CS771: Introduction to Machine Learning
Assignment 2

Team: Neurons

Rupesh Kumar Meena : 210881, EE
rupeshkm21@iitk.ac.in

Keerthi S : 210504, CHM
keerthis21@iitk.ac.in

Aditya Neelabha : 190064, Mech
adityan19@iitk.ac.in

Deepak Kumar : 220332, Math
deepakkr22@iitk.ac.in

Abstract

This document presents the submission from team **Neurons** for Assignment 2. This file outline and explain the various design decisions involved in developing the machine learning algorithm, including the decision tree algorithm, the criteria for splitting nodes, stopping conditions, pruning strategies, and hyperparameters.

Design Decisions and Detailed Calculations

1. Feature Extraction and Representation

Bigram Extraction:

- The function `create_bigrams(word)` generates all possible bigrams from a given word.
- The function `unique_sorted_bigrams(word)` processes these bigrams by removing duplicates, sorting them lexicographically, and retaining only the first 5 bigrams.

Justification:

- This mirrors the constraints of the problem where only the first 5 lexicographically sorted bigrams are considered, allowing us to mimic the test conditions during training.

2. Vectorization

Vectorization of Words:

- The function `vectorize_word(word, all_bigrams, bigram_index)` converts a word into a feature vector based on the presence of bigrams.
- Each word is represented as a binary vector where each element indicates the presence (1) or absence (0) of a particular bigram from the list of all unique bigrams.

Justification:

- Using binary vectors is a straightforward and efficient way to represent the presence of bigrams in each word, enabling the decision tree classifier to easily learn patterns from the data.

3. Pruning Rare Bigrams

Prune Rare Bigrams:

- The function `prune_rare_bigrams(words, threshold=5)` filters out bigrams that occur less than a specified threshold (default is 5).
- This helps in reducing the dimensionality of the feature space and removes noise.

Justification:

- Rare bigrams are likely not useful for classification and can introduce noise, making the model more complex without adding significant predictive power.

4. Decision Tree Classifier

Classifier Choice:

- A `DecisionTreeClassifier` from `scikit-learn` is used for the task.

Splitting Criterion:

- **Criterion:** gini
- The Gini impurity measure is used to decide how to split at each node.
- Gini impurity measures the frequency at which any element of the dataset would be randomly mislabeled if it was randomly labeled according to the distribution of labels in the subset.

Stopping Conditions:

- **Max Depth:** `max_depth=10`
  - Limits the depth of the tree to prevent overfitting.
  - A maximum depth of 10 is chosen to balance the model's ability to capture complex patterns while avoiding overfitting to the training data.
- **Min Samples Split:** `min_samples_split=5`
  - A node must have at least 5 samples to be split further.
  - This prevents the tree from making splits that do not have enough data to provide meaningful insights.
- **Min Samples Leaf:** `min_samples_leaf=3`
  - A leaf node must have at least 3 samples.
  - This ensures that the leaf nodes are not too specific and can generalize better.

Pruning Strategy:

- No explicit post-pruning strategy is implemented, but pre-pruning is achieved through the stopping conditions (max depth, min samples split, min samples leaf).

Justification:

- Decision trees are chosen for their interpretability and ability to handle categorical features well.
- The specific hyperparameters (gini criterion, max_depth, min_samples_split, and min_samples_leaf) are set to prevent overfitting and to ensure the model generalizes well to unseen data.

5. Training and Prediction

Training (`my_fit` function):

- The training process involves generating the feature vectors for each word and fitting the decision tree classifier on these vectors.

Prediction (`my_predict` function):

- The prediction process involves converting the input bigram list into a feature vector and using the trained decision tree classifier to predict possible words.
- The function ensures that no more than 5 guesses are returned, aligning with the problem constraints.

Conclusion

By following the above design decisions, we ensure that the machine learning model is trained effectively to guess words based on their bigram lists. The use of a decision tree classifier with appropriate hyperparameters and feature extraction methods enables the model to learn and generalize from the training data while adhering to the problem constraints. The decisions on splitting criteria, stopping conditions, and pruning strategies are critical to balancing model complexity and performance, ensuring that the model can perform well on unseen data.