CIS 521 Final Project

November 29, 2017

In this assignment you will use the forward and backward algorithms to compute the probability of an observation sequence given a hidden Markov model (HMM) and then to re-estimate the parameters of the HMM to maximize the probability. The terminology used throughout this assignment follows that of the lecture slides on HMMs except that it uses $t = \{0, \ldots, T-1\}$ instead of $t = \{1, \ldots, T\}$.

A skeleton file `homework11.py` containing empty definitions for each question has been provided. Since portions of this assignment will be graded automatically, none of the names or function signatures in this file should be modified. However, you are free to introduce additional variables or functions if needed.

You may import definitions from any standard Python library and are encouraged to do so in case you find yourself reinventing the wheel.

You will find that in addition to a problem specification, most programming questions also include a pair of examples from the Python interpreter. These are meant to illustrate typical use cases, and should not be taken as comprehensive test suites.

You are strongly encouraged to follow the Python style guidelines set forth in PEP 8, which was written in part by the creator of Python. However, your code will not be graded for style.

Once you have completed the assignment, you should submit your file on Eniac using the following turnin command, where the flags -c and -p stand for "course" and "project", respectively.

```
turnin -c cis521 -p hw11 homework11.py
```

You may submit as many times as you would like before the deadline, but only the last submission will be saved. To view a detailed listing of the contents of your most recent submission, you can use the following command, where the flag -v stands for "verbose".

```
turnin -c cis521 -p hw11 -v
```

1 Hidden Markov Model [95 points]

1.1 [5 points]

Write a function `load_corpus(path)` to load the corpus and return it as a space-separated string. All letters should be converted to lowercase and any characters other than letters or space should be removed. Punctuation within words should be removed and repeated spaces should be replaced with single spaces.

```
>>> s = load_corpus("Brown_sample.txt")
>>> len(s)
2824
>>> s[1208:1228]
"to the best interest"
```

1.2 [5 points]

Many computations in this assignment use values derived from repeated multiplications of probabilities. These probabilities are typically less than 1 so repeated multiplication will eventually result in underflow. This can be prevented by instead using the logarithms of the probabilities, with multiplication becoming addition and division becoming subtraction. However there is no logarithm equivalent of summation so the values must be exponentiated before being summed. This can be done without underflow by use of something called the log-sum-exp trick which uses the following identity:

$$log \sum_{n=1}^{N} exp(x_n) = a + log \sum_{n=1}^{N} exp(x_n - a)$$

Setting a to be the max over all x_n will increase the values so that they can be summed without underflow.

Write a function `load_parameters(path)` that loads the probability vector at the given path, converts the probabilities to log probabilities using base e, and returns it. The probability vector is a tuple of the form (initial probabilities, transition probabilities, emission probabilities).

Initial probabilties: dictionary of the form {state$_i$: probability that state$_i$ is a start state}

Transition probabilties: dictionary of the form {state$_i$: {state$_j$: probability of transitioning from state$_i$ to state$_j$}}

Emission probabilities: dictionary of the form {state$_i$: {symbol$_k$: probability of being in state$_i$ and emitting symbol$_k$}}

```
>>> p = load_parameters(
...                "simple.pickle")
>>> p[1][1]
{1: -0.6931471805599453,
 2: -0.6931471805599453}
```

```
>>> p = load_parameters(
...                "prob_vector.pickle")
>>> p[1][1]
{1: -0.9394332049935226,
 2: -1.663989250166688,
 3: -2.563281364593492,
 4: -1.070849586518945}
```

Then, in the HMM class, write an initialization method `__init__(self, probabilities)` that uses a vector of log probabilities to initialize the internal parameters of the hidden Markov model and a method `get_parameters(self)` which returns the parameters of the HMM in the same format as the probability vectors, but as probabilities between 0 and 1, rather than as log probabilities.

```
>>> p = load_parameters(
...                "simple.pickle")
>>> h = HMM(p)
>>> n = h.get_parameters()
>>> n[1][1]
{1: 0.5, 2: 0.5}
```

```
>>> p = load_parameters(
...                "prob_vector.pickle")
>>> h = HMM(p)
>>> n = h.get_parameters()
>>> n[1][1]
{1: 0.3908493040227309,
 2: 0.18938197908059773,
 3: 0.0770514911339938,
 4: 0.3427172257626776}
```

1.3 [10 points]

Write a method `forward(self, sequence)` which takes a sequence and returns a list of dictionaries. Each dictionary is of the form $\{state_i : \alpha_t(i)\}$, $1 \leq i \leq N$, $0 \leq t \leq T-1$, where N is the number of states and T is the length of the input sequence. Recall from the lecture slides that $\alpha_t(i)$ is the probability that the partial sequence up to time t has been seen and the current state is i.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> f = h.forward(s)
>>> f[10]
{1: -22.0981588201684,
 2: -22.0981588201684}
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> f = h.forward(s)
>>> f[1400]
{1: -4570.10024680558,
 2: -4569.896509256886,
 3: -4569.956231590213,
 4: -4569.542222483007}
```

1.4 [10 points]

Write a method `forward_probability(self, alpha)` that returns the forward probability of this HMM for that sequence.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> h.forward_probability(h.forward(s))
-36.972292832050975
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> h.forward_probability(h.forward(s))
-9201.34957430782
```

1.5 [10 points]

Write a method `backward(self, sequence)` which takes a sequence and returns a list of dictionaries. Each dictionary is of the form $\{state_i : \beta_t(i)\}$, $1 \leq i \leq N$, $0 \leq t \leq T-1$, where N is the number of states and T is the length of the input sequence. Recall from the lecture slides that $\beta_t(i)$ is probability that the partial sequence from $t+1$ to $T-1$ will be seen given that the current state is i.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> b = h.backward(s)
>>> b[9]
{1: -17.513191341497826,
 2: -17.513191341497826}
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> b = h.backward(s)
>>> b[1424]
{1: -4553.117090965298,
 2: -4553.249309905892,
 3: -4553.085375790753,
 4: -4553.140279571696}
```

1.6 [10 points]

Write a method `backward_probability(self, beta, sequence)` that returns the backward probability of this HMM for that sequence. This should approximately match the forward probability, but may not be an exact match due to floating point inaccuracies.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> h.backward_probability(
...              h.backward(s),s)
-36.97229283205097
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> h.backward_probability(
...              h.backward(s),s)
-9201.349574307758
```

1.7 [10 points]

Write a method `xi_matrix(self, t, sequence, alpha, beta)` which returns a dictionary of the form $\{\text{state}_i\text{: }\{\text{state}_j\text{: }\xi_t(i,j)\}\}$, where $\xi_t(i,j)$ is the probability of being in state i at time t and going to state j given the current parameters of the HMM.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> h.xi_matrix(5,s,h.forward(s),
...              h.backward(s))[2]
{1: -1.3862943611198943,
 2: -1.3862943611198943}
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> h.xi_matrix(500,s,h.forward(s),
...              h.backward(s))[1]
{1: -2.5704875729134073,
 2: -3.418873166145204,
 3: -3.8974061320204783,
 4: -2.080487933135373}
```

1.8 [25 points]

Write a method `forward_backward(self, sequence)` that re-estimates the parameters of the HMM and returns them in the same format as `load_parameters(path)`, that is, a tuple of log probabilities of the form (initial probabilities, transition probabilities, emission probabilities). The formulas for re-estimation can be found in the lecture slides on HMMs, but they will need to be adjusted to account for changed range of t.

Re-estimation of initial probabilities:

$\hat{\pi}_i = \textit{expected number of times in state } s_i \textit{ at time } 0$

Re-estimation of transition probabilities:

$$\hat{a}_{i,j} = \frac{\textit{expected number of transitions from state } s_i \textit{ to state } s_j}{\textit{expected number of transitions from } s_i}$$

Re-estimation of emission probabilities:

$$\hat{b}_i(k) = \frac{\textit{expected number of times in state } s_i \textit{ and observe symbol } v_k}{\textit{expected number of times in state } s_i}$$

The re-restimations will require computing $\gamma_t(i)$ which is the probability of being in state s_i at time t and can be computed in two ways:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \qquad \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)}$$

One important thing to note is that re-estimation of the transition probabilities $\hat{a}_{i,j}$ is done over the range $t = \{0, \ldots, T-2\}$ while re-estimation of the emission probabilities $\hat{b}_i(k)$ is done over the range $t = \{0, \ldots, T-1\}$, where T is the length of the input sequence. Both of these re-estimations use $\gamma_t(i)$, but when $t = T-1$, $\gamma_t(i)$ cannot be computed using the ξ values and must be computed using the other method.

```
>>> s = "the cat ate the rat"
>>> p = load_parameters(
...              "simple.pickle")
>>> h = HMM(p)
>>> p2 = h.forward_backward(s)
>>> h2 = HMM(p2)
>>> h2.forward_probability(
...              h2.forward(s))
-34.37400550438377
```

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters(
...              "prob_vector.pickle")
>>> h = HMM(p)
>>> p2 = h.forward_backward(s)
>>> h2 = HMM(p2)
>>> h2.forward_probability(
...              h2.forward(s))
-8070.961574771892
```

1.9 [10 points]

Write a method `update(self,sequence,cutoff_threshold)` that repeatedly runs `forward_backward(sequence)` to iteratively re-estimate the HMM probabilities. The re-estimation should stop once the increase in probability falls below the cutoff threshold, otherwise it will run much longer as it makes incremental improvements to the probability.

```
>>> s = load_corpus("Brown_sample.txt")
>>> p = load_parameters("prob_vector.pickle")
>>> h = HMM(p)
>>> h.update(s, 1)
>>> h.forward_probability(h.forward(s))
-7383.3361451482
```

2 Feedback [5 points]

2.1 [1 points]

Approximately how long did you spend on this assignment?

2.2 [2 points]

Which aspects of this assignment did you find most challenging? Were there any significant stumbling blocks?

2.3 [2 points]

Which aspects of this assignment did you like? Is there anything you would have changed?