CSCI 3202 Introduction to Artificial Intelligence
Instructor: Hoenigman
Assignment 5
Due: Wednesday, Dec 10 before 3pm.

Hidden Markov Models

This assignment is about hidden Markov models (HMMs) and their many potential applications. The main components of the assignment are the following:

1. Implement a method to build an HMM from data;
2. Implement the Viterbi algorithm for finding the most likely sequence of states through the HMM, given "evidence"; and
3. Run your code on several datasets and explore its performance.

There is also an optional (extra credit) part to this assignment involving second-order Markov models, as described below.

Building an HMM from data

The first part of the assignment is to build an HMM from data. Recall that an HMM involves a hidden state that changes over time, as well as observable evidence that is used to infer the hidden state. In our example from lecture, we used the observation of an umbrella to infer whether it was raining outside. An HMM is defined by three sets of probabilities:

1. For each state s, the probability of observing each output o at that state ($P(E[t]=o \mid X[t]=s)$
2. From each state s, the probability of traversing to every other state s' in one time step ($P(X[t+1]=s' \mid X[t]=s)$)
3. A distribution over the start state ($P(X[0])$).

For the start state distribution (P(X[0])), in this assignment, you will assume that there is a single dummy start state, distinct from all other states, and to which the HMM can never return. Even so, you will need to estimate the probability of making a transition from this dummy start state to each of the other states (this is implicit in part 2, but may need to be done explicitly by your program).

Generate the conditional probability tables

For items 1 and 2, your job will be to compute estimates of these probabilities from data. There are files on Moodle with training data consisting of one or more sequences of state-output pairs, i.e., sequences of the form $x[1]$, $e[1]$, $x[2]$, $e[2]$, ..., $x[n]$, $e[n]$. You will use this training data to estimate the conditional probability tables that define the HMM. For instance, to estimate the probability of output o being observed in state s, you count the number of times that output o appears with state s in the given data, and divide by a normalization constant (so that the probabilities of all outputs from that state add up to one). In this case, that normalization constant would simply be the number of times that state s appears at all in the data.

Smooth the estimates

In generating these conditional probabilities, you also need to *smooth* the estimates. As an example, consider flipping a coin for which the probability of heads is p, where p is unknown, and our goal is to estimate p. The obvious approach is to count how many times the coin comes up heads and divide by the total number of coin flips. If we flip the coin 1000 times and it comes up heads 367 times, it is very reasonable to estimate p as approximately 0.367. However, suppose we flip the coin only twice and we get heads both times. Is it reasonable to estimate p as 1.0? Intuitively, given that we only flipped the coin twice, it seems a bit rash to conclude that the coin will always come up heads, and smoothing is a way of avoiding such rash conclusions.

A simple smoothing method, called Laplace smoothing (or Laplace's law of succession or add-one smoothing in your textbook), is to estimate p by (one plus the number of heads) / (two plus the total number of flips). Said differently, if we are keeping count of the number of heads and the number of tails, this rule is equivalent to starting each of our counts at one, rather than zero. This latter view generalizes to the case in which there are more than two possible outcomes (for instance, estimating the probability of a die coming up on each of its six faces).

Another advantage of Laplace smoothing is that it avoids estimating any probabilities to be zero, even for events never observed in the data. For HMMs, this is important since zero probabilities can be problematic for some algorithms.

For this assignment, you should use Laplace-smoothed estimates of probabilities. For instance, returning to the problem of estimating the probability of output o being observed in state s, you would use *one plus* the number of times output *o* appears in state *s* in the given data, divided by a normalization constant. In this case, the normalization constant would be the number of times state *s* appears in the data, plus the total number of possible outputs. You will need to also work out Laplace-smoothed estimates for item 2, i.e., for the probability of making a transition from one state to another, as well as the probability of making a transition from the dummy start state to any of the other states. You do not need to do Laplace smoothing for the P(E[t] = o | X[t] = s) calculations.

Finding the most likely sequence

Once you have your conditional probability tables for your HMM built, the second part of the assignment is to write code that computes the most probable sequence of states (according to the HMM that you built from data) for a given sequence of outputs. This is essentially the problem of implementing the Viterbi algorithm as described in class and in your textbook.

The second part of each of the provided datasets consists of test sequences of state-output pairs. You will provide your Viterbi code with just the output part of each of these sequences, and from this, you must compute the most likely sequence of states to produce such an output sequence. The state part of these sequences is provided so that you can compare the estimated state sequences generated by your code to the actual state sequences that generated this data. Note that these two sequences will *not* necessarily be identical, even if you have correctly implemented the Viterbi algorithm.

A numerical tip: the Viterbi algorithm involves multiplying many probabilities together. Since each of these numbers is smaller than one (possibly much smaller), you can end up working with numbers that are tiny enough to be computationally indistinguishable from zero. To avoid this problem, it is recommended that you work with log probabilities. To do so, rather than multiplying two probabilities *p* and *q*, simply add their logarithms using the rule:

$$\log(pq) = \log(p) + \log(q).$$

You will probably find that there is never a need to store or manipulate actual probabilities; instead, everything can be done using log probabilities.

Exploring performance on actual datasets

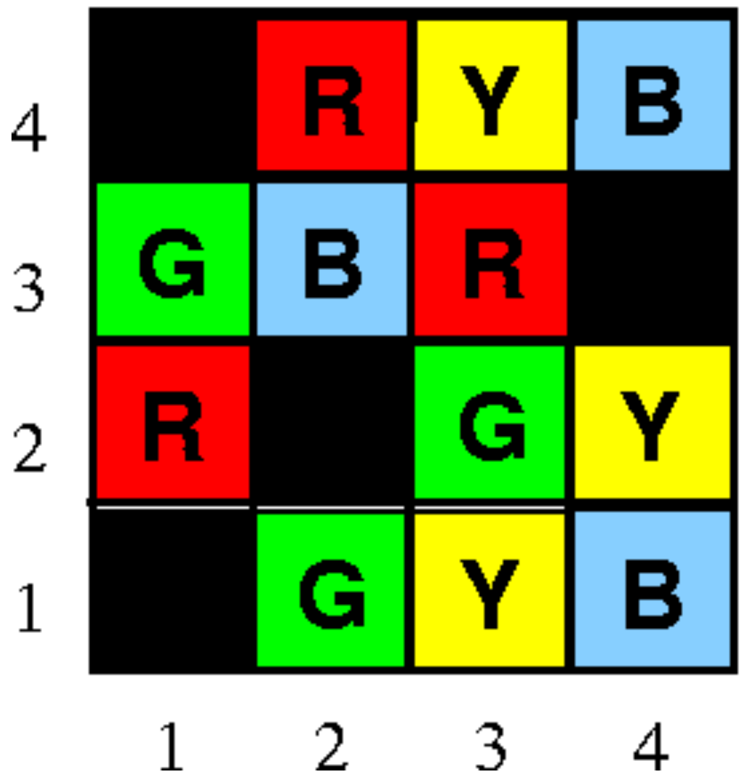
There is a zip file on Moodle called Assignment5DataSets that includes all data sets needed for this assignment. You will use these data sets with your HMM to solve:

1. A robot toy problem in which the goal is to infer the sequence of locations visited by the robot based on sensor readings;
2. The problem of correcting typographical errors without a dictionary; and
3. The problem of inferring and tracking changing topics in a stream of text.

Your algorithms should be implemented in such a way that they are generic enough to solve all three of these problems without any re-implementations of functionality. For example, the routine your write to build the HMM should work for all three data sets.

Toy robot

In this problem, a robot is wandering through the following small world:



The robot can only occupy the colored squares. At each time step, the robot attempts to move up, down, left or right, where the choice of direction is made at random. If the robot attempts to move onto a black square, or to leave the confines of its world, its action has no effect and it does not move at all. The robot can only sense the color of the square it occupies. However, its sensors are only 90% accurate, meaning that 10% of the time, it perceives a random color rather than the true color of the currently occupied square. The robot begins each walk in a randomly chosen colored square.

In this problem, state refers to the location of the robot in the world in *x*:*y* coordinates, and output refers to a perceived color (`r`, `g`, `b` or `y`). Thus, a typical random walk looks like this:

```
3:3 r
3:3 r
3:4 y
2:4 b
3:4 y
3:3 r
2:3 b
1:3 g
2:3 b
2:4 r
3:4 y
4:4 y
```

Here, the robot begins in square 3:3 perceiving red, attempts to make an illegal move (to the right), so stays in 3:3, still perceiving red. (You can determine that the robot attempted to move right because the square to the right is illegal, and this is the only move that can explain the robot staying in the same square.) On the next step, the robot moves up to 3:4 perceiving yellow, then left to 2:4 perceiving blue (erroneously), and so on.

By running your program on this problem, you will build an HMM model of this world. Then, given only sensor information (i.e., a sequence of colors), your program will re-construct an estimate of the actual path taken by the robot through its world.

The data for this problem is on Moodle, called `robot_no_momentum.data`, and it contains 200 training sequences (random walks) and 200 test sequences, each sequence consisting of 200 steps.

There is another file on Moodle, called `robot_with_momentum.data`, that contains data on a variant of this problem in which the robot's actions have "momentum" meaning that, at each time step, with 85% probability, the robot continues to move in the direction of the last move. For example, if the robot moved (successfully) to the left on the last move, then with 85% probability, it will again attempt to move left. If the robot's last action was unsuccessful, then the robot reverts to choosing an action at random.

Correcting typos without a dictionary

Problem 2 deals with the problem of correcting typos in text without using a dictionary. Here, you will be given text containing many typographical errors and the goal is to correct as many typos as possible.

In this problem, state refers to the correct letter that should have been typed, and output refers to the actual letter that was typed. Given a sequence of outputs (i.e., actually typed letters), the problem is to reconstruct the hidden state sequence (i.e., the intended sequence of letters). Thus, data for this problem looks like this:

```
i i
n n
t t
r r
o o
d x
u u
c c
t t
i i
o i
n n
_ _
t t
h h
e e
_ _
```

where the left column is the correct text and the right column contains text with errors.

Data for this problem was generated as follows: starting with a text document, in this case, the Unabomber's Manifesto, which was chosen not for political reasons, but for its convenience being available on-line and of about the right length, all numbers and punctuation were converted to white space and all letters converted to lower case. The remaining text is a sequence only over the lower case letters and the space character, represented in the data files by an underscore character. Next, typos were artificially added to the data as follows: with 90% probability, the correct letter is transcribed, but with 10% probability, a randomly chosen neighbor (on an ordinary physical keyboard) of the letter is transcribed instead. Space characters are always transcribed correctly. In a harder variant of the problem, the rate of errors is increased to 20%. The first (roughly) 20,000 characters of the document have been set aside for testing. The remaining 161,000 characters are used for training.

As an example, the original document begins:

```
introduction the industrial revolution and its consequences have been a
disaster for the human race they have greatly increased the life
expectancy of those of us who live in advanced countries but they have
destabilized society have made life unfulfilling have subjected human
beings to indignities have led to widespread psychological suffering in
the third world to physical suffering as well and have inflicted severe
damage on the natural world the continued development of technology
will worsen the situation it will certainly subject human beings to
greater indignities and inflict greater damage on the natural world it
will probably lead to greater social disruption and psychological
suffering and it may lead to increased physical suffering even in
advanced countries the industrial technological system may survive or
it may break down if it survives it may eventually achieve a low level
of physical and psychological suffering but only after passing through
a long and very painful period of adjustment and only at the cost of
permanently reducing human beings and many other living organisms to
engineered products and mere cogs in the social machine
```

With 20% noise, it looks like this:

```
introductipn the industfial revolhtjon and its consequences bafw newn a
diszster rkr the yumab race thdy have grwatky increased the ljte
esoectandy od thosr of is who libe in advanced coubfries but they have
fewtabipuzee xociwty have made life ujfuorillkng have wubjwdted humah
beints to incihbjtids have led to qidespreze lsyxhllotical shffeding kn
tne third wkrld to phyxicql sufcefimg as weol and hqve ingoidtex srvere
damsge on the natural world the confinued developmeng of twvhjllogy
will wotsen thd situation it wull certaknly sunjrct yyman beingw tl
greater ibdignities snd infpixt greagwr damsge on fhe natural alrld it
wjlk probably lwad tk grezter sofiqp disrupgiln and pstchokofucal
wufterkng anc it may kead fl uncreqxed pgusiczl sucfreinh even in
acgajved countries the indhsteial tedhnologicak system may survivr or
ut nay brezk down uf it survives it nay evenyuakly achieve a los lwvel
of phyxkcal and psycyoligical sufveribg but only after passing theough
a long amd very painful periox od adjuwtmebt and only at the fost kf
permsnently reducing hymaj veings abs nsjy otgwr kuving orbanisms to
envineered leoduxfs amd mere clgs in thr soxiap maxhjne
```

The error rate (fraction of characters that are mistyped) is about 16.5% (less than 20% because space characters were not corrupted).

The text reconstructed using an HMM with the Viterbi algorithm looks like this:

```
introduction the industrial revoluthon and its consequences bare neen a
dissster ror the tuman race they have greatly increased the lite
esoectandy od those of is who libe in advanced counfries but they have
festabupusee cocisty have made live intiorilling have wibjested human
beints to incingitids have led to widesprese lsysullotical suffeding in
the third world to physical surcefing as weol and have ingoistes severe
damage on the natural world the continued developmeng of techillogy
will wotsen the situation it will certaknly sunirct tyman beinge tl
```

```
greater indithities and infoist greager damage on the natural aleld it
will probably owad to grester sofial distuption and pstchomofucal
wiftering and it may kead fl increqxed ogusical suctreing even in
achanved countries the industeial technologicak system may survive or
ut nay break down if it survives it nay eventually achieve a los level
of physical and psycholigical survering but only arter passing theough
a long and very paindul perios od adjustment and only at the fost of
permanently reducing human veings ans nany other kiving organisms to
envineered leodusts and mere clys in the social machine
```

The error rate has dropped to about 10.4%.

If you do the extra credit part of this assignment, which involves building a second-order Markov process, you will get reconstructed text that looks like this:

```
introduction the industrial revolution and its consequences have neen a
disaster for the human race they have greatly increased the lite
expectandy of those of is who live in advanced coubtries but they have
restabilized society have made life untiorilling have subjected human
beints to incihbuties have led to widesprese psychological suffering in
the third world to physical suffering as well and have ingoisted severe
damage on the natural world the confinued developmeng of technology
will witsen the situation it will certainly subject human beinge to
greater indignities and inflist greater damage on the natural alrld it
will probably lead to greater social disruption and psychological
suffering and it may lead to uncreased physical suffering even in
actaived countries the industrial technological system may survive or
it may break down if it survives it may eventually achieve a lis level
of physical and psychological suffering but only after passing through
a long and very painful perild of adjuwtment and only at the fost of
permanently reducing human beings ans many other living organisms to
envineered produsts and mere clgs in the social machine
```

The error rate now has dropped even further to about 5.8%.

Data for this part of the assignment is in `typos10.data` and `typos20.data`, representing data generated with a 10% or 20% error rate, respectively.

Tracking a changing topic

Problem 3 deals with the problem of tracking a changing topic, for instance, in a conversation or while watching the news. Your program will be provided with a stream of words, first on one topic, then on another topic, and so on. The goal is to segment the stream of text into blocks, and to identify the topic of each of these blocks. There are six possible topics, namely, baseball, cars, guns, medicine, religion and windows (as in Microsoft). A state in this problem is an underlying topic. An output is an actual word appearing in the text. So for instance, data might look like this:

```
baseball when
baseball are
baseball the
baseball yankees
baseball planning
baseball on
baseball activating
baseball melido
baseball perez
medicine one
medicine of
medicine my
medicine friends
medicine has
medicine had
medicine to
medicine go
medicine to
medicine the
medicine doctor
medicine because
medicine he
medicine had
medicine chest
medicine pains
guns     what
guns     to
guns     do
guns     if
guns     you
guns     shoot
guns     somebody
```

To simplify the data file, lines on which the output changes but not the state can be combined. Thus, in terms of coding the data, the above data is exactly equivalent to the following:

```
baseball when are the yankees planning on activating melido perez
medicine one of my friends has had to go to the doctor
medicine because he had chest pains
guns     what to do if you shoot somebody
```

Note that there are *many* possible outputs for this problem, essentially one for every word. Your code should be efficient enough to handle such a large number of possible outputs.

Data for this problem was created as follows: the bodies of 5302 articles were collected from six newsgroups (namely, rec.sports.baseball, rec.auto, talk.politics.guns, sci.med, talk.religion.misc and comp.os.ms-windows.misc) corresponding to the six topics. All punctuation was removed (converted to white space) and all letters converted to lower case. The articles were then randomly permuted and concatenated together forming a sequence of states (topics) and outputs (words). 1500 articles were saved for use in testing, the rest for training.

Using an HMM with the Viterbi algorithm on this data will produce a sequence of topics attached to each of the words. For instance, in the case above, we might get something like the following (This is made up -- not based on an actual run):

```
baseball when
baseball are
baseball the
baseball yankees
baseball planning
baseball on
baseball activating
baseball melido
baseball perez
baseball one
baseball of
baseball my
baseball friends
medicine has
medicine had
medicine to
medicine go
medicine to
medicine the
medicine doctor
medicine because
medicine he
medicine had
guns     chest
guns     pains
guns     what
guns     to
guns     do
guns     if
guns     you
guns     shoot
guns     somebody
```

This corresponds to breaking up the text (rather imperfectly) into three blocks: "`when are the yankees planning on activating melido perez one of my friends`" with topic `baseball`; "`has had to go to the doctor because he had`" with topic `medicine`; and "`chest pains what to do if you shoot somebody`" with topic `guns`.

Data for this part of the problem is in the file `topics.data`.

Second-order Markov models (extra credit)
For extra credit, you can redo the entire assignment using a second-order Markov model. In such a model, the next state depends not only on the current state, but also on the last state. Thus, *X*[*t*+*1*] depends both on *X*[*t*] and on *X*[*t*-*1*]. (However, the evidence *E*[*t*] still depends only on the current state *X*[*t*].) These models are described in your textbook. You will need to create a separate code base with a different representation of an HMM and a different Viterbi algorithm. Your implementation should be fast enough for us to test on the kind of data provided in a reasonable amount of time. For this optional part of the assignment, you should also run your second-order model on the robot, typos, and topic datasets.

The extra credit portion is worth 15 points.

What your code should do

We will test your code by running

>>python assignment5.py –p <problem number> -o <hmm order>
where <problem number> is 1, 2, or 3 to mean toy robot, typo correction, or topic change respectively. Your code needs to run all functionality associated with that problem.

The second argument <hmm order> expects either a 1 or 2 to represent first- or second-order HMM. Even if you are not implementing the extra-credit portion of the assignment, you still need to handle this argument. If we type –o 2 and this functionality is not implemented, your code should output "Functionality not implemented". We will type –o 1 to run the first-order HMM.

Your program should output, in a nicely formatted and readable way, the conditional probability tables generated from the data and the most likely state sequence generated by your algorithm.

Note: If your code generates an error when we run it you will receive a 40%. If you can't get all functionality implemented, you are better off displaying a message "functionality not implemented" than submitting broken code.