COMS W4701: Artificial Intelligence, Spring 2022

Homework 4

Instructions: Compile all solutions to the written problems on this assignment in a single PDF file. **Show your work by writing down relevant equations or expressions, and/or by explaining any logic that you are using to bypass known equations**. Coding solutions may be directly implemented in the provided Python file(s). When ready, follow the submission instructions to submit all files to Gradescope. Please be mindful of the deadline, as late submissions are not accepted, as well as our course policies on academic honesty.

Problem 1: Conditional Independence in Markov Chains (20 points)

The basic conditional independence assumption of Markov chains is that the state X_t is conditionally independent of X_0, ..., X_{t-2} given X_{t-1}. A corollary of this assumption is that X_t is conditionally independent of X_j given X_k for *any* j and k satisfying $0 \leq j < k < t$. We will prove this property by showing that $\Pr(X_t \mid x_k, x_j) = \Pr(X_t \mid x_k)$.

(a) Show the relationship between $\Pr(X_t \mid x_k, x_j)$ and $\Pr(X_t, x_k, x_j)$. What is the interpretation of each term?

(b) Given a subset of the full joint distribution of the Markov chain, the probabilities of $\Pr(X_t, x_k, x_j)$ can be obtained as follows:

$$\Pr(x_t, x_k, x_j) = \sum \Pr(x_0, x_1, \ldots, x_t). \tag{1}$$

What are the indices of summation; in other words, which variables in the joint distribution do we need to sum over?

(c) Recall that joint probabilities can written in terms of the Markov chain parameters as

$$\Pr(x_0, x_1, \ldots, x_t) = \Pr(x_0) \prod_{i=1}^{t} \Pr(x_i \mid x_{i-1}). \tag{2}$$

Inserting Eq. (2) into Eq. (1) will give you a summation over a product. Show that this summation can be rewritten into the following product:

$$\Pr(x_t, x_k, x_j) = \Pr(x_j) \Pr(x_k \mid x_j) \Pr(x_t \mid x_k) \tag{3}$$

As a hint, you should *factor* the initial summation into terms representing each of three components above, each of which may still be written as a summation.

(d) Eq. (3) holds for all values of x_t. Show how we can conclude that X_t is conditionally independent of X_j given X_k.

Problem 2: Robot Localization (30 points)

A robot is wandering around a room with some obstacles, labeled as # in the grid below. It can occupy any of the free cells labeled with a letter, but we are uncertain about its true location and thus keep a belief distribution over its current location. At each timestep it moves from its current cell to a neighboring free cell in one of the four cardinal directions with uniform probability; it cannot stay in the same cell. For example, from A the robot can move to either B or D with probability $\frac{1}{2}$, while from B it can move to A, C or E, each with probability $\frac{1}{3}$.

A	B	C
D	E	#
#	F	#

The robot may also make an observation after each transition, returning what it sees in a random cardinal direction. Possibilities include observing #, "wall", or "empty" (for a free cell). For example, in D the robot observes "wall", # (each with probability $\frac{1}{4}$), or "empty" (probability $\frac{1}{2}$).

(a) Suppose the robot wanders around forever without making any observations. What is the stationary distribution π over the robot's predicted location?

(b) Now suppose that we know that the robot is in state E; i.e., $\Pr(X_0 = E) = 1$. Starting from this state, the robot makes one transition and observes $e_1 =$ "wall". What is the belief distribution $\Pr(X_1 \mid e_1)$?

(c) The robot makes a second transition and observes $e_2 = \#$. What is the updated belief distribution $\Pr(X_2 \mid e_1, e_2)$?

(d) Compute the joint distribution $\Pr(X_1, X_2 \mid e_1, e_2)$. You do not need to explicitly list the values that have probability 0. What is the most likely state sequence(s)?

(e) Compute $\mathbf{b} = \Pr(e_2 \mid X_1)$. Briefly explain what this quantity represents.

(f) Compute the smoothed distribution $\Pr(X_1 \mid e_1, e_2)$ by multiplying $\mathbf{f} = \Pr(X_1 \mid e_1)$ with $\mathbf{b} = \Pr(e_2 \mid X_1)$ and normalizing the result. Confirm that the distribution is the same as that obtained from marginalization of $\Pr(X_1, X_2 \mid e_1, e_2)$.

Problem 3: Part-of-Speech Tagging

In this problem you will explore part-of-speech (POS) tagging, a standard task in natural language processing. The goal is to identify parts of speech and related labels for each word in a given corpus. Hidden Markov models are well suited for this problem, with parts of speech being hidden states and the words themselves being observations. We will be using data from the English EWT treebank from Universal Dependencies, which uses 17 POS tags. We are providing clean versions of training and test data for you. The data format is such that each line contains a word and associated tag, and an empty line signifies the end of a sentence. Feel free to open the files in a text editor to get an idea.

The provided Python file contains a couple of functions that can be used to read a data file (you do not need to call these yourself). The global variable `POS` contains a list of all possible parts of speech. You will be filling in the remaining functions in the file and running the code in main where instructed.

3.1: Supervised Learning (10 points)

Your first task is to learn the three sets of HMM parameters from the training data. The initial distribution $\Pr(X_0)$ will be stored in a 1D array of size 17. The transition probabilities will be stored in a 2D array of size 17×17. The observation probabilities will be stored in a dictionary, where each key is a word and the value is a 1D array (size 17) of probabilities $\Pr(\text{word} \mid \text{POS})$. These probabilities should follow the same order as in the `POS` list.

Implement `learn_model` so that it compiles and returns these three structures. The `data` input is a list of sentences, each of which is a list of `(word, POS)` pairs. Your method should iterate over each sentence in the training data, counting the POS appearances in the first word, the number of POS to POS transitions, and the number of POS to word observations. Treat each sentence independently; do not count transitions between different sentences.

Be sure to correctly normalize all of these distributions. Make sure that the quantities $\sum_i \Pr(X_0)_i$, $\sum_i \Pr(X_{t+1} = i \mid X_t = j)$, and $\sum_k \Pr(E_t = k \mid X_t)$ are all equal to 1.

3.2: Viterbi Algorithm (10 points)

The next task is to implement the Viterbi algorithm to predict the most likely sequence of states given a sequence of observations. We will break this into two pieces: `viterbi_forward` and `viterbi_backward`. `viterbi_forward` takes in four parameters: initial distribution `X0` (1D array), transition matrix `Tprobs` (2D array), observation probabilities `Oprobs` (dictionary), and observation sequence `obs` (list of word observations) of length `T`.

`viterbi_forward` should compute and return two quantities: $\max_{x_1,\ldots,x_{T-1}} \Pr(x_1, \ldots, x_{T-1}, X_T, e_{1:T})$ as a 1D array of size 17, and a `T` $\times 17$ 2D array of pointer indices. Row `i` of the pointer array should contain $\operatorname{argmax}_{x_{i-1}} \Pr(x_1, \ldots, x_{i-1}, X_i, e_{1:i})$. For simplicity, pointers will be the *indices* of the POS in the `POS` array rather than strings. Note that it is possible for an observation e to not exist in the `Oprobs` dictionary if it was not present when the model was trained. If this occurs, you may simply take $\Pr(e \mid x) = 1$ for all POS x; this is equivalent to skipping the observation step.

`viterbi_backward` takes the two quantities returned by `viterbi_forward` as parameters. It should start with the most likely POS according to $\max_{x_1,\ldots,x_{T-1}} \Pr(x_1, \ldots, x_{T-1}, X_T, e_{1:T})$ and then follow the pointers backward to reconstruct the entire POS sequence $\operatorname{argmax}_{x_1,\ldots,x_T} \Pr(x_1, \ldots, x_T \mid e_{1:T})$. The returned object should be a list of POS (strings) from state 1 to state `T`. Note that the predicted state for X_0 should not be included, so your list should be length `T`.

3.3: Model Evaluation (10 points)

Your Viterbi implementation can now be used for prediction. `evaluate_viterbi` takes in the HMM parameters, along with a `data` list in the same format as in `learn_model`. Complete the function so that it runs Viterbi on each sentence separately on the data set. Then compare all returned POS predictions with the true POS in `data`. Compute and return the **accuracy rate** as the proportion of correct predictions (this should be a number between 0 and 1). After implementing this function, run the associated code in main and answer the following questions.

(a) Report the accuracies of your Viterbi implementation on each data set. Why is the accuracy on the test data set lower than that of the training set?

(b) Why can we not expect 100% accuracy on the training set, despite defining the HMM parameters to maximize the likelihood on the training data set?

3.4: Forward and Backward Algorithms (10 points)

Next, implement the the forward, backward, and forward-backward algorithms (ideally fewer than 5 lines of code each). `forward` takes in the same four parameters as `viterbi_forward` above, and it computes and returns $\Pr(X_k, e_{1:k})$ (no normalization). `backward` also takes in four parameters, dropping `X0` but newly including the state index `k`. It should compute and return $\Pr(e_{k+1:T} \mid X_k)$. Note that `k` follows Python indexing; in other words, `k=0` corresponds to X_1 and `backward` should return $\Pr(e_{2:T} \mid X_1)$. Again, to deal with the scenario in which a word is not in the observation model, you should explicitly check when this occurs and if so simply use $\Pr(e \mid x) = 1$.

Once you have both `forward` and `backward`, it should be straightforward to call these to implement `forward_backward`, which computes the smoothed state distribution $\Pr(X_k \mid e_{1:T})$. Note that the call to `forward` should only use the observations up to (and including) the `k`th one. Don't forget to normalize the smoothed distribution.

3.5: Expected Emissions (10 points)

The forward-backward algorithm is used in the Baum-Welch algorithm to perform unsupervised learning of the observation probabilities. In particular, the smoothed distribution $\Pr(X_k \mid e_{1:T})$ gives the *expected* proportion of hidden states that produce the observation e_k. Recording these expected state-observation proportions for all k and then normalizing will give us a new set of observation probabilities.

Write the `expected_emissions` function, which takes in `X0`, `Tprobs`, `Oprobs`, and `data`, the latter as a list of sentences, each of which is a list of words. Within each sentence, compute the expected distribution of hidden states for each word. Save each one in a dictionary just like `Oprobs` (if a word appears more than once, add the resultant state distributions together). Finally, normalize the distributions and return the new observation probabilities.

(a) When finished, run the last couple lines in main, which will call `expected_emissions` on the first sentence of the test data set. Print and examine the observation probabilities. Which words appear to be ambiguous in specifying the POS that generated them and why?

Submission

You should have one PDF document containing your solutions for problems 1-2, as well as your responses to 3.3 and 3.5. You should also have a completed Python file implementing problem 3. **Please keep the filename and all function headers unchanged, and save and upload your code as a `.py` file.** Submit the document and code file to the respective assignment bins on Gradescope. For full credit, you must tag your pages for each given problem on the former.