CS771 : Introduction to Machine Learning

Assignment - 1

Team: Profanities

Aryan Prajapati
210203
Material Science and Engineering
aryanp21@iitk.ac.in

Manish Kumar
210584
Material Science and Engineering
manishkv21@iitk.ac.in

Manasvi Jeevan Kweera
210582
Earth Sciences
manasvi21@iitk.ac.in

Kumar Gunjan
210542
Material Science and Engineering
kgunjan21@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

The task is to create a machine learning algorithm that can predict a word from a list of up to 5 bigrams. The challenge is to accurately identify the word despite sorting, removing duplicates, and truncating the bigrams. We selected a Decision Tree Classifier because it is easy to understand and works well with categorical data.

1 Introduction

The task is to develop a machine learning algorithm to predict a word from a list of up to 5 bigrams. The challenge is to identify the word correctly despite the sorting, duplicate removal, and truncation of the bigrams. We chose a Decision Tree Classifier because it's easy to understand and handles categorical data well.

Code zip file

2 Data Preparation and Feature Extraction

2.1 Extracting Bigrams:

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

2.2 Preprocessing Bigrams:

- **Function:** `preprocess_word(word)`
- **Description:** Sort each word's bigrams alphabetically, remove duplicates, and keep 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, creating a dataset where each word is linked to 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 good for categorical features and can effectively handle the presence or absence of specific bigrams. They are also easy to understand.

3.2 Feature Representation:

- **Bigram Features:**The feature space is made up of all unique bigrams in the dictionary. Each word is represented by a binary feature vector that shows 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:** The decision tree splits the data at each node based on the presence or absence of a bigram to maximize the similarity within the resulting nodes. Gini impurity and entropy are used to measure node impurity.

3.4 Stopping Criterion:

- **Minimum Samples per Leaf:** A node is not split further if it has fewer than a specified number of samples (e.g., 5). This prevents overfitting.
- **Maximum Depth:** The tree's depth is limited (e.g., 10) to avoid overfitting. Deeper trees can capture more complex patterns but are also more prone to overfitting.

3.5 Pruning Strategies:

- **Cost Complexity Pruning:** Post-pruning removes branches that add little value by balancing tree depth against complexity to prevent overfitting.
- **Cross-Validation:** Hyperparameters like tree depth and minimum samples per leaf are tuned using cross-validation to ensure the model generalizes well to new data.

4 Model Training and Prediction

4.1 Training:

- **Function:** `fit(dictionary)`
- **Process:** Convert words to their bigram-based feature vectors and 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, 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 algorithm effectively predicts words based on their bigrams. By preprocessing the data, extracting relevant features, and using a decision tree classifier with proper stopping and pruning strategies, we achieve a balance between accuracy and generalization. Hyperparameters are tuned using cross-validation, and the final model is evaluated for precision in its predictions. This approach provides a robust solution to the word prediction problem, handling potential ambiguities in bigram processing.