22p-9318_Ahmed-Ali_BS(SE)-6B_Assignment-2_AI

April 21, 2025

[2]:

```
import math
import csv
```

[3]:

```
stop_words = [
    'a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and',
    'any', 'are', "aren't", 'as', 'at', 'be', 'because', 'been', 'before',␣
 ↪'being',
    'below', 'between', 'both', 'but', 'by', "can't", 'cannot', 'could',␣
 ↪"couldn't",
    'did', "didn't", 'do', 'does', "doesn't", 'doing', "don't", 'down', 'during',
    'each', 'few', 'for', 'from', 'further', 'had', "hadn't", 'has', "hasn't",
    'have', "haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here',
    "here's", 'hers', 'herself', 'him', 'himself', 'his', 'how', "how's", 'i',
    "i'd", "i'll", "i'm", "i've", 'if', 'in', 'into', 'is', "isn't", 'it',␣
 ↪"it's",
    'its', 'itself', "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself',
    'no', 'nor', 'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other',␣
 ↪'ought',
    'our', 'ours', 'ourselves', 'out', 'over', 'own', 'same', "shan't", 'she',
    "she'd", "she'll", "she's", 'should', "shouldn't", 'so', 'some', 'such',␣
 ↪'than',
    'that', "that's", 'the', 'their', 'theirs', 'them', 'themselves', 'then',
    'there', "there's", 'these', 'they', "they'd", "they'll", "they're",␣
 ↪"they've",
    'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 'very',␣
 ↪'was',
    "wasn't", 'we', "we'd", "we'll", "we're", "we've", 'were', "weren't", 'what',
    "what's", 'when', "when's", 'where', "where's", 'which', 'while', 'who',
    "who's", 'whom', 'why', "why's", 'with', "won't", 'would', "wouldn't", 'you',
    "you'd", "you'll", "you're", "you've", 'your', 'yours', 'yourself',
    'yourselves'
]
```

[4]:

```
def clean_and_tokenize(text):
    """
    Clean and tokenize text without using external libraries
    """
    text = text.lower()

    #replacing non alphabatical chars without use of regex
    cleaned_text = ""
    for char in text:
        if char.isalpha() or char.isspace():
            cleaned_text += char
        else:
            cleaned_text += " "

    tokens = cleaned_text.split()

    tokens = [token for token in tokens if token not in stop_words]

    return tokens
```

[5]:

```
def create_word_freq_dict(tokens):
    """
    Create a dictionary of word frequencies
    """
    word_freq = {}
    for token in tokens:
        if token in word_freq:
            word_freq[token] += 1
        else:
            word_freq[token] = 1
    return word_freq
```

[6]:

```
def load_data(filename):
    """
    Load data from TSV file without using pandas
    """
    data = []
    with open(filename, 'r', encoding='utf-8') as file:
        reader = csv.reader(file, delimiter='\t')
        header = next(reader)
        for row in reader:
            if len(row) >= 2:
                genre = row[0]
                description = row[1]
                data.append((genre, description))
    return data
```

[7]:

```
def train_naive_bayes(training_data):
    """
    Train a Naive Bayes classifier from scratch
    """
    #count of occurrences of each word per genre
    word_counts_per_genre = {}
    genre_counts = {}
    vocabulary = set()

    for genre, description in training_data:
        #updating genre counts
        if genre in genre_counts:
            genre_counts[genre] += 1
        else:
            genre_counts[genre] = 1
            word_counts_per_genre[genre] = {}

        #processing the description
        tokens = clean_and_tokenize(description)
        for token in tokens:
            vocabulary.add(token)
            if token in word_counts_per_genre[genre]:
                word_counts_per_genre[genre][token] += 1
            else:
                word_counts_per_genre[genre][token] = 1

    #calc word probabilities using the technique Laplace smoothing, A new thing␣
↪for me
    vocab_size = len(vocabulary)
    word_probs_per_genre = {}
    total_documents = len(training_data)

    for genre in genre_counts:
        word_probs_per_genre[genre] = {}
        total_words_in_genre = sum(word_counts_per_genre[genre].values())

        word_probs_per_genre[genre]['__prior__'] = math.log(genre_counts[genre] /
↪ total_documents)

        #calc conditional probabilities for each word
        for word in vocabulary:
            count = word_counts_per_genre[genre].get(word, 0)
            #Laplace smoothing (also known as add one smoothing)
            prob = (count + 1) / (total_words_in_genre + vocab_size)
            word_probs_per_genre[genre][word] = math.log(prob)

    return word_probs_per_genre, vocabulary
```

[8]:

```
def predict_genre(description, word_probs_per_genre, vocabulary):
    """
    Predict genre for a given description
    """
    tokens = clean_and_tokenize(description)
    scores = {}

    #calc score for each genre
    for genre in word_probs_per_genre:
        #starting with prior probability
        scores[genre] = word_probs_per_genre[genre]['__prior__']

        #add log probabilities for each word
        for token in tokens:
            if token in vocabulary:  #only consider words in given vocabulary
                scores[genre] += word_probs_per_genre[genre].get(token, 0)

    #return genre with highest score
    return max(scores, key=scores.get)
```

[9]:

```
def evaluate_classifier(test_data, word_probs_per_genre, vocabulary):
    """
    Evaluate classifier performance on test data
    """
    results = {}

    for genre, description in test_data:
        #initialize checks if not seen genre before
        if genre not in results:
            results[genre] = {'correct': 0, 'incorrect': 0, 'total': 0}

        #increment total count for genre
        results[genre]['total'] += 1

        #making prediction
        predicted_genre = predict_genre(description, word_probs_per_genre,␣
 ↪vocabulary)

        #update correct/incorrect counts
        if predicted_genre == genre:
            results[genre]['correct'] += 1
        else:
            results[genre]['incorrect'] += 1

    return results
```

[10]:

```
def main():
    training_data = load_data('film-genres-train.csv')
    test_data = load_data('film-genres-test.csv')
    print("Training classifier...")
    word_probs_per_genre, vocabulary = train_naive_bayes(training_data)

    print("Evaluating classifier...")
    results = evaluate_classifier(test_data, word_probs_per_genre, vocabulary)

    print("\nClassification Results:")
    print("-" * 60)
    print(f"{'Genre':<12} {'Correct':<10} {'Incorrect':<10} {'Total':<10}␣
↪{'Accuracy (%)':<10}")
    print("-" * 60)

    total_correct = 0
    total_total = 0

    for genre, counts in results.items():
        accuracy = (counts['correct'] / counts['total']) * 100 if␣
↪counts['total'] > 0 else 0
        print(f"{genre:<12} {counts['correct']:<10} {counts['incorrect']:<10}␣
↪{counts['total']:<10} {accuracy:.2f}")
        total_correct += counts['correct']
        total_total += counts['total']

    print("-" * 60)
    overall_accuracy = (total_correct / total_total) * 100 if total_total > 0␣
↪else 0
    print(f"{'Overall':<12} {total_correct:<10} {total_total - total_correct:
↪<10} {total_total:<10} {overall_accuracy:.2f}")
```

[11]:

```
if __name__ == "__main__":
    main()
```

```
Training classifier...
Evaluating classifier...

Classification Results:
------------------------------------------------------------
Genre        Correct    Incorrect  Total      Accuracy (%)
------------------------------------------------------------
Documentary  527        125        652        80.83
Comedy       510        614        1124       45.37
Drama        1990       217        2207       90.17
Horror       62         190        252        24.60
Western      197        42         239        82.43
------------------------------------------------------------
Overall      3286       1188       4474       73.45
```

[ ]:

[ ]:

[ ]: