Hidden Markov Models for Part of Speech Recognition and Prediction

Denali Rao

Tufts University
Mathematical Modeling
December 2015

Abstract

Hidden Markov models are statistical models that were first described in the 1950s and 1960s. They were first used for mathematical smoothing of curves, and were soon adopted into the biomedical world for use in bioinformatics. Today they are used for gene prediction, sequence alignment, and protein folding.

They are an interesting way to look at the emissions and transitions of a system, so they lend themselves to analysis and modeling of temporal data. This includes data for speech recognition, speech analysis, and handwriting analysis, among others.

Because hidden Markov models are stochastic and almost completely based on probabilities, they are very sensitive to even small changes in their parameters. Depending on the desired application, it can be simple or very difficult to build the hidden Markov model. Even when it is simple, different parameters can be modified to "tune" the model to different scenarios.

This paper will demonstrate how to implement a hidden Markov model and begin to apply it to a simple problem, and then discuss how to move further with the model. The paper will investigate the sensitivity of the model and the difficulty of achieving accuracy.

Finally, this paper will discuss the possibilities of improving an implemented hidden Markov model, including machine learning.

Introduction

Markov chains are a useful method of modeling real-life transition state networks. A Markov chain model consists of a known set of possible states and a known set of transition probabilities between these states, as well as a set of initial probabilities. The probability distribution of the state of the system changes every time step. The next state probability distribution is dependent on the current state probability distribution, and nothing previous to that. This property — that the next state depends on the current state and not on the past states — is called the Markov property, and applies to all forms of Markov models.

Hidden Markov models are similar to Markov chains in that they model states in changing systems. The Markov property applies here too. However, in hidden Markov models, the state of the system is not observable (hence *hidden*). Instead of observing transitions between states, the viewer is able to observe outputs from the model. Where Markov chains had only transition probabilities, hidden Markov models have both transition probabilities and emission probabilities (the probability that a given state will produce a specific output), as well as initial probabilities and the state set and emission set. With the transition and the emission probabilities, useful information about the system can be calculated. The output can either be discrete or continuous, depending on the type of model.

To illustrate the mechanics of hidden Markov models, we will look at a common trivial example. Say there are two states of weather, sunny and cloudy. On the first day, it is equally likely to be sunny or cloudy. If it is sunny, the next day is 75% likely to be sunny and 25% likely to be cloudy. If it is cloudy, the next day is 60% likely to be sunny and 40% likely to be cloudy. Peter decides what he will do on a particular day based on the weather. If it is sunny, he is 80% likely to go to the beach and 20% likely to stay inside. If it is cloudy, he is 10% likely to go to the beach and 90% likely to stay inside.

You are in a different country and cannot see the weather, but you know what he does on each day. Over three days, he goes to the beach, stays inside, and goes to the beach again. Using a hidden Markov model, you will be able to find the most likely sequence of weather patterns over those three days. Your state set is [Sunny, Cloudy]. Your emission set is [Beach, Inside]. The initial probabilities are [0.5, 0.5]. The transition probabilities are [0.75 0.25; 0.60 0.40]. Finally, the emission probabilities are [0.80 0.20; 0.10 0.90]. Using these variables in the model we will create later in this paper, we could answer the following questions about the weather.

The common questions asked of hidden Markov models are, in order of (increasing) complexity:

1. Given a set of states, a set of emissions, transition probabilities, emission probabilities, initial probabilities, and an output sequence, how likely was it that the model produced that output sequence?

2. Given a set of states, a set of emissions, transition probabilities, emission probabilities, initial probabilities, and an output sequence, what was the most likely sequence of states to lead to that output sequence, and how likely was it?

This paper will design and implement a hidden Markov model to answer these questions. The model will then be modified and applied to classifying and predicting parts of speech in simple sentences.

Implementing the Hidden Markov Model

The hidden Markov model was implemented in parts, in the order needed to solve the two problems above. The first step was to set up the model, which consists of a set of states, a set of emissions, transition probabilities, emission probabilities, and initial probabilities. These sets were represented as matrices.

The set of states represents all possible states of the model. The set of emissions contains all possible emissions of the model, from any state. In the initial implementation of the model, only 3 states and 2 outputs were assumed, for simplicity of design and analysis (See Fig. 1).

From here on, N refers to the number of states (3, in this initial model), and M refers to the number of outputs (2 in the initial model). All code shown is reprinted at the end of this paper as the full program, but is reproduced here in parts to illustrate each step of the process.

```
% set of states
% states = ['A', 'B', 'C'];
states = [1,2,3];

% set of outputs
% outputs = ['a', 'b'];
outputs = [1,2];
```

Fig 1. Matrix representations of states and outputs of the model. For simplicity in future code, an abstraction of integers was used instead of characters.

The transition probabilities were represented in an NxN matrix, where *A(i, j)* gave the probability of state *i* transitioning to state *j*. In the initial implementation of the model, the transition probabilities were arbitrary trivial values (See Fig. 2). The emission probabilities were represented in an NxM matrix, where *A(i, j)* gave the probability of state *i* emitting output *j* (See Fig. 3). The initial probabilities were represented as a matrix of size N, where *A(i)* gave the probability of the model starting in state *i* (See Fig. 4).

```
%    A   B   C
% A .50 .25 .25
% B .25 .50 .25
% C .25 .25 .50
```

Fig 2. Matrix representation of transition probabilities. For every state, the probability transitioning to itself was two times higher than the probability of transitioning to another state. This is a pattern seen in many real-world problems and so it was used in this model as well.

```
%    'a' 'b'
% A  .8  .2
% B  .1  .7
% C  .1  .1
```

Fig 3. Matrix representation of emission probabilities. Semi-arbitrary, skewed values were chosen that would make it easy to compare the intuitive output of states to the model's output. Note that the sum of each column is one: stochastic columns are required for the model.

```
% set initial probabilities
initial_probabilities = [.333,.333,.333];
```

Fig 4. Matrix representation of initial probabilities. There was an equal probability of the model starting in each state.

This was the framework of the hidden Markov model. It still cannot do anything without more supporting code.

Probability of an Output Sequence

The second step in implementing the hidden Markov model simulation was to try to solve the first problem: Given a model and an output sequence, what was the probability of that output being produced?

Trivially, there is a "brute-force" way to solve the problem. First, one would enumerate every possible state sequence and the probability of that sequence. State sequences are calculated by iterating through the set of states and adding another to the end of each sequence for every time step. The probability of each state sequence is the sum of the product of the transition probabilities from *i* to *j,* where the state at time *t* is *i* and the state at time *t+1* is *j*.

Next, calculate the probability of the state sequence producing the output sequence, for every calculated state sequence. This probability will be the product of the emission probabilities of the state at time *t* producing the output at time *t*, for every *t* in the length of the output sequence.

The answer to the question, the probability of the output sequence being produced from the model, would be given by the sum of the product of the probability of every state sequence and the probability that that state sequence produces the output sequence. With this method, we would also have the answer to the second problem (What is the most likely state sequence that produced this output sequence?) by taking the state sequence with the highest probability of producing the output sequence. Unfortunately, **the time complexity of this algorithm makes it infeasible**. Enumerating every possible state sequence for an output sequence of length *T* would take on the order of N^T calculations, where *N* is the number of states, because at every time *t* the existing state sequences become *N* times more complex. For example, with our simple model,

At $t = 1$, there are $N = 3$ possible state sequences:
A, B, C
At $t = 2$, there are $N^2 = 9$ possible state sequences:
A-A, A-B, A-C, B-A, B-B, B-C, C-A, C-B, C-C
At $t = 3$, there are $N^3 = 27$ possible state sequences, and
At $t = T$, there are $N^T = ?$ possible state sequences.

On top of calculating these N^T possibilities, we have to multiply every one of them by the probability of it producing the output sequence, which takes *T* additional calculations, one multiplication for each state in the sequence. Even for models on the order of ours, with 3 possible states and 2 possible outputs, if the output is 20 characters, we have to do $3^{20} = 3486784401$ calculations. With more possible states and a bigger output sequence, this method quickly becomes infeasible.

We need a different, more efficient algorithm. Our new algorithm will iterate over time steps, like the previous one, but we will need an important insight. That is, there are only *N* possible states at time *t*. Regardless of the number of partial state sequences that could have led up to time *t*, they can all be merged into *N* paths at every time *t*.

To illustrate the difference between the two strategies, imagine we have output sequence ‘a, b, a, a, b’. At time *t = 1*, we are considering output ‘a’. Three partial state sequences could have led to this point: A, B, or C. We gave them equal initial probabilities, so the probability of each partial state sequence is the initial probability multiplied by the emission probability of ‘a’ from the state. Fig. 5 displays these.

state sequence	probability of state sequence
A	0.333 * 0.8 = 0.2664
B	0.333 * 0.1 = 0.0333
C	0.333 * 0.1 = 0.0333

Fig 5. Partial state sequences of output ‘a, b, a, a, b’ at *t=1* — naive method.
Probability of output sequence ‘a’ = sum of state sequence probabilities = 0.3333.

Intuitively, when we move to *t = 2* and consider output ‘a, b’ the possible state sequences are: A-A, A-B, A-C, B-A, B-B, B-C, C-A, C-B, C-C, and the probabilities are the previous probability of the first state sequence multiplied by the product of the transition probability to the new state and the emission probability of ‘b’ from the new state (See Fig. 6). The probability of the output sequence up to time *t* being produced is given by the sum of the probabilities of all of the partial state sequences at time *t*.

state sequence	probability of state sequence
A-A	0.2664 * (0.50 * 0.2) = 0.02664
A-B	0.2664 * (0.25 * 0.7) = 0.04662
…	…
C-C	0.0333 * (0.50 * 0.1) = 0.001665

Fig 6. Partial state sequences of output ‘a, b, a, a, b’ at *t=2* — naive method.
Probability of the output sequence ‘a, b’ = sum of all state sequence probabilities = 0.1031.
Complete version attached at end of paper.

However, we will soon run into the high complexity mentioned before. To avoid this, we need to make the insight that because there are *N* states, we can merge previous partial state sequences together into *N* partial state sequences. Put another way, **we consider the probability of being in state *s* at time *t*, not the separate probability of every state sequence that is in state *s* at time *t*.** This means that we no longer get the answer to the second problem (What is the most likely state sequence that produced this output sequence?) because we can not and do not track states. We *do* get the answer to the first problem (What is the probability of this output sequence being produced?) by taking the sum of the probabilities of the partial state sequences at *time = T*.

This new algorithm is called the forward algorithm. The tables in Fig 7 and Fig 8 show the model at *t = 1* and *t = 2* under the forward algorithm. Fig 9 shows and explains the mechanism of the forward algorithm.

state sequence	probability of state sequence
A	0.333 * 0.8 = 0.2664
B	0.333 * 0.1 = 0.0333
C	0.333 * 0.1 = 0.0333

Fig 7. Partial state sequences of output ‘a, b, a, a, b’ at *t=1* — forward method.
Probability of output sequence ‘a’ = sum of state sequence probabilities = 0.3333.

state sequence	probability of state sequence
*-A = A-A + B-A + C-A	0.02664 + 0.001665 + 0.001665 = 0.02997
*-B = A-B + B-B + C-B	0.04662 + 0.01155 + 0.005775 = 0.063945
*-C = A-C + B-C + C-C	0.00666 + 0.0008325 + 0.001665 = 0.0091575

Fig 8. Partial state sequences of output ‘a, b, a, a, b’ at *t=2* — forward method.
Probability of output sequence ‘a,b’ = sum of state sequence probabilities = 0.1031.

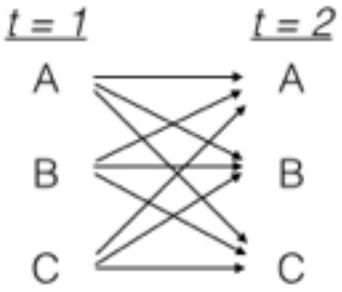

Fig 9. A diagram of state transitions from *t=1* to *t=2* using the forward algorithm. Any state can move to any other state, and because we are no longer producing full state sequences, this does not depend on any states before the previous one, in accordance with the Markov property.

The forward method is commonly used in Hidden Markov models to calculate the joint probability of an emission sequence and a state sequence. With this, we have solved the first problem. The code for implementing the forward method can be found at the end of this paper in Fig. 23.

Most Likely State Sequence

Next, we look at the second problem we will tackle — given a set of states, a set of emissions, transition probabilities, emission probabilities, initial probabilities, and an output sequence, what was the most likely sequence of states to lead to that output sequence, and how likely was it?

First, we will define the “most likely” state sequence to be the state sequence with the highest probability *along the entire path*. We do not want the state sequence of the most probable state per time, because we want a connected path.

As we saw when solving the first problem, using a brute force technique will quickly become too expensive and time-consuming. Again, we need a more insightful method. The Viterbi algorithm is most commonly used to solve this problem in hidden Markov models. It is based on dynamic programming, meaning that it breaks the problem into subproblems and stores the answer to those subproblems in order to make future computation quicker.

We need to keep track of two things: the path with the best score, for every *t*, and the state which led to that best score, for every *t*. Both of these variables will be stored as TxN matrices, where *N* = number of states and *T* = length of output. We'll call the first variable *probs* and the second *bp*, for backpointers.

For every *t*, we fill in a column of *probs* and a column of *bp*. We do this with the following formulas:

*probs(t, i) = (max(probs(t-1, j) * transition_prob(j, i)) * emission_prob(i, output(t))*
*bp(t, i) = (argmax(probs(t-1, j) * transition_prob(j, i))*

In English, the highest partial state sequence probability at time *t* and state *i* is the maximum of the products of the probability of being in state *j* in time *t-1* and the probability of transitioning to state *i* from state *j,* for all *j*, all multiplied by the probability that state *i* emits the correct output at time *t*.

To illustrate this algorithm, let us again assume a given output of 'a, b, a, a, b'. Our *N* is 3 and our *T* is 5, so we need two 5x3 matrices. The first column of *probs* and *bp* are initialized such that:

*probs(1, i) = initial_prob(i) * emission_prob(i, output(1))*
bp(1, i) = 0

Fig. 10-13, located at the end of the paper, show the progression of this algorithm. At *t = 5*, we will have the following:

probs, t = 5					**Fig. 14**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
probability of state 1	0.333 * 0.8 = 0.2664	0.02664	0.0107	0.0043	0.0004
probability of state 2	0.333 * 0.1 = 0.0333	0.04662	0.0023	0.0003	0.0007
probability of state 3	0.333 * 0.1 = 0.0333	0.00666	0.0012	0.0003	0.0001

bp, t = 5					**Fig. 15**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
origin of state 1	0	1	1	1	1
origin of state 2	0	1	2	1	1
origin of state 3	0	1	2	1	1

From here, it is easy to see how we trace back through the matrices to construct the most likely sequence. We start at the last column, where $t = T$, and find the highest probability in *probs*. The corresponding entry in *bp* will tell us ("point" to) what the most likely state was at $t = T - 1$. From there, we follow entries ("pointers") back though *bp,* constructing the sequence backwards until we get to $t = 1$.

In this example, we start at the highest state probability at time T, which occurs at *probs(5, 2)*. We add state 2 to our most likely sequence. The corresponding entry is *bp(5, 2),* and it points to state 1. We add state 1 to our most likely sequence. We go back to *bp(4, 1)*, and see that it also points to state 1. Again, we add state 1 to our most likely sequence. We then go back to *bp(3, 1)* and see that it also points to state 1. We follow this procedure until we reach $t = 1$, when we know we have constructed a full state sequence: 2, 1, 1, 1, 1.

The Viterbi algorithm runs in $T*N^2$ time complexity, much better than the brute force method.

Now our model can answer two very useful but basic questions: "What was the probability a given output sequence was produced?", and "What was the most likely state sequence that produced that probability?" However, it still has only three states and two outputs, and only models arbitrary, trivial transitions between those states. The next, more exciting, step is to apply the hidden Markov model to a problem.

Part of Speech Tagging and Prediction

Hidden Markov models have been discussed in the literature for part-of-speech tagging for many years. Part of speech taggers have been implemented in various ways, with varying levels of complexity. We will see that our model, too, can be adapted to do part-of-speech tagging and even part-of-speech prediction of unknown words.

The accepted definition of part-of-speech tagging is that given a sentence or partial sentence, a model will label each word with the most probable part of speech. This implementation of part-of-speech tagging will work with basic declarative sentences, to simplify the model, but can always be expanded for compatibility with other types of sentences.

In solving this problem, the set of states are made to represent the set of parts of speech. Each emission in the set of emissions will represent words. Thus, we will give our model a sequence of emissions (a sentence), and it will calculate the most likely sequence of parts of speech, which we can parse into the most likely part of speech for each word.

We begin implementing this in our model by modifying the states from the trivial to the more complex. Only the basic parts of speech will be used: nouns, verbs, adjectives, and adverbs. The model also needs a vocabulary of words it can recognize and knows the part of speech identity of. The vocabulary should be basic for simplicity's sake, but there are certain requirements. There should be a certain percentage of nouns, verbs, adjectives, and adverbs, so that the model will have a basis to recognize others of each of those categories. Fig. 16 shows the set of states and set of emissions that were used in the model.

The vocabulary used has six verbs (run, walk, sleep, eat, is, and are); six nouns (you, me, we, he, she, and it); two adverbs (fast, slow); and one unknown variable. The unknown variable will be used in the prediction phase of the model — if an unknown word is given to the model in a sentence, it should be able to predict what part of speech that word is using its context.

```
% set of states
% states = [noun, verb, adjective, adverb];
  states = [1, 2 , 3, 4];

% set of outputs
% outputs = [run, walk, sleep, eat, you, me, we, fast, slow, he, she, it, is, are, food, unknown];
outputs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16];
```

Fig. 16. Set of states and emissions. N = 4; M = 16.

Most words are actually more complex than suggested, in that they are not simply one part of speech. ‘Run’, for example, can be a noun or a verb, as can most of the other verbs. The nouns are all simply nouns, but ‘fast’ and ‘slow’ are even more interesting cases. ‘Fast’ can be a noun, meaning to abstain from food; a verb, meaning the action of abstaining from food; an adjective, meaning quick; and an adverb, meaning to perform an action quickly. These complications are incorporated into the model in the emission probabilities, which is represented as a NxM matrix (See Fig. 17). The emission probabilities are an abstraction for the part of speech each word is most likely to be.

```
% set of emission probabilities
emission_probabilities = [
%run, walk, sleep, eat, you, me, we, fast, slow, he, she, it, is, are, food, unknown
 .2   .2     .2    .2    1   1   1   .1    0    1    1    1   0   0    1   .25; % noun
 .8   .8     .8    .8    0   0   0   .1    .1   0    0    0   1  .15   0   .25; % verb
  0    0     0     0     0   0   0   .3    .8   0    0    0   0  .85   0   .25; % adjective
  0    0     0     0     0   0   0   .4    .1   0    0    0   0   0    0   .25  % adverb
];
```

Fig. 17 Emission probabilities.
emission_probabilities(i, j) represents the probability that word *i* acts as part of speech *j* in a sentence.

We see that words like ‘fast’, which can be any of the listed parts of speech, are given probabilities for each part of speech state. These probabilities can be thought of as the likelihood that, when we see ‘fast’ in a sentence, it is acting as that part of speech. The adjective and adverb ‘fast’ are more commonly used than the noun or verb ‘fast’, so they are given higher probabilities. The emission probabilities of the unknown variable are all 25% — we are assuming that an unknown word could be any of the parts of speech.

Next, we have to define transition probabilities for the model: the likelihood that any part of speech comes after any other part of speech. These were roughly calculated by looking at sample sentences and consulting grammatical rules and observing which parts of speech were more likely to come after one another. The sample sentences were all simple declarative sentences, so the model will be tuned for that kind of sentence. Fig. 18 shows the conclusions of this rough calculation.

```
%       noun  verb  adj  adv
% noun  .05    .5   .15   .2
% verb  .3      0   .3    .4
% adj   .65    .1   .2     0
% adv   0      .4   .35   .4
```

Fig. 18 Transition probabilities between parts of speech. *transition_probabilities(i, j)* represents the probability of a transition from state *i* to state *j*, or the likelihood that part of speech *j* follows part of speech *i* in a sentence.

We see in the transition probabilities that nouns are more likely to follow adjectives than verbs, other adjectives, or adverbs. This makes sense because adjectives mainly describe and apply to nouns. Similarly, nouns are most precede verbs because verbs usually act on nouns. The transition probabilities are what makes this model interesting — they define the types of sentences that the model tries to fit emissions into. Fine-tuning the transition probabilities could produce interesting changes in the model.

Finally, we define initial probabilities for the model (Fig. 19). A basic declarative sentence is far more likely to begin with a noun than a verb, so the initial probabilities reflect that.

```
% set initial probabilities
initial_probabilities = [.65, .05, .2, .1];
```

Fig. 19 Initial probabilities

At this point, the model is ready to start modeling sentences and classifying words. Simple converters can be implemented to convert a sentence into a sequence of numbers, which is what the model accepts as emission sequences, and to convert the most likely state sequence, which it will output as a sequence of numbers, into a sequence of parts of speech. Code for these two converters can be found in Fig. 20-21 at the end of the paper.

Consider the simple declarative sentence “We eat food.” This is composed of words in the model’s vocabulary, so it should be able to classify it with little trouble. And indeed, if we give the model the sentence “We eat food,” it returns the correct most likely state sequence ‘noun, verb, noun’.

Now consider the sentence “We eat good food.” This time, it contains the word ‘good’, which is not in the model’s vocabulary. If we input this to the model, it returns . . . ‘noun, verb, adjective, noun’. Fig. 22 shows a table of more inputs and the most likely sequence returned by the model.

(input) emission sequence	(output) most likely state sequence
We eat food.	noun, verb, noun
We eat **good** food.	noun, verb, adjective, noun
We eat **fish**.	noun, verb, **adverb**
We eat **ravenously**.	noun, verb, adverb
We **hungrily** eat food.	noun, adverb, verb, noun
We eat food **happily**.	noun, verb, noun, adverb
We **devour** food.	noun, verb, noun

(input) emission sequence	(output) most likely state sequence
You run fast.	noun, verb, adverb
You run **very** fast.	noun, verb, adverb, adverb

Fig. 22 Input emission sequences and output most likely state sequences. Words unknown to the model are shown in bold in the left column, and incorrect answers are shown in bold in the right column.

Nearly all the examples are correct. The exception is with the sentence "We eat fish," which produced the result 'noun, verb, adverb'. This can be explained. First, nearly all verbs can precede adverbs, while only some verbs can precede nouns. The verbs in the model's vocabulary are verbs that are more likely to have adverbs following them ('sleep', for example, cannot be followed by a noun). Thus the model is tuned to the words in its vocabulary. With a bigger vocabulary, it would be a better model for more of the English language.

The inclusion of heuristics would also make the model better and more interesting. For example, a text analysis on unknown words would help determine what part of speech they were (words that end with 'ly' are more likely to be adverbs, words that start with capital letters in the middle of the sentence are more likely to be nouns). If a word contained another word that was in the model's vocabulary, the model could make a more educated guess about its part of speech.

Conclusions

Obviously, the model we have created will not completely accurately model the English language. It is limited in that the variables, in particular the transition probabilities, were manually curated to predict parts of speech in simple declarative sentences. In other words, the probabilities used reflect the probabilities of parts of speech in simple declarative sentences. It is also limited in that it only recognizes four of the more than ten parts of speech. Finally, it is limited by the size and constitution of its vocabulary. With more words and a better variety of words, the model would improve.

One way to improve all these factors would be to manually add more words to the vocabulary, add more parts of speech to the state probabilities We would also have to analyze more types of sentence (interrogative, imperative) for the part of speech transitions, and use that to improve the transition probabilities.

Instead of doing all of this manually, it would be interesting to use machine learning. Hidden Markov models are often used with machine learning. The model is 'trained' on a data set where the correct answers are known, and refines its own transition and emission probabilities based on these answers. Training this model on different sentence types and more sentences of each type would unquestionably improve the accuracy of the model and increase its applicability.

Another method of improving our hidden Markov model would be to have it look at the immediate context of a word to help determine part of speech. A second-order hidden Markov model would accomplish this without violating the Markov property by using a second level of abstraction, and could look at the just the two words preceding the word in question.

The hidden Markov model is a very useful statistical modeling technique for temporal data. Because there are always adjustments that can be made to the model to make it more accurate, the possibilities are endless.

References

"5 Parts of Speech." Your Dictionary (2005): n. pag. Your Dictionary. Web. <http://www.yourdictionary.com/index.php/pdf/articles/162.partsofspeechexamples.pdf>.

Collins, Michael. "Hidden Markov Models." *Springer Reference* (2011): n. pag. *Columbia University*. Columbia University. Web. <http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/hmms.pdf>.

Mitrophanov, A. Y., Lomsadze, A., & Borodovsky, M.. (2005). Sensitivity of Hidden Markov Models. Journal of Applied Probability, 42(3), 632–642. Retrieved from http://www.jstor.org/stable/30040846

Rabiner, Lawrence. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition - Proceedings of the IEEE (n.d.): n. pag. University of British Columbia. University of British Columbia, Feb. 1989. Web. <http://www.cs.ubc.ca/~murphyk/Bayes/rabiner.pdf>.

Scarince, Christine. "Declarative Sentence: Definition & Examples | Study.com." Study.com. Study.com, n.d. Web. 17 Dec. 2015. <http://study.com/academy/lesson/declarative-sentence-definition-examples.html>.

Singh, Mona. "Hidden Markov Models." *SpringerReference* (2011): n. pag. *Princeton University*. Princeton University, Oct. 1999. Web. <https://www.cs.princeton.edu/~mona/Lecture/HMM1.pdf>.

Sleator, Daniel. "Sample Sentences." Sample Sentences. Carnegie Mellon University, n.d. Web. 17 Dec. 2015. <http://www.link.cs.cmu.edu/link/batch.html>.

Tishby, Naftali, Shai Fine, and Yoram Singer. "The Hierarchical Hidden Markov Model: Analysis and Applications." *Machine Learning* (n.d.): n. pag. *Princeton University*. Princeton University, 1998. Web. <The Hierarchical Hidden Markov Model: Analysis and Applications>.

Additional Figures & Code

state sequence	probability of state sequence
A-A	0.2664 * (0.50 * 0.2) = 0.02664
A-B	0.2664 * (0.25 * 0.7) = 0.04662
A-C	0.2664 * (0.25 * 0.1) = 0.00666
B-A	0.0333 * (0.25 * 0.2) = 0.001665
B-B	0.0333 * (0.50 * 0.7) = 0.01155
B-C	0.0333 * (0.25 * 0.1) = 0.0008325
C-A	0.0333 * (0.25 * 0.2) = 0.001665
C-B	0.0333 * (0.25 * 0.7) = 0.005775
C-C	0.0333 * (0.50 * 0.1) = 0.001665

Fig 6. (Complete)
Partial state sequences of output ‘a, b, a, a, b’ at *t=2* — naive method.
Probability of the output sequence ‘a, b’ = sum of all state sequence probabilities = 0.1031.

```
function [ p1_output_prob, p2_max_final_prob, p2_most_likely_seq, cml_seq] = hmmWrapper()

emission = {'we', 'eat', 'food'};
cemission = zeros(1, numel(emission));
% convert input to numbers
% outputs = [run, walk, sleep, eat, you, me, we, fast, slow, he, she, it, is, are, food, unknown];
outputs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14, 15];
unknown = 0;

for i = 1:numel(emission)
   if strcmp(emission(i),'run') || strcmp(emission(i),'runs')
       cemission(i) = 1;
   elseif strcmp(emission(i), 'walk') || strcmp(emission(i),'walks')
       cemission(i) = 2;
   elseif strcmp(emission(i), 'sleep') || strcmp(emission(i), 'sleeps')
       cemission(i) = 3;
   elseif strcmp(emission(i), 'eat')||strcmp(emission(i), 'eats')
       cemission(i) = 4;
   elseif strcmp(emission(i), 'you')
       cemission(i) = 5;
   elseif strcmp(emission(i), 'me')||strcmp(emission(i), 'I')
       cemission(i) = 6;
   elseif strcmp(emission(i), 'we')
       cemission(i) = 7;
   elseif strcmp(emission(i), 'fast')
       cemission(i) = 8;
   elseif strcmp(emission(i), 'slow')
       cemission(i) = 9;
   elseif strcmp(emission(i), 'he')
       cemission(i) = 10;
   elseif strcmp(emission(i), 'she')
       cemission(i) = 11;
   elseif strcmp(emission(i), 'it')
       cemission(i) = 12;
   elseif strcmp(emission(i), 'is')
       cemission(i) = 13;
   elseif strcmp(emission(i), 'are')
       cemission(i) = 14;
   elseif strcmp(emission(i), 'food')
       cemission(i) = 15;
   else
       cemission(i) = 16;
       unknown = i;
   end
end


[ p1_output_prob, p2_max_final_prob, p2_most_likely_seq] = hmm(cemission);

% convert ml_seq back to parts of speech
% states = [noun, verb, adjective];
% states = [1, 2 , 3];
cml_seq = cell(length(p2_most_likely_seq), 1);
for i = 1:length(p2_most_likely_seq)
    if p2_most_likely_seq(i) == 1
        cml_seq{i,1} = 'noun';
    elseif p2_most_likely_seq(i) == 2
        cml_seq{i,1} = 'verb';
    elseif p2_most_likely_seq(i) == 3
        cml_seq{i,1} = 'adjective';
    elseif p2_most_likely_seq(i) == 4
        cml_seq{i,1} = 'adverb';
    end
end

if unknown > 0
    fprintf('"%s" is most likely a(n) %s', emission{1,unknown}, cml_seq{unknown, 1});

end

```

Fig. 20,21 Converters between numbers and inputs/outputs. Wrapper function to run Hidden Markov Model. By saving the unknown word, it is able to specifically print the part of speech predicted for it.

Fig. 10 and 11 show the initial values of *probs* and *bp*, at *t = 1*.
Fig. 12 and 13 show the values of *probs* and *bp* after one time step, at *t = 2*.

probs, t = 1					**Fig. 10**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
probability of state 1	0.333 * 0.8 = 0.2664	-	-	-	-
probability of state 2	0.333 * 0.1 = 0.0333	-	-	-	-
probability of state 3	0.333 * 0.1 = 0.0333	-	-	-	-

Fig. 10

bp, t = 1					**Fig. 11**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
origin of state 1	0	-	-	-	-
origin of state 2	0	-	-	-	-
origin of state 3	0	-	-	-	-

Fig. 11

probs, t = 2					**Fig. 12**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
probability of state 1	0.333 * 0.8 = 0.2664	0.02664	-	-	-
probability of state 2	0.333 * 0.1 = 0.0333	0.04662	-	-	-
probability of state 3	0.333 * 0.1 = 0.0333	0.00666	-	-	-

Fig. 12

bp, t = 2					**Fig. 13**
	t = 1	**t = 2**	**t = 3**	**t = 4**	**t = 5**
origin of state 1	0	1	-	-	-
origin of state 2	0	1	-	-	-
origin of state 3	0	1	-	-	-

Fig. 13

```
function [ p1_output_prob, p2_max_final_prob, p2_most_likely_seq] = hmmBasic(test_output)

% ------------------------------------------------------------------------
% SET UP THE MODEL

% set of states
% states = ['A', 'B', 'C'];
states = [1,2,3];

% set of outputs
% outputs = ['a', 'b'];
outputs = [1,2];

% set state transition probabilities
transition_probabilities = zeros(3,3);
for i = 1:3
    for j = 1:3
        transition_probabilities(i,j) = .25;
    end
end
transition_probabilities(1,1) = .5;
transition_probabilities(2,2) = .5;
transition_probabilities(3,3) = .5;

%    A   B   C
% A .50 .25 .25
% B .25 .50 .25
% C .25 .25 .50

% set emission probabilities
emission_probabilities = [.8, .2;
                          .1, .7;
                          .1, .1];

%    'a' 'b'
% A  .8  .2
% B  .1  .7
% C  .1  .1


% set initial probabilities
initial_probabilities = [.333,.333,.333];

%------------------------------------------------------------------------

% PROBLEM 1
% given an observation sequence of length T, what was the probability of
% that output?
%test_output = [1,2, 1, 1, 2];
forward_var = zeros(length(test_output), length(states));
forward_var(1,1) = initial_probabilities(test_output(1));

for i = 1:length(states)
    forward_var(1,i) = initial_probabilities(i) * emission_probabilities(i, test_output(1));
end

for t = 1:(length(test_output)-1)
    for j = 1:length(states)
        sum = 0;
        for i = 1:length(states)
            sum = sum + forward_var(t,i) * transition_probabilities(i,j);
        end
        forward_var(t+1, j) = sum * emission_probabilities(j, test_output(t+1));
    end
end
```

Fig. 23 Basic Hidden Markov Model, part 1.
Continued on next page.

```
66
67
68 -  p1_output_prob = 0;
69 -  for i = 1:length(states)
70 -      p1_output_prob = p1_output_prob + forward_var(length(test_output), i);
71 -  end
72
73
74    %-------------------------------------------------------------------------
75    % PROBLEM 2
76    % given an observation sequence of length T, what was the most likely
77    % sequence of states to lead to it? (and the probability of that sequence?)
78
79    % will hold our final answer - the most likely sequence
80 -  p2_most_likely_seq = zeros(length(test_output),1);
81
82    % delta will hold the best score for paths up to that point, for every t
83    % phi keeps track of which state was the best, for every t
84    % We will use both delta and phi to trace back and find the most likely seq
85 -  probs = zeros(length(test_output), length(states));
86 -  backpointers = zeros(length(test_output), length(states));
87
88    % initialize
89 -  for i = 1:length(states)
90 -      probs(1,i) = initial_probabilities(i) * emission_probabilities(i, test_output(1));
91 -      backpointers(1,i) = 0;
92 -  end
93
94    %
95 -  for t = 2:length(test_output)
96 -      for j = 1:length(states)
97 -          max = -1;
98 -          argmax = -1;
99 -          for i = 1:length(states)
100 -             if (probs(t-1, i) * transition_probabilities(i,j)) > max
101 -                 max = (probs(t-1,i) * transition_probabilities(i,j));
102 -                 argmax = i;
103 -             end
104 -         end
105 -         probs(t,j) = max * emission_probabilities(j,test_output(t));
106 -         backpointers(t,j) = argmax;
107 -     end
108 - end
109
110 - p2_max_final_prob = -1;
111 - argmax = -1;
112 - for i = 1:length(states)
113 -     if probs(length(test_output), i) > p2_max_final_prob
114 -         p2_max_final_prob = probs(length(test_output),i);
115 -         argmax = i;
116 -     end
117 - end
118
119 - p2_most_likely_seq(length(test_output)) = argmax;
120
121 - for t = length(test_output)-1:-1:1
122 -     p2_most_likely_seq(t) = states(backpointers(t+1, p2_most_likely_seq(t+1)));
123 - end
124
125
```

Fig. 23 Basic Hidden Markov Model, part 2.