HOMEWORK 7
HIDDEN MARKOV MODELS[1]

10-301 / 10-601 INTRODUCTION TO MACHINE LEARNING (SPRING 2020)
https://mlcourse.org

OUT: April 02, 2020
DUE: April 10, 2020 11:59 PM
TAs: Mike Chen, Eu Jing Chua, Filipp Shelobolin, Ayushi Sood

Summary In this assignment you will implement a new named entity recognition system using Hidden Markov Models. You will begin by going through some multiple choice warm-up problems to build your intuition for these models and then use that intuition to build your own HMM models.

START HERE: Instructions

- **Collaboration Policy**: Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the collaboration policy on the website for more information: http://www.cs.cmu.edu/~mgormley/courses/10601/about.html#7-academic-integrity-policies
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/about.html#late-homework-policy
- **Submitting your work:** You will use Gradescope to submit answers to all questions and your code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as derivations, proofs, or plots we will be using Gradescope (https://gradescope.com/). Submissions can be handwritten, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Upon submission, label each question using the template provided. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed on a separate page.
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.6.9, Octave 4.2.2, OpenJDK 11.0.5, g++ 7.4.0) and versions of permitted libraries (e.g. `numpy` 1.17.0 and `scipy` 1.4.1) match those used on Gradescope. (Octave users: Please make sure you do not use any Matlab-specific libraries in your code that might make it fail against our

[1] Compiled on Monday 13th April, 2020 at 03:46

tests.) You have a **total of 10 Gradescope programming submissions.** Use them wisely. In order to not waste code submissions, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before any Gradescope coding submission.

- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on Piazza.

Linear Algebra Libraries When implementing machine learning algorithms, it is often convenient to have a linear algebra library at your disposal. In this assignment, Java users may use EJML[a] and C++ users Eigen[b]. Details below. (As usual, Python users have numpy; Octave users have built-in matrix support.)

Java EJML is a pure Java linear algebra package with three interfaces. We strongly recommend using the SimpleMatrix interface. Gradescope will use EJML version 0.38. The command line arguments above demonstrate how we will call you code. The classpath inclusion `-cp "./lib/ejml-v0.38-libs/*:./"` will ensure that all the EJML jars are on the classpath as well as your code.

C++ Eigen is a header-only library, so there is no linking to worry about—just `#include` whatever components you need. Gradescope will use Eigen version 3.3.7. The command line arguments above demonstrate how we will call you code. The argument `-I./lib` will include the `lib/Eigen` subdirectory, which contains all the headers.

Do **not** include EJML or Eigen in your Gradescope submission tar; the autograder will ensure that they are in place.

[a] https://ejml.org

[b] http://eigen.tuxfamily.org/

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

● Matt Gormley

○ Marie Curie

○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

● Matt Gormley

○ Marie Curie
~~●~~ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

■ Stephen Hawking

■ Albert Einstein

■ Isaac Newton

□ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are scientists?

■ Stephen Hawking

■ Albert Einstein

■ Isaac Newton
~~■~~ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-601

10-~~7~~601

1 Written Questions [20 pts]

1.1 Multiple Choice [10 pts]

In this section we will test your understanding of several aspects of HMMs. Shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions below.

1. (2 points) (**Select all that apply**) Which of the following are true under the (first-order) Markov assumption in an HMM:

   □ The states are independent

   □ The observations are independent

   □ $y_t \perp y_{t-1} | y_{t-2}$

   ■ $y_t \perp y_{t-2} | y_{t-1}$

   □ None of the above

2. (2 points) (**Select all that apply**) Which of the following independence assumptions hold in an HMM:

   ■ The current observation x_t is conditionally independent of all other observations given the current state y_t

   ■ The current observation x_t is conditionally independent of all other states given the current state y_t

   ■ The current state y_t is conditionally independent of all states given the previous state y_{t-1}

   □ The current observation x_t is conditionally independent of x_{t-2} given the previous observation x_{t-1}.

   □ None of the above

   In the remaining questions you will always see two quantities and decide what is the strongest relation between them. (? means it's not possible to assign any true relation). As such there is **only one correct answer**. α **and** β **in the following questions and questions in section 1.2 follow the definition in section 2.**

3. (2 points) (**Select one**) What is the relation between $\sum_{i=0}^{N-1}(\alpha_5(i) * \beta_5(i))$ and $P(\mathbf{x})$? Select only the **strongest** relation that necessarily holds.

   ● $=$

   ○ $>$

   ○ $<$

   ○ $\leq$

   ○ $\geq$

   ○ ?

4. (2 points) (**Select one**) What is the relation between $P(y_4 = s_1, y_5 = s_2, \mathbf{x})$ and $\alpha_4(s_1) \cdot \beta_5(s_2)$? Select only the **strongest** relation that necessarily holds.

   ○ $=$

   ○ $>$

   ○ $<$

   ● $\leq$

   ○ $\geq$

   ○ ?

5. (2 points) (**Select one**) What is the relation between $\alpha_5(i)$ and $\beta_5(i)$? Select only the **strongest** relation that necessarily holds.

   ○ $=$

   ○ $>$

   ○ $<$

   ○ $\leq$

   ○ $\geq$

   ● ?

1.2 Viterbi Decoding

Suppose we have a set of sequence consisting of T observed states, $x_1, ..., x_T$, where each $x_t \in \{X, Y, Z\}$. Each observed state is associated with a hidden state $y_t \in \{A, B\}$. In the Viterbi algorithm we seek to find the most probable hidden state sequence $y_1, ..., y_T$ given the observation $x_1, ..., x_T$. To compute such sequence, recall from lecture the following dynamic programming algorithm:

1. Initialize $v_1(s_j) = \pi_{s_j} b_{s_j x_1}$ and $p_1(s_j) = s_j$
2. For $t > 1$, we have

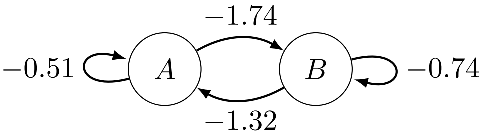
$$v_t(s_j) = \max_{k \in \{1,...,J\}} b_{s_j x_t} a_{s_k s_j} v_{t-1}(s_k)$$
$$p_t(s_j) = \operatorname*{argmax}_{k \in \{1,...,J\}} b_{s_j x_t} a_{s_k s_j} v_{t-1}(s_k)$$

where $v_t(s_j)$ is the highest product of all the probabilities taken through path $y_1, ..., y_{t-1}$ that ends with y_t at state s_j, and $p_t(s_j)$ is the backpointers that store the path through hidden states that give us the highest product.

We can obtain the most probable sequence by backtracing through the backpointers as follows:

1. $\hat{y}_T = \text{argmax}_{k \in \{1,...,J\}} v_T(s_k)$.
2. For $t = T, ..., 1$: $\hat{y}_{t-1} = p_t(\hat{y}_t)$
3. Return $\hat{y}_1, ..., \hat{y}_T$

For the following question, consider the Hidden Markov Model specified below. Working with probabilities in the logarithm scale, we have that $\log p(y_1 = A) = \log p(y_1 = B) = -1$, and the state transition model and emission probability tables are given as follows.



x_t	$\log p(x_t\|y_y = A)$	$\log p(x_t\|y_t = B)$
X	-1	-1.74
Y	-1.32	-1
Z	-3.32	-2.32

We observed $x_1 = X$ and $x_2 = Y$. (Note that taking the maximum of log probabilities will give you the same result as taking the maximum of probabilities as log is a monotonically increasing function.)

1. (2 points) Compute $\log v_1(s_j = A)$ and $\log v_1(s_j = B)$. If your answers involves decimal numbers, please round your answer to **TWO** decimal places.

   $\log v_1(s_j = A)$:

   -2

   $\log v_1(s_j = B)$:

   -2.74

2. (2 points) (**Select one**) Which of the following is the most likely sequence of hidden states?

   ● $y_1 = A$, $y_2 = A$

   ○ $y_1 = A$, $y_2 = B$

   ○ $y_1 = B$, $y_2 = A$

   ○ $y_1 = B$, $y_2 = B$

   ○ Don't have enough information.

1.3 Warm-up Exercise: Forward-Backward Algorithm [4 pts]

To help you prepare to implement the HMM forward-backward algorithm (see Section 2.3 for a detailed explanation), we have provided a small example for you to work through by hand. This toy data set consists of a training set of three sequences with three unique words and two tags and a test set with a single sequence composed of the same unique words used in the training set. Before going through this example, please carefully read the algorithm description in Sections 2.2 and 2.3. **Note:** pseudo count used in section 2 is also used here.

Training set:

```
you_B eat_A fish_B
you_B fish_B eat_A
```

```
eat_A fish_B
```

Where the training word sequences are:

$$x = \begin{bmatrix} \text{you} & \text{eat} & \text{fish} \\ \text{you} & \text{fish} & \text{eat} \\ \text{eat} & \text{fish} & \end{bmatrix}$$

And the corresponding tags are:

$$y = \begin{bmatrix} \text{B} & \text{A} & \text{B} \\ \text{B} & \text{B} & \text{A} \\ \text{A} & \text{B} & \end{bmatrix}$$

Test set:

```
fish eat you
```

or

$$x = \begin{bmatrix} \text{fish} & \text{eat} & \text{you} \end{bmatrix}$$

The following four questions are meant to encourage you to work through the forward backward algorithm by hand using this test example. Feel free to use a calculator, being careful to carry enough significant figures through your computations to avoid rounding errors. For each question below, please report the requested value in the text box next to the question (these boxes are only visible in the template document). When a number is requested, only write the number in the box. When a word/tag is requested, only write that word or tag. **DO NOT** include explanations or derivation work in the text box. Points will be deducted if anything else is included in the box.

1. (1 point) Compute $\alpha_2(A)$, the α value associated with the tag "A" for the second word in the test sequence. Please round your answer to **THREE** decimal places.

   0.131

2. (1 point) Compute $\beta_2(B)$, the β value associated with the tag "B" for the second word in the test sequence. Please round your answer to **THREE** decimal places.

   0.250

3. (1 point) Predict the tag for the third word in the test sequence.

   B

4. (1 point) Compute the log-likelihood for the entire test sequence, "`fish eat you`". Please round your answer to **THREE** decimal places.

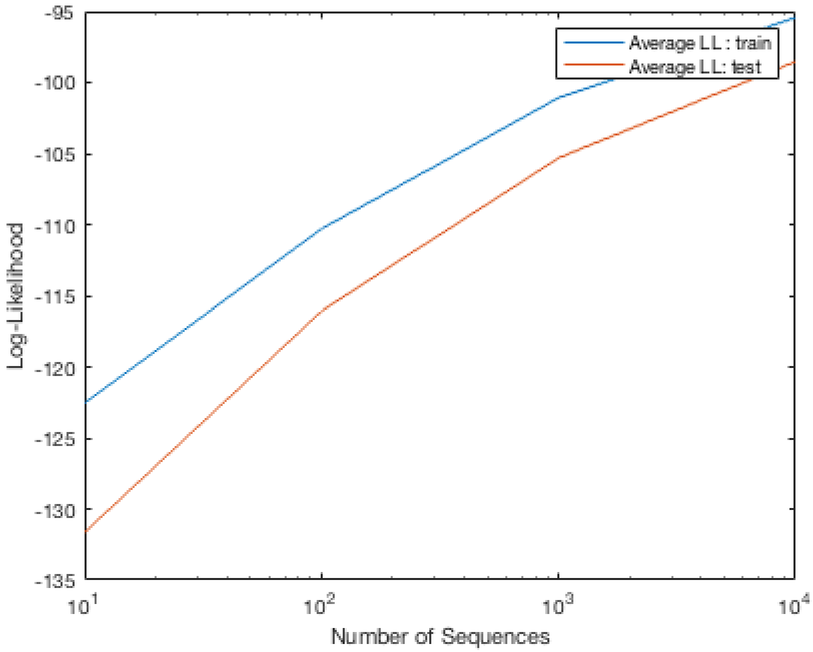
   -3.044

1.4 Empirical Questions [6 pts]

Return to these questions after implementing your `learnhmm.{py|java|cpp|m}` and `forwardbackward.{py|java|cpp|m}` functions.

Using the fulldata set **trainwords.txt** in the handout using your implementation of `learnhmm.{py|java|cpp|m}` to learn parameters for an hmm model using the first 10, 100, 1000, and 10000 sequences in the file. Use these learned parameters perform prediction on the **trainwords.txt** and the **testwords.txt** files using your `forwardbackward.{py|java|cpp|m}`. Construct a plot with number of sequences used for training on the x-axis (log-scale) and average log likelihood across all sequences from the **trainwords.txt** or the **testwords.txt** on the y-axis (see Section 2.3 for details on computing the log data likelihood for a sequence). Each table entry is worth 0.5 points. Write the resulting log likelihood values in the table in the template. Include your plot in the large box in the template (2 points). To receive credit for your plot, you must submit a computer generated plot. **DO NOT** hand draw your plot.

#sequences	Train average log-likelihood	Test average log-likelihood
10	-122.542	-131.649
100	-110.314	-116.0916
1000	-101.072	-105.3082
10000	-95.437	-98.528


Average LL : train
Average LL: test
Log-Likelihood
-95
-100
-105
-110
-115
-120
-125
-130
-135
10^1
10^2
10^3
10^4
Number of Sequences

1.5 Collaboration Policy

After you have completed all other components of this assignment, report your answers to the collaboration policy questions detailed in the Academic Integrity Policies found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? Is so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? Is so, include full details.
3. Did you find or come across code that implements any part of this assignment ? If so, include full details.

1) No
2) No
3) No

2 Programming [80 pts]

2.1 The Tasks and Data Sets

The handout file for this assignments contains several files that you will use in the homework. The contents and formatting of each of these files is explained below.

1. **trainwords.txt** This file contains labeled text data that you will use in training your model in the **Learning** problem. Specifically the text contains one sentence per line that has already been preprocessed, cleaned and tokenized. You should treat every line as a separate sequence and assume that it has the following format:

   ```
   <Word0>_<Tag0> <Word1>_<Tag1> ...  <WordN>_<TagN>
   ```

   where every `<WordK>_<TagK>` unit token is separated by white space.

2. **testwords.txt**: This file contains labeled data that you will use to evaluate your model in the **Experiments** section. This file contains the gold standard labels. This file has the same format as **trainwords.txt**.

3. **index_to_word.txt** and **index_to_tag.txt**: These files contain a list of all words or tags that appear in the data set. In your functions, you will convert the string representation of words or tags to indices corresponding to the location of the word or tag in these files. For example, if Austria is on line 729 of **index_to_word.txt**, then all appearances of Austria in the data sets should be converted to the index 729. This index will also correspond to locations in the parameter matrices. For example, the word Austria corresponds to the parameters in column 729 of the matrix stored in **hmmemit.txt**. This will be useful for your forward-backward algorithm implementation (see Section 2.3).

4. **toytrain.txt**, **toytest.txt**, **toy_index_to_word.txt**, and **toy_index_to_tag.txt**: These files are analogous to **trainwords.txt**, **testwords.txt**, **index_to_word.txt**, and **index_to_tag.txt** and are used to compare your implementation to your hand calculations in Section 1.3.

5. **predicttest.txt** The **predicttest.txt** file contains labeled data that you will use to debug your implementation. The labels in this file are not gold standard but are generated by running our decoder using the features from **trainwords.txt**. This file has the same format as **trainwords.txt**.

6. **metrics.txt** The **metrics.txt** file contains the metrics you will compute for the dataset. For this assignment, you need to compute the average log likelihood and your prediction accuracy on the test data. Note that in Named Entity Recognition, F-1 score is a more common metric to evaluate the model performance, here you only need to report your accuracy for tag prediction of each word.

   Sample metrics file for the toy dataset

7. **hmmtrans.txt, hmmemit.txt and hmmprior.txt**: These files contain pre-trained model parameters of an HMM that you will use in testing your implementation of the **Learning** and **Evaluation and Decoding** problems. The format of the first two files are analogous and is as follows. Every line in these files consists of a conditional probability distribution. In the case of transition probabilities, this distribution corresponds to the probability of transitioning into another state, given a current state. Similarly, in the case of emission probabilities, this distribution corresponds to the probability of emitting a particular symbol, given a current state. For example, every line in **hmmtrans.txt** has the following format:

   hmmtrans.txt:

   ```
   <ProbS1S1> ...  <ProbS1SN>
   ```

```
<ProbS2S1> ...  <ProbS2SN>...
```

and every line in **hmmemit.txt** has the following format:

hmmemit.txt:

```
<ProbS1Word1> ...  <ProbS1WordN>
<ProbS2Word1> ...  <ProbS2WordN>...
```

In both cases, elements in the same row are separated by white space. Each row corresponds to a line of text (using `\n` to create new lines).

The format of **hmmprior.txt** is similarly defined except that it only contains a single probability distribution over starting states. Therefore each row only has a single element. Therefore **hmmprior.txt** has the following format:

hmmprior.txt:

```
<ProbS1>
<ProbS2>...
```

Note that the data provided to you is to help in developing your implementation of the HMM algorithms. Your code will be tested on Autolab using different data with different HMM parameters, likely coming from a different domain although the format will be identical.

2.2 Learning

Your first task is to implement an algorithm to learn the hidden Markov model parameters needed to apply the forward backward algorithm (See Section 2.3). There are three sets of parameters that you will need to estimate: the initialization probabilities $\boldsymbol{\pi}$, the transition probabilities $\mathbf{A}$, and the emission probabilities $\mathbf{B}$. For this assignment, we model each of these probabilities using a multinomial distribution with parameters $\pi_j = P(y_1 = j)$, $a_{jk} = P(y_t = k | y_{t-1} = j)$, and $b_{jk} = P(x_t = k | y_t = j)$. These can be estimated using maximum likelihood, which results in the following parameter estimators:

1. $P(y_1 = j) = \pi_j = \frac{N_\pi^j + 1}{\sum_{p=1}^J (N_\pi^p + 1)}$, where N_π^j equals the number of times state s_j is associated with the first word of a sentence in the training data set.
2. $P(y_t = k | y_{t-1} = j) = a_{jk} = \frac{N_A^{jk} + 1}{\sum_{p=1}^J (N_A^{jp} + 1)}$, where N_A^{jk} is the number of times state s_j is followed by state s_k in the training data set.
3. $P(x_t = k | y_t = j) = b_{jk} = \frac{N_B^{jk} + 1}{\sum_{p=1}^M (N_B^{jp} + 1)}$, where N_B^{jk} is the number of times that the state s_j is associated with the word k in the training data set.

Note that for each count, a "+1" is added to make a **pseudocount**. This is slightly different from pure maximum likelihood estimation, but it is useful in improving performance when evaluating unseen cases during evaluation of your test set.

You should implement a function that reads in the training data set (**trainwords.txt**), and then estimates π, A, and B using the above maximum likelihood solutions.

Your outputs should be in the same format as **hmmprior.txt**, **hmmtrans.txt**, and **hmmemit.txt** (including the same number of decimal places to ensure there are no rounding errors during prediction). The autograder will use the following commands to call your function:

```
For Python: $ python learnhmm.py [args...]
For Java:   $ javac -cp "./lib/ejml-v0.33-libs/*:./" learnhmm.java;
            $ java -cp "./lib/ejml-v0.33-libs/*:./" learnhmm [args...]
For C++:    $ g++ -g -std=c++11 -I./lib learnhmm.cpp; ./a.out [args...]
For Octave: $ octave -qH learnhmm.m [args...]
```

Where above `[args...]` is a placeholder for six command-line arguments:`<train_input>` `<index_to_word>` `<index_to_tag>` `<hmmprior>` `<hmmemit>` `<hmmtrans>`. These arguments are described in detail below:

1. `<train_input>`: path to the training input `.txt` file (see Section 2.1)
2. `<index_to_word>`: path to the `.txt` that specifies the dictionary mapping from words to indices. The tags are ordered by index, with the first word having index of 1, the second word having index of 2, etc.
3. `<index_to_tag>`: path to the `.txt` that specifies the dictionary mapping from tags to indices. The tags are ordered by index, with the first tag having index of 1, the second tag having index of 2, etc.
4. `<hmmprior>`: path to output `.txt` file to which the estimated prior ($\boldsymbol{\pi}$) will be written. The file output to this path should be in the same format as the handout `hmmprior.txt` (see Section 2.1).
5. `<hmmemit>`: path to output `.txt` file to which the emission probabilities ($\mathbf{B}$) will be written. The file output to this path should be in the same format as the handout `hmmemit.txt` (see Section 2.1)
6. `<hmmtrans>`: path to output `.txt` file to which the transition probabilities ($\mathbf{A}$) will be written. The file output to this path should be in the same format as the handout `hmmtrans.txt` (see Section 2.1)..

2.3 Evaluation and Decoding

2.3.1 Forward Backward Algorithm and Minimal Bayes Risk Decoding

Your next task is to implement the forward-backward algorithm. Suppose we have a set of sequence consisting of T words, $x_1, ..., x_T$. Each word is associated with a label $y_t \in \{1, ..., J\}$. In the forward-backward algorithm we seek to approximate $P(y_t|x_1, ..., x_T)$ up to a multiplication constant. This is done by first breaking $P(y_t|x_1, ..., x_T)$ into a "forward" component and a "backward" component as follows:

$$\begin{aligned} P(y_t = s_j|x_1, ..., x_T) &\propto P(y_t = s_j, x_{t+1}, ..., x_T|x_1, ..., x_t) \\ &\propto P(y_t = s_j|x_1, ..., x_t)P(x_{t+1}, ..., x_T|y_t = s_j, x_1, ..., x_t) \\ &\propto P(y_t = s_j|x_1, ..., x_t)P(x_{t+1}, ..., x_T|y_t = s_j) \\ &\propto P(y_t = s_j, x_1, ..., x_t)P(x_{t+1}, ..., x_T|y_t = s_j) \end{aligned}$$

where $P(y_t = s_j|x_1, ..., x_t)$ is computed by passing forward recursively through the model and $P(x_{t+1}, ..., x_T|y_t = j)$ is computed by passing recursively backwards through the model.

Forward Algorithm

Define $\alpha_t(j) = P(y_t = j, x_{1:t})$. We can rearrange our definition of $\alpha_t(j)$ as follows:

$$\begin{aligned}
\alpha_t(j) =& P(y_t = s_j, x_{1:t}) \\
=& \sum_k P(y_t = s_j, y_{t-1} = s_k, x_{1:t}) \\
=& \sum_k P(y_{t-1} = s_k, x_{1:t} | y_t = s_j) P(y_t = s_j) \\
=& \sum_k P(x_t | y_t = s_j) P(y_{t-1} = s_k, x_{1:t-1} | y_t = s_j) P(y_t = s_j) \\
=& P(x_t | y_t = s_j) \sum_k P(y_t = s_j, y_{t-1} = s_k, x_{1:t-1}) \\
=& P(x_t | y_t = s_j) \sum_k P(y_t = s_j, x_{1:t-1} | y_{t-1} = s_k) P(y_{t-1} = s_k) \\
=& P(x_t | y_t = s_j) \sum_k P(y_t = s_j | y_{t-1} = s_k) P(x_{1:t-1} | y_{t-1} = s_k) P(y_{t-1} = s_k) \\
=& P(x_t | y_t = s_j) \sum_k P(y_t = s_j | y_{t-1} = s_k) P(y_{t-1} = s_k, x_{1:t-1}) \\
=& b_{jx_t} \sum_k a_{kj} \alpha_{t-1}(k) \qquad (2.1)
\end{aligned}$$

Using this definition, the α's can be computed using the following recursive procedure:

1. $\alpha_1(j) = \pi_j b_{jx_1}$.
2. For $t > 1$, $\alpha_t(j) = b_{jx_t} \sum_{k=1}^{J} \alpha_{t-1}(k) a_{kj}$

Backward Algorithm Define $\beta_t(j) = P(x_{t+1}, ..., x_T | y_t = s_j)$. We can rearrange our definition of $\beta_t(j)$ as follows:

$$\begin{aligned}
\beta_t(j) &= P(x_{t+1}, ..., x_T | y_t = s_j) \\
&= \sum_{k=1}^{J} P(y_{t+1} = s_k, x_{t+1}, ..., x_T | y_t = s_j) \\
&= \sum_{k=1}^{J} P(x_{t+1}, ..., x_T | y_t = s_j, y_{t+1} = s_k) P(y_{t+1} = s_k | y_t = s_j) \\
&= \sum_{k=1}^{J} P(x_{t+1} | y_{t+1} = s_k) P(x_{t+2}, ..., x_T | y_{t+1} = s_k) P(y_{t+1} = s_k | y_t = s_j) \\
&= \sum_{k=1}^{J} b_{kx_{t+1}} \beta_{t+1}(k) a_{jk} \qquad (2.2)
\end{aligned}$$

Just like the α's, the β's can also be computed using the following backward recursive procedure:

1. $\beta_T(j) = 1$ (All states could be ending states)

2. For $1 \leq t < T$, $\beta_t(j) = \sum_{k=1}^{J} b_{kx_{t+1}} \beta_{t+1}(k) a_{jk}$ (Generate x_{t+1} from any state)

Forward-Backward Algorithm As stated above, the goal of the Forward-Backward algorithm is to compute $P(y_t = s_j | x_1, ..., x_T)$. This can be done using the following equation:

$$P(y_t = s_j | x_1, ..., x_T) \propto P(y_t = s_j, x_1, ..., x_t) P(x_{t+1}, ..., x_T | y_t = s_j)$$

After running your forward and backward passes through the sequence, you are now ready to estimate the conditional probabilities as:

$$P(y_t | x_1, ..., x_T) \propto \alpha_t \circ \beta_t$$

where $\circ$ is the element-wise product.

Minimum Bayes Risk Prediction We will assign tags using the minimum Bayes risk predictor, defined for this problem as follows:

$$\hat{y}_t = \underset{j \in \{1,...,J\}}{\text{argmax}}\ P(y_t = s_j | x_1, ..., x_T)$$

To resolve ties, select the tag that appears earlier in the `<index_to_tag>` input file.

Computing the Log Likelihood of a Sequence When we compute the log likelihood of a sequence, we are interested in the computing the quantity $P(x_1, ..., x_T)$. We can rewrite this in terms of values we have already computed in the forward-backward algorithm as follows:

$$\begin{aligned} \log P(x_1, ..., x_T) &= \log \Big(\sum_j P(x_1, ..., x_T, y_T = s_j) \Big) \\ &= \log \Big(\sum_j \alpha_T(j) \Big) \end{aligned}$$

2.3.2 Implementation Details

You should now implement your forward-backward algorithm as a program, `forwardbackward.{py|java|cpp|m}`. The program will read in test data and the parameter files produced by `learnhmm.{py|java|cpp|m}`. The autograder will use the following commands to call your function:

For Python:
```
$ python forwardbackward.py [args...]
```
For Java:
```
$ javac -cp "./lib/ejml-v0.33-libs/*:./" forwardbackward.java;
java -cp "./lib/ejml-v0.33-libs/*:./" forwardbackward [args...]
```
For C++:
```
$ g++ -g -std=c++11 -I./lib forwardbackward.cpp; ./a.out [args...]
```
For Octave:
```
$ octave -qH forwardBackward.m [args...]
```

Where above `[args...]` is a placeholder for seven command-line arguments: `<test_input>` `<index_to_word>` `<index_to_tag>` `<hmmprior>` `<hmmemit>` `<hmmtrans>` `<predicted_file>` `<metric_file>`. These arguments are described in detail below:

1. `<test_input>`: path to the test input `.txt` file that will be evaluated by your forward backward algorithm (see Section 2.1)

2. `<index_to_word>`: path to the `.txt` that specifies the dictionary mapping from words to indices. The tags are ordered by index, with the first word having index of 1, the second word having index of 2, etc. This is the same file as was described for `learnhmm.{py|java|cpp|m}`.
3. `<index_to_tag>`: path to the `.txt` that specifies the dictionary mapping from tags to indices. The tags are ordered by index, with the first tag having index of 1, the second tag having index of 2, etc. This is the same file as was described for `learnhmm.{py|java|cpp|m}`.
4. `<hmmprior>`: path to input `.txt` file which contains the estimated prior ($\boldsymbol{\pi}$).
5. `<hmmemit>`: path to input `.txt` file which contains the emission probabilities ($\mathbf{B}$).
6. `<hmmtrans>`: path to input `.txt` file which contains transition probabilities ($\mathbf{A}$).
7. `<predicted_file>`: path to the output `.txt` file to which the predicted tags will be written. The file should be in the same format as the `<test_input>` file.
8. `<metric_file>`: path to the output `.txt` file to which the metrics will be written.

Example command for python users:

```
$ python forwardbackward.py toydata/toytrain.txt \
toydata/toy_index_to_word.txt toydata/toy_index_to_tag.txt hmmprior.txt \
hmmemit.txt hmmtrans.txt predicted.txt metrics.txt
```

After running the command above, the `<predicted_file>` output should be:

```
you_B eat_A fish_B
you_B fish_B eat_A
eat_A fish_B
```

And the `<metric_file>` output should be:

```
Average Log-Likelihood: -2.776793
Accuracy: 1.0
```

Take care that your output has the exact same format as shown above. There should be a single space after the colon preceding the metric value (e.g. a space after `Average Log-Likelihood:`). Each line should be terminated by a Unix line ending `\n`.

2.3.3 Log-Space Arithmetic for Avoiding Underflow

Handling underflow properly is a critical step in implementing an HMM. And the most generalized way of handling numerical underflow due to products of small positive numbers (like probabilities) is to calculate everything in **log-space**, i.e., represent every quantity by their logarithm,

For this homework, using log-space trick starts with transforming Eq.(2.1) and Eq.(2.2) into logarithmic form (how to do that is straightforward and left as an exercise). Please use e as the base for logarithm calculation (natural log).

After transforming the equations into log form, you may discover calculation of the following type:

$$\log \sum_i \exp(v_i)$$

This may be programmed as is, but $\exp(v_i)$ may cause underflow when v_i is large and negative. One way to avoid this is to use the log-sum-exp trick. We provide the pseudo code for this trick in Algorithm 1:

```
Algorithm 1: Log-Sum-Exp Trick
Input  : $v_1, v_2, \cdots, v_n$
$m = \max(v_i)$ ; ;
$i = \{1, 2, \cdots, n\}$ ; ;
Return: $m + \log(\sum_i \exp(v_i - m))$
```

2.4 Gradescope Submission

You should submit your `learnhmm.{py|m|java|cpp}` and `forwardbackward.{py|m|java|cpp}` to Gradescope. Note: please do not use other file names. This will cause problems for the autograder to correctly detect and run your code.

Some additional tips: Make sure to read the autograder output carefully. The autograder for Gradescope prints out some additional information about the tests that it ran. For this programming assignment we've specially designed some buggy implementations that you might do, and try our best to detect those and give you some more useful feedback in Gradescope's autograder. Make wise use of autograder's output for debugging your code.

Note: For this assignment, you may make up to 10 submissions to Gradescope before the deadline, but only your last submission will be graded.