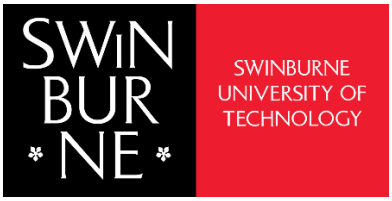
Student ID: 103128918

SWINBURNE UNIVERSITY OF TECHNOLOGY



MASTER OF DATA SCIENCE

COS80027 Machine Learning

Semester 1, 2025

Assignment 1: Sentiment Analysis on

Word count: 2278

Workshop Tutor: Ms Atefeh Azin Kousha

Submitted by: Thi Ngan Ha Do

Student ID: 103128918

4.1 Bag-of-Words Design Decision Description

1) Data cleaning and standardisation

For normalization and cleaning, I created function clean(), which includes lower casing the texts using lower() method and removing punctuation using translate() and str.maketrans(), in which str.maketrans() creates a translation table mapping characters to delete and translate() applies that translation table to substitute those characters with none. I applied this cleaning function to the training and test sets using pandas'.apply() method.

2) Vocabulary

To build the final vocabulary, I used scikit-learn's TfidfVectorizer, which transforms the text into numerical features through term frequency–inverse document frequency (TF-IDF). This method differs from CountVectorizer, where it counts the number of times a word is repeated, but this also accounts for how common or uncommon a word is in all the documents. This makes the model give more weight to words that contribute information to distinguishing between other categories of reviews.

- Words inclusion and exclusion

Issue	Method	Justification
Stop word removal	Removed using stop_words from ENGLISH_STOP_WORDS	Common words are likely to carry little informative value for discriminating between classes, and removing them reduces noise and dimensionality [1]
Negation words inclusion	Kept "not", "no", "nor", "never" by modifying the stop-word list	Removing negators may lead to misinterpretations, such as equating "not good" with "good"
Rare words	Excluded words appearing in fewer than 2 documents (min_df=2)	Rare words, either typos or outliers, can add unnecessary noise
Frequent words	Excluded words appearing in more than 70% of documents (max_df=0.7)	Frequent terms can dilute the impact of more meaningful words
Tokens	Limited to alphabetic tokens ≥2 characters (token_pattern=r"[a-zA-Z]{2,}")	Removed numbers, symbols, and single-letter tokens which typically do not carry semantic meaning

Table 1. Vocabulary filtering strategies and Justifications

- Vocabulary size

We can view the vocabulary size by printing word_tfidf.get_feature_names_out(), which is a list of all the words present in the final vocabulary after preprocessing and filtering. Then, len() was called to determine how many words are present in that list, and this provides us with the vocabulary size. With a vocabulary size of 2111 words out of 2400 text documents, the output is justifiable.

- Out-of-vocabulary words

In order to make sure the model learns solely from the training data, I used fit_transform() on the training data (x_train). This builds the vocabulary and computes the importance of each word solely on the basis of what appears in the training reviews. Then, when I did transform() on the test data (x_test), it used the same training vocabulary and ignored any new words it had not seen before. These unseen words are actually handled as zeros in the feature matrix, i.e., they do not affect the model's prediction.

This is a good practice because it avoids data leakage and keeps the test set completely isolated, hence the evaluation remains fair and realistic.

3) N-grams

The parameter ngram_range=(1, 2) was set specifically to pull both single-word tokens (unigrams) and two-word expressions (bigrams). Incorporating bigrams as well as unigrams enriches the feature space so that the model can gain a richer context and catch more nuanced linguistic patterns that would be lost if only unigrams were used. Phrase-level sentiment such as "not good," "very bad," or "highly recommend" contain meaning outside of their component words.

4) Character TF-IDF

As word-level features tend to miss important details, especially if there are spelling mistakes, slang, or creative expressions, I also included character-level features, with analyzer='char' and ngram_range=(3, 5).

Character-level TF-IDF breaks the text down into overlapping character sets (n-grams), enabling the model to learn sub-word patterns that are likely to carry substantial meaning. For example, prefixes like "un-" or "dis-" and suffixes like "-less" or "-ful" often predict polarity or sentiment. Also, in online reviews, people tend to use nonstandard spelling or stress exaggeration, such as "soooo bad". Word-level vectorizers will learn these as new tokens and may drop them if they're not common, but character n-grams like "eee", "rea", and "aat" are still regular in similarly exaggerated statements. That way, the model learns to generalize more and identify emotion or stress even if the spelling is incorrect.

5) Domain specification

The dataset includes reviews from three different domains (Amazon, IMDB, and Yelp) which may differ in vocabulary, tone, and how sentiment is expressed. To help the model account for these differences, I included the domain of each review as an extra feature by applying OneHotEncoder and converting each domain into a separate binary feature.

6) Combine features

Finally, all three kinds of features were combined all into a big feature matrix using hstack() from SciPy, which positions the features alongside each other in sequence (one after another horizontally).

4.2 Cross Validation and Hyperparameter Selection Design Description

1) Train-Validation Split

To find the best split ratio, I experimented with two train-validation ratios: 80/20 and 90/10. The idea was to see how much training data the model needs to learn well while preserving sufficient data aside for stable validation while tuning hyperparameters. A 90/10 split gives the model more training data but reduces the size of the validation set. My findings had smaller differences between test accuracy and validation accuracy in the 80/20 split, which means the model was generalising better and not fitting as well to training data. The 90/10 split showed a larger drop (almost 2%) from validation to test accuracy, which could mean the smaller validation set made the tuning slightly too optimistic.

2) Classification model selection

Support vector machines (SVM) are a class of supervised learning models that are particularly suited for binary classification issues and high-dimensional feature spaces, both of which are common in text classification [2]. In high-dimension spaces such as those created by bag of words, SVMs are helpful even when the number of features exceeds the number of samples. Within the SVM family, LinearSVC is a variant for linearly separable problems. For this task, where the feature vectors are constructed from TF-IDF representations of unigrams, bigrams, and character n-grams, the data is linearly separable to a large degree, and the interpretability and training time afforded by LinearSVC makes it a suitable choice.

3) Performance Metric

The hyperparameter search was optimized based on accuracy, which directly measures the proportion of correctly classified samples on the held-out validation data. Accuracy is calculated by the number of correct labels divided by total predictions, and it is a good choice in this case as the classes in the dataset are reasonably well balanced (50/50). Also, the goal of this sentiment classification task is to label as many reviews correctly as possible, either positively or negatively.

4) Hyperparameter Search

Hyperparameter tuning was performed using a Grid Search strategy (GridSearchCV), which is a method that tries all possible combinations from a pre-defined grid of parameter values. For parameters, GridSearchCV searched over different values for the regularization strength C, the loss function (hinge and squared_hinge), the class_weight parameter, and tolerance values (tol). Another consideration was to use RandomizedSearchCV, but the search space is quite small in this case, and it was possible to go through all combinations to find the best one.

5) Source of Held-out Data and Cross-Validation

GridSearchCV also performed cross-validation (5-fold CV) on each setting, which gives a rough estimate of the performance of each setting on new data. That is, the training set was split into 5 equal-sized folds, with each fold containing approximately 10% of the training data (both in the 80/20 and 90/10 split). The model was trained on 9 of the folds and tested on the other fold, and this was done 5 times so that each fold was used once as a validation set. The overall performance score for each hyperparameter setting was calculated by averaging the accuracy over all 5 folds.

The held-out validation set was created by dividing the initial training data through stratified sampling using train_test_split() from scikit-learn. Stratification combined label and domain information explicitly to preserve class and source balance in the validation split.

4.3 Hyperparameter Selection Figure for Classifier

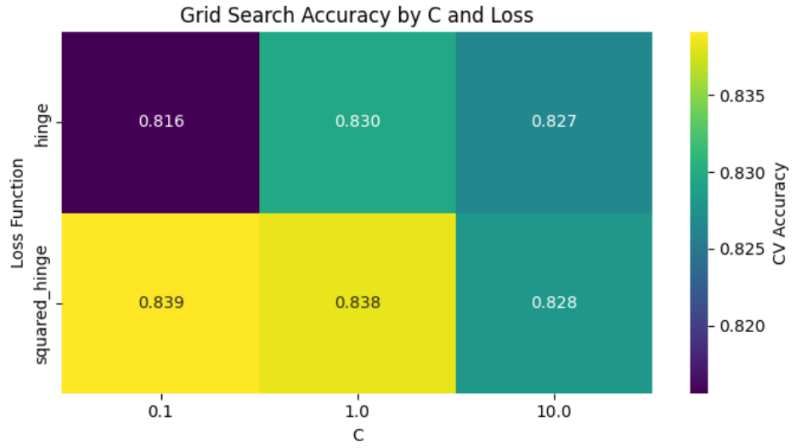


Figure 1. Grid Search Cross-Validation Accuracy for LinearSVC

Figure 1 Observations:

GridSearchCV tested different combinations of the regularization parameter C and loss functions, with cv=5 used for model assessment. Heatmap summarizes the average accuracy across folds for each set of parameters. The highest accuracy (0.838) was achieved with C=0.1 and loss='squared_hinge', which was used as the final model parameter.

The best hyperparameters selected through grid search were: C=0.1, class_weight='balanced', loss='squared_hinge', and tol=0.001. To determine the most suitable number of folds for cross-validation, I compared performance across different cv values [3]. While all values produced similar accuracy, cv=5 gave the highest mean score (0.8385) with the lowest standard deviation (0.0029), indicating a strong balance between performance and evaluation stability.

4.4 Analysis of Predictions for the Classifier

From the misclassified samples, several consistent trends are observed in the error patterns of the model. One is a tendency to struggle with handling mixed or conflicting sentiment in one review. In Example 1, the model likely sensed the positive first sentence and did not catch the negative outcome if any. Another challenge is handling negation and emphasis. A very positive review like Example 2 was misclassified as negative, suggesting the model may misinterpret repeated negation. The model also tends to rely on surface sentiment words (Example 3) which was actually part of a negative review.

Item	Domain	Reviews	Ground truth	Predicted
1	Amazon	This item worked great, but it broke after 6 months of use.	Negative	Positive
2	Amazon	No shifting, no bubbling, no peeling, not even a scratch, NOTHING!I couldn't be more happier with my new one for the Droid.	Positive	Negative
3	IMDB	The only consistent thread holding the series together were the amazing performances of Leni Parker and Anita LaSelva as the two Taelons in quiet ideological conflict.	Negative	Positive
4	IMDB	Don’t be afraid of subtitles it’s worth a little aversion therapy 10/10	Positive	Negative
5	Yelp	My boyfriend and I came here for the first time on a recent trip to Vegas and could not have been more pleased with the quality of food and service	Positive	Negative

Table 2. Examples of Misclassified Reviews with Domain Context

To further analyse the limitations of the model, I examined the three most apparent issues.

1) Sentence Length

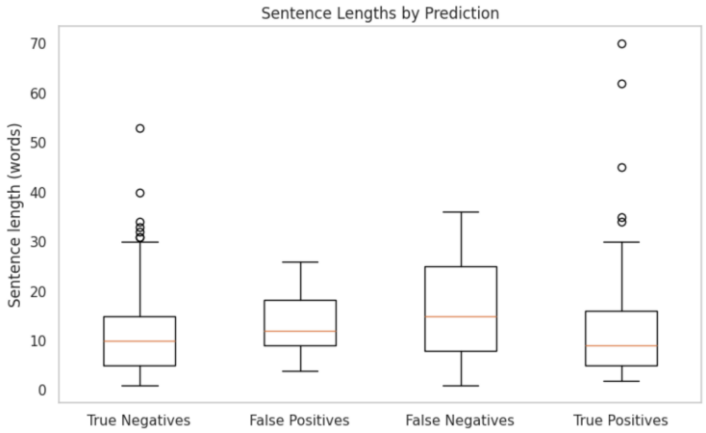


Figure 2. Distribution of Sentence Lengths by Prediction Outcome

Figure 2 Observations:

Correct classifications also include shorter average sentence lengths. The median of both is unmistakably lower than that for the incorrect predictions.

False negatives have the longest average sentence lengths, with a broader interquartile range and lots of outliers larger than 60 words. False positives also have slightly longer sentences than True Negatives but less variability than False Negatives.

Key takeaways: The model performs more efficient with shorter and simple reviews because such reviews carry stronger indicator signals of the sentiment. Lengthy reviews would likely carry mix sentiments, suspicious words, and context-specific terms which are liable to confuse labelling, especially that of positive sentiment [4].

2) Review Categories

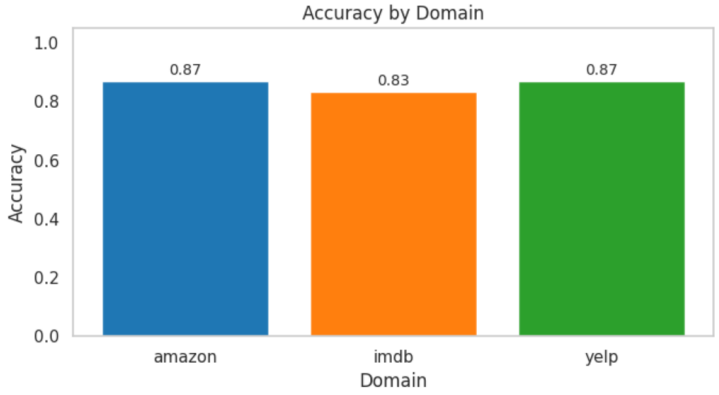


Figure 3. Model Accuracy Across Review Domains

Figure 3 Observations:

The model performs better on Amazon and Yelp reviews, both achieving an accuracy of 0.87, compared to IMDB reviews, which show a slightly lower accuracy of 0.83.

Key takeaways: The model performs poorer on IMDB reviews, as they tend to be longer, more complex, and often include mixed or nuanced sentiments, which can be harder for models trained on Bag-of-Words features to classify correctly. In contrast, Amazon and Yelp reviews may contain more direct and sentiment-heavy language, making them easier to interpret using surface-level lexical features.

3) Negation words

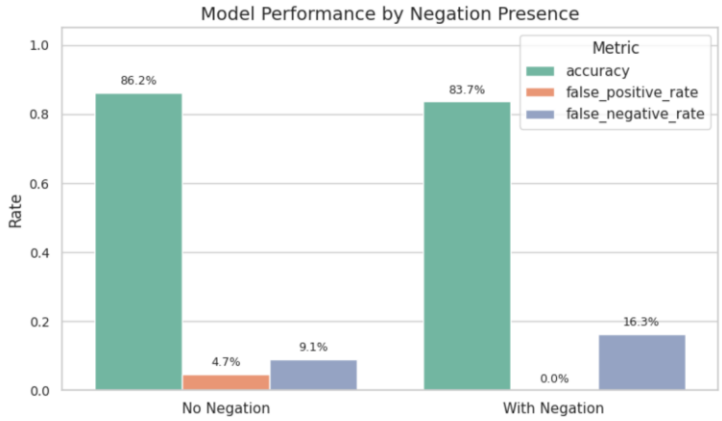


Figure 4. Model Performance on Sentences With and Without Negation

Figure 4 Observations:

Accuracy drops from 86.2% (no negation) to 83.7% (with negation). False negative rate increases sharply from 9.1% to 16.3% in sentences containing negation, while false positive rate drops to 0% in sentences with negation.

Key takeaways: The model tends to miss positive sentiment when it's expressed through negation (and incorrectly classify as negative. Negation introduces complexity, which the model struggles to handle, despite custom preprocessing steps to preserve key negation words [5]. The false positive rate being 0% with negation indicates that the model is being overly cautious, predicting negative when it detects negation, even if the overall sentiment is actually positive.

4) Strategies to improve model performance

Based on the errors of the model, several strategies can be used to enhance its performance. By enhancing its control over language structure, review length, and variation across review sources, these techniques try to enhance the model's accuracy and flexibility:

- Expand to trigrams or dependency-based features to better capture complex negation patterns
- Chunk long reviews into smaller parts, classify them individually, and combine the results
- Add text length as a feature
- Apply custom preprocessing per domain
- Train separate models for each domain

4.5 Performance on Test Set

The bar chart compares validation and test performance across key evaluation metrics (accuracy, precision, recall, and F1 score). Overall, the model shows no signs of overfitting and generalizes well to unseen data:

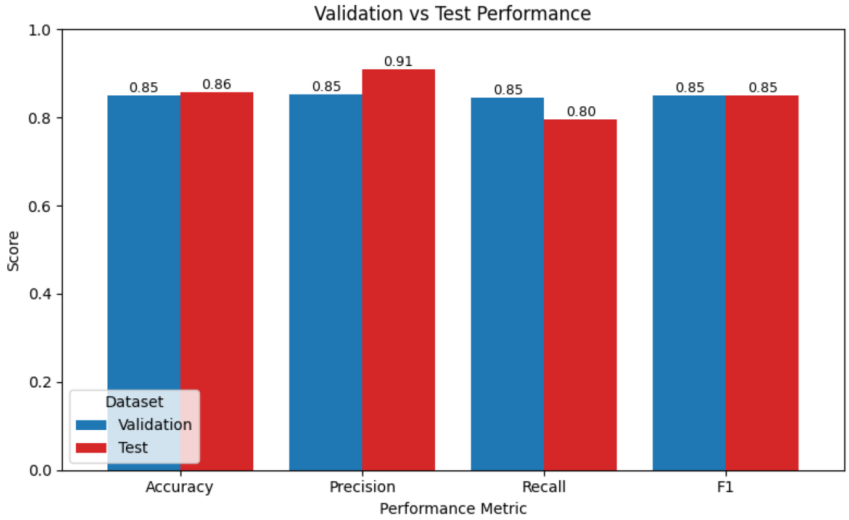


Figure 5. Model Evaluation on Validation and Test Sets

- Accuracy: Consistent between validation and test
- Precision: Improved on the test set, showing the model is effective at reducing false positives
- Recall: Slight drop from validation to test, showing the model misses more actual positives during testing
- F1 Score: Consistent between validation and test

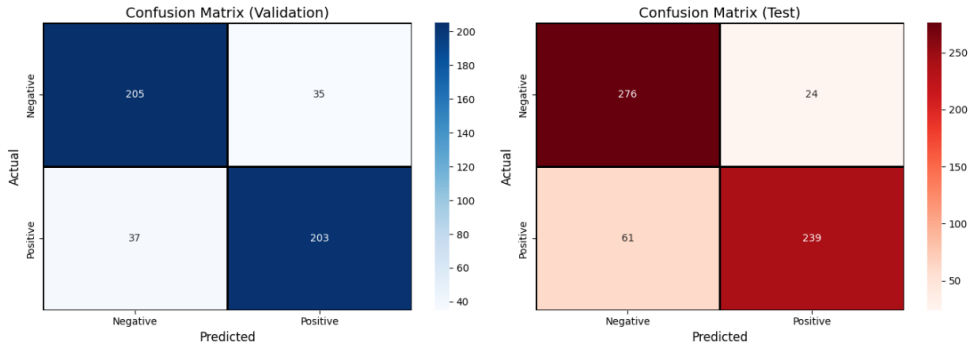


Figure 6. Confusion Matrices for Validation and Test Sets

The confusion matrices show a balanced classification performance across both the validation and test sets, with a slightly more conservative tendency, favoring precision (fewer false positives) over recall (slightly more missed positives):

- Validation set: 205 true negative and 203 true positive, with a relatively low number of false positives (35) and false negatives (37).
- Test set: Better results for negative reviews (276 correct and 24 false positives), but made more false negatives (61).

Conclusion

This report develops a sentiment classification model on the basis of a Bag-of-Words approach and LinearSVC to classify Amazon, IMDB, and Yelp reviews. Text data was preprocessed and vectorized by TF-IDF with word and character n-grams, and negators were kept to retain sentiment cues. Domain data was included using one-hot encoding. A 5-fold cross-validation GridSearchCV was used in order to tune the crucial hyperparameters (C, loss, tol, and class_weight). Among different train-validation splits attempted, the 80/20 split gave the optimal trade-off between accuracy and generalization. The model performed well overall, but error analysis indicated inaccuracy in longer reviews and multi-word negations.

Reference List

[1] Y. HaCohen-Kerner, D. Miller, and Y. Yigal, “The influence of preprocessing on text classification using a bag-of-words representation,” *PLoS ONE*, vol. 15, no. 5, p. e0232525, May 2020, doi: 10.1371/journal.pone.0232525.

[2] B. P. Yadav, S. Ghate, A. Harshavardhan, G. Jhansi, K. S. Kumar, and E. Sudarshan, “Text categorization Performance examination Using Machine Learning Algorithms,” *IOP Conference Series Materials Science and Engineering*, vol. 981, no. 2, p. 022044, Dec. 2020, doi: 10.1088/1757-899x/981/2/022044.

[3] P. Mukherjee, Y. Badr, S. Doppalapudi, S. M. Srinivasan, R. S. Sangwan, and R. Sharma, “Effect of negation in sentences on sentiment analysis and polarity detection,” *Procedia Computer Science*, vol. 185, pp. 370–379, Jan. 2021, doi: 10.1016/j.procs.2021.05.038.

[4] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning--based text classification: a comprehensive review," *ACM computing surveys (CSUR),* vol. 54, no. 3, pp. 1-40, 2021.

[5] I. K. Nti, O. Nyarko-Boateng, and J. Aning, “Performance of Machine Learning Algorithms with Different K Values in K-fold CrossValidation,” *International Journal of Information Technology and Computer Science*, vol. 13, no. 6, pp. 61–71, Dec. 2021, doi: 10.5815/ijitcs.2021.06.05.