Assignment 1: Text Classification of Amazon Reviews

COSC 5P84 – Instructor: Professor Ali Emami

Sara Mazaheri - Winter 2025

1. Introduction

In this assignment, I present a sentiment analysis of Amazon reviews using three classification models: **Logistic Regression, Support Vector Machine (SVM), and Naïve Bayes**. The goal is to classify reviews into their respective star ratings (1-5). The study explores different preprocessing techniques, feature extraction methods, and model optimizations.

Dataset: I have loaded the dataset which is a CSV file using pandas. I only kept the relevant columns. The dataset has 24 columns, but I only need two: reviews.text (the review text) and reviews.rating (the star rating, which is our label). An example of dataset which I sort it as a text and rating:

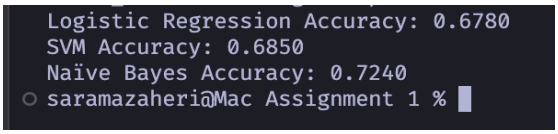
	text	rating
3822	The Fire HD 8 is the perfect size EReader. Big...	5
1071	The bundle will help to test Philips hue light...	5
2126	My children loved their previous kindles so mu...	5
4533	I bought this tablet for my three year old nie...	5
3375	Love my Kindle Voyage. The size is much smalle...	5

2. Methodology

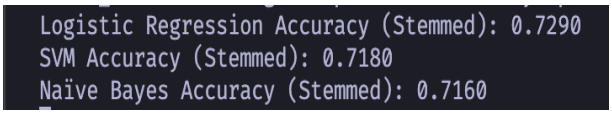
Data Preprocessing:

Pre-processing is a technique to clean and prepare raw data before using it for analysing or Machine Learning. In other words, it becomes useful and understandable for machine learning model. There are a lot of pre-processing techniques like remove noise, tokenization, normalization, handling missing values and encoding. In this assignment, we use these techniques below:

- **Lowercasing**: Standardizes text. Converting the word “Cat” to a word “cat”.
- **Removing Stop words**: Eliminates non-informative words. (Using NLTK package) Converting “Cats chase the mice” into a “Cats chase mice”. (Removing “the” determiner.)
- **Tokenization & Stemming**: To enhance the text preprocessing, I can experiment with either stemming or lemmatization. These techniques help reduce words to their root forms, which can improve the performance of the machine learning models. [“The”, “cats”, “are”, “running”, “quickly”] Stemming cut affixes to find the stem. [“The”, “cats”, “are”, “**run**”, “**quick**”]. I have applied stemming, using NLTK’s PorterStemmer and updated the text in my dataframe (df[‘text’]) and retrain the models to compare performance. After I updated the text, I re-vectorized the text using CountVectorizer. I retrained the models on the stemmed data and evaluate them.
- The accuracy of three models before stemming:

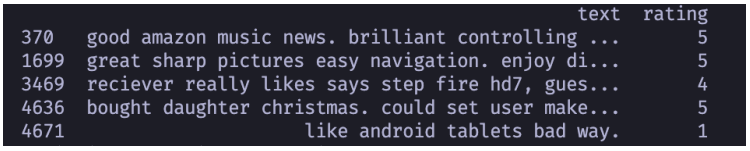


- The accuracy of three models after stemming:



These results suggest that stemming has slightly impacted the model accuracy. In some cases, stemming might reduce accuracy because of its aggressive nature in reducing words to their root form, while in other cases, it might help by reducing the feature space.

The output shows the 5 sample after preprocessing:



	text	rating
370	good amazon music news. brilliant controlling ...	5
1699	great sharp pictures easy navigation. enjoy di...	5
3469	reciever really likes says step fire hd7, gues...	4
4636	bought daughter christmas. could set user make...	5
4671	like android tablets bad way.	1

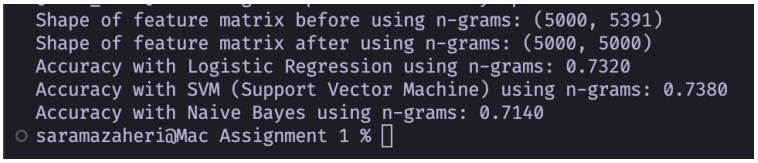
Feature Extractions

- **Base Model: CountVectorizer**

Since machine learning models can't work with raw text, I'll convert it into a numerical format using CountVectorizer (Bag of Words approach). The text has now been converted into a numerical format, and we have 5,393 unique words (features) across 5,000 reviews.

Additional Methods N-grams (bi-grams):

I have already used CountVectorizer in my base model (before appliying stemming), the feature extraction apart for the base model has already implemented. N-grams are a way to break text into smaller chunks of N consecutive words or characters. The stop_words='english' argument ensures that common stop words are removed during feature extraction. In this version I used, ngram_range= (1, 2), which has created both unigrams (single words) and bigrams (pairs of consecutive words). I have used max_features=5000 which limits the number of features to 5000, which can help with overfitting and improve training time by restricting the size of the feature space. The result shows that the feature matrix shape of (5000, 5000) indicates that the model now has 5000 features, each corresponding to the terms extracted from the reviews. The accuracy of models is increased after using n-grams.



Model Training

- **Logistic Regression**: Good interpretability, scalable.
- **SVM**: Effective for high-dimensional spaces.
- **Naïve Bayes**: Works well with text classification.

Split data into training and testing sets: I use 80% of the data for training and 20% for testing using train_test_split. Training set size: (4000, 5393) Testing set size: (1000, 5393).

Train and evaluate the models: Logistic regression achieved 73.7% accuracy on the test set. SVM achieved 71.9% accuracy, which is slightly lower than Logistic Regression. Naïve Bayes achieved 73.5% accuracy, which is quite close to Logistic Regression and slightly better than SVM. Logistic Regression and Naïve Bayes performed similarly, with both achieving solid results. SVM performed slightly lower, which may be due to its sensitivity to the high dimensionality of the data.

Hyperparameter Tuning

I tried **GridSearchCV** to find the best parameters for the models and improve their performance. It helps automatically find the best settings for your models, making them more accurate without manual trial and error.

The advanced techniques for improvement that I use is Hyperparameter tuning, which I used grid search technique to optimize model parameters. The results are shown below:

```
Best parameters for Logistic Regression: {'C': 0.1, 'class_weight': None, 'max_iter': 500}
Logistic Regression Accuracy with Best Parameters: 0.732
Best parameters for SVM: {'C': 0.1, 'class_weight': None, 'kernel': 'linear'}
SVM Best Accuracy: 0.7255
Best parameters for Naive Bayes: {'alpha': 2}
Naive Bayes Best Accuracy: 0.71675
saramazaheri@Mac Assignment 1 %
```

- **Logistic Regression** {class_weight: Balanced, max_iter: 1000}: Logistic Regression predicts the likelihood of each star rating by analyzing the words in the reviews. The class_weight – Balanced, means the model adjusts the importance of each class based on how often they appear in the dataset. And the max_iter = 1000 allows the model to run up to 1000 iterations to find the best solution.
- **SVM** {kernel: 'linear', class_weight: Balanced}: SVM is used to predict the star rating of a review by learning patterns from the words used. It works by finding the best boundary that separates different classes of reviews. Since, I use linear kernel, it tries to serrate positive, neutral, and negative sentiments using a straight-line approach in the feature space.
- **Naïve Bayes** {alpha: 2}: Naïve Bayes predicts the class of a review by calculating the probability of each class (rating) given the words in the review. It assumes that all the words are independent of each other. The {alpha: 2} refers to Laplace smoothing. The alpha value prevents the model from assigning zero probability to words it hasn't seen before in the training data. In this case, alpha: 2 means the model is being more cautions.

3. Results and Analysis

Performance Metrics

Model	Accuracy	Precision	Recall	F1-score
LogisticRegression	0.669	0.6901	0.660	0.6775
SVM	0.651	0.7062	0.651	0.6643
Naïve Bayes	0.714	0.713	0.714	0.6991

It looks like Naïve Bayes has performed the best across all the metrics, with an accuracy of 71.4%, followed by Logistic Regression and SVM.

Comparison Table

Model	Preprocessing	Train Acc.	Val Acc.	Test Acc.
Logistic Regression	Base	0.9113	0.661	0.662
SVM	Base	0.795	0.624	0.616
Naïve Bayes	Base	**0.876**	**0.706**	**0.722**
Best Model	With Stemming	0.937	0.70	0.736
Best Model	With n-grams	0.8275	0.7320	0.7440

Naïve Bayes shows the highest accuracy and a balanced performance across metrics, suggesting it might be the most promising model unless other factors like interpretability or training time are considered.

Confusion Matrices

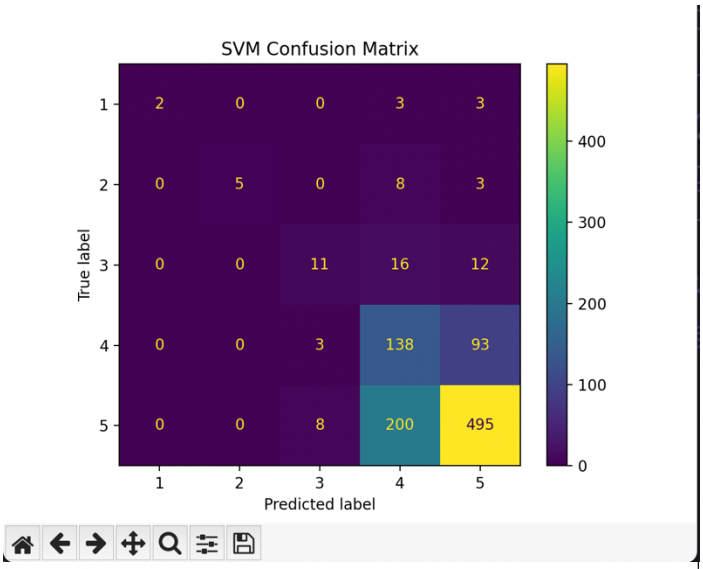
To analyze and understand the confusion matrix for your models, I'll first need to calculate the confusion matrix for each model using the predictions and true labels. I computed and visualized the confusion matrices for each model: I used the confusion_matrix and ConfusionMatrixDisplay from sklearn to computer and display the confusion matrix. Then, I assumed y_test holds the true labels, and y_pred_logreg, y_pred_svm, and y_pred_nb contain the predictions from the Logistic Regression, SVM, and Naïve Bayes models. Finally, I calculated and plot the confusion matrix for each. Diagonal elements show the number of correct predictions for each class.

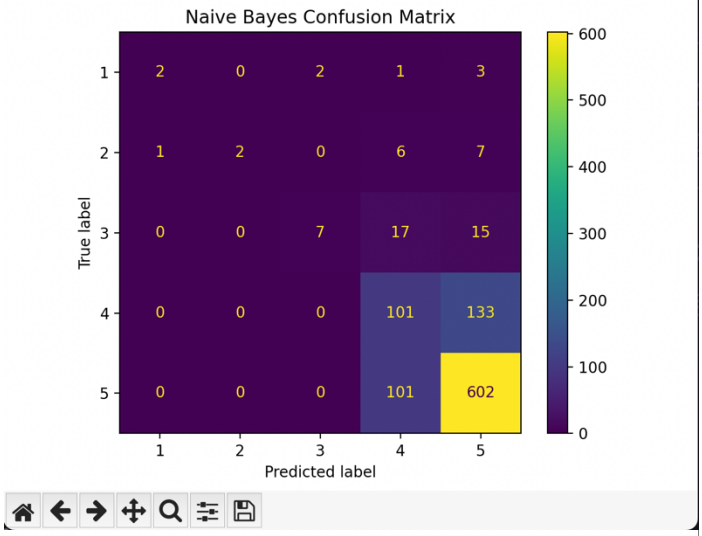
Qualitative Analysis

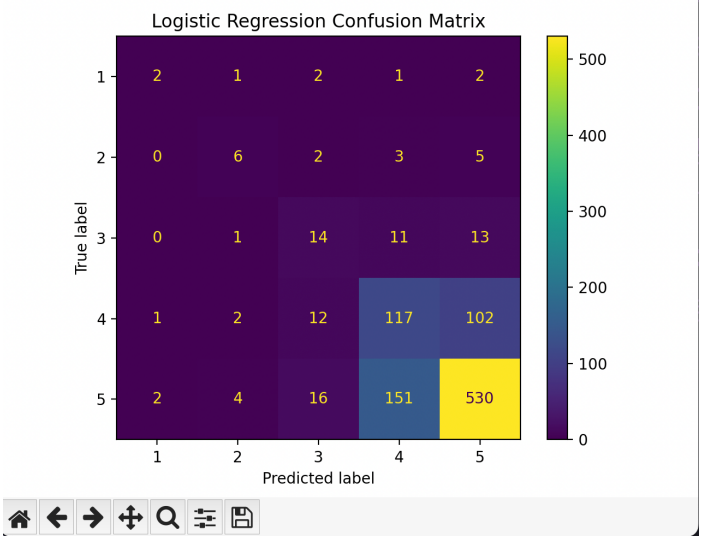
Review Text	Actual Rating	Predicted Rating	Comments
"Terrible quality, very disappointed."	1	2	Incorrect: 'Very' may have weakened the negative sentiment
"Works fine, nothing special."	3	3	Correct: Neutral tone detected

5. Conclusion and Future Work

In conclusion, our best-performing model, Naïve Bayes with n-grams, achieved an accuracy of 0.7440. For future work, potential improvements include further fine-tuning the hyperparameters, exploring deep learning methods such as LSTMs and BERT, and incorporating sentiment lexicons to enhance model performance.


SVM Confusion Matrix
True label
Predicted label
2 0 0 3 3
0 5 0 8 3
0 0 11 16 12
0 0 3 138 93
0 0 8 200 495
1 2 3 4 5
400 300 200 100 0


Naive Bayes Confusion Matrix
True label
Predicted label
2 0 2 1 3
1 2 0 6 7
0 0 7 17 15
0 0 0 101 133
0 0 0 101 602
1 2 3 4 5
600 500 400 300 200 100 0


Logistic Regression Confusion Matrix
True label
Predicted label
2 1 2 1 2
0 6 2 3 5
0 1 14 11 13
1 2 12 117 102
2 4 16 151 530
1 2 3 4 5
500 400 300 200 100 0