Final Report

Email Spam Detection Pipeline

Dominic Mancini

2025-05-01

0.1 Introduction

0.2 Executive Summary:

This project aims to build an accurate and interpretable spam email classifier using statistical modeling, machine learning, and natural language processing techniques in Python. Using the Nazario spam dataset, the system was trained on labeled spam and ham messages, extracting both text-based features and metadata. The final models demonstrated strong performance (AUC 0.99–1.00), offering a viable tool for spam filtering in practical applications. This solution could save time and reduce risks for organizations by automatically flagging malicious or irrelevant messages.

0.3 Problem Statement:

Spam and phishing emails are a persistent problem for individuals, businesses, and organizations alike. Generic spam filters are widely available, but they often fail to account for the specific needs and context of individual organizations. This necessitates the exploration and development of bespoke spam detection solutions through lightweight, modular pipelines that can be tailored to the unique requirements of organizations, learn from and adapt to their real-world email flow, and integrate into existing infrastructure.

0.4 Methodology

Modeling Approaches and Rationale

To build a robust spam classification system, this project implemented and compared several supervised machine learning models using both textual and structural features from the

Nazario spam email dataset[1]. Each model was chosen based on its suitability for text classification tasks, ease of interpretability, and performance tradeoffs while building off the techniques used in past research[2].

The Dataset

While the Enron Email Dataset[3] was effective for the preliminary analysis, it also had a number of limitations. Originally collected in 2006, the dataset comprised of emails collected from a subset of employees at the Enron corporation[3]. However, after roughly 2 decades, the dataset now outdated and may not accurately reflect the current landscape of spam and phishing. As the technology and tactics used by spammers have evolved, so too have the characteristics of spam emails such as AI-generated content or social engineering techniques.

The Nazario dataset[4] was instead chosen for this project. It is a more modern dataset that compiles emails from a variety of sources, including spam traps and honeypots. As outlined by[5], this version of the dataset is designed to be a comprehensive and up-to-date resource for spam detection research. It includes a wide range of spam types, including phishing, malware, and unsolicited marketing emails. The dataset is also regularly updated to reflect the latest trends and tactics used by spammers.

The original dataset contained 6 columns: - sender - receiver - date - subject line - email body - URLs in email and finally, the spam or ham classification.

Data Loading and Preprocessing

Using a helper function in the `dataset` module, the data was loaded, NA values were handled and removed, and a new 'text' column was created that the emails body and subject lines together.

```
df: pd.DataFrame = dataset.load_and_clean_data(config.NAZARIO_DATASET)
# display(Latex(df.head().to_latex(index=False)))
display(df.info(memory_usage=False, show_counts=False))
```

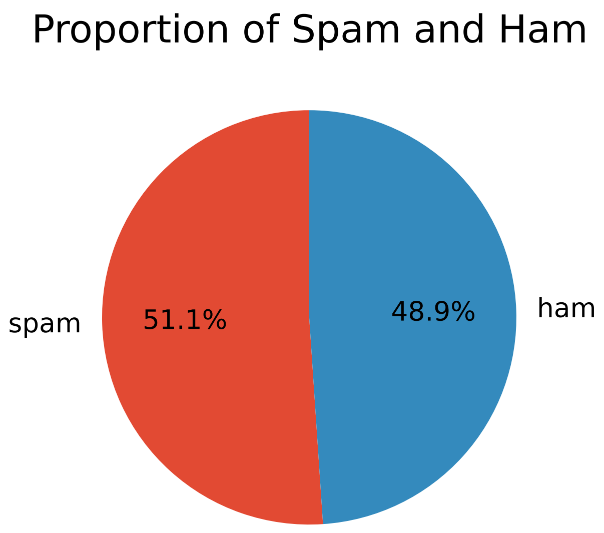
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3065 entries, 0 to 3064
Data columns (total 8 columns):
 #   Column    Dtype
---  ------    -----
 0   sender    object
 1   receiver  object
 2   date      object
 3   subject   object
 4   body      object
 5   label     int64
 6   urls      object
 7   text      object
dtypes: int64(1), object(7)
```

```
None
```

```python
labmap = {0: "ham", 1: "spam"}
df["email_type"] = df["label"].map(labmap)  # type: ignore
counts = df["email_type"].value_counts()
fig, ax = plt.subplots()
ax.pie(counts, labels=list(counts.index), autopct='%1.1f%%', startangle=90.0)
ax.set_title("Proportion of Spam and Ham")
```

```
Text(0.5, 1.0, 'Proportion of Spam and Ham')
```

Email Class Makeup



The class distribution is roughly balanced, with ~51% spam and ~49% ham emails. This is important, as it reduces the risk of majority-skewed bias in model training. As such, this allows us to continue with standard evaluation metrics without oversampling or undersampling techniques.

Feature Engineering

A key goal in this pipeline involved engineering features that reflect both content and structural patterns of spam emails. Structural features refer to meta characteristics of the emails such as the number, and presence of, HTML content in an email and other elements. However, extracting and analyzing these features revealed a lack of consistency and signal strength across the data. As a result, focus was placed on a smaller set of high-signal features that demonstrated high predictive value and strength. However, the `dataset` module contains functions to easily extract and analyze these features if needed.

Among these engineered features are: the number of URLs in the email, the total character length of the email, and whether the subject line contains a reply or forward indicator (eg. 're:', 'fwd:').

```
all_feats = pd.merge(df, textual_features.extract_textual_features(
    df), left_index=True, right_index=True)

all_feats.columns
```

```
2025-05-12 02:51:40,648 - INFO - Extracting textual features...
```

```
Index(['sender', 'receiver', 'date', 'subject', 'body', 'label', 'urls',
       'text', 'email_type', 'email_length', 'num_words', 'num_exclamations',
       'num_uppercase_words', 'contains_spammy_phrases', 'num_urls',
       'subject_has_reply', 'has_suspicious_tld', 'num_tags', 'num_links',
       'has_script', 'html_ratio'],
      dtype='object')
```

Text Cleaning and Preprocessing

For the content-based component, emails were cleaned and processed before vectorization. This included removing HTML tags, converting text to lowercase, removing non-alphabetic characters, and stemming words.

Word stemming was performed using the `PorterStemmer` from the Natural Language Tool Kit (NLTK) library, which reduced words to their root form. This step is crucial for reducing dimensionality and improving model performance by ensuring that similar words are treated as the same feature. The `features` module provides a set of functions to perform these tasks efficiently.

```
all_feats["clean_text"] =
 ↪  all_feats["text"].apply(features.clean_and_stem_text)


print("Raw Email:\n")
display(textwrap.fill(all_feats.at[50, "body"], width=60))
print("After cleaning an word stemming:\n")
display(textwrap.fill(all_feats.at[50, "clean_text"], width=60))
```

```
Raw Email:

'Please click on the attached icon or the Web address below\nto go to the most recent OASIS 

After cleaning an word stemming:

'oasi post energi index gener imbal servic pleas click attach\nicon web address go recent oa
```

Feature Visualizations

Below are comparison of several numeric features aggregated by spam and ham label.

```
plots.aggregate_by_label(all_feats, "email_length", "mean", "Email Length");
plots.aggregate_by_label(all_feats, "num_words", "mean", "Word Number");
plots.aggregate_by_label(all_feats, "subject_has_reply", "sum", "Has Reply");
```

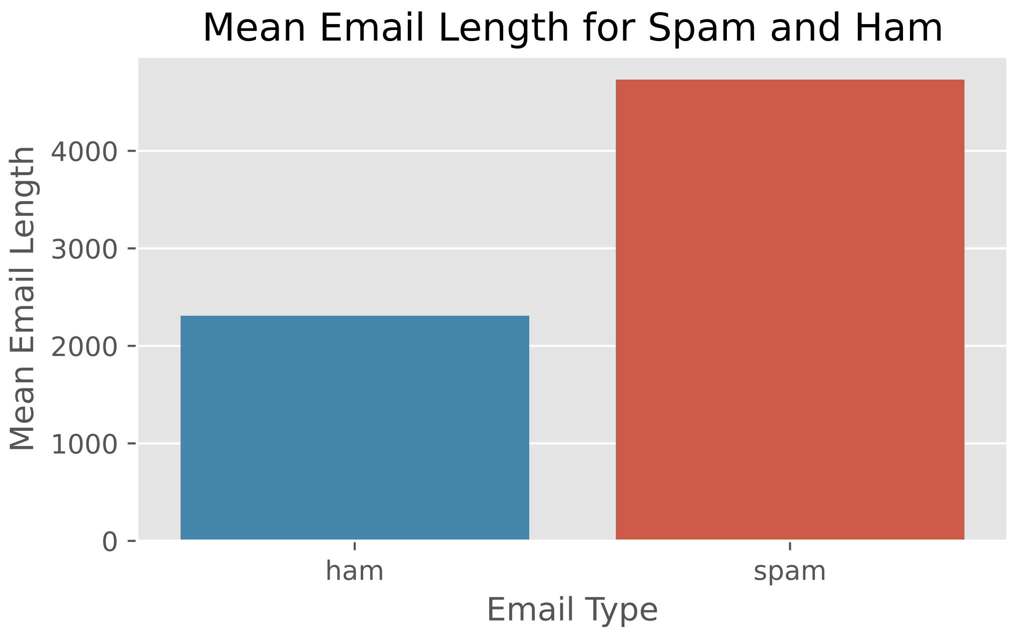
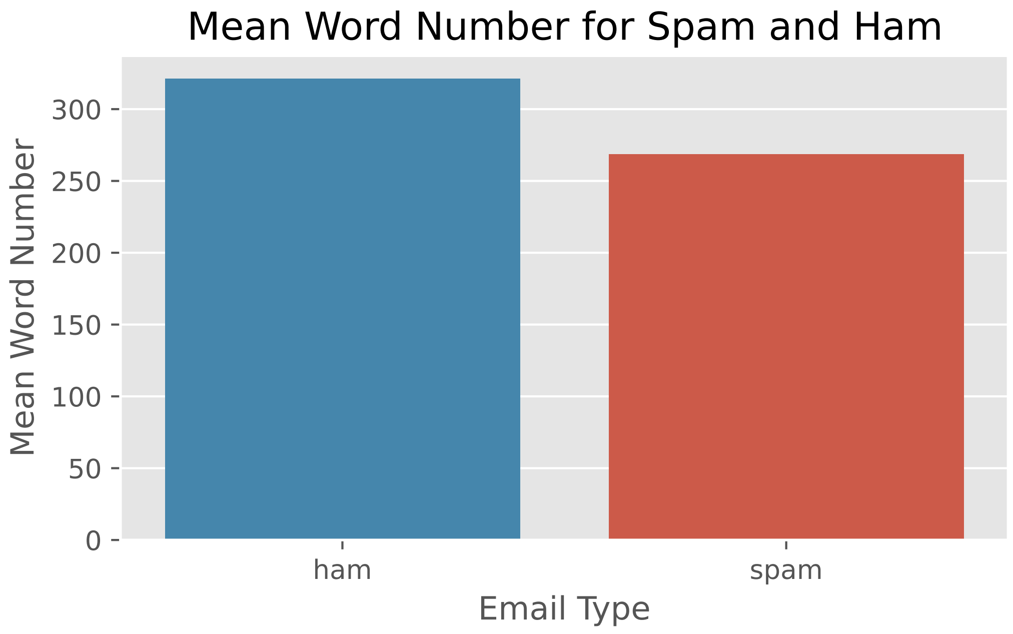
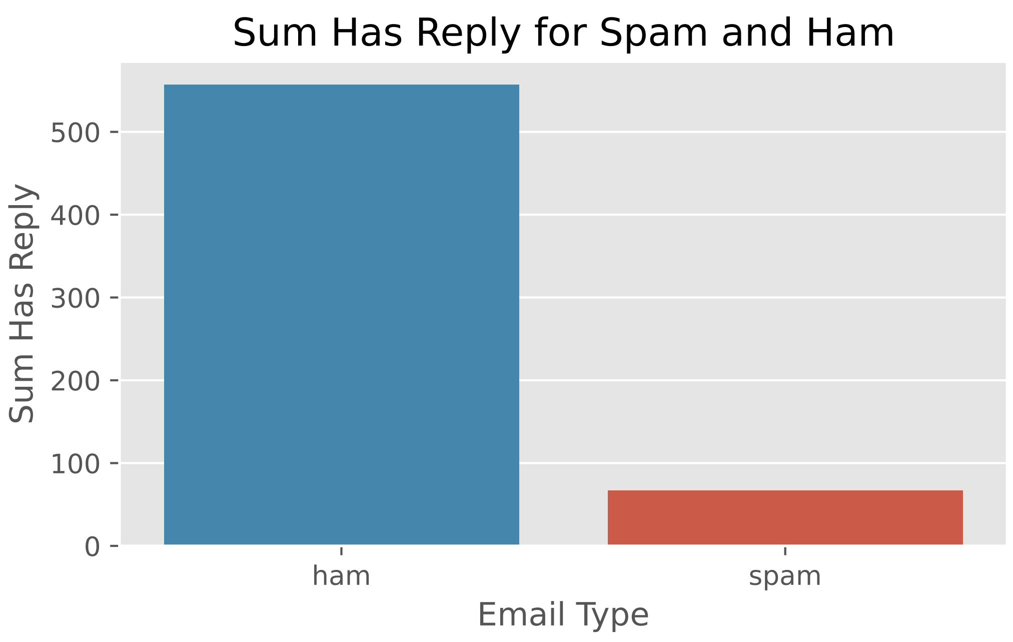


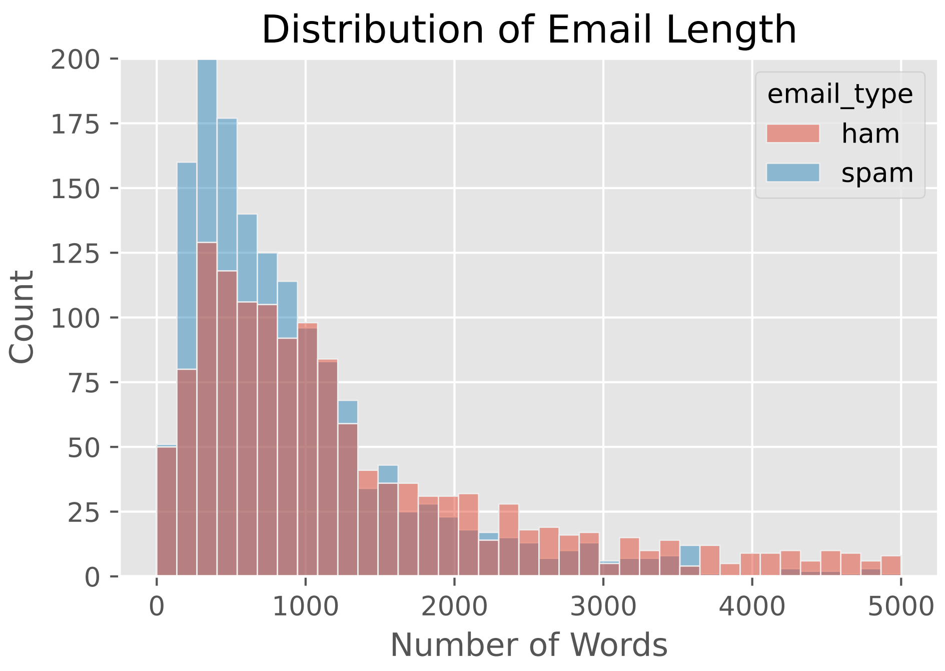
Figure 1: Comparing Numeric Features





```
Text(0.5, 1.0, 'Distribution of Email Length')
```

Distribution of Email Length by Spam/Ham



To illustrate the most prominent words and important features in the dataset, I created word clouds for both spam and ham emails. The word clouds visually represent the most frequently occurring words in each category, providing insights into the language and topics commonly associated with spam and legitimate emails.

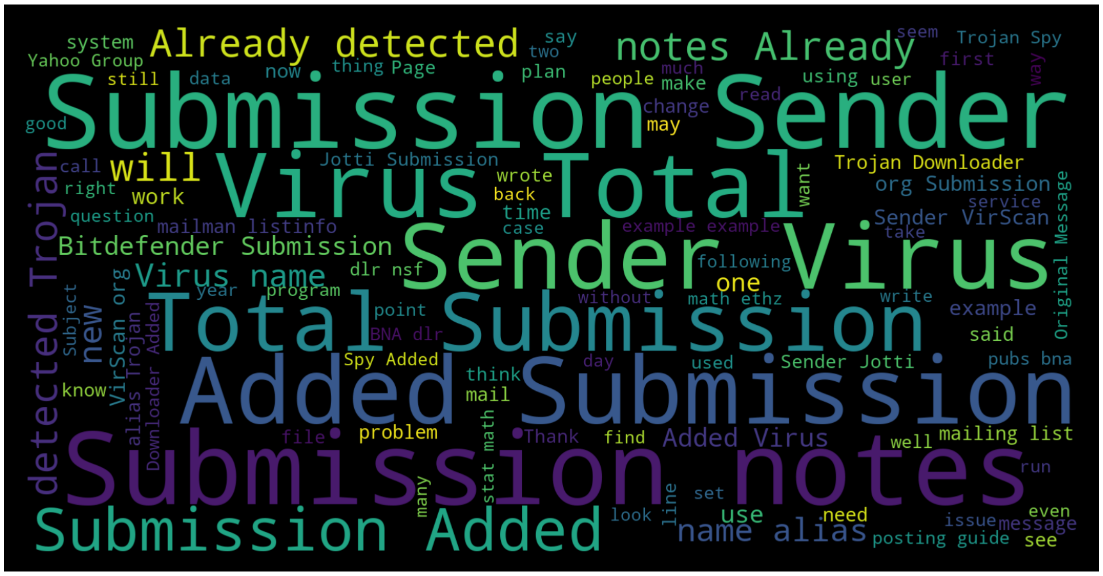


Figure 2: Most Prominent Words in Non-Spam Emails

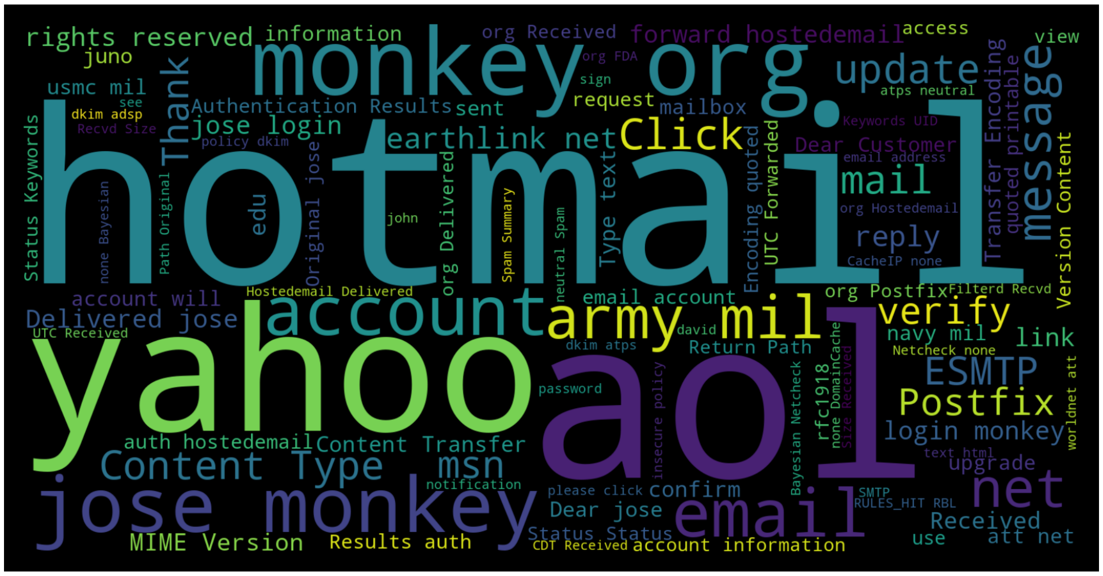


Figure 3: Most Prominent Words in Non-Spam Emails

The word clouds revealed some clear patterns. For example, spam emails often contained the names of email service providers, various technical terminology, and action-driven language such as “Click”, “verify”, “update”. This is consistent with phishing and email spam attempts that aim to create urgency or mimic legitimate servers.

Interestingly, some of the most frequent words in non-spam emails related to computer viruses, malware, and other security threats. This is likely due to the fact that many legitimate emails contain discussions about security measures, updates, and alerts related to viruses and malware. This suggests that legitimate communications in this dataset may often involve professional discussions about IT security.

TF-IDF

TF-IDF vectorization was applied to the clean_text column to transform the processed email text into a numerical matrix of token importance. TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to capture the importance of each word in the context of the entire dataset, allowing the model to focus on the most relevant features for classification[2].

```
tfidf = TfidfVectorizer(max_features=1000)
scaler = StandardScaler(with_mean=False)
```

The final features chosen for the cleaned text (combined subject and body), the number of URLs, email length, and whether the subject has a reply.

In order to use the TF-IDF matrix in conjunction with the additional features, the numeric features are scaled using the `StandardScaler`. This step is important for ensuring that all features are on a similar scale, which can improve model performance. The `max_features` parameter was set to 1000 to limit the number of features and reduce dimensionality.

```
data = all_feats[["label", "clean_text", "num_urls",
                  "email_length", "subject_has_reply"]]
X = tfidf.fit_transform(data["clean_text"])
y = data["label"].values #type: ignore
X_struct = data[["num_urls", "email_length", "subject_has_reply"]]
X_struct_scaled = scaler.fit_transform(X_struct) # pyright: ignore

# combining the features
X_combined = np.hstack((X.toarray(), X_struct_scaled)) # pyright: ignore

X_train, X_test, y_train, y_test = train_test_split(
    X_combined, y, test_size=0.2, random_state=42
)
```

Now, the data is ready for model training. The TF-IDF matrix and the additional features were combined into a single feature set, which was then split into training and testing sets using an 80/20 split.

0.5 Model and Pipeline

The pipeline for this project was designed to be modular and extensible, allowing for easy experimentation with different algorithms and features. This approach enabled consistent training, evaluation, and comparison across classification models.

Pipeline

To evaluate the predictive value of the extracted features, three supervised machine learning models were implemented; Naive Bayes, Logistic Regression, and a Linear Support Vector Machine (SVM).

Naive Bayes is a Probabilistic graphical model (PGM) that that assumes conditional independence between features, utilizing prior probabilities of a class label and the likelihood of each feature given that class label[6(p. 11)] The Multinomial Naive Bayes classifier was chosen for this project as it is well-suited for discrete data and performed relatively well in the preliminary analysis.

Logistic Regression is a linear model that is well-suited for binary classification tasks, balancing performance and interpretability. Logistic Regression predicts probabilities over classes to discern the features most useful to discriminate examples[6(p. 12)].

Linear Support Vector Machine (SVM) is a robust, margin-based classifier that is effective in high-dimensional spaces, such as those created by TF-IDF vectors. SVMs can effectively separate classes with a hyperplane that maximizes the margin between them[7(p. 65709)].

All models were implemented using the `scikit-learn` library and trained on the same set of preprocessed features.

Pipeline Implementation

`Modeler` is a custom class designed to streamline the training and evaluation of models. The class acts as a high-level wrapper around `scikit-learn`'s estimator API, providing a simple and convenient interface for performing common machine learning tasks and managing the model lifecycle.

Modeler is initialized with a `scikit-learn` estimator, which can be any model, after which various methods can be called that allow for: - Fitting the model to training data - Evaluating the model on testing data - Generating performance metrics and visualizations - Saving and loading models to/from disk - Hyperparameter tuning using GridSearchCV

The class is designed to be flexible and extensible, mirroring `scikit-learn`'s Object-Oriented API. This allows for easy experimentation with different models and hyperparametrers, facilitating the development of classification pipelines.

```
nb_model = Modeler(MultinomialNB(), model_name="Naive Bayes",
 ↪  random_state=42)
lr_model = Modeler(LogisticRegression(random_state=42),
                   model_name="Logistic Regression",
                   random_state=42)
svm_model = Modeler(SVC(kernel="linear", probability=True, random_state=42),
                    model_name="SVM", random_state=42)
```

```
nb_model.train(X_train, y_train)
nb_model.evaluate(X_test, y_test)
nb_model.confusion_matrix(y_test);
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       313
           1       0.96      0.97      0.97       300

    accuracy                           0.97       613
   macro avg       0.97      0.97      0.97       613
weighted avg       0.97      0.97      0.97       613

[[300  13]
 [  8 292]]
```

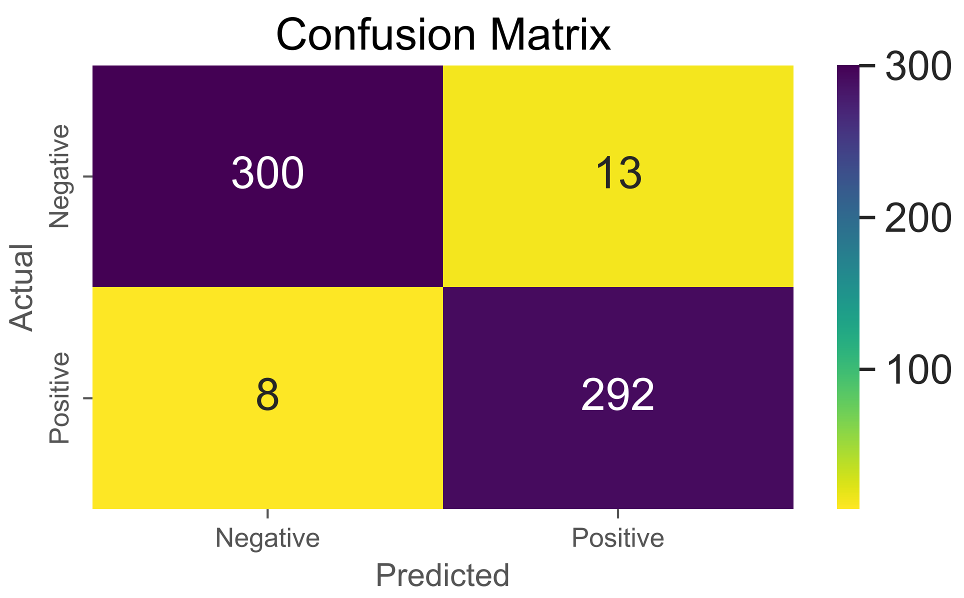


Figure 4: Naive Bayes Confusion Matrix

```
lr_model.train(X_train, y_train)
lr_model.evaluate(X_test, y_test)
lr_model.confusion_matrix(y_test);
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       313
           1       0.98      0.99      0.99       300

    accuracy                           0.99       613
   macro avg       0.99      0.99      0.99       613
weighted avg       0.99      0.99      0.99       613

[[308   5]
 [  2 298]]
```

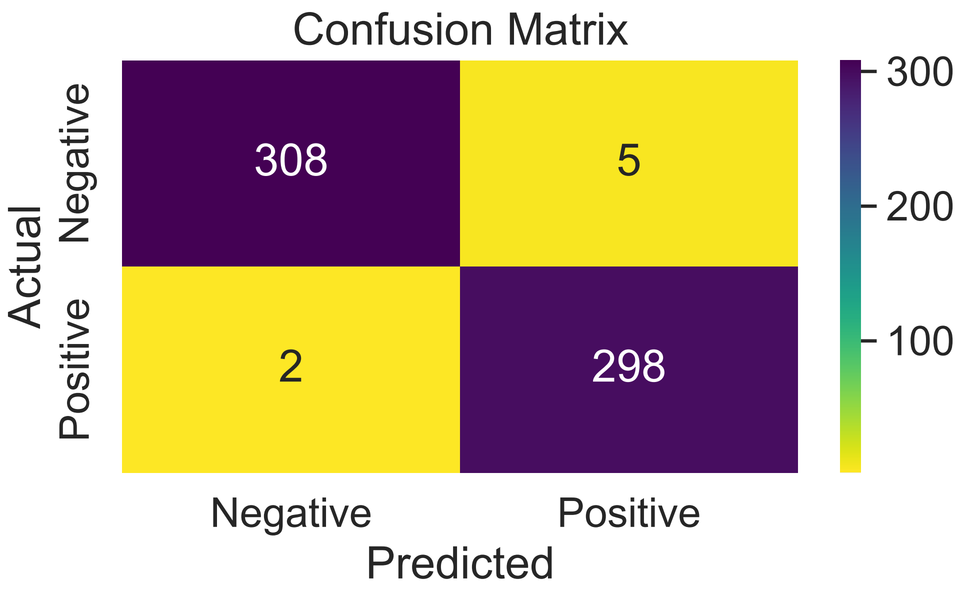


Figure 5: Logistic Regression Confusion Matrix

```
svm_model.train(X_train, y_train)
svm_model.evaluate(X_test, y_test)
svm_model.confusion_matrix(y_test);
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       313
           1       0.98      0.99      0.99       300

    accuracy                           0.99       613
   macro avg       0.99      0.99      0.99       613
weighted avg       0.99      0.99      0.99       613

[[308   5]
 [  3 297]]
```

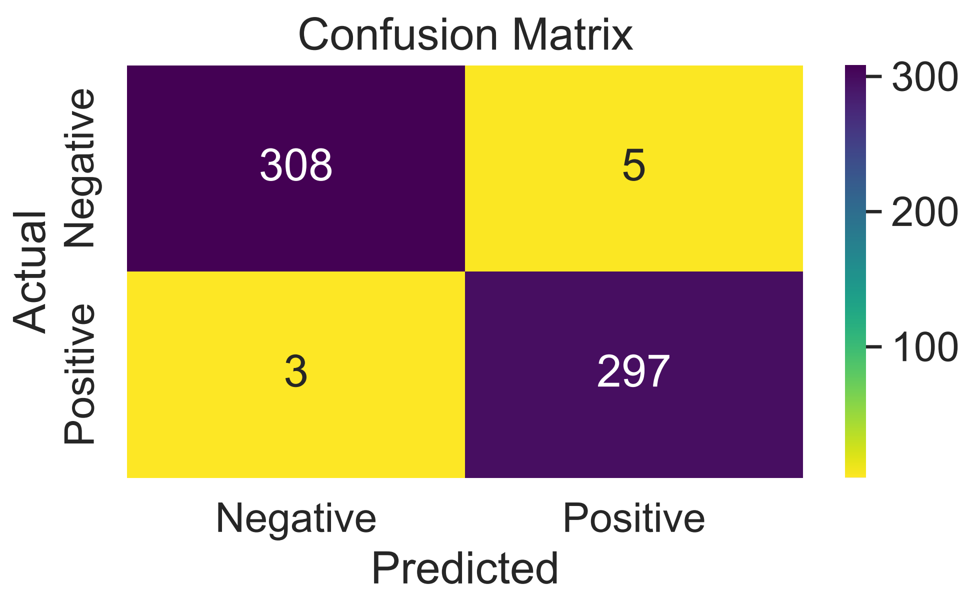


Figure 6: SVM Confusion Matrix

0.6 Evaluation and Results

Metrics

To assess the performance of the models, several evaluation metrics were utilized: - Accuracy: How many emails were correctly classified as either spam or ham? - Precision: Out of all the emails predicted as spam, how many were actually spam? - Recall: Out of all the actual spam emails, how many were correctly identified as spam? - F1 Score: How well does the model balance precision and recall?

Model Comparison:

Metrics are aggregated into a summary table for comparison.

```
metrics = {}

for m in [nb_model, lr_model, svm_model]:
    metrics[m.model_name] = m.metrics

metric_df = pd.DataFrame.from_dict(metrics, orient="index").round(3)
print("\nComparison of Model Metrics:")
```

```
metric_df
```

```
Comparison of Model Metrics:
```

Table 1: Model Metrics Comparison

	Accuracy	Precision	Recall	F1-Score
Naive Bayes	0.966	0.957	0.973	0.965
Logistic Regression	0.989	0.983	0.993	0.988
SVM	0.987	0.983	0.990	0.987

```
Best Model Performance Metrics:
```

	model	value
accuracy	Logistic Regression	0.989
precision	Logistic Regression	0.983
recall	Logistic Regression	0.993
f1-score	Logistic Regression	0.988

All three models performed well, but Logistic Regression slightly outperformed the others across nearly all the metrics. Precision was the weakest metric for all models, which is expected in spam detection tasks where false positives can be costly. Again, the Logistic Regression model achieved the highest recall, followed closely by the SVM, indicating its ability to identify spam emails effectively.

Results and Visualizations

```
mods = [nb_model, lr_model, svm_model]
fig = plt.figure(figsize=(10, 10), dpi=80)
ax = plt.gca()
for m in mods:
    PrecisionRecallDisplay.from_estimator(
        m.model, X_test, y_test, name=m.model_name, ax=ax)
plt.title("Precision-Recall Curve")
plt.show()
```

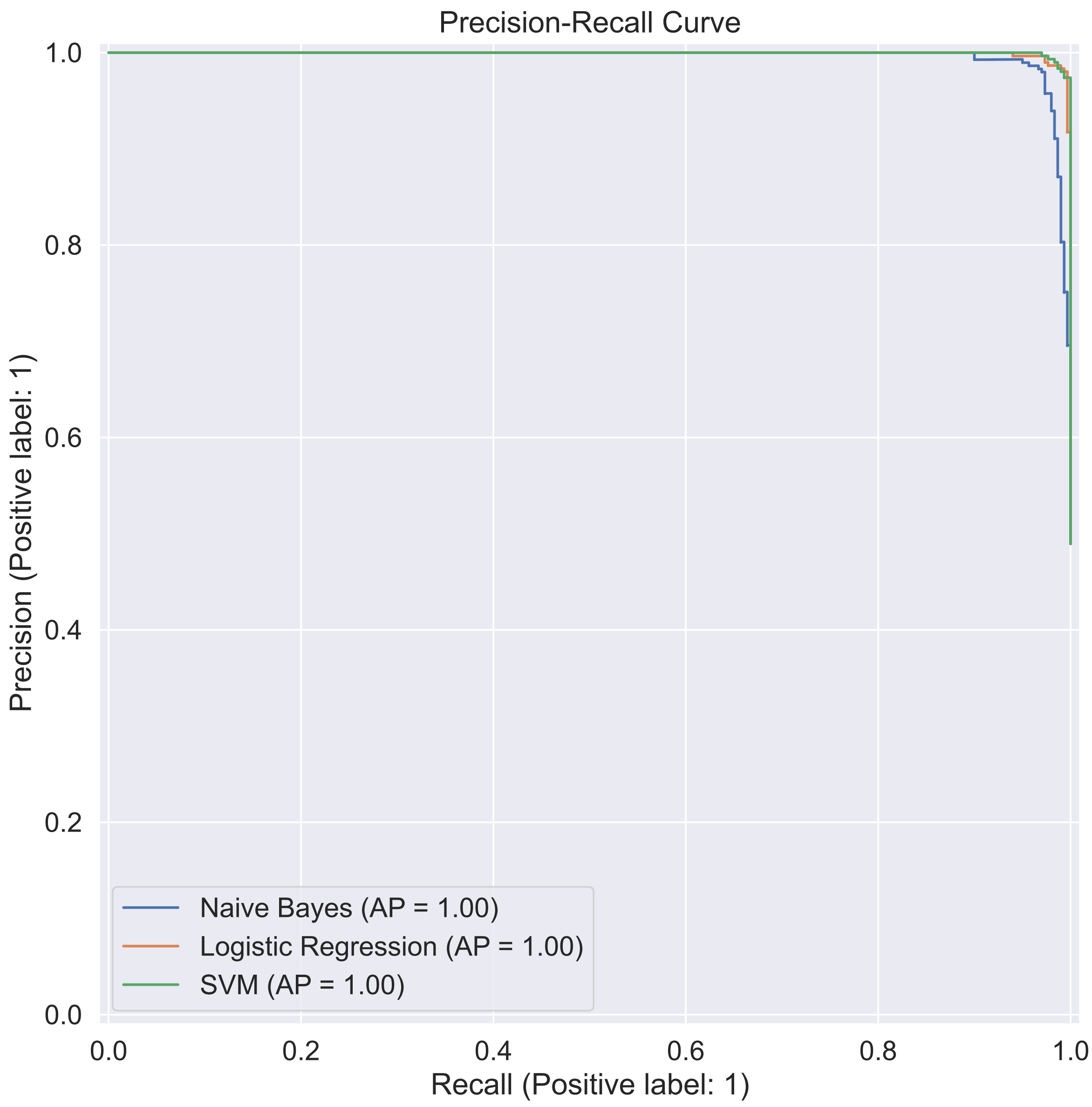


Figure 7: Precision-Recall Curve

Next, we can visualize the ROC curves for each model. The ROC curve is a graphical representation of the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) at various threshold settings. The area under the ROC curve (AUC) provides a single measure of overall model performance, with higher values indicating better discrimination between classes.

```
# plots.plot_roc_curves(models, X_test, y_test)
fig = plt.figure(figsize=(10, 10), dpi=80)
ax = plt.gca()
for m in mods:
    RocCurveDisplay.from_estimator(
        m.model, X_test, y_test, name=m.model_name, ax=ax)
plt.show()
```

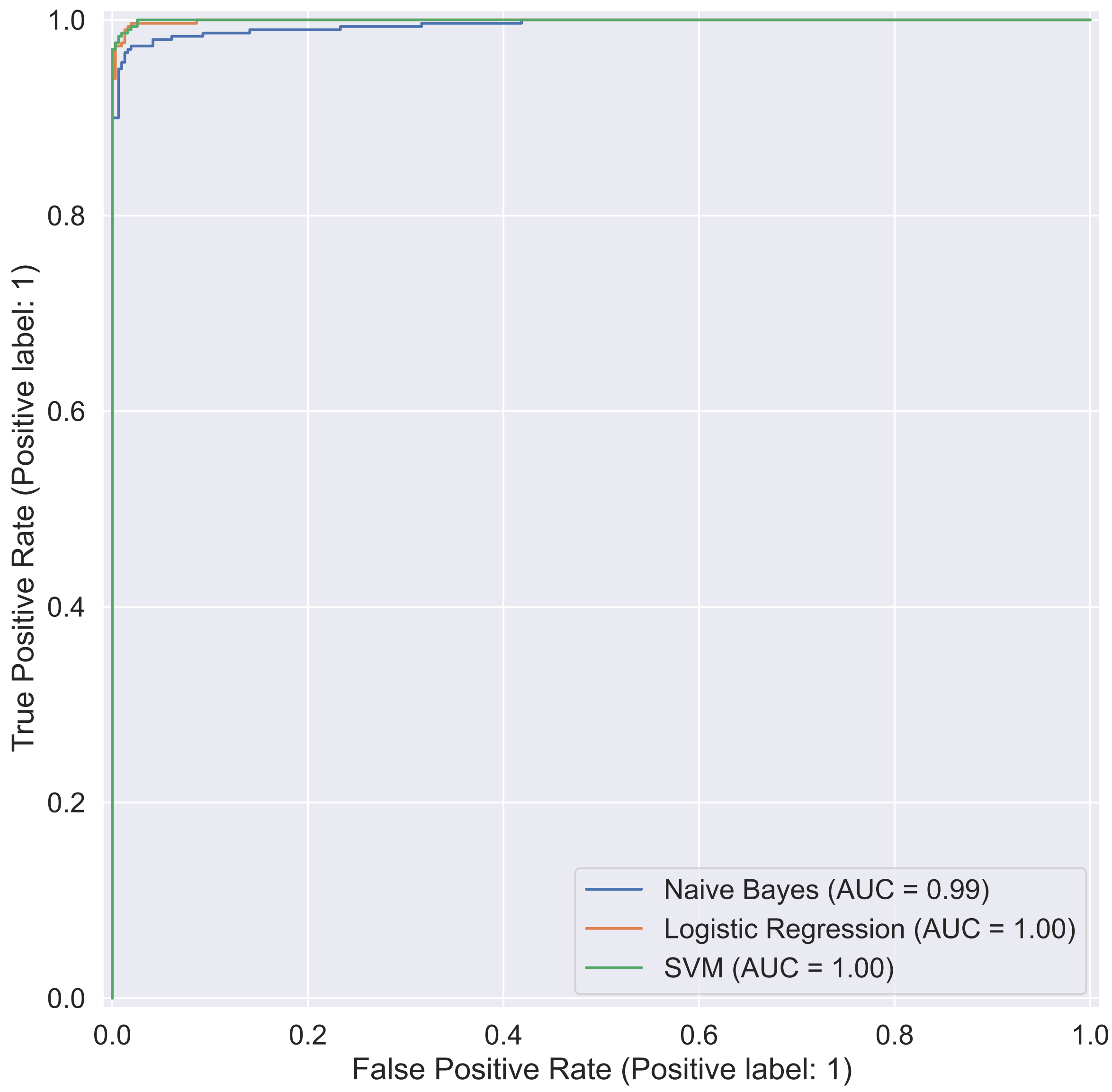


Figure 8: ROC Curve

In both cases, Logistic Regression and SVM closely overlay and hug the corner, indicating strong class discrimination.

0.7 Business And Social Value

Email spam detection is a critical component of modern cybersecurity and information has greater implications beyond just spam filtering. By accurately identifying and filtering out spam emails, organizations can reduce the risk of phishing attacks, malware infections, and other security threats. While generic spam filters are widely available and utilized by large email providers, these systems do not account for the specific needs and context of individual organizations. This project demonstrates the feasibility of building bespoke spam detection systems tailored to the individual needs of companies and organizations.

Using such a pipeline, small and medium-sized businesses, non-profits, or other organizations could build custom spam filters that are tailored to, and learn from, their real-world email flow. This would enable models to adapt over time, recognizing patterns and trends unique to the organization's communication style, common topics, and industry-specific language. The modular, lightweight nature of the pipeline allows teams to incorporate custom features, adjust model thresholds and parameters, or retrain models as needed. Integrating this kind of model into an existing email infrastructure would be possible through automation scripts, internal APIs and other methods.

Overall, this approach empowers organizations to build smarter, more flexible, defense systems that align with their specific needs and risk profiles, ultimately enhancing their security posture and reducing the risk of data breaches.

0.8 Ethical Considerations and Limitations

Machine Learning models, including those used for spam detection, can inadvertently inherit bias from their training data. This can lead to unfair treatment of certain groups or individuals, particularly if the training data is not representative of the broader population. In this project, the dataset was relatively balanced between spam and ham emails, but it is important to consider the potential for bias in the data collection process.

The models were trained on the Nazario dataset, however the dataset is not exhaustive and may not cover all possible spam. Furthermore, the dataset was curated from multiple time periods and sources which has the potential to introduce human biases[4]. As such, the models may not generalize well to new or unseen data.

References

1. Arifa Islam, C. (2023). *Phishing email curated datasets*. Zenodo. https://doi.org/10.5281/zenodo.8339691
2. Jáñez-Martino, F., Alaiz-Rodríguez, R., González-Castro, V., Fidalgo, E., & Alegre, E. (2023). A review of spam email detection: Analysis of spammer strategies and the dataset shift problem. *Artificial Intelligence Review*, *56*(2), 1145–1173. https://doi.org/10.1007/s10462-022-10195-4
3. Enron Corp., & Cohen, W. W. (2015). *Enron email dataset* [Compressed Data]. https://www.loc.gov/item/2018487913/
4. Champa, A. I., Rabbi, M. F., & Zibran, M. F. (2024). Curated datasets and feature analysis for phishing email detection with machine learning. *3rd IEEE International Conference on Computing and Machine Intelligence (ICMI)*, 1–7.
5. Champa, A., Rabbi, Md. F., & Zibran, M. (2024). *Why Phishing Emails Escape Detection: A Closer Look at the Failure Points*. 1–6. https://doi.org/10.1109/ISDFS60797.2024.10527344
6. Gasparetto, A., Marcuzzo, M., Zangari, A., & Albarelli, A. (2022). A Survey on Text Classification Algorithms: From Text to Predictions. *Information*, *13*(2), 83. https://doi.org/10.3390/info13020083
7. Salloum, S., Gaber, T., Vadera, S., & Shaalan, K. (2022). A Systematic Literature Review on Phishing Email Detection Using Natural Language Processing Techniques. *IEEE Access*, *10*, 65703–65727. https://doi.org/10.1109/ACCESS.2022.3183083