Drew Lickman
CSCI 4820-001
Project #3
Due: 10/9/24

A.I. Disclaimer: Work for this assignment was completed with the aid of artificial intelligence tools and comprehensive documentation of the names of, input provided to, and output obtained from, these tools is included as part of my assignment submission in ai_usage.pdf.

Lexicon-Based Sentiment Analysis using Custom Logistic Regression

Assignment Requirements:

Input

- Positive words
- Negative words
- IMDb reviews

Processing

- There are two classifiers
  - Custom Logistic Regression
  - sklearn LogisticRegression
- Implement a Python class (CustomLogisticRegression)
  - __init__(self, learning_rate, num_iters)
    - self.learning_rate
    - self.num_iters
    - self.weights
    - self.bias
  - sigmoid(z)
    - return sigmoid function
  - fit(X, y)
    - Sets weights to correct shape and initializes them to 0
    - Applies batch gradient descent to the entire dataset in a loop for num_iters
    - Updated weights and biases
  - predict(X)
    - z = w dot x + b
    - return sigmoid(z)

Output

- For each trial and for each classifier
  - Print the sklearn confusion_matrix and classification_Report
- Output the average of the confusion matrices across trials for each classifier

Python Code

1. Load and process positive and negative sentiment lexicon words

In [1]:
```
sentimentWords = []
posWords = []
negWords = []
# Save positive words to array
with open("positive-words.txt", encoding="utf-8") as positivewords:
        lines = positivewords.readlines()
        for line in lines:
                if line[0] != ";" and line.strip() != '':
                        posWords.append(line.rstrip('\n'))
# Save negative words to array
```

```
with open("negative-words.txt", encoding="utf-8") as negativewords:
        lines = negativewords.readlines()
        for line in lines:
                if line[0] != ";" and line.strip() != '':
                        negWords.append(line.rstrip('\n'))
sentimentWords = posWords + negWords
```

2. Load and process IMDb reviews

In [2]:
```
# Add each line of the IMDb reviews to the reviews array
reviews = []
trueValues = []
with open("imdb_reviews.txt", encoding="utf-8") as imdbreviews:
        lines = imdbreviews.readlines()
        for line in lines:
                splitLine = line.rstrip().rsplit(' ', 1)
                reviews.append(splitLine[0]) # removes true sentiment label from data

                sentiment = splitLine[1].strip()[-8:] # the last 8 characters are either positive or negative
                if sentiment == "positive":
                        trueValues.append(1)
                elif sentiment == "negative":
                        trueValues.append(0)
                else:
                        print("Error: sentiment analysis not found at end of line!")
```

3. Create Features(X) table and Labels(y) array

In [3]:
```
import numpy as np

X = np.zeros((len(reviews), len(sentimentWords)), dtype=bool)   # Features
y = np.array(trueValues, dtype=int)                                                # Labels

# Count how many positive / negative words show up in each review
posCount = 0
negCount = 0

# If a sentiment word is in the review, mark it as True in the X feature table
# for review in range(len(reviews)):                                   # 25,000
#       for word in range(len(sentimentWords)):                        # * 6,786
#               if sentimentWords[word] in reviews[review]:     # = 169,650,000 loops
#                       X[review, word] = True                                         # Takes 2-5 minutes

# cursor-small improved my time complexity from O(n*m*k) to O(n*k)
# Convert sentimentWords list to a set for faster membership testing
sentimentWords_set = set(sentimentWords) # Convert array to unsorted set
for review_index, review in enumerate(reviews):
    words = review.split() # Check each word in every review line
    for word in words:
        if word in sentimentWords_set:  # This check is now O(1) by using set data structure
            X[review_index, sentimentWords.index(word)] = True  # Reduces time to <15 seconds
```

(Debug viewing)

In [4]:
```
if False:
        print(X.shape)
        print(y.shape)
        # Only show the first 10 reviews to make sure things are loading properly
        for review in range(10):
                print(reviews[review])
                for sentimentWord in range(len(sentimentWords)):
                        if X[review, sentimentWord] == True:
                                # Prints all sentiment words that occur in each review line
                                print(f"{sentimentWords[sentimentWord]}", end=" ")
                # Display if a review is positive or negative
                print()
                if y[review] == 1:
                        print(f"Review {review} is positive!")
                elif y[review] == 0:
                        print(f"Review {review} is negative!")
                print("---")
```

4. Define Custom Logistic Regression class

In [5]:
```
class CustomLogisticRegression():
        # Constructor
        def __init__(self, learning_rate, num_iters):
                self.learning_rate = learning_rate
                self.num_iters = num_iters
```

```
            self.weights = None
            self.bias = None

    # Function for X dot W + b
    def linearTransform(self, X):
            z = np.dot(X, self.weights) + self.bias
            return z

    # Inputs either scalar or array and outputs sigmoid function of the scalar or array
    def sigmoid(self, z):
            # Formula from LogisticRegression slide 28
            output = 1 / (1 + np.exp(-z)) # np.exp does e^(-z) for all samples in the reviews array
            return output

    # Calculate probability of a sample being a class (positive or negative)
    def predict(self, X):
            prob = self.sigmoid(self.linearTransform(X))
            #prob = int(prob >= 0.5) # Convert to binary output
            return prob

    # Train the model using gradient descent
    # X is training features, y is labels
    def fit(self, X, y):
            # Sets the weights to the correct shape and initializes them to 0
            features = X.shape[1] # Literally just how many sentiment words there are
            self.weights = np.zeros(features) # weight for each feature
            self.bias = 0

            # Apply batch gradient descent on entire dataset
            # This for loop was written by Claude 3.5 Sonnet and modified by myself
            for _ in range(self.num_iters):         # Apply gradient descent num_iters times
                    predictions = self.predict(X)   # Calculate array of sigmoidal probabilities
                    error = predictions - y                 # Calculate the difference between predicted and

                    # Compute gradient for weights
                    # Calculate how much each feature contributes to the error across all samples
                    weightGradient = (1 / len(y)) * np.dot(X.T, error)      # X.T is transposed so the dot p
                    # Compute gradient for bias
                    # Calculate how much bias needs to be adjusted based on overall error
                    biasGradient = (1 / len(y)) * np.sum(error)             # Average of all errors

                    # Update weights and biases
                    self.weights -= self.learning_rate * weightGradient
                    self.bias -= self.learning_rate * biasGradient
```

5. Run 5 trials of the SKLearn Linear Model

scikit-learn documentation

```
- https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.train_test_split.html

- https://scikit-
learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.Lo


- https://scikit-
learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.Lo


- https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.confusion_matrix.html

- https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.classification_report.html
```

In [6]:

```
from sklearn import linear_model as lm
from sklearn import model_selection as ms
from sklearn import metrics

# Initialize variables to store average confusion matrices
avgConfusionMatrix_skllr = np.zeros((2, 2))
avgConfusionMatrix_mylr = np.zeros((2, 2))

trialCount = 5
iterationCount = 500
# Takes about 20 seconds per combined 1*10 iterations;
# Combined: 1*500 took 8 minutes, 5*500 took 40 minutes
# SKLearn LR: takes about 15 seconds for 500 iterations
# My Custom LR: takes almost 8 minutes for 500 iterations
for trial in range(trialCount):
        # Shuffle input data
        # Split into 80% 20% split of training and test sets
```

```python
        # Line from Claude 3.5 Sonnet
        X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size=0.2, random_state=trial)

        # SKLearn library Logistic Regression class and methods
        skllr = lm.LogisticRegression(solver='sag', C=0.001, max_iter=iterationCount)
        skllr.fit(X_train, y_train)                              # Only use the 80% of the data marked for train
        skllrPredictions = skllr.predict(X_test)        # Use the remaining 20% of the data marked for testing

        # Initialize CustomLogisticRegression class,
        # Then train it over iterationCount times, which adjusts the weights and bias
        # Then predict each sample, using the updated weights and bias
        mylr = CustomLogisticRegression(learning_rate=0.1, num_iters=iterationCount)
        mylr.fit(X_train, y_train)                                                               # Only
        mylrPredictions = (mylr.predict(X_test) >= 0.5).astype(int) # Use the remaining 20% of the data marked

        # Calculate confusion matrices
        skllr_confMat = metrics.confusion_matrix(y_test, skllrPredictions)
        mylr_confMat = metrics.confusion_matrix(y_test, mylrPredictions)

        # Generate classification reports
        skllr_report = metrics.classification_report(y_test, skllrPredictions, target_names=["Positive", "Negat
        mylr_report = metrics.classification_report(y_test, mylrPredictions, target_names=["Positive", "Negative

        # Evaluate sklearn model
        print(f"Trial {trial + 1} - Sklearn LogisticRegression:")
        print(skllr_confMat)
        print(skllr_report)

        # Evaluate custom model
        print(f"Trial {trial + 1} - Custom LogisticRegression:")
        print(mylr_confMat)
        print(mylr_report)

        # Update average confusion matrices
        avgConfusionMatrix_skllr += skllr_confMat
        avgConfusionMatrix_mylr += mylr_confMat

# Calculate and print average confusion matrices
avgConfusionMatrix_skllr /= trialCount
avgConfusionMatrix_mylr /= trialCount

# After all trials are completed, print average of the trials
print("Average Confusion Matrix - Sklearn LogisticRegression:")
print(avgConfusionMatrix_skllr)
print("Average Confusion Matrix - Custom LogisticRegression:")
print(avgConfusionMatrix_mylr)
```

```
Trial 1 - Sklearn LogisticRegression:
[[1905  613]
 [ 412 2070]]
              precision    recall  f1-score   support

    Positive       0.82      0.76      0.79      2518
    Negative       0.77      0.83      0.80      2482

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.79      5000
weighted avg       0.80      0.80      0.79      5000

Trial 1 - Custom LogisticRegression:
[[1949  569]
 [ 400 2082]]
              precision    recall  f1-score   support

    Positive       0.83      0.77      0.80      2518
    Negative       0.79      0.84      0.81      2482

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.81      5000
weighted avg       0.81      0.81      0.81      5000

Trial 2 - Sklearn LogisticRegression:
[[1914  628]
 [ 379 2079]]
              precision    recall  f1-score   support

    Positive       0.83      0.75      0.79      2542
    Negative       0.77      0.85      0.81      2458

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.80      5000
weighted avg       0.80      0.80      0.80      5000

Trial 2 - Custom LogisticRegression:
```

```
[[1965  577]
 [ 373 2085]]
              precision    recall  f1-score   support

    Positive       0.84      0.77      0.81      2542
    Negative       0.78      0.85      0.81      2458

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.81      5000
weighted avg       0.81      0.81      0.81      5000

Trial 3 - Sklearn LogisticRegression:
[[1928  588]
 [ 397 2087]]
              precision    recall  f1-score   support

    Positive       0.83      0.77      0.80      2516
    Negative       0.78      0.84      0.81      2484

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.80      5000
weighted avg       0.80      0.80      0.80      5000

Trial 3 - Custom LogisticRegression:
[[1977  539]
 [ 401 2083]]
              precision    recall  f1-score   support

    Positive       0.83      0.79      0.81      2516
    Negative       0.79      0.84      0.82      2484

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.81      5000
weighted avg       0.81      0.81      0.81      5000

Trial 4 - Sklearn LogisticRegression:
[[1916  541]
 [ 442 2101]]
              precision    recall  f1-score   support

    Positive       0.81      0.78      0.80      2457
    Negative       0.80      0.83      0.81      2543

    accuracy                           0.80      5000
   macro avg       0.80      0.80      0.80      5000
weighted avg       0.80      0.80      0.80      5000

Trial 4 - Custom LogisticRegression:
[[1954  503]
 [ 432 2111]]
              precision    recall  f1-score   support

    Positive       0.82      0.80      0.81      2457
    Negative       0.81      0.83      0.82      2543

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.81      5000
weighted avg       0.81      0.81      0.81      5000

Trial 5 - Sklearn LogisticRegression:
[[1935  577]
 [ 401 2087]]
              precision    recall  f1-score   support

    Positive       0.83      0.77      0.80      2512
    Negative       0.78      0.84      0.81      2488

    accuracy                           0.80      5000
   macro avg       0.81      0.80      0.80      5000
weighted avg       0.81      0.80      0.80      5000

Trial 5 - Custom LogisticRegression:
[[1980  532]
 [ 401 2087]]
              precision    recall  f1-score   support

    Positive       0.83      0.79      0.81      2512
    Negative       0.80      0.84      0.82      2488

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.81      5000
weighted avg       0.81      0.81      0.81      5000
```

```
Average Confusion Matrix - Sklearn LogisticRegression:
[[1919.6  589.4]
 [ 406.2 2084.8]]
Average Confusion Matrix - Custom LogisticRegression:
[[1965.   544. ]
 [ 401.4 2089.6]]
```