CS771
Assignment 2

Group Name: ML EXPRESS

Name	Roll Number
Kumar Sambhav	210545
Mali Ashish Ramniwas	210574
Yogesh	211205
Sunny Munda	211077
Jahnvi Singh	210460

Question 1

1. Feature Extraction

The goal is to create a feature vector representation of each word using its bigrams.

Bigrams

A bigram is a sequence of two adjacent elements from a string of text. For example, the word "word" has the bigrams "wo", "or", and "rd".

Feature Vector

- For each word, generate all possible bigrams.
- Sort and deduplicate the bigrams.
- Select the top 5 bigrams for consistency across feature vectors.
- Represent the presence of each bigram in a binary feature vector.

Mathematical Explanation

Given a word w of length n:

- The number of possible bigrams is $n - 1$.
- Let $B(w)$ be the set of bigrams of w. The feature vector $\mathbf{v}(w)$ is defined as:

  $$\mathbf{v}(w) = [b_1, b_2, \ldots, b_k]$$

  where $b_i \in \{0, 1\}$ indicates the presence (1) or absence (0) of the i-th bigram in w, and k is the total number of unique bigrams in the dataset.

2. Machine Learning Model

A Random Forest classifier was chosen for its robustness and ability to handle the feature space effectively.

Random Forest:

- An ensemble method that constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees.
- It reduces overfitting by averaging multiple decision trees trained on different parts of the dataset.

Mathematical Explanation

- Let $\{T_1, T_2, \ldots, T_m\}$ be the set of decision trees in the random forest.

- The final prediction $\hat{y}$ for an input vector $\mathbf{x}$ is given by:

$$\hat{y} = \arg\max \sum_{i=1}^{m} 1(T_i(\mathbf{x}) = y)$$

  where 1 is the indicator function.

3. Splitting Criterion

The decision tree splits are based on the presence of bigrams in words.

Entropy

- Entropy is used to measure the impurity or randomness in the data. A node with lower entropy indicates higher purity.
- For a split based on a bigram b, the entropy H is calculated as:

$$H = -\sum_{c \in C} p(c) \log_2 p(c)$$

  where C is the set of classes, and $p(c)$ is the proportion of samples belonging to class c.

Information Gain

- The information gain IG of a split is the reduction in entropy from a parent node to the child nodes:

$$IG = H_{\text{parent}} - \sum_{i} \frac{N_i}{N} H_i$$

  where N_i is the number of samples in the i-th child node, and H_i is the entropy of the i-th child node.

4. Stopping Criterion

The decision tree stops growing when a certain condition is met, such as reaching a maximum depth or a minimum number of samples per leaf.

Stopping Conditions

- **Maximum Depth:** Prevents the tree from growing too deep and overfitting the training data.
- **Minimum Leaf Size:** Ensures each leaf has a sufficient number of samples to make reliable predictions.

Mathematical Explanation

- A node becomes a leaf if:

$$\text{depth} \geq \text{max\_depth} \quad \text{or} \quad |N_{\text{samples}}| \leq \text{min\_leaf\_size}$$

5. Pruning Strategies

Pruning reduces the complexity of the model and enhances generalization by removing nodes that provide little power in predicting the target variable.

Post-Pruning

- After the tree is fully grown, nodes that do not contribute significantly to the model's accuracy on validation data are pruned.
- This can be achieved by evaluating the performance of the tree on a validation set and removing nodes that do not improve accuracy.

Hyperparameters

- **n_estimators:** The number of trees in the forest.
- **max_depth:** The maximum depth of each tree.
- **min_samples_split:** The minimum number of samples required to split an internal node.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node.

Question 2

```python
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier


# Function to generate bigrams from a word
def generate_bigrams(word):
    return [word[i:i+2] for i in range(len(word)-1)]

# Function to process and select top bigrams
def process_bigrams(bigrams):
    bigrams = sorted(set(bigrams))
    return bigrams[:5]

# Function to generate features for a word
def generate_features(word):
    bigrams = generate_bigrams(word)
    processed_bigrams = process_bigrams(bigrams)
    return ' '.join(processed_bigrams)

def my_fit(dictionary):
################################
#  Non Editable Region Ending  #
################################

    # Generate features for the dictionary words
    features_list = [generate_features(word) for word in dictionary]

    # Encode features
    vectorizer = CountVectorizer(analyzer='word', token_pattern=r'\S+')
    X = vectorizer.fit_transform(features_list).toarray()

    # Encode labels
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(dictionary)
```

```
    # Train RandomForestClassifier
    rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_clf.fit(X, y)

    # Return the model as a dictionary
    model = {
        'vectorizer': vectorizer,
        'label_encoder': label_encoder,
        'classifier': rf_clf
    }

    return model


f my_predict(model, bigram_list):
##############################
 Non Editable Region Ending  #
##############################

    vectorizer = model['vectorizer']
    label_encoder = model['label_encoder']
    rf_clf = model['classifier']

    # Generate features for the input bigrams
    input_features = ' '.join(bigram_list)
    input_vector = vectorizer.transform([input_features]).toarray()

    # Predict probabilities for the input vector
    probabilities = rf_clf.predict_proba(input_vector)[0]

    # Get the top 5 indices with the highest probabilities
    top_indices = np.argsort(probabilities)[-5:][::-1]
```

```
# Get the corresponding top words
top_words = label_encoder.inverse_transform(top_indices)

return top_words.tolist()
```