HOMEWORK 7
HIDDEN MARKOV MODELS AND BAYES NETS [*]

10-301 / 10-601 INTRODUCTION TO MACHINE LEARNING (FALL 2021)
http://mlcourse.org

OUT: Nov. 3, 2021
DUE: Nov. 12, 2021
TAs: Weyxin, Kevin, Joseph, Mukund, Brendon

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 and short answer 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**: Please read the collaboration policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html
- **Late Submission Policy:** See the late submission policy here: http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. If your scanned submission misaligns the template, there will be a 5% penalty. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader.
  - **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 3.9.6, OpenJDK 11.0.11, g++ 7.5.0) and versions of permitted libraries (e.g. `numpy` 1.21.2 and `scipy` 1.7.1) match those used on Gradescope. You have 10 free Gradescope programming submissions. After 10 submissions, you will begin to lose points from your total programming score. We recommend debugging your implementation on your local machine (or

[*]Compiled on Thursday 11th November, 2021 at 04:48

the Linux servers) and making sure your code is running correctly first before submitting your code to Gradescope.

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

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] or ND4J[b] and C++ users may use Eigen[c]. Details below. (As usual, Python users have NumPy.)

EJML for Java EJML is a pure Java linear algebra package with three interfaces. We strongly recommend using the SimpleMatrix interface. The autograder will use EJML version 0.41. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.41-libs/*:linalg_lib/nd4j-v1.0.0-M1.1-libs/*:./"` to ensure that all the EJML jars are on the classpath as well as your code.

ND4J for Java ND4J is a library for multidimensional tensors with an interface akin to Python's NumPy. The autograder will use ND4J version 1.0.0-M1.1. When compiling and running your code, we will add the additional command line argument `-cp "linalg_lib/ejml-v0.41-libs/*:linalg_lib/nd4j-v1.0.0-M1.1-libs/*:./"` to ensure that all the ND4J jars are on the classpath as well as your code.

Eigen for C++ Eigen is a header-only library, so there is no linking to worry about—just `#include` whatever components you need. The autograder will use Eigen version 3.4.0. The command line arguments above demonstrate how we will call you code. When compiling your code we will include, the argument `-I./linalg_lib` in order to include the `linalg_lib/Eigen` subdirectory, which contains all the headers.

We have included the correct versions of EJML/ND4J/Eigen in the `linalg_lib.zip` posted on the Coursework page of the course website for your convenience. It contains the same `linalg_lib/` directory that we will include in the current working directory when running your tests. Do **not** include EJML, ND4J, or Eigen in your homework submission; the autograder will ensure that they are in place.

[a] https://ejml.org
[b] https://javadoc.io/doc/org.nd4j/nd4j-api/latest/index.html
[c] 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 / Henry Chai

○ 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 / Henry Chai

○ 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

□ None of the above

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 (50 points)

1.1 Hidden Markov Models

1. (4 points) 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.

   (a) (2 points) (**Select all that apply**) Let Y_t be the state at time t. Which of the following are true under the (first-order) Markov assumption in an HMM:

   ☐ The states are independent

   ☐ The observations are independent

   ☐ $Y_t \perp\!\!\!\perp Y_{t-1} \mid Y_{t-2}$

   ■ $Y_t \perp\!\!\!\perp Y_{t-2} \mid Y_{t-1}$

   ☐ None of the above

   (b) (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**

   ☐ The current observation X_t is conditionally independent of Y_{t-2} given the previous observation X_{t-1}

   ☐ None of the above

2. (8 points) In the remaining questions, you will always see two quantities and decide what is the strongest relation between them. There is **only one correct answer**. Use the following definitions: $\alpha_t(s_j) = P(Y_t = s_j, x_{1:t})$ and $\beta_t(s_j) = P(x_{t+1:T} \mid Y_t = s_j)$.

   Note: **?** means it's not possible to assign any true relation.

   (a) (3 points) (**Select one**) Let N denote the number of possible hidden states. In other words, each $Y_t \in \{s_i\}_{i=1}^N$. What is the relation between $\sum_{i=1}^N (\alpha_5(s_i)\beta_5(s_i))$ and $P(x_{1:T})$? Select only the **strongest** relation that necessarily holds.

   ● $=$
   ○ $>$
   ○ $<$
   ○ $\leq$
   ○ $\geq$
   ○ ?

   (b) (3 points) (**Select one**) What is the relation between $P(Y_4 = s_1, Y_5 = s_2, x_{1:T})$ and $\alpha_4(s_1)\beta_5(s_2)$? Select only the **strongest** relation that necessarily holds.

   ○ $=$
   ○ $>$
   ○ $<$
   ● $\leq$
   ○ $\geq$
   ○ ?

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

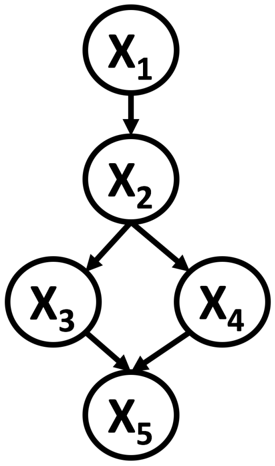
   ○ $=$
   ○ $>$
   ○ $<$
   ○ $\leq$
   ○ $\geq$
   ● ?

1.2 Graphical Models

In the Kingdom of Westeros, Summer has come. Jon Snow, the King in the North, has taken the responsibility to defeat the Giants and protect the realm.

If Jon Snow can get Queen Cersei and Daenerys Queen of the Dragons to help him Jon is likely to beat the giants. Cersei and Daenerys are powerful women who are skeptical of the existence of Giants and will most likely only consider joining Jon if the are shown evidence of an imminent Giant attack. They can only be shown of an attack if Jon captures a live Giant.

The Bayes net that represents the relationship between the events described above is shown below. Use the following notation for your variables: Jon Snow captures a live Giant (X_1), Jon shows Censei and Daenerys a live Giant (X_2), Cersei agrees to help (X_3), Daenerys agrees to help (X_4) and Giants defeated (X_5).



1. (1 point) Write down the factorization of the above directed graphical model.

   $P(X_5|X_3, X_4)P(X_3|X_2)P(X_4|X_2)P(X_2|X_1)P(X_1)$

2. (1 point) Each random variable represented in the above Bayesian network is binary valued (i.e. either the event happens or it does not). If we use the same conditional independence assumptions made in part (1), state the minimum number of parameters you need to fully specify this Bayesian network.

   11

3. (1 point) If we didn't use the conditional independence assumptions made in part (1), what would be the minimum number of parameters we would need to model any joint distribution over the same set of random variables?

   31

4. (5 points) For the following questions fill in the blank with the smallest set $\mathcal{S}$ of random variables needed to be conditioned on in order for the independence assumption to hold. For example $X_i \perp X_j \mid \mathcal{S}$. What is the smallest set $\mathcal{S}$ that makes this statement true? The empty set $\emptyset$ is a valid answer, additionally if the independence assumption cannot be satisfied no matter what we condition on then your answer should be 'Not possible'.

   (a) (1 point) $X_1 \perp X_3 \mid$ $\boxed{X_2}$

   (b) (1 point) $X_1 \perp X_5 \mid$ $\boxed{X_2}$

   (c) (1 point) $X_2 \perp X_4 \mid$ $\boxed{\text{Not possible}}$

   (d) (1 point) $X_3 \perp X_4 \mid$ $\boxed{X_2}$

   (e) (1 point) $X_2 \perp X_5 \mid$ $\boxed{X_3, X_4}$

5. (6 points) Jon gets his friend Sam to calculate some estimates of his chances. Sam returns to Jon with the following conditional probabilities tables:

$X_1 = 0$	0.3
$X_1 = 1$	0.7

	$X_1 = 0$	$X_1 = 1$
$X_2 = 0$	0.8	0.25
$X_2 = 1$	0.2	0.75

	$X_2 = 0$	$X_2 = 1$
$X_3 = 0$	0.5	0.6
$X_3 = 1$	0.5	0.4

	$X_2 = 0$	$X_2 = 1$
$X_4 = 0$	0.3	0.2
$X_4 = 1$	0.7	0.8

	$X_3 = 0, X_4 = 0$	$X_3 = 0, X_4 = 1$	$X_3 = 1, X_4 = 0$	$X_4 = 1, X_3 = 1$
$X_5 = 0$	0.4	0.7	0.8	0.5
$X_5 = 1$	0.6	0.3	0.2	0.5

Table 1: Sam's Conditional Probability tables

Using the conditional probabilities in Table 1 for our graphical model, compute the following (Your answers should be given to 2 decimal places):

(a) (2 points) $P(X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 1, X_5 = 0)$.

Answer	Work
0.020	

(b) (4 points) $P(X_1 = 1 | X_3 = 1)$

Answer	Work
0.67	

1.3 Viterbi Decoding

1. (4 points) Suppose we have a set of sequences consisting of T observed states, $x_1, \dots, x_T$, where each $x_t \in \{1, 2, 3\}$. Each observed state is associated with a hidden state $Y_t \in \{C, D\}$. Let $s_1 = C$ and $s_2 = D$.

   In the Viterbi algorithm, we seek to find the most probable hidden state sequence $\hat{y_1}, \dots, \hat{y_T}$ given the observations $x_1, \dots, x_T$.

   We define:

   - $\mathbf{B}$ is the transition matrix: $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$
   - $\mathbf{A}$ is the emission matrix: $A_{jk} = P(X_t = k \mid Y_t = s_j)$
   - $\boldsymbol{\pi}$ describes Y_1's initialization probabilities: $\pi_j = P(Y_1 = s_j)$
   - $\omega_t(s_k)$ is the maximum product of all the probabilities taken through path $Y_1, \dots, Y_{t-1}$ that ends with Y_t at state s_k.

   $$\omega_t(s_k) = \max_{y_1,\dots,y_{t-1}} P(x_{1:t}, y_{1:t-1}, Y_t = s_k) \tag{1}$$

   - $b_t(s_k)$ are the backpointers that store the path through hidden states that give us the highest product.

   $$b_t(s_k) = \operatorname*{argmax}_{y_1,\dots,y_{t-1}} P(x_{1:t}, y_{1:t-1}, Y_t = s_k) \tag{2}$$

   We outline the Viterbi Algorithm below:

   1. Initialize $\omega_1(s_j) = \pi_j A_{jx_1}$ and $b_1(j) = j$
   2. For $t > 1$, we have

   $$\omega_t(s_j) = \max_{k\in\{1,\dots,J\}} A_{jx_t} B_{kj} \omega_{t-1}(s_k)$$
   $$b_t(s_j) = \operatorname*{argmax}_{k\in\{1,\dots,J\}} A_{jx_t} B_{kj} \omega_{t-1}(s_k)$$

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

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

For the following question, consider the Hidden Markov Model specified below. Working with probabilities in the logarithm scale, we have that $\ln P(Y_1 = C) = \ln P(Y_1 = D) = -0.69$, and the state transition model and emission probability tables are given as follows.

$$\begin{tikzcd}
C \arrow[loop left, "-0.51"] \arrow[r, bend left, "-1.74"] & D \arrow[loop right, "-0.74"] \arrow[l, bend left, "-1.32"]
\end{tikzcd}$$

k	$\ln P(X_t = k \mid Y_t = C)$	$\ln P(X_t = k \mid Y_t = D)$
1	-0.69	-1.21
2	-0.91	-0.69
3	-2.30	-1.61

We observed $X_1 = 1$ and $X_2 = 2$. (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.)

(a) (2 points) Compute $\ln \omega_1(C)$ and $\ln \omega_1(D)$. If your answers involves decimal numbers, please round your answer to **TWO** decimal places.

Note: Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur. Only your answer in the left box will be graded.

What is $\ln \omega_1(C)$?

$\ln \omega_1(C)$	Work
-1.38	

What is $\ln \omega_1(D)$?

$\ln \omega_1(D)$	Work
-1.90	

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

[✓ $Y_1 = C, Y_2 = C$]

$Y_1 = D, Y_2 = D$

$Y_1 = D, Y_2 = C$

$Y_1 = C, Y_2 = D$

Not enough information.

1.4 Forward-Backward Algorithm

1. (14 points) To help you prepare to implement the HMM forward-backward algorithm (see Section 2.5 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 validation set with a single sequence composed of the same unique words used in the training set.

   Training set:

```
you     D
eat     C
fish    D

you     D
fish    D
eat     C

eat     C
fish    D
```

   Where the training word sequences are:

$$\mathbf{x}^{(1)} = [\texttt{you eat fish}]^T$$
$$\mathbf{x}^{(2)} = [\texttt{you fish eat}]^T$$
$$\mathbf{x}^{(3)} = [\texttt{eat fish}]^T$$

   And the corresponding tags are:

$$\mathbf{y}^{(1)} = [D \;\; C \;\; D]^T$$
$$\mathbf{y}^{(2)} = [D \;\; D \;\; C]^T$$
$$\mathbf{y}^{(3)} = [C \;\; D]^T$$

   Validation set:

```
fish
eat
you
```

   Where the validation word sequences are:

$$\mathbf{x}_{\text{validation}} = \left[\texttt{fish eat you}\right]^T$$

In this question, we define

- Each observed state $x_t \in \{1, 2, 3\}$, where 1 corresponds to `you`, 2 corresponds to `eat`, and 3 corresponds to `fish`
- Each hidden state $Y_t \in \{C, D\}$. Let $s_1 = C$ and $s_2 = D$.
- **B** is the transition matrix, where $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$. Note, here $\mathbf{B}$ is a 2×2 matrix.
- **A** is the emission matrix, where $A_{jk} = P(X_t = k \mid Y_t = s_j)$. Note, here $\mathbf{A}$ is a 2×3 matrix. As an example, B_{23} denotes $P(X_t = 3 \mid Y_t = s_2)$, or the probability X_t corresponds to `fish` given the hidden state $Y_t = D$
- $\boldsymbol{\pi}$ describes Y_1's initialization probabilities: $\pi_j = P(Y_1 = s_j)$

- $\alpha_t(j) = P(Y_t = s_j, x_{1:t})$, which can be computed recursively:
  1. $\alpha_1(j) = \pi_j A_{jx_1}$.
  2. For $t > 1$, $\alpha_t(j) = A_{jx_t} \sum_{k=1}^{J} \alpha_{t-1}(k) B_{kj}$
- $\beta_t(j) = P(x_{t+1:T} \mid Y_t = s_j)$, which can be computed recursively:
  1. $\beta_T(j) = 1$
  2. For $1 \leq t \leq T-1$, $\beta_t(j) = \sum_{k=1}^{J} A_{kx_{t+1}} \beta_{t+1}(k) B_{jk}$

The following four questions are meant to encourage you to work through the forward backward algorithm by hand using this validation 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 under the question (these boxes are only visible in the template document).

Note: pseudocounts used in section 2.4 should also be used here.

Note: Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur. Only your answer in the left box will be graded.

(a) (5 points) Compute $\alpha_2(C)$, the α value associated with the tag "C" for the second word in the validation sequence. Please round your answer to **TWO** decimal places.

$\alpha_2(C)$	Work
0.13	

(b) (3 points) Compute $\beta_2(D)$, the β value associated with the tag "D" for the second word in the validation sequence. Please round your answer to **TWO** decimal places.

$\beta_2(D)$	Work
0.25	

(c) (3 points) Predict the tag for the third word in the validation sequence.

Tag	Work
D	

(d) (3 points) Compute the log-likelihood for the entire validation sequence, "`fish eat you`". Please round your answer to **TWO** decimal places.

Log-Likelihood	Work
-3.04	

2. (6 points) Return to these questions after implementing your `learnhmm.{py|java|cpp}` and `forwardbackward.{py|java|cpp}` functions. Please ensure that you have used the log-sum-exp trick in your programming as described in section 3.3 before answering these empirical questions.

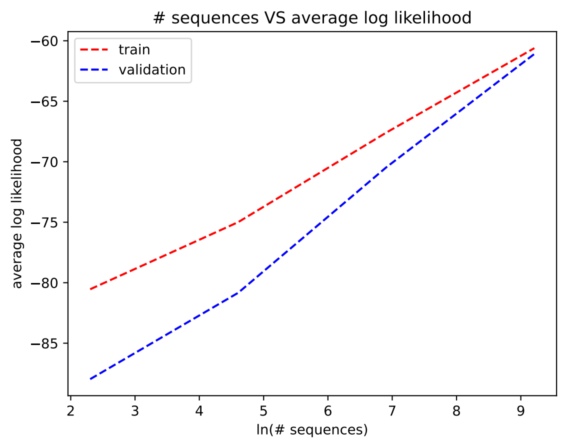
   Using the full data set **train.txt** in the handout, use your implementation of `learnhmm.{py|java|cpp}` to learn parameters for an hmm model using the first 10, 100, 1000, and 10000 sequences in the file. Use these learned parameters to perform prediction on the English **train.txt** and the **validation.txt** files using your `forwardbackward.{py|java|cpp}`. Construct a plot with number of sequences used for training on the x-axis (log-scale with base e) and average log likelihood across all sequences from the English **train.txt** and the **validation.txt** on the y-axis (see Section 2.5 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 rounded to two decimal places 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.

   (a) (4 points) Fill in this table.

# Sequences	Train Average Log-Likelihood	Validation Average Log-Likelihood
10	-80.54	-87.97
100	-74.99	-80.83
1000	-67.59	-70.46
10000	-60.61	-61.09

   (b) (2 points) Put your plot below:

Plot



2 Programming (80 points)

2.1 The Task

In the programming section you will implement a named entity recognition system using Hidden Markov Models (HMMs). Named entity recognition (NER) is the task of classifying named entities, typically proper nouns, into pre-defined categories, such as person, location, organization. Consider the example sequence below, where each word is appended with a tab and then its tag:

“	O
Rhinestone	B-ORG
Cowboy	I-ORG
”	O
(	O
Larry	B-PER
Weiss	I-PER
)	O
-	O
3:15	O

`Rhinestone` and `Cowboy` are labeled as an organization (`ORG`), while `Larry` and `Weiss` is labeled as a person (`PER`). Words that aren't named entities are assigned the `O` tag. The `B-` prefix indicates that a word is the beginning of an entity, while the `I-` prefix indicates that the word is inside the entity.

NER is an incredibly important task for a machine to begin to analyze and interpret a body of natural language text. For example, when designing a system that automatically summarizes news articles, it is important to recognize the key subjects in the articles. Another example is designing a trivia bot. If you can quickly extract the named entities from the trivia question, you may be able to more easily query your knowledge base (e.g. type a query into Google) to request information about the answer to the question.

2.2 The Dataset

WikiANN is a “silver standard” dataset that was generated without human labelling. The English Abstract Meaning Representation (AMR) corpus and DBpedia features were used to train an automatic classifier to label Wikipedia articles. These labels were then propagated throughout other Wikipedia articles using the Wikipedia's cross-language links and redirect links. Afterwards, another tagger that self-trains on the existing tagged entities was used to label all other mentions of the same entities, even those with different morphologies (prefixes and suffixes that modify a word in other languages). Finally, the amassed training examples were filtered by “commonness” and “topical relatedness” to pick more relevant training data.

The WikiANN dataset provides labelled entity data for Wikipedia articles in 282 languages. We will be primarily using the English subset, which contains 14,000 training examples and 3,300 test examples, and the French subset, which contains around 7,500 training examples and 300 test examples. On a technical level, the main task is to implement an algorithm to learn the HMM parameters given the training data and then implement the forward backward algorithm to perform a smoothing query which we can then use to predict the hidden tags for a sequence of words.

2.3 File Formats

The handout files 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. **train.txt** This file contains labeled text data that you will use in training your model in the Learning problem. Specifically the text contains one word per line that has already been preprocessed, cleaned and tokenized. Every sequence has the following format:

   ```
   <Word0>\t<Tag0>\n<Word1>\t<Tag1>\n ...  <WordN>\t<TagN>\n
   ```

   where every `<WordK>\t<TagK>` unit token is separated by a newline. Between each sequence is an empty line. If we have two three-word sequences in our data set, the data will look like so:

   ```
   <Word0>\t<Tag0>\n
   <Word1>\t<Tag1>\n
   <Word2>\t<Tag2>\n
   \n
   <Word0>\t<Tag0>\n
   <Word1>\t<Tag1>\n
   <Word2>\t<Tag2>
   ```

   Note: Word 2 of the second sequence does not end with a newline because it is the end of the data set.

2. **validation.txt**: This file contains labeled data that you will use to evaluate your model in the Experiments section. This file has the same format as **train.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. The format is simple:

index_to_word.txt	**index_to_tag.txt**
`<Word0>\n`	`<Tag0>\n`
`<Word1>\n`	`<Tag1>\n`
`<Word2>\n`	`<Tag2>\n`
⋮	⋮

   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 2.5).

4. **predicted.txt**: This file contains labeled data that you will use to debug your implementation. The labels in this file are generated by running our decoder using the features from **train.txt**. This file has the same format as **train.txt**.

5. **metrics.txt**: This file contains the metrics you will compute for the validation data. The first line should contain the average log likelihood, and the second line should contain the prediction accuracy. There should be a single space after the colon preceding the metric value; see the reference output file for more detail.

6. **hmmtrans.txt, hmmemit.txt and hmminit.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. The formats of these files are:

   hmmtrans.txt:
   ```
   <ProbS1S1> <ProbS1S2> ...  <ProbS1SN>\n
   <ProbS2S1> <ProbS2S2> ...  <ProbS2SN>\n...
   ```

   hmmemit.txt:
   ```
   <ProbS1Word1> <ProbS1Word2> ...  <ProbS1WordN>\n
   <ProbS2Word1> <ProbS2Word2> ...  <ProbS2WordN>\n...
   ```

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

   hmminit.txt:
   ```
   <ProbS1>\n
   <ProbS2>\n...
   ```

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

Note that the data provided to you is to help in developing your implementation of the HMM algorithms. Your code will be tested on Gradescope using different subsets of data with identical formatting.

2.4 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 2.5). There are three sets of parameters that you will need to estimate: the initialization probabilities $\boldsymbol{\pi}$, the transition probabilities $\mathbf{B}$, and the emission probabilities $\mathbf{A}$. For this assignment, we model each of these probabilities using a multinomial distribution with parameters $\pi_j = P(Y_1 = s_j)$, $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$, and $A_{jk} = P(X_t = k \mid Y_t = s_j)$. These can be estimated using maximum likelihood, which results in the following parameter estimators:

1. $P(Y_1 = s_j) = \pi_j = \frac{N_{Y_1=s_j}+1}{\sum_{p=1}^{J}(N_{Y_1=s_p}+1)}$, where $N_{Y_1=s_j}$ equals the number of times state s_j is associated with the first word of a sentence in the training data set.

2. $P(Y_t = s_k \mid Y_{t-1} = s_j) = B_{jk} = \frac{N_{Y_t=s_k,Y_{t-1}=s_j}+1}{\sum_{p=1}^{J}(N_{Y_t=s_p,Y_{t-1}=s_j}+1)}$, where $N_{Y_t=s_k,Y_{t-1}=s_j}$ is the number of times state s_j is followed by state s_k in the training data set.

3. $P(X_t = k \mid Y_t = s_j) = A_{jk} = \frac{N_{X_t=k,Y_t=s_j}+1}{\sum_{p=1}^{M}(N_{X_t=p,Y_t=s_j}+1)}$, where $N_{X_t=k,Y_t=s_j}$ 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 validation set.

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

Your outputs should be in the same format as **hmminit.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:
```
$ python3 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...]
```

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

1. `<train_input>`: path to the training input `.txt` file (see Section 2.2)
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 0, the second word having index of 1, 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 0, the second tag having index of 1, etc.
4. `<hmminit>`: path to output `.txt` file to which the estimated initialization probabilities ($\boldsymbol{\pi}$) will be written. The file output to this path should be in the same format as the handout `hmminit.txt` (see Section 2.2).
5. `<hmmemit>`: path to output `.txt` file to which the emission probabilities ($\mathbf{A}$) will be written. The file output to this path should be in the same format as the handout `hmmemit.txt` (see Section 2.2)
6. `<hmmtrans>`: path to output `.txt` file to which the transition probabilities ($\mathbf{B}$) will be written. The file output to this path should be in the same format as the handout `hmmtrans.txt` (see Section 2.2).

2.5 Evaluation and Decoding

2.5.1 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, $x_1, \dots, x_T$. Each word is associated with a label $Y_t \in \{1, \dots, J\}$. In the forward-backward algorithm we seek to approximate $P(Y_t \mid x_{1:T})$ up to a multiplication constant. This is done by first breaking $P(Y_t \mid x_{1:T})$ into a "forward" component and a "backward" component as follows:

$$\begin{aligned} P(Y_t = s_j \mid x_{1:T}) &\propto P(Y_t = s_j, x_{t+1:T} \mid x_{1:t}) \\ &\propto P(Y_t = s_j \mid x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j, x_{1:t}) \\ &\propto P(Y_t = s_j \mid x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j) \\ &\propto P(Y_t = s_j, x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j) \end{aligned}$$

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

Forward Algorithm

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

$$\begin{aligned} \alpha_t(j) =& P(Y_t = s_j, x_{1:t}) \\ =& \sum_k P(Y_t = s_j, Y_{t-1} = s_k, x_{1:t}) \\ =& \sum_k P(Y_{t-1} = s_k, x_{1:t} \mid Y_t = s_j) P(Y_t = s_j) \\ =& \sum_k P(x_t \mid Y_t = s_j) P(Y_{t-1} = s_k, x_{1:t-1} \mid Y_t = s_j) P(Y_t = s_j) \\ =& P(x_t \mid Y_t = s_j) \sum_k P(Y_t = s_j, Y_{t-1} = s_k, x_{1:t-1}) \\ =& P(x_t \mid Y_t = s_j) \sum_k P(Y_t = s_j, x_{1:t-1} \mid Y_{t-1} = s_k) P(Y_{t-1} = s_k) \\ =& P(x_t \mid Y_t = s_j) \sum_k P(Y_t = s_j \mid Y_{t-1} = s_k) P(x_{1:t-1} \mid Y_{t-1} = s_k) P(Y_{t-1} = s_k) \\ =& P(x_t \mid Y_t = s_j) \sum_k P(Y_t = s_j \mid Y_{t-1} = s_k) P(Y_{t-1} = s_k, x_{1:t-1}) \\ =& A_{jx_t} \sum_k B_{kj} \alpha_{t-1}(k) \end{aligned} \tag{3}$$

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

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

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

$$\begin{aligned}
\beta_t(j) &= P(x_{t+1:T} \mid Y_t = s_j) \\
&= \sum_{k=1}^{J} P(Y_{t+1} = s_k, x_{t+1:T} \mid Y_t = s_j) \\
&= \sum_{k=1}^{J} P(x_{t+1:T} \mid Y_t = s_j, Y_{t+1} = s_k) P(Y_{t+1} = s_k \mid Y_t = s_j) \\
&= \sum_{k=1}^{J} P(x_{t+1} \mid Y_{t+1} = s_k) P(x_{t+2:T} \mid Y_{t+1} = s_k) P(Y_{t+1} = s_k \mid Y_t = s_j) \\
&= \sum_{k=1}^{J} A_{k x_{t+1}} \beta_{t+1}(k) B_{jk} \qquad (4)
\end{aligned}$$

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 $1 \le t \le T-1$, $\beta_t(j) = \sum_{k=1}^{J} A_{k x_{t+1}} \beta_{t+1}(k) B_{jk}$ (Generate x_{t+1} from any state)

Forward-Backward Algorithm As stated above, the goal of the Forward-Backward algorithm is to compute $P(Y_t = s_j \mid x_{1:T})$. This can be done using the following equation:

$$P(Y_t = s_j \mid x_{1:T}) \propto P(Y_t = s_j, x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j)$$

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

$$P(Y_t \mid x_{1:t}) \propto \alpha_t \circ \beta_t$$

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:

$$\hat{Y}_t = \underset{j \in \{1,\ldots,J\}}{\text{argmax}}\ P(Y_t = s_j \mid x_{1:T})$$

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 $log(P(x_{1:T}))$. We can rewrite this in terms of values we have already computed in the forward-backward algorithm as follows:

$$\begin{aligned}\log P(x_{1:T}) &= \log\big(\sum_j P(x_{1:T}, Y_t = s_j)\big)\\ &= \log\big(\sum_j \alpha_T(j)\big)\end{aligned}$$

2.5.2 Implementation Details

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

```
For Python: $ python3 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...]
```

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

1. `<validation_input>`: path to the validation input `.txt` file that will be evaluated by your forward backward algorithm (see Section 2.2)
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 0, the second word having index of 1, etc. This is the same file as was described for `learnhmm.{py|java|cpp}`.
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 0, the second tag having index of 1, etc. This is the same file as was described for `learnhmm.{py|java|cpp}`.
4. `<hmminit>`: path to input `.txt` file which contains the estimated initialization probabilities ($\boldsymbol{\pi}$).
5. `<hmmemit>`: path to input `.txt` file which contains the emission probabilities ($\mathbf{A}$).
6. `<hmmtrans>`: path to input `.txt` file which contains transition probabilities ($\mathbf{B}$).
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 `<validation_input>` file.
8. `<metric_file>`: path to the output `.txt` file to which the metrics will be written.

Example command for python users:

```
$ python3 forwardbackward.py toy_data/validation.txt \
toy_data/index_to_word.txt toy_data/index_to_tag.txt \
toy_data/hmminit.txt toy_data/toy_hmmemit.txt \
toy_data/hmmtrans.txt toy_data/predicted.txt \
toy_data/metrics.txt
```

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

```
fish    D
eat     C
you     D
```

And the `<metric_file>` output should be:

```
Average Log-Likelihood: -3.0438629330222424
Accuracy: 0.3333333333333333
```

where average log-likelihood and accuracy are evaluated over the validation set.

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`.

2.5.3 Log-Space Arithmetic for Avoiding Underflow

Handling underflow properly is a critical step in implementing an HMM. 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 starts with transforming Eq.(3) and Eq.(4) 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

```
1: procedure LogSumExpTrick($(v_1, v_2, \cdots, v_n)$)
2:     $m = \max(v_i)$ for $i = \{1, 2, \cdots, n\}$
3:     return $m + \log(\sum_i \exp(v_i - m))$
```

Note: The autograder test cases account for numerical underflow using the Log-Sum-Exp Trick. If you do not implement forwardbackward.py with the trick, you might only receive partial credit.

2.6 Gradescope Submission

You should submit your `learnhmm.{py|java|cpp}` and `forwardbackward.{py|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 have 10 submissions to Gradescope before the deadline, but only your last submission will be graded.

3 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found here.

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

Your Answer