SENG 550: FINAL PROJECT REPORT

Preamble:

Kya - Created the primary code for data analysis and model implementation, including building the machine learning models and testing different preprocessing methods.
Ian - Added the creation of custom cleaning functions, including defining and implementing domain-specific stopword removal and word balancing logic.
Quin - Worked alongside group members to develop the codebase, implemented the experiments, implemented code cleaning, and handled model evaluation.
Gillian - Focused on preprocessing and writing significant portions of reporting. Analyzed final cleaning and aided in finishing touches.

Abstract:

The project focuses on building a sentiment analysis pipeline for the IMDB dataset of 50,000 Labeled movie reviews. Our goal was to use binary classification of text, to perform sentiment analysis and attempt to classify reviews and predict them as being either positive or negative using our model. A logistic regression, naive-bayes and decision tree model were all used and compared against with the logistic regression model being our primary classifier. We utilized extensive text preprocessing, including tokenization, stop-word removal, and TF-IDF vectorization to help improve the precision and accuracy of all of our models. Experiments evaluate the impact of different data splits, preprocessing techniques, and feature configurations on model performance. Our findings demonstrate that logistic regression achieves an F1-score of close to one.

Introduction:

Sentiment analysis of the movie reviews has significant influence on customer feedback analysis, marketing, and recommendation systems. In this project, we aim to classify IMDB movie reviews as positive or negative, enabling us to infer user sentiment from text data. This involves solving a binary text classification problem using machine learning techniques.

The dataset used is the IMDB dataset, comprising 50,000 reviews evenly split between positive and negative sentiment labels. Each review is a free form text string, while the sentiment label serves as the target variable. The dataset is balanced and includes an equal number of training and test samples, making it suitable for evaluating classification models. You can access the dataset directly using this link: IMDB Kaggle Dataset

Sentiment analysis is a critical task in natural language processing. This technique is employed when businesses need to understand customer opinions at scale. Accurate sentiment prediction can influence decision making in areas such as product development and marketing strategies.

Previous studies have applied methods such as Naive Bayes, SVMs, and deep learning models seen with RNNs for sentiment analysis. Deep learning achieves state-of-the-art accuracy, simpler models like logistic regression offer competitive performance with lower computational costs, especially for structured datasets like the IMDB reviews being used.

Deep learning models are effective, but often require significant computational resources and are prone to overfitting on small datasets. This project seeks to explore the potential of logistic regression combined with robust feature engineering to achieve comparable results while being more interpretable and efficient.

We also have an overarching goal, in our project, to differ from other models in our extensive data analysis and cleaning. Other methods we discovered have lacked proper pre-processing and analysis. Many failed to go beyond using a simple built- in stop word function.. We aimed to improve our model, by taking a bigger look at some of the aspects of our data that should be cut or modified to improve our model before even writing it!

Before starting our project, we asked ourselves the following data analysis questions:

- What factors should we explore in our preliminary analysis?
- How can our preliminary analysis influence our data cleaning decisions?
- How do certain preprocessing measures affect the performance of the model?
- Which model performs the best?
- What were the key factors that affected model accuracy?

We seeked to answer these throughout our analysis, cleaning, and model implementation/evaluation, and the answers to these questions will be explored below in this paper.

Our Proposal:

As mentioned above, we are proposing the creation of a ML algorithm, utilizing Linear Regression, to complete binary text data classification of a dataset. We seek to compare it with other common algorithms, to see if Linear Regression can compete with some stronger Binary classification algorithms given proper data preprocessing and cleaning – given the writer of the ML algorithm understands their data.

Main findings:

We found that linear regression models did, in fact, compete with other classification methods. It performed better than expected, and displayed that understanding your data is an essential step to creating a model for your data.

Methodology:

1. **General Data Cleaning**

Before diving into specific cleaning strategies, the dataset underwent a general cleaning process to standardize the text. This included:

- Removing punctuation, such as commas and periods.
- Standardizing whitespace by trimming unnecessary spaces.
- Converting all text to lowercase to avoid case sensitivity issues.

This general preprocessing step ensured uniformity across the dataset and improved the model's ability to process the text accurately, particularly in tokenization and frequency analysis.

Following general cleaning, deeper exploratory analysis revealed significant insights into the dataset. For instance, after the initial round of cleaning, the dataset still contained hundreds of thousands of words. As a result, we would need to do even deeper cleaning, to ensure our dataset is relevant and practical for our ML models.

2. Explorations of data features, further cleaning of our data, and the results this provided

With a large dataset such as that we worked with in this project, understanding and being able to pull out key features from your data is essential. There were four main factors we noted, just upon reading the data that we knew we wanted to explore. These included *contrasting the number of positive reviews to negative reviews, exploring our data types, the descriptive statistics of the text-based review data, exploring "quirks" in our text-based data,* and *identifying the most primary words used in the dataset*.

Below, I will give detailed descriptions about my reasonings, findings, and comments, on each feature we explored.

Contrasting number of positive and negative reviews in the dataset

For any ML problem, a major factor in sub-optimal performance on real-world data, could be data or class imbalances. This happens when one class of a binary model is represented significantly more than another. This is referred to as a data skew, and is a well known issue in the world of Machine Learning. This issue, of having one side overrepresented, can lead to a couple of major issues in a model, including Biases towards the majority class, and evaluation problems.

We performed an analysis of our data, to see if data skewing would have to be a consideration in our model, but were lucky to find that the data seemed to be evenly spread across both positive and negative reviews.

Exploring data types

This source of analysis often speaks for itself, and is one of the most basic points of analysis, but also the most important. Our data Included types as expected – with both the reviews and sentiment being Strings. However, we knew that we would have to go on to convert the string sentiment into an integer, as ML models work largely on numerical data when possible.

Descriptive statistics of text based data

This form of analysis is used to help a user understand their large inputs of text data. How long is each review? How often are words used? Do some words seem to be equally distributed amongst both types of review sentiments? What does this mean for our data?

These were questions we explored through taking descriptive statistics of our review data. We started with the basics – trying to understand the unique words and their associated counts (ie how many times a word appears in our dataset). We were very surprised to find that the median "count" of each unique word was actually 1. Our dataset consisted of a lot of words that appeared only once. At this point in our code, we hypothesized that these could be "safe" words to remove. To ensure this, however, we would have to test our model before and after cleaning these from it.

Another factor with word counts, and removing those with low counts from our dataset, was our cutoff for words we would deem to be “not influential” on our model. Was a count of 1 the only cutoff? What about 2? To explore this, and make some educated decisions, we opted to graph out our text data – getting the frequencies of associated counts. To our surprise, it seemed the majority of words only appeared 1-5 times in the dataset. This brought up a hypothesis to remove these entirely from the dataset, and see how our model performed, in response.

We attempted to do this, but lacked the computing power to fully complete this step. With a dataset as large as ours, the process of cleaning rare words took close to 3 hours. After this completed, the rest of our pipeline and training processes ran so slow, that we decided it would be best to remove this step from our data.

Another aspect we examined was the balance of word occurrences across positive and negative reviews. For example, we observed that certain words appeared in nearly equal proportions across both sentiment classes. These “balanced” words, such as "movie" or "film," offered limited discriminatory power. By visualizing the percentage difference in occurrences between the two classes, we identified many of these words. We managed to clean these in 2 ways – that of which I will discuss below, and stopwords, which is discussed in the next section.

This is when we discovered IDF - a way to get around our computing power issue for our unique words, and also apply some weights to our words that seem to be balanced. This function essentially added “weights” to words for our model, valuing words that appear somewhat frequently as high importance, and words that appear a lot or very little as “less important.” This essentially encapsulated our main idea revolving around infrequent words, and added on a layer of frequent word analysis we had not considered.

Adding this function in gave our model:

- Reduced Noise: Words that occurred rarely were often typos, niche terms, or contextually irrelevant, which could introduce noise rather than valuable insights. Removing them streamlined the feature space.
- Improved Generalization: With fewer irrelevant/redundant features, the model could focus more effectively on meaningful patterns, enhancing its ability to generalize to unseen data.

Identifying primary words in our dataset

Right away, we opted to print out the 20 most used words in our reviews. Unsurprisingly, these were bland words, with little-to-no sentiment meaning – words such as “to”, “the”, “and”, and so on. These words, as we discovered, are called *stopwords*, formally defined as any words commonly recurring in a dataset, with little to no meaning in improving model training.

After cleaning the data with a regular stopwords library, and following this discovery, and proceeding to run the same analysis on the now cleaned data, we discovered something else – there remained, still,

many stopwords. The words “movie” “film” “think,” and so on, all seemed to be the top words in a data set, of which (after identifying the % of positive and negative reviews they were a part of, and discovering the difference in these to be minimal) offered no insight to our model training.

So, we opted to create another function of custom stopwords, based off of some hand-picked words from the list of the top 50 words remaining in the dataset, and removed even more stopwords from the data. This iterative cleaning process had a measurable impact on the model. With improved precision and recall, removing irrelevant and domain specific stopwords reduced noise in the dataset, enabling the model to focus on words with actual sentiment significance.

Streamlined feature space enables eliminating unnecessary words, the feature space becomes more concise, reducing the risk of overfitting and improving training efficiency. Also, higher AUC scores, the refined dataset contributed to a higher AUC for logistic regression, increasing from 0.910 (without custom stopwords removal) to 0.933 (with additional cleaning). Overall, the additional cleaning steps allowed us to create a dataset that better captured sentiment relevant features, improving the accuracy of our sentiment classification model.

Exploring quirks in the text based data

As with other forms of data, text based data comes with its own unique challenges. Ours, specifically, came with a lot of minor flaws. Case discrepancies, commas, spaces, quotation marks, and so on. These all came up in the primary data analysis, where we printed some raw data, and were able to explore the reviews in their raw form. This gave us a strong idea as far as what we would need to be cleaning in the process of data refinement, before feeding it into our model.

- Preprocessing and Refinement:
  The text reviews undergo multiple preprocessing steps:
  - Lowercasing all text to ensure uniformity.
  - Tokenizing text into individual words.
  - Removing stop-words to eliminate common but non-informative words.
  - Applying stemming or lemmatization to reduce words to their root forms.
  - Vectorizing text data using the TF-IDF method, incorporating unigrams and bigrams to capture context.

3. Model set up

We experimented with a couple of different data splits. These were tested to explore the trade off between model training volume and reliability. This tradeoff largely related to overfitting, and failing to properly test our data. To our surprise, the 80/20 split seemed to perform the best, while we had all expected a more “balanced” approach of a 70/30 split to perform best. This could potentially be related to the size of our data – with so much data, the discrepancies between an 80/20 split and 70/30 split are not huge. We also attempted a 50/50 split, but immediately found this to be insufficient – we were clearly underfitting our model to our data.

We also created a validation set, which is typically used in hyperparameter training.

Below is a short summary of our set up and experimentation factors:

- Setup:
    - Data Splits: Various train-test splits (80/20, 70/30, and 50/50) were tested to evaluate the trade-offs between model training volume and reliability.
    - Validation Set: Used for hyperparameter tuning, ensuring the test set remained unseen until final evaluation.

- Experimentation Factors:
    - Hyperparameters: Regularization strength in logistic regression was optimized using grid search.
    - Metrics: Models were evaluated using accuracy, F1-score, precision, recall, and AUC.

4. The process

We followed a typical process, as far as preparing and feeding data into our model went. We began with what was discussed above : splitting up the dataset into optimal sections. We then went on to begin preprocessing the pipeline on the training set. It is important to note that we decided to apply transformations separately to validation and test sets to avoid data leakage during this process.

In the pipeline, we committed the following transformations:

1. String indexing

This function essentially takes binary text data, and converts it to integer values. In our dataset, it converted “positive” sentiments to 1, and “negative” sentiments to 0. This step is crucial to the efficiency and performance of our model – it understand numerical data extremely well, so it is beneficial to convert everything possible into numerical data.

2. Tokenization

This just split the review data columns into words, to help our stopwords function run.

3. Stop Word Removal

The reasoning behind this was discussed above, but this step was the actual implementation of removing stopwords.

4. Hashing TF

This function was used for a similar sentiment as the first function: to convert string data (our words) into integer vectors. This was solely done to help improve our model performance and understanding of the data we were feeding it.

5. IDF

This was, again, discussed above. It was essential this step went after the hashing TF, so we could apply this function on numerical data over text data

After this, we proceeded to train our data on **4 different models:**

1. Logistic regression

Logistic regression is a classification algorithm that predicts the probability of a binary outcome. In this case, it was used to classify the sentiment of the reviews as either "positive" or "negative."

Why Logistic Regression?
Logistic regression is very powerful for binary classification problems, such as sentiment analysis. It performs well on linearly separable data and is an efficient model.

2. Logistic regression with cross validation

Once the Logistic Regression model was implemented, we introduced **cross-validation to** see if we could improve the model's performance by making sure it properly generalized unseen data.

What is Cross-Validation?

Cross-validation is a technique used to assess how well a machine learning model performs on different sections of data. It reduces overfitting and makes sure that the model's performance is not too dependent on any train test split.

Why Use Cross-Validation?

- **Prevents Overfitting**: By training on multiple subsets of the data and testing on the remaining portion, cross-validation reduces the risk of overfitting
- **Reliable Performance Metrics**: it provides a more reliable measure of model performance
- **Helps Tune Hyperparameters**: Cross-validation allows for better tuning of hyperparameters, due to it's increased accuracy

3. Decision tree

A **Decision Tree** is a non-linear classifier. It splits features into regions based off of their values, creating a tree-like structure, where each node represents a "decision" based on a feature, each leaf is a label (in the context of our model, these labels were ' positive ' and ' negative. '

Why Use a Decision Tree?

- **Non-Linear Relationships**:Given this model is not linear, we can understand more complex, non-linear features of our data

- **Interpretability**: Decision trees are easy to interpret. They can be visualized to better understand their classifying process

4. Naive Bayes

The **Naive Bayes** classifier is a machine learning algorithm that employs Bayes theorem. It's known for its performance on text analysis problems, which encompases our goals with our model. Given its prominence in Binary text classification, we decided to use this as our baseline in training our LR model.

Why Use Naive Bayes?

- **Works Well with High-Dimensional Data**:
  In text classification, especially with TF-IDF, we end up with a lot of features. Naive Bayes is programmed to work well with this issue.
- **Handles Categorical Data**:
  Naive Bayes is based on conditional probability and works well with categorical data, which fits our problem description well.

Below is a brief summary of everything I have highlighted above:

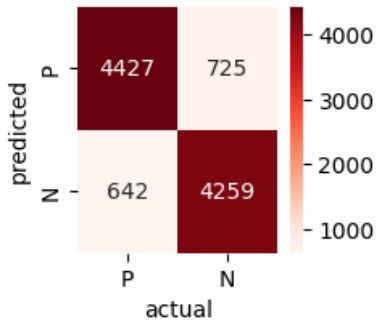
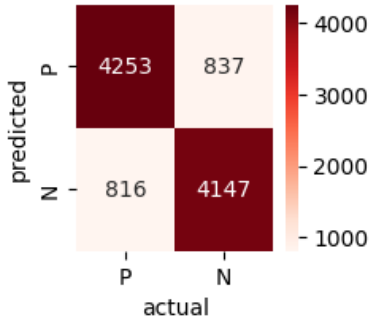
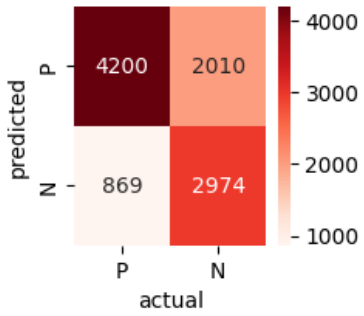
- Experiment process:
  - Split the dataset into training, validation, and test sets.
  - Fit preprocessing pipelines on the training set
  - Train logistic regression and compare it against Naive Bayes as a baseline.
  - Optimize hyperparameters and evaluate performance using cross-validation.

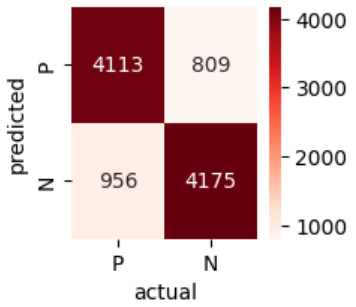
- Performance metrics:

Model	Accuracy	Precision	Recall	F-Score	AUC
Logistic Regression (w/ Cross Validation)	0.8640	0.8593	0.8733	0.8663	0.9331
Logistic Regression (w/o Cross Validation)	0.8356	0.8356	0.8390	0.8373	0.9106

Decision Tree	0.7136	0.6763	0.8286	0.7447	0.6777
Naive Bayes	0.8244	0.8356	0.8114	0.8233	0.4927

Confusion Matrices

Logistic Regression (w/ Cross Validation)	
Logistic Regression (w/o Cross Validation)	
Decision Tree	

Naive Bayes	

After trying the four different models, we concluded that the logistic regression with cross validation model was our best performing model. From there we did some testing to see how many folds of cross validation would give us the optimal performance. The following table shows the results.

Folds	**Accuracy**	**Precision**	**Recall**	**F-Score**	**AUC**
3	0.8671043469611062	0.860815774212256	0.878477017163148 5	0.869556727201718 3	0.933024364096495 9
5	0.8640206903411917	0.859277950310559	0.873347800355099 6	0.866255747969865 9	0.933118015526979 8
7	0.8640206903411917	0.859277950310559	0.873347800355099 6	0.866255747969865 9	0.933118015526979 8
10	0.8640206903411917	0.859277950310559	0.873347800355099 6	0.866255747969865 9	0.933118015526979 8
15	0.8640206903411917	0.859277950310559	0.873347800355099 6	0.866255747969865 9	0.933118015526979 8

From the above results, we saw that the AUC which we used as our metric to evaluate these models plateaued after 5 folds of cross validation and that any amount more would not have any performance increases that would necessitate the amount of time taken in training the models.

Results:

Key Findings:

- Data Cleaning Impact:
    - **Removal of Infrequent Words**

By removing majority of the infrequent words found in the reviews, we were able to reduce the feature sets dimensionality. This helped to improve our efficiency when actually running the evaluation but also decreased the noise in the datasets, leading to better performance when we ran the models.

- **Lowercasing and Noise Removal**
  The cleaning we did to remove differences in reviews such as lowercasing all the text to avoid mismatching, to ensure consistency across the models and features, which was further improved by cleaning techniques such as removing punctuation and any special characters. We were able to further reduce noise by using sparks stopword remover which removed any irrelevant or frequently occurring words, helping our models to focus on a simplified version of the data which was easier to analyze and less noisy.

  Overall our cleaning and preprocessing proved to dramatically improve the performance of all of our models mainly through the removal of noise and irrelevant details to achieve consistency.

- <u>IDF implications:</u>
  - Using IDF after we hashed our features helped to highlight the more important features which helped to improve precision and recall in our models, especially seen in our results from Logistic Regression.

- <u>Cross-Validation Advantage:</u>
  - After the addition of cross-validation in our logistic regression model, we saw an improvement in our AUC, going from ~91% AUC without cross validation to around ~93% AUC using cross validation. Therefore , our cross validated logistic regression model proved to be the best performing. These results proved our initial thoughts that the cross validation could help mitigate any overfitting that might be happening in our logistic regression model.
- <u>Data Splits:</u>
  - Based on the industry standards that we found in our research, and proven by the results we obtained, the 80-20 split between training and test data that we used proved to be an optimal split which was able to balance the accuracy of our models while still providing enough testing data to validate its performance.

- <u>Model Performance:</u>
  - From the above charts, we observed that the **Logistic Regression model with Cross-Validation** proved to be the best performing model when compared against Logistic Regression without Cross-validation, Naive Bayes and a Decision tree model. It achieved the following performance:
    - Accuracy: 0.8640
    - Precision: 0.8593
    - Recall: 0.8733
    - F-Score: 0.8663

    - AUC: 0.9331

- In our Confusion matrices, Logistic Regression using cross-validation proved to also minimize false negatives (642) and false positives (725) when compared against the other models, highlighting its high recall and precision performance.

- Conclusion:
  This project demonstrates the effectiveness of logistic regression for sentiment analysis when combined with thoughtful feature data engineering. By removing infrequent and balanced words, we significantly improved model performance while maintaining interpretability. Our model achieved an AUC of 0.933 with cross-validation, showcasing the importance of advanced preprocessing techniques. Future work could involve experimenting with deep learning models or applying these techniques to other sentiment analysis datasets for broader validation.

References:

Ganesan, K. (2023, March 16). *What are stop words?*. Kavita Ganesan, PhD. https://kavita-ganesan.com/what-are-stop-words/#:~:text=Stop%20words%20are%20a%20set,on%20the%20important%20words%20instead.

GeeksforGeeks. (2024a, July 12). *Cross validation function for logistic regression*. https://www.geeksforgeeks.org/cross-validation-function-for-logistic-regression-in-r/

GeeksforGeeks. (2024b, July 12). *Cross validation function for logistic regression in R*. https://www.geeksforgeeks.org/cross-validation-function-for-logistic-regression-in-r/

Jain, A. (2024, February 4). *TF-IDF in NLP (term frequency inverse document frequency)*. Medium. https://medium.com/@abhishekjainindore24/tf-idf-in-nlp-term-frequency-inverse-document-frequency-e05b65932f1d#:~:text=In%20the%20realm%20of%20Natural,information%20retrieval%2C%20and%20document%20classification.

Kinguistics. (2016, November 27). *Classifying user gender based on tweet text*. Kaggle. https://www.kaggle.com/code/kinguistics/classifying-user-gender-based-on-tweet-text

Singh, R. (2024, September 29). *Natural language processing: Linear text classification*. Medium. https://medium.com/@RobuRishabh/natural-language-processing-linear-text-classification-898f64a1e04a#:~:text=Logistic%20Regression%20is%20a%20linear,a%20logistic%20(sigmoid)%20function.

Encord. (n.d.). *How to split data into train, validation, and test sets: Best practices*. Encord. https://encord.com/blog/train-val-test-split/