CS 486/686 Assignment 3
Fall 2022
(125 marks)

Blake VanBerlo

Due Date: 11:59 pm ET on Tuesday, November 15, 2022

Instructions

- Submit the signed academic integrity statement any written solutions in a file to the Q0 box in the A3 project on Crowdmark. **(5 marks)**.
- Submit your written answers to questions 1 and 2 as PDF files to the Q1 and Q2 boxes respectively in the A3 project on Crowdmark. I strongly encourage you to complete your write-up in LaTeX, using this source file. If you do, in your submission, please replace the author with your name and student number. Please also remove the due date, the Instructions section, and the Learning Goals section. Thank you!
- Submit any code to `Marmoset` at `https://marmoset.student.cs.uwaterloo.ca/`. Be sure to submit your code to the project named `Assignment 3 - Final`.
- No late assignment will be accepted. This assignment is to be done individually.
- Lead TAs:
  - Connor Raymond Stewart (`crstewart@uwaterloo.ca`)
  - Dake Zhang (`dake.zhang@uwaterloo.ca`)

  The TAs' office hours will be scheduled and posted on LEARN and Piazza.

Learning goals

Hidden Markov Models (HHM)

- Derive the formula for the prediction inference task in HMMs
- Trace the forward-backward algorithm

Decision Trees

- Implement the decision tree learner algorithm to learn a decision tree using a dataset with real-valued features.
- Determine the accuracy of a decision tree on a dataset.
- Perform pre-pruning and post-pruning. Determine the best parameter value for pruning using k-fold cross validation.

1 Hidden Markov Models (30 marks)

(a) In Lectures 10 and 11, we used a different form of the Bayes rule in our derivations for the filtering and smoothing formulas. Show that $P(A \mid B \wedge C) = \alpha P(B \mid A \wedge C) P(A \mid C)$, where α is a normalizing constant. In your derivation, clearly identify any probability rules that you use (see the rules in Lecture 6).

Marking Scheme:

(2 marks) Correct derivation

(2 marks) Correct rules cited

(b) Several commodities have been experiencing volatile prices recently, often due to supply chain issues and product shortages. Olive oil is an example of a product that might experience product shortages this year due to droughts in Europe.

Suppose that you would like to determine the state of olive oil availability, given the price that you observe during your weekly grocery shopping trip. You model this problem as a Hidden Markov Model (HMM), where the hidden state S_t is the state of olive oil supply (normal, supply chain backup, or shortage) and the observation is the price you see in the grocery store for a 1 L bottle (low, normal, high). You make some assumptions regarding the transition and sensor distributions. For 3 weeks, you take note of the olive oil price, obtaining the HMM shown below:

State S_t:

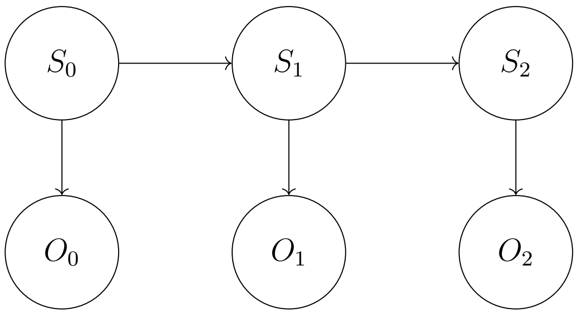
0:	Normal supply
1:	Supply chain issues
2:	Product shortage

Observation O_t:

0:	Low price
1:	Normal price
2:	High price

$$\boxed{\begin{array}{l} P(S_0 = 0) = 0.5 \\ P(S_0 = 1) = 0.3 \end{array}}$$

$$\boxed{\begin{array}{l} P(S_t = 0 \mid S_{t-1} = 0) = 0.7 \\ P(S_t = 1 \mid S_{t-1} = 0) = 0.2 \\ P(S_t = 0 \mid S_{t-1} = 1) = 0.3 \\ P(S_t = 1 \mid S_{t-1} = 1) = 0.5 \\ P(S_t = 0 \mid S_{t-1} = 2) = 0.3 \\ P(S_t = 1 \mid S_{t-1} = 2) = 0.1 \end{array}}$$



$$\boxed{\begin{array}{l} P(O_t = 0 \mid S_t = 0) = 0.3 \\ P(O_t = 1 \mid S_t = 0) = 0.6 \\ P(O_t = 0 \mid S_t = 1) = 0.1 \\ P(O_t = 1 \mid S_t = 1) = 0.25 \\ P(O_t = 0 \mid S_t = 2) = 0.05 \\ P(O_t = 1 \mid S_t = 2) = 0.3 \end{array}}$$

You observed that the price of olive oil was normal in week 0, high in week 1, and high in week 2. Execute the Forward Backward Algorithm to determine the probability distribution for the hidden state at each of weeks 0, 1, and 2. That is, calculate $P(S_0 \mid O_0 = 1 \land O_1 = 2 \land O_2 = 2)$, $P(S_1 \mid O_0 = 1 \land O_1 = 2 \land O_2 = 2)$, and $P(S_2 \mid O_0 = 1 \land O_1 = 2 \land O_2 = 2)$. Normalize each forward message. Show all calculations and round each calculation to 4 decimal places.

Marking Scheme:

(10 marks) Correct execution of the Forward Backward Algorithm

(2 marks) Calculations are displayed clearly

(c) Recall that *prediction* in Hidden Markov Models is the task of computing the probability distribution for the hidden state at some future time step. More specifically, prediction computes the distribution $P(S_k \mid o_{0:t})$, where t is the current time step and k is some future time step ($k > t$).

Given the state distribution for time step $k \geq t$, show that the distribution for the state at time step $k + 1$ can be recursively computed as follows:

$$P(S_{k+1} \mid o_{0:t}) = \sum_{s_k} P(S_{k+1} \mid s_k) P(s_k \mid o_{0:t})$$

In your derivation, clearly identify any conditional independence assumptions and/or probability rules that you use.

Marking Scheme:

(6 marks) Correct derivation

(2 marks) Correct rules/assumptions cited

(d) After many time steps (i.e., the *mixing time*), the distribution returned by prediction will converge to a fixed point. In other words, once convergence is reached, $P(S_k \mid o_{0:t})$ does not change as k increases. This distribution is referred to as the *stationary distribution* of the HMM.

What is the stationary distribution of the HMM in Question 1(b)? Show your calculations.

Marking Scheme:

(4 marks) Correct answer and justification

(2 marks) Calculations are displayed clearly

2 Decision Trees (90 marks)

You will implement an algorithm to build a decision tree for the HTRU2 dataset. Read more about the dataset here.

Information on the provided code

We have provided a dataset and four Python files. Please read the detailed comments in the provided files carefully.

1.	`data.csv`	Includes the dataset. **Do not change this file.**
2.	`dt_global.py`	Defines several useful global variables **Do not change the function signatures.**
3.	`dt_provided.py`	Defines some useful functions for reading the dataset and splitting the dataset into different folds. Do not modify or submit this file. **Do not change this file.**
4.	`dt_core.py`	Contains empty functions for generating the tree and performing pruning. Include this file in your submission. **Do not change the function signatures.**
5.	`dt_cv.py`	Contains empty functions for performing cross validation. You need to complete all the empty functions in these files. **Do not change the function signatures.**

The dataset has real-valued features. When generating a decision tree, use **a binary split** at each node. At each node in the decision tree, choose a feature and a split point for the feature using the procedure described on slide 48 of Lecture 13 to choose potential split points. The node should test whether a feature has a value greater than the split point or not. Along a path from the root node to a leaf node, we may test a real-valued feature multiple times with different split points.

The Attributes in an Anytree Node

We use the `anytree` package to store the decision tree. Our unit tests make use of several custom attributes in an Anytree Node. Make sure that you use these attributes so that you will pass our tests.

- `name`: Each Node requires the `name` attribute, which is a string. The `name` of each node should be unique. You can generate this attribute in any way you like. We won't test this attribute.
- `parent`: To construct the tree properly, you either need to set a Node's `parent` attribute (which is a Node as well) or a Node's `children` attributes. See the Node documentation for more details. We recommend setting the `parent` attribute of each

Node. Once the `parent` attribute is set, the `children` attribute will be set automatically by Anytree. If you set the left child node's parent attribute before setting the right child node's parent attribute, then you can retrieve the left child node as `children[0]` and the right child node as `children[1]`.

- The `feature` attribute stores the chosen feature as a string. The `split` attribute stores the split point value as a float. Any non-leaf node should have the `feature` and `split` attributes.
- The `decision` attribute stores the decision as an integer. Any leaf node should have the `decision` attribute.

Tie-Breaking Rules

Please use the following tie-breaking rules to ensure that your program passes the unit tests.

1. If a leaf node has examples with different labels, we need to determine a decision using majority vote. If there is a tie, return the label with the smallest value. For example, if a leaf node has two examples with label 4 and two examples with label 5, the majority decision should be 4.
2. Given a feature, if there are multiple split points with the same maximum information gain, choose the split point with the smallest value. For example, suppose that, the split points `9.3` and `9.5` tie for the maximum information gain among all the spilt points for the feature, we will choose to split on `9.3`.
3. Suppose that, for each feature, we have identified the split point with the maximum information gain. Given the best split points for the features, if multiple split points have the same maximum information gain, choose the first feature based on the order of the features in the first row of the `data.csv` file.

   For example, assume that feature A comes before feature B in the order. Suppose that the information gain for the best split point for A and the best split point for B tie for the maximum information gain (among the best split points for all the features), we will choose to split on the best split point for A.

Floating Point Comparisons

A correct implementation may fail the Marmoset tests due to floating point issues. To prevent this, please compare floating-point values using a tolerance. We have provided two functions `less_than` and `less_than_or_equal_to` in `dt_provided.py`. Make sure that you use these functions when comparing floating-point values.

k-Fold Cross-Validation

The purpose of performing cross validation is to choose the best value for a parameter. We will use k-fold cross validation with $k = 10$ to choose the best parameter value in pre-pruning and post-pruning.

In 10-fold cross-validation, each data point serves double duty — as training data and validation data. Below are the steps for conducting 10-fold cross-validation.

1. Split the data into 10 subsets using `preprocess` in `dt_provided.py`.
2. For each parameter value, perform 10 rounds of learning. In the i^{th} round, the validation set is the i^{th} fold and the training set consists of the remaining 9 folds.
3. In the i^{th} round, learn a decision tree with the parameter value using the training set. Determine the prediction accuracy of the decision tree on the training set and on the validation set. The prediction accuracy is the fraction of examples that the tree predicts correctly. (Don't worry if you generate different trees in different rounds. We do not care about the trees generated. We only care about the prediction accuracies.)
4. After the 10 rounds, calculate the average prediction accuracy of the decision tree on the training set and on the validation set, where the average is taken over the 10 rounds. For each parameter value, you will produce two numbers: the average prediction accuracy on the training set and the average prediction accuracy on the validation set.
5. Choose the parameter value with the highest average prediction accuracy on the validation set.

Packages:

You can assume that numpy (version 1.19.5) and anytree (version 2.8.0) are available in our testing environment.

Efficiency:

The unit tests will evaluate your implementation for correctness and efficiency. If your implementation does not terminate within a pre-defined time limit, it will fail the unit test. We set the time limits by taking our run-times and multiplying it by a small constant factor. For your information, our program terminates within 5 s for part 2(b), 160 s for part 2(c), and 70 s for part 2(d).

Here is some advice for improving your program's efficiency.

- Limit the external packages that your program uses. Do not use `pandas` — it will slow down your program considerably. Limit your use of `numpy`. It's sufficient to use Python arrays and dictionaries only.

- Think of ways to perform pruning efficiently. You may be able to performing pruning without modifying the tree. You may also be able to start with a previously generated tree instead of building a full tree all over again.

Please complete the following tasks.

(a) Complete all the empty functions in `dt_core.py` and `dt_cv.py` and submit both files on Marmoset in a zipped folder.

Marking Scheme: (85 marks)

- `get_splits`
  (1 public test + 4 secret tests) * 2 marks = 10 marks
- `choose_feature_split`
  (1 public test + 4 secret tests) * 3 marks = 15 marks
- `split_examples`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `learn_dt` (and `split_node`)
  1 public test * 1 mark + 2 simple secret tests * 2 marks + 2 full-tree secret tests * 5 marks = 1 + 4 + 10 = 15 marks
- `predict`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `get_prediction_accuracy`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `post_prune`
  (1 public test + 4 secret tests) * 2 marks = 10 marks
- `cv_pre_prune`
  (1 public test + 4 secret tests) * 2 mark = 10 marks
- `cv_post_prune`
  (1 public test + 4 secret tests) * 2 mark = 10 marks

(b) Suppose that we built a decision tree called `tree-full` using the dataset. What is the maximum depth of `tree-full`?

Marking Scheme:

(1 mark) Correct value of the maximum depth of the tree.

(c) Suppose that we want to pre-prune `tree-full` using the maximum depth criterion and using majority voting to make decisions at leaf nodes. What is the best value of the tree's maximum depth through ten-fold cross-validation? Please use the range `list(range(0, 40))` for the maximum depth.

Marking Scheme:

(1 mark) Correct value of the best maximum depth for pre-pruning.

(d) Suppose that we want to post-prune `tree-full` using the minimum expected information gain criterion.

For post-pruning, grow a full tree first. Define a value for the minimum expected information gain. Next, keep track of all the nodes that only has leaf nodes as its descendants. Let's call these nodes `leaf parents`. For each `leaf parent` node, if the expected information gain is less than the pre-defined value, then delete its children and convert this node to a leaf node with majority decision. `Repeat` this process until the expected information gain at every `leaf parent` node is greater than or equal to the pre-defined value.

What is the best value for the minimum expected information gain through 10-fold cross-validation? Use the range `list(np.arange(0, 1.1, 0.1))` for the minimum expected information gain.

Marking Scheme:

(1 mark) Correct value of the best minimum expected information gain for post-pruning.

(e) In the previous parts, you have experimented with several ways of building a decision tree for the dataset. If you had a choice, what is the best strategy you can use to generate a decision tree for this dataset? You can choose one of the strategies in this assignment or you can think of any other strategies.

Explain your strategy and justify why it is your best choice.

Marking Scheme:

(2 marks) A reasonable explanation