1. Overview

The goal of this project was to build a Hidden Markov Model (HMM) text extraction system and to explore the effects of context width, presence of structure in the form of HTML tags, and shrinkage mixture settings on the text extraction results. Overall the HMM approach to text extraction seems justified in that it offers the flexibility of a probabilistic model while providing many of the same benefits of rule-based systems through the use of model structure. The implementation given here is based on (Freitag & McCallum, 1999) but differs in a few aspects which will be discussed below.

2. Software Architecture

While most of the algorithm details will be postponed to the section on algorithm description, a brief description of the overall system may help the reader understand how all of the components of the project fit together with respect to the goal of text extraction.

The basic goal of this system is to take a set of documents in HTML format and learn a generative probabilistic model of the underlying state transition structure of the document from a set of tagged training data. This model is not just one HMM however, it is a mixture of HMM's organized in a hierarchical structure to help the system cope with data sparseness. Given a trained probabilistic mixture model of the data, the system should then be able to apply this model to new unseen HTML documents to predict which portions of these documents are likely targets according to the training data template.

A system such as this requires an architecture similar to that displayed in figure 1 consisting of the following five basic blocks:

1. *Input Subsystem* – This sub-system is responsible for reading all of the data and converting it to a tokenized sequence that can be used to form the internal representation of the document structure.
2. *Document Interface* – This component interfaces with all of the other sub-systems to input the data,
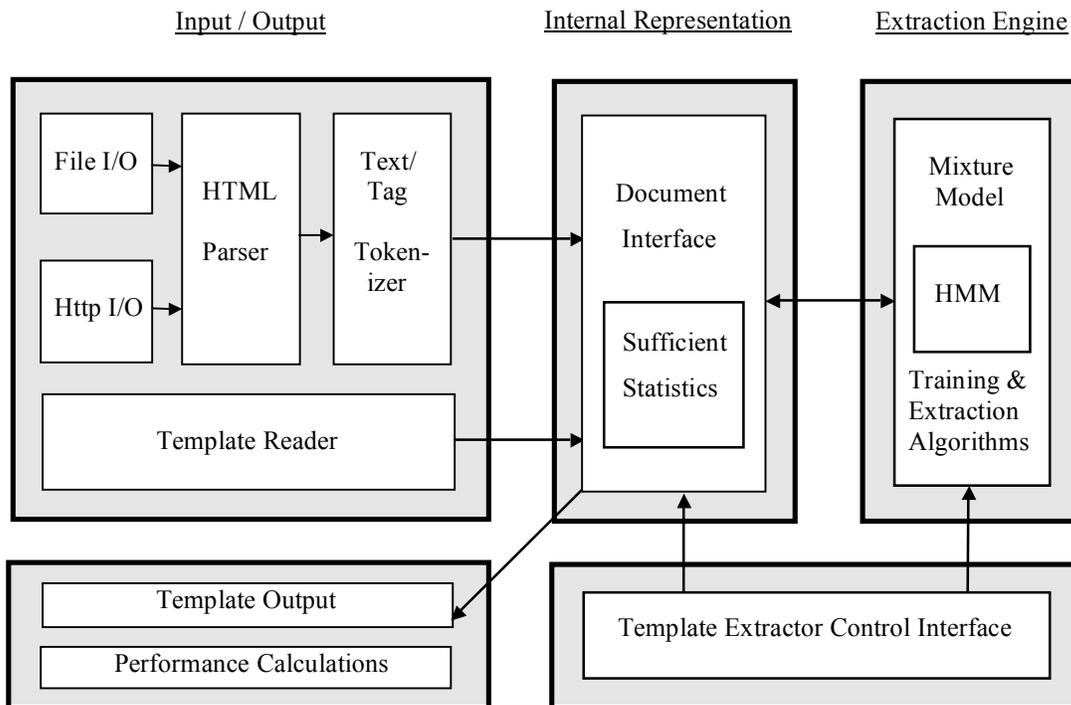
Figure 1. Diagram of the software architecture for the HMM-based template extraction system.

cache the sufficient statistics required by the HMM, train the HMM mixture model and obtain a predicted state sequence for the test data, and output the extracted templates and performance statistics.

3. *Mixture Model* – This system contains a mixture of HMM's according to the structure defined in the training template which is provided in an external file. It uses this structure to model the underlying state and observation sequence of a set of training documents and to predict state sequences for new documents using the Viterbi algorithm (to find the most likely path through the HMM). The mixture model is responsible for making access to the underlying HMM structure as transparent as possible so that all algorithms "see" a single HMM.

4. *Control Interface* – This system simply interacts with each of the other components to oversee the sequence of operations from reading the data and training the mixture model to extracting text and formatting it for output purposes.

5. *Output Subsystem* – This is by far the simplest component. It need only translate the predicted document structure to a coherent output format and calculate performance statistics if the test data is marked up and can therefore be used to test the system's predictions.

While at this point the overall algorithms used by the system may not be clear, the basic structure and interaction of the system components should be somewhat well understood.

3. Algorithm Description

With the basic software architecture in place, it is now time to cover the basic algorithms used by the HMM-based text extraction system.

3. 1 State and Observation Model Structure

Shown below in figure 2 is the basic format by which a document is modeled with a state and observation sequence. (The example uses an excerpt from a Computer Science course description web page that composes the main set of test data that will be used in the section on testing and analysis.)

There are two fundamental issues to note in this model:

1. The observation sequence is always observed but the state sequence is only known for the training data. In this data, the *start* and *end* states are trivially known, the *target* state is marked up as described in the next section and is therefore known, the *prefix* and *suffix* states can be easily determined by knowledge of the *target* state and the context width defined in the training template, and all other states are considered to be *background* states. Knowledge of both the observation and hidden states makes training the model a simple case of maximum likelihood probability estimation.

2. The goal of the template extractor will be to generate the state sequence for an untagged document
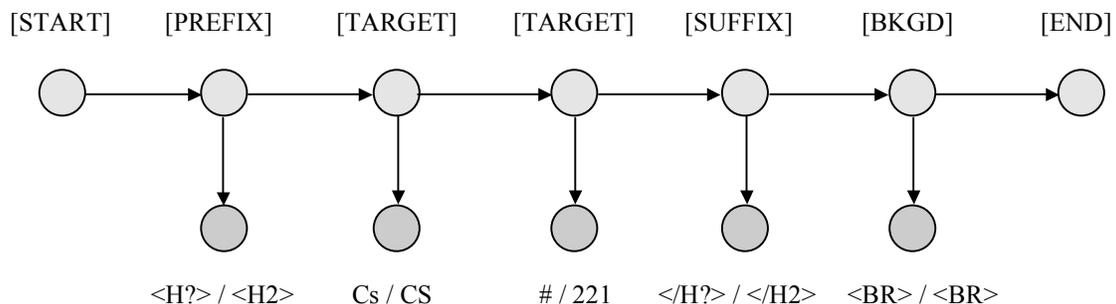
Figure 2. Sample Hidden Markov Model showing parallel state and observation sequence (LHS shows the token, RHS shows the actual text). Note that tokens such as numbers and some HTML tags (i.e. anchor, heading, and table tags) are generalized to avoid capturing irrelevant information in the observation sequence.

using the previously trained model. From this sequence it should be fairly obvious that the text to be extracted is the text that is labeled as a target state by the extraction algorithm.

3.2 Marking Up the Training Data

Given that training the model is fairly straightforward, there is only one slight issue which involves developing a markup method for the training data that will not interfere with the actual data itself. Since all comments in a document are discarded for training (i.e. one cannot really expect any regularities among the comments in a document), it makes sense to embed the training labels as HTML comments themselves. Consequently the template which is used to define the mixture model and target text contains a reference to the comment tag which identifies the target in the training file. For example, a template whose purpose was to extract the course title from a web page would indicate the tag *TITLE* so that anything preceded by *<!--T-TARGET-->* in the document would be considered a target. An example excerpt from actual training data is shown below which includes markers for the course title and instructor.

Sample marked-up web page, note the `<!--T-TITLE-->` and `<!--T-INST-->` tags:

```
<HEAD>
<TITLE>
<!--T-TITLE-->CS 367 - Lecture 2
</TITLE>
</HEAD>
<BODY>

<H1>
<!--T-TITLE-->CS 367-2 Introduction to Data Structures <br>Fall 1996
</H1>
<P>
Course email address:
<A HREF="http://www.cs.wisc.edu/m?cs367-2"> cs367-2@cs.wisc.edu
</A><br>
Course home page:
<A HREF="http://www.cs.wisc.edu/~cs367-2/cs367.html">
http://www.cs.wisc.edu/~cs367-2/cs367.html</A> <br>

<P>
INSTRUCTOR:  <a name="yannis">
<b><!--T-INST-->Yannis Ioannidis</b> <br>
</a>
```

Note in the above example that any text following a tag is considered part of a target until a new-line or something non-text (i.e. a tag or comment) is encountered.

3.3 Parsing and Tokenizing the Document

With the markup method defined for identifying targets in an HTML document, the next step is to actually parse the HTML document and tokenize sub-parts of the HTML document such as tags and text.

For parsing, an HTML parser under the GNU general public license was used to separate the document into a set of tag, text, and comment tokens. While this is a first step in the parsing process, it is not enough to pass this structure directly on to the training algorithm.

For example, anchor tags (i.e. <a href="…">) often precede important information (e.g. a map to a room location or an instructor's home page) and are therefore useful in predicting the likelihood that a document token is a target. However, an anchor tag in its complete form differs from document to document because

the *href* or some other attribute value is almost always going to change based on the context. Consequently, it is important for a tokenizer to discard irrelevant information in tags and include only the information that is likely to occur in other documents, e.g. the fact that it is an anchor, header, or table element tag.

Another reason for tokenization beyond the HTML level includes the terminology that is commonly found in context specific language, namely the use of acronyms and abbreviations. For example, in CS course web pages, a course is almost always titled in the format 'cs###'. Since we want to be able to generalize from this to other course titles it is not a good idea to store 'cs221' directly as the observation sequence 'cs221'. Rather it is a good idea to separate letters from collocated numbers, capitalize all letters (not always a good idea, but often useful), and tokenize the number as a generic number rather than it's actual value. Consequently, 'cs221' would thus be tokenized as two tokens, 'CS' and '#' (or some other indicator for a number). A tokenization such as this increases the chances that future course can be generalized to, e.g. a system that tokenizes in this manner should be able recognize 'Cs 223b' as a course as well.

Consequently tokenizing text beyond the HTML level is quite useful for training and generalization and is the reason for the additional *text and tag tokenizer* intervening between the *HTML parser* and the *document representation* in the software architecture.

3.4 Modeling the Document Structure

In a generic Hidden Markov Model, one would simply define a number of states and a number of transitions between those states. The more complex the HMM, the better it can represent a document, but also the more data that is needed to dependably train the model and avoid errors due to noise in the data. Consequently there is an apparent tradeoff between representational efficacy and training efficiency and this tradeoff varies from domain to domain.

Therefore, rather than decide on just one model, it is often easier to use a mixture of models and decide later (perhaps empirically) on how much to weight each model. This approach is quite effective because it allows one to model a document at varying degrees of granularity (effectively using a hierarchical model), yet retain the advantages of each model. That is, if the data is sparse, the simpler model will likely perform better and thus be weighted more during the extraction phase. And if there is an abundance of data, the more complex model can be robustly trained and be weighted more heavily during the extraction phase on account of its superior prediction accuracy.

For this text extraction system, a basic set of three mixture models were used and are shown in figure 3 on the following page. These models are somewhat simpler than those used by Freitag and McCallum (which make use of fairly complex prefix and suffix structures) but empirically perform well on some classes of text extraction. Where these simpler models break down however is where the text to be extracted is heavily context dependent such that a reversal of two context words could reverse the classification of an excerpt of text as a target. The models used here are obviously incapable of modeling these cases since all prefixes and suffixes are treated in an order-independent manner.

Despite the simplicity of these models however, they yield the one main benefit that they can be trained on very sparse data. (This was an issue with this project where a limited amount of time that could be allotted to marking up data sets by hand.)

Uniform Model   Context Model   Full Transition Model
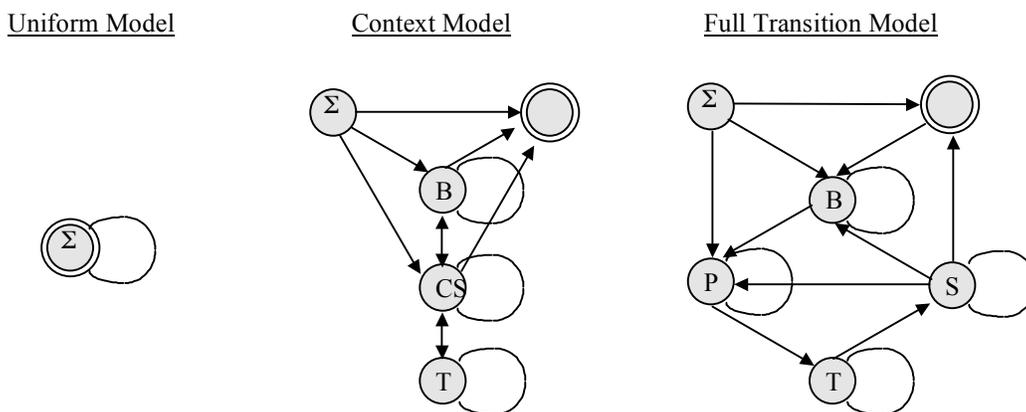
Figure 3. Diagram of the three Hidden Markov Models composing the mixture model. Σ is the start state, the double circle signifies the end state, and the letters B, C, P, S, and T stand for the background, context, prefix, start, and target states, respectively.

3.5 Training the Model

As discussed previously, training the model is a straightforward process of maximum likelihood parameter estimation. From the document training set we can easily obtain the sufficient statistics concerning the frequency that a given state or observation occurred and the frequency with which a state transition or observation emission was made.

Thus, state transition probability can be trained as follows:

$$P(S_{t+1}|S_t) = \frac{Freq(S_{t+1} \& S_t)}{Freq(S_t)} \qquad \text{(Eq. 1)}$$

And likewise, the observation emission probability can be trained as follows:

$$P(O_t|S_t) = \frac{Freq(O_t \& S_t)}{Freq(S_t)} \qquad \text{(Eq. 2)}$$

There might be some slight confusion as to how all three models are trained off of the same set of sufficient statistics but it is fairly straightforward to show that the sufficient statistics for the full transition model can be converted to the sufficient statistics for any of the simpler models. For instance, since the prefix and suffix states are collapsed to the same context state in the context model, one can simply combine these sufficient statistics and estimate the transition probabilities for the context model accordingly.

This suffices for training the individual models but it does not explain how to combine the models. To do this, we use a concept known as *shrinkage* which states quite simply that each model contributes a weighted estimate to the overall calculation subject to the constraint that the sum of the weights is 1.0. A bit more formally, we can estimate a mixture state transition parameter using the following equation:

$$P(S_{t+1}|S_t) = \lambda_{Uniform} \cdot P_{Uniform}(S_{t+1}|S_t) + \lambda_{Context} \cdot P_{Context}(S_{t+1}|S_t) + \lambda_{Full} \cdot P_{Full}(S_{t+1}|S_t)$$

(Eq. 3)

Subject to the constraint that:

$$\lambda_{Uniform} + \lambda_{Context} + \lambda_{Full} = 1.0 \qquad \text{(Eq. 4)}$$

One can perform an analogous computation to compute the mixture estimation of the observation emission probabilities.

One can empirically learn the λ weights using the EM algorithm however for purposes of testing and time, these weights are simply specified in the template file which contains the model structure and parameters.

3.6 Extracting a State Sequence

Once an HMM has been trained on a set of tagged training data, it can then be applied to data to label the most probable state sequence for a document simply given a set of observations (which is in fact the document itself).

The trained mixture appears to be a single HMM for all intents and purposes. (At least the purpose of the mixture model class is to make transparent the fact that the estimates for the transition and emission probabilities are actually based on multiple sub-models.) Consequently, we can apply the standard HMM algorithm to the mixture model to extract the most probable state sequence given a set of observations. This algorithm is known as the Viterbi algorithm and is covered in many texts, Rabiner (1989) being an excellent reference.

It suffices to say that the Viterbi algorithm is a dynamic programming algorithm requiring time $O(TS^2)$ (where T is the number of time steps and S is the number of states) where at each time step it computes the most probable path for each state given that the most probable path for all previous time steps has been computed.

3.7 Output

While perhaps a trivial feature of the software, it is worth mentioning that once a state sequence has been determined for a document, the final step is to determine which words conform to the target states in the document and output these word sets with the appropriate label defined by the training template (i.e. course title or instructor, etc…). Additionally if the test data is tagged, it is also possible to compare the performance predictions of the system with the actual answers. Precision, recall, and the F1 measure are supported by this software.

4. Testing and Analysis

Before analyzing the results of a number of test cases, it is interesting to look at the learned models in the context of the testing domain in order to gain an intuition for what the system is learning. From this point, we can apply the system to a number of test cases and evaluate its performance in light of this initial experimental investigation.

4.1 Testing Domain

The testing domain chosen for this analysis was a set of Computer Science course web pages provided by Tom Mitchell's WebKb project. One hundred of these pages were marked up with four different targets: course title, course instructor, course time(s), and course location.

These 100 web pages were divided into two independent sets, one set of 80 marked-up web pages to be used for training and one set of 20 web pages to be used for performance testing.

4.2 Learning Evaluation

It is interesting to examine the basic transition structure of the marked up training documents. A printout of the transition frequency of the 80 training documents for the 'course title' template with context width 3 is provided below:

Full Transition Count: row(to-state), column(from-state)

```
        STA    BKG    PRE    SUF    TAR    END
       -----  -----  -----  -----  -----  -----
```

```
START        0        0        0        0        0        0
BKGD         7    52622        0      104        0        0
PREFIX      73       33      296      119        5        0
SUFFIX       0        0       27      314      159        0
TARGET       0        0      159        5      919        0
END          0       80        0        0        0        0
```

From this data it is readily apparent that most of the document states are background states, that the document is likely to start out with a prefix sequence, and that quite often a suffix sequence leads right into a prefix sequence.

The latter two phenomena can be easily explained by two common facts about course web pages. First, the title of the web page is often listed in the initial *<title>* portion of the document. This explains the frequency of initial prefix states. Second, the first portion of a course web page beyond the *<title>* section is an actual header itself which states the title of the course. Consequently, it seems that within the context-width of the suffix of the first *<title>* declaration of the course name, there occur quite frequently the second set of prefix tokens for the subsequent page header.

Translating these frequencies into state transition probabilities, we get the following table for $P(S_{t+1}|S_t)$ for the full transition model:

Full Transition Probabilities: row(to-state), column(from-state)

	STA	BKG	PRE	SUF	TAR	END
START :	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
BKGD :	0.08974	0.99790	0.00000	0.21987	0.00000	0.00000
PREFIX :	0.91026	0.00063	0.65055	0.25159	0.00000	0.00000
SUFFIX :	0.00000	0.00000	0.00000	0.66385	0.14750	0.00000
TARGET :	0.00000	0.00000	0.34945	0.00000	0.85250	0.00000
END :	0.00000	0.00148	0.00000	0.00000	0.00000	0.00000

This is in some sense just a base frequency normalized version of the transition frequencies. Note however that there are many valid transitions which have zero probability. For instance, the probability of transitioning from *suffix* to *end* is zero whereas this could be a perfectly valid transition and is in fact allowed by the model. However, the presence of a zero here would prevent the Viterbi algorithm from ever exploring this path, i.e. there will always be some path greater than one with zero probability (except for the parasitic case where all paths are of zero probability).

However, the fact that the mixture model contains as a component the *uniform* model causes all transitions to have some small probability and this effectively resolves the issue of valid but unobserved state (and observation) transitions in the training data.

To verify that indeed, the mixture model does have non-zero probabilities on account of the *uniform* model, the following mixture model conforming to this test case is given below (where the uniform model has weight $\lambda = 0.001$):

Mixture Transition Probabilities: row(to-state), column(from-state)

	STA	BKG	PRE	SUF	TAR	END
START :	0.00100	0.00100	0.00100	0.00100	0.00100	0.00100
BKGD :	0.09065	0.99790	0.05692	0.16686	0.00100	0.00100

```
PREFIX : 0.91035   0.00163   0.73279   0.53331   0.07691   0.00100
SUFFIX : 0.45522   0.00131   0.40751   0.73944   0.15066   0.00100
TARGET : 0.00100   0.00100   0.26391   0.08919   0.85265   0.00100
END    : 0.00100   0.00248   0.00100   0.00100   0.00100   0.00100
```

Also, for purposes of curiosity, a small subset of the observation emission mixture probabilities are listed below:

Mixture Observation Probabilities: row(observation), column(from-state)

	STA	BKG	PRE	SUF	TAR	END
<TITLE>:	0.00000	0.00006	0.12111	0.03979	0.00000	0.00000
CS :	0.00000	0.00536	0.00054	0.00160	0.08340	0.00000
- :	0.00000	0.03594	0.00000	0.00000	0.03892	0.00000
[NUMBER:	0.00000	0.06356	0.01246	0.01969	0.18905	0.00000
HOME :	0.00000	0.00121	0.00217	0.00213	0.03522	0.00000
PAGE :	0.00000	0.00182	0.00271	0.00373	0.03985	0.00000
</TITLE:	0.00000	0.00002	0.05125	0.12178	0.00000	0.00000
<BODY> :	0.00000	0.00000	0.00217	0.00213	0.00000	0.00000
<H?> :	0.00000	0.01207	0.34603	0.18459	0.00649	0.00000
</H?> :	0.00000	0.00972	0.11028	0.21917	0.00649	0.00000
DATABAS:	0.00000	0.00076	0.00000	0.00000	0.00463	0.00000
SYSTEMS:	0.00000	0.00224	0.00054	0.00160	0.02039	0.00000
AND :	0.00000	0.01516	0.00054	0.00160	0.02039	0.00000
INFORMA:	0.00000	0.00244	0.00054	0.00160	0.00093	0.00000
RETRIEV:	0.00000	0.00017	0.00000	0.00000	0.00093	0.00000
<A> :	0.00000	0.03101	0.00325	0.00532	0.00000	0.00000
DEPARTM:	0.00000	0.00045	0.00000	0.00000	0.00000	0.00000
OF :	0.00000	0.01224	0.00000	0.00000	0.00741	0.00000
COMPUTE:	0.00000	0.00222	0.00054	0.00160	0.02039	0.00000
SCIENCE:	0.00000	0.00083	0.00000	0.00000	0.00741	0.00000
</A> :	0.00000	0.03092	0.01036	0.00482	0.00000	0.00000

Some features of this data are quite interesting. For instance, *<TITLE>* and *<H?>* tags are likely indicators of a prefix symbol. 'CS' has a high probability of being emitted from a target state as do the words 'SYSTEMS' and interestingly, 'AND'. Additionally, likely suffixes are simply the closing tags of the prefixes, *</TITLE>* and *</H?>*. This also indicates that HTML structure can be extremely useful for predicting targets in text extraction.

In any event, most of these transition and observation probabilities conform to our intuitive notions of what the documents actually look like, and reflect the fact that to some extent, what these models are predicting is not drastically different from the abstract procedures we follow when looking at a web page. Based on the structure and our probabilistic intuitions, it seems that we are more or less performing a very similar task. Whether or not this observation is relevant to the task is perhaps unimportant, it just means there is a very intuitive interpretation for what the system is doing.

4.3 Performance Analysis

Now that we have an intuitive feel for what the system is estimating and predicting, it is useful to get a more concrete result on the system's performance.

Consequently, *attached at the end* of this document are a set of graphs conforming to the six experiments discussed below:

1. *Performance of Text Extractor vs. HMM Mixture Weights* – This graph analyzes the F1 Measure performance of the text extractor over various mixture weights for each of the four data sets (course title, instructor, time, and location). The mixture weights vary uniformly from being completely biased toward the context model on the left-hand side to being completely biased toward the full transition model on the right hand side (the uniform model maintains a weight of 0.001 throughout these experiments). Note that extraction of the title yields excellent results while the other four data sets show rather consistent poor performance. This is mainly due to two reasons. First, while Computer Science courses tend to draw from a similar set of universal names, instructor names are institution dependent as are locations. Thus, it is hard to generalize to new names or institutions unless the surrounding context gives enough support for such a prediction. Second, many course web pages list many times: course and recitation hours, office hours, due dates and times, etc… The presence of so many times is extremely confusing to the system and is largely responsible for the poor F1 measure performance. Here the system would benefit from a more elaborate context.

2. *Performance of Text Extractor vs.Training Context Width* – This graph is extremely similar to the previous one except that the X axis displays five different values for the context width. Overall results show a common peak performance around a context width of two or three which seems a quite reasonable result. (i.e. Too little of a context width yields too few predictions and too high of a context width yields ambiguous predictions.)

3. *Performance of Text Extractor vs. HTML Tag Inclusion* – As this graph title implies, it compares the F1 measure performance of the system not using HTML tags as observations to a version using HTML tags as observations. Without much need for explanation, it is apparent that with the exception of some statistical noise in one case, the use of HTML tags generally helps text extraction. This is quite simply due to the fact that HTML structure often provides special indicators of the importance of a piece of text and this important text is quite often a target of the text extractor.

4. *Performance of Text Extractor on Title Data Set vs. HMM Mixture Weights* – This graph is extremely similar to the first graph except that it limits data to just the Title data set and provides the three performance measures: performance, recall, and F1 measure. In this graph, precision and recall are approximately equivalent for biases toward the full transition model. For biases toward the context model, the precision is slightly higher than the recall likely due to the fact that this model is picking up fewer target words due to the less elaborate transition structure while the targets it *is* picking up are highly likely to be correct.

5. *Performance of Text Extractor on Title Data Set vs.Training Context Width* – This graph is analogous to the second graph except that it provides precision, recall, and F1 measure performance for the Title data set as the context width varies. This graph indicates that precision and recall are fairly similar no matter what the context width but that a context width of three yields peak performance for the Title template.

6. *Performance of Text Extractor on Title Data Set vs. HTML Tag Inclusion* – This graph is analogous to third graph except that it provides precision, recall, and F1 measure performance for the Title data set for the case of ignoring HTML tags vs. the case of using them. These graphs show that while there is little difference between precision and recall for each choice, there is an overall drastic absolute difference which indicates that using HTML tags is better under all performance measures compared to a system which discards them.

4.4 Qualitative Analysis

While we have seen an evaluation of what the system has learned and some statistical results for its performance, we have not yet obtained a good qualitative feel for how useful the system is. Consequently,

following is some selected sample output from the system trained on the previous set of 80 training examples and then applied to 20 web pages that are currently on-line. (Note that the WebKb data was posted in 1996, so the training data is effectively five years older than the testing pages.)

Output of the text extraction system for three on-line web pages. Note that the results are mostly correct with a few minor errors and one omission of location for the third class.

```
Extraction results for document:
'http://www.stanford.edu/class/cs223b/'
------------------------------------------------------------------
TITLE: CS 223 B
INST:  Chris Bregler
TIME:  Wed 5 - 9 PM ,
TIME:  Wednesdays 11 : 00 - 12 : 15
LOC:   B 26 B Gates

Extraction results for document:
'http://www.cs.utexas.edu/users/risto/cs394n/'
------------------------------------------------------------------
TITLE: CS 394 N Neural Networks , Fall 2001
TITLE: CS 394 N Neural Networks , Fall 2001
INST:  Risto Miikkulainen
TIME:  Wed 3 : 30 - 6 : 30 pm , PAI 5
TIME:  4 17 : 25 : 28
LOC:   PAI 5 . 60

Extraction results for document:
'http://www.cs.utexas.edu/users/lavender/courses/cs386/index.html'
------------------------------------------------------------------
TITLE: CS 386 L
TITLE: CS 386 L : Graduate Programming Languages
INST:  Dr . Greg Lavender
TIME:  10 - 11 am <br> Tel : 232 - 7891
TIME:  2 - 3 , Th 11 - 12
```

For these web pages, the system performed extremely well, managing to extract names and room locations that were not even mentioned in the training set (i.e. Stanford courses were not in the set of training data). Overall, the text extraction system yielded about 85% accuracy in prediction of the course titles and similar performance to the previous test set for each of the instructor, location, and time features.

As mentioned in the performance analysis section, the one place where the implementation defined here breaks down is where context is the only useful indicator of target-hood since targets such as names and locations are likely to have not been seen before. The context mechanism in these models is weak to some extent and could benefit from more elaboration. As far as the poor prediction of times is concerned, again additional context structure would help since the context should offer a good indicator of why a time is being mentioned.

5. Conclusions

Overall we see that the Hidden Markov Model text extraction method works better when combined with the concept of shrinkage (i.e. a system mixing multiple models iss superior to a system using just one). We also see that a limited amount of context is extremely important to text extraction since quite often the words themselves aren't known to be the targets (e.g. instructor names or class locations). Additionally we see there are some very promising results when using document structure such as HTML markup to help

the system predict targets correctly. In fact, we saw a performance spike of 35% on precision and recall just by allowing the use of HTML tags in HMM prediction.

Where this model requires improvement however is in the HMM structures it uses to model the document since for three of the four datasets, context structure was very important yet somewhat deemphasized in this system implementation. More elaborate context structure would be extremely useful for extracting templates such as course times since the times in general are prevalent on a web page and it is only through contextual disambiguation that the purpose of the time listing can be determined (what if it is for office hours instead of class time?).

Given these results there are two questions which come to mind. First, is maximum-likelihood parameter estimation ideal for learning HMM transition and emission probabilities? Perhaps a system which makes fewer assumptions such as a max-entropy training rule would make more sense. Some initial research has been done in this direction (McCallum, Freitag, & Pereira, 2000) but there is certainly more to explore and experiment with on this topic.

Additionally, it would also be interesting to ask if not only the probabilities but the actual HMM structure could be learned. Again, there has been some promising initial research in this area (Freitag & McCallum, 2000) which attempts in some sense to mutate an initial model in a hill-climbing sense but there is obviously still more work to be explored here.

One research direction which could also yield increased performance would be more aggressive linguistic morphology during the tokenization phase of the system. Although very few linguistic assumptions are made in an HMM text extractor, one approach that did seem to work well over previous versions of the software was the assumption that text should be broken down into its most atomic components. Consequently an even better tokenizer than the one used in the current system would attempt to remove plural endings or reduce verbs to a common derivational form when generating observation token sequences. This would greatly improve the ability of the system to generalize and seems to be an interesting direction for future research.

Consequently, we see that HMM/Shrinkage-based text extraction is a very promising approach with encouraging results so far (even in a simple system such as this one) and there are multiple directions in which future research can lead to further enhance the performance of these systems.

6. Bibliography

Freitag, D., & McCallum, A. 1999. *Information Extraction with HMM's and Shrinkage*. AAAI '99 Workshop on Machine Learning for Information Extraction.

McCallum, A., Freitag, D., & Pereira, F. 2000. *Maximum Entropy Markov Models for Information Extraction and Segmentation.*. Proceedings of the Seventeenth Interational Conference on Machine Learning, ICML-2K. Stanford, CA.

Freitag, D., & McCallum, A. 2000. *Information Extraction with HMM Structures Learned by Stochastic Optimization*. Proceedings of AAAI-2000. Austin, TX.

Rabiner, L. R. 1989. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proceedings of the IEEE, 77 , no. 2, 257-285, February.

7. Code and File Description

The code for this project was written in Java and is provided as a TAR file due to the nested directory structure of the software and data. Use the command 'tar xvf extractor.tar' to setup the directory structure and then follow the directions in the README file in the 'Extractor' directory for compilation and execution instructions.

Following is a brief description of each of the source files (~2,200 lines total *excluding* the HTML parser):

- *Extractor/TemplateExtractor.java* – The main interface for the template extractor. Loads the template and document sets and invokes the training, extraction, and comparison procedures.

- *Extractor/HtmlCollector.java* – A basic interface definition implemented by HtmlWriter and HtmlExporter for collecting the contents of an HTML stream.

- *Extractor/HtmlWriter.java* – A utility class for printing out HTML document contents.

- *Extractor/HtmlExporter.java* – Interfaces with the HTML parser to collect the tokens from an HTML stream.

- *Extractor/HtmlExtractor.java* – Resolves whether a file is located on the web or the local filesystem, loads the appropriate source, and exports the HTML contents to a *DocSet*.

- *Extractor/TextTokenizer.java* – Performs rule-based tokenization of the text portions of HTML documents.

- *Extractor/DocSet.java* – Processes a list of documents or web addresses and loads them into an internal document data structure.

- *Extractor/DocToken.java* – Represents tokens for the hidden state, observation, hash string, actual string, and relevant information for each document element.

- *Extractor/Template.java* – Reads the template files format and represents the target information, structure of multiple HMM's and their associated weights, and other template features such as context width and whether to ignore HTML document markup.

- *Extractor/SuffStatistics.java* – Caches the sufficient statistics in a document set to be used for HMM training.

- *Extractor/MixtureHMM.java* – Builds a set of HMM's according to the parameters defined in the *Template*. Contains the mixture-model version of the Viterbi algorithm for text extraction and contains interfaces for training and accessing each of the sub-HMM's.

- *Extractor/HMM.java* – Contains the HMM structure for the given model and provides algorithms to train the HMM from a provided set of sufficient statistics.

- *com/quiotix/html/parser/*.class* – Class files for a GNU GPL HTML parser used in conjunction with the above files.

Following is a brief description of the training and testing files to be provided on the command line:

- *trainfiles1.txt* – A set of 80 cached CS course web pages from the WebKb project which I have hand-labeled with title, instructor, time, and location markers. Used for training in the above experiments. (Note: *./train/** is the actual location of the marked up data.)

- *testfiles1.txt* – A separate set of 20 cached CS course web pages from the WebKb project which I have hand-labeled with title, instructor, time, and location markers. Used for testing in the above experiments. (Note: *./train/** is the actual location of the marked up data.)

- *webaddr1.txt* – A list of 20 web pages of CS courses currently being offered by Stanford, CMU, U. Texas, and Wisconsin. Although these pages obviously are not marked with the appropriate tags, one can verify from the template output that the correct course title is identified in 17 of the 20 web pages.

Following is a brief description of the template files to be provided on the command line:

- *title1.template* – The optimal HMM template for extracting course titles.
- *inst1.template* – The optimal HMM template for extracting course instructors.
- *time1.template* – The optimal HMM template for extracting course times.
- *loc1.template* – The optimal HMM template for extracting course locations.

Following is a sample template file with an explanation of each line:

```
Context-Width 3              ← Number of words on each side of target to count as context.
Target        TITLE          ← Target tag to search for when parsing HTML files.
Ignore-Tags   FALSE          ← Indicates whether or not to discard all HTML tags.
Add-Model Uniform 0.001      ← Add a Uniform HMM to the mixture model with λ = 0.001.
Add-Model Context 0.250      ← Add a Context HMM to the mixture model with λ = 0.250.
Add-Model All     0.749      ← Add an All (i.e. full transition) HMM to the mixture model
                                with λ = 0.749.
```

8. Graphed Data