Phishing Email Detection: SVMs, and XGBoost

Liem, Josef & Kriter, Andrew

May 2025

1 Introduction

1.1 Background & Research Question

Phishing emails pose a substantial threat even today, seeking to exploit individuals into revealing sensitive information, carry out scams, and identify technical vulnerabilities. As Morrow et al. explained, in one study conducted within a university, over one fourth of students opened fake phishing emails and/or opened listed urls [1]. Phishing employees techniques in social engineering, looking to deceive targeted individuals into providing financial information or passwords, often for seemingly legitimate reasons. Techniques of this deception range anywhere from faking the need for a signature, security and password updates, job offerings, to outright extortion [1].

In this report, we aimed to understand how different statistical learning techniques could be applied to identify phishing emails. Namely, we asked ourselves the following: 1. Are support vector machine (SVM)-based approaches appropriate and effective in classification tasks over phishing email text identification? 2. How does a combined SVM-PCA approach compare against other statistical learning methods such as XGBoost on text for reference?

1.2 Dataset Overview

In this project, we evaluated the performance of various supervised learning methods, mainly support vector machines (SVMs), over a dataset of phishing and non-phishing emails released as a part of The Conference on Email and Anti-Spam (CEAS) challenge from 2008 [2]. This dataset consisted of roughly 39,154 observations, for which the sender, receiver email, date, subject, text, binary variable indicating the presence of urls, and the label was provided (Table 1).

Field	Value
Sender	John Smith <SmithJ@phishing.com>
Receiver	user4@gvc.ceas-challenge.cc
Date	Tue, 05 Aug 2008 19:20:27 -0500
Subject	Items for the man
Body	No time to look for watch? Replica Classics Replica Rolex Swiss Watches Classical Bvlgari watches at Replica Classics www.some-evil-url-that-does-nefarious-things.com
Label (Phishing)	1
URLs	1

Table 1: Example Phishing Email (Label = 1). Links and identifying information removed for safety.

The dataset appeared slightly balanced, in favor of phishing emails overall (Figure 1). We justified that accuracy is an adequate measure of model performance on balanced dataset, with confusion matrices for good measure. Given these metrics, a good expected model performance, in order, would be a high accuracy and no or few false positives (FPs) or false negatives(FNs), then a high accuracy with few or no FNs and a moderate to low number of FPs. In this scenario, we were more concerned with detecting and flagging potential phishing emails even if they were safe, rather than letting some slip past. We noted that phishing emails generally tended to have external links more often that non-phishing emails (Figure 3).

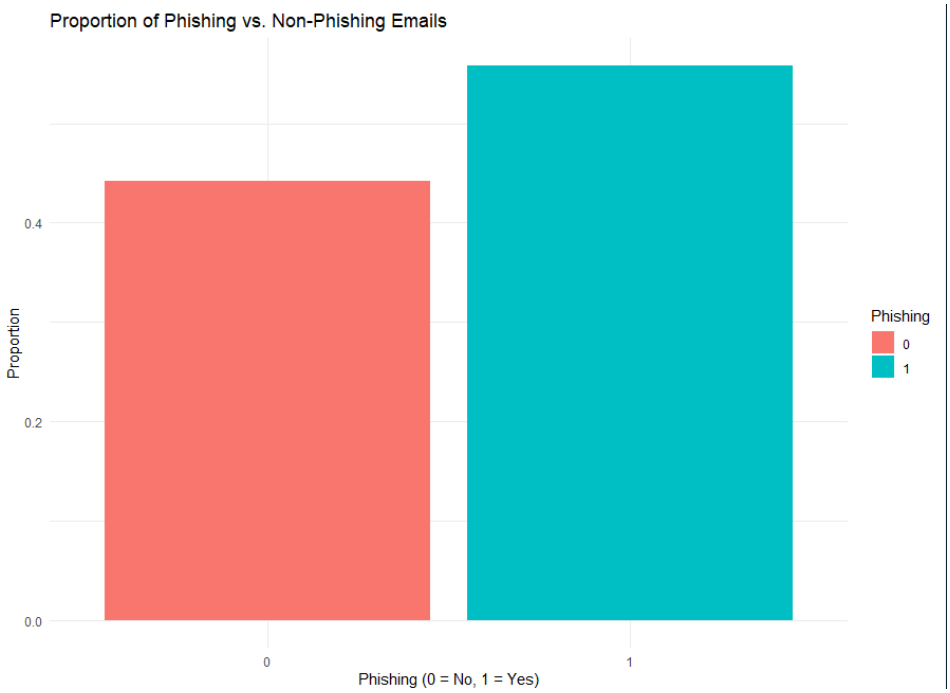

Figure 1: Proportion of phishing and non-phishing emails. Code written to generate the figure shown below.

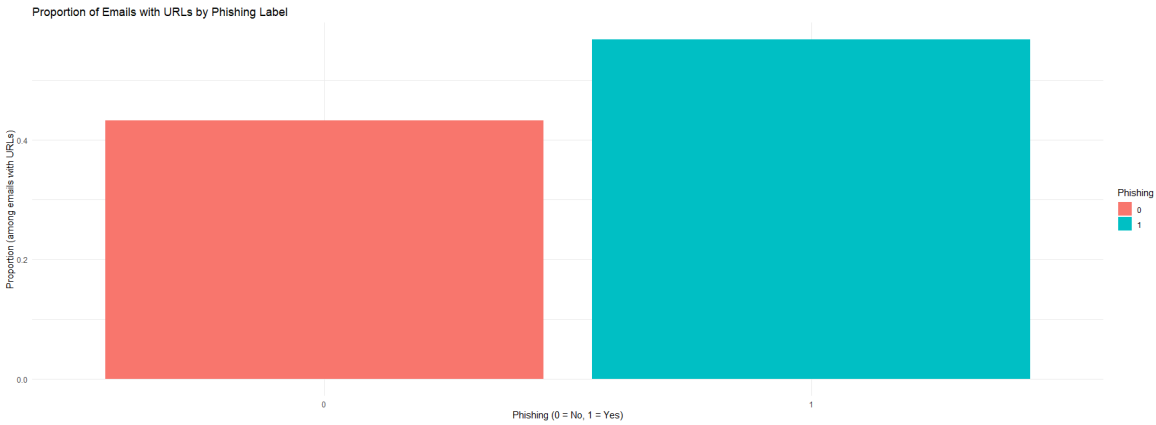

Figure 2: Proportion of phishing and non-phishing emails containing urls. Code to generate the figure was omitted as it was almost identical for Figure 1. See CEAS_08_EXPLORATORY.R for implementation. Nearly 43% of non-phishing emails contained links compared against 57% for phishing.

```
raw_phish = read_csv("CEAS_08.csv")
phish = raw_phish |>
  rename(text = `body`, phishing = `label`) |>
  mutate(
    text = str_to_lower(text),
    # https://stackoverflow.com/questions/41984513/r-str-replace-all-except-periods-and-dashes/41984645
    text = str_replace_all(text, "[^a-z\\s]", " "),
    text = str_replace_all(text, "\\s+", " ")
  ) |>
  filter(!is.na(text), text != "empty")

phish |>
  count(phishing) |>
  mutate(proportion = n / sum(n)) |>
  ggplot(aes(x = factor(phishing), y = proportion, fill = factor(phishing))) +
  geom_col() +
  labs(
    title = "Proportion of Phishing vs. Non-Phishing Emails",
    x = "Phishing (0 = No, 1 = Yes)",
    y = "Proportion",
    fill = "Phishing"
  ) +
  theme_minimal()
```

Code Example 1: Implementation of Dataset Balance Plot.

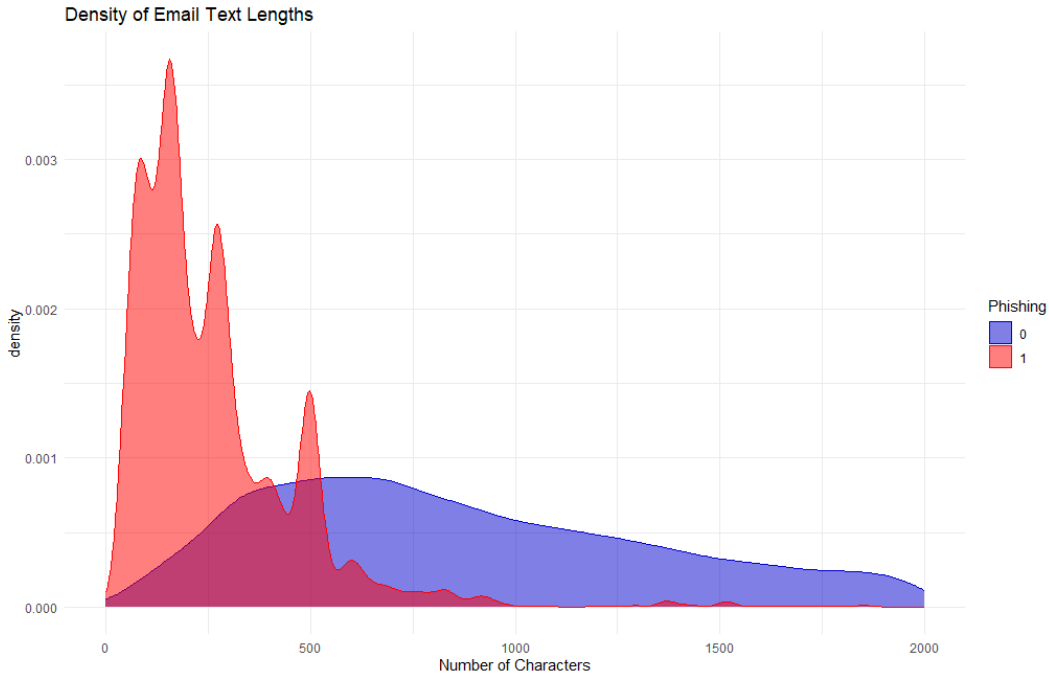

Figure 3: Distribution of number of characters in given emails by phishing-email status.

Here, we noted that emails determined to be phishing tended to have a fewer overall number of characters. Interestingly, we suspected that the many sharp peaks within the phishing email distribution corresponded to identical or highly similar phishing email contents sent to different recipients in our dataset, as we observed upon manual data inspection.

```
phish |>
  mutate(text_length = nchar(text)) |>
  ggplot(aes(x = text_length, color = factor(phishing), fill = factor(
      phishing))) +
  geom_density(alpha = 0.5) +
  scale_color_manual(values = c("blue3", "red1")) +
  scale_fill_manual(values = c("blue3", "red1")) +
  labs(
    title = "Density of Email Text Lengths",
    x = "Number of Characters",
    fill = "Phishing",
    color = "Phishing"
  ) +
  theme_minimal() +
  xlim(0, 2000)
```

Code Example 2: Implementation of Phishing Dataset Email Length Plot.

2 Principle Component Analysis

We first considered the use of PCA, as phishing email text data was highly dimensional when tokenized and converted into a document-term matrix (DTM). PCA is an eigendecomposition technique which seeks to reduce the dimensionality/number of features within our data, while trying to maintain as much of the variance or information as possible. In this case, PCA would capture relationships between words that frequently co-occur within the same given email body, revealing certain "themes" of emails which might also be predictive of whether they were phishing or not. Our DTM, was a sparse matrix representation of frequency of appearance (in a given email) of the thousand most frequently occurring words for phishing and non-phishing emails overall. Prior to producing this DTM, however, we applied word stemming, such that words containing same base root generally shared the same semantic meaning. This process ensured that we reduced the overall size of the vocabulary considered, ideally improving classification performance.

```r
tokens = phish |>
  mutate(ID = row_number()) |>
  unnest_tokens(word, text) |>
  anti_join(stop_words, by = "word") |>
  mutate(stem = wordStem(word))

vocab = tokens |>
  count(phishing, stem, sort = TRUE) |>
  group_by(phishing) |>
  slice_max(order_by = n, n = 1000) |>
  ungroup() |>
  distinct(stem) |>
  pull(stem)

# Filter tokens and create DTM
dtm_labeled = tokens |>
  filter(stem %in% vocab) |>
  count(ID, stem) |>
  pivot_wider(names_from = stem, values_from = n, values_fill = 0) |>
  left_join(phish |> mutate(ID = row_number()) |> select(ID, phishing), by =
      "ID")

# Prepare matrix for PCA
dtm_matrix = dtm_labeled |>
  select(-ID, -phishing) |>
  as.matrix()
```

Code Example 3: Implementation of Phishing Dataset tokenization, word stemming, and DTM preparation.

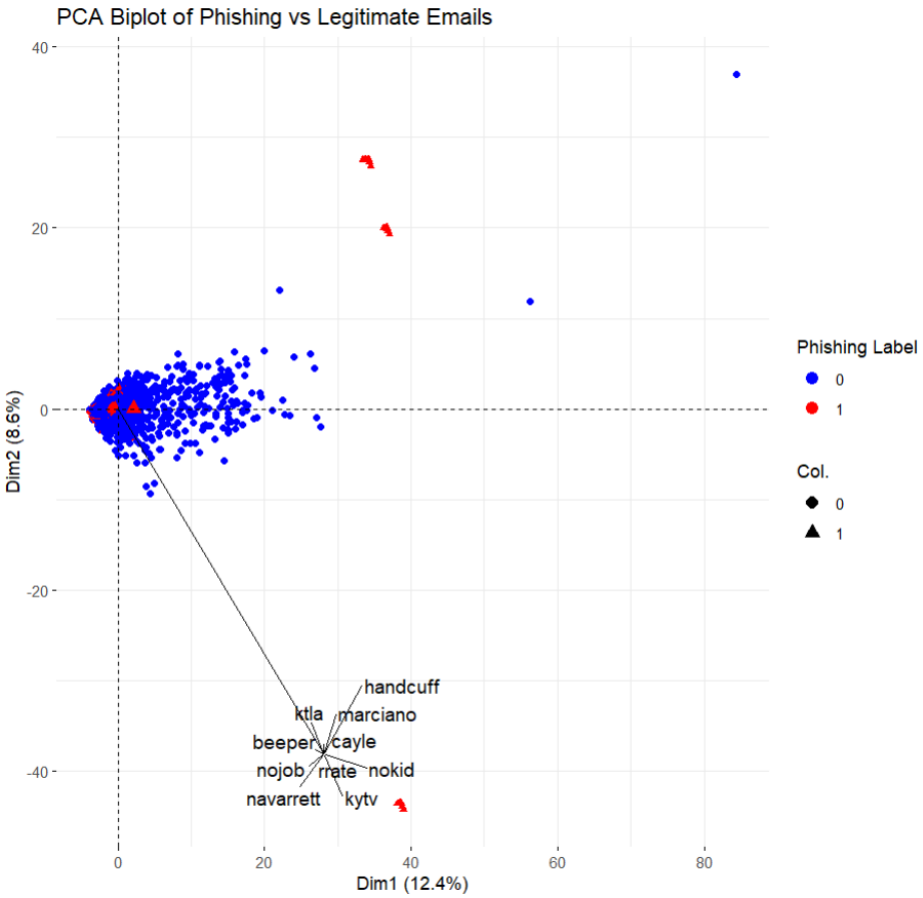

Figure 4: Biplot of 1st and 2nd principle components for PCA on phishing data.

As is shown in the biplot, roughly 12.4% and 8.6% of the total variance in our data could be explained by the first two principle components or "axes" alone(Figure 4). We noticed that among phishing emails, very distinct, tight clusters emerged, likely reflecting very similarly worded phishing emails sent from the same individual or group of senders, or phishing emails containing urls with these terms. We omit including these urls for reference for safety reasons.

```r
pca_phish = prcomp(dtm_matrix, scale. = TRUE, center = TRUE)

# PCA biplot: show top contributing stem features
fviz_pca_biplot(pca_phish,
                repel = TRUE,
                select.var = list(contrib = 20),
                col.ind = factor(dtm_labeled$phishing),
                col.var = "black",
                label = "var",
                axes = c(2, 3)) +
  scale_color_manual(values = c("0" = "blue", "1" = "red")) +
  labs(title = "PCA Biplot of Phishing vs Legitimate Emails",
       color = "Phishing Label")
```

Code Example 4: Implementation of Phishing PCA and biplot visualization.

3 Machine Learning Methods

3.1 Support Vector Machines

In order to classify emails as phishing or not phishing, we employ SVMs, which fall under the category of supervised or labeled-learning techniques. Before classification, we apply Principal Component Analysis (PCA) to the email data in order to reduce dimensionality. After dimensionality reduction, the SVM identifies an optimal hyperplane that separates phishing from non-phishing emails in this transformed space. Less technically, what this means is that a given hyperplane serves as a decision boundary separating our phishing and non-phishing emails; in two dimensional space, this can be interpreted as the line separating two classes of points.

To achieve this, SVM attempts to maximize a margin from this hyperplane to the nearest point, also known as the support vectors. Highly separable margins generally indicate better performance for our purposes. This hyperplane is defined as $\vec{w}^T \cdot \vec{x} + b = 1$ or -1 depending on the corresponding class, where $\vec{x}$ denotes the vector of feature values, $\vec{w}$ denotes a weighting, and b the bias term. To find a solution, it then optimizes by trying to minimize the magnitude (specifically the squared Euclidian norm $\frac{1}{2}||w||^2$) of the weight vector $\vec{w}$ under the constraint of $y_i(\vec{w}\vec{x} + b)$ (i.e. all points are correctly classified with a certain distance from the hyperplane itself). Overall, the important takeaway is that SVM learns the specific pattern, defined in terms of weight vectors, feature vectors, and a bias, such that the phishing PCA data best separates phishing emails from legitimate ones with a margin of separation.

The kernel, in SVMS, attempts to map non-linearly separable data to a higher dimensional space in which they can be separated. For SVMs, we test a variety of kernels or "mappings" to these higher dimensional spaces, including the linear and polynomial kernel. For the linear kernel, this is defined as $K(x_i, x_j) = x_i^T \cdot x_j$ where x_i, x_j are feature vectors corresponding to phishing emails. Overall, in slightly less technical terms, this particular kernel finds a linear decision boundary separating emails into phishing or non-phishing types.

The polynomial kernel is a type of kernel that allows for non-linear decision boundaries, useful for when the data exhibits highly complex relationships in terms how PCA text features of phishing or

non-phishing emails relate to the their class. Technically speaking, the linear kernel is just a special case of the polynomial kernel $K(x_i, x_j) = (x_i^T \cdot x_j)^d$, where degree of the polynomial $d = 1$.

We began by splitting the data 80/20 for training and testing sets. Only considering the first twenty components over our SVM with linear and polynomial kernels of different degrees, we observed that all models performed extremely well at classifying phishing emails. However, the polynomial kernel with degree three performed slightly better, with an accuracy of 97%. We suspect that the relationship of our data and the corresponding type of the email better corresponded to a non-linear decision boundary for SVM, even if the linear kernel performed extremely well (Table 2, 3, 4). Importantly, the degree three polynomial SVM only missed about 1.6% of phishing emails, indicating extremely good model recall.

Table 2: Confusion Matrix and Accuracy for Linear Kernel SVM

Actual \ Predicted	**0**	**1**
0	3252	95
1	211	4207

Accuracy: 0.9606%

Table 3: Confusion Matrix and Accuracy for Polynomial Kernel Degree 2 SVM

Actual \ Predicted	**0**	**1**
0	3276	58
1	187	4244

Accuracy: 0.9684%

Table 4: Confusion Matrix and Accuracy for Polynomial Kernel Degree 3 SVM

Actual \ Predicted	**0**	**1**
0	3308	69
1	155	4233

Accuracy: 0.9712%

```r
set.seed(42)
train_idx = sample(nrow(pca_scores), size = 0.8 * nrow(pca_scores))
train_data = pca_scores[train_idx, ]
test_data  = pca_scores[-train_idx, ]

pca_scores = as.data.frame(pca_phish$x[, 1:20])
pca_scores$phishing = factor(dtm_labeled$phishing)

svm_model = svm(phishing ~ ., data = train_data,
                 kernel = "linear", scale = FALSE)

#EVALUATE
preds = predict(svm_model, newdata = test_data)

# Confusion matrix
table(Predicted = preds, Actual = test_data$phishing)

# Accuracy
mean(preds == test_data$phishing)
```

Code Example 5: Implementation of linear kernel SVM and corresponding performance metrics. Other kernel "polynomial" with degree = 3, "polynomial" with degree = 2 were also tested.

3.2 XGBoost

We then considered applying an XGBoost model over our phishing email text data. XGBoost is a gradient boosted decision tree algorithm, which build sequential models where each tree attempts to correct errors in the previous tree. Over the first twenty components, we observed the highest accuracy relative to any other method attempted (Table 5). Though, the minor discrepancy in accuracy could probably be explained by resampling of the training and testing datasets alone. The model showed a slightly higher number of false positive than false negatives, with 125 false positives and 58 false negatives.

Actual Label	**Predicted 0**	**Predicted 1**
0	3,351	58
1	125	4,285

Table 5: Confusion matrix for XGBoost classification using first 20 PCA components. Accuracy = 97.67%.

```r
library(xgboost)
library(shapviz)

#xgboost model

x_data <- as.data.frame(pca_phish$x[, 1:20])      # all PCA components
y_label <- as.numeric(dtm_labeled$phishing)  # labels: 0 or 1

# Train/test split (same as earlier)
x_train <- x_data[train_idx, ]
y_train <- y_label[train_idx]
x_test  <- x_data[-train_idx, ]
y_test  <- y_label[-train_idx]

# Create DMatrix
dtrain <- xgb.DMatrix(data = as.matrix(x_train), label = y_train)
dtest  <- xgb.DMatrix(data = as.matrix(x_test), label = y_test)

# Train XGBoost classification model
#list() inspired from eXtreme Gradient boosting Training manpage example
set.seed(1)
xgb_model <- xgb.train(
  data = dtrain,
  nrounds = 100,
  params = list(
    objective = "binary:logistic",
    eval_metric = "auc",
    eta = 0.1,
    max_depth = 6
  )
)

# Predict on test set
preds_prob <- predict(xgb_model, newdata = dtest)
preds_label <- as.numeric(preds_prob > 0.5)

# Confusion matrix and accuracy
table(Predicted = preds_label, Actual = y_test)
mean(preds_label == y_test)
```

Code Example 6: Implementation of XGBoost.

4 Conclusion

In this report, we set out to evaluate the effectiveness of various statistical learning methods, including Support Vector Machines (SVMs) with dimensionality reduction via Principal Component Analysis (PCA) and XGBoost in identifying phishing emails from a labeled dataset. Our findings affirm that both SVM and XGBoost models are highly capable of distinguishing phishing content, with accuracies consistently in the high 90% range.

Among the SVM models, the polynomial kernel of degree three demonstrated the best overall performance, achieving a 97.12% accuracy rate and particularly strong recall, missing only a small fraction of phishing emails. This suggests that the non-linear structure of the data is best captured by more flexible, higher-degree decision boundaries. One may think that if we just did a very high degree, it would always be the best one, however by splitting the data 80/20 for training and testing data, we were able to show that a degree of 3 generalizes well to new data through our testing data. XGBoost, achieved slightly higher accuracy than the SVM models, though its marginal improvement may be attributable to data partitioning effects rather than just being a better algorithm. Still, it remains a strong candidate for deployment in phishing detection systems.

Ultimately, our results show that SVM-PCA models and XGBoost can be effectively used in text-based phishing detection tasks. Given the persistent and evolving nature of phishing attacks, the application of such machine learning approaches offers a valuable defense mechanism in real-world cybersecurity systems.

References

[1] E. Morrow, "Scamming higher ed: An analysis of phishing content and trends," *Computers in Human Behavior*, vol. 158, p. 108274, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563224001420

[2] CEAS Challenge Organizers, "Ceas 2008 spam challenge," https://www.ceas.cc/2008/challenge.html, 2008, organized by IBM Research, University of Waterloo, and Jozef Stefan Institute. [Accessed 18-May-2025].