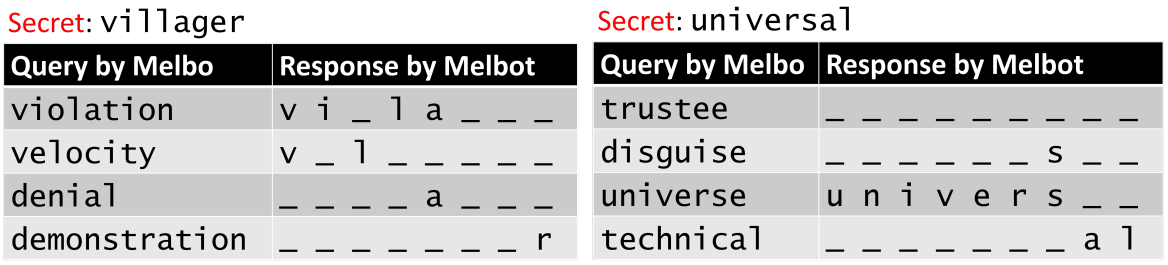

Secret: villager

Query by Melbo	Response by Melbot
violation	v i _ l a _ _ _
velocity	v _ l _ _ _ _ _
denial	_ _ _ _ a _ _ _
demonstration	_ _ _ _ _ _ _ r

Secret: universal

Query by Melbo	Response by Melbot
trustee	_ _ _ _ _ _ _ _ _
disguise	_ _ _ _ _ _ s _ _
universe	u n i v e r s _ _
technical	_ _ _ _ _ _ _ a l

Figure 1: Some examples of the response Melbot would give to Melbo's queries on the words villager and universal. Note that if the query word is longer than the secret word, the extra characters are ignored. If the query word is shorter than the secret word, the remaining positions are padded with underscores. A character is revealed only if both the query word and the secret word have that character at the same location.

Problem 2.1 (Playing with the Melbot). Melbo has purchased a new toy to improve his vocabulary called the *Melbot*. The toy lets Melbo play a word-guessing game. There is a dictionary of N words that is known to both Melbo and Melbot. In each round of this game, the following steps are taken:

1. Melbot chooses one of the words from the dictionary as secret say `villager` for this round.
2. Melbot tells Melbo the number of characters in that word by sending Melbo the string `"_ _ _ _ _ _ _ _"` (without the quotes)
3. The following steps are repeated until the round is terminated by either Melbo or Melbot.
   (a) Melbo guesses a word from the dictionary by taking an index i between 0 and $N-1$ and sending it to Melbot as a query. For example, Melbo chooses the index 4972 that corresponds to the word `violation`.
   (b) Melbot check's Melbo's query to see if it is a valid one. If the query index is invalid i.e. $i \notin [0, N-1]$, then Melbot assumes that Melbo no longer wants to continue and terminates the round.
   (c) If the query index is valid, Melbot checks if Melbo's guess is indeed the secret word and if so, the round is terminated and the win count is incremented by 1.
   (d) If the query word is not the secret word and Melbo has made too many queries in this round (the limit is $Q = 15$ queries per round), then Melbot terminates the round.
   (e) If the query word is not the secret word but Melbo has not reached the query limit, then Melbot reveals to Melbo all characters that are common to the query word and the secret word (only if those characters are in the correct location as well). For example, if the secret word is `villager` and the query word is `violation`, then Melbot would return the string `"v i _ l a _ _ _"` (without the quotes). See the figure for more examples.

Melbo plays N rounds of this game, once with each word in the dictionary. Melbo's performance is judged based on the number of words guessed correctly within the query limit (the win count divided by N), the average number of queries asked per round and some other measures described below. Note that at any point, Melbo can ask any word as a query so long as it is in the dictionary. It is not necessary that if Melbo is at a certain node in the decision tree, then only one of the words that reached that node must be asked – words that did not reach that node may also be asked if they help discriminate between words that reached that node.

Before proceeding, **please take a look at the Google Colab validation code and the dummy submission file `dummy_submit.py`** provided with the assignment package to get an idea of how the above protocol works and how your evaluation would be done.

Note that Melbo can also terminate a round by setting the `is_done` flag but that is automatically done when Melbo reaches the leaf of the decision tree. There is no way for you to specify this flag in your solution. When you are writing a code and wish to terminate a round, you should instead send an illegal index to signal termination of the round.

Your Data. We have provided you with a dictionary of $N = 5167$ words. Each word in the dictionary is written using only lower-case Latin characters i.e. `a - z`.

Your Task. You need to develop a decision tree learning algorithm that can play this game. However, note that your algorithm will be tested on a secret dictionary that we have not revealed to you. More on this later. The following enumerates the 2 parts to the question. Part 1 needs to be answered in the PDF file containing your report. Part 2 needs to be answered in the Python file.

1. Give detailed calculations explaining the various design decisions you took to develop your decision tree algorithm. This includes the criterion to choose the splitting criterion at each internal node (which essentially decides the query word that Melbo asks when that node is reached), criterion to decide when to stop expanding the decision tree and make the node a leaf, any pruning strategies and hyperparameters etc. (10 marks)

2. Write code implementing your decision tree learning algorithm. You are not allowed to use any library other than numpy. This means that even use of scikit-learn is prohibited. Use of other libraries such as scipy, skopt, etc is also forbidden. Submit code for your chosen method in `submit.py`. Your code must implement a `my_fit()` method that takes a dictionary as a list of words and returns a trained decision tree as a model. The trained decision tree as a model should be a tree object. Every node in that tree should be a node object. There is no restriction on what attributes the tree object or the node objects may have and what methods those classes implement (i.e. feel free to implement your own Tree and Node classes) but the Node class must implement at least 2 methods:

   (a) Every non-leaf node should implement a `get_child()` method that takes a response and decides which child node to move to.

   (b) Every node (leaf as well as non-leaf) should implement a `get_query()` method that tells what query Melbo should ask when at that node. For leaf nodes, this would be the final query in that round after which the round will be terminated.

   We will evaluate your method on a different dictionary than the one we have given you and check how good is the algorithm you submitted (see below for details). **Please go over the Google Colab validation code and the dummy submission file** `dummy\_submit.py` to clarify any doubts about data formats, protocol etc. (30 marks)

Part 1 needs to be answered in the PDF file containing your report. Part 2 needs to be answered in the Python file.

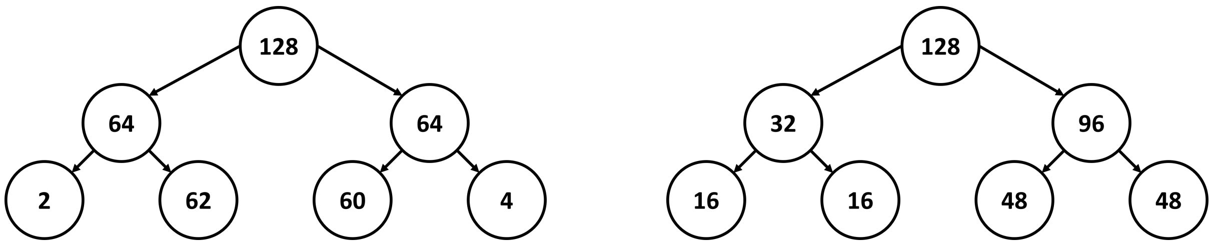

Figure 2: Greedy splitting can be suboptimal. In the example on the left, the split at the root was very nice (entropy 6.0) but it eventually led to a bad set of grandchildren (entropy 5.73). In the example on the left, the split at the root was not that nice (entropy 6.19 ¿ 6.0) but the set of grandchildren it produced had lower entropy (entropy 5.19).

Evaluation Measures and Marking Scheme. We have two dictionaries with us, a public one and a secret one. The public one has been given to you in the assignment package but the secret one is safe with us. The query limit per round will be $Q = 15$ for both dictionaries i.e. Melbo can make upto 15 queries in each round before Melbot terminates the round. Both dictionaries will be composed of words that are written using only lower-case Latin characters i.e. `a - z`. However, whereas the public dictionary is composed of English words, the secret dictionary may be in another language e.g. Spanish, French, Italian, Hindi, Bangla, etc but there will be no accents or special characters – only lower-case Latin characters will be used. Thus, do not assume that character frequencies will be the same e.g. the character `e` may not be the most common in the secret dictionary. However, your `my_fit()` function can do operations to find out which is the most common character etc. The number of words in the secret dictionary will be between 5000 and 6000 and the length of each word will be between 4 and 15 characters.

1. How fast is your `my_fit` method able to finish training (5 marks)
2. What is the on-disk size of your learnt model (after a pickle dump) (5 marks)
3. How many queries does your model make per round on average (15 marks)
4. How high is the win rate offered by your model (on what fraction of the secret dictionary is your model able to guess the word correctly within 15 queries)? (5 marks)

Thus, the total marks for the code evaluation is 30 marks. For more details, please check the evaluation script on Google Colab (linked below). Once we receive your code, we will execute the evaluation script to award marks to your submission.

You would have noticed the relatively less marks for win rate. This is because it is pretty simple to get a high win rate simply by growing the tree larger and larger. The challenge is to get a high win rate will as few queries as possible.

Possible Solution Strategies. You may of course follow standard information-theoretic entropy reduction algorithms such as we have discussed in class but you will need to implement these yourself. However, there is much more you can experiment with:

1. A slightly expensive but solid strategy is to implement *lookahead.* Notice that instead of choosing the split that gives the maximum entropy reduction right now (as we saw in class), a better strategy is too choose a split that will result in maximum entropy reduction two levels or three levels from now. The figure above shows an example where a greedy split may end up giving poorer result. However, implementing large lookahead can quickly

blow up training time. If you at all wish to experiment with lookahead, do so once the number of words in the nodes has reduced to smaller numbers i.e. don't do lookahead at the root – do so a few levels down the root.

2. You may ditch entropy altogether and instead directly aim for minimizing the number of queries it takes to correctly guess the word and win rate. Suppose you have 100 words in a node at depth 5. Find out all subtrees that can be built from this node onward by trying out all possible queries at this node, then all possible queries at this node's children and so on for 10 levels (we have to stop at 10 levels since we cannot make more than 15 queries and the node at which we started was already at level 5). Of all these subtrees, choose the one that has the highest value of average win rate across the 100 words or lowest average number of queries asked for those 100 words before success, or some combination thereof. Again, do not experiment with this technique close to the root. Try this only when the node sizes have become manageably small.

3. You may experiment with splits that are not just based on entropy reduction or expected time to success but that also offer a balanced split. Think of the balance of a split being a regularization term that prevents you from choosing an imbalanced split that simply reduces entropy a lot.

Validation on Google Colab. Before making a submission, you must validate your submission on Google Colab using the script linked below.

Link: `https://colab.research.google.com/drive/1_qlxLBXcauDRwHGIZUcppUEMtV92SR9j?usp=sharing`

Validation ensures that your `submit.py` does work with the automatic judge and does not give errors due to file formats etc. Please use IPYNB file at the above link on Google Colab and the dummy secret dictionary (details below) to validate your submission.

Please make sure you do this validation on Google Colab itself. **Do not download the IPYNB file and execute it on your machine – instead, execute it on Google Colab itself.** This is because most errors we encounter are due to non-standard library versions etc on students personal machines. Thus, running the IPYNB file on your personal machine defeats the whole purpose of validation. You must ensure that your submission runs on Google Colab to detect any library conflict. **Please note that there will be penalties for submissions which were not validated on Google Colab and which subsequently give errors with our automated judge.**

Dummy Submission File and Dummy Secret Dictionary. In order to help you understand how we will evaluate your submission using the evaluation script, we have included a dummy secret dictionary and a dummy submission file in in the assignment package itself (see the directory called `dummy`). However, note that the dummy secret dictionary is just a copy of the public dictionary. However, the dummy submission file may prove useful to you since it contains code to implement a fully functional decision tree for the problem. However, please note the following points

1. You are allowed to use a different Tree and Node class implementation in your submission so long as your Node class implements the `get_child()` and `get_query()` methods as mentioned above.

2. The dummy implementation stores a lot of information at each node e.g. node history etc. Feel free to discard this information if you are not using it to decide your query since it will just increase your model size.