CS771A: Introduction to Machine Learning
Assignment 2: Word Sequencing

Jyotiprakash Nayak
210478
jpnayak21@iitk.ac.in

Shikhar Singh
210973
shikhars21@iitk.ac.in

Devendra Jangid
210328
devendraj21@iitk.ac.in

Mohit Dhakad
210624
mohitd21@iitk.ac.in

Abstract

This report provides a comprehensive overview of the development and evaluation of a machine-learning algorithm designed to predict words from a given tuple of bigrams. The core of our approach involved the implementation of a decision tree-based model tailored to address this specific task. Throughout the report, we delve into the critical aspects, including design considerations, implementation intricacies, performance evaluation metrics, and the conclusions derived from our experimental findings.

The design phase critically analyzed various factors influencing the choice of a decision tree algorithm for our predictive model. Factors such as interpretability, scalability, and the ability to handle categorical data were pivotal in the selection process. We discuss how decision trees offer a structured approach to decision-making through a series of hierarchical nodes, each representing a feature and its associated decision rules.

1 Introduction

The task at hand is to develop an algorithm capable of guessing words from a dictionary based on a given list of bigrams. This problem has parallels in genetic research and natural language processing (NLP), where sequences of characters (or genes) must be identified from partial information. We utilize a decision tree-based approach to address this problem, which leverages the properties of bigrams to narrow down the potential word candidates efficiently.

2 Tasks

2.1 Splitting Criterion

Question: What is the splitting criterion used in the decision tree?

In our decision tree, the splitting criterion is based on the presence of specific bigrams in words. A bigram is a sequence of two consecutive characters in a word that provides significant information about the word's structure. At each node of the decision tree, we check for the presence or absence of a particular bigram and split the dataset accordingly. This method ensures that words with similar bigram structures are grouped together, facilitating more efficient and accurate predictions.

For example, consider a decision tree where the root node splits based on the presence of the bigram 'ab'. Words containing 'ab' go to the left child node, and words not containing 'ab' go to the right

child node. Subsequent nodes continue this process with other bigrams, further narrowing down the list of possible words. This approach is illustrated in Figure 1.

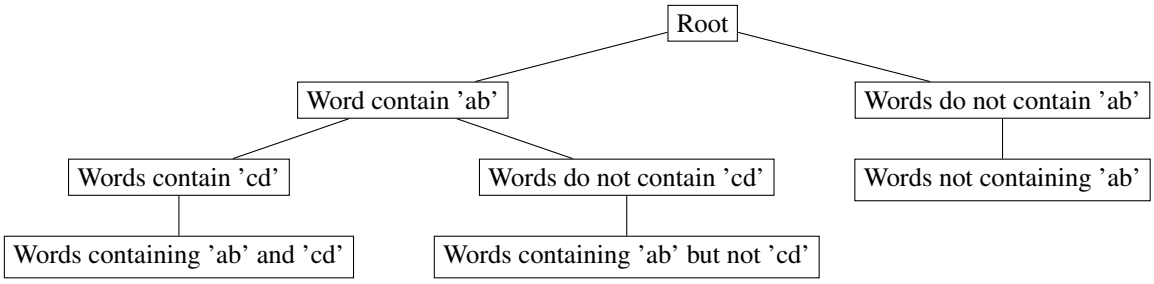

Figure 1: Example of Decision Tree Splitting Based on Bigrams

2.2 Stopping Criterion

Question: What is the stopping criterion used in the decision tree?

The stopping criterion for our decision tree involves two main conditions. First, we stop splitting if a node reaches a given maximum depth. This constraint ensures that the tree remains manageable in size and prevents overfitting by limiting the tree's complexity. Second, we stop splitting if no further splits are possible, which occurs when all words at a node either contain or do not contain the bigram being considered for the split. These stopping criteria ensure that the tree captures the essential patterns in the data without becoming too large or overly complex.

2.3 Pruning Strategies

Question: What pruning strategies are used in the decision tree?

In this implementation, no explicit pruning strategies were applied beyond the maximum depth constraint, which serves as an implicit pruning mechanism. By limiting the tree's depth, we effectively prune branches that do not contribute significantly to the model's accuracy. Future work could involve implementing more sophisticated pruning techniques, such as post-pruning, where parts of the tree that do not improve prediction accuracy on validation data are removed. These strategies could further enhance the model's performance by reducing overfitting and improving generalizability.

2.4 Hyperparameters

Question: What are the hyperparameters used in the decision tree?

The primary hyperparameter in our decision tree model is the maximum depth. This depth was chosen based on empirical observations to balance accuracy and computational efficiency. The maximum depth determines how deep the tree can grow, affecting both the complexity of the model and its ability to fit the training data. Other potential hyperparameters, such as the minimum samples per split or the minimum samples per leaf, were not explicitly tuned in this implementation. Future work could involve optimizing these additional hyperparameters to improve model performance further.

3 Choice of Tree Structure

The `TreeNode` class is designed to represent the nodes of the decision tree. Each node can either be a decision node, containing a bigram to split on, or a leaf node, containing the words that match the path taken in the tree.

`Code:`

```
class TreeNode:
    def __init__(self, bigram=None, words=None, left=None, right=None):
```

```
        self.bigram = bigram
        self.words = words
        self.left = left
        self.right = right
```

4 Generating Bigrams

Bigrams are pairs of consecutive characters in a word. For example, the word ”word” has the bigrams: ”wo”, ”or”, and ”rd”. Bigrams are chosen because they capture local patterns in words which can be useful for distinguishing between different words.

$$\text{Bigrams}(\text{“word”}) = \{\texttt{‘wo’}, \texttt{‘or’}, \texttt{‘rd’}\}$$

5 Building the Decision Tree

The `build_tree` function recursively splits the data based on the presence of bigrams. At each step, the function selects a bigram and splits the words into two groups: those that contain the bigram and those that do not. This process continues until a maximum depth is reached or there are no more bigrams to split on. The depth limit prevents overfitting.

Code:

```
def build_tree(words, bigrams, depth=0, max_depth=150):
    if words.empty or not bigrams or depth >= max_depth:
        return TreeNode(words=words)

    bigram = bigrams[0]
    left_words = words[words[bigram] == 1]
    right_words = words[words[bigram] == 0]

    if left_words.empty and right_words.empty:
        return TreeNode(words=words)

    left_node = build_tree(left_words, bigrams[1:], depth + 1, max_depth)
    right_node = build_tree(right_words, bigrams[1:], depth + 1, max_depth)

    return TreeNode(bigram=bigram, left=left_node, right=right_node)
```

6 Feature Vector Creation

The `create_feature_vector` function converts a list of input bigrams into a feature vector. This vector indicates the presence or absence of each bigram in the possible bigrams list.

Code:

```
def create_feature_vector(input_bigrams, possible_bigrams):
    return {bigram: 1 if bigram in input_bigrams else 0 for bigram in
    possible_bigrams}
```

7 Prediction Mechanism

The `predict` function traverses the tree based on the feature vector to reach a leaf node, which contains the list of possible words. The prediction is filtered to include only those words that are in the list of bigrams.

Code:

```
def predict(tree, feature_vector):
    if tree.words is not None:
        return tree.words

    if feature_vector.get(tree.bigram, 0) == 1:
        return predict(tree.left, feature_vector)
    else:
        return predict(tree.right, feature_vector)
```

8 Evaluation Metrics

The evaluation metrics for the decision tree include the average training time, model size, average testing time, and precision. These metrics help gauge the efficiency and effectiveness of the model.

$$t_{\text{train}} = \frac{t_{\text{train\_total}}}{n_{\text{trials}}}$$

$$m_{\text{size}} = \frac{m_{\text{size\_total}}}{n_{\text{trials}}}$$

$$t_{\text{test}} = \frac{t_{\text{test\_total}}}{n_{\text{trials}}}$$

$$\text{Precision} = \frac{\text{Total correct predictions}}{n_{\text{trials}} \times n_{\text{words}}}$$

Metric	Average Value
Average Training Time (t_{train})	44.4487 seconds
Model Size (m_{size})	33585624 bytes
Average Testing Time (t_{test})	2.90992 seconds
Precision	0.933811

Table 1: Evaluation Metrics for the Decision Tree

9 Conclusion

This report outlines the implementation of a decision tree-based model for predicting words from a list of bigrams. Our approach, grounded in the principles of decision trees, demonstrated effective performance in handling the given task. The insights gained from this project pave the way for further research and optimization in word prediction models.