH even though ZH itself has little data
- Similar sharing mechanisms can be used to learn good models for diphones and triphones that are poorly represented in the training data
- All of this depends on the validity of the sharing assumptions
  - E.g. $P_{ZH}(O|\ s_2) = P_{SH}(O|\ s_2)$

Parameter Sharing Criterion

- How do we decide what parameters to share?
- Ad-hoc reasoning is not likely to be effective
  - May not be supported by data
- The most common approach is to base it on empirically determined similarity
  - Train a model for SH from whatever data we have
  - Train a model for ZH from available data
  - Compare the output distributions of the second state of ZH to the output distributions of every state of SH
  - "Tie" $P_{ZH}(O|\, s_2)$ to the closest state output distribution of SH
    - This may not be the second state of SH
  - The same mechanism may be used to "tie" the states of various diphones and triphones
- This resolves the issue of learning HMMs for units for which we have only small amounts of training data
- But how do we compute HMMs for units for which we have no data at all?

Predictive Parameter Sharing

- To compose the HMM for a novel unit (phoneme, diphone or triphone), we must determine the following parameters for it:
  - Set of all transition probabilities
    - Initial state probabilities are transitions from a generating state
- Set of all state output distributions



- We will predict these through other known characteristics of the unit
  - Such as the known linguistic characteristics of the phoneme/diphone/triphone

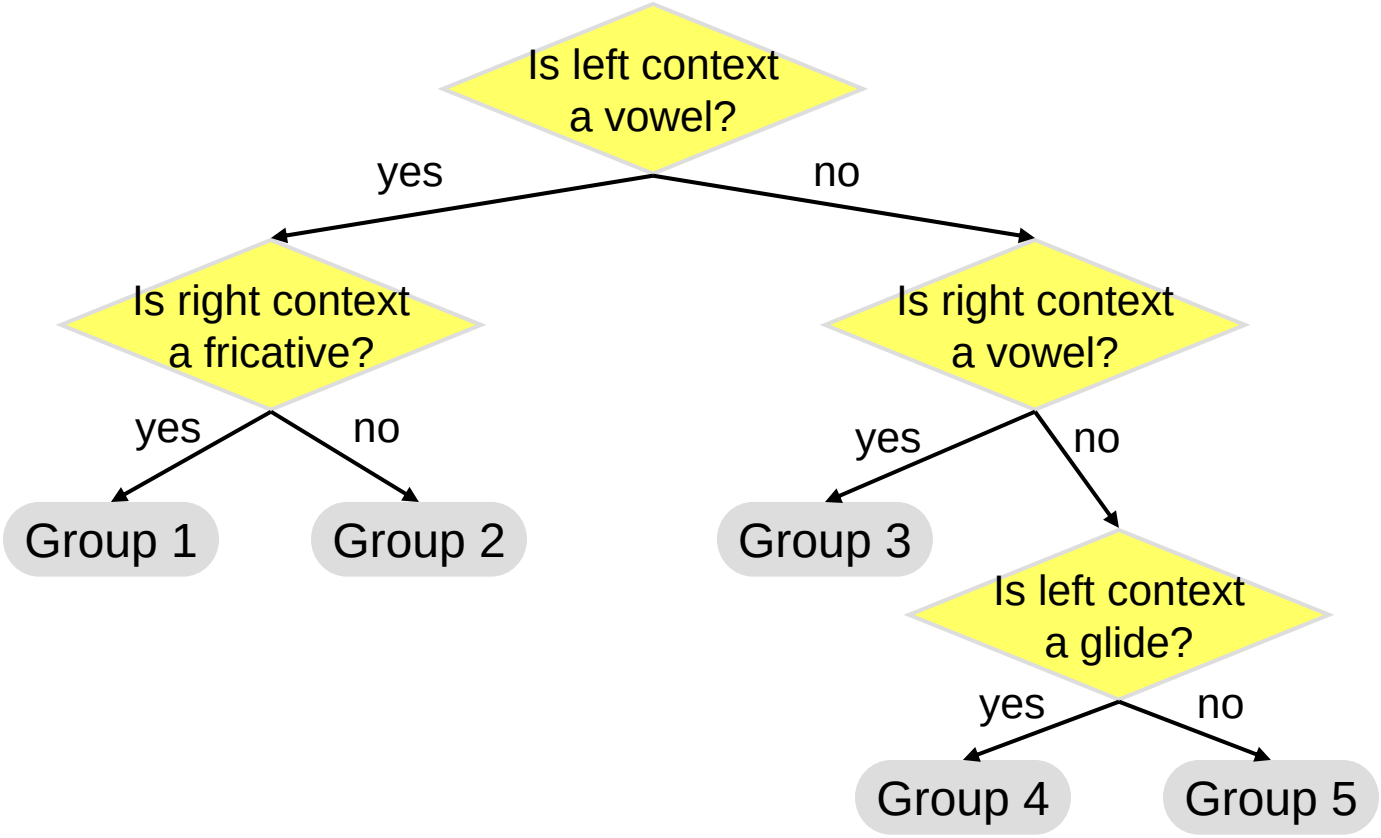
Building HMMs for unseen Phonemes and Triphones

- Phonemes: Use simple linguistic similarity
  - E.g. ZH is known to be a voiced variant of SH. If we do not observe ZH in the training data, copy the entire model for SH into ZH
  - This includes transition matrices and state output distributions

- Triphones:
  - Transition matrix: During training stipulate that all triphones of the same phoneme, e.g. all triphones of the kind AX(*,*) will share the same transition matrix
    $\mathbf{T}_{AX(*,*)} = \mathbf{T}_{AX}$
  - For an unseen new triphone of AX, use this common transition matrix ($\mathbf{T}_{AX}$)
  - To determine state output distributions use *decision trees*

Decision Trees

- A decision tree is literally a tree of decisions
- It is used to cluster triphone units based on “linguistic questions”.
- For example, a decision tree might look like this:



Decision Trees: The Rationale

- Decision trees are prediction mechanisms that identify what any component of a triphone might look like
- The tree itself is built from a combination of data and "expert" knowledge about the phonetics of the language
  - Although this expert knowledge is not essential and may also be derived from data
- To explain the process of building decision trees, let us briefly revisit the training process…

Training acoustic models

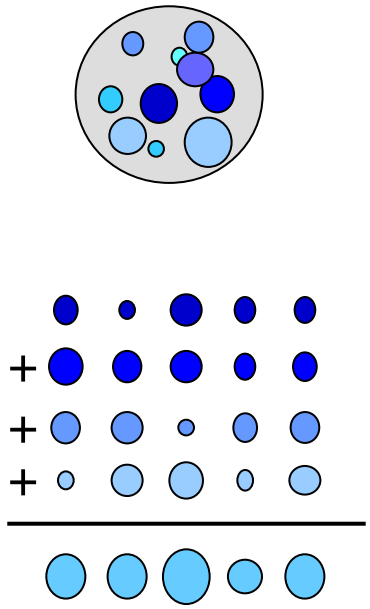
Dictionary

Eight: ey t
Four: f ow r
One: w a n
Zero: z iy r ow
Five: f ay v
Seven: s e v e n
Enter: e n t e r
Two: t uw

All words specified in terms of phonemes

- Phoneme and N-phone models begin by representing all words in terms of phonemes, like in the dictionary above

CI models



- Each instance of the phoneme is (soft) “segmented” in to Nstates parts
- All data in a k-th part are aggregated to compute the distribution of the k-th state

- Training involves grouping data from phonemes followed by parameter estimation
- Indiscriminate grouping of vectors of a unit from different locations in the corpus results in Context-Independent (CI) models
- Explicit boundaries (segmentation) of phonemes not available
- Explicit boundaries are not needed
- Data for individual states obtained through soft decisions

Context Dependency

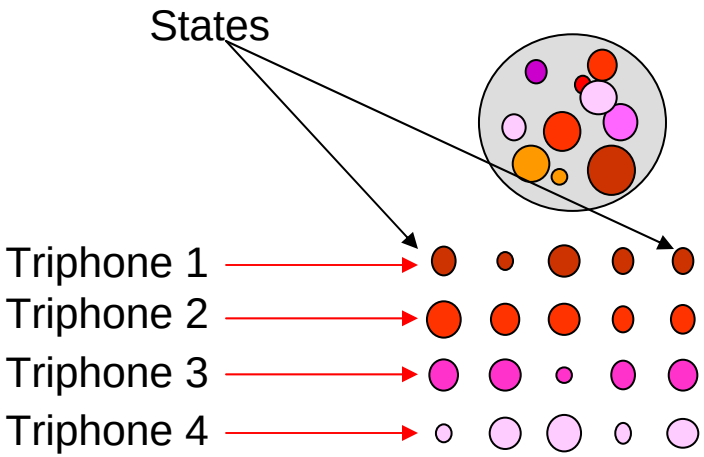
Dictionary

Eight: ey t
Four: f ow r
One: w a n
Zero: z iy r ow
Five: f ay v
Seven: s e v e n
Enter: e n t e r
Two: t uw

All instances of a subword unit in a particular context can be treated as a single entity

- Context dependent units consider the neighbors
  - The two “OW”s above are different if context is considered

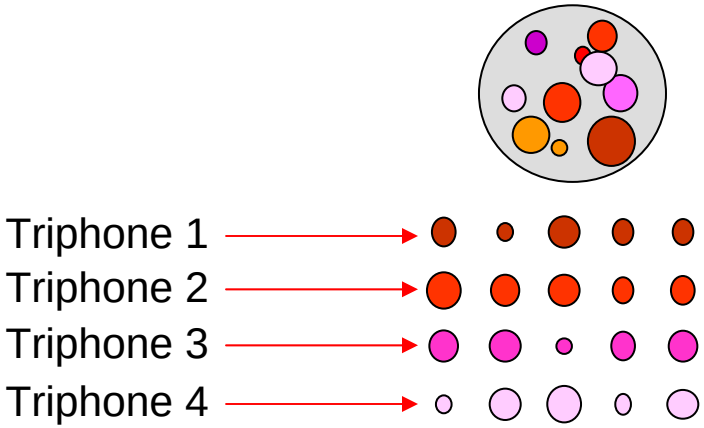
Context dependent (triphone) models



- Context based grouping of observations results in finer, Context-Dependent (CD) models

- The big ball represents all instances of a phoneme
- Each row represents a phoneme in a particular context
  - Each ball represents *all* data from a specific state of the context-dependent phoneme
  - E.g. if the big ball were “AH”, the first row might represent the data for AH (B, T) (AH in the context of B and T)
  - The small ball in the top left corner would represent all data from the first state of AH(B,T)

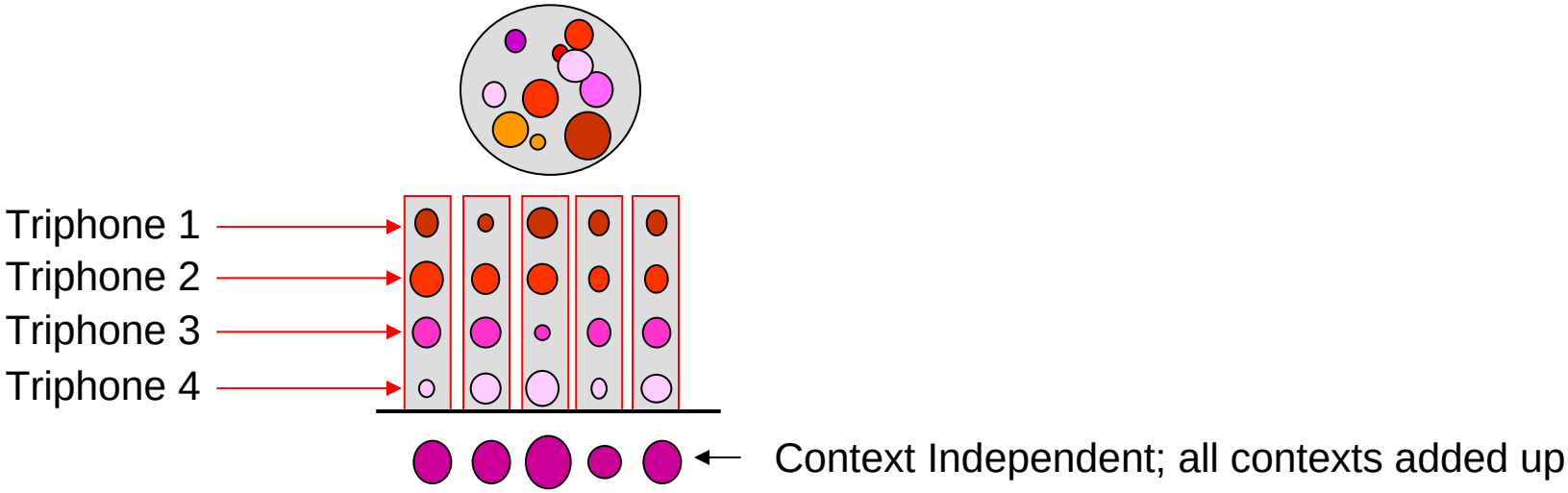
Context dependent (triphone) models



- Context based grouping of observations results in finer, Context-Dependent (CD) models

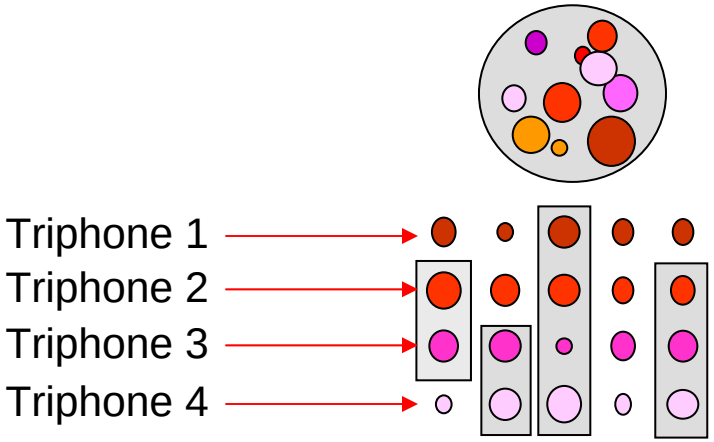
- If a separate HMM were trained for each row, we have separate models for each CD phoneme
  - Data insufficiency problems

Context dependent (triphone) models



- If a separate HMM were trained for each row, we have separate models for each CD phoneme
  - Data insufficiency problems
- If all the data in each column were aggregated to train the distribution for the state we would get a CI model
  - Sufficient data, but eliminates context information – poor models

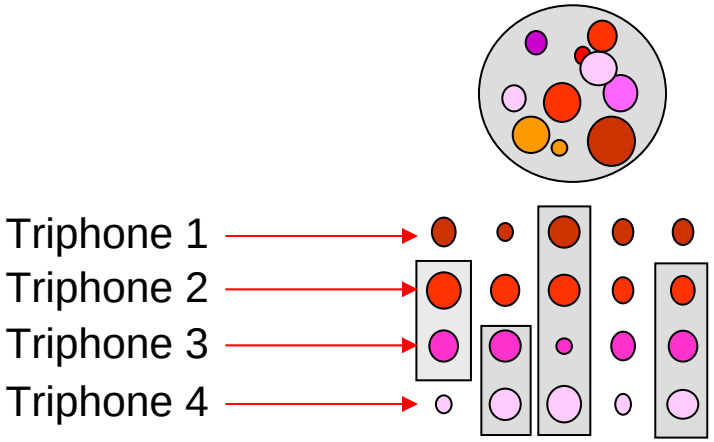
Context dependent (triphone) models



Groups are "tied" states because they "tie" data from the states of different triphones together

- A compromise: Group subsets of data in each column
- Train separate distributions for each group
- Not as coarse as CI models
- Does not suffer the data insufficiency of CD models
  - Each group has sufficient data to learn a good distribution
  - Each group represents a "TIED" state
- Triphones retain identity
  - In terms of the specific sequence of tied states

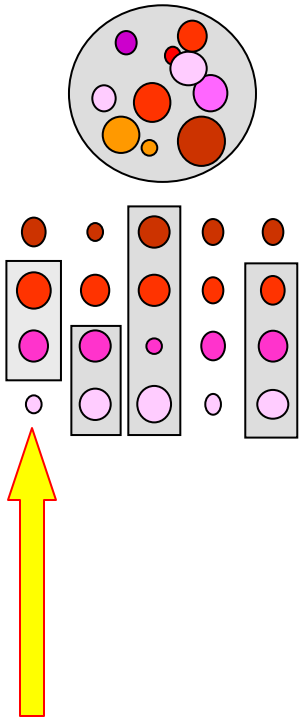
Context dependent (triphone) models



Groups are "tied" states because they "tie" data from the states of different triphones together

- Note: Grouping is within-state
  - E.g. we only group data from the first state of any triphone with data from the *first* state of any other triphone
    - There is a reason for this – the ability to predict new triphones
    - As we will see, this lets us select a distribution for the first state (for example) when building a model for a triphone that was not seen in training
- However, the precise manner in which this grouping is performed is important

Training context dependent (triphone) models: Parameter Sharing

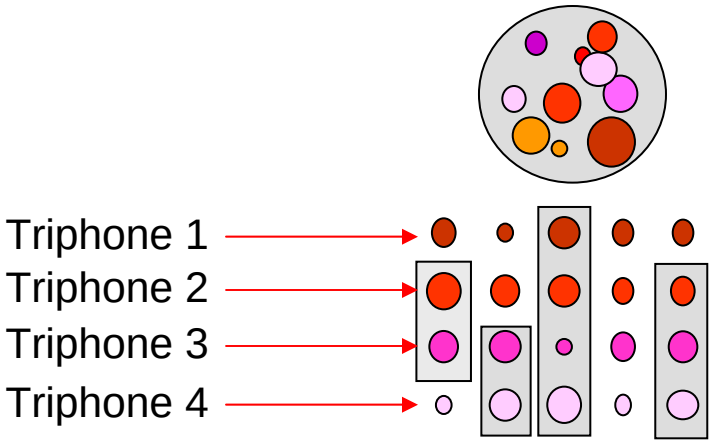


3 separate “groups” for the first state alone

How to determine this grouping?

Do it such that each group is most consistent

Defining Consistency



- We want all the data within any group to be as similar to one another
- We want data from different groups (in the same column) to be dissimilar
- Having this distinction will enable us to distinguish one triphone from another
- Therefore, the objective is to identify the grouping that maximizes within-group similarity and minimizes cross-group similarity
  - A very difficult problem
- For this we need an objective function that captures this notion of consistency

Defining Consistency

- The consistency of a data set can be defined in terms of the *expected log likelihood* of the data on its own distribution
  - Typically we assume this distribution to be Gaussian
  - E.g. if the mean of the data is μ and the variance is C, then the "consistency" score for a single vector is:

$$E\left[\log\left(\frac{1}{\sqrt{2\pi^d|C|}}e^{-0.5(x-\mu)^T C^{-1}(x-\mu)}\right)\right]$$

    - Here "E[]" is an expectation operator computed against the Gaussian distribution itself
- For a set with *N* data points this can be shown to be simply:

$$-0.5Nd - 0.5N\log\left(2\pi^d|C|\right)$$

    - *d* is the dimensionality of the vectors in the data
  - This is only a function of the covariance of the data and *N*

Improvement of Consistency From Splitting a data set

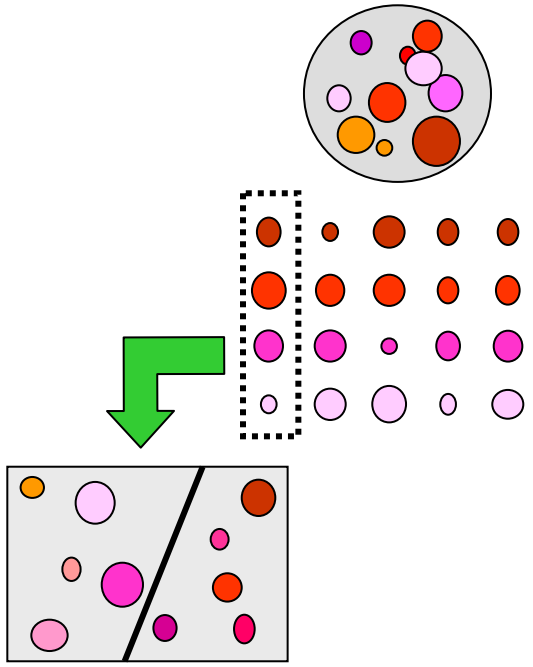
- If a set of N data points is split into two sets of size N_1 and N_2, such that $N = N_1 + N_2$, with variances C_1 and C_2, then each of the resultant data sets has its own consistency value
  - Original set:
  - Child set no. 1: $-0.5N_1 d - 0.5N_1 \log\left(2\pi^d |C_1|\right)$
  - Child set no. 2: $-0.5N_2 d - 0.5N_2 \log\left(2\pi^d |C_2|\right)$
- This split results in a change in the overall consistency of the sets
  - Consistency of set1 + Consistency of set2 – Consistency of original set

$$\left(-0.5N_1 d - 0.5N_1 \log\left(2\pi^d |C_1|\right)\right) + \left(-0.5N_2 d - 0.5N_2 \log\left(2\pi^d |C_2|\right)\right) - \left(-0.5Nd - 0.5N \log\left(2\pi^d |C|\right)\right)$$

$$=$$

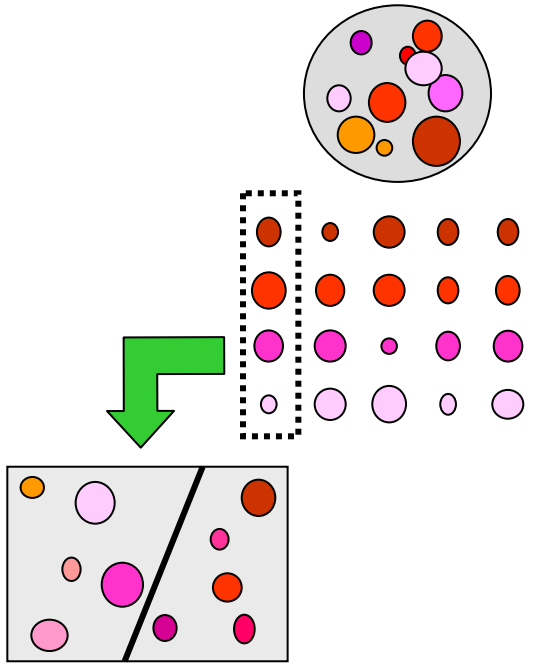
$$N \log\left(2\pi^d |C|\right) - 0.5N_1 \log\left(2\pi^d |C_1|\right) - 0.5N_2 \log\left(2\pi^d |C_2|\right)$$

Grouping of context-dependent units for parameter estimation



- Partitioning the data in any column will result in an increase in consistency

Grouping of context-dependent units for parameter estimation



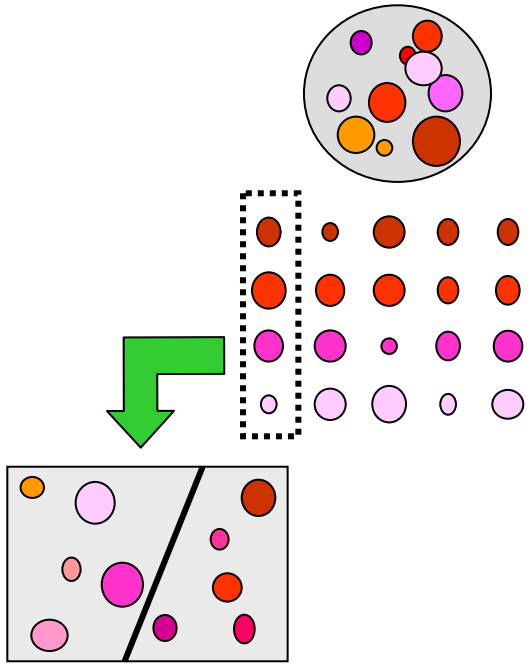
- If we partition a set of ***N*** vectors with mean μ and variance ***C*** into two sets of vectors of size N_1 and N_2, with means μ_1 and μ_2 and variances C_1 and C_2 respectively, the total expected log-likelihood of the vectors after splitting becomes

$$-0.5N_1 d - 0.5N_1 \log\left(2\pi^d |C_1|\right) - 0.5N_2 d - 0.5N_2 \log\left(2\pi^d |C_2|\right)$$

- The total log-likelihood has increased by

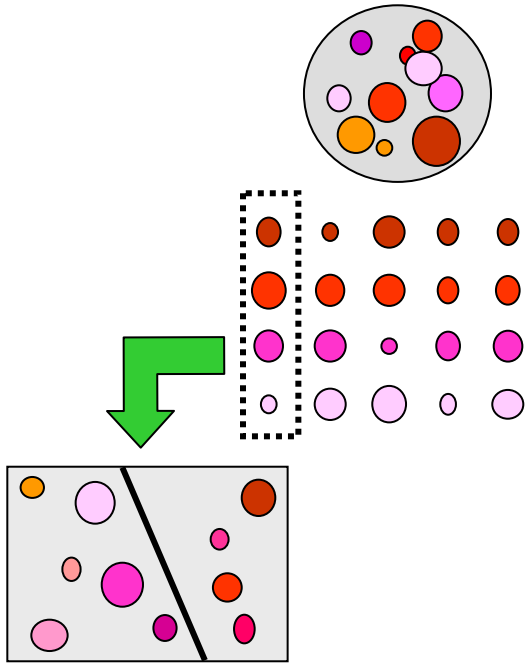
$$N \log\left(2\pi^d |C|\right) - 0.5N_1 \log\left(2\pi^d |C_1|\right) - 0.5N_2 \log\left(2\pi^d |C_2|\right)$$

Grouping of context-dependent units for parameter estimation



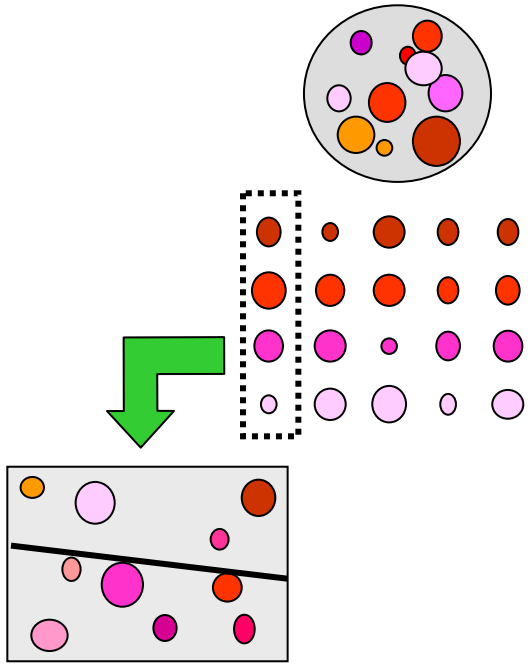
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)

Grouping of context-dependent units for parameter estimation



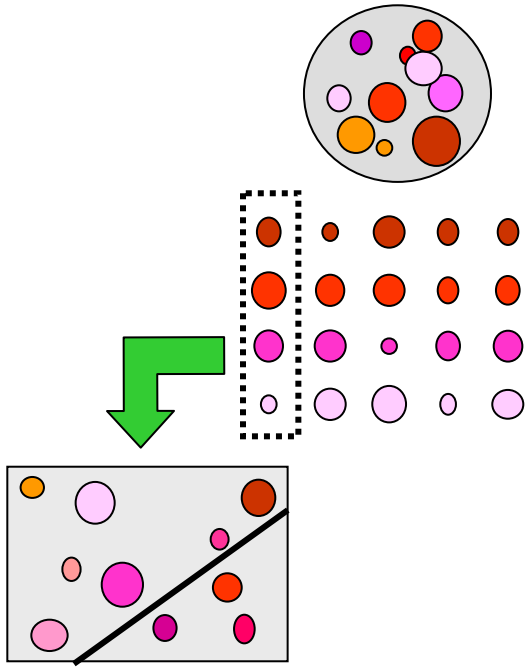
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)

Grouping of context-dependent units for parameter estimation



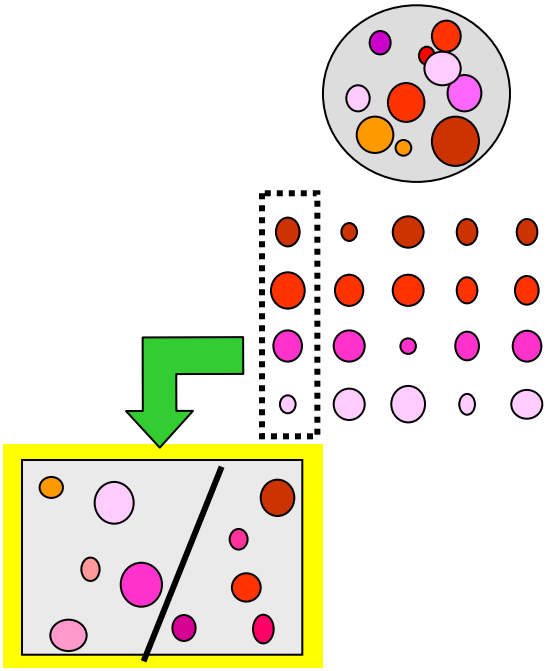
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)

Grouping of context-dependent units for parameter estimation



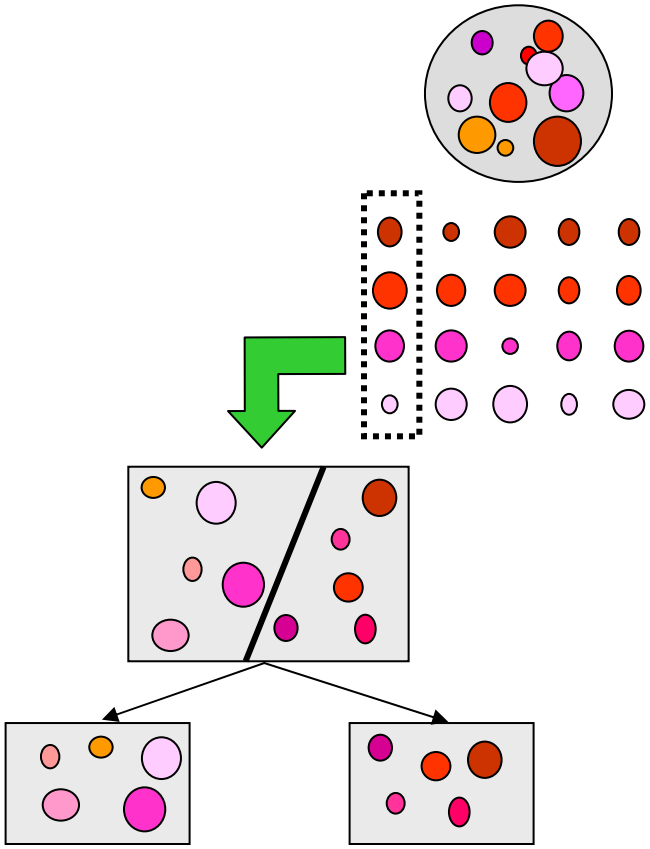
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)

Grouping of context-dependent units for parameter estimation



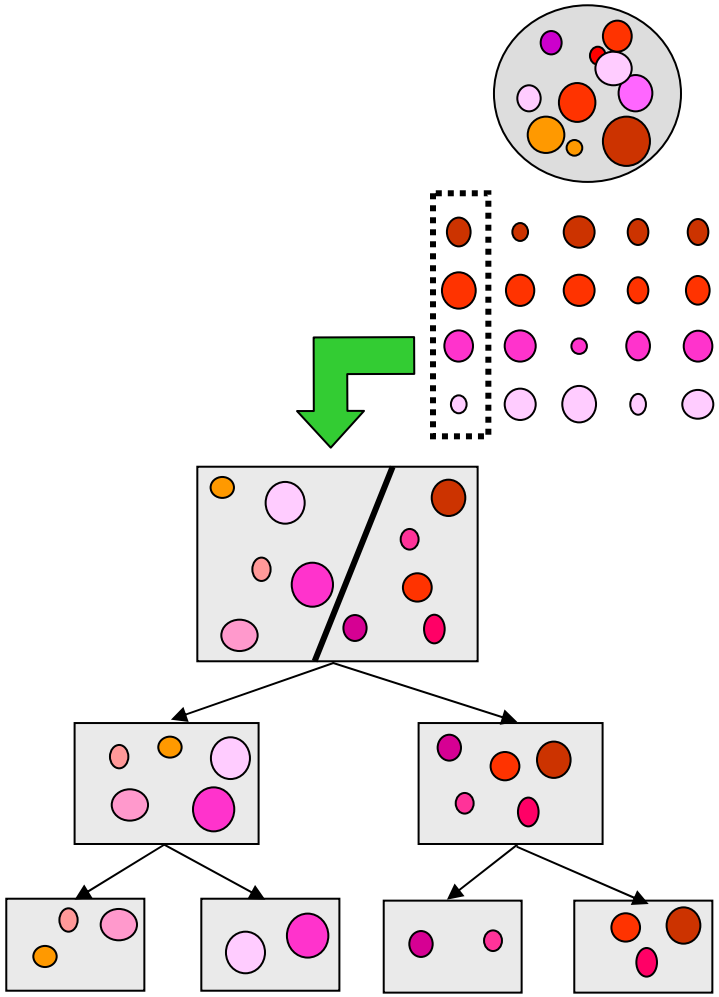
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)

Grouping of context-dependent units for parameter estimation



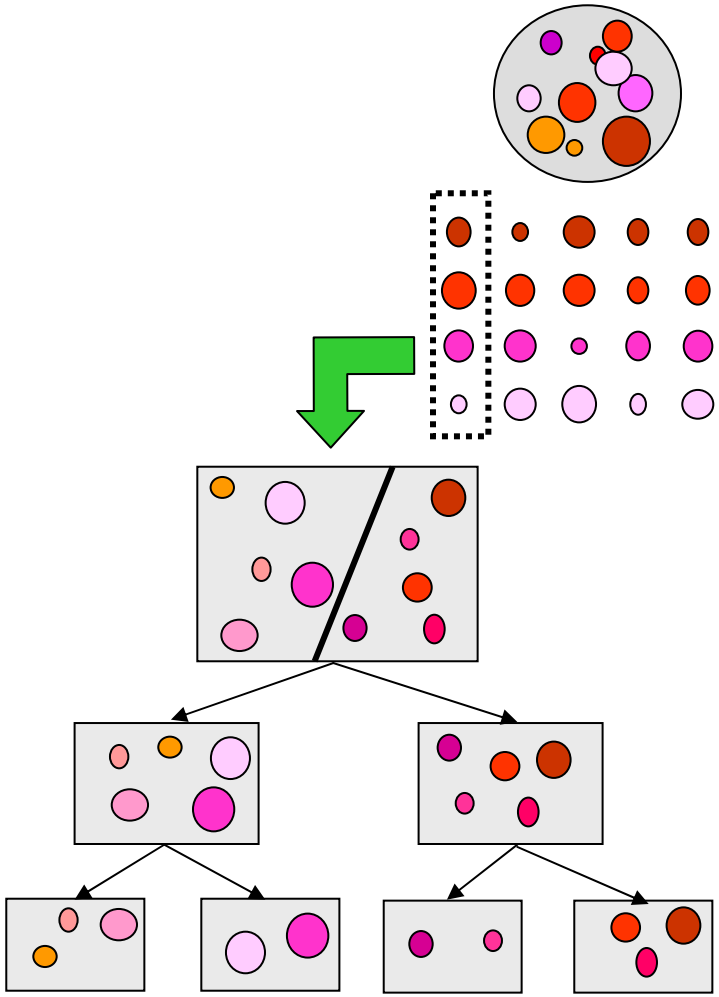
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)
- Recursively partition the sets in the same manner

Grouping of context-dependent units for parameter estimation



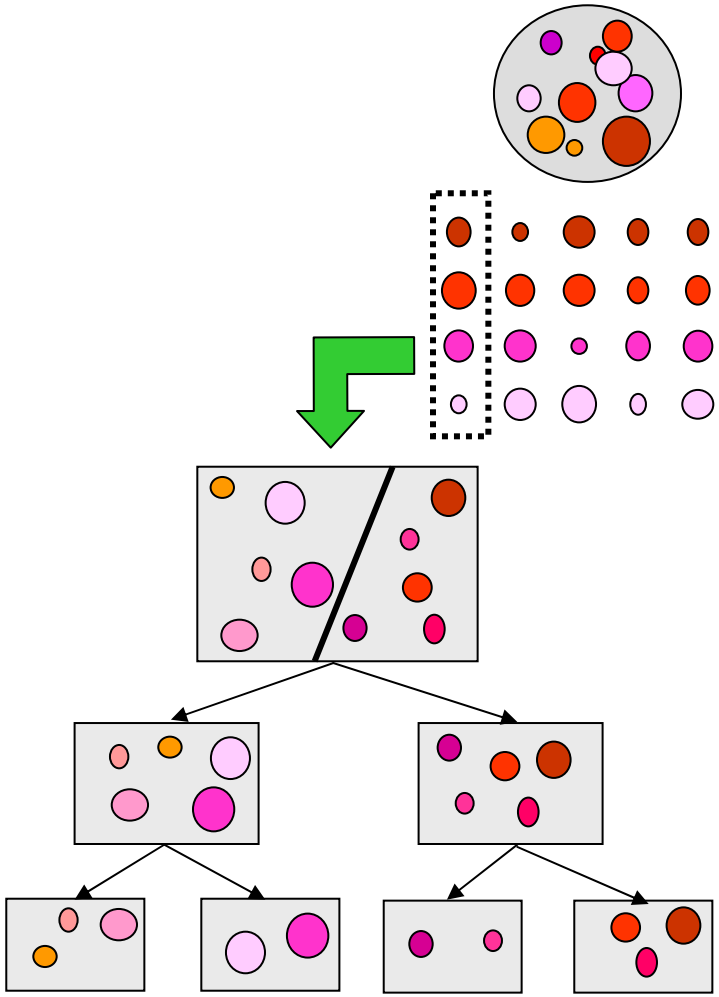
- Observation vectors partitioned into groups to maximize within class likelihoods
- Evaluate partitions until we identify the partition that results in the most increase in consistency (likelihood)
- Recursively partition the sets in the same manner

Grouping of context-dependent units for parameter estimation



- Observation vectors partitioned into groups to maximize within class likelihoods
- Recursively partition vectors into a complete tree
- **The leaves of this tree will represent families of triphones with the most similar data**

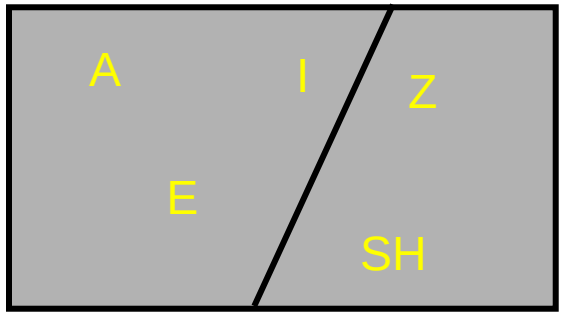
Partitioning the data



- The data at each node must be partitioned so that the children of the node are most internally consistent
- 2^{n-1} possible partitions for n vector groups. Exhaustive evaluation too expensive
- Exhaustive evaluation may also select “splits” that only capture vagaries of a specific training data
- Instead we only evaluate a smaller number of splits that are based on known good rules for splitting
- These are “linguistic questions”

Linguistic Questions

- Linguistic questions are pre-defined phone classes. Candidate partitions are based on whether a context belongs to the phone class or not
- Linguistic questions must be meaningful in order to deal effectively with unseen contexts
  - Must represent some underlying acoustic grouping



Meaningful Linguistic Questions?

Left context: (A,E,I,Z,SH)

ML Partition: (A,E,I) (Z,SH)

(A,E,I) vs. Not(A,E,I)

(A,E,I,O,U) vs. Not(A,E,I,O,U)

Linguistic Questions

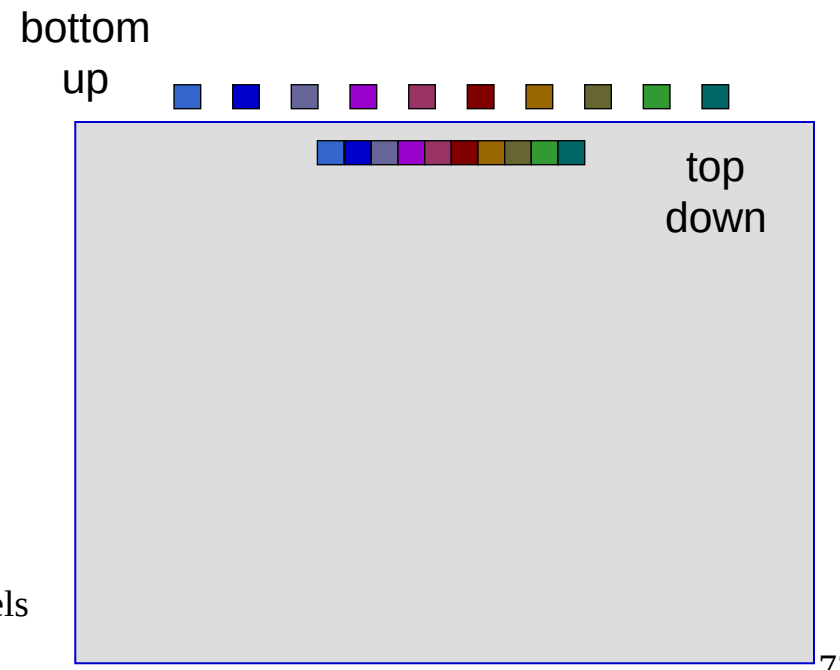
- Must capture known phenomena
- Typically defined by linguists
  - Based on knowledge of sound production and perceptual similarity
  - Sounds produced with similar articulator configurations will have similar spectral structure
    - Laws of physics
  - Linguistic questions attempt to identify groupings that will predict similarity in spectral structure
    - E.g. “[VOWELS]”, or “[SH ZH]” or “[L W R]”
    - Groupings with similar production mechanisms and spectral similarities
- Linguistic questions can also be automatically deduced if required
  - Since the goal is to identify spectral similarity

Automatic Generation of "Linguistic" Questions

- Attempt to deduce various groupings of phonemes that are spectrally similar
- We need *multiple* such groupings
- Techinque:
  - Train CI models for all phonemes
  - Group phonemes based on the similarity of the state output distributions of their CI models
- Grouping method that produces multiple groupings: Any clustering method would work
- However, an optimal clustering technique results in better "questions"
- For this we can use the hybrid bottom-up-top-down technique described next

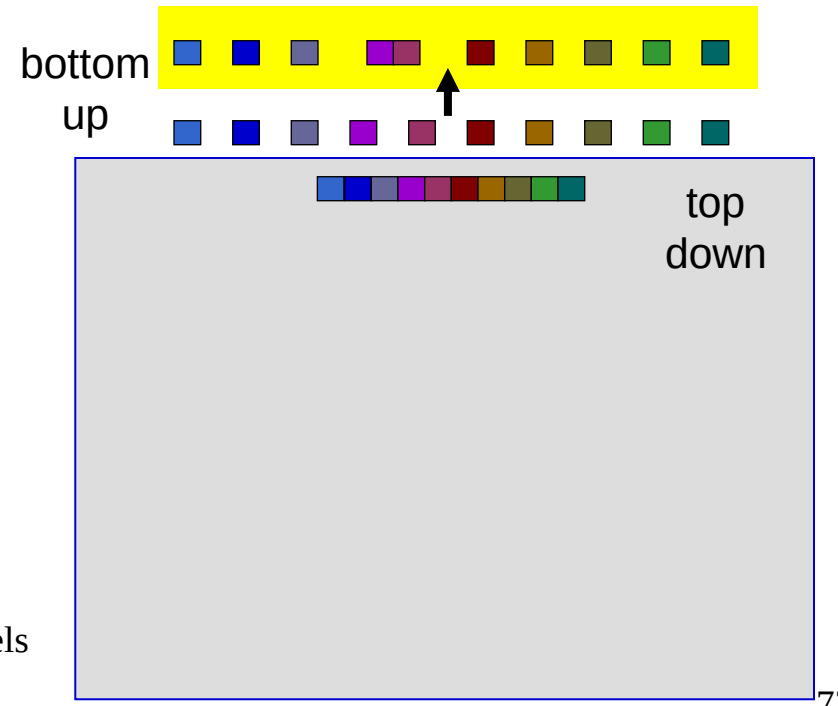
Hybrid Procedure - I

1. Begin by maintaining all CI phone states as separate entities (labelled bottom-up in figure)
   - Simutaneously group all of them together (labelled top-down)
   - Note, we only use one of the states from each CI phone



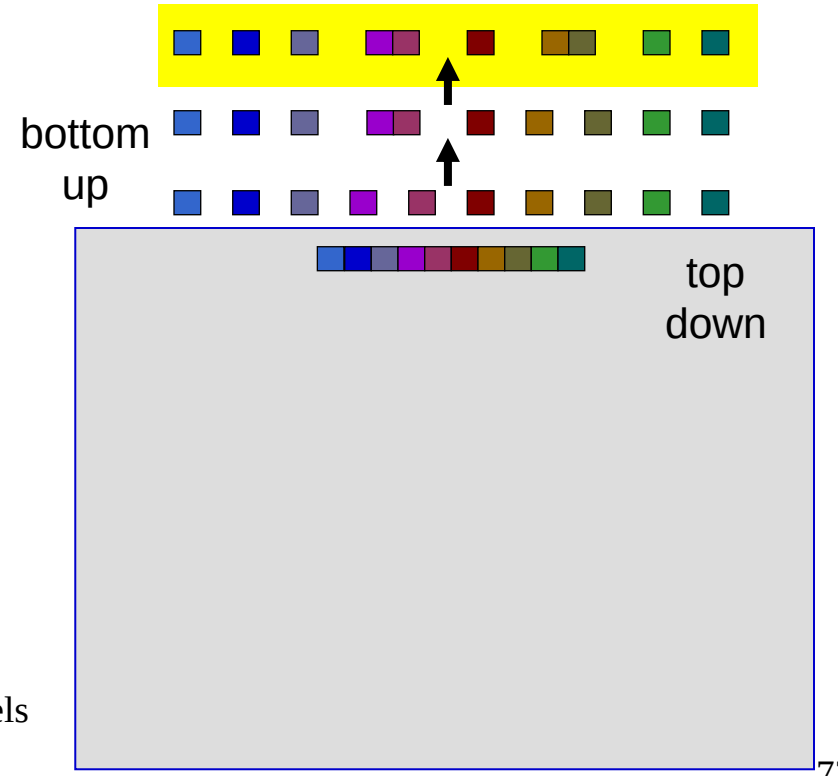
Hybrid Procedure - II

2. Cluster the two closest groups in bottom up layer



Hybrid Procedure - II

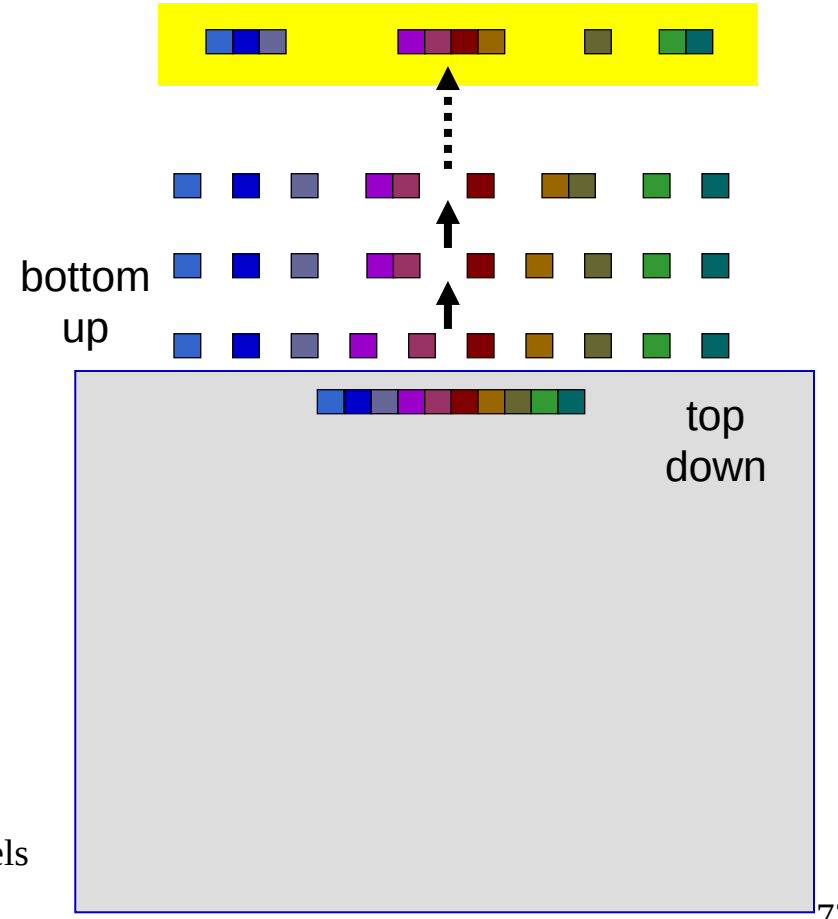
2. Cluster the two closest groups in bottom up layer



Hybrid Procedure - II

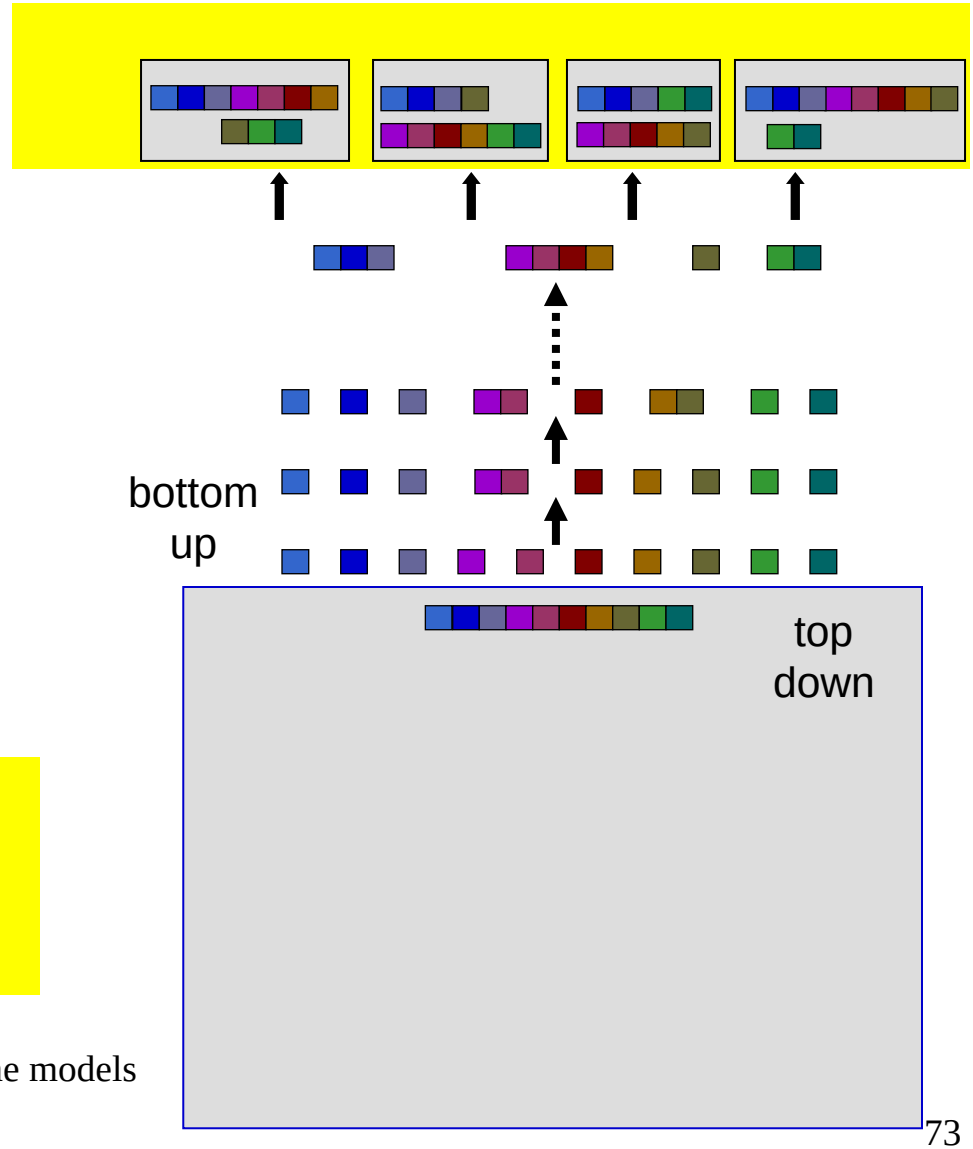
2. Cluster the two closest groups in bottom up layer

3. Repeat this process until only a small number *K* (*K* between 4 and 16) groups remain
   - This number is small enough that we can now exhaustively evaluate all partitions



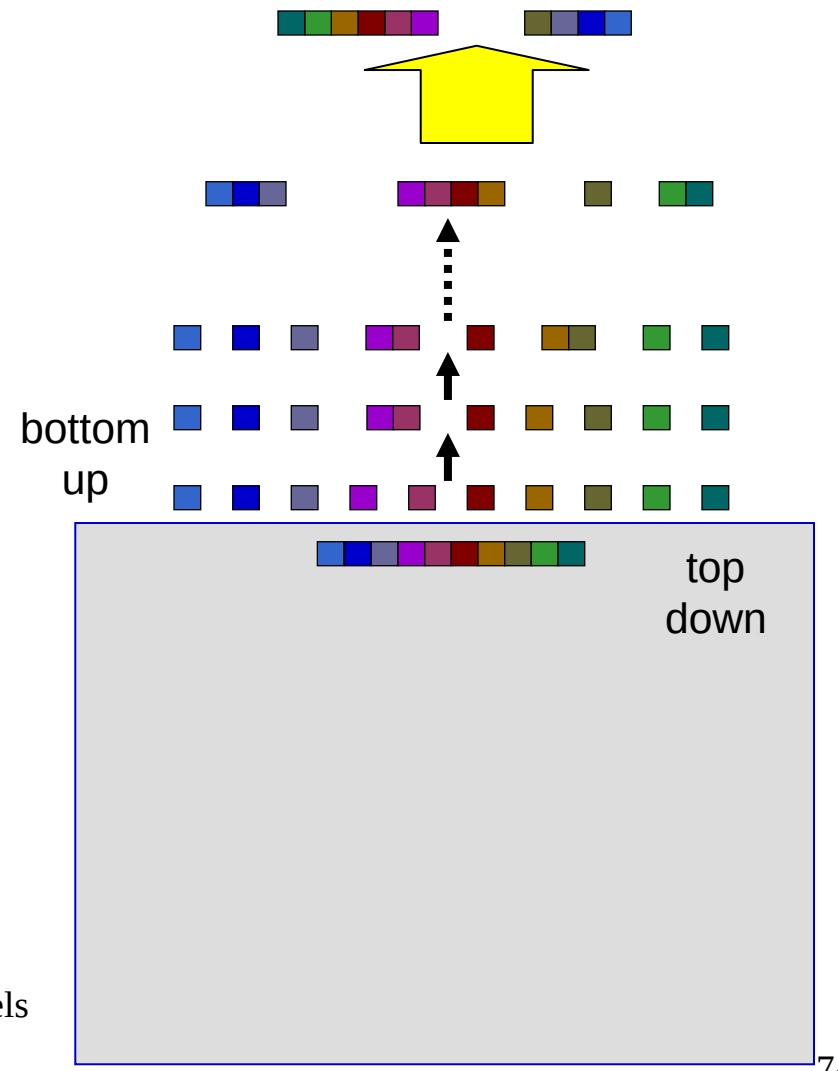
Hybrid Procedure - II

2. Cluster the two closest groups in bottom up layer
3. Repeat this process until only a small number K (K between 4 and 16) groups remain
   - This number is small enough that we can now exhaustively evaluate all partitions
4. Exhaustively evaluate all 2^{K-1} partitions of the remaining K groups



Hybrid Procedure -III

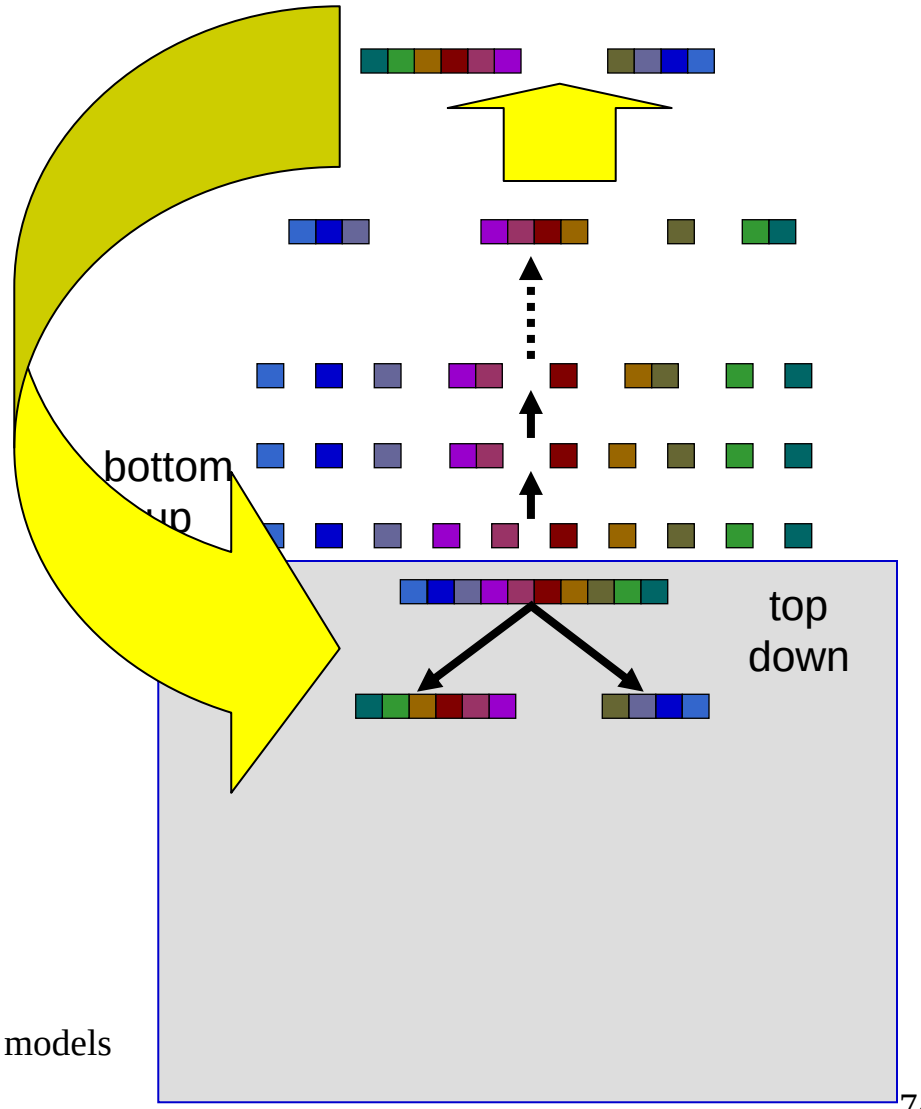
5. The best partition of the K bottom-up generated sets into two groups represents our guess for the best partition of the overall training set into two clusters



Hybrid Procedure -III

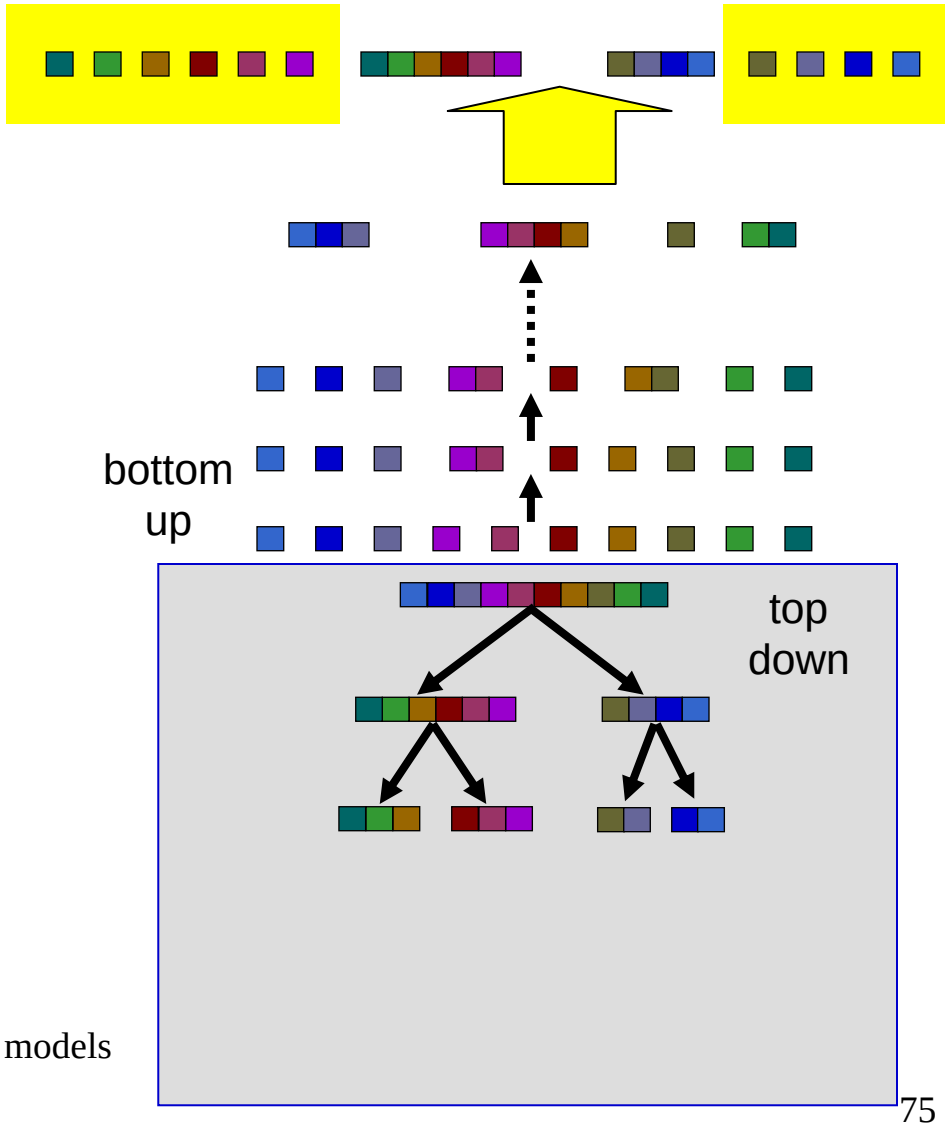
5. The best partition of the K bottom-up generated sets into two groups represents our guess for the best partition of the overall training set into two clusters

6. Set the two clusters as the second level of the top down tree



Hybrid Procedure - IV

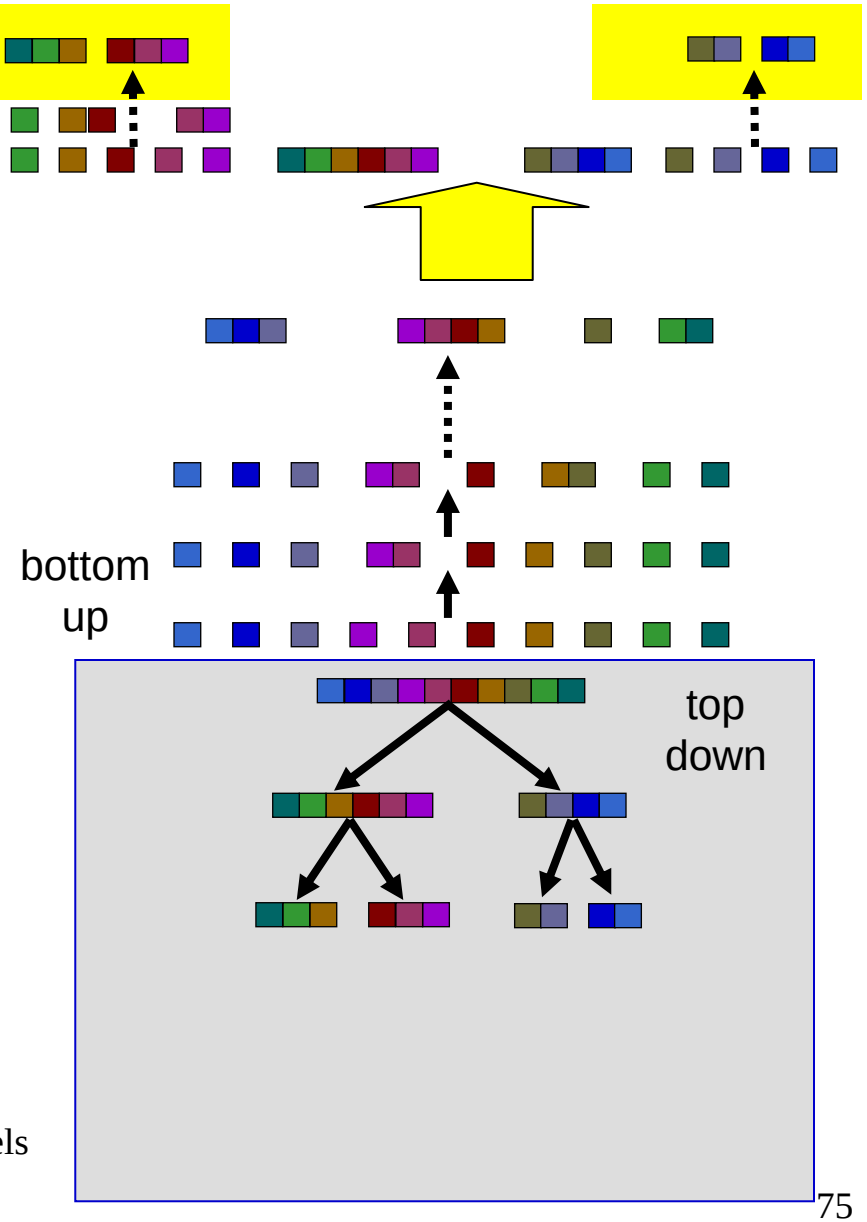
7. Each of the two clusters is in fact a group of states



Hybrid Procedure - IV

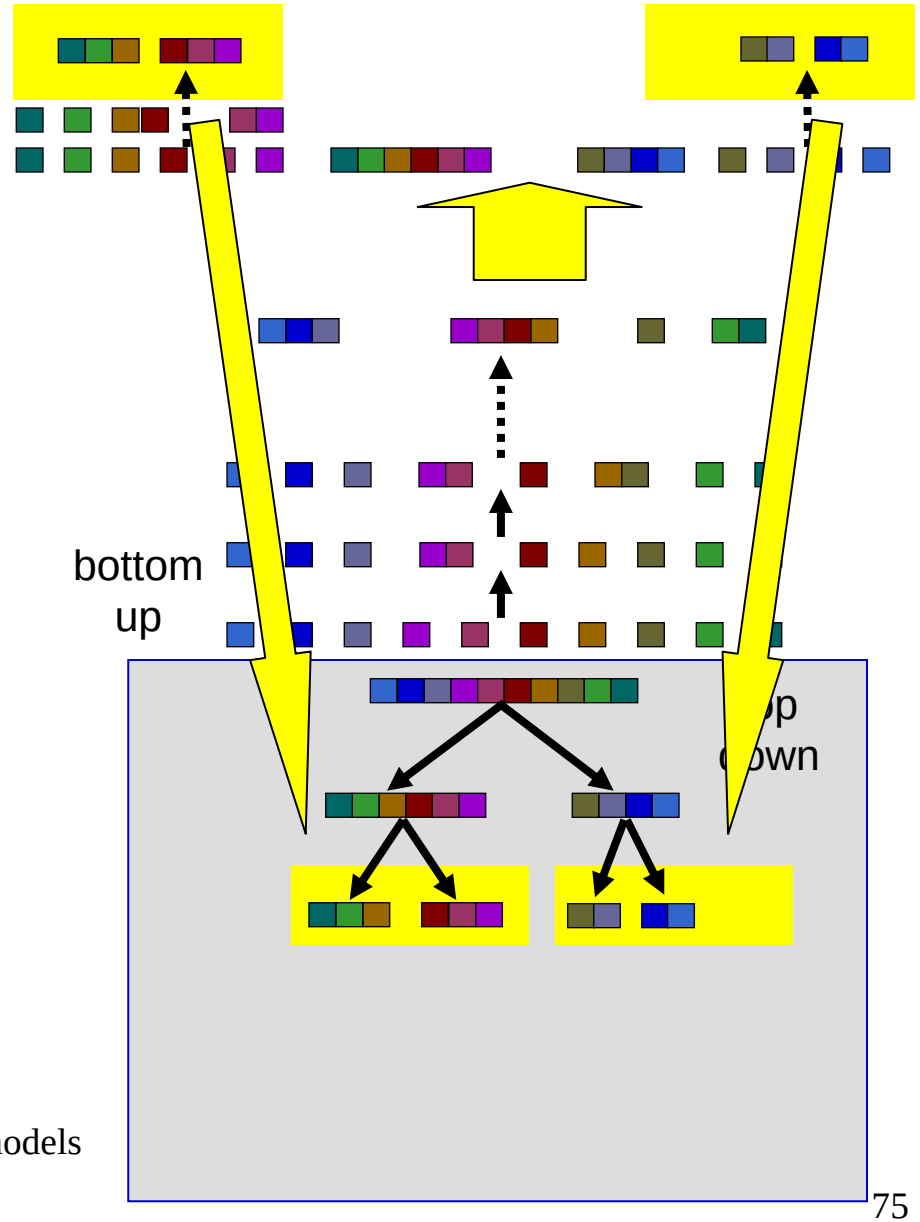
7. Each of the two clusters is in fact a group of states

8. They can each similarly be clustered bottom up until *K* groups remain, and then partitioned exhaustively to give two clusters



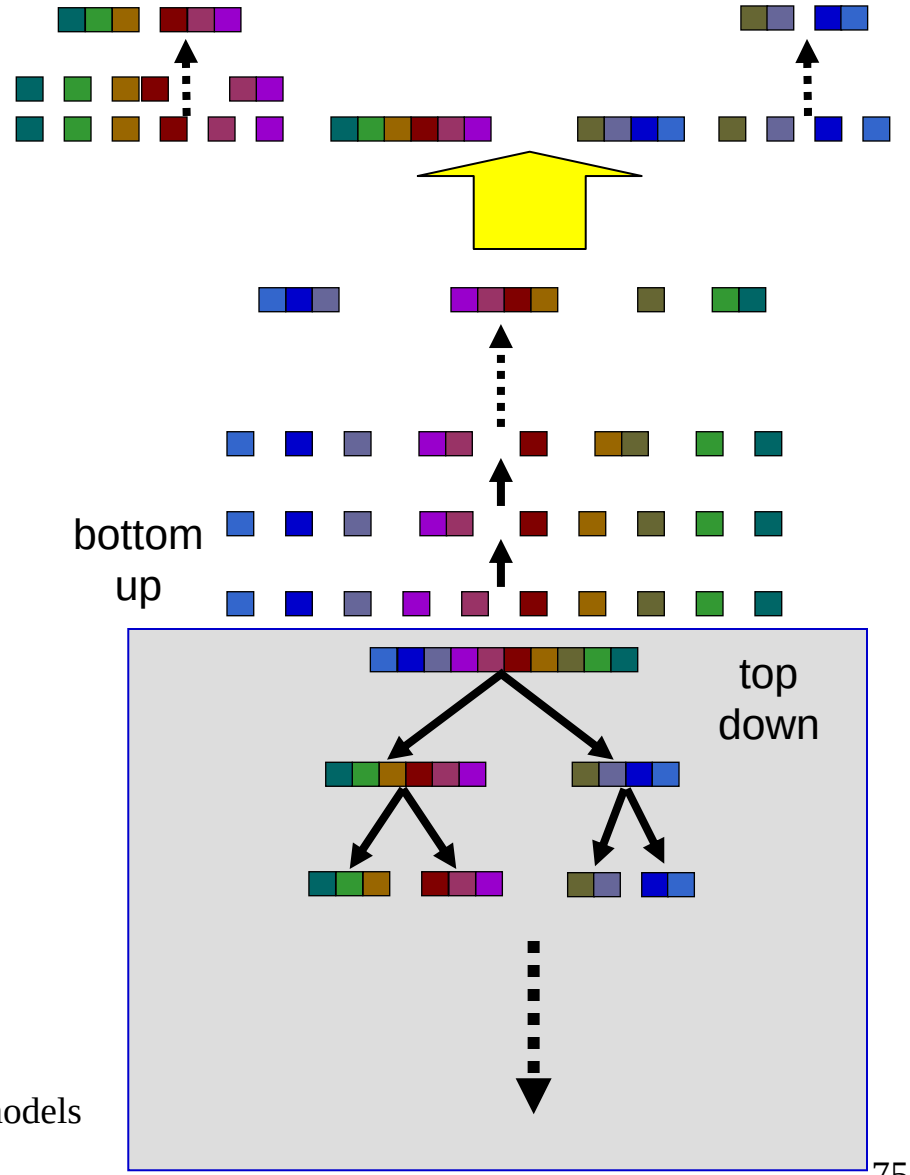
Hybrid Procedure - IV

7. Each of the two clusters is in fact a group of instances
8. They can each similarly be clustered bottom up until *K* groups remain, and then partitioned exhaustively to give two clusters
9. These new clusters would form the third level of the top-down tree

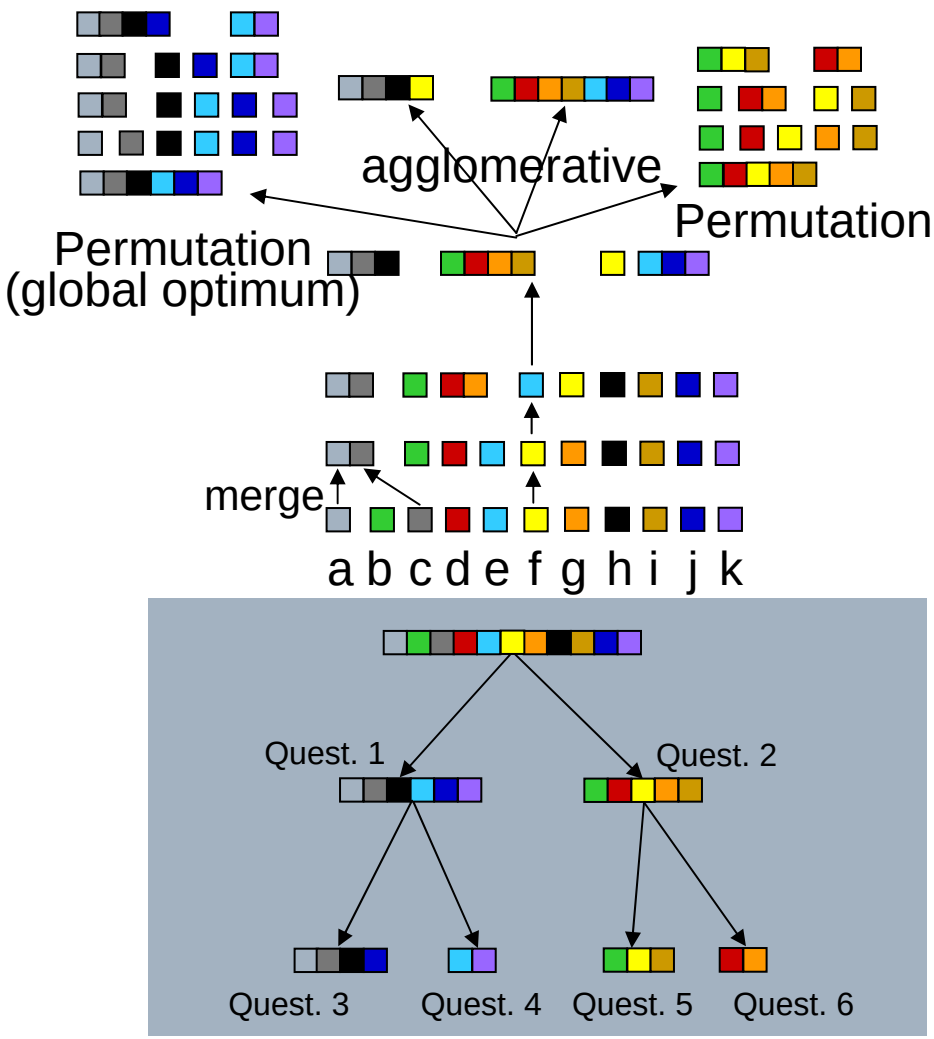


Hybrid Procedure - IV

7. Each of the two clusters is in fact a group of instances
8. They can each similarly be clustered bottom up until *K* groups remain, and then partitioned exhaustively to give two clusters
9. These new clusters would form the third level of the top-down tree
10. The process can be recursed to complete the top-down tree

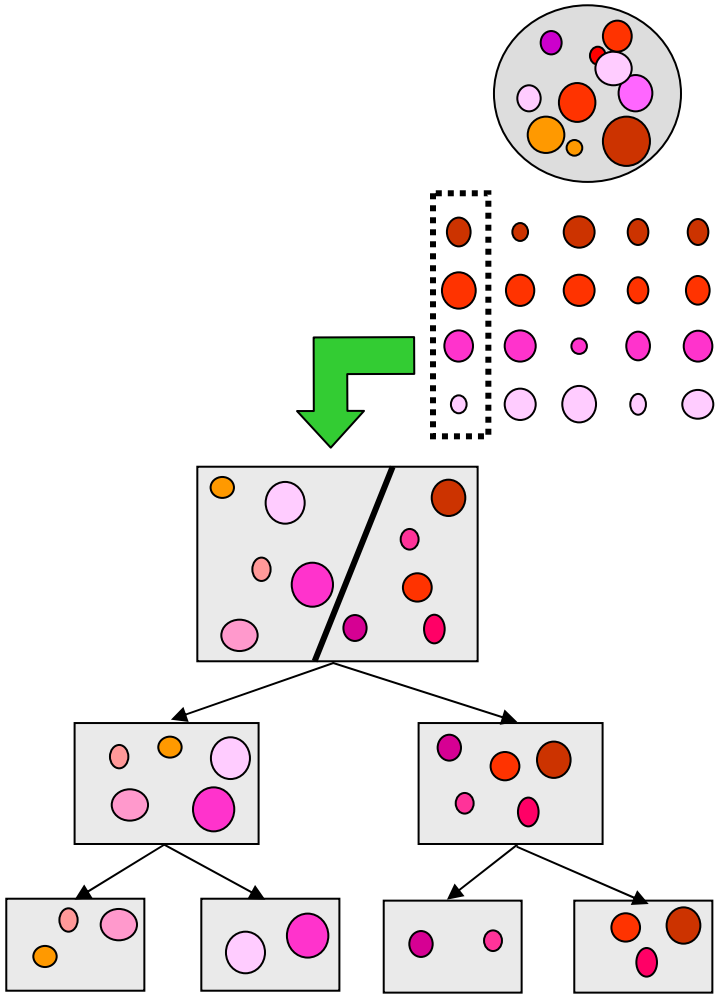


Algorithm for generating phone classes



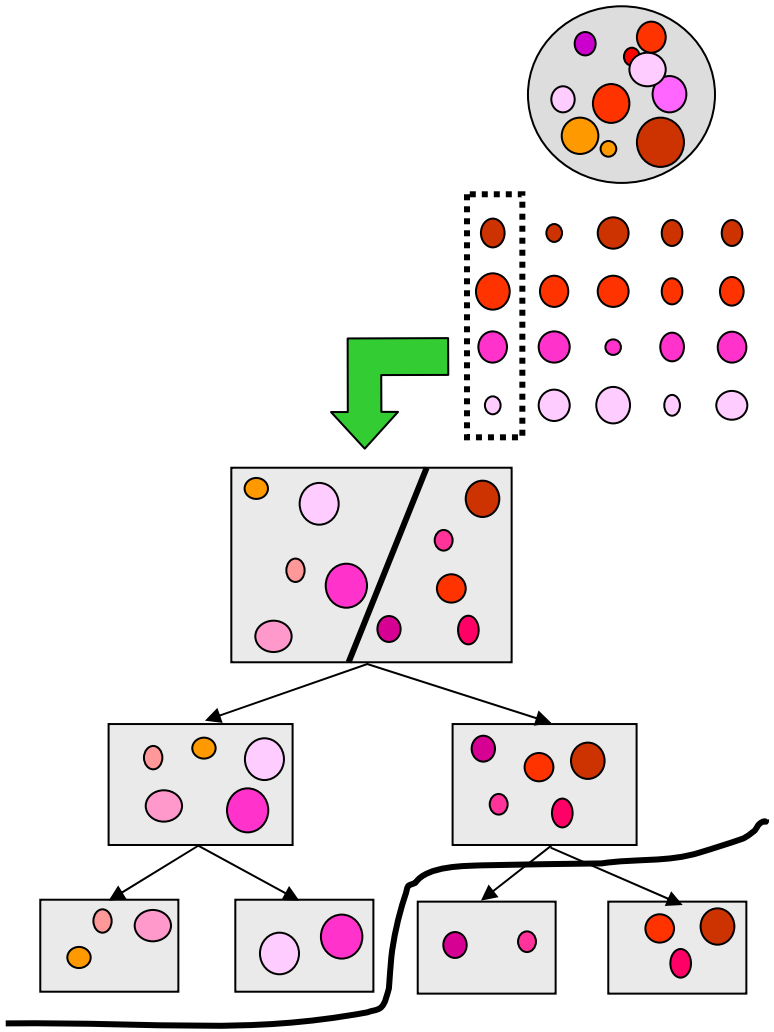
- The resulting groups in the top-down tree represent linguistic questions
- Why the procedure is optimal:
  - Top-down clustering is known to be optimal but expensive
  - Bottom up clustering is tractable, but suboptimal
  - This procedure strikes a balance

Grouping of context-dependent units for parameter estimation



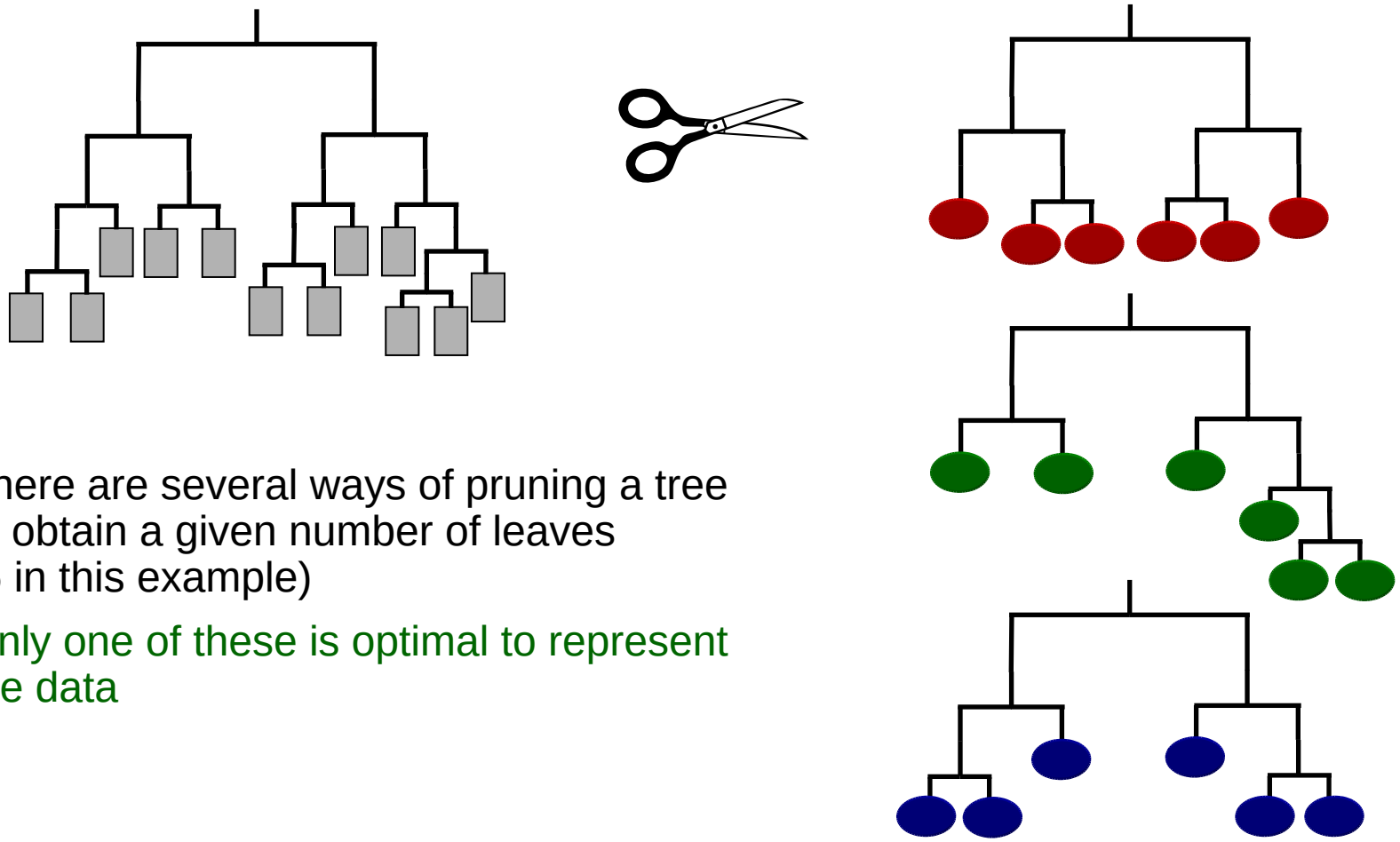
- When expert-given linguistic questions are not available, the procedure just described can be used to compute linguistic questions
- In this case the procedure for building the decision tree is *entirely* data-driven
  - No human input
  - Useful when experts are not available

Grouping of context-dependent units for parameter estimation



- The resultant *decision* tree will have many leaves
  - Eventually one leaf per triphone
- To obtain *groupings* we must prune the tree so that leaves represent *sets* of triphones
- Pruning leaves behind a shallower tree
- The degree of pruning determines how shallow the tree is
  - Leaves of shallower trees have triphones with more variations among theselves
    - Not good
  - However leaves of shallower trees will also have more data associated with them
    - Good

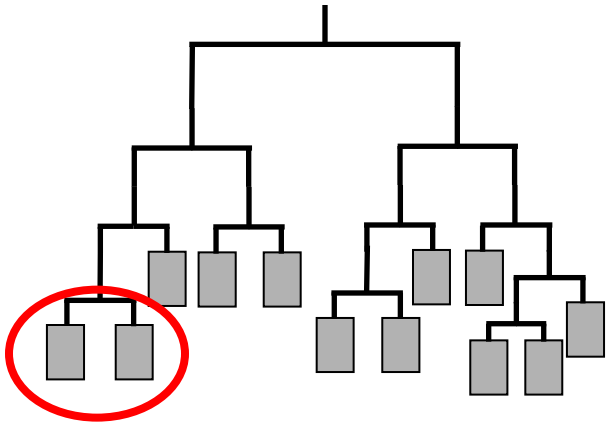
Pruning Decision Trees



There are several ways of pruning a tree to obtain a given number of leaves (6 in this example)

Only one of these is optimal to represent the data

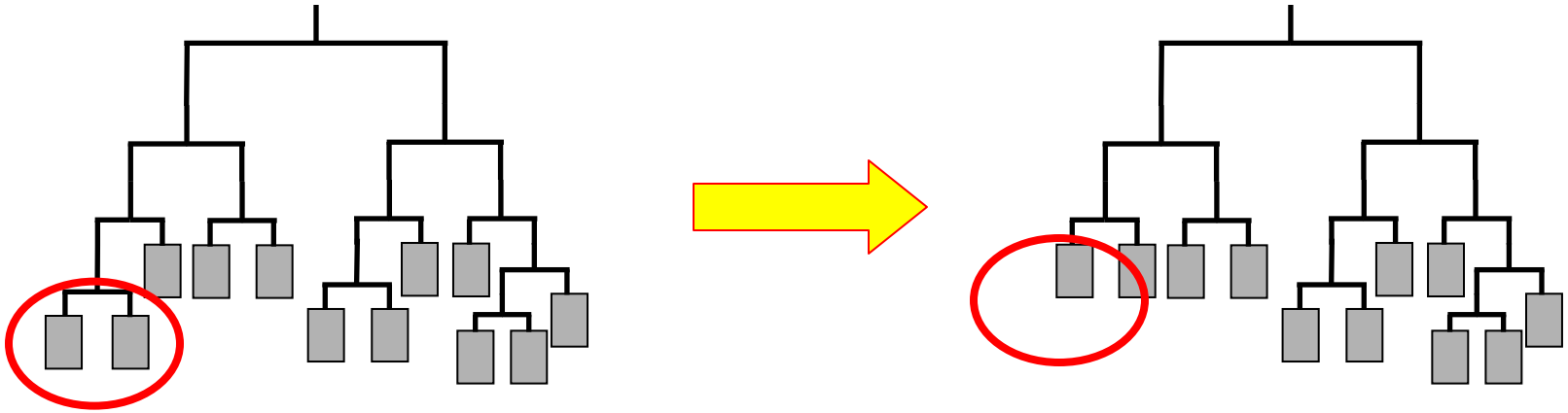
Pruning Decision Trees



Identify the pair of leaves that resulted in the smallest increase in likelihood with respect to the parent

$N \log\left(2\pi^d |C|\right) - 0.5 N_1 \log\left(2\pi^d |C_1|\right) - 0.5 N_2 \log\left(2\pi^d |C_2|\right)$ was lowest for this pair

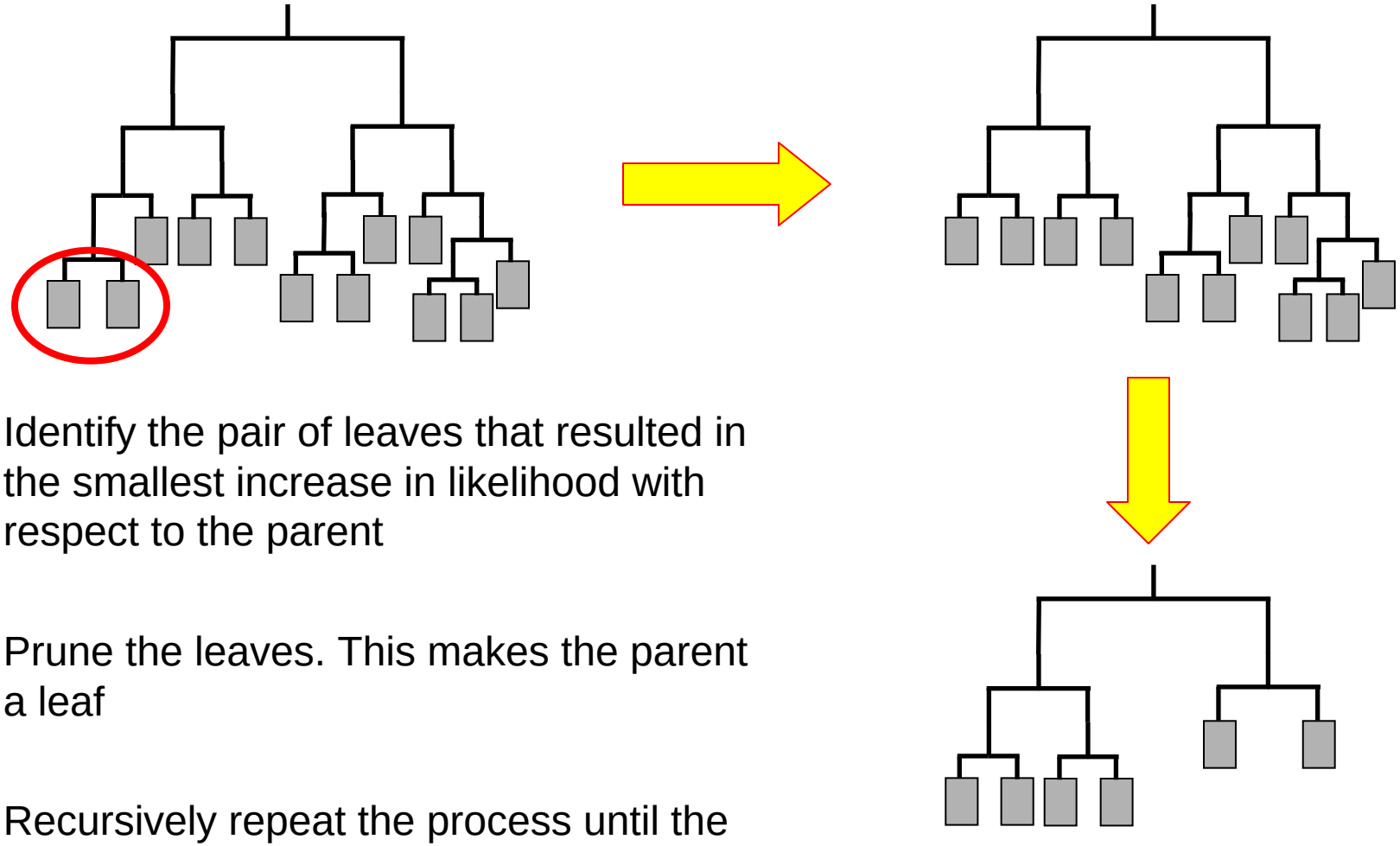
Pruning Decision Trees



Identify the pair of leaves that resulted in the smallest increase in likelihood with respect to the parent

Prune the leaves. This makes the parent a leaf

Pruning Decision Trees



Identify the pair of leaves that resulted in the smallest increase in likelihood with respect to the parent

Prune the leaves. This makes the parent a leaf

Recursively repeat the process until the desired number of leaves is obtained

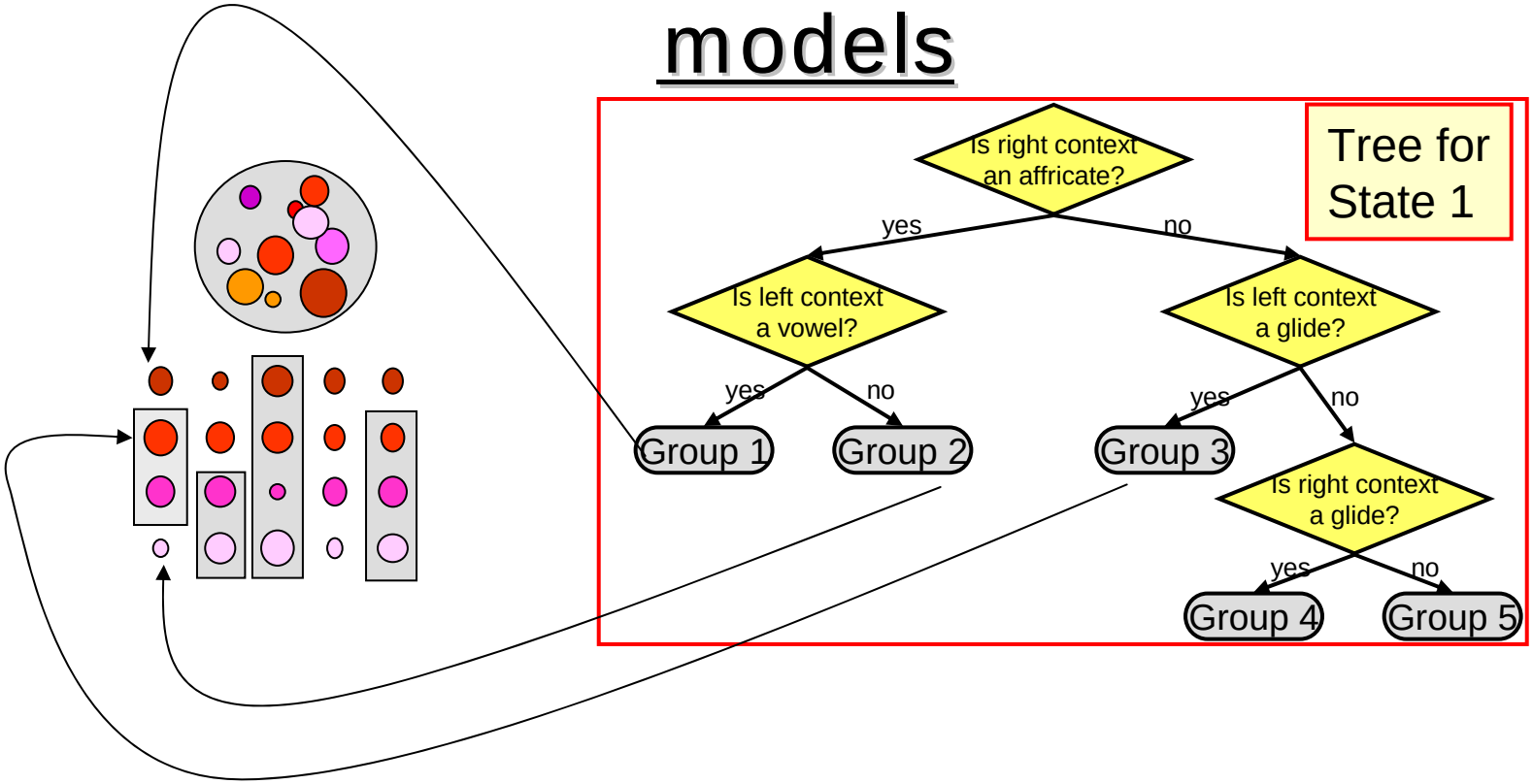
Training Procedure for Decision Trees: 1

- Train HMMs for all triphones in training data with no sharing of parameters
- This stage of training is the "context-dependent untied training"
- Use these "untied" HMM parameters to build decision trees
  - A separate decision tree is built for each state of each phoneme
  - The decision tree for any state of a phoneme describes the grouping of that state for the triphones of that phoneme
    - E.g. A decision tree for state 1 of AX represents the clustering of the 1st state of all triphones of AX

Training Procedure for Decision Trees: 2

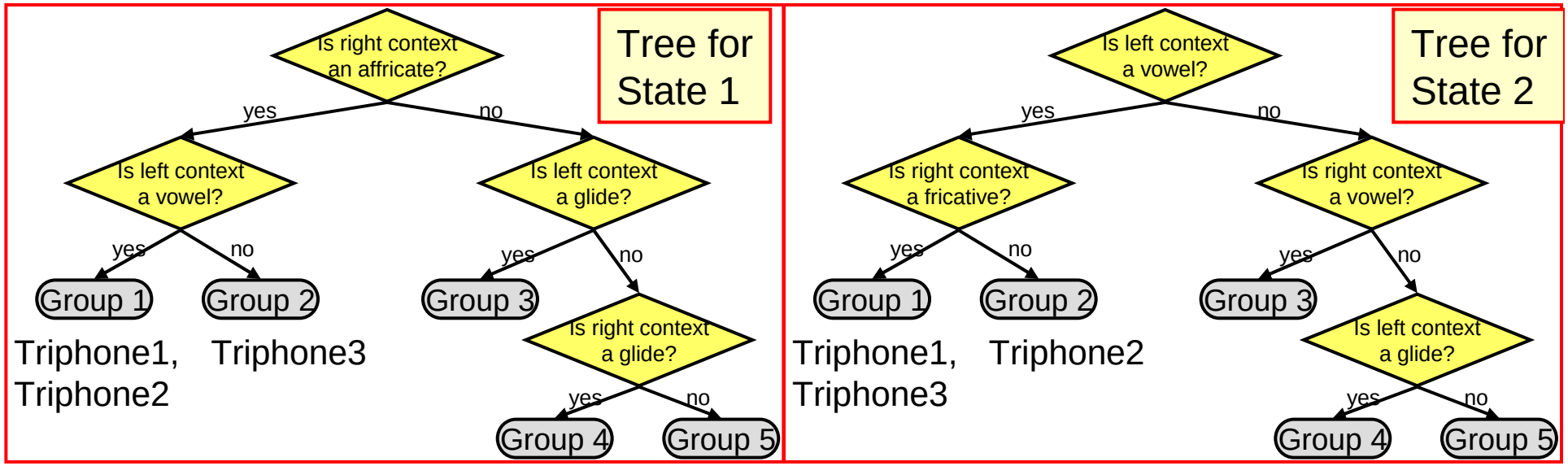
- The set of all decision trees for all states of all phonemes is then pruned
- When selecting a specific pair of leaves to prune, we choose the best pair to prune from *among all the decision trees*
  - The best pair will result in least decrease of consistency as specified by $N \log\left(2\pi^d |C|\right) - 0.5 N_1 \log\left(2\pi^d |C_1|\right) - 0.5 N_2 \log\left(2\pi^d |C_2|\right)$
- The final number of leaves required in the set of all decision trees must be specified
  - This is the number of tied-states the system will have
- Decision trees are pruned until this number of leaves is achieved
  - Always selecting the best pair of leaves to prune from among all trees

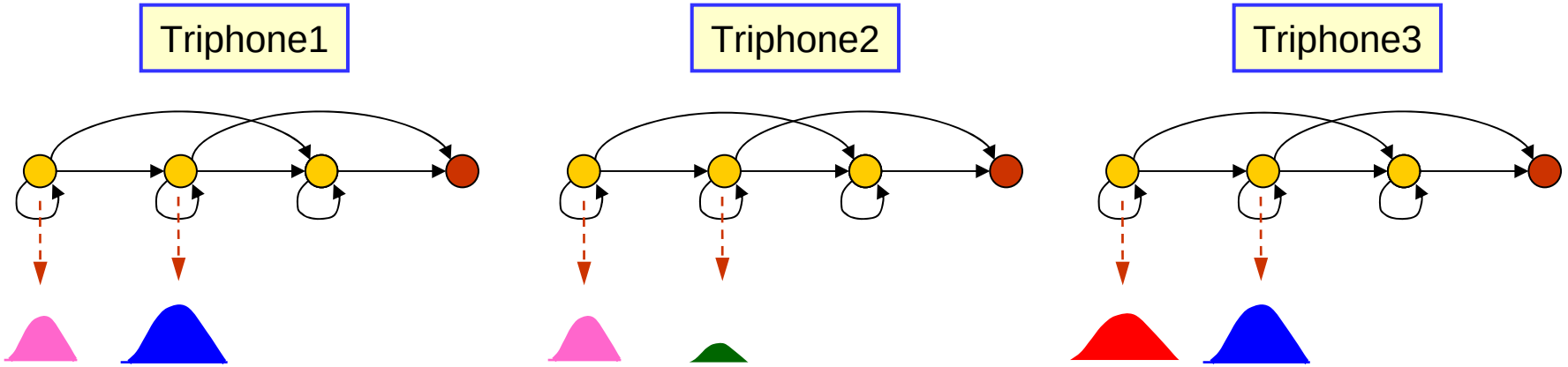
Context dependent (triphone) tied state models



- Pruned trees are used to determine how states must be grouped for tying
  - E.g. leaves 1 2 and 3 in the illustration determine the three groups for the first state of the triphones of this phoneme
- All triphones whose 1st state are grouped together *share the state output distribution* for their first states
- In general, the purpose of the decision trees is to determine how triphones share state output distributions

State-level tying





- To find the state output density of any state of the HMM for a triphone, pass it down the decision tree for that state

Components of Triphone HMMs

- The HMM for any triphone has two sets of components
  - The transition matrix
  - The set of state output distributions
    - For Bakis topology HMMs with a fixed start state, initial state probabilities are not a factor
- To build the HMM, both components must be determined
- The state tying procedure only specifies how state output distributions are obtained
- Transition matrices are also required
- Ineffective to train separate transition matrix for each triphone
  - Data insufficiency bites!
  - Transition matrices are also shared

Overall Process for Building a Triphone Model

- Transition Matrix:
  - All triphones of a given phoneme use the same transition matrix
    - This is the transition matrix of the context-independent phoneme itself
- State output densities: For each state
  - Use the triphone identity (including all features used to build a decision tree) to identify a leaf of the decision tree
  - Use a state output distribution associated with that leaf
- The same procedure is used to build HMMs for all triphones regardless of whether they are seen in training data or not
  - The procedure can be used to compose an HMM for *any* triphone

The outcome of parameter sharing: Tied-state HMMs

- The HMMs for various triphones will share parameters including transition matrices and state output densities
  - Typically all triphones of the same phoneme (e.g. all triphones of the kind AX(*,*)) will share the same transition matrix
  - State output densities will be shared according to state-dependent decision trees
- We will have HMMs for all triphones
  - Including ones that were not seen in training
  - There will be NO triphones for which a model cannot be constructed
    - Even if it was not seen in the training data
    - Every triphone has a base phoneme whose transition matrix is shared
    - Every triphone will arrive at some leaf of the decision trees, and share the state output densities associated with those leaves

Recap

- For each phoneme (AX, AH, AY, … IY,.., S, .. ZH)
- For each state (0,1,2..)
  - Gather all the data for that state for all triphones of that phoneme together
  - Build a decision tree
  - The distributions of that state for each of the triphones will be tied according to the composed decision tree
- Aggregate transition information for all triphones
  - Compute a common transition matrix for all triphones of the phoneme

- Assumption: All triphones of a phoneme have the same number of states and topology
- If the HMM for all triphones of a phoneme has K states, we have K decision trees for the phoneme
- For N phonemes, we will learn N*K decision trees

State tying information in the Sphinx

Entries from a *model definition* file

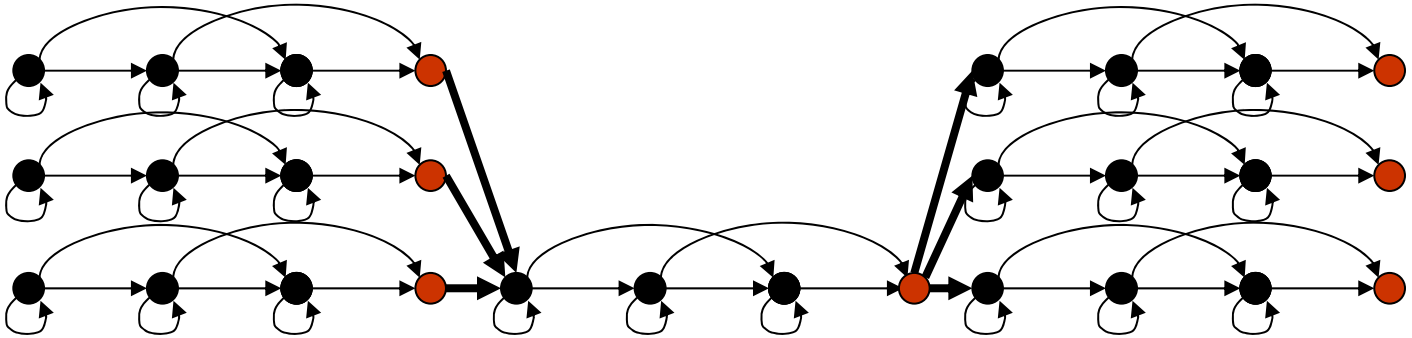
#base	left	right	position	attribute	HMM id	…HMM state id's ...			
AX	DX	R	i	n/a	1	1001	1221	1487	N
AX	DX	R	b	n/a	1	1001	1222	1487	N
AX	B	M	i	n/a	1	1001	1223	1493	N
AX	K	N	e	n/a	1	1002	1222	1493	N

- The rows represent triphones
- The HMMid represents the id of the shared transition matrix
  - All shown triphones share transition matrix no. 1
- The numbers to the right show ids of shared state densities
  - The first state of AX(DX,R,i), AX(DX,R,b) and AX(DX,B,i) share the same density
  - The second state of AX(D,R,b) and AX(K,N,e) share the same density
  - Etc.

Parameter Sharing helps HMM size Reduction

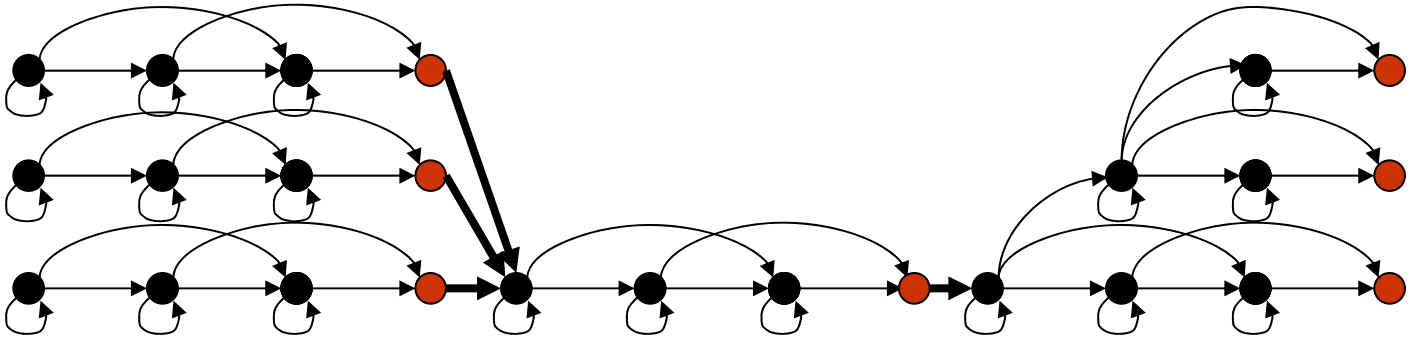
- In addition to reducing data-insufficiency and enabling composition of models for triphones that were not seen in training
- The total number of parameters in the system is greatly reduced
- The size of models and the computation required, both for training and recognition is reduced
- The size of HMMs can also be reduced by taking advantage of state tying

Parameter Sharing helps HMM Reduction



Composed HMM for ROCK

- Without state tying. By taking advantage of state tying, this could reduce to (only showing effect on models for “K”):



Building and Pruning Decision Trees with SphinxTrain

- A simple exercise:
  - Train “untied” triphone models using a small corpus
  - Build decision trees from the corpus
  - Prune it