CSCI 3202 Introduction to Artificial Intelligence
Instructor: Hoenigman
Assignment 8
Part I
Due Friday, November 13 by 4pm.
Part II
Due Friday, November 20 by 4pm.

For this assignment, submit your code to Moodle only. You do not need to have your code on github.

Hidden Markov Models

In this assignment, you will build a Hidden Markov Model from a data set to detect typos in text. The main components of the assignment are the following:

Part I

1. Implement a method to parse the data file and build an HMM from the data.

Part II

2. Implement the Viterbi algorithm for finding the most likely sequence of states through the HMM, given "evidence"; and
3. Run your code on a separate dataset and explore its performance.

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 ($P(E[t]=o \mid X[t]=s)$. These are the emission probabilities.
2. From each state s, the probability of traversing to every other state s' in one time step ($P(X[t+1]=s' \mid X[t]=s)$). These are the transition probabilities.
3. A distribution over the start state ($P(X[0])$). This is the initial state distribution.

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. When you implement the Viterbi algorithm in Part II, you will need to include the probability of making a transition from this dummy start state to each of the other states.

Generate the conditional probability tables

To generate the transition and emission probability tables, use the data file called *typos20.data* on Moodle that contain sequences of state-output pairs, i.e., sequences of the form:

$x[1]$ $e[1]$
$x[2]$ $e[2]$
.
.
.
$x[n]$ $e[n]$.

In this data, the *x[i]* is the state and the *e[i]* is the observation at time i.

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.

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.

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).

Smooth the estimates

When you generate the conditional probabilities, you will 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, we don't have enough data 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:

$$\frac{1 + number\ of\ heads}{2 + number\ of\ flips}$$

We add 1 to each output that could be observed in the numerator, and add the total number of states to the denominator.

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.

Part II

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 the 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.

There is another file on Moodle called *typos20Test.data* that contains 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.

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%.

What your code should do

We will test your code by running

>>python assignment8.py >>someOutputFile

where *someOutputFile* will contain everything that your program prints out. We will inspect your program output by looking at that file. For Part I of the assignment, we will be inspecting your conditional probability tables. Those tables should be formatted as:

Transition Probabilities:

X[t+1] X[t]

x[t+1] x[t],

For all states X found in the data

Emission Probabilities:

E[t] | X[t]

e[t],

For all evidence E found for states X.

For Part II, we will be inspecting the state sequence and error rate of your Viterbi output.

Format your output as:

State sequence:

for all t:

  print x[t]

Error rate: 1 - (correct states / t)