CS771: Introduction to Machine Learning
Assignment - 2
Team: StakeInsight

Dhananjay Dixit
220351
Dept. of Aerospace Engineering
IIT Kanpur
dhananjayd2222@iitk.ac.in

Sahil Kumar
22936
Dept. of Mechanical Engineering
IIT Kanpur
sahilkur22@iitk.ac.in

Pratham Garg
220808
Department of Electrical Engineering
IIT Kanpur
prathamg22@iitk.ac.in

Ritesh Hans
220893
Dept. of Computer Science and Engineering
IIT Kanpur
riteshhans22@iitk.ac.in

Aryan Gautam
220222
Dept. of Aerospace Engineering
IIT Kanpur
garyan22@iitk.ac.in

Abstract

This is our solution to the second assignment of the CS771 course. The task requires developing an ML algorithm to predict a sequence of words given a list of bigrams. The decision tree classifier was chosen for its simplicity and interpretability. Below are the detailed calculations and design decisions made during the development of the algorithm.

Preprint. Under review.

1 Data Preprocessing

1.1 Bigram Generation

Function: `create_bigrams(term, limit=None)`

- **Calculation:**
  - Given a word term of length n, the bigrams are generated by taking all consecutive pairs of characters. The number of possible bigrams is $n - 1$.
  - **Example:** For the word "hello", the bigrams are ["he", "el", "ll", "lo"].
- **Design Decision:**
  - Limit the number of bigrams to 5 to avoid high dimensionality.

1.2 Multi-Hot Encoding

Function: `generate_multi_hot(terms)`

- **Calculation:**
  - For each word, create a binary vector (multi-hot vector) representing the presence or absence of bigrams in the vocabulary.
  - Vocabulary of bigrams is created from all unique bigrams across the dataset.
  - **Example:** If the vocabulary of bigrams is ["he", "el", "ll", "lo", "oo"], the word "hello" would be encoded as [1, 1, 1, 1, 0].
- **Design Decision:**
  - Use multi-hot vectors to represent words, which allows the decision tree to learn patterns based on the presence of bigrams.

2 Model Training

2.1 Entropy and Information Gain

Entropy: Entropy is a measure of impurity or randomness in a set of examples.

$$\text{Entropy}(S) = -\sum_{i=1}^{m} p_i \log_2 p_i \tag{1}$$

where S is the set of examples, m is the number of classes, and p_i is the proportion of examples in class i.

Information Gain: Information gain measures the reduction in entropy after splitting a dataset S on an attribute A.

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v) \tag{2}$$

where $\text{Values}(A)$ are the possible values of attribute A, S_v is the subset of examples in S for which attribute A has value v.

2.2 Classifier Selection

Model: `DecisionTreeClassifier`

- **Criterion:**
  - **Gini Impurity:** Measures the frequency at which any element of the dataset would be misclassified when it is randomly labeled according to the distribution of labels in the subset.

- **Formula:**

$$\text{Gini}(D) = 1 - \sum_{i=1}^{n} p_i^2 \tag{3}$$

where p_i is the probability of class i in subset D.

- **Design Decision:**
  - Gini impurity was chosen as it is computationally efficient and provides good performance for classification tasks.

2.3 Hyperparameters

- **Hyperparameters Used:**
  - `criterion='gini'`
  - `max_depth=50`
  - `min_samples_split=2`
  - `min_samples_leaf=1`
  - `random_state=0`
- **Design Decisions:**
  - **Max Depth:**
    - Limited to 50 to prevent overfitting and to ensure the tree is not excessively deep.
  - **Min Samples Split:**
    - Minimum number of samples required to split an internal node is set to 2, allowing the tree to split whenever possible.
  - **Min Samples Leaf:**
    - Minimum number of samples required to be at a leaf node is set to 1, ensuring all nodes are considered.
  - **Random State:**
    - Set to 0 for reproducibility of results.

3 Prediction

3.1 Bigram Vectorization

Function: `my_predict(model, bigrams)`

- **Calculation:**
  - Convert the input bigrams into a multi-hot vector using the same vocabulary as used during training.
  - **Example:** If the vocabulary is ["he", "el", "ll", "lo", "oo"] and the input bigrams are ["he", "el"], the vector would be [1, 1, 0, 0, 0].
- **Design Decision:**
  - Ensure the input vector matches the format used for training to maintain consistency.

3.2 Prediction Process

- **Prediction:**
  - The trained decision tree model predicts the word indices based on the input bigram vector.
  - The predicted indices are mapped back to their corresponding words.
  - Return the top 5 predictions.
  - **Example:**
    - Given an input vector, the model predicts the word indices with the highest probabilities and maps them back to words.

4 Stopping Criteria and Pruning

4.1 Stopping Criteria

- **Criterion:**
  - The tree stops expanding when it reaches the maximum depth (`max_depth=50`) or when a node has fewer than the minimum number of samples required to split (`min_samples_split=2`).

4.2 Pruning Strategies

- **Post-Pruning:**
  - Simplifies the tree after it has been fully grown by removing nodes that do not provide significant information gain.
  - This helps in reducing the model complexity and prevents overfitting.
- **Cost-Complexity Pruning:**
  - Balances the trade-off between tree complexity and its performance.
  - The cost complexity measure considers both the number of leaves and the error rate.

5 Results and Conclusion

- **Accuracy:**
  - The model achieved an accuracy of between 60% and 70% on the test set, which indicates a good performance for this problem.
- **Conclusion:**
  - The decision tree classifier effectively utilized the bigram patterns to distinguish between different words. Further improvements could include experimenting with more complex models or additional features to enhance accuracy.
  - The Python code for the second question has been successfully submitted.