CS771 Assignment 2

The Brainiacs

Aishwarya Srivastava (200064)

Akanksha Singh (200070)

Anushka Panda (200174)

Kshiteesh Bhardwaj (200525)

Raj Vardhan Singh (200760)

1 Playing with the MELBOT

1.1 Approach

Our code is based upon a decision tree algorithm for word guessing games. Nodes in the code represent questions that can be asked about the secret word. Each internal node of the tree asks a question about the secret word, and each leaf node of the tree represents a guess for the secret word. The algorithm proceeds by first splitting the data based upon the lengths of the word given in the dictionary. Then further splitting is done in a way that minimizes the uncertainty about the secret word to get maximum information gain at each step given the responses to the questions.

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

1.2 Implementation

This is a simple implementation of a decision tree algorithm. In the my_fit function, we create a Tree object and train it using a list of words given in the dictionary. The fit method in the Tree class creates a root node and trains it recursively with a given set of words. The Node class represents a node in the tree. We first split the data based upon length of the words given in the dictionary. After this division, we get child nodes of the root node, all of which are of

Preprint. Under review.

different lengths. Proceeding further, each non-leaf node selects a word from the list of words to use as a query. The query word is chosen based upon the maximum reduction in entropy or maximum information gain that can be achieved by splitting the current set of words using that word. The entropy reduction is calculated as the sum of the number of characters that are revealed by the query for each word in the current set of words.
The entropy is calculated as : $H(S) = -\sum_{i=1}^{|Y|} p_i \log_2(p_i)$ where,

- $H(S)$ is the entropy of the set S
- Y is the set of all possible class labels
- $|Y|$ is the cardinality of the set Y (i.e., the number of unique class labels)
- p_i is the proportion of instances in S that belong to class i, i.e., $p_i = \frac{|x \in S : x \text{ belongs to class } i|}{|S|}$ (where $|S|$ is the total number of instances in S)

Suppose, asking a question splits the set S into K subsets $S_1, S_2, ..S_K$ such that $S_i \cap S_j = \phi$ $if\, i \neq j$ and $U_{k\in[K]} S_k = S$, then the entropy of the collection of sets is defined as:
$H(S) = \sum_{k\in[K]} \frac{n_k}{n} H(S_k)$, where $H(S_k)$ denotes the entropy of the subset S_k, n_k denotes the number of elements in S_k, and n is the total number of elements in S.
The word that results in the maximum entropy reduction is chosen as the query word. We recursively split the data based on the selected feature to get maximum information gain and threshold until the decision stump becomes pure i.e. it achieves a perfect classification on the training set, meaning that it perfectly separates the training data into their respective classes with zero training error. We make these pure nodes as leaves and do not split them further.

The reveal method of the Node class takes two words as input and returns a string that represents the overlap between the two words. The process_leaf method returns the first word in the list of words, and the process_node method selects a query word as described above and splits the current set of words into subsets based on the revealed characters for each word.

1.3 Explanation of code

The `my_fit` function is a function that generates an instance of the Tree class and fits the tree to the data. The function takes a list of words as input and returns an instance of the Tree class. The Tree class represents the decision tree. The `_init_` method initializes the tree with a root node and sets the minimum leaf size and maximum depth of the tree. The fit

method creates a root node and recursively builds the tree using the Node class. It takes a list of words as input and returns the fitted tree.
The Node class defines each node in the decision tree. Each node has a depth, a link to its parent, a link to all the words in the data set, a link to the words that reached that node, a dictionary to store the children of a non-leaf node, and a flag to indicate if the node is a leaf node. As the node is reached, the `get_query` method generates the query which is sent to the node, and the `get_child` method takes the response and selects one of its children based on that response .

The `process_leaf` method is a dummy leaf action that just returns the first word. The reveal method takes in a word and a query and returns a string that reveals the intersections between the query and the word. The method finds the best query to split the data based on the revealed intersections between the query and the words. The fit method recursively builds the decision tree. Nodes that are too small or too deep are classified as leaf nodes. According to the responses to the questions, the splitting minimizes the uncertainty about the secret word.
The fit method of the Node class takes the following arguments:
`all_words`: a list of all the words in the dictionary
`my_words_idx`: a numpy array of indices that represent the words that reached this node
`min_leaf_size`: the minimum number of words needed to form a leaf node
`max_depth`: the maximum depth of the tree
`fmt_str`: a string used for printing the tree
`verbose`: a boolean flag indicating whether to print verbose output

First, the method sets `all_words` (all words) and `my_words` (my words) to the provided values. Then, it checks whether the node should be a leaf based on the `min_leaf_size` and `max_depth` parameters. If the node is a leaf, it sets the `is_leaf` attribute to True and returns the result of calling the `process_leaf` method, which in this case simply returns the first word in `my_words_idx`. If the node is not a leaf, the method calls the `process_node` method to determine how to split the node. The process_node method returns a tuple consisting of the index of the query word and a dictionary that maps query responses to sets of word indices. The fit method then sets the `query_idx` attribute of the node to the index of the query word and iterates over the items in the split dictionary to create child nodes.
During each key-value pair generation in the dictionary, the method creates a child node, sets

the history attribute of the child to a copy of the history attribute of the current node plus the current query, and recursively calls the fit method on the child node, passing in the appropriate arguments. The fit method returns the node after creating all of its child nodes and setting their `is_leaf` attributes to False.

This way, the decision tree is recursively built and optimized to minimize uncertainty about the secret word given the responses to the questions.

1.4 Dataset

We have been provided with a dictionary of N = 5167 words.

1.5 Libraries used

- Numpy

2 Code implementing our decision tree learning algorithm

...can be checked out in `submit.py :)`