CS771 Assignment 2

Adarsh Kumar
210048
Dept. of Civil Engineering
adarshkum21@iitk.ac.in

Aryan
210198
Dept. of Chemical Engineering
aryan21@iitk.ac.in

Narottam Kumar Pankaj
210652
Dept. of Chemical Engineering
narottamkp21@iitk.ac.in

Paras
210698
Dept. of Chemistry
paras21@iitk.ac.in

Rajat Phogat
210814
Dept. of Chemical Engineering
rajatp21@iitk.ac.in

Sumit Saurabh
211075
Dept. of Mechanical Engineering
sumits21@iitk.ac.in

Umesh Rathore
211125
Dept. of Mechanical Engineering
umeshr21@iitk.ac.in

Abstract

This document describes the methodology and approach used by our group in the Assignment-2 of the course CS771: An Introduction to Machine Learning, offered at IITK in the semester 2023-24-Summer.

PROBLEM STATEMENT

Give detailed calculations explaining the various design decisions you took to develop your ML algorithm. For instance, if you did use a decision tree algorithm, this includes the criterion to choose the splitting criterion at each internal node (e.g. if a certain bigram is present in the bigram list or not), criterion to decide when to stop expanding the decision tree and make the node a leaf, pruning strategies, hyperparameters etc.

1 Introduction

The task involves developing an ML algorithm to predict a word given a list of up to 5 bigrams. The challenge is to correctly identify the word while dealing with potential ambiguities due to sorting, duplicate removal, and truncation of bigrams. We chose a Decision Tree Classifier due to its interpretability and capability to handle categorical data efficiently.

2 Data Preparation and Feature Extraction

2.1 Extracting Bigrams:

- **Function:** `extract_bigrams(word)`
- **Description:** For each word, generate all possible bigrams. This involves creating pairs of adjacent characters.
- **Exapmle:** For the word "optional", the bigrams are `['op', 'pt', 'ti', 'io', 'on', 'na', 'al']`.

Preprint. Under review.

2.2 Preprocessing Bigrams:

- **Function:** `preprocess_word(word)`
- **Description:** Process each word's bigrams by sorting them lexicographically, removing duplicates, and retaining only the first 5 bigrams.
- **Example:** For the word "optional", the processed bigrams are `['al', 'io', 'na', 'on', 'op']`.

2.3 Preparing the Dataset:

- **Function:** `prepare_dataset(dictionary)`
- **Description:** For each word in the dictionary, generate and preprocess the bigrams, resulting in a dataset where each word is associated with its processed bigrams.
- **Output:** A list of tuples, each containing a word and its bigram list.

3 Model Design: Decision Tree Classifier

3.1 Choice of Model:

- **Model:** Decision Tree Classifier
- **Rationale:** Decision trees are well-suited for categorical feature spaces and can handle the presence or absence of specific bigrams effectively. They are also interpretable, allowing us to understand the decision-making process.

3.2 Feature Representation:

- **Bigram Features:** The set of all unique bigrams in the dictionary forms the feature space. Each word is represented by a binary feature vector indicating the presence (1) or absence (0) of each bigram.
- **Example:** For a dictionary containing words like "optional", "proportional", etc., the feature vector for "optional" based on the top bigrams might be `[1, 1, 1, 1, 1, 0, 0, ..., 0]`.

3.3 Splitting Criterion:

- **Criterion:** Gini Impurity or Entropy
- **Description:** At each node, the decision tree algorithm splits the data based on the presence or absence of a bigram to maximize the homogeneity of the resulting nodes. Gini impurity and entropy are common criteria that measure the impurity of the node.

3.4 Stopping Criterion:

- **Minimum Samples per Leaf:** The node is not split further if it contains fewer than a specified number of samples (e.g., 5). This prevents overfitting and ensures that each leaf node has enough samples to make a reliable prediction.
- **Maximum Depth:** The maximum depth of the tree is limited (e.g., 10) to prevent overfitting. Deeper trees can capture more complex patterns but are also more prone to overfitting.

3.5 Pruning Strategies:

- **Cost Complexity Pruning:** Post-pruning is applied to remove branches that have little importance. This is done by balancing the depth of the tree against its complexity to avoid overfitting.
- **Cross-Validation:** Hyperparameters such as tree depth and minimum samples per leaf are tuned using cross-validation on the training set to ensure the model generalizes well to unseen data.

4 Model Training and Prediction

4.1 Training:

- **Function:** `fit(dictionary)`
- **Process:** Convert words to their feature vectors based on bigrams, fit the decision tree classifier to this dataset.
- **Output:** A trained decision tree model capable of predicting words based on bigram presence.

4.2 Prediction:

- **Function:** `predict(bigram_tuple)`
- **Process:** Convert the input bigram tuple to a feature vector, use the trained model to predict the probabilities of each word being the correct one, and return the top 5 predictions.
- **Scoring:** Precision is calculated by dividing the score by the number of guesses made if the correct word is present in the predictions.

5 Conclusion

The designed algorithm effectively addresses the challenge of predicting words based on their bigrams. By preprocessing the data, extracting relevant features, and using a decision tree classifier with appropriate stopping and pruning strategies, we ensure a balance between accuracy and generalization. The hyperparameters are tuned using cross-validation, and the final model is evaluated based on its precision in making correct predictions. This approach provides a robust solution to the word sequencing problem while handling potential ambiguities due to the processing of bigrams.