Assignment 1

Ahnaf Mozib Samin, Ayça Avcı, Shantanu Nath
(s4996763), (s4505972), (s4998405)
a.m.samin@student.rug.nl, a.avci@student.rug.nl, s.n.nath@student.rug.nl

1 Introduction

In this assignment, we train a machine learning model called Naive Bayes classifier to classify text retrieved from Twitter. The model performs two classification tasks namely on the data set:

1. **Binary classification (sentiment analysis):** Trained model classifies text data as "neg" (negative) or "pos" (positive).
2. **Multi-class classification:** Trained model classifies text data as "books", "music", "camera", "health", "dvd" or "software".

Experiment is using two different feature vectors for each classification tasks. Each feature vector is generated by the following tools provided by scikit-learn library in Python:

1. **Bag of Words (Count) Vectorizer:** Converts text data into a vector of token counts, the number of times a word appears in the document.
2. **TF-IDF (Term Frequency - Inverse Document Frequency) Vectorizer:** Converts text into a vector of overall document weightage of a word, weights the word counts by a measure of how often they appear in the document.

We evaluate our model performance measuring the *model accuracy*, *precision*, *recall*, and *f-score* values. We visualize classification results by printing confusion matrix in our code file. Confusion matrix is used for visualizing important predictive analytics such as recall, specificity, accuracy, and precision since it consists TP (true positives), FP (false positives), TN (true negatives), and FN (false negatives) to calculate mentioned analytics. We compare the accuracy of the model with the majority baseline accuracy. We also compare accuracy of the model in terms of used vectorizer and whether the data is shuffled before split up as training and test sets. We print the *posterior* and *prior probabilities* of both classification tasks performed by Naive Bayes classifier, and, try to find some examples from the test set in which the classifier was wrong by a large margin, which correct class had a very low probability, for the sentiment analysis task.

In Section 2, information about data set used in the model training and testing is provided. In Section 3, we explain experimental setup and evaluation methods that are used to assess model performance. In Section 4, we provide our results, and in Section 5, we discuss our finding and make conclusions on the results that are presented in Section 4. In Section 6, we answer more general questions in the assignment that are not directly answered in the previous sections.

2 Data Set

For the text classification task, we utilize a publicly available data set containing 6000 textual data collected from a popular social networking platform - "Twitter". These text samples have already been annotated and preprocessed for our task. With these samples, both binary and multi-class classification are performed. For binary classification, we analyse the sentiments (either positive or negative) of the samples. Moreover, we classify the samples into six classes (health, dvd, software, music, books and camera) for doing the multi-class classification. All the samples in the data set are randomly split into two sets namely training set and test set. 20% of the whole data set is kept for the test set. Table 1 reports the distribution of the samples into these

Table 1: Data set divided into different classes for both binary and multi-class classification

		No. of Samples		
Classification Type	Classes	Train	Test	Total Samples (per class)
Binary	Positive	2351	617	2968
	Negative	2449	583	3032
Multi-class	Health	788	198	986
	DVD	820	192	1012
	Software	789	205	994
	Music	808	219	1027
	Books	806	187	993
	Camera	789	199	988

two sets for different classes for both binary and multi-class classification settings. As we can see from the table, we ensure almost even distribution of samples for each classes and thus develop a balanced data set for our experiments.

Apart from the above-mentioned approach of using fixed training and test sets, we also perform 5-fold cross validation with our training set to observe if our model can generalize and does not get biased. To keep the fairness of the experiment, we do not use our test set, instead utilize the training set by splitting it into 5 sets (4 of them are used as training set and the other one used as the validation set in an iterative way).

3 Method/Approach

To run the experiment, we split our data set into a training set (80%) and test set (20%). We do not use the dev set since we do not have any specified parameters to tune in our model. Main purpose of not including the test set in model training is to get unbiased estimate of model performance from the test set. In the code, there is a shuffle option that shuffles the data set before splitting it. We run our experiment with both shuffled and unshuffled data. We use two type of feature vector (produced by Bag of Words and TF-IDF vectorizer packages from scikit-learn) for each classification tasks (binary and multi-class classification) for each of the runs and printed accuracy, precision, recall and f-score values of the model. We compare these results in Section 5. We use cross-validation to test trained model's ability to predict new data which is not used in estimation, to identify problems like selection bias, over-fitting, and to provide an insight on how the trained model will generalize on an independent data set. We calculate the prior probabilities for each classification tasks on each runs to see whether each class has a similar distribution on test set. We also calculate the posterior probabilities of each classes for each data item for both classification tasks. In this case, posterior probability is a probability of occurrence of a class C for a given data item X, and, prior probability is frequency of occurrence of the class C in given test set. Both prior and posterior probabilities are calculated as below in Equation 1 and Equation 2 respectively, where $P(C)$ is prior probability, $P(C|X)$ is posterior probability, $P(X|C)$ is likelihood and $P(X)$ is probability of data item X occurring in test set.

$$P(C) = \frac{\#ofitemsbelongtoclasscintestset}{\#ofitemsintestset} \quad (1)$$

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \quad (2)$$

We train a Naive Bayes model to perform binary and multi-class classification tasks. Naive Bayes relies on strong independence assumption between features, and assumes that the presence of a particular feature in a class is not related with the presence of any other feature. It calculates the posterior probabilities for each class for every item in the test set, and assigns each data item to a specific class c^* which has higher probability compared to other classes. Equation 3 represents the Naive Bayes formula.

Table 2: Comparison between the majority baseline and Naive Bayes approaches with TF-IDF and Bag-of-words input features separately for both binary and multi-class classification. Here, P, R and F1 denote the precision, recall and F1-score, respectively. Same test set used for the evaluation.

	Binary				Multi-class			
Model	P	R	F1	Accuracy (%)	P	R	F1	Accuracy (%)
Majority Baseline	-	-	-	51.4	-	-	-	18.3
Naive Bayes with TF-IDF	0.81	0.79	0.79	78.9	0.91	0.90	0.89	90.0
Naive Bayes with Bag-of-words	0.81	0.81	0.81	80.8	0.90	0.89	0.89	89.3

Table 3: Precision (P), recall (R), F1-score (F1) and prior probabilities (PP) calculated per class on our test set for binary and multi-class classification. Here, model trained with Naive Bayes algorithm with TF-IDF features. features.

Classification Type	Classes	P	R	F1	PP
Binary	Positive	0.91	0.65	0.76	0.51
	Negative	0.72	0.93	0.81	0.49
Multi-class	Health	0.97	0.76	0.86	0.17
	DVD	0.87	0.90	0.89	0.16
	Software	0.87	0.93	0.90	0.17
	Music	0.96	0.95	0.95	0.18
	Books	0.94	0.93	0.93	0.16
	Camera	0.81	0.93	0.87	0.17

$$c^* = argmax_{c \in C} \frac{P(X|c)P(c)}{P(X)} \quad (3)$$

4 Results

In this section, we are going to show the outcomes of our experiments. In Table 2, the precision, recall, F1-score and accuracy are presented calculated using both the majority baseline and Naive Bayes algorithm. With the Naive Bayes, two different input feature types (TF-IDF and bag-of-words) are also explored. Here, our majority baseline is the model which always predicts the most frequent class. As seen from the table, Naive Bayes outperforms the majority baseline by a large margin getting more than 78% accuracy for both types of classification whereas the latter only achieves 51.4% and 18.3% accuracy for binary and multi-class classification, respectively. For different input features, Naive-Bayes based model with bag-of-words features yields higher accuracy, getting 80.8%, than the TF-IDF which results in 78.9% accuracy. However, for the multi-class classification, TF-IDF performs slightly better, achieving 90% accuracy while bag-of-words approach gets 89.3% accuracy.

We also calculate the precision, recall and F1-score per class using Naive Bayes with TF-IDF features for both classification settings (See Table 3). While performing the binary classification, we see 91% of precision and 65% of recall for the "positive" class. In contrast, the negative class scores 72% of precision and 93% of recall. Negative class is predicted more correctly than the positive class by our model, getting a F1-score of 81% for the negative and 76% for the positive class.

For the multi-class classification, we obtain more than 85% of precision and recall values for DVD, software, music and books classes. The health class has the maximum difference between its precision and recall values and also the lowest

F1-score (86%). The top three classes getting the highest F1-score are music (95%), books (93%) and software (90%), respectively.

For both binary and multi-class text classification, we do not observe a specific class performing significantly worse than the other ones. This implies the fact that the data set we have used in our experiment is well-balanced. The almost similar prior probabilities for each classes within a particular classification type strengthen our above-mentioned findings.

We also perform 5-fold cross validation on the training set and get 77.46% accuracy which is slightly lesser than what we find on using the fixed test set (78.9% accuracy) for the binary classification using TF-IDF features. For the multi-class classification, using cross-validation we obtain 88.71% accuracy and without cross-validation, it is 90.0% accuracy. We also shuffle the data set and then train with Naive Bayes. By adopting this approach, we get 78.08% and 89.0% accuracy for binary and multi-class classification, respectively.

5 Discussion/Conclusion

In this assignment, we experiment with Naive-Bayes algorithm with TF-IDF and bag-of-words features using an open-sourced and well-balanced data set containing 6000 annotated textual data. From our experiments, we can observe that Naive Bayes performs exceedingly well while doing the text classification tasks although being a simple probabilistic algorithm (See Table 2). We exploit both TF-IDF and bag-of-words features and find that TF-IDF produces better accuracy for the multi-class classification and bag-of-words for the binary classification. However, using only such a small-sized data set, we cannot come to an conclusion on which feature vectors are more suitable for the Naive Bayes to yield better performance.

We also analyze the precision, recall, F1-score and prior probabilities per class for binary and multi-class classification to see whether our model tends to get biased towards a particular class. We observe a noticeable differences in the precision and recall values per class for the binary classification and to a lesser extent, for the multi-class classification. For example, there are comparatively higher number of false negative cases and lesser false positive cases for the "positive" class as a result of having a high precision and low recall value. (See Table 3). If we want to increase the recall value for this class, our model needs to predict more positive cases, however, this might eventually increase the false positive cases. As a result, the precision for the "positive" class will start to decrease if we increase the recall. The same precision-recall trade-off can be seen in case of the "negative" class.

Apart from these metrics, we also compute F1-score per class and find out that all the classes in our experiments obtain reasonably good F1-score due to the balanced data set we use for our experiment. However, if we look closely, we can observe that negative class is better predicted than the positive class by our model for the binary classification. In case of multi-class classification, we find the music, books and dvd among the most well-predicted classes. More importantly, these well-predicted classes can be found in comparatively higher number than the rest of the classes in our data set which leads to the findings that to increase the accuracy of a particular class, we have to include enough examples of it in the data set.

Our study with/without cross-validation yields almost similar accuracy and thus ensures the credibility of our findings and makes sure that our model is not biased. Also, when we shuffle the data set before splitting, we see slightly lower accuracy which is not significant for consideration.

6 Answers to additional questions

Accuracy is used when the TP (true positives) and TN (true negatives) are more important. F1-score is used when the FN (false negatives) and FP (false positives) are more crucial. It is a good approach to use accuracy if the class distribution is similar, in other words if data set is balanced. On the other hand, F1-score is a better metric if there are imbalanced classes in the data set. In our case, since our data set balanced, meaning that classes have similar distribution across the data set, accuracy is a better metric to evaluate model performance.

F1-score is a function of Precision and Recall. Per

class F1-score is just the harmonic mean of Precision and Recall of that class. It is calculated as follows:

$$F1-score = \frac{2 \times (precision \times recall)}{(precision + recall)} \quad (4)$$

Now if we want to convert these numbers into a single number, we can use several methods like macro, micro, weighted average F1-score. Macro Average is an arithmetic mean of per class F1-Score. Every instance considered as equal weight.

$$Macro-F1 = \frac{(Class_{1f}1 + Class_{1f}2 + Class_{1f}3)}{3} \quad (5)$$

Whereas, Micro Average considers total TP, TN, FP and FN. Without considering the prediction for each label. Here, every classes are treated as same to calculate the overall performance of the classifier.

$$Micro-P = \frac{(TP_1 + \ldots + TP_n)}{(TP_1 + \ldots + TP_n + FP_1 + \ldots + FP_n)} \quad (6)$$

After calculating posterior probabilities for binary classification (sentiment analysis), we realize that some of the classes are classified wrong by a large margin by the classifier. Texts from the file reviews.txt with indices 4804, 4816, 4838, 4839, 4840 are actually negative reviews, but our model predicts them as positive. After reading each text in these indices, we see that these texts have lots of words that can be considered as positives. A sentence with the index 4840 from reviews.txt is presented below:

I don't loathe this work, it is enjoyable in many parts for its color characterizations and depth of detail. the narrator of the audio book has a wonderful voice and good pacing. but this is highly flawed in its editorial approach to historical biography.

Though it contains positive words such as *wonderful voice*, *good pacing*, *enjoyable*, it is actually a negative review from its meaning.

On the other hand, we also encounter some results classified as negative which are actually positive reviews. A text sample with index 4833 from reviews.txt is as follows:

this scale is a great buy, for only $10 how can you go wrong!

It describes the price as cheap and decent. But, our classifier classifies it as a negative review like it looks like buying such scale would be worthless.

We conclude that our classifier is confused by occurrences of some specific words (like enjoyable, good pacing, don't, wrong etc.) and make wrong classification by large margin. Hence, our model is not good at catching the relationship between words occurring in the same text since Naive Bayes classifier assumes all the features are independent, which is not correct in many real word scenarios.