USING HIDDEN MARKOV MODELS (HMMS) TO UNDERSTAND STUDENT LEARNING ACTIVITY

By
Won Ng
whn@cs.washington.edu

Advised by Professor Steve Tanimoto

Department of Computer Science
University of Washington

18 June 2004

Introduction

Data mining to extract information from log files is currently a very active research topic. One particular area that has been receiving attention recently is examination of log files generated by students using computer-based learning tools. This unobtrusive educational assessment is meant to complement more traditional methods such as multiple-choice tests given to the students in an effort to evaluate the effectiveness of the online-learning environment and to suggest possible changes to the educators' teaching approach.

Since manually reviewing each log can be a long and involved process, we investigated the use of the statistical method of hidden Markov modeling to analyze logs. To date, Hidden Markov Models (HMMs) are used for the most part to improve machine speech recognition and to analyze protein sequences. The log files studied for this project were generated by college freshmen and sophomores using a digital image processing application called PixelMath available through the online learning environment INFACT. The goal of this project is two-fold: (1) to discover what kind of information about students' learning patterns can be extracted from these logs using HMMs, and (2) on a higher level, to explore the potential of using HMMs in educational assessment. We experimented with and modified the model parameters to determine the types of events in the log files that yield interesting or valuable results.

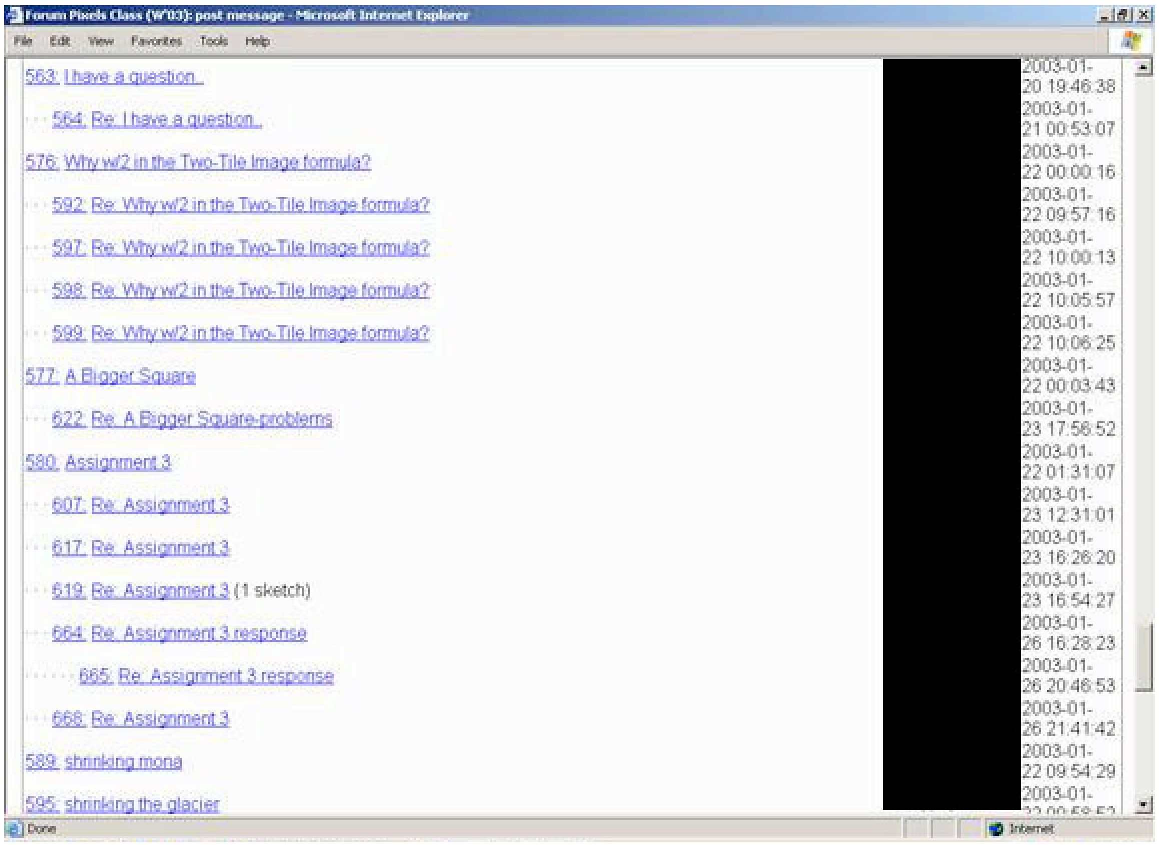
Learning Environment

INFACT (Interactive Networked Facet-Based Assessment Capture Tool) is developed at the University of Washington. INFACT has a variety of educational tools, including INFACT-FORUM, a threaded discussion forum for instructors and students to post questions and comments, shown in Figure 1.

Another important tool available through INFACT is PixelMath, an application to process images using a graphical calculator, shown in Figure 2. Images can be loaded and modified by applying formulas entered into the calculator. The destination field (Dest) in the calculator specifies which image window will be affected by the formula. The Src1 and Src2 fields denotes which image(s) to use in the formulas Source1(x,y) and Source2(x,y) respectively.

PixelMath also allows for other operations such as checking the dimensions of the image, opening multiple images, creating a new blank image, selecting an area of the image, zooming, and scrolling. Placing the cursor over a pixel will show its *xy*-coordinates and the RGB values of that pixel. PixelMath also includes Scheme to allow certain image changes using a programming language.

The sequential events of each PixelMath session are stored in a log file. Typical events that are logged include: calculations entered into the calculator, zooming in and zooming out, and undo. PixelMath can be used to perform color operations such as brightening an image, geometric transformations such as scaling or rotating an image, and other image manipulations.



Forum Pixels Class (W'03): post message - Microsoft Internet Explorer

File Edit View Favorites Tools Help

563: I have a question.	[redacted]	2003-01-20 19:46:38
··· 564: Re: I have a question.	[redacted]	2003-01-21 00:53:07
576: Why w/2 in the Two-Tile Image formula?	[redacted]	2003-01-22 00:00:16
··· 592: Re: Why w/2 in the Two-Tile Image formula?	[redacted]	2003-01-22 09:57:16
··· 597: Re: Why w/2 in the Two-Tile Image formula?	[redacted]	2003-01-22 10:00:13
··· 598: Re: Why w/2 in the Two-Tile Image formula?	[redacted]	2003-01-22 10:05:57
··· 599: Re: Why w/2 in the Two-Tile Image formula?	[redacted]	2003-01-22 10:06:25
577: A Bigger Square	[redacted]	2003-01-22 00:03:43
··· 622: Re: A Bigger Square-problems	[redacted]	2003-01-23 17:56:52
580: Assignment 3	[redacted]	2003-01-22 01:31:07
··· 607: Re: Assignment 3	[redacted]	2003-01-23 12:31:01
··· 617: Re: Assignment 3	[redacted]	2003-01-23 16:26:20
··· 619: Re: Assignment 3 (1 sketch)	[redacted]	2003-01-23 16:54:27
··· 664: Re: Assignment 3 response	[redacted]	2003-01-26 16:28:23
······ 665: Re: Assignment 3 response	[redacted]	2003-01-26 20:46:53
··· 668: Re: Assignment 3	[redacted]	2003-01-26 21:41:42
589: shrinking mona	[redacted]	2003-01-22 09:54:29
595: shrinking the glacier	[redacted]	2003-01-

Done Internet

Figure 1: INFACT-FORUM. The name of the posting author has been obscured to protect the participants' identity.

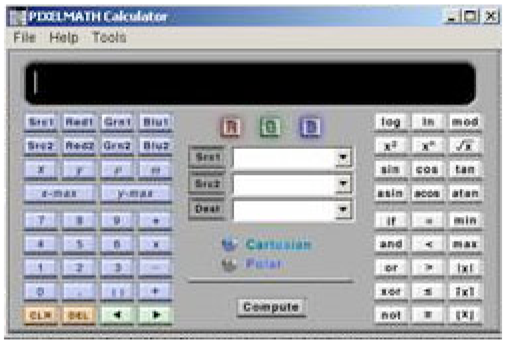


Figure 2: PixelMath Calculator

Hidden Markov Models

Hidden Markov Models are based on Markov Models, which are statistical models equivalent to probabilistic finite state machines. State transitions generate an observation sequence. For example, weather patterns may be represented by the Markov model shown in Figure 3.

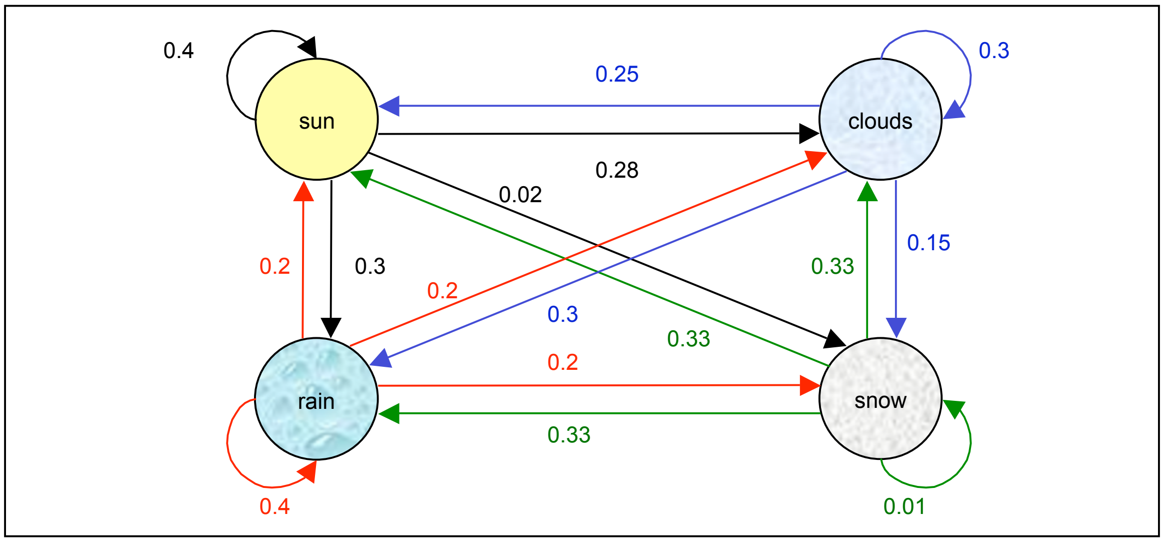


Figure 3: A Markov Model representation of the weather patterns

The weather of each day can be represented by one of four states: sunny, rainy, cloudy and snowing. The state is determined by the "symbol" that is observed at noon of each day: sun, rain, clouds or snow. This assumes that only one symbol is observable at a given time, so both the sun and clouds will not be seen at the same time. The sequence of states observed each day for a number of days form an observation sequence of the weather. From the observation sequence (or a set of observation sequences), the probabilities of state transitions can be calculated to produce the model. In this example, if one day is rainy, the next day has a 40% of raining again. Then the model can be used to predict observation sequences.

For a Markov Model, there is a one-to-one relationship between the state and symbol, meaning each state (e.g. rainy) is represented by only one symbol (e.g. rain), and each symbol can only be generated by one state. In other words, it is rainy if and only if there is rain. The state and its corresponding symbol are equivalent.

For a Hidden Markov Model, the states are *hidden*, so only the symbols are directly observable. Each symbol may be caused by any of states, so there is a probability distribution of each state causing a symbol. It may be able possible to infer a state from the observed symbols, but it is not possible to determine definitely what state produced the symbol.

The weather model can be extended to an HMM by relaxing the one-to-one assumption. Even on sunny days, clouds may be observed at noon. Therefore, if there are clouds at noon, there may be a 60% chance that the day is cloudy, a 30% chance it is sunny, a 20%

chance it is rainy and a 10% chance it is snowing, so it may be likely the day was cloudy, but the weather of the day cannot be absolutely determined based on the observation at noon.

The model parameters are: (1) N, the number of states, (2) M, the observation symbol set, (3) A, the state transition probabilities, (4) B, the symbol distribution probabilities, which gives the likelihood that given a state the symbol will be seen, and (5) π, the initial state probabilities. There are three useful problems that HMMs can solve [2].

1. Problem 1 is the evaluation problem, which determines the probability that an observation sequence was generated by a given model. This problem is especially useful for classification, where each model may represent a category, and the sequence most likely belongs to the category characterized by the model which yields the highest probability. The Forward-Backward algorithm is one efficient solution to the problem.
2. Problem 2 is the decoding problem, which finds the most likely state sequence for the corresponding observation sequence, given an HMM model. One solution is the Viterbi algorithm, which optimizes the single best state sequence by choosing the state most likely to have generated the observation symbol in the sequence. A shortcoming of this approach occurs since the Viterbi algorithm maximizes the number of correct individual states, so it may result in some state transitions are not possible according to the model.
3. Problem 3 is the learning problem used to train the models based on an observation sequence. The Baum-Welch method reiteratively adjusts the model parameters A, B and π to maximize the probability that the model generates the observation sequence. This provides a way to create meaningful models, starting with initial guesses for the distributions. The Baum-Welch method can be modified to train a model based on a set of observation sequences instead of only one sequence.

HMMs are appropriate to the problem of analyzing log files for a number of reasons. HMMs obey the first-order Markov hypothesis that the next state depends only on the current state, so the history of the sequence can be discarded. Also, HMMs can handle analyzing observation sequences of variable length [2], making them a suitable choice for student log files which show a diverse set and sequence of events.

While the events recorded in the log file are observable, the student's state of mind when executing the action is not obvious. Using the Viterbi algorithm, it may be possible to conjecture the student's states of mind based on the log file. Although this is only an educated guess, it may still be useful to educational assessment.

HMM's flexibility in training data is also an asset because either single observation sequences of sets of sequences can be used in the Baum-Welch method. There may be only a limited number of log files available for training a model, but the model can be trained using only one sequence. If a model is likely to produce sequences exhibiting different patterns, then multiple sequences matching the model can be used together as a training set for the model.

Experimental Procedure

Data Collection

Log files of PixelMath sessions were chosen for the study. The log files were generated during the Winter 2003 and Winter 2004 offerings of GIS (General & Interdisciplinary Studies) 130: Pixels, Numbers, and Programs, which is only open to college freshmen and sophomores. The course description [4] is:

> Digital images and how they are represented, manipulated, and used. Using unique, state-of-the-art software, students create visual effects using techniques from mathematics, computing, and art. Topics include digital cameras, image enhancement, geometric distortions, 3-D perception and stereograms, anamorphic art, and scripting with a programming language.

No prior experience of computer proficiency was required. Class sessions were one-hour long and typically involved the instructors' presentation of material, a class exercise and time to do the activity during class.

Often, the students were given worksheets to complete using PixelMath. The worksheets described some image theory, and then they detailed a number of exercises that students should try using PixelMath. The instructor posted a question related to each exercise to the INFACT-FORUM, and when students completed the activity, they posted a reply. The log files of the PixelMath sessions were stored and indexed by the forum posting. This meant that if a student did not post to the forum, the log file for the session was not saved. Students were encouraged but not necessarily required to complete all the exercises and to post to the forum. Since PixelMath is available online, students had the option of continuing the activity outside of class.

An example of an activity involving geometric manipulations is "A Two-Tile Image." Students were asked to scale down two images, then create a new image with the shrunken images side-by-side, resulting in a two-tile image. A copy of the worksheet, "Making Big 'Tiled' Images" is included in Appendix A.

Figure 4 shows an excerpt from a student's log file for this activity. The "event type" indicates the type of operation and the "event attr-value pairs" provides a concise description of the event's parameters. The time of each event relative to the beginning of the session is also logged; the event start time and event end time are the same for PixelMath events because the events execute instantaneously.

Some of the logged events were automated processes that were triggered by the user's action. For example, when a student opened an image using the calculator's menu bar, three events were logged: (1) "CalcFrameMenu" showing the "Open" command was selected, (2) "OpenImageDialog" signifying the image chosen was the Mona Lisa, and (3) "OpenImageDialog" indicating the window number of the image. Another example is when the user input a formula into the PixelMath calculator. The event "CalcPaneCompute" was logged with the formula, Source1, Source2 and destination images information. Then the formula was "Parsed" to verify the correct syntax. If there was an error, the description noted the event as "Unsuccessful," else it was "Successful."

```
Event type: "CalcFrameMenu"
Event start time: 0.0
Event end time: 0
Event attr-value pairs: "Description:
java.awt.MenuItem[menuitem0,label=Open...]"

Event type: "OpenImageDialog"
Event start time: 5.0
Event end time: 5
Event attr-value pairs: "Description: filename=mona-rgb.jpg"

Event type: "OpenImageDialog"
Event start time: 8.0
Event end time: 8
Event attr-value pairs: "Description: WindowNum=1; File=mona-rgb.jpg"

Event type: "CalcFrameMenu"
Event start time: 17.0
Event end time: 17
Event attr-value pairs: "Description:
java.awt.MenuItem[menuitem1,label=New...]"

Event type: "NewImageDialog"
Event start time: 41.0
Event end time: 41
Event attr-value pairs: "Description: W=100; H=100; R=255; G=255;
B=255”

Event type: "CalcPaneCompute"
Event start time: 70.0
Event end time: 70
Event attr-value pairs: "Description: Src1=1: mona-rgb.jpg; Src2=1:
mona-rgb.jpg; Dest=2: Untitled Image; Formula=; CoordSys=CART; RGB=RGB"

Event type: "Parse"
Event start time: 70.0
Event end time: 70
Event attr-value pairs: "Description: Unsuccessful"

Event type: "CalcPaneCompute"
Event start time: 299.0
Event end time: 299
Event attr-value pairs: "Description: Src1=1: mona-rgb.jpg; Src2=1:
mona-rgb.jpg; Dest=2: Untitled Image; Formula=Source1(x,y)*x;
CoordSys=CART; RGB=RGB"

Event type: "Parse"
Event start time: 299.0
Event end time: 299
Event attr-value pairs: "Description: Successful"

Event type: "ImageCanvas"
Event start time: 307.0
Event end time: 307
Event attr-value pairs: "Description: WindowNum=2; ZoomIn:x=77;y=78"
```

Figure 4: Excerpt of a log file.

General Process
Pre-processing the data required converting the events into numerical symbols. The symbol set chosen depended on the goal of the experiment and the exercise analyzed. Perl filters were created to convert the log files into observation sequences. Some of the filters used are listed in Appendix B.

The number of states and their significance also had to be decided upon before building the default HMM models. Once the states and symbol set were determined, an observation sequence, or set of observation sequences, was used as training data to create the model. Models then could be used to solve Problem 1, the evaluation problem, or Problem 2, the decoding problem.

Experiment 1: Activity Classification
The first experiment tested whether HMMs could accurately classify the exercises represented by the log files. Log files from three activities were chosen: (1) “Making Purple,” from the “Additive Color Mixing” worksheet, which was one of the initial exercises using PixelMath, (2) “Brightness,” from the “Brightness, Contrast, Quantization and Thresholding” worksheet, and (3) “A Two-Tile Image” from the “Making Big ‘Tiled’ Images” worksheet, which introduced geometric manipulations. All three worksheets are included in Appendix A for reference.

Because the worksheets described multiple activities that students typically completed in one PixelMath session, the log files included events from more than one activity. These longer log files were used without segmentation into separate activities because (a) manually segmenting the log files into separate activities would be very time consuming, (b) it was difficult to determine the transition point from one exercise to the other for some of the log files, and most importantly, (c) longer log files provided more information to differentiate the types of activities from one another because each activity involved very distinct formulas and sequence of actions. “Making Purple” suggested changing the RGB values of a blank canvas by toggling the enable buttons on the calculator and inputting numbers as the formula. The log for setting the red value to 150 is:

```
Event attr-value pairs: "Description: Src1=1: Untitled
Image; Src2=1: Untitled Image; Dest=1: Untitled Image;
Formula=150; CoordSys=CART; RGB=R"
```

“Brightness” suggested either adding a constant value to the RGB values of an image or multiplying the RGB values by a value greater than one. Examples of the logs of these formulas are:

```
Event attr-value pairs: "Description: Src1=1: mona-rgb.jpg;
Src2=1: mona-rgb.jpg; Dest=1: mona-rgb.jpg;
Formula=Source1(x,y)+50; CoordSys=CART; RGB=RGB"

Event attr-value pairs: "Description: Src1=1: mona-rgb.jpg;
Src2=1: mona-rgb.jpg; Dest=1: mona-rgb.jpg;
Formula=Source1(x,y)*1.25; CoordSys=CART; RGB=RGB"
```

Another distinctive action for "Brightness" was the use of the "Undo" command, which was explicitly recommended in the worksheet.

For "A Two-Tile Image," images had to be scaled down to make a smaller version of the original. Once there were two small images, they were placed next to each other to create the tiled image. The logs of the two formulas are shown below.

```
Event attr-value pairs: "Description: Src1=1: mona-rgb.jpg;
Src2=1: mona-rgb.jpg; Dest=2: Untitled Image;
Formula=Source1((w1/w)*x,(h1/h)*y); CoordSys=CART; RGB=RGB"

Event attr-value pairs: "Description: Src1=1: Untitled
Image; Src2=3: Untitled Image; Dest=5: Untitled Image;
Formula= if x<w/2 then Source1(x,y) else Source2(x-w/2,y);
CoordSys=CART; RGB=RGB"
```

The formula for tiling was given in the worksheet, so almost all log files for this exercise included it.

The models

Numerous experiments were conducted by adjusting the number of states in each model, the symbol set used to represent the log file as an observation sequence, and the data used to train the model. The most successful experiment is explained here. Details of other experiments are included in Appendix C.

A total of eleven models were created, each trained using the Baum-Welch method with only one observation sequence. The starting models had equal distributions for A and π and estimated probabilities for B before training. Each of the three activities had an Expert model, trained using the observation sequence from a log file that strictly followed the instructions on the worksheet. I generated this log file myself to create a baseline model. Each activity also had a Student model based on one student's log files for each respective exercise. Only one observation sequence was used for training because the modification to the Baum-Welch method that allowed multiple training sequences was not yet implemented. This also served to test how sensitive the models were to the data. Because of this sensitivity, "skewed" data was also created to train three additional models. The skewed data was basically the Expert observation sequence modified with additional actions such as zooming and undo commands. This was meant to represent a student who may be considered as Almost Expert, but performed some extraneous actions.

"Making Purple" had another model trained on a shortened skewed sequence to test the effect of truncating a log file. "Making Purple" was chosen because, unlike the other two exercises, there is no clear ending point for this activity. Finally, "Brightness" also had another model trained using a shortened Student sequence because at the end of the activity, the student had experimented with formulas relating to geometric manipulations. Due to the presence of these calculations, the Student model for both "Brightness" and

"A Two-Tile Image" were similar, so the log file for "Brightness" was truncated to exclude the experimentation.

Each model had the same number of states and the same symbol set, differing only in the A, B and π distributions after training.

The models had four states:

1. color mixing (for "Making Purple" and "Brightness") or geometric manipulation (for "A Two-Tile Image)
2. analyzing
3. "mistakes"
4. set up

There were eleven symbols for the actions:

0. Short Zoom In (one to three continuous zoom in instances)
1. Short Zoom Out (one to three continuous zoom out instances)
2. Selection, which is when a user clicks and drags the mouse to select a region of the window
3. Undo
4. Domain Transformation Calculation, which is when a user modifies the (x,y) value in the source expression (e.g. `Formula=Source1(x/2,y)`), used for geometric manipulations
5. Other Calculation, including range transformation formulas that modify the result of the source expression (e.g. `Formula=Source1(x,y)+100`), used for color manipulations
6. New Window
7. Open Picture
8. Unsuccessful Parse
9. Long Zoom In (more than three continuous zoom in instances)
10. Long Zoom Out (more than three continuous zoom out instances)

The long zooming symbols were used in an attempt to make the observation sequences for each activity more uniform and similar to one other. The guess was that this would increase the overall probability for Problem 1, although for the goal of classification, only the relative probabilities of each model producing the sequence were necessary.

Results

Twelve observation sequences from the Winter 2003 Pixels class were tested, four of each activity, including the sequence used to train the Student model. One other observation sequence from another color manipulation worksheet, "Color Saturation Formula" was also tested. The results are shown in Table 1. Logarithmic values were calculated using the Forward-Backward algorithm to prevent underflow, so the more negative the number, the lower the probability of the model generating the sequence.

All the sequences except for the one from "Color Saturation Formula," which matched the modified Skewed "Making Purple," and a "Brightness" sequence, identified with the

Student model of the correct activity best. The overall accuracy, excluding the three sequences used to train the Student models, is 80%. The Student model for “Brightness” trained using the truncated log file yielded more accurate results than the Student model from the unmodified log file. Comparison between the two shows that the model using truncated data resulted in higher probabilities for “Brightness” sequences model and lower probabilities for “A Two-Tile Image” belonging to this model than the unmodified Student model. Without this modified model, one of the “Brightness” sequences would have been incorrectly classified as “A Two-Tile Image.”

Table 1: Classifying activities. Nine models were trained using Expert (E), Skewed (Sk), and Student (St) data. Model Sk’ for “Making Purple” used a shortened Skewed sequence. Model St’ for “Making Brightness used a shortened Student sequence. The first four sequences are from “Making Purple,” the next four from “Brightness,” and the last four from “A Two-Tile Image.” The first sequence of each activity is the sequence used to train the Student model. The PixelMath session number associated with the log file is listed in the left-hand column. The negative sign of the probabilities has been dropped for readability.

		Making Purple				Brightness				A Two-Tile Image		
#	length	E	Sk	Sk’	St	E	Sk	St	St’	E	Sk	St
52	90	349	71	88	**65**	1306	694	815	815	2049	736	705
32	102	460	142	153	**65**	1477	508	669	669	2307	518	481
37	89	278	78	91	**53**	1224	457	599	599	2026	502	477
44	105	116	42	54	**24**	1316	212	370	370	2395	263	209
415	58	838	237	202	378	275	162	**34**	140	1121	194	107
420	73	931	232	191	310	275	138	136	**101**	1548	213	126
426	76	1024	281	240	316	389	292	237	**218**	1617	353	279
436	75	942	313	276	421	389	230	234	228	1415	279	**147**
590	58	1119	471	446	744	1033	442	472	868	514	148	**76**
585	40	711	318	305	382	605	268	319	404	539	197	**128**
591	15	319	110	109	229	253	108	110	234	106	27	**21**
596	76	1493	422	418	896	933	376	400	828	903	249	**170**
131	77	748	207	**187**	361	631	288	373	317	1529	350	280

Experiment 2: A Two-Tile Image

There were two aims to the next experiment: (1) solve Problem 1 to classify log files into expertise levels (e.g. “Expert” or “Novice), similar to the previous classification experiment, and (2) solve Problem 2 to determine if there were any learning patterns depicted by the state transitions. The log files for “A Two-Tile Image” were chosen for analysis because this activity tested a common “pull vs. push” misconception, where students may confuse dividing instead of multiplying source values to shrink an image. This activity was also appropriate because the worksheet included clear instructions for creating the two-tile image, so there should not be large variations in log files for successful completion of the exercise.

In summary, the steps are:

1. Open a copy of the Mona Lisa.
2. Create a new blank image of size 100 x 100 pixels.
3. Set the Source1 window to the Mona Lisa and the Destination to the blank image. (This action is not explicitly recorded in the log file).
4. Input into the calculator the formula "Source1(x_{source} , y_{source})" where x_{source} is x multiplied by the ratio of the width of the Mona Lisa / the width of the blank image, and y_{source} is y multiplied by the ratio of the heights.
5. Make the second image: Open a copy of the Nisqually Glacier picture of Mount Rainier.

6-8. Repeat steps 2-4 for the Glacier image.

9. Create a new blank image of size 200 x 200 pixels.
10. Set the Source1 window to the small Mona Lisa, the Source2 window to the small Glacier, and the Destination to the new blank image.
11. Tiling by inputting into the calculator the formula "If x < w/2 Then Source1(x,y) Else Source2(x – w/2, y)."

This exercise can be completed in with a minimum of eight logged events. Figure 5 shows the result of the tiling process.

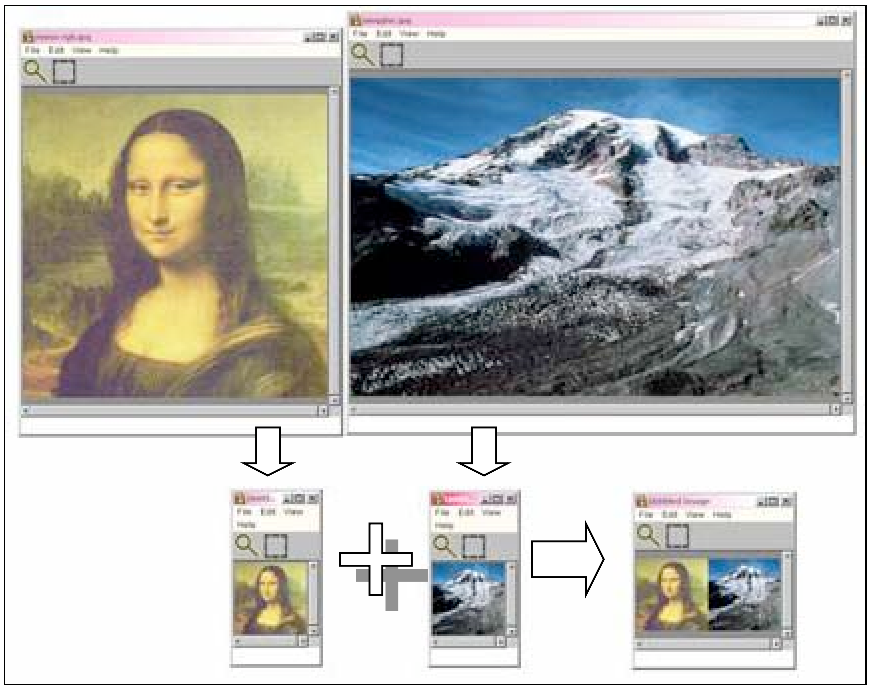


Figure 5: Making a Two-Tile Image from the Mona Lisa and Nisqually Glacier.

Because the worksheets included multiple activities, it was necessary to segment the log files into separate activities. This was done using a simple Perl script that identified a time difference of at least 200 seconds, or 3 minutes and 20 seconds, in between logged events. Then the segmented files were manually examined to verify or correct the accuracy of the segmentation. Appendix B lists the script used to truncate the data.

The models

The log files were manually reviewed to determine which commands were used, and the symbol set was modified to include the actions of interests. The major differences between the symbol set used in Experiment 1 and here is the division of geometric transformations into one of five categories:

1. "expertCalc," where students will use variables in their formulas, (e.g. `Source1(x*w1/w,y*h1/h)`)
2. "rightCalc," where students use the actual dimensions of the image (e.g. `Source1(x*422/100,y*420/100)`)
3. "endCalc," which is the tiling formula listed in Step 11
4. "push," which is the "push-pull" misconception of dividing instead of multiplying
5. "wrongCalc"

The description of the full symbol set is given in Appendix D.

There were four states representing possible student mindsets or goals while doing the exercise:

0. set up
1. correct calculations
2. confused
3. mistakes

Four models were created: Expert, Right, Confused and Mistaken. The Expert model had only three states because it did not include the confused state. As with the previous experiment, I artificially generated the data for the Expert model. The Right, Confused and Mistaken models were each trained on one student observation sequence. These sequences were chosen because based on manual inspection. The Right sequence only had two extraneous commands. This model was meant to represent a student who might have made a few mistakes or even was confused but proved to be a quick learner. The Confused sequence performed "lost" events such as checking the menu options and repeatedly entering formulas with syntax errors. The Mistakes sequence contained many wrong calculations and undos

A fifth model was added after some experimentation. The second Confused model, Confused' had equal probabilities for A, B and π. This was prompted by the difficulty of finding an observation sequence representative of a confused student since confused students may exhibit a range of behaviors, so the hypothesis was that there was an equal chance of the student performing any action.

Unlike Experiment 1, where the correct classifications were known, the categorization of student expertise level was subjective. In particular, it was very difficult to confidently judge from inspection of the log files whether a student was mistaken or confused, because often students displayed both "traits" dominate in both models: wrong calculations and zooming. The log files were classified as belonging to the Right model if the student completed the exercise with equal to or less than twenty logged events.

Results
The four observation sequences used for training the models, thirteen sequences from the Winter 2003 Pixels class, and four from the Winter 2004 Pixels class were tested. The results are shown in Table 2. The Viterbi algorithm was used to solve Problem 2 for four of the sequences that were manually classified as Right. Figure 6 shows these results.

Table 2: Expertise levels for "A Two-Tile Image." The bottom four results are for sequences from Winter 2004. The PixelMath session number associated with the log file is listed in the left-hand column. The negative sign of the probabilities has been dropped for readability.

#	Length	Expert	Right	Mistaken	Confused	Confused'
Expert	8	**8**	58	57	74	32
Right (593)	10	54	**10**	60	100	43
Mistaken (596)	49	971	615	**50**	200	87
Confused (599)	26	466	380	239	**15**	34
583	22	288	224	**118**	197	85
584	22	307	237	**136**	175	76
585	12	100	**56**	73	104	45
586	20	219	**108**	123	180	78
588	30	368	256	**164**	230	100
590	14	146	100	**91**	107	47
591	31	560	450	**220**	346	150
592	37	521	310	**281**	402	156
594	26	444	352	**146**	271	118
598	12	78	**34**	62	118	52
600	15	213	147	**45**	130	56
601	36	558	416	**275**	308	134
602	31	516	474	**217**	**217**	114
1276	13	168	123	**36**	104	36
1291	14	191	192	**118**	140	39
1305	16	237	172	**79**	94	44
1311	10	99	78	**17**	77	27

```
PixelMath #585     Logarithmic probability of sequence:  -56.6
Obs sequence:    7 6 15 5 15 13 10 7 6 10 6 12
State sequence: 0 0 3  2 3  2  1  0 0 1  0 1

PixelMath #586     Logarithmic probability of sequence:  -108
Obs sequence:    7 6 15 5 15 5 10 7 6 10 5 5 5 10 5 6 5 10 6 12
State sequence: 0 0 3  2 3  2 1  0 0 1  0 3 2 1  0 3 2 1  0 1

PixelMath #590     Logarithmic probability of sequence:  -102
Obs sequence:    7 6 14 14 15 15 15 15 10 7 6 10 6 12
State sequence: 0 0 1  0  3  3  3  3  1  0 0 1  0 1

PixelMath #598     Logarithmic probability of sequence:  -35.0
Obs sequence:    7 6 11 6 15 5 10 7 6 10 6 12
State sequence: 0 0 1  0 3  2 1  0 0 1  0 1
```
Figure 6: Most likely states for Right sequences.

All seventeen student log files tested matched the Confused' model best. The second most likely match for fourteen (82.3%) of the sequences was the Mistaken model. None matched the Expert or Confused model, and only three matched with the Right model as second-best. Because the log files were classified as Right based on sequence length, these were used to evaluate the accuracy of the HMM results. Not accounting for the Confused' model, three of the eight sequences (37.5%) were "correctly" classified as Right.

Three of the four Right sequences showed state transitions through all four states. The remaining sequence, which was classified as Mistaken, did not go through the confused state. For the explanation of observation symbols, refer to Appendix D.

Discussion

Experiment 1: Activity Classification

Without the second Student model for the "Brightness" activity, trained using a truncated log file, excluding the three sequences used to train the Student models and the one from the "Color Saturation Formula" activity, only eight of the nine sequences (89%), would have been accurately classified. Because the original log file included formulas that should have been unique to "A Two-Tile Image," it was more difficult to differentiate whether a log file belonged to "Brightness" or "A Two-Tile Image." With the second Student model, there was 100% accuracy. This shows the importance of choosing training data that is representative of the model.

The hypothesis that log files could be classified based on the exclusivity of the formulas was also supported by the 100% accuracy when considering only the Expert models (again, excluding the "Color Saturation Formula" sequence). The Expert models were trained using data that strictly followed the instructions on the worksheets, so did not include any extraneous or mistaken events and adhered to the presence of distinctive calculations for each activity. However, the observation sequences tested matched better with the Student models than the Expert models because the Expert log files only completed the first activity of each worksheet, whereas student log files typically included events from more than one activity. These continuous student log files could have been segmented into separate exercises through manual inspection or by automatic pre-processing if there was some noticeable ending point (for example, "A Two-Tile Image" had a distinct calculation used to tile the images), but by using the whole log, the sequences were longer and showed more distinct patterns and actions exceptional to each type of activity (additive color mixing, color manipulation, or geometric manipulation).

Longer sequences were not necessarily better for classification, as evidenced by the Student data used to train the "Brightness" model. Because the student had experimented with geometric transformations at the end of the session, the property of each type of activity showing only certain formulas was violated. This was part of the motivation for segmenting the log files for the second experiment.

Another note of caution is that an observation sequence does not necessarily have to match one of the available models. The probabilities of any models generating a sequence from a different activity should be very low. However, the “Color Saturation Formula” sequence matched the modified Skewed’ “Making Purple” best with a logarithmic probability of -187, which is comparable to the highest probability of other sequences with similar lengths. This is rather surprising and may be misleading because it seems that this sequence was confidently from the “Making Purple” exercise. One explanation for this mistake is due to the lack of fine distinction between the kinds of color manipulation formulas. All color mixing and color manipulating formulas were considered as “Other Calculation” and assigned the same symbol, so the log file for “Color Saturation Formula” appeared to belong to the “Making Purple” model. This emphasizes the need to include a level of detail (which was also exhibited by the initial experiments described in Appendix C), although it must be balanced by avoiding too much detail and complicating the model too much with a large symbol set.

The sequence from session #420 is another example of how the results may be misleading. The sequence was correctly matched with the Student’ model of “Brightness” with a logarithmic probability of -101, but the logarithmic probability from the Student model of “A Two-Tile Image” was only slightly higher at -126. If the Student’ model were not present, the activity would have been confidently misclassified because of the high probability given the sequence length.

HMMs proved useful in classifying activities, but the results were not completely accurate and problems occurred because of the cross-grained translation from log file to observation sequence and the inappropriate training data used for the “Brightness” Student model. Also, because only relative probabilities were considered when matching a sequence to a model, if the correct model was not included, some activities were misclassified with a high probability.

Experiment 2: A Two-Tile Image
All the seventeen observation sequences that were tested matched best with the Confused’ model, which had equal probability distributions for A, B and π. This is likely an indication that the models were over trained specifically to only the training sequence. Unfortunately, the matching of sequences to the Confused’ model does not yield any particular information regarding the sequences, so the following analysis is based on the second-best matches.

74% of the observation sequences tested were classified secondly as Mistaken, which was not surprising given the available data. It is important to note that since INFACT only stored the PixelMath sessions when students posted to the forum, if a student did not reply to the instructor’s post, his log file was not saved. For example, if a student ran out of time to complete the exercise during class, he may not have finished it outside of class and posted a reply. Since almost all the stored log files were by students who successfully finished the activities; students who may not have understood the material and were excessively confused may not have posted to the forum, so their log files were lost. Therefore, the pool of data included mostly log files by remaining students who completed the exercise after some trial-and-error. Unfortunately, this convention of storing log files only when there was a posting meant that the students who would probably most benefit from instructor intervention and guidance were not identified, because there was no log file to analyze and determine they may have needed help.

Unlike Experiment 1, where the Expert model could have been used for accurate classification of log files into activities, the Expert model for this experiment did not yield interesting or helpful results. No student log files matched this model because every student whose log file was tested performed some extraneous action. The closest log file had two extra commands: the student did not change the Destination to a new blank window, so he shrunk the original Mona Lisa and had to undo the action (PixelMath session #1311 from Winter 2004). This was technically a mistake, but did not reflect a misunderstanding of the concepts, only carelessness or perhaps haste in finishing the activity. However, instead of matching the Right model, which was meant to represent students who made some mistakes but in general understood the problem well and finished with minimal commands, this sequence matched the Mistaken model. Since 50% of the sequences manually classified as Right were misclassified by the HMM results, this may be another instance emphasizing the importance of choosing appropriate training data.

For Problem 1 and the classification of expertise level, the number of states and their interpretation may not have a significant effect on the outcome. In contrast, the states of the model, and what they represent, are extremely important to Problem 2. The results from the Viterbi algorithm shown in Figure 6 show that the states may have a significant impact on the classification. Of the four sequences tested, the only one that did not go through the confused state was the one classified as Mistaken. Because the Right model was meant to account for a few mistakes, it would most likely generate sequences from all four states of setup, correct calculations, confused and mistakes. It is possible that if a state sequence did not exhibit the confused state, it was considered less likely as belonging to the Right model. Another possibility is that the arbitrary sequence length limit of twenty was unsupported, and that a student who made many mistakes compared to the other Right students should not be considered as Right.

The other three sequences showed a pattern of state transitions from setup to mistakes to confused to correct calculations. It seems plausible that for a student who could be considered as Right or a quick learner, he might have made an initial error, been confused when the formula did not modify the image as expected, then realized the problem and fixed it. It is difficult to determine the accuracy of this hypothesis through unobtrusive educational assessment. It may be necessary to have direct student feedback about their thought process to verify the validility of this suggestion. It is also important to note that this is only one possible learning pattern for a Right student, as evidenced by the other student being classified as Mistaken although on manual inspection he was classified as Right. The student might have been incorrectly classified manually or multiple models might have been needed to identify each expertise level because of the different patterns students exhibited.

Conclusions and Future Work

Log files are rich with information, but extracting meaningful knowledge from them is a difficult task. HMMs have the potential to be a valuable tool in endeavor, although they also have their limitations. As shown in Experiment 1, HMMs can be used for classification if the models are trained with appropriate data. However, if a sequence does not belong to any of the models, it may be misclassified, like the “Color Saturation Formula” sequence was.

Experiment 2 emphasizes the challenges of using HMMs to analyze log files. Since the states are hidden, it is not only difficult to estimate what state likely generates a symbol or causes the event, it is also uncertain what the states are. While the actual number of states and their significance is not as important to Problem 1 [3], solving Problem 2 requires careful consideration to what the states represent. With the variety of learning behaviors and thinking patterns of students, it may not be practical to guess a student’s state of mind using HMMs.

Although the Confused’ model in Experiment 2 did not yield any definite information about student expertise level or state of mind, it did illustrate the some models may be too dependent on the training data, resulting in overtraining the model so it was specific only to that particular sequence. One possible way to mitigate this problem is to use training sets instead of just a single sequence. The hardest challenge with using either single or multiple sequences is selecting appropriate training data. In particular with subjective classifications such as expertise level for a confused student, it may be difficult to find more than one sequence that reflects a certain trend. Data that is too varied or different from one another will likely result in models close to equal probability distributions like the Confused’ model. If there is more than one behavior or pattern typical of a classification like expertise level, then multiple models representing the same classification should be developed.

The Default model with equal A, B and π distributions may be used as a comparison between trained and untrained models. If a sequence matches best with the Default model, as occurred with the sequences in Experiment 2 matching best with the Confused’ model, this may be an indication that no appropriate model has been created. Preliminary experiments of testing the activity classifications in Experiment 1 are promising. In Experiment 1, the “Color Saturation Formula” sequence matched with a high probability (logarithmic probability of -187) to the “Making Purple” model. A brief experiment testing the same sequence using the default model returned a slightly higher logarithmic probability of -184. Further investigation is needed to determine if the Default model is an accurate indicator that a sequence does not fit any of the available models.

Another modification that may improve results is the elimination of long zoom in and long zoom out. The use of these two symbols was meant to shorten the observation sequences and compact the information, but one advantage of using HMMs is its flexibility in dealing with sequences of variable lengths, including long sequences. While shorter observation lengths do tend to increase the HMM probabilities, often only relative probabilities are of interest, as with classification and comparison between different models, so long sequences are not a problem. The other reason for using these symbols is to make observation sequences more uniform by not relying on the number of individual zoom in and zoom out's, but this may not be desirable. If the observation sequences reflected the actual number of zooms, some pattern may emerge that this condensation hides. In addition, removing these two symbols reduces the size of the symbol set, making the models simpler, which may improve accuracy. The experiments should be re-run without the long zooms to test if any differences occur.

Taking into account the times of the events is another approach. The log files record the time, a factor which was mainly ignored for the experiments, other than using the presence of a 200 second interval between events to segment activities for "A Two-Tile Image" in Experiment 2. The times were not used in the HMMs, although the HMMs can be modified to account for time duration [2]. The use of time to segment activities showed that time is an important indication of learning. When a long interval followed an ending calculation like in "A Two-Tile Image," the student likely completed one activity and was reading the next part of the worksheet. If a long interval occurred during the middle of an activity, it may be an indication that the student was confused and needed help. If the analysis of log files could be done in real-time, then by identifying confused students, instructors can intervene and provide guidance.

Future work should also concentrate on integrating HMMs into INFACT so instructors can use this tool in an easy, efficient manner. Perhaps by using the Gnome environment developed by Nick Benson [1], HMM analysis could possibly be done in real-time. Instructors could then be immediately notified of students suffering from common misconceptions as indicated by their log files. There are certainly difficulties with this integration, including designing a user-friendly interface for educators without a significant background in programming to use, but the potential of HMMs in improving educational assessment makes this is a worthwhile goal.

Acknowledgements
Professor Steve Tanimoto has been a guiding force behind this project, providing me with endless encouragement and suggestions. His knowledge of INFACT and perspective as an instructor has helped me understand better how educators benefit from educational assessment and how may be HMMs can be to aid them in this endeavor. I would also like to extend my gratitude to the Winter 2003 CSE 590d group for their insights and advice, especially Adam Carlson and Nick Benson. In addition, I owe many thanks to Ilya Maykov for helping me write and troubleshoot the Java code.

References

1. Benson, Nick. (2004). The INFACT Gnome Environment: A Platform for Autonomous Agents Generating Automatic Student Assessments and Feedback.

2. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE,* 77 (2), 257-286.

3. Soller, A.,Wiebe, J. and Lesgold, A. (2002). A Machine Learning Approach to Assessing Knowledge Sharing During Collaborative Learning Activities. *Proc. CSCL 2002*, 128-137.

4. University of Washington Course Catalog, http://www.washington.edu/students/crscat/gis.html, (June 2004)

APPENDICES

APPENDIX A: ACTIVITY WORKSHEETS

Additive Color Mixing

Purposes: to explore the mixing of colors in the RGB color system, and to use the web-based communication system INFACT-FORUM.

Instructions, Part I:

Log into INFACT-FORUM as follows. Start a web browser and visit our seminar home page: http://ole.cs.washington.edu/pixels/

Click on the link to INFACT-FORUM.

Enter your username and password for INFACT-FORUM. Each of you has been assigned to a group within the INFACT-FORUM system. INFACT-FORUM lets you post messages to your group.

Experiment… Start up PixelMath, then go to the File menu on the calculator and select New Image. Use the default values and click OK. Next zoom into the image by clicking on it (assuming you have the magnifying glass selected). Click until you can see the individual pixels. Now choose the selection tool (with the rectangle icon). Click and drag on the image to select some pixels. Enter new values for the Red component of the pixels using the calculator. Chapter 4 of the notes explains how to do this by using the enable-red, enable-green, and enable-blue toggles on the calculator.

1. Try to create the color purple by entering appropriate values for the Red, Green, and Blue components of the pixels. What values did you use for red, green, and blue? Report your answer in INFACT-FORUM by responding to the message about "Making Purple."

2. Now try to make yellow. Explain your approach by answering the question about "Making Yellow" in INFACT-FORUM.

3. Next try to make gray, pink, and brown, in that order. Report your solutions in INFACT-FORUM by responding to the question about "Gray, Pink, and Brown."

Instructions, Part II:

In this part of the activity, your assignment is to create a kind of "rainbow" using PixelMath. The idea is to explore creating a variety of colors, and if you like, to make an artistic or fanciful design at the same time. After you have created your rainbow, use INFACT-FORUM to give a couple of sentences describing your approach. Do this by responding to the question about "Painting Rainbows."

General Interdisciplinary Studies 130
Winter 2004

"PIXELS, NUMBERS AND PROGRAMS"

Brightness, Contrast, Quantization and Thresholding

Overview: Each pixel in an image uses numbers to represent a brightness value (in a monochrome image) or a color. Performing arithmetic on these numbers can be used to adjust the brightness or contrast of an image. The appearance of pixels can also be changed for the purpose of using fewer bits (and thus saving memory) or to obtain graphic arts effects.

Brightness: Try brightening the Mona Lisa image by adding 50 to each color component of each pixel. How does she look after this change? Undo that change and try multiplying each pixel component by 1.25 instead. Is there any difference in the appearance of these two results? Post your answers in INFACT in response to the post by Steve with subject "Brightness."

Contrast: The contrast in an image refers to how different the dark colors are from the light colors. An image with high contrast is one that contains a significant number of pixels that are very dark as well as many that are very light. We can usually increase the contrast in an image by multiplying pixel values by a number greater than 1. However, this generally has the effect of brightening most non-zero pixels, too. To compensate for this general brightening, some number can then be subtracted from each value.

In general contrast is increased by applying a formula such as

2 * Source1(x,y) – 255

This has the general form

m * Source(x,y) + b

This is a linear function in which the "independent variable" or the input to the function, is Source(x,y). The "slope" m represents the "gain" or factor by which the contrast is increased, and the "intercept" b represents a brightness adjustment term. In the example above, b is –255.

Try adjusting the slope and intercept of such a formula so that when applied to the Mona Lisa image, we get an image that looks like it is being viewed by candlelight in a dark room. What was your strategy? What formula did you end up with? (Post as a reply to the "Contrast" message.)

Quantization: The manner in which the numeric pixel values actually stand for particular colors or brightness values is often known as quantization. Quantization methods must find a good compromise between fidelity (the accurate representation of the brightness or color), and economy (usually inversely proportional to the number of bits used for the numbers). If each pixel were to be represented with only a single bit, then 0 could represent black and 1 could represent white. Of course, we could also make up the rule that 0 represents red and 1 represents green, and every pixel must either be red or green.

PixelMath normally allocates 8 bits to each color component of each pixel. That permits $2^8 = 256$ different values for each component. To test the effects of reduced quantization, we can force certain bits of an image to have value 0, thus eliminating their influence. If we are working with a normal image with brightness values well distributed in the range 0 to 255, we would normally reduce quantization by setting one or more bits of each component to 0 starting with the "least significant bit." This is the rightmost binary digit of the binary representation of the component. For example 75 is written in binary as 1001011 meaning $2^6 + 2^3 + 2^1 + 2^0$. The rightmost bit is the one that represents 2^0. The following PixelMath formula has the effect of setting the rightmost bit of each pixel component to 0

2 * Floor(Source1(x,y)/2)

The effect of this change is subtle. More obvious is the effect of setting, say four of the eight bits to zero:

16 * Floor(Source1(x,y)/16)

The most drastic quantization of this type is to leave only the most significant bit to represent the pixel. This following formula accomplishes this:

128 * Floor(Source1(x,y)/128).

Try some of these formulas on two or three images from the PixelMath web site, including the Mona Lisa. What effect do you get? Can you get an artistic effect that looks appealing? (Post a brief answer as a reply to "Quantization" in INFACT.)

The above method for quantization doesn't offer any flexibility on the ranges of values that correspond to the various colors that we end up with. Also, it doesn't give us much control of what the colors are that are used to represent each range. The following method for reducing quantization lets us set the ranges.

If s1(x,y) < 33 then 0 else if s1(x,y) < 150 then 63 else if s1(x,y) < 165 then 128 else 192

Note that s1(x,y) is just an abbreviation for Source1(x,y). Try this formula on the Mona Lisa.

Thresholding: The preceding example used three test values: 33, 150, and 165. Applying this kind of simplification to image, when there is just one test value is called thresholding and the test value is called the threshold. Here's a formula that applies thresholding:

If 0.33 * (red1(x,y)+green1(x,y)+blue1(x,y)) < 92 then 0 else 255

The threshold is used to separate the pixels values that will be set to the minimum from those that will be set to the maximum.

Because the results of this operation are the same for each of the red, green, and blue components of each pixel, we get an output image containing only black pixels and white pixels. If, on the other hand, we were to use Source1(x, y) < 92 as the condition, then there would be 8 possible distinct colors in the result.

Try thresholding the Mona Lisa image. Adjust the threshold until the result is as close as possible to the unthresholded image. What is your threshold value? How did you choose it? (Reply to "Thresholding" in INFACT.)

General Interdisciplinary Studies 130
Winter, 2004
"PIXELS, NUMBERS AND PROGRAMS"

Making Big "Tiled" Images

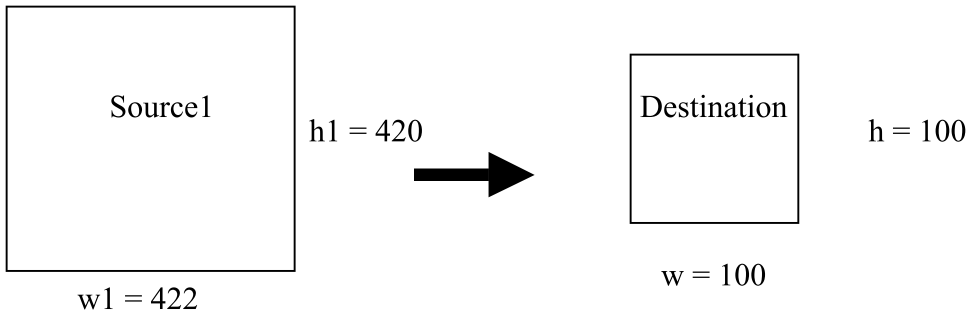
Overview: When creating multiple images, they can either be put in separate windows or in one large window. This activity follows the latter approach. There are three key ideas in making this work: (1) creating a destination buffer big enough for all the new images, (2) using one or more IF-THEN-ELSE operators in the PixelMath formula where the condition tests what tile of the large image a given pixel is in, and (3) applying a spatial shift (known as translation) to make the various images show up in the right parts of the large image.

Start by Scaling: Let's start this activity with a practical step that will speed up the rest of the activity: creating an image with a conveniently small size. Since the Mona Lisa image (having size 422 by 420) is so large, let's make a 100 by 100 version of it.

First load a copy of the original Mona Lisa.

Next, create a new blank (white) image in a new window and give it a size of 100 by 100.

Set the Source1 window to the original and the Destination to the new image.



Now let's figure out what formula to use to make the reduction. The x values (i.e., the column indices) for the source image run from 0 to xmax1, whereas in the destination, they run from 0 to xmax. Now, let $x_{destination}$ represent the x coordinate of some destination pixel, and let x_{source} represent the x coordinate of the corresponding pixel in the source image. For example, if $x_{destination} = 99$, then $x_{source} = 421$. Any destination pixel in column $x_{destination}$ should have its values taken from a pixel in what column from the source image? $x_{source} =$ (___ / ___) * $x_{destination}$
You fill in the blanks. Also, post your ratio in INFACT in response to the message with subject "Ratio for reduction".

What is the corresponding formula for the y values?

Now make up the PixelMath formula. Hint: although we have x_{source} in the above equation, we will not mention this variable anywhere in the PixelMath formula. Instead, we will substitute the corresponding expression (right hand side of the equation above) in the x position in the Source1 function call. So just plug in the expressions for x_{source} and y_{source} into Source1(x_{source} , y_{source}).

Note: because PixelMath uses the pull method for computing pixels, we represent the variable $x_{\text{destination}}$ in PixelMath simply by x.

Try out your formula and if it does not produce the scaled-down version of the Mona Lisa, study the problem closely and fix your formula so it works.

One more original: Apply the same technique to another image, such as the Nisqually Glacier picture of Mount Rainier. After you've done this, you'll have two different images, each of size 100 by 100 pixels.

A Two-Tile Image: To start our actual tiling, let's create a new image in a new window that is large enough to hold both of the small images, side by side. This means it should be 200 pixels wide and 100 pixels high.

After you've created the blank image, make this image be the destination. Make the small Mona be Source1 and make your other small image be Source2. Use the following formula to create the tiled image.

```
If x < w/2 Then Source1(x,y) Else Source2(x – w/2, y)
```

There are two occurrences of w/2 in this formula. What's the purpose of the first occurrence? (Answer this in INFACT by replying to "Why the first w/2 in the Two-Tile Image formula?")

Now, what's the reason for the second occurrence? Respond to the message with subject "Why the second w/2 in the Two-Tile Image Formula?"

A Filmstrip: Now here's your next challenge. Extend the above technique of tiling images so that you create one wide image with four different versions of the Mona Lisa in it. The first should be the original small image of Mona. The second should be Mona with 64 added to all pixel values, modulo 256. If we were computing this version by itself, we would use the formula (Source1(x,y) + 64) Mod 256. The third should be adding 128 instead of 64, and the fourth should be adding 192. By the way, what would happen if we added 256 this way? (Answer not in INFACT but here: ___________________________.)

A Big Square: Now try creating the same four versions of the little Mona Lisa, but instead of arranging them in a short, wide image, arrange them in a 200 by 200 pixel image. You'll have to use IF-THEN-ELSE to test the y values in addition to the IF-THEN-ELSEs used to test the x values. Post the formula you come up with in response to the message with subject "The 200 by 200 square".

A Bigger Square: OK, you're ready to try a bigger challenge and a bigger tiled image. Try making one that's 400 by 400. Come up with your own variations of the Mona Lisa so that each of the 16 tiles is different from the others in some way. Since there are so many cases, see if you can set up the formula to handle them automatically. If you succeed in doing this, post answers on INFACT (as a reply to "A Bigger Square") to the questions: What strategy did you use for this? What did your image look like?

APPENDIX B: FILES

Pre-processing
To use the data filters on multiple log files, all the files were put into a folder, then `runDataFilter.pl` was run with the arguments of the filter followed by the folder name. All the observation sequences were stored in an "obs" subfolder with "obs_" prefixed to the original file name.

Experiment 1 used `exp1dataFilter1.pl` and `exp1dataFilter2.pl` for the initial experiments to convert the log files into observation sequences. The final filter used in the pre-processing, with M = 11, was `exp1dataFilter3.pl`.

`truncateData.pl` was used to segment the log file for "A Two-Tile Image" from the rest of the activities on the worksheet.

The log files for Winter 2003 were slightly different from Winter 2004 because Winter 2004 did not include the "Unsucessful Parse" event. Therefore, for Experiment 2, two data filters were used: `dataFilterTwoTileImageWin03.pl` and `dataFilterTwoTileImageWin04.pl`.

The Models
Initial HMM models had equal probability A and π distributions, and a biased B representing an estimate. The B distributions were created using `BDataBuilder.java`. It takes as an argument a text file. The text file's first line is the number of states and number of models, separated by a space. The following lines give the preference for specific symbol distributions for each state. The general syntax is: state number, symbol number, and the likelihood, which is either not likely (N), somewhat likely (S) and very likely (V). The following line can omit the state and will assume that the symbol and likelihood listed is for the state given in the previous line. For any symbol of any state that is not explicitly included in the file, the default likelihood is not likely. This is a compact way to create large simple bias matrices reflecting which symbol is most likely to occur for each state. In summary, the format is:

```
n (int) m (int)
(state) (symbol) (likelihood) |
(symbol) (likelihood)
```

Running the tests
The main program is `HMM.java`. Once a model is created, it can be trained by using the Baum-Welch algorithm. The Forward-Backward algorithm can be called to solve Problem 1. It uses scaling as described in [2], so it returns the probability as logarithmic values. The Viterbi algorithm can be called to solve Problem 2 and also uses scaling. Refer to the code for more details.

APPENDIX C: DETAILS OF EXPERIMENT 1: CLASSIFICATION

There were many other experiments conducted in the attempt to classify the three activities: “Making Purple,” “Brightness,” and “A Two-Tiled Image.” One of them is listed below. Again, all models were trained using only one observation sequence.

The first try used two-state HMMs:

(1) State 1 represented “trying to affect the color”
(2) State 2 represented “trying to manipulate the geometry.”

There were six symbols for the actions:

(1) Zoom In
(2) Zoom Out
(3) Selection, which is when a user clicks and drags the mouse to select a region of the window
(4) Undo
(5) Domain Transformation Calculation, which is when a user modifies the (x,y) value in the source expression (e.g. `Formula=Source1(x/2,y)`), used for geometric manipulations
(6) Other Calculation, including range transformation formulas that modify the result of the source expression (e.g. `Formula=Source1(x,y)+100`), used for color manipulations.

Fourteen sequences were tested by running the Forward-Backward algorithm: two “Making Purple,” two “A Two-Tiled Image,” and ten “Brightness.” The “Making Purple” and “A Two-Tiled Image” sequences matched the best with their respective models, but “Brightness” yielded mixed results. Four matched to the “Making Purple” model, four to the “A Two-Tiled Image” model, and two correctly classified as “Brightness.” Figure C1 shows the results of some of these tests. Also included in Figure C1 are the results for the same sequences compared to three-state HMMs (the additional state is for “analyzing,” which was thought to be associated with zooming and selection actions). The two-state and three state HMMs resulted in the same classifications, but many of the classifications were incorrect. This led to increasing the symbol set, especially to include the setup state and associated actions.

```
----------------------------------------------------
Testing for Making purple.
----------------------------------------------------
Observation sequence:
0 0 0 0 0 0 0 0 1 1 1 2 5 5 5 2 2 2 5 5 5 5 5 2 5 2 5 2 2 5 5 5 2 2 2 5 5 5 5 5 2 5 2
2 5 5 2 2 5 2 5 2 2 5 5 2 5 2 2 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 2 5 2 2 5 5 5 5 5 2
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5

Three state models
Making Purple: -31.247310950840614
Brightness: -802.7130913469595
Two-Tile Image: -788.9520690573758
******************************************

Two state models
Making Purple: -31.919723099018835
Brightness: -772.2251370376788
Two-Tile Image: -786.7894761115949
```

```
*****************************************

---------------------------------------------------
Testing for Two-tiled Image.
---------------------------------------------------
Observation sequence:
5 3 4 3 4 3 4 3 4 0 0 3 4 3 4 3 4 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 4 5 3 5 3 5 3 5 3
5 5 5 5 3 5 3 5 5 5 5 3 5 3 5 3 5 5 5 5 0 3 5

Three state models
Making Purple: -517.3156862406246
Brightness: -463.9805409140954
Two-Tile Image: -48.12988935407444
*****************************************

Two state models
Making Purple: -514.8287467408221
Brightness: -460.2107008670781
Two-Tile Image: -49.932750372645216
*****************************************

---------------------------------------------------
Testing for Brightness.
---------------------------------------------------
Observation sequence:
5 3 5 3 5 3 5 5 3 0 5 1 3 3 3 3 3 2 2 5 3 5 3 5 3 5 3 5 3 5 3 2 2 5 3 3 3 5 5 5 3 5 3
5 3 5 3 5 2 3 5 2 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5

Three state models
Making Purple: -98.8179871241183
Brightness: -182.0774932481273
Two-Tile Image: -208.17359808585266
*****************************************

Two state models
Making Purple: -94.08991175989718
Brightness: -182.7012755350575
Two-Tile Image: -218.9920964775508
*****************************************

---------------------------------------------------
Testing for Brightness2.
---------------------------------------------------
Observation sequence:
0 1 5 5 5 5 5 5 5 5 5 5 5 3 3 3 5 3 5 3 5 3 5 3 5 3 5 5 3 5 5 3 5 5 5 3 5 3 5 3 5 3 5
3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 4 4

Three state models
Making Purple: -112.38060830421217
Brightness: -109.31744623546683
Two-Tile Image: -67.23140218692144
*****************************************

Two state models
Making Purple: -106.49923038608101
Brightness: -75.79799375731056
Two-Tile Image: -60.053541295142914
*****************************************
```

```
----------------------------------------------------
Testing for Brightness3.
----------------------------------------------------
Observation sequence:
5 5 5 0 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 5 3 3 3 5 3 5 3 5 5 5 3 3 3 3 3 5 3 3 3 5 3
5 3 5 3 5 3 5 3 5 5 3 5 3 5 3 5 3 5 3 5 5 3 5 3

Three state models
Making Purple: -95.96675720838579
Brightness: -21.24438084752857
Two-Tile Image: -47.485848679799815
******************************************

Two state models
Making Purple: -86.61397096993122
Brightness: -28.992456920411296
Two-Tile Image: -62.046681403341594
******************************************

----------------------------------------------------
Testing for Brightness4.
----------------------------------------------------
Observation sequence:
5 3 5 3 5 3 5 3 5 5 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 5 5
3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 5 5 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5 3 5
3 5 3 5 3 5 3 5 3 5 3 5 3

Three state models
Making Purple: -126.75265504578314
Brightness: -9.68940315038547
Two-Tile Image: -55.18424300694613
******************************************

Two state models
Making Purple: -111.13619856340391
Brightness: -23.02630116391803
Two-Tile Image: -88.73507940536976
******************************************
```

Figure C1: An excerpt from the results for Experiment 1 using N=2 and N=3 HMMs.

Appendix D: Details of Experiment 2: A Two-Tile Image

The symbol set used was:
M = 15:

- 0 (zoomIn – 1 to 3 instances),
- 1 (zoomOut – 1 to 3 instances),
- 2 (long zoom in)
- 3 (long zoom out)
- 4 (selection),
- 5 (undo),
- 6 (new window)
- 7 (open picture)
- 8 (checkbox)
- 9 (size)
- 10 (expert calculation: uses variables, e.g. x*(w1/w))
- 11 (right calculation: uses numbers, e.g. x*(422/100))
- 12 (ending calculation: if x < w/2…)
- 13 (push)
- 14 (syntax errors, parsing errors)
- 15 (wrong calculations, other calculations)