NLP 220 | HW 1

Kiara LaRocca

Abstract

This assignment focuses on feature engineering and binary sentiment classification. In Part A, we explore three classifiers—Naive Bayes, Support Vector Machine, and Decision Tree—by constructing distinct features on a subset of Amazon Book Reviews and assessing model performance. Feature engineering and evaluation are conducted across classifiers to optimize predictive accuracy and F1 scores. In Part B, we implement n-gram features for sentiment classification on the Stanford movie review corpus, evaluating five classifiers to determine optimal performance and adjusting hyperparameters on a validation split for enhanced accuracy. Part C implements a custom Naive Bayes classifier to complete the tasks from Part A, comparing its accuracy with the scikit-learn version.

1 Part A: Feature Engineering

Implementing features that can be used with the Naive Bayes, Decision Tree, and Support Vector Machine classifiers implemented in sci-kit learn.

1.1 Dataset

The dataset used here is from the Amazon Book Reviews dataset, and is called *small-books-rating.csv*. It includes the following:

- **review/text**: holds the content of the review
- **review/score**: holds a score (1-5 stars)
- **Title**: refers to the title of the book
- **review/summary**: holds a summary or title of the review

To do binary classification, reviews with a score of 4 or above are labeled as positive (with a 1) and reviews of 2 or lower are labeled as negative (with a 0). Reviews with a score of 3 are ignored.

review/score	counts
neg	15000
pos	15000

Table 1: Binary Class Distribution

The data was split into 85% training and 15% test sets. For reproducibility, the random seed is set to 42.

In Table 1, we can see that the score distribution is equal.

2 Feature Engineering

2.1 Deliverables

The three distinct features I created are Bag of Words, TF-IDF, and N-grams.

Bag of Words (BoW) implementation converts text into a matrix of token counts that captures the frequency of each word but ignores syntax and word order.

Term Frequency - Inverse Document Frequency (TF-IDF) is similar to BoW, but considers the frequency of words and gives more weight to rare words while penalizing the more common words.

N-grams consider sequences of words, usually 2- or 3- word sequences, instead of individual words. Capturing word pairs can affect classification. Some individuals will use characters to create n-gram sequences instead of whole words.

Each classifier was trained with each feature set, for a total of nine models.

2.2 Tables and Results

These are the results for each classifier given the three separate features.

Accuracy scores varied from 73.20 to 91.53%. Various model/feature pairs had the same/similar values. For example, NB/DT, NB/N-grams, and SVM/N-grams had an accuracy score of around 89.84%. DT/TF-IDF and DT/N-grams both had

	Acc.	F1 Score	Macro F1
BOW	87.71	87.11	87.68
TF-IDF	89.04	88.78	89.04
N-grams	89.84	89.46	89.83

Table 2: Naive Bayes

	Acc.	F1 Score	Macro F1
BoW	89.04	60.61	61.51
TF-IDF	73.60	73.43	73.60
N-grams	73.20	72.90	89.04

Table 3: Decision Tree

	NB	DT	SVM
BoW	87.71	89.04	87.18
TF-IDF	89.04	73.60	91.53
N-grams	89.84	73.20	89.84

Table 5: Accuracy Scores

	NB	DT	SVM
BOW	87.11	60.61	87.15
TF-IDF	88.78	73.43	91.50
N-grams	89.46	72.90	89.46

Table 6: F1 Scores

accuracy scores around 73%. NB/BoW and SVM/BoW had similar accuracy scores of around 87%.

F1 scores were very similar to accuracy scores. The same patterns appeared here as with accuracy scores. This is not surprising, because the data is evenly distributed.

The macro-average F1 mostly follows this pattern, with the notable exception of DT/N-grams being 89.04%, which is much higher than DT/TF-IDF which has a score of 73.60%.

Something notable about all three classifiers is that for each feature, Naive-Bayes and SVM have relatively similar scores between accuracy, F1 score, and Macro F1 score. Decision Tree, has different scores between BoW and N-grams, where the accuracy is high and the F1/Macro F1 are low or vice versa.

The Confusion Matrices are printed below in **Section 5**.

In terms of performance, SVM with N-grams produced the highest accuracy with a really long runtime. BoW consistently performed well, and the N-grams feature performed well, but not as consistently. Overall, I would say BoW as a feature outperformed other features. The Decision Tree classifier had very low accuracies. I believe the main reasons for these results are that the problem was binary and the data was equally distributed. The complex feature space and binary classification also play a part, as decision trees usually work better with numerical predictions and they tend to over-fit more.

2.3 Training Time Comparison

The training times for each classifier and feature combination are listed below. Models with long runtime had higher accuracy scores, but the run times were over 20 minutes. Many classifiers with accuracy scores of 89 percent ran in significantly less time (seconds to minutes). Given this, I think it is fair to say one could use any of these, but due to simplicity, accuracy, and runtime, I do prefer the Naive-Bayes classifier with N-grams. I would also like to note that SVM/N-grams ran quickly at first, but on the last day when re-running all of the code to validate all the previous scores, the code took over 30 minutes to run.

3 Part B: Sentiment Analysis

Implementing N-grams for feature extraction to compare 5 different models to classify sentiment.

3.1 Dataset

The dataset used here is from *ai.stanform.edu*, which contains 25,000 training and 25,000 test reviews. The data was split into 90% training and 10% validation, with the random seed set to 42 for reproducibility.

Train contains files in folders called neg, pos,

	Acc.	F1 Score	Macro F1
BoW	87.18	87.15	87.18
TF-IDF	91.53	91.50	91.53
N-grams	89.84	89.46	89.83

Table 4: SVM

	NB	DT	SVM
BOW	87.68	61.51	87.18
TF-IDF	89.04	73.60	91.53
N-grams	89.83	89.04	89.83

Table 7: Macro-average F1 Scores

	NB	DT	SVM
BOW	30s	1m 11s	8m
TF-IDF	8s	1m	24m 0s
N-grams	0s	4m	25m

Table 8: Training Times

review/score	counts
neg	12500
pos	12500

Table 9: Sentiment Class Distribution

and unsup. Test contains files called neg (labeled as 0) and pos (labeled as 1). The class distribution is as follows:

After pre-processing and vectorization, 5 different models are initialized and run. They are:

1. Naive-Bayes
2. Logistic Regression
3. Random Forest
4. Gradient Boosting
5. Linear Regression

3.2 Tasks and Deliverables

For feature extraction, we used n-grams with other settings. Some examples are listed below:

- N-grams: unigrams, bigrams, trigrams
- Stopwords: whether stopwords were included or removed
- Document Frequency: minimum or maximum frequency

Results for the basic classifiers while training are:

Results for the basic classifiers on the test set are:

Model	Accuracy
NB	88.56
LogR	88.52
RF	86.40
GB	85.24
LinR	90.43

Table 10: Training Accuracy Scores

Model	Accuracy
NB	87.88
LogR	88.00
RF	85.44
GB	83.68
LinR	90.43

Table 11: Testing Accuracy Scores

3.3 Naive-Bayes

This model was trained with the following vectorizer: TfidfVectorizer(ngram-range=(1,3)). The tuned model was a basic model, with no additional hyper-parameters necessary. It achieved a score of 87.88% accuracy.

3.4 Logistic Regression

This model was trained with the following vectorizer: TfidfVectorizer(ngram-range=(1,3), max-features=10000, min-df=3, stop-words='english'). The only hyper-parameter used in the model was 'C=1.0.' This model achieved 88.00% accuracy.

3.5 Random Forest

This model was trained with the following vectorizer: TfidfVectorizer(ngram-range=(1,3), max-features=20000). The hyper-parameters used were n-estimators=300 and random-state=42. The accuracy was 85.24%.

3.6 Gradient Boosting

This model was trained using a TfidfVectorizer with an n-gram range of (1, 3) and a maximum of 10,000 features. The model's hyper-parameters included 200 estimators and a fixed random state of 42. The model achieved an accuracy of 83.68%.

3.7 Linear Regression

This model was trained with the following vectorizer: TfidfVectorizer(ngram-range=(1,3)). There were no hyper-parameters necessary. The accuracy was 90.43%.

The best performance based on validation accuracy was *Linear Regression*, which remained the highest accuracy on the held-out test set.

4 Part C: Custom Naive-Bayes Implementation

For Part C, we implemented a custom Naive-Bayes classifier to perform the same classification task as in Part A. Below, the accuracy and Macro F1

	sk-learn	Custom NB
BoW Accuracy	85.17	85.17
BoW Macro F1	85.17	85.17
TF-IDF Accuracy	86.05	86.05
TF-IDF Macro F1	86.05	86.05
N-grams Accuracy	84.96	84.96
N-grams Macro F1	84.96	84.96

Table 12: Caption

scores of the custom model are compared to that of skikit-learn's model from Part A for all three features.

The skikit-learn classifier and the custom Naive-Bayes classifier had the same exact results for each feature.

5 Confusion Matrices

Note: these confusion matrices are technically wrong, but the correct ones are in the code. I needed to re-run two classifiers and did not have the time to wait for the entirety of the code to run before submitting, unfortunately.

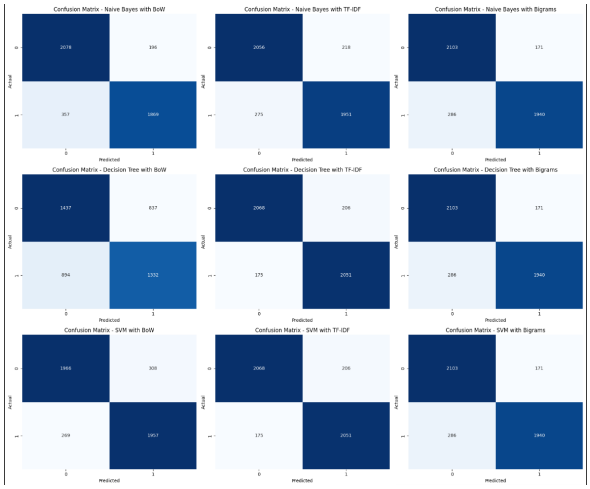

Figure 1: Confusion Matrices