IST 664 FINAL PROJECT

Summary

Sentiment analysis is a critical area of natural language processing that involves understanding the emotional tone behind a body of text. This is particularly useful in analyzing movie reviews, where understanding sentiment can help gauge the overall reception of a film.

This report details the execution of sentiment analysis on a dataset of movie reviews through three distinct methodologies. The methods include two classical model approaches, one utilizing vectorization and the other manual feature engineering and a deep learning approach. Each method is distinguished by its specific preprocessing routines, feature extraction techniques, and modeling strategies.

2. Methodology

2.1 Data Preprocessing

Data preprocessing is crucial to prepare raw text for further analysis and model training. The preprocessing steps implemented across all models include:

- **Tokenization:** Splitting text into individual words or tokens. This is fundamental as it transforms raw text into an analyzable format.
- **Removal of Noise:** Stripping away unnecessary characters such as punctuation and numbers, which might lead to overfitting on irrelevant features.
- **Case Normalization:** Converting all tokens to lower case to ensure that the same words in different cases are treated as identical.
- **Stopword Removal:** Eliminating common words (e.g., “and” "the", etc.) that might dilute the predictive power of important words. Notably, **negation words like "not" are retained as they are crucial for sentiment analysis.**
- **Lemmatization:** Reducing words to their base or root form, which helps in generalizing different forms of the same word to a single form, enhancing model robustness.
- **Addressing Class Imbalance** To address class imbalance, a manual balancing technique is applied, ensuring equal representation by oversampling minority classes and under sampling majority ones. This approach reduces bias by randomizing and equalizing the training data across all sentiment classes.

2.2 Feature Engineering and Vectorization Techniques:

Type 1: Classical Model with Vectorization Techniques

Vectorization:

- **Bag-of-Words (BoW):** This model transforms text into a fixed-length set of features, with each feature representing the count of a specific word in the text. We extend this by using both unigrams and bigrams to capture more context.
- **Implementation:** Utilized **CountVectorizer** from scikit-learn, configuring it to generate both unigram and bigram tokens.

Model Implementation with Logistic Regression:

Logistic Regression is applied in two variations.

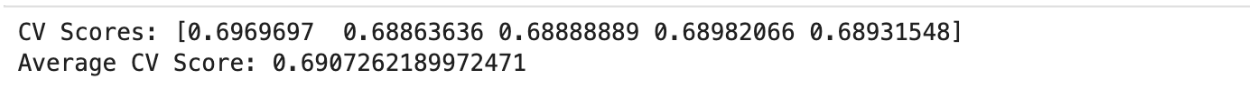
- The first uses the model's default settings, providing a baseline for performance.
- The second variation involves tuning parameters such as the regularization strength and choice of solver (algorithm for optimization) using GridSearchCV, aiming to optimize the model by finding the best settings that minimize prediction error and handle overfitting.

Evaluation via Cross-Validation:

- The logistic regression models are evaluated using cross-validation.
- The model is trained on multiple combinations of these subsets and tested on the remaining parts to ensure the model's effectiveness and generalizability across different data samples within the dataset.

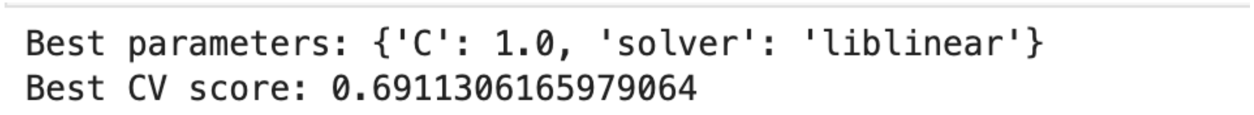
Result:

Without Hyper Parameter Tuning:



```
CV Scores: [0.6969697  0.68863636 0.68888889 0.68982066 0.68931548]
Average CV Score: 0.6907262189972471
```

With Hyper Parameter Tuning:



```
Best parameters: {'C': 1.0, 'solver': 'liblinear'}
Best CV score: 0.6911306165979064
```

Type 2: Classical Model by Creating Feature Sets

Custom Feature Sets:

- **Subjectivity Lexicon:** Incorporating a lexicon that includes words labeled with their polarity (positive/negative) and strength (strong/weak). This allows the models to have a basic understanding of sentiment from the lexical features.

  We also explored manually created features, such as:
- **Presence of Words:** Creating a binary feature for each word to denote its presence or absence in a document.
- **Part-of-Speech Tags:** Utilizing NLTK's POS tagger, we categorized words into their respective parts of speech, which helps in distinguishing between different uses of the same word in different contexts.

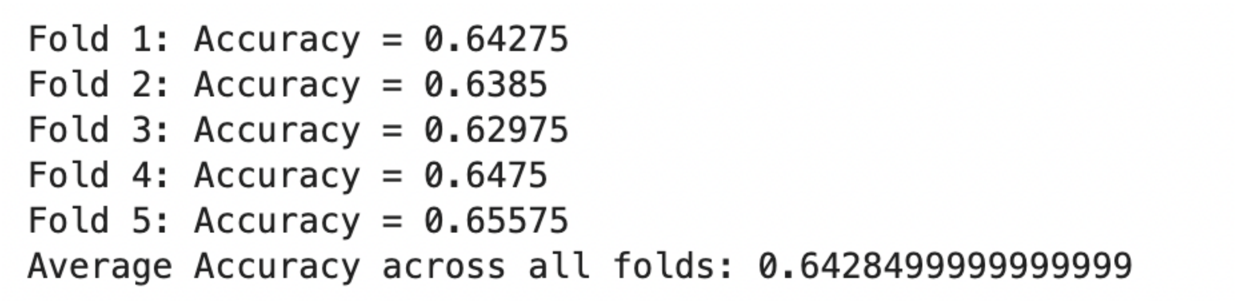
Model Implementation with Random Forest Classifier:

- The Random Forest Classifier is employed to handle the potentially sparse and high-dimensional data created by the manual feature sets.
- This model type is particularly good at managing overfitting through its ensemble approach, where multiple decision trees contribute to the final decision, making the model robust against noise.

Evaluation with Custom Cross-Validation:

- Customized cross-validation methods are used to assess the model's performance, ensuring that the evaluation considers the balance across different sentiment classes.
- This helps in verifying that the model performs well across diverse data scenarios and is not biased toward a particular sentiment.

Result:



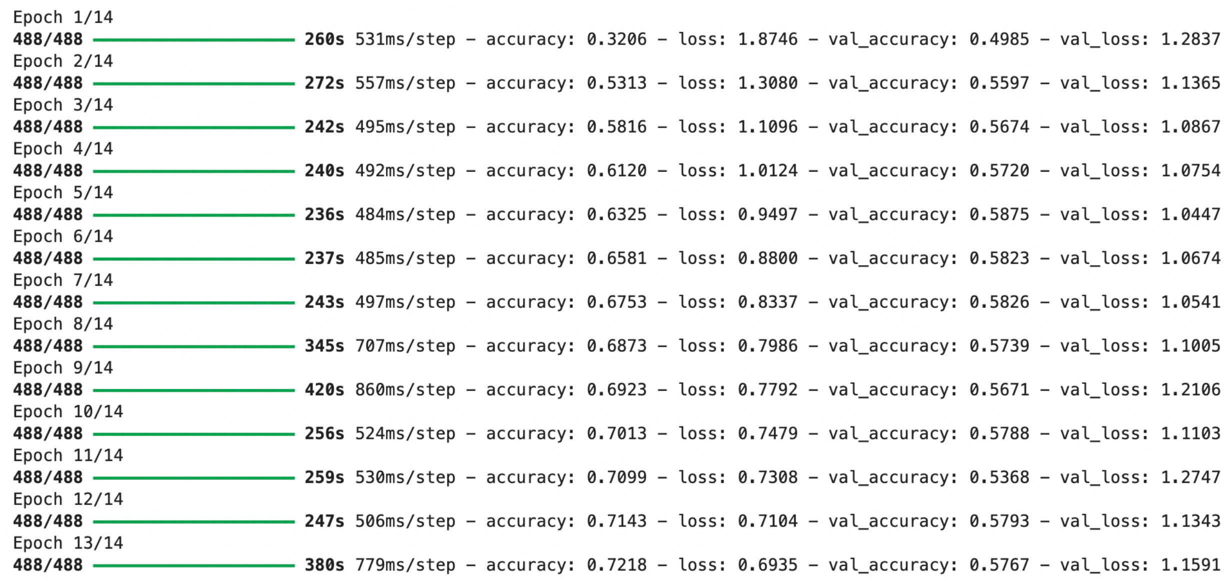
```
Fold 1: Accuracy = 0.64275
Fold 2: Accuracy = 0.6385
Fold 3: Accuracy = 0.62975
Fold 4: Accuracy = 0.6475
Fold 5: Accuracy = 0.65575
Average Accuracy across all folds: 0.6428499999999999
```

Type 3: Deep Learning Model:

Neural Network Configuration with Embeddings and LSTM:

- **Embedding Layer**: Converts words into dense vectors of fixed size where semantically similar words are mapped to similar points in the vector space. This captures more information per word than one-hot encoding.
- **Bidirectional LSTM Layers**: These layers allow the network to have both forward and backward information about the sequence at every point. This is beneficial for understanding the context and dependencies in language data, as the meaning of a word can depend on the words that come before and after it.
- **Regularization Techniques**:
  - **Dropout**: Randomly drops units (along with their connections) during the training process to prevent overfitting.
  - **Batch Normalization**: Normalizes the input layer by adjusting and scaling activations, which helps to stabilize and speed up the training process.
- **Training and Optimization**: The model is trained and fine-tuned using the Adam optimizer, a method that adjusts the learning rate dynamically. Adjustments in learning rate and the number of training epochs are made based on performance metrics observed during validation, ensuring the model is neither underfitting nor overfitting.

RESULTS:



```
Epoch 1/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 260s 531ms/step - accuracy: 0.3206 - loss: 1.8746 - val_accuracy: 0.4985 - val_loss: 1.2837
Epoch 2/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 272s 557ms/step - accuracy: 0.5313 - loss: 1.3080 - val_accuracy: 0.5597 - val_loss: 1.1365
Epoch 3/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 242s 495ms/step - accuracy: 0.5816 - loss: 1.1096 - val_accuracy: 0.5674 - val_loss: 1.0867
Epoch 4/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 240s 492ms/step - accuracy: 0.6120 - loss: 1.0124 - val_accuracy: 0.5720 - val_loss: 1.0754
Epoch 5/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 236s 484ms/step - accuracy: 0.6325 - loss: 0.9497 - val_accuracy: 0.5875 - val_loss: 1.0447
Epoch 6/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 237s 485ms/step - accuracy: 0.6581 - loss: 0.8800 - val_accuracy: 0.5823 - val_loss: 1.0674
Epoch 7/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 243s 497ms/step - accuracy: 0.6753 - loss: 0.8337 - val_accuracy: 0.5826 - val_loss: 1.0541
Epoch 8/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 345s 707ms/step - accuracy: 0.6873 - loss: 0.7986 - val_accuracy: 0.5739 - val_loss: 1.1005
Epoch 9/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 420s 860ms/step - accuracy: 0.6923 - loss: 0.7792 - val_accuracy: 0.5671 - val_loss: 1.2106
Epoch 10/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 256s 524ms/step - accuracy: 0.7013 - loss: 0.7479 - val_accuracy: 0.5788 - val_loss: 1.1103
Epoch 11/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 259s 530ms/step - accuracy: 0.7099 - loss: 0.7308 - val_accuracy: 0.5368 - val_loss: 1.2747
Epoch 12/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 247s 506ms/step - accuracy: 0.7143 - loss: 0.7104 - val_accuracy: 0.5793 - val_loss: 1.1343
Epoch 13/14
488/488 ━━━━━━━━━━━━━━━━━━━━ 380s 779ms/step - accuracy: 0.7218 - loss: 0.6935 - val_accuracy: 0.5767 - val_loss: 1.1591
```

Team Members and their tasks:

- Subhiksha Murugesan - **Classical Model with Vectorization Techniques**
- Nithish Kumar Senthil Kumar - **Classical Model by Creating Feature Sets**
- Nagul Pandian - **Deep Learning Model**