Project 1: Scam detection with Naive Bayes

Kevin Yu
Student ID: 1462539
The University of Melbourne

April 14, 2025

1 Supervised model training

The prior probability of **non-malicious class was** 0.7995 (rounded to 4 d.p.), while the **scam class was** 0.2005. This indicates that the dataset is imbalanced, with a higher proportion of non-malicious instances. The distribution aligns well with real-world data, where non-malicious messages significantly outnumber spam.

Table 1 shows the top 10 most "probable" words for each class, based on the conditional probability $P(w \mid c)$. In contract, Table 2 shows the top 10 most "predictive" words for each class, calculated by the ratio, $P(w \mid c)/P(w \mid \neg c)$. Notably, the words in each table differ; although both measure the association of words with a particular class, they serve a slightly different purpose.

For example, most probable words include words such as 'call' ($p = 0.0274$), 'customer' ($p = 0.0092$) and 'reply' ($p = 0.0080$). These words appear frequently in scam messages, which makes them probable; however, intuitively, we expect legitimate messages to frequently contain such words, especially those from customer service. In contrast, the most predictive words include terms such as 'prize' ($p = 88.80$) and 'claim' ($p = 41.21$). These words can appear in legitimate context, however, they are far more likely to be used in scam messages. (How often do you randomly win a prize?)

For non-malicious messages, the most predictive words include slang such as 'lor' ($p = 32.16$), 'wat' ($p = 19.53$) and 'lol' ($p = 17.80$). These casual words are more likely to be used in non-malicious messages, and are not commonly found in scam messages. This serves as a good example of how the model can learn to distinguish between the two classes based on word usage.

Overall, the result is reasonable and it seems feasible to distinguish between scam and non-malicious messages using a multinomial naïve Bayes model. The differences in both frequent and predictive vocabulary across classes suggest the model can effectively learn class-specific language patterns, despite the simplicity of its assumptions.

Scam Class		**Non-malicious Class**	
Word	**Probability**	**Word**	**Probability**
call	0.0274	go	0.0161
free	0.0137	get	0.0143
claim	0.0100	gt	0.0085
customer	0.0092	lt	0.0084
txt	0.0090	call	0.0083
ur	0.0085	ok	0.0078
text	0.0082	come	0.0075
stop	0.0082	ur	0.0075
reply	0.0080	know	0.0075
mobile	0.0078	good	0.0071

Table 1: Top 10 most probable words for each class

Scam Class		Non-malicious Class	
Word	$P(w \mid \textbf{scam})/P(w \mid \textbf{ham})$	**Word**	$P(w \mid \textbf{ham})/P(w \mid \textbf{scam})$
prize	88.80	gt	60.30
tone	57.46	lt	59.73
select	41.79	lor	32.16
claim	41.21	hope	27.57
50	34.82	ok	27.57
paytm	33.08	da	22.40
code	31.34	let	20.10
award	28.73	wat	19.53
won	27.86	oh	18.38
18	26.12	lol	17.80

Table 2: Top 10 most predictive words for each classe based on likelihood ratios

2 Methodology

Describe the methods and techniques used in your work. Include equations, algorithms, or diagrams if necessary.

2.1 Data Preprocessing

Explain how the data was prepared for analysis, including any cleaning, transformation, or feature extraction steps.

2.2 Model Description

Provide details about the model or approach used, such as Naive Bayes, decision trees, etc.

3 Results

Present the results of your analysis or experiments. Use tables, figures, and graphs to support your findings.

Metric	Value 1	Value 2
Accuracy	0.95	0.90
Precision	0.92	0.88
Recall	0.93	0.89

Table 3: Example table caption.

4 Discussion

Interpret the results and discuss their implications. Highlight any limitations or challenges encountered.

5 Conclusion

Summarize the key findings and contributions of the report. Suggest future work or improvements.

References

1. Author Name, *Title of the Paper/Book*, Publisher, Year.

2. Author Name, "Title of the Article," *Journal Name*, vol. X, no. Y, pp. Z, Year.

Since the dataset is already preprocessed, I'm directly supplying vocabulary=vocabulary to CountVectorizer without calling fit(), to avoid any unintended token filtering (e.g., removing tokens like 'hi') that may occur with the default tokenizer during fitting.