

Statistical Learning and Inference Methods for Reasoning in Games

March 30, 2005

Brian Mihok
Draper Laboratory



Michael Terry
CSAIL, MIT



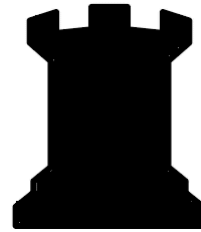
Outline

- Intro to Games
- Texas Hold 'em Demo
- Hidden Markov Models – Structure Learning
- Bayesian Nets – Interpolating Conditional Density Trees
- Project Preview – Level 1 Reasoning in Poker using HMMs and Bayesian Nets
- Summary

In this lecture, we present two key techniques for reasoning in games. To motivate our talk, we begin with a demonstration from the game of Texas Hold'Em. We use this demonstration to provoke some key questions in reasoning, and follow that up with lectures on two key techniques: Bayesian Nets and Hidden Markov Models. For these topics, we will begin with a review some of the fundamentals in each of these areas, then move into describing some more recent developments in research for the advanced portions of the talk. We then wrap up the information presented, and describe how we will use these methods in our final project.

Why Games?

- Economic Models
- Combat Scenario Models
- AI Benchmarks
- Fun
- \$\$\$



I'd like to point out that lessons learned from modeling and learning games and game theory have been extended to a number of domains:

For example, Adam Smith's classic, "Wealth of Nations", can be modeled as a zero sum game. Also, simulation of warfare typically involves adversary modeling. Finally, games have a set of clearly defined rules, providing good AI benchmarks with a good way of evaluating algorithms on specific problem domains.

These are a few of the many domains where principles of games and game theory can be applied.

Texas Hold ‘em Poker

- 2-10 players
- Goal: make best 5 card poker hand out of 7
- Betting proceeds after every round
- Round 1: Each player dealt 2 cards face down
- Round 2: Flop
 - 3 cards face up for all players to use
- Round 3 and 4: Turn, River
 - Turn and River adds a card each to community cards
- Showdown

To demonstrate the techniques we will cover in this lecture in action, we will now have a demonstration of how Texas Hold ‘em. We will do this by having four volunteers from the crowd come and play a hand. Before we ask for volunteers, I’ll give you a basic overview of the game. Texas Hold ‘em is typically played with 2-10 players all sitting around a common table. The goal of the game is to make the best poker hand, consisting of 5 cards, from a total of 7 cards. The game starts by each player receiving two cards face down. These cards are only for the individual player to see. Then, each player bets upon the strength of these cards. Next, the flop comes. In the flop, 3 cards are dealt face up in the middle of the table. These are the start of 5 community cards that all the players can use. Based upon the strength of these cards and the two face down cards, the players bet again. This round of betting is followed by the “Turn,” which deals another card face up in the middle of the table. This is again followed by a round of betting. The final card is dealt face up in what is called the “River.” This is followed by a final round of betting. After the betting concludes, each player shows their hand in “the showdown.” The winner is determined by who has the best hand using their own two individual cards and the 5 community cards.

Now that you know how the game is played, can we get 4 volunteers to play a round of Hold ‘em?

(Demo a round of Hold ‘em. Talk through the game at each and every point to minimize pressure and stress. Try to make new players comfortable.)

Keys to Hold'em

- Hand Strength
- Hand Potential
- Odds
- Bluffing
- Psychology
- **Multi-Level Reasoning**

As we saw in the demo, there are a number of factors that go into deciding one's actions in Hold'em. At the most basic level, we are reasoning about our own hand and its potential. As we start to reason beyond these concepts, we realize that the odds, bluffing, and psychology are a few of the many factors that determine how us and our opponents act. I would like to break Hold'em strategy down into what I call a multi-level reasoning problem:

Level 0: Reasoning about your hand

Level 1: Reasoning about your opponents hand

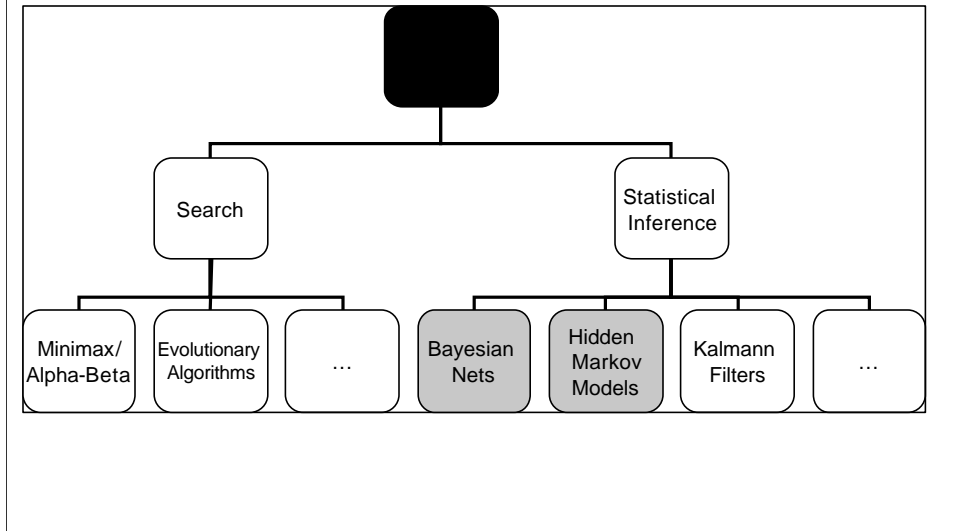
Level 2: Reasoning about what your opponent thinks about his and your hand

Level 3: What to do about what your opponent thinks

...

Professional players are often thinking on level 4 or 5, which would be incredibly difficult for a machine to accomplish without the lower level reasoning.

Reasoning Techniques for Games



In this lecture we will focus on two techniques for reasoning about games to address the Level 1 reasoning problems just introduced. These techniques fall under the category of statistical inference. We chose these two techniques because they are very powerful for reasoning about situations with hidden information.

In general, these techniques use statistics in the form of large mined datasets to learn the structures and parameters for graphical methods. We describe the representation for these methods, along with a number of recent advances in research used to more efficiently learn this representation.

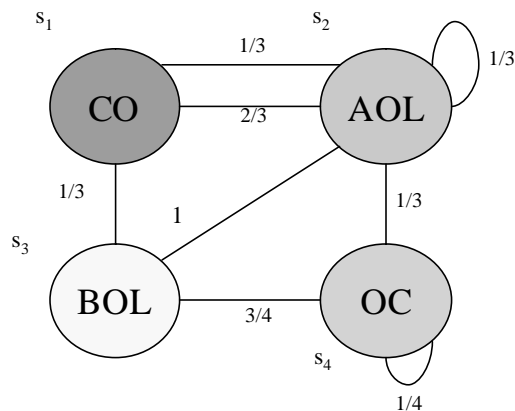
Outline for HMM Section

- Background
 - HMM intro and formalization
 - HMM example related to Hold'em
 - Problems to solve and general solutions
- Info extraction using HMM structures learned by stochastic optimization
- Summary

Here is an outline of the HMM section of the lecture. First, I will discuss some background about HMMs. I will start off with a graphical explanation of HMMs and then proceed into a more formal definition. I will follow up this formalization with an example of how a Hold 'em hand can be represented with HMMs. Specifically, this will be used to show how the model changes as more information becomes available and the essence of the problem related to inferring your opponent's hole cards given only his/her observations. Then, this example will be generalized to problems related to all HMMs and a mention of how to solve these problems will be mentioned. Then, I will go into the advanced portion of the lecture. This section deals with how a HMM structure can be learned through stochastic processes to best extract information from human readable text. This is important when trying to learn information needed to model opponents from previous hand-histories. I will give a more detailed outline when I get to that section of the lecture when I get there.

HMM Graphical Representation

- Finite num states, N
- a – Transition matrix
 - $N \times N$
 - $[0? 1]$
- b – Observation matrix
 - $N \times M$
 - $[0? 1]$



(Need to go through this slide quickly as there is a lot of material in the lecture. You need to gauge the audience's prior knowledge on this topic. If they already know this, breeze through it to save time. If, however, they don't, then the time here is well spent because if they don't get this, everything else will be completely over their heads.)

Basic Properties of Markov Systems:

- Finite number of unique states (s_1, s_2, \dots, s_N)
- System in only 1 state at a time
- Transition from state i to state j :
 - Given through a defined probability
 - Occurs at discrete time steps
 - Independent of previous history
(depends only on current state)

When Markov Systems are represented as a directed graph

- States are represented by nodes
- State transitions are represented by edges
- Edge labels are $P(s_{t+1} = s_j | s_t = s_i)$
- Notice that the transition probabilities sum to 1

HMM Formalization

$$? = \langle N, M, \{p_i\}, \{a_{ij}\}, \{b_i(j)\} \rangle$$

- States are labeled $S_1 S_2 \dots S_N$
- N – number of states
- M – number of observations
- p_i – $P(q_0 = S_i)$
- a_{ij} – transition matrix
- $b_i(j)$ – observation probability matrix

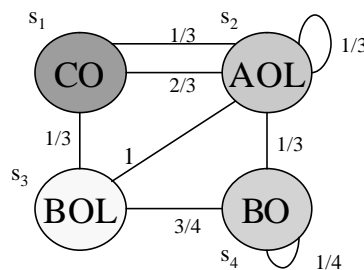
With that graphical example in mind, let's now turn to a formalization of HMMs:

• 5-tuple usually denoted by $?$. Variables include:

- N , which is the number of states
- M , number of observations (Helps to flip to the previous slide and point out that the unique observations in the graphical example are A, B, C, L, and O, making $M = 5$, with $N = 4$)
- p_i , which is the probability of any given state being the initial state. This is a very important point for Hold 'em. When you are trying to infer your opponent's cards, you are essentially interested in knowing their initial state, which is their hole cards.
- a , which is the transition matrix and
- b , which is the observation probability matrix

The two matrices are shown on the next slide to solidify their meaning.

Demo of Matrices



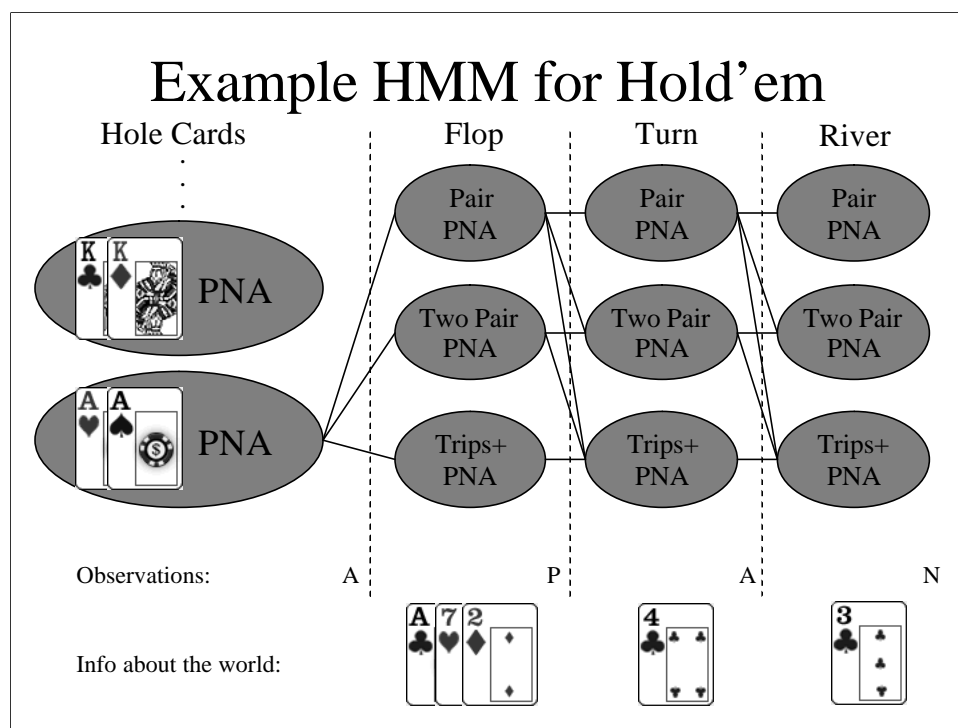
$$\begin{array}{cccc}
 a_{11} = 0 & a_{12} = 2/3 & a_{13} = 1/3 & a_{14} = 0 \\
 a_{21} = 1/3 & a_{22} = 1/3 & a_{23} = 0 & a_{24} = 1/3 \\
 a_{31} = 0 & a_{32} = 1 & a_{33} = 0 & a_{34} = 0 \\
 a_{41} = 0 & a_{42} = 0 & a_{43} = 3/4 & a_{44} = 1/4
 \end{array}$$

} The state transition probabilities
 $P(q_{t+1} = S_j | q_t = S_i) = a_{ij}$

The observation probabilities, $P(O_t = k | q_t = S_i) = b_i(k)$

$$\begin{array}{ccccc}
 b_1(A) = 0 & b_1(B) = 0 & b_1(C) = 5/6 & b_1(L) = 0 & b_1(O) = 1/6 \\
 b_2(A) = 3/4 & b_2(B) = 0 & b_2(C) = 0 & b_2(L) = 1/12 & b_2(O) = 1/6 \\
 b_3(A) = 0 & b_3(B) = 1/3 & b_3(C) = 0 & b_3(L) = 1/3 & b_3(O) = 1/3 \\
 b_4(A) = 0 & b_4(B) = 0 & b_4(C) = 1/2 & b_4(L) = 0 & b_4(O) = 1/2
 \end{array}$$

(Talk through the population of the matrices for one particular node in each matrix. For example, go through row one in a and row 2 in b. Note that the assignment of the probabilities in b was completely arbitrary. In addition make these points)



(This is probably one of the “cooler” slides in the presentation. When done probably, it will definitely catch people’s attention. However, it is very easy to spend a great deal of time on this slide. If the demo took longer than you thought it would, if you started late, etc, give a compressed version of this slide, especially if you analyzed the play of the participants in the demo).

(Before first click)

Now, we are going to transition into how we can represent a hand of Texas Hold ‘em using HMMs and give an explicit example of the type of problem that we’re attempting to solve. As we’ve seen, Hold ‘em starts off with each player being dealt two cards, face down.

(Click – Cards dealt)

In this particular hand, we were dealt two kings, known in poker terminology as pocket kings. This is a great start! As mentioned during the formalization, the hole cards are used to define the initial state of the HMM.

(Click – States surrounding cards appear, along with “Hole Cards” text and the three dots)

As you can see, come bucketing can be done to limit the search space of the problem. The state can be defined in terms of important poker hands instead of each particular combination of cards. For example, we don’t care if we have the K of clubs and K of diamonds or the K of hearts and the K of spades, all we care about is that we have pocket kings. Similarly, the observations can be grouped into three major categories: passive, normal, and aggressive.

What's Hidden??

Nothing if these are your cards, but could be:

- Path of states
- Info about underlying system
 - Probability of transition between states
 - Probability of an output for a given state
 - Available outputs for a state
 - Actual state representation itself

The example just presented gives an idea of what information is hidden in the case of Hold 'em. The state transition probabilities are known ahead of time and the known information produced from the games include the observations, the community cards, and your two hole cards. However, you don't know your opponent's initial state, nor do you know the path which the opponent's hand took during the game. We can no generalize this to all HMM models. Poker represent a class of problems where you know information about the underlying system, but you don't know about the specific path traveled. However, another class of problems appear when you are missing information about the underlying system. For instance, could now what the all the possible states are, but not know the probability of transitioning between them. Or perhaps you don't know what the probability of any given output is for a particular state. This is critical in Hold 'em. In the previous example, the player used the advanced technique of slow-playing the hand. However, maybe a more novice player would have just bet aggressively. One of the keys to figuring out what your opponent has based upon their observations is to know how they bet given a particular hand. Said differently, one of the keys is to figure out the probability of some output for a given state. Another piece of information that could be missing about a particular state is the outputs that are even available in a state. This isn't a problem for Hold 'em, but thinking back to our string output example, this would be the case if we didn't know that state 1 had a possible output of 'C' or 'O.' Finally, you could be missing the state representation entirely. You could literally know nothing about which states are present, the probability of transitioning between the various states, or the outputs that are available in any particular state. This is the type of problem we will be addressing in the advanced portion of the lecture. We'll talk extensively about this in a couple slides, but the essence is that we are trying to generate the underlying system that best allows us to extract information from human readable text.

3 Big Questions

- Know underlying system, want to know prob of a particular output sequence
 - Forward algorithm
- Know underlying system, observations, want to know most probable path (MPP) that lead to observations
 - Viterbi algorithm
- Know observations, want to determine underlying system
 - Baum-Welch algorithm

These types of problems can then be lumped into three basic categories. In the first, we know the underlying system, but we want to know the probability of a particular output sequence. This basic type of problem can be solved with the forward algorithm. The next major category occurs when we know the underlying system and are given a set of observation. With this information, we want to know what the most probable path of states was that generated the observation sequence. This class of problems is solved using the Viterbi algorithm. Finally, as previously mentioned, we could have only the observation sequence and from that, we want to determine the underlying system. In this case, once the structure is learned the Baum-Welch algorithm can be used to assign the probabilities of the state transitions and the output. We will see this come up again when we discuss the advanced technique. The Baum-Welch algorithm and the Viterbi algorithm were discussed in the prerequisite course 16.410. As a result, these algorithms will not be discussed in this lecture. However, if you would like to know more about them, Dr. Williams has slides on them to which you can refer.

Paper Specifications

- Information Extraction with HMM Structures Learned by Stochastic Optimization by Dayne Freitag and Andrew McCallum (2000)

It's now time to turn to the advanced portion of the lecture. The material for this section of the lecture is taken from the paper cited here.

Outline for Advanced HMM

- Problem statement and background info
- Traditional uses for HMMs in IE
- Extension of HMM to include target, surrounding context
- Show how structure is learned, evaluated
- Experiments and results

Here's the outline of the advanced section of the talk that I promised you earlier.

First, we'll start off by defining what the problem is and go over the background information required to understand and motivate it. We will see that we want to use HMMs to perform information extraction from human readable text. Specifically, we want to be able to learn the best structure of an HMM to use in order to accomplish this task.

In order to motivate this, after the background, we will take a look at how HMMs are currently used in IE problems.

Problem Statement

- Learn structure and parameters of HMM to best extract info from human-readable text
- Extract Text from Poker Hand Histories in Database:

*** Dealing Flop ** [4s, Jc, 8h]
player111 checks.
holdem23 checks.
** Dealing Turn ** [Qc]
player111 checks.
holdem23 bets [\$6].*

(Read bullet)

Bullet implies two parts: 1) Learning an HMM structure and parameters and 2) extract info from human-readable text

Learning HMM structure is the crux of the algorithm that is presented in the paper

Learning the parameters is accomplished by Baum-Welch.

Extract info from human-readable text:

Example:

Given an announcement of a seminar on a college campus, extract out the person giving the lecture and the location at which it is held.

In this case, you have a human-readable announcement, perhaps in email form, but you want to be able to automate the process of extracting the relevant information from the announcement. To better define this task, let's first define some background definitions related to the task and also establish the underlying assumption made by the authors.

Tying this back into poker, this example demonstrates how we could generate the Hidden Markov Model for extracting relevant information from a database of Poker Hand Histories for our final project.

Background Definitions & Assumption

- Information Extraction
- Sparse Extraction
- Fields
- Template Assumption

Definitions

- Information Extraction – The process of filling fields in databases by automatically extracting fragments of human-readable text.
- Sparse Extraction – The object is to extract relevant phrases from documents containing much irrelevant text
- Fields – Relevant fragments in the text

Underlying assumption

- Template Assumption – Assume that for every document in a corpus, there is a corresponding relational record (template), each slot of which is either empty or is filled with a fragment of text from the document. This means that you are not just given an email and told to extract the important text. Instead, you are given an email, told that it is a lecture announcement, and that you want to extract the name of the speaker and the location of the seminar.

HMMs in Information Extraction

- Determine most probable sequence of states to have generated whole document
- Extract symbols associated with designated “target” states
- Viterbi algorithm

Now that we have a sense of the problem, let’s see how HMMs are traditionally used in IE.

Two main tasks for HMM in IE

- Determine MPP
- Extract symbols from target states

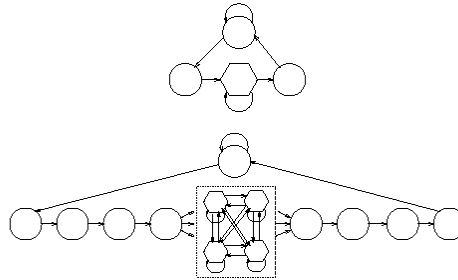
Can be done in a computationally efficient manner using Viterbi algorithm mentioned in background section

Usually don’t have anything about learning the structure. Usually the structure is already defined, typically done through expert analysis.

Once we have the HMM, we can use Viterbi to get the MPP. Thus, this then begs the question, how do we get the HMM? As we’ve said before, the task of determining the HMM can be divided into two parts, learning the structure and training the parameters.

Towards the goal of learning the structure, the existence of “target” states is new. Before, all the states were of equal importance and history independent. Now, however, there are only certain states that produce output of interest to us and these states are fit into the context of the surrounding document instead of being considered in isolation. Let’s take a look at an example (Next slide)

Example HMM Structure



- Learner models both the field and its context on either side

The context is established through the simultaneous modeling of target states with the output directly neighboring these fields.

In the graph, the target states are distinguished between non-target states through the use of different shapes.

- Targets are shown as hexagons
- Non-targets are shown as circles

Non-target states can be broken down further into prefix, suffix, and background

Terminology and definitions:

- Target – States required to model the content of target phrases (analogous to fields in a database)
- Prefix – A set of one or more states connected together as a string. A prefix state transitions only to the next state in the string or, if it is the last state in the string, to one or more of the target states. Models are designed such that if a state sequence passes through a target state, it must first pass through a prefix.
- Suffix – Similar to prefix. Any state sequence must pass through a suffix upon leaving the set of target states.
- Background – States that model any text not modeled by other kinds of states. A background state has only transitions to itself and to the beginning of all

Model Properties

- Each HMM extracts just one field
- Does not require preprocessing
- Contain two kinds of states:
Target and Non-Target
 - Targets used to extract tokens of interest
- Not fully connected

The generic model type shown in the previous slides have the following properties:

- Each HMM extracts only one type of field, such as “seminar speaker”. When multiple fields are to be extracted from the same document, such as “seminar speaker” and “seminar location,” a separate HMM is constructed for each field.
- The HMM models the entire document. As a result, no pre-processing is required to segment the document into sentences or other pieces. The entire enter text of each training document is used to train transition and emission probabilities.
- (No additional talking points needed for third bullet)
- Restricted transition structures captures context that helps improve extraction accuracy. Forces the structure to go from background, to prefix, to target, to suffix, and back to background.

Additional Variables in HMM

- $L(s)$
 - Binary value indicating whether s is a target
- $\{l_1, l_2, \dots, l_m\}$
 - Binary value indication whether observation is a target
 - Used to train model

These models follow the same basic form of an HMM.

- Can only be in one state at a time
- There is an observation associated with an given state
- Transitions defined by probability
- Finite number of states
- Transition between states at discrete intervals

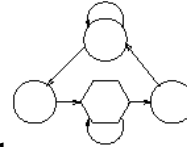
However, additional information now is needed to successfully encode the existence of target states

- First additional variable is $L(s)$: Binary value for each state simply telling you whether or not that state is a target
- Analogously, $\{l_1, l_2, \dots, l_m\}$ are binary values saying whether an observation is a target or not

This additional information is used to train the model, which is discussed a little later.

How is Structure Learned?

- Start w/ simple model
- Hill-climbing in space of possible structure
- Apply each of a set of operators
- Best resulting model in each iteration is saved
- Highest scorers tested against each other and final model is chosen



This is the essence of the algorithm presented in the paper.

It starts with the simple model that has just a single background, prefix, target, and suffix state.

From here, it performs a hill climbing method to search in the space of possible structures.

At each iteration of the hill climbing procedure, a set of operators is performed on the model to alter its current structure.

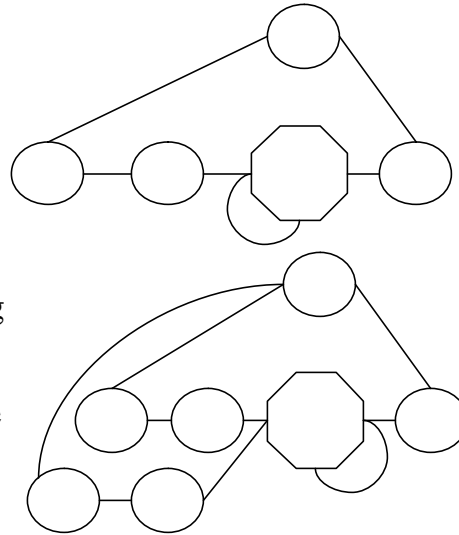
All the resulting structures in a giving iteration are scored and the best performer is picked.

Finally, all the high scorers are compared and the final model is chosen as the highest rated model

A logical question given this high-level presentation of the algorithm is then what are the operators that are performed on the structure (next slide).

Operators

- Lengthen a prefix
- Split a prefix
- Lengthen a suffix
- Split a suffix
- Lengthen a target string
- Split a target string
- Add a background state



Mentioned in the previous slide that you have a set of operators to perform on the model at each iteration of the hill-climbing method. This slide gives an exhaustive list of the operators.

Operators:

- Lengthen a prefix: A single state is added to the end of a prefix. The penultimate state now transitions only to the new state. The new state transitions to any target states to which the penultimate state previously transitioned.
- Split a prefix: A duplicate is made of some prefix. Transitions are duplicated so that the first and last states of the new prefix have the same connectivity to the rest of the network as the old prefix.
- Lengthen a suffix: Identical to the prefix-lengthening operation
- Split a suffix: Identical to the prefix-splitting operation
- Lengthen a target string: Similar to the prefix-lengthening operation, except that all target states, in contrast with prefix and suffix states, have self-transitions.
- Split a target string: Identical to prefix-splitting
- Add a background state: Add a new background state to the model with the same connectivity to non-background states as all other background states.

Examples:

Algorithm to Learn Structure

```
procedure LearnStructure(LabeledSet, Ops)
  ValidSet  $\leftarrow$  1/3 of LabeledSet
  TrainSet  $\leftarrow$  LabeledSet - ValidSet
  CurModel  $\leftarrow$  the simple model
  Keepers  $\leftarrow$  {CurModel}
   $I \leftarrow 0$ 
  while  $I < 20$  and CurModel has fewer than 25 states
    Candidates  $\leftarrow$  { $M \mid M \in \text{op}(\text{CurModel}) \wedge \text{op} \in \text{Ops}$ }
    for  $M \in$  Candidates
      score( $M$ )  $\leftarrow$  average of 3 runs trained on
        TrainSet and scored for F1 on ValidSet
    CurModel  $\leftarrow$   $M \in$  Candidates with highest score
    Keepers  $\leftarrow$  Keepers  $\cup$  {CurModel}
     $I \leftarrow I + 1$ 
  for  $M \in$  Keepers
    score( $M$ )  $\leftarrow$  average F1 from
      3-fold cross-validation on LabeledSet
  return  $M \in$  Keepers with highest score
```

Background:

- ValidSet: All of the labeled documents set aside for the validation of the model
- TrainSet: All of the labeled documents set aside for the training of the model
- CurModel: Used to store the model on which the operations are performed
- Candidates: All the candidate models in a given iteration to be scored. Populated by applying the operations to the current model.
- Keepers: Used to store the highest scoring models for each iteration of the hill-climbing

Start with a set of labeled documents

Set aside 1/3 of them for the validation stage of the model

The other 2/3 are then used to “train” the model. In this algorithm, however, the training of the model is really developing the underlying HMM structure used to extract the information.

Initialize the algorithm by setting the current model to the simple model, Keepers to have just the simple model, and the iteration number to be 0

Candidates variable is populated by performing every possible operation to the current model.

Performance Metric

- Must identify target or decline to perform extract
- Precision
- Recall
- F1
- Multiple predictions

Further talking point on first bullet is that only one correct answer exists.

Precision – Number of correct extractions divided by the number of documents for which the learner issued any prediction

Recall – Number of correction extractions divided by the number of documents containing one or more target functions

F1 – Harmonic mean of precision and recall

Multiple predictions – If the learner issues multiple predictions for a document, only the one with the highest confidence is considered.

What is Meant By Training Model?

- Transmission & emission probabilities
- Estimated with labeled data and Baum-Welch algorithm

Q: What is meant by training the model? Or similarly, what is being training?

A: Transmission & emission probabilities

The first step in doing this is that the data need to be labeled. Labeled training data consists of a sequences of words with the target words already identified.

- Sometimes this yields a unique path
- Much more often, however, Baum-Welch needed to iteratively estimate the parameters and fill in the missing path.

This fits into the larger context of structure and parameter learning that Mike will cover in more detail with regards to Bayesian nets later in the lecture.

Data Sets

- SA: Speaker Announcement
 - speaker, loc
- Acq: Corporate Acquisition Article
 - acquired, dlramt
- Jobs: Usenet Job Announcement
 - company, title
- CFP: Call for Papers
 - conf, deadline

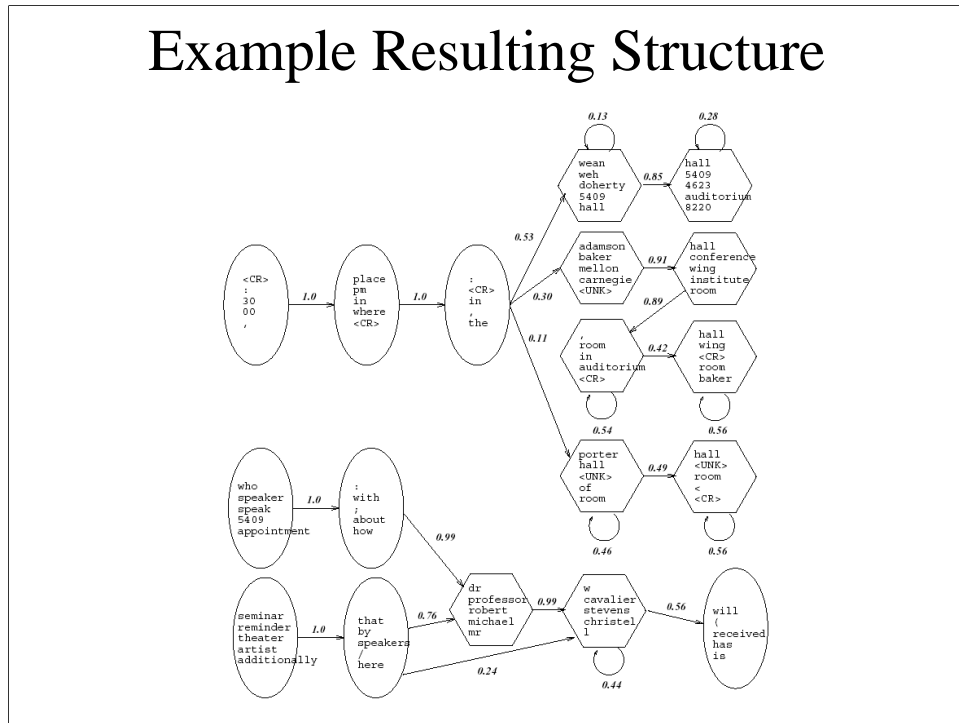
Experiments run:

- SA: A collection of 485 seminar announcements posted electronically at CMU. Fields include speaker, the name of the speaker at the seminar, and location, the location (e.g., room number) of the seminar.
- Acq: A collection of 600 Reuters articles detailing corporate acquisitions. Fields include acquired, the name of the company to be purchased, and dlramt, the purchase or estimated purchase price.
- Jobs: A collection of 298 Usenet job announcements. Fields include company, the name of the company seeking to hire, and title, the job title.
- CFP: A collection of 363 Internet “Call for Paper” announcements in ASCII format. Fields include conf, the name of the conference, and deadline, the full-paper submission deadline.

Additional Background info:

- The document collection is partitioned several times into a training set and a testing set. The learner is trained using the training set and its performance is measured using the testing set.
- Training sets used to both select a model structure and to set its parameter.
- Training and testing sets are roughly equal size, except for jobs, which were 90% training and 10% testing. This was done for comparison purposes with a previously published paper.

Example Resulting Structure



As stated before, prefix and suffix nodes are circular/ovular while the targets are hexagonal.

Transition between the states are labeled with the associated probability.

Only probabilities ≥ 0.01 are shown

Top graph represents the location field of the seminar announcement problem.

•Prefixes

- Figure suggests a single, relatively short prefix context is needed.
- Only a single prefix is retained
- Suggests unambiguous and length invariant types of language leading up to the location
- Phrases include:
 - “Place:”
 - “Where:”
 - “in the”
 - “00 PM”

•This also shows that the location is usually preceded by time

•Targets

- Partitioned into three parallel paths, top and bottom have a length of 2 and middle has a length of 4

HMM Summary

- IE, sparse extraction problem
- Learn an HMM structure to perform IE
- Context-based alterations to traditional HMM
- Hill climbing method to learn structure
- Baum-Welch to learn parameters
- Experiments and Results

We first established that the problem was an information extraction problem from sparse, human-readable text to extract target information.

Specifically, the authors sought to learn an HMM structure that performed this type of IE with maximum accuracy.

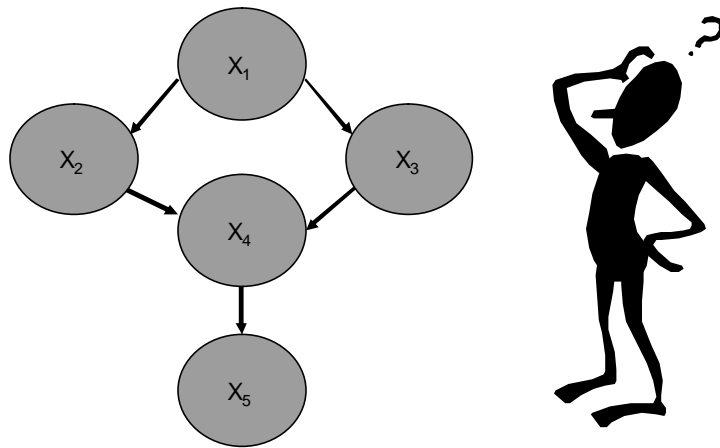
With this application in mind, we then established how HMMs are usually used in IE.

From there, we said that the two parts of creating an HMM are to first create the structure and then learn the parameters. We then defined a new variant to the traditional HMM structure that denoted target states and used the context of the words surrounding these targets to aid in the extraction accuracy. We then discussed the addition to the traditional HMM that resulted from this alteration.

We then introduced the big picture of the algorithm used to create the new HMM structure and stepped through each line.

We quickly established that Baum-Welch was used to obtain the parameters and then finished by discussing the experiments run on the algorithm and there associated results.

Bayesian Nets



I now introduce Bayesian Networks as a powerful technique used to model and visualize the complex causal relationships between variables within the domain of interest. These graphical structures, also known as belief networks, allow us to make educated guesses about the value or distribution of these variables, given evidence in the form of observations. This area is a relatively recent development in research, and is still a very hot topic today. After presenting the fundamentals of Bayesian Net Inference, I will present an advanced technique that has been pioneered in recent years, which involves the generation of the conditional probability information from statistical data.

But first, let me move into a quick review of Baye's rule...

Bayes Rule

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

$$\textit{posterior} = \frac{\textit{likelihood} \times \textit{prior}}{\textit{evidence}}$$

Recall from intro Probability class the definition of Bayes rule, which gives us a way to quantify the conditional probability of occurrence of a particular event A, given we have observed some new evidence B. A good way to look at this Rule is the second form here, where the posterior probability is computed by the product of the prior probability and the likelihood, divided by the unconditioned probability that we see a particular observation. The basic idea is that we would like to update our beliefs about the world given some evidence.

Bayes Rule: An Example

- Event A: Snow in Boston
 - Event B: Season is Winter
 - We want $P(A|B)$
 - *Prior*: $P(A) \sim .05$
 - *Evidence*: $P(B) \sim .25$
 - *Likelihood*: $P(B|A) \sim .5$
- Therefore, $P(A|B) = (.5*.05)/.25$
 $=.1$

Just a quick example of Bayes rule in action: suppose we have two events, Event A corresponds to an event that on a given day it snows in Boston. Event B corresponds to Season being Winter.

We know the a priori probabilities for each of these, based on statistics, so we can then move toward a conditional understanding one given the other.

But what happens when we are given additional information?

What about related events?

- Want: Safe Road Trip
- Safe Road Conditions?
- Raining?
- Snowing?

Now suppose you'd like to plan a road trip, but you're a little concerned the road conditions will be too dangerous. How can we reason about the decision to proceed, based upon related information? Snow, Rain, Time of year, road conditions, etc. all factor into our decision, but the relationship between these variables is not immediately clear.

Representation of Information

Discrete or Continuous Variables

X1: Season {Winter, Spring, Summer, Fall}

X2: Snowing {t, f}

X3: Raining {t, f}

X4: Good Road Conditions {t, f}

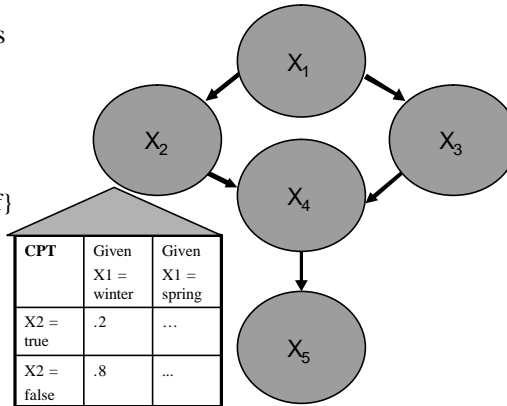
X5: Safe Trip {t, f}

Connections Between Variables

Conditional Probability Tables

$P(X2 = t \mid X1 = \text{Winter}) \dots$

Conditional Independence



How is information represented in a Bayesian Net?

- 1) All variables are graphically represented as nodes in this graph
- 2) The relationships between these nodes is represented as connections between these variables. When a parent points to a child, it indicates a direct relationship.
- 3) Finally, the conditionally probabilities are represented by conditional probability tables, for the case of discrete variables, or conditional density trees as I will describe in a few slides.

By modeling parent/child relationships as singular edges, we have simplified the problem to reasoning about pairwise relationships

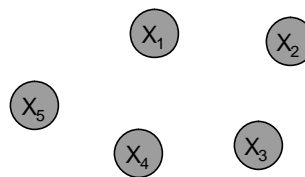
Generating a Bayesian Net

Steps:

- *Declare variables of interest*
- Connect variables in acyclic graph structure
- Approximate Conditional Probabilities

Variables:

- X_1 : Villain is a passive player
- X_2 : % of Hands Played by Villain
- X_3 : Villain is a 'rock'
- X_4 : Villain check-raises turn
- X_5 : Opponent has a full house



Now that we know the nature of the type of information represented in a Bayesian net, I would like to describe how we go about learning the structure and parameters for this graphical method.

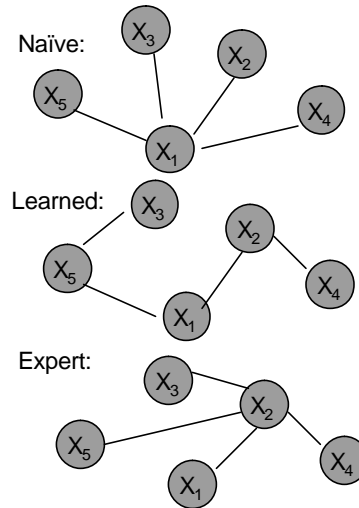
Suppose you are given a database of previously played poker hands, and you'd like to learn an appropriate Bayesian net for reasoning about hidden state in any given hand, so that we may make better decisions. Let's go through this exercise step by step...

Step one is to declare the variables of interest. Notice here that we have both continuous and discrete variables.

Generating a Bayesian Net

Steps:

1. Declare variables of interest
2. *Connect variables in acyclic graph structure*
3. Approximate conditional probabilities



Step two in the process is to connect all variables of interest in an acyclic graph. This process is called structure generation. A number of techniques have been published on how to generate these structure. For a reference to some of these advanced techniques, see Andrew Moore and Scott Davies work at CMU.

For the sake of time, I will only describe the three general classes of structure learning methods.

- 1) Naïve – structure learning using a Naïve approach yields a structure where all variables influence the variable of interest directly.
- 2) Learned - This is an advanced technique in research. By using search algorithms and an evaluation of yielded structures, we can potentially learn accurate structures for these Bayesian nets
- 3) Expert – Humans that have knowledge of the domain of interest can also generate these structures. However, relying on this approach can yield a number of problems. For example, some domains do not have human experts, while other domains have so many variables of interest, that the complexity exceeds the abilities of the domain experts.

Generating a Bayesian Net

Steps:

1. Declare variables of interest
2. Connect variables in acyclic graph structure
3. *Approximate conditional probabilities*

Problem:

Mixed Continuous and Discrete

Solution:

Conditional Density Trees: **Next!**

Once we have information about the structure of our Bayesian net, Step three in the generation of a Bayesian net is the estimation of conditional probability distributions for each node, given information about the parents of each of those nodes. Although I have chosen not to focus on advanced techniques for structure learning, I will be describing a number of recent developments in interpolating these conditional density trees.

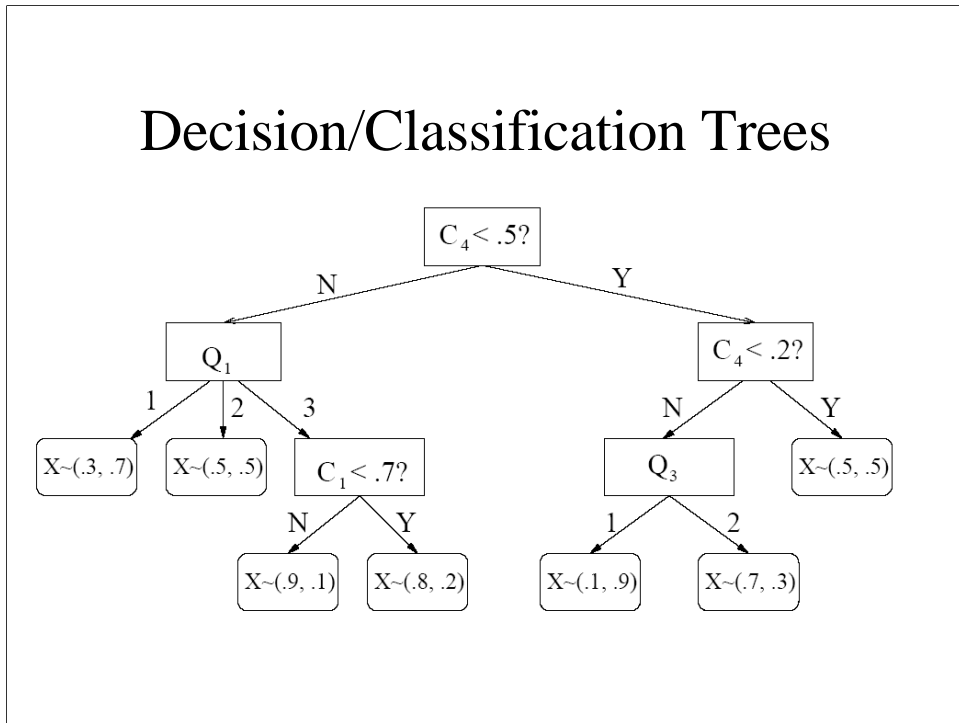
Interpolating Conditional Density Trees

- Mix Continuous and Discrete variables
- Decision Trees for Conditional Distribution
 - Tree-growing algorithms
- Advanced Technique: Approximately Conditionalized Joint Density Trees [2]

So for every node in our Bayesian net, we need to learn an accurate representation for the distribution of that variable, given information about all parents. This can be done using joint distributions or conditional distributions. The difficulty of this problem increases when you introduce continuous variables. To address this issue, I will describe a number of tree-based density estimators and a tree-growing algorithm.

The goal is to have a fast and accurate way to generate these structures given our training data, so I will also demonstrate one advanced technique that was published in 2002 out of CMU for doing so.

Decision/Classification Trees

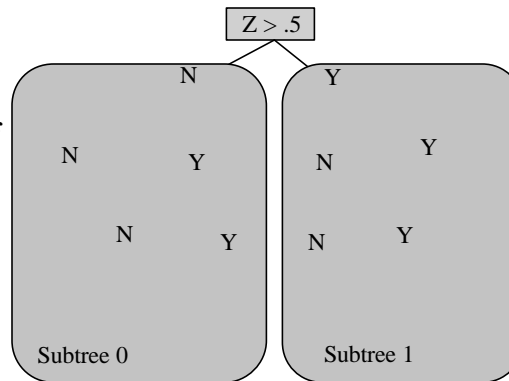


A fundamental way to represent conditional distributions over continuous and discrete variables is through the use of decision, or classification trees. Our goal is to come up with a concise way to represent the conditional distribution of the variable of interest, given one or more parent variables.

By branching on every all parent variables, we are able to generate the appropriate restrictions with which our probabilities will become conditioned. Note that at each leaf in the tree, the conditional distribution is represented mathematically or through a lookup table.

Stratified Conditional Density Trees

- Tests on all Parents
First, $\vec{\Pi}_i$
- Tests on Variable of Interest Last, X_i
- Subtrees, $P(X_i|\vec{\Pi}_i)$
- Accurate, but
Computationally
Expensive to Learn!



The key to represented conditional probabilities is to branch on all parent variables first, then create subtrees which represent all information about the variable of interest given those parent conditions. This is known as *stratification*.

These stratified trees are very accurate for representing the conditional distributions of variables, but they are computationally expensive to generate.

Joint Density Trees

- Joint Distributions at Leaves
- \vec{S}_i : all variables
- \vec{C}_i : all continuous variables
- \vec{D}_i : all discrete variables
- l : single leaf
- Leaf estimation techniques: Constant, Gaussian, Exponential, Linear, Multilinear

One compromise is to generate Joint Density Trees instead of Conditional Density Trees. Given these structures, we can extract the same information about conditional probabilities using Bayes Rule.

It turns out that approximating conditional densities is through the use of Bayes Rule and Joint Density Trees is more accurate than computing the conditional density information from CDTs directly! In addition, these trees are much less computationally intensive to generate than the Stratified CDTs I just spoke about.

There are a few key differences between CDTs and JDTs, as highlighted by the following notation:

The important thing to take away is that once we have branched on all the parents, the leaves are estimated using one of the following techniques: constant, gaussian, exponential, linear, multilinear. These mathematic representations are easier to generate for joint densities than for single densities.

Tree-Growing Algorithms

```
function TreeGrow (data)
  IF data is leaf
    return "Leaf distribution";
  ELSE
    choose branching variable and threshold
    for each child
      set childpointer = TreeGrow(data)
```

Regardless of whether we are generating conditional distribution trees or joint distribution trees, the methods described in this paper have focused on a top-down approach to generating these decision tree structures. Here is an outline of the general recursive algorithm for creating the structures:

The basic idea is that if you can recognize a leaf, you can use one of the previously mentioned estimators to come up with a distribution for it. The techniques for doing so could be the focus for their own lecture and are thus beyond the scope of this discussion. When a leaf is not detected, one must decide how to branch. Coming up with the thresholds on how to split a continuous variable have been investigated in a number of papers. For discrete variables, we just branch on every possible value of that variable then recursively call the function on each branch.

Conditional Density Trees from Joint Density Trees

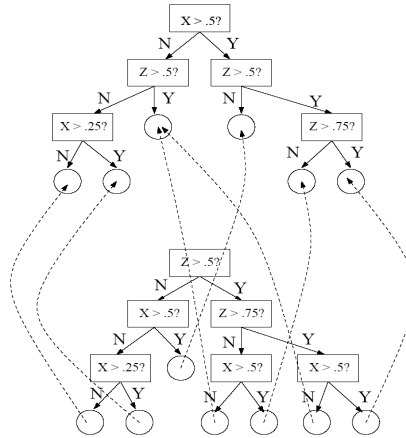
$$\begin{aligned}P(x_i|\vec{\pi}_i) &= \sum_l P(l|\vec{\pi}_i) \cdot P(x_i|\vec{\pi}_i, l) \\&= \sum_l \frac{P(l) \cdot P(\vec{\pi}_i|l)}{\sum_{l'} P(l') \cdot P(\vec{\pi}_i|l')} \cdot P(x_i|\vec{\pi}_i, l) \\&= \frac{P(l_c) \cdot P(\vec{\pi}_i|l_c) \cdot P(x_i|\vec{\pi}_i, l_c)}{\sum_{l'} P(l') \cdot P(\vec{\pi}_i|l')}\end{aligned}$$

- More Accurate than Computing CDT Directly!
- Denominator Computation is Expensive

This equation gives us a way of obtaining conditional probabilities from joint distributions. Just to reiterate, the challenge is to grow the tree-structure quickly, then extract *conditional* probabilities from that structure. Although JDTs are easier to generate, computing the denominator for this expression is expensive because we must search for the leaves of interest, then sum over all of their products.

JDT Speedup: Auxiliary Tree

- FRACTURE and COLLAPSE procedures



To speed up this evaluation, we generate an auxiliary CDT, but do not perform estimation on the leaves. Instead, we generate pointers to appropriate estimators on the JDTs. The procedure for generating these substructure CDTs are described in the full version of this paper, Scott Davies PhD dissertation, under FRACTURE and COLLAPSE procedures.

Approximately Conditionalized Joint Density Trees

- Generate auxiliary tree to speed up x2 by approximating
- Approximate Conditional Probability

$$\begin{aligned} P(x_i|\vec{\pi}_i) &\approx \frac{P(l_c) \cdot \hat{P}_s(\vec{\pi}_i|l_c)}{\sum_{l'} P(l') \cdot \hat{P}_s(\vec{\pi}_i|l')} \cdot P(x_i|\vec{\pi}_i, l_c) \\ &= \alpha_c^s P(x_i|\vec{\pi}_i, l_c) \end{aligned}$$

- α_c^s . Constant

As I mentioned earlier, generating conditional probability distributions from a joint distribution tree is very expensive. In addition to generating smaller auxiliary trees, an additional speedup is introduced in the paper I have investigated. This speedup is based upon the approximation, \hat{P} , that the probability of occurrence of a particular set of parent assignments can be estimated using the average over a few similar leaves. Once this approximation is made, we can describe it as a constant α_c^s and use it throughout the remainder of the calculations.

Bayesian Nets: Future Directions

- More Conditional Density Estimation
- Structure Learning
- Learn from Large Datasets Quickly

The techniques I have just described represent a significant movement in Bayesian Nets toward increased accuracy and computational speedup of learning the conditional probabilities given a known structure. As I alluded to previously, there is a significant amount of research being dedicated to the automatic generation and evaluation of the structure of these graphs. Finally, a theme throughout these and many other statistical inference tools is always learning your models quickly based upon very large data sets.

Bayesian Nets Summary

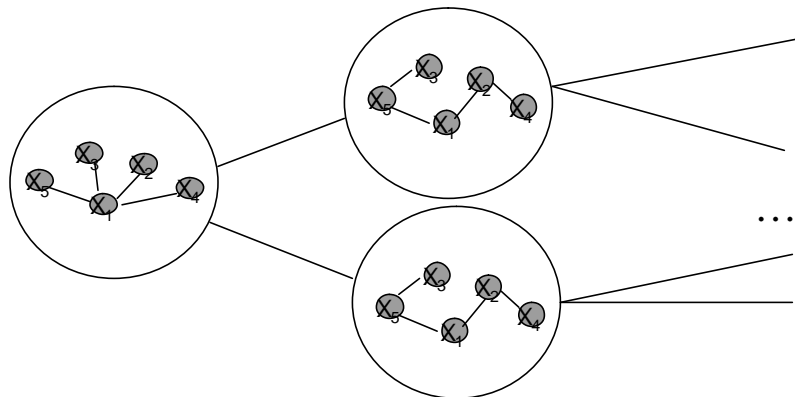
- Graphical Method: Conditional Probabilities
- Learning Bayesian Net Structure and Parameters from Statistics:
 - Step 1: Declare variables as nodes
 - Step 2: Connect nodes in acyclic structure
 - Step 3: Determine Conditional Probabilities
 - Conditional Decision Trees
 - CDTs from Joint Decision Trees

I would like you to take home the following points about Bayesian Nets;

- 1) They are *graphical* methods, similar to HMMs introduced by Brian, for representing probabilistic information. These techniques aid in making inferences based upon evidence in a world with unknown information.
- 2) These tools can be learned in a given problem domain given statistical samples in the form of mined data sets. I have gone through a number of advanced techniques for generating these, and broken up the process into three phases:
 - i) Declaring the right variables of interest
 - ii) Using a Naïve, Expert, or Learned method, find a way to connect up the relationships between the variables of interest
 - iii) Finally, use statistics to infer the conditional probabilities for a given structure. Advanced Technique: Decision Trees!

BN + HMM = Multivariable Inference

- Bayesian Net Variables = State

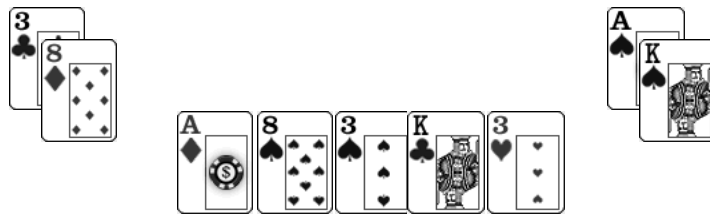


Recall that for Hidden Markov Models, we typically have a single variable, State, representing each state in the model. However, in any given state in poker, we have a number of variables which influence the observations we see. We would like a way to represent this multivariable state information as a composite single state variable, and this representation comes in the form of Bayesian Nets embedded in the Hidden Markov Model.

These embedded nets provide us with a way to represent a number of quantities at any given time, some of which we may only have a probabilistic estimate of. By having a Bayesian Net represent the state, we generate a more dynamic model of the system upon which we are attempting to make inferences.

Level 0 Reasoning: Hand Strength

- Hand Rank by Enumeration
- Probabilities of Improvement
- No Hidden Information



Now that we have described the advanced topics for reasoning about hidden information, we would like to provoke some of the questions that will be driving our final project.

As I described earlier in the Multi-Level Reasoning Model, Level 0 Reasoning in holdem is a simple lookup task. Source code can be downloaded from the web for this level of reasoning. The real inference problems come when you begin looking at...

Level 1 Reasoning: Opponent's Hand

- Opponent's cards are not known
 - Hidden information
- Have known information
 - Your cards, community cards
 - Opponent-specific information
 - Previous hand history
 - Skill level
 - Type of player
- Have observations
 - Betting that has occurred in current hand
 - Could be noisy (bluffing)

Level 1 Reasoning. How can we infer information about an adversary's hand, based upon evidence from his/her betting patterns?

We've already described the nature of the problem in terms of what we know, and what we can observe.

Level 1 Reasoning: Opponent's Hand

- Infer initial state
- Form hypotheses about information
- Test each hypothesis as more info becomes available
- If inconsistent, modify model

The way humans attempt this level of reasoning is often done through hypothesis testing, and that is precisely what we would like to obtain using HMMs. We start with a basic estimate of the range of hands our opponent may have. As more information and observations become available, we update our model of the distribution of initial states. Using HMMs, we model the transitions between various hand states. Using Bayesian Nets, we will make inferences about every piece of information we know about the hand and our opponents. Combining these techniques will be the thrust of our final project.

Project Goals

- Learned Hidden Markov Model from Poker Database
- Learned Bayesian Net from Poker Database
- Human-Competitive Level 0 and 1 Reasoning

The existing research in poker is concerned with developing a bot that can be competitive with and ultimately beat human players. However, we feel that in order for this goal to be accomplished, the bot must first must have the foundation of level 0 and level 1 reason. That is, before it can be successful able deciding how to play, it must first be able to know what it has and infer what it's opponent has.

Towards this end, we are going to use HMM to extract information about hand histories from various on-line databases. Then, we will use this information to learn a Bayesian net, which will model a given opponent. The Bayesian net then is used to train the probability of output for the states in the HMM representing the available poker hands shown in the Hold 'em demo earlier in the presentation.

In review, we have a HMM to model the Hold 'em hand. The initial states are unknown and are formed by the hole cards. The transition from any pair of cards to any subsequent hand is predefined ahead of time, so we don't need to know the transition probabilities. However, we do need to know the probability of an observation for a given state. This gap will be filled in by the Bayesian nets. The Bayesian nets get their information from the online poker databases. To extract this information, we use an HMM in the manner described in the advanced lecture.

Summary

- Texas Hold'em Demo
- Reasoning Methods for games
- HMM Inference and Learning
- Bayesian Nets Inference and Learning
- Project Goals

Questions?

References

- [1] Freitag, D., & McCallum, A. (2000). *Information extraction with HMM structures learned by stochastic optimization*. Proceedings of the Eighteenth Conference on Artificial Intelligence (AAAI-2000).
- [2] Davies, A., & Moore, A.(2002). *Interpolating Conditional Density Trees*. Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2002.