

Experiment No :-8. Sentiment analysis using Naive Bayes on some published dataset IMDb Movie Reviews Dataset using bag-of-words model. Implement with 1-gram and 2-gram. Show results with and without Laplace (+1 smoothing). You can remove unknown words in the test dataset.

```
import nltk
import shutil
import os

# Define the path where you want to store NLTK data
nltk_data_path = os.path.expanduser('~/.nltk_data')


# Remove the directory if it exists (optional, for a clean installation)
if os.path.exists(nltk_data_path):
    shutil.rmtree(nltk_data_path)

# Create the directory if it doesn't exist
os.makedirs(nltk_data_path, exist_ok=True)

# Set the NLTK data path
nltk.data.path.append(nltk_data_path)

# Download the 'punkt_tab' resource to the specified path
nltk.download('punkt_tab', download_dir=nltk_data_path)

# Now you can use word_tokenize
from nltk.tokenize import word_tokenize
print(word_tokenize("Hello world!"))
```

 [nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
['Hello', 'world', '!']

```
import nltk
import random
import re
import math
from nltk.corpus import movie_reviews
from collections import defaultdict, Counter
from sklearn.model_selection import train_test_split

# Download dataset if not already available
nltk.download('movie_reviews')
nltk.download('punkt')

# Load IMDb movie reviews dataset from NLTK
def load_imdb_data():
    documents = []
    for category in movie_reviews.categories():
        for fileid in movie_reviews.fileids(category):
            words = movie_reviews.words(fileid)
            documents.append(" ".join(words), category)) # Convert words into a single string
    random.shuffle(documents) # Shuffle data
    return documents

# Preprocessing: Tokenization, Lowercasing, Removing Special Characters
def preprocess(text):
    text = text.lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text) # Remove special characters and numbers
    words = nltk.word_tokenize(text) # Tokenization
    return words

# Generate n-grams (unigrams or bigrams)
def generate_ngrams(words, n=1):
    if n == 1:
        return words # Return individual words for unigrams
    elif n == 2:
        return list(nltk.bigrams(words)) # Generate bigrams (word pairs)
    return []

# Create Bag-of-Words model
def create_bow(data, n=1):
    bow = []
    for text, label in data:
        words = preprocess(text)
        ngrams = generate_ngrams(words, n)
        bow.append((ngrams, label))
    return bow

# Train Naïve Bayes Model
```

```

def train_naive_bayes(bow, laplace_smoothing=True):
    word_counts = defaultdict(Counter)
    class_counts = Counter()
    vocabulary = set()

    # Count words per class
    for words, label in bow:
        class_counts[label] += 1
        for word in words:
            word_counts[label][word] += 1
            vocabulary.add(word)

    vocab_size = len(vocabulary)
    total_docs = sum(class_counts.values())
    epsilon = 1e-6 # Small constant to prevent log(0)

    # Calculate log probabilities
    log_prior = {label: math.log(class_counts[label] / total_docs + epsilon) for label in class_counts}
    log_likelihood = defaultdict(lambda: defaultdict(float))

    for label in word_counts:
        total_words = sum(word_counts[label].values())

        for word in vocabulary:
            count = word_counts[label][word]
            if laplace_smoothing:
                prob = (count + 1) / (total_words + vocab_size) # Laplace smoothing
            else:
                prob = count / total_words if total_words > 0 else epsilon # Avoid division by zero

            log_likelihood[label][word] = math.log(prob + epsilon) # Avoid log(0)

    return log_prior, log_likelihood, vocabulary

# Predict sentiment of a given review
def predict(review, log_prior, log_likelihood, vocabulary, n=1):
    words = preprocess(review)
    ngrams = generate_ngrams(words, n)

    scores = {label: log_prior[label] for label in log_prior}

    for word in ngrams:
        if word in vocabulary:
            for label in scores:
                scores[label] += log_likelihood[label].get(word, 0) # Add log probability if word exists in training

    return max(scores, key=scores.get) # Return label with highest score

# Evaluate model accuracy
def evaluate(test_data, log_prior, log_likelihood, vocabulary, n=1):
    correct = 0
    total = len(test_data)

    for review, label in test_data:
        prediction = predict(review, log_prior, log_likelihood, vocabulary, n)
        if prediction == label:
            correct += 1

    return correct / total * 100

# Main Execution
if __name__ == "__main__":
    # Load IMDb dataset
    data = load_imdb_data()

    # Split into training and testing sets (80% train, 20% test)
    train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

    print("Training on unigrams (1-gram) without Laplace smoothing...")
    bow_train = create_bow(train_data, n=1) # Unigram model
    log_prior_no_smooth, log_likelihood_no_smooth, vocab_no_smooth = train_naive_bayes(bow_train, laplace_smoothing=False)
    accuracy_no_smooth = evaluate(test_data, log_prior_no_smooth, log_likelihood_no_smooth, vocab_no_smooth, n=1)
    print(f"Accuracy without Laplace Smoothing: {accuracy_no_smooth:.2f}%")

    print("\nTraining on unigrams (1-gram) with Laplace smoothing...")
    log_prior_smooth, log_likelihood_smooth, vocab_smooth = train_naive_bayes(bow_train, laplace_smoothing=True)
    accuracy_smooth = evaluate(test_data, log_prior_smooth, log_likelihood_smooth, vocab_smooth, n=1)
    print(f"Accuracy with Laplace Smoothing: {accuracy_smooth:.2f}%")

    print("\nTraining on bigrams (2-gram) with Laplace smoothing...")
    bow_train_bigram = create_bow(train_data, n=2) # Bigram model
    log_prior_bigram, log_likelihood_bigram, vocab_bigram = train_naive_bayes(bow_train_bigram, laplace_smoothing=True)
    accuracy_bigram = evaluate(test_data, log_prior_bigram, log_likelihood_bigram, vocab_bigram, n=2)

```

```
print(f"Accuracy on bigrams with Laplace Smoothing: {accuracy_bigram:.2f}%")
```

```
↗ [nltk_data] Downloading package movie_reviews to /root/nltk_data...  
[nltk_data] Unzipping corpora/movie_reviews.zip.  
[nltk_data] Downloading package punkt to /root/nltk_data...  
[nltk_data] Unzipping tokenizers/punkt.zip.  
Training on unigrams (1-gram) without Laplace smoothing...  
Accuracy without Laplace Smoothing: 79.50%  
  
Training on unigrams (1-gram) with Laplace smoothing...  
Accuracy with Laplace Smoothing: 79.75%  
  
Training on bigrams (2-gram) with Laplace smoothing...  
Accuracy on bigrams with Laplace Smoothing: 84.75%
```