Assignment 1 – Task 4 Report

Name: Marcus Tran

Student ID: 105149160

Subject code: COS80027

Subject name: Machine Learning

Semester 1

# Bag-of-words Design Decision Description

As with any raw data, text comments require preprocessing to be effective in downstream machine learning analysis. In this case, I used Bag of Words (BoW) representation with unigrams (single words) and word count as features. I cleaned and standardised the text data by applying a preprocessing pipeline: converting all words to lowercase (e.g. ‘Bad’ and ‘bad’ are treated the same), removing punctuations to reduce noise and removing English stop words (such as “and”, “the”, “is”) to focus on meaningful words that better indicate a positive or negative sentiment. This processing function was applied for both training data and test data.

After preprocessing, I applied CountVectorizer from scikit-learn, which extracted the vocabulary and keep track of the count of each word appearing in each comment. I chose to use unigrams (single words) and raw counts (which meant neither binary nor Term Frequency-Inverse Document Frequency – TF-IDF) to tabulate how frequently each word appeared in a document. CountVectorizer is simpler and more interpretable compared to TF-IDF. Raw counts could also be more effective for short texts – such was the case of analysing review comments. Furthermore, TF-IDF might undermine the importance of key words such as ‘good’, ‘bad’ since they appear more frequently and were good determinants for sentiment analysis. Indeed, opting for unigrams only could lose out on important context, for example ‘not good’ indicating a negative sentiment, however unigram becomes ‘not’, ‘good’ and loses out on the meaning combined by those two. Even so, unigrams allow for faster training and easier interpretation and could perform well as a baseline in tandem with simple classifier models.

The final vocabulary size was 4564 words, which was the number of unique tokens after preprocessing. I decided not to exclude words because the amount is not too big to hinder downstream analysis, and no exclusion could lead to better prediction. Notably, the vocabulary was composed of training data’s words, and the same vectorizer was used to transform the test set. Thus, any out-of-vocabulary words in the test set were ignored.

Overall, here is the pseudocode for the entire process:

**def preprocess(text):**

**text = text.lower() # convert to lower case**

**text = remove\_punctuation(text) # remove punctuations**

**text = remove\_stopwords(text) # remove English stop words**

**return text**

**vectorizer = CountVectorizer()**

**X\_feature\_train = vectorizer.fit\_transform(x\_train) # transform x\_train to the feature matrix**

**X\_feature\_test = vectorizer.transform(x\_test) # apply the transformation to x\_test set**

# Cross Validation and Hyperparameter Selection Design Description and Figure

I divided the training data into train set and validation set. More importantly, there were three ways the data was split by train/validation set: 80/20 (the standard ratio), 70/30 and 90/10. This was done to measure how robust the machine learning model was, whether the accuracies were consistent across different splits. Additionally, this could help with selecting the most suitable hyperparameter.

The classifier model chosen was Multinomial Naïve Bayes (MultinomialNB), which works well with high-dimensional text data. Also, this model pairs well with CountVectorizer - as done in the previous steps - since it uses the frequency of words as the predictor. Indeed, MultinomialNB is a simple algorithm, however it is fast and effective for a problem such as sentiment prediction from texts.

I used k-fold cross validation (CV) and a hyperparameter grid search to analyse performance on held-out training data. In MultinomialNB, the main hyperparameter is a smoothing parameter alpha, which prevents zero probabilities for words not seen in the training set but encountered in validation or testing data. I conducted grid search over these alpha values: [0.1, 0.5, 1, 2, 5, 10], where alpha = 1 is the standard value. I included smaller and larger values to ensure the grid search covered and compared enough alpha values. To evaluate these parameters, F1-score was chosen as it provided a balanced measure between precision and recall, providing a more nuanced assessment rather than using just accuracy alone. For each of the 3 splits, I applied 5-fold CV on the training set before applying to the validation set. I chose 5-fold because it was a good middle ground between sufficient data for training and computationally efficient.

5 folds split the data into 5 equal parts, each fold becomes the validation set once while the remaining 4 were used for training. To give an idea of how large a fold is, for instance, in the 80/20 split, 80% of the original train dataset, which was 1920 comments, now split into 5-fold, so each fold contained 384 comments. Subsequently, the tested hyperparameter would be used on the 20% remaining of the original set (480 comments) and the F1-score calculated. Across the 3 different splits, the alpha score 0.5 was chosen to be the most suitable, as the 80/20 and 90/10 split chose it, in addition with higher F1-score (Figure 1). Only 70/30 split chose alpha=1, however it achieved a lower F1-score than 80/20 and 90/10 splits. It was possible that there were less training data affecting the performance.

A computer screen with numbers and symbols

Description automatically generated

Figure : GridSearchCV result

# Analysis of Predictions for the Classifier

Subsequently, once the best hyperparameter was discovered, I applied alpha=0.5 to each of the 80/20, 70/30 and 90/10 splits to examine their consistency amongst each other. Figure 2,3,4 detailed classification reports and confusion matrix for each split.

A screenshot of a computer screen

Description automatically generated

Figure : 80/20 split evaluation result

# A screenshot of a computer screen Description automatically generated

Figure : 70/30 split evaluation result

# A screenshot of a computer screen Description automatically generated

Figure : 90/10 split evaluation result

80/20 split offered the best evaluation across different metrics, with weighted average of precision, recall, f1-score and accuracy being 0.83, followed by 70/30 split (0.81) and 90/10 (0.8). The evaluation metrics appeared consistent, solidifying the robustness of the model.

To evaluate what kind of mistakes the model was making, several false positives and false negatives were captured, respectively in Figure 5. From Figure 5, some false positives did not make sense, such as “Very disappointing” being labelled as positive (1 value), and the model prediction of it being negative (0) was reasonable, as without more context it appeared to be a negative sentiment. Unfortunately, BoW preprocessing step could transform negative sentences such as “I would not recommend this item to anyone” to become “would recommend”, lowering the model prediction capability.

A screenshot of a computer program

Description automatically generated

Figure : False positives and negatives of 80/20 split

Despite these setbacks, 0.83 accuracy seemed to be effective, as the MultinomialNB did better on sentences without negation words (Figure 6), as it predicted shorter positive sentences adequately, and even negative sentences such as “That one didn’t work either”.

A screenshot of a computer

Description automatically generated

Figure : Accurate prediction of 80/20 split

Future adjustments could be the inclusion of bigrams to ensure combined words express itself better, especially “not” in combination with an adjective.

# Performance on Test Set

To obtain the final model, I retrained the classifer using the entire training set (X\_feature\_train and y\_train\_series) with alpha=0.5 – the best parameter, then applied to the test set. I wanted to make sure that the model benefited from the largest possible training data before evaluation on unseen data. Figure 7 contained the evaluation metric of test set.

A screenshot of a computer screen

Description automatically generated

Figure : Test Set evaluation result

Even though the metrics were lower compared to 80/20 training split, they appeared consistent with the training set splits as a whole – with values around 0.80-0.82 – reflecting robust generalisation of the model.

Some potential differences could be attributed to the presence of out-of-vocabulary words in the test set, as CountVectorizer fitted only on the training data. Even so, MultinomialNB maintained stable performance, showing that the model generalised well even without seeing all possible vocabulary during training.